# TABLE OF CONTENTS

## Chapter 1. Release Notes

1.1. CUDA 7.0 ................................................................. 1

1.2. CUDA 6.5 ................................................................. 2

1.3. CUDA 6.0 ................................................................. 2

1.4. CUDA 5.5 ................................................................. 3

1.5. CUDA 5.0 ................................................................. 3

1.6. CUDA 4.2 ................................................................. 4

1.7. CUDA 4.1 ................................................................. 4

## Chapter 2. Getting Started

2.1. Getting CUDA Samples .................................................. 6

2.2. Building Samples ......................................................... 7

2.3. CUDA Cross-Platform Samples ...................................... 8

2.4. Using CUDA Samples to Create Your Own CUDA Projects ... 10

## Chapter 3. Samples Reference

3.1. Simple Reference ...................................................... 13

asyncAPI ................................................................. 13

cdpSimplePrint - Simple Print (CUDA Dynamic Parallelism) ........... 14

cdpSimpleQuicksort - Simple Quicksort (CUDA Dynamic Parallelism) ... 14

clock - Clock ............................................................. 15

clock_nvrtc - Clock libNVRTC ........................................ 15

cppIntegration - C++ Integration ....................................... 15

cppOverload ............................................................. 16

cudaOpenMP .............................................................. 16

inlinePTX - Using Inline PTX ........................................ 16

inlinePTX_nvrtc - Using Inline PTX with libNVRTC .................. 17

matrixMul - Matrix Multiplication (CUDA Runtime API Version) .... 17

matrixMul_nvrtc - Matrix Multiplication with libNVRTC ............. 17

---

www.nvidia.com
CUDA Samples

TRM-06704-001_v7.0 | ii
matrixMulCUBLAS - Matrix Multiplication (CUBLAS) .................................................. 18
matrixMulDrv - Matrix Multiplication (CUDA Driver API Version) .............................. 19
simpleAssert ........................................................................................................ 19
simpleAssert_nvrtc - simpleAssert with libNVRTC............................................................. 19
simpleAtomicIntrinsics - Simple Atomic Intrinsics ...................................................... 20
simpleAtomicIntrinsics_nvrtc - Simple Atomic Intrinsics with libNVRTC...........................20
simpleCallback - Simple CUDA Callbacks ..........................................................................20
simpleCubemapTexture - Simple Cubemap Texture......................................................... 21
simpleIPC ........................................................................................................................... 21
simpleLayeredTexture - Simple Layered Texture.............................................................. 22
simpleMPI ............................................................................................................. 22
simpleMultiCopy - Simple Multi Copy and Compute ............................................................ 22
simpleMultiGPU - Simple Multi-GPU ............................................................................... 23
simpleOccupancy .............................................................................................................. 23
simpleP2P - Simple Peer-to-Peer Transfers with Multi-GPU ............................................ 23
simplePitchLinearTexture - Pitch Linear Texture............................................................. 24
simplePrintf ........................................................................................................................ 24
simpleSeparateCompilation - Simple Static GPU Device Library .................................... 24
simpleStreams ..................................................................................................................... 25
simpleSurfaceWrite - Simple Surface Write ...................................................................... 25
simpleTemplates - Simple Templates ............................................................................... 25
simpleTemplates_nvrtc - Simple Templates with libNVRTC ............................................. 26
simpleTexture - Simple Texture ....................................................................................... 26
simpleTextureDrv - Simple Texture (Driver Version).......................................................... 26
simpleVoteIntrinsics - Simple Vote Intrinsics ................................................................ 27
simpleVoteIntrinsics_nvrtc - Simple Vote Intrinsics with libNVRTC ..................................27
simpleZeroCopy ....................................................................................................... 28
template - Template ............................................................................................................ 28
template_runtime - Template using CUDA Runtime ......................................................... 28
UnifiedMemoryStreams - Unified Memory Streams.......................................................... 29
vectorAdd - Vector Addition .............................................................................................. 29
vectorAdd_nvrtc - Vector Addition with libNVRTC............................................................ 29
vectorAddDrv - Vector Addition Driver API ........................................................................ 30
3.2. Utilities Reference ......................................................................................................... 30
bandwidthTest - Bandwidth Test ...................................................................................... 30
deviceQuery - Device Query ............................................................................................... 31
deviceQueryDrv - Device Query Driver API ...................................................................... 31
p2pBandwidthLatencyTest - Peer-to-Peer Bandwidth Latency Test with Multi-GPUs ............31
3.3. Graphics Reference ...................................................................................................... 32
bindlessTexture - Bindless Texture .................................................................................... 32
Mandelbrot ............................................................................................................................ 32
marchingCubes - Marching Cubes Isosurfaces .................................................................. 33
simpleD3D10 - Simple Direct3D10 (Vertex Array) .............................................................. 33
simpleD3D10RenderTarget - Simple Direct3D10 Render Target .......................................................... 34
simpleD3D10Texture - Simple D3D10 Texture .................................................................................. 34
simpleD3D11Texture - Simple D3D11 Texture .................................................................................. 34
simpleD3D9 - Simple Direct3D9 (Vertex Arrays) ............................................................................. 35
simpleD3D9Texture - Simple D3D9 Texture .................................................................................. 35
simpleGL - Simple OpenGL .................................................................................................................. 36
simpleGLES - Simple OpenGL ............................................................................................................ 36
simpleGLES_screen - Simple OpenGL on Screen .............................................................................. 37
simpleTexture3D - Simple Texture 3D .................................................................................................. 37
SLID3D10Texture - SLI D3D10 Texture ............................................................................................... 38
volumeFiltering - Volumetric Filtering with 3D Textures and Surface Writes ...................................... 38
volumeRender - Volume Rendering with 3D Textures ........................................................................ 38

3.4. Imaging Reference ..................................................................................................................... 39
bicubicTexture - Bicubic B-spline Interpolation .................................................................................. 39
bilateralFilter - Bilateral Filter ............................................................................................................ 40
boxFilter - Box Filter .......................................................................................................................... 40
convolutionFFT2D - FFT-Based 2D Convolution ............................................................................. 41
convolutionSeparable - CUDA Separable Convolution .................................................................... 41
convolutionTexture - Texture-based Separable Convolution ............................................................. 41
cudaDecodeD3D9 - CUDA Video Decoder D3D9 API ....................................................................... 42
cudaDecodeGL - CUDA Video Decoder GL API ................................................................................. 42
dct8x8 - DCT8x8 ............................................................................................................................ 43
dwtHaar1D - 1D Discrete Haar Wavelet Decomposition .................................................................... 43
dxtc - DirectX Texture Compressor (DXTC) ..................................................................................... 44
histogram - CUDA Histogram ............................................................................................................. 44
HSOpticalFlow - Optical Flow ............................................................................................................. 44
imageDenoising - Image denoising ...................................................................................................... 45
postProcessGL - Post-Process in OpenGL ....................................................................................... 45
recursiveGaussian - Recursive Gaussian Filter .................................................................................. 46
simpleCUDA2GL - CUDA and OpenGL Interop of Images ................................................................. 46
SobelFilter - Sobel Filter .................................................................................................................... 47
stereoDisparity - Stereo Disparity Computation (SAD SIMD Intrinsics) ............................................. 47

3.5. Finance Reference ...................................................................................................................... 47
binomialOptions - Binomial Option Pricing ...................................................................................... 47
binomialOptions_nvrtc - Binomial Option Pricing with libNVRTC .................................................. 48
BlackScholes - Black-Scholes Option Pricing ..................................................................................... 48
BlackScholes_nvrtc - Black-Scholes Option Pricing with libNVRTC ................................................ 49
MonteCarloMultiGPU - Monte Carlo Option Pricing with Multi-GPU support ................................ 49
quasirandomGenerator - Niederreiter Quasirandom Sequence Generator ..................................... 50
quasirandomGenerator_nvrtc - Niederreiter Quasirandom Sequence Generator with libNVRTC ...... 50
SobolQRNG - Sobol Quasirandom Number Generator ..................................................................... 50

3.6. Simulations Reference ................................................................................................................. 51
3.7. Advanced Reference .................................................................................... 55
alignedTypes - Aligned Types.............................................................................55
cdpAdvancedQuicksort - Advanced Quicksort (CUDA Dynamic Parallelism)..............55
cdpBezierTessellation - Bezier Line Tessellation (CUDA Dynamic Parallelism)..............56
cdpLUDecomposition - LU Decomposition (CUDA Dynamic Parallelism)......................56
cdpQuadtree - Quad Tree (CUDA Dynamic Parallelism) ..............................................57
concurrentKernels - Concurrent Kernels....................................................................57
eigenvalues - Eigenvalues....................................................................................57
fastWalshTransform - Fast Walsh Transform...........................................................58
FDTD3d - CUDA C 3D FDTD..................................................................................58
FunctionPointers - Function Pointers........................................................................58
interval - Interval Computing..................................................................................59
lineOfSight - Line of Sight......................................................................................59
matrixMulDynlinkJIT - Matrix Multiplication (CUDA Driver API version with Dynamic Linking Version)......................................................................................59
mergeSort - Merge Sort.........................................................................................60
newdelete - NewDelete..........................................................................................60
pxjit - PTX Just-in-Time compilation.......................................................................60
radixSortThrust - CUDA Radix Sort (Thrust Library)...............................................60
reduction - CUDA Parallel Reduction.............................................................. 61
scalarProd - Scalar Product....................................................................................61
scan - CUDA Parallel Prefix Sum (Scan)..............................................................62
segmentationTreeThrust - CUDA Segmentation Tree Thrust Library..........................62
shfl_scan - CUDA Parallel Prefix Sum with Shuffle Intrinsics (SHFL_Scan)..................62
simpleHyperQ.........................................................................................................62
sortingNetworks - CUDA Sorting Networks..........................................................63
StreamPriorities - Stream Priorities........................................................................63
threadFenceReduction..............................................................................................63
threadMigration - CUDA Context Thread Management..........................................64
transpose - Matrix Transpose................................................................................64
3.8. Cudalibraries Reference ..............................................................................65
batchCUBLAS......................................................................................................65
boxFilterNPP - Box Filter with NPP.........................................................................65
conjugateGradient - ConjugateGradient..................................................................65
conjugateGradientPrecond - Preconditioned Conjugate Gradient............................66
Chapter 4. Dependencies

Third-Party Dependencies

FreelImage

Message Passing Interface

Only 64-Bit

DirectX

OpenGL

OpenGL ES

OpenMP

Screen

X11

CUDA Features

CUFFT Callback Routines

CUDA Dynamic Parallelism

CUBLAS

CUDA Interprocess Communication

CUFFT

CURAND

CUSPARSE

NPP

NVRTC

NVCUVID
# LIST OF TABLES

Table 1  Supported Target Arch/OS Combinations ...........................................................8

Table 2  Basic Key Concepts and Associated Samples .....................................................79

Table 3  Advanced Key Concepts and Associated Samples ...............................................84

Table 4  CUDA Driver API and Associated Samples .........................................................89

Table 5  CUDA Runtime API and Associated Samples ......................................................93
This section describes the release notes for the CUDA Samples only. For the release notes for the whole CUDA Toolkit, please see CUDA Toolkit Release Notes.

1.1. CUDA 7.0

- Removed support for Windows 32-bit builds.
- The Makefile x86_64=1 and ARMv7=1 options have been deprecated. Please use TARGET_ARCH to set the targeted build architecture instead.
- The Makefile GCC option has been deprecated. Please use HOST_COMPILER to set the host compiler instead.
- The CUDA Samples are no longer shipped as prebuilt binaries on Windows. Please use VS Solution files provided to build respective executable.
- Added 0_Simple/clock_nvrtc. Demonstrates how to compile clock function kernel at runtime using libNVRT to measure the performance of kernel accurately.
- Added 0_Simple/inlinePTX_nvrtc. Demonstrates compilation of CUDA kernel having PTX embedded at runtime using libNVRT.
- Added 0_Simple/matrixMul_nvrtc. Demonstrates compilation of matrix multiplication CUDA kernel at runtime using libNVRT.
- Added 0_Simple/simpleAssert_nvrtc. Demonstrates compilation of CUDA kernel having assert() at runtime using libNVRT.
- Added 0_Simple/simpleAtomicIntrinsics_nvrtc. Demonstrates compilation of CUDA kernel performing atomic operations at runtime using libNVRT.
- Added 0_Simple/simpleTemplates_nvrtc. Demonstrates compilation of templatized dynamically allocated shared memory arrays CUDA kernel at runtime using libNVRT.
- Added 0_Simple/simpleVoteIntrinsics_nvrtc. Demonstrates compilation of CUDA kernel which uses vote intrinsics at runtime using libNVRT.
- Added 0_Simple/vectorAdd_nvrtc. Demonstrates compilation of CUDA kernel performing vector addition at runtime using libNVRT.
- Added 4_Finance/binomialOptions_nvrtc. Demonstrates runtime compilation using libNVRT of CUDA kernel which evaluates fair call price for a given set of European options under binomial model.
- Added **4_Finance/BlackScholes_nvrtc**. Demonstrates runtime compilation using libNVRTC of CUDA kernel which evaluates fair call and put prices for a given set of European options by Black-Scholes formula.
- Added **4_Finance/quasirandomGenerator_nvrtc**. Demonstrates runtime compilation using libNVRTC of CUDA kernel which implements Niederreiter Quasirandom Sequence Generator and Inverse Cumulative Normal Distribution functions for the generation of Standard Normal Distributions.

### 1.2. CUDA 6.5

- Added **7_CUDALibraries/cuHook**. Demonstrates how to build and use an intercept library with CUDA.
- Added **7_CUDALibraries/simpleCUFFT_callback**. Demonstrates how to compute a 1D-convolution of a signal with a filter using a user-supplied CUFFT callback routine, rather than a separate kernel call.
- Added **7_CUDALibraries/simpleCUFFT_MGPU**. Demonstrates how to compute a 1D-convolution of a signal with a filter by transforming both into frequency domain, multiplying them together, and transforming the signal back to time domain on Multiple GPUs.
- Added **7_CUDALibraries/simpleCUFFT_2d_MGPU**. Demonstrates how to compute a 2D-convolution of a signal with a filter by transforming both into frequency domain, multiplying them together, and transforming the signal back to time domain on Multiple GPUs.
- Removed **3_Imaging/cudaEncode**. Support for the CUDA Video Encoder (NVCUVENC) has been removed.
- Removed **4_Finance/ExcelCUDA2007**. The topic will be covered in a blog post at Parallel Forall.
- Removed **4_Finance/ExcelCUDA2010**. The topic will be covered in a blog post at Parallel Forall.
- The **4_Finance/binomialOptions** sample is now restricted to running on GPUs with SM architecture 2.0 or greater.
- The **4_Finance/quasirandomGenerator** sample is now restricted to running on GPUs with SM architecture 2.0 or greater.
- The **7_CUDALibraries/boxFilterNPP** sample now demonstrates how to use the static NPP libraries on Linux and Mac.
- The **7_CUDALibraries/conjugateGradient** sample now demonstrates how to use the static CUBLAS and CUSPARSE libraries on Linux and Mac.
- The **7_CUDALibraries/MersenneTwisterGP11213** sample now demonstrates how to use the static CURAND library on Linux and Mac.

### 1.3. CUDA 6.0

- New featured samples that support a new CUDA 6.0 feature called UVM-Lite
Added **0_Simple/UnifiedMemoryStreams** - new CUDA sample that demonstrates the use of OpenMP and CUDA streams with Unified Memory on a single GPU.

Added **1_Utility/p2pBandwidthTestLatency** - new CUDA sample that demonstrates how measure latency between pairs of GPUs with P2P enabled and P2P disabled.

Added **6_Advanced/StreamPriorities** - This sample demonstrates basic use of the new CUDA 6.0 feature stream priorities.

Added **7_CUDAPlatforms/ConjugateGradientUM** - This sample implements a conjugate gradient solver on GPU using cuBLAS and cuSPARSE library, using Unified Memory.

### 1.4. CUDA 5.5

- Linux makefiles have been updated to generate code for the AMRv7 architecture. Only the ARM hard-float floating point ABI is supported. Both native ARMv7 compilation and cross compilation from x86 is supported.
- 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 CUDA sample that demonstrates how to use C++ overloading with CUDA.
- Added **6_Advanced/cdpBezierTessellation** - new CUDA sample that demonstrates an advanced method of implementing Bezier Line Tessellation using CUDA Dynamic Parallelism. Requires compute capability 3.5 or higher.
- Added **7_CUDAPlatforms/jpegNPP** - new CUDA 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.

### 1.5. CUDA 5.0

- New directory structure for CUDA samples. Samples are classified accordingly to categories: **0_Simple, 1_Utilsitities, 2_Graphics, 3_Imaging, 4_Finance, 5_Simulations, 6_Advanced, and 7_CUDAPlatforms**.
- Added **0_Simple/simpleIPC** - 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.
- Added **0_Simple/simpleSeparateCompilation** - 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.

- **Added 2_Graphics/bindlessTexture** - demonstrates use of `cudaSurfaceObject`, `cudaTextureObject`, and MipMap support in CUDA. Requires Compute Capability 3.0 or higher.

- **Added 3_Imaging/stereoDisparity** - demonstrates how to compute a stereo disparity map using SIMD SAD (Sum of Absolute Difference) intrinsics. Requires Compute Capability 2.0 or higher.

- **Added 0_Simple/cdpSimpleQuicksort** - demonstrates a simple quicksort implemented using CUDA Dynamic Parallelism. This sample requires devices with compute capability 3.5 or higher.

- **Added 0_Simple/cdpSimplePrint** - demonstrates simple printf implemented using CUDA Dynamic Parallelism. This sample requires devices with compute capability 3.5 or higher.

- **Added 6_Advanced/cdpLUDEcomposition** - demonstrates LU Decomposition implemented using CUDA Dynamic Parallelism. This sample requires devices with compute capability 3.5 or higher.

- **Added 6_Advanced/cdpAdvancedQuicksort** - demonstrates an advanced quicksort implemented using CUDA Dynamic Parallelism. This sample requires devices with compute capability 3.5 or higher.

- **Added 6_Advanced/cdpQuadtree** - demonstrates Quad Trees implemented using CUDA Dynamic Parallelism. This sample requires devices with compute capability 3.5 or higher.

- **Added 7_CUDALibraries/simpleDevLibCUBLAS** - 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.

### 1.6. CUDA 4.2

- **Added segmentationTreeThrust** - demonstrates a method to build image segmentation trees using Thrust. This algorithm is based on Boruvka’s MST algorithm.

### 1.7. CUDA 4.1

- **Added MersenneTwisterGP11213** - implements Mersenne Twister GP11213, a pseudorandom number generator using the cuRAND library.

- **Added 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.

- **Added volumeFiltering** - demonstrates basic volume rendering and filtering using 3D textures.
- Added **simpleCubeMapTexture** - demonstrates how to use `texcubemap` fetch instruction in a CUDA C program.
- Added **simpleAssert** - demonstrates how to use GPU assert in a CUDA C program.
Chapter 2.  
GETTING STARTED

The CUDA Samples are an educational resource provided to teach CUDA programming concepts. The CUDA Samples are not meant to be used for performance measurements.
For system requirements and installation instructions, please refer to the Linux Getting Started Guide, the Windows Getting Started Guide, and the Mac Getting Started Guide.

2.1. Getting CUDA Samples

Windows
On Windows, the CUDA Samples are installed using the CUDA Toolkit Windows Installer. By default, the CUDA Samples are installed in:

\[ C:\ProgramData\NVIDIA Corporation\CUDA Samples\v7.0\ \]

The installation location can be changed at installation time.

Linux
On Linux, to install the CUDA Samples, the CUDA toolkit must first be installed. See the Getting Started Guide for more information on how to install the CUDA Toolkit.
Then the CUDA Samples can be installed by running the following command, where \(<target\_path>\) is the location where to install the samples:

\[ $\ cuda-install-samples-7.0.sh <target\_path> \]

Mac OSX
On Mac OSX, to install the CUDA Samples, the CUDA toolkit must first be installed. See the Getting Started Guide for more information on how to install the CUDA Toolkit.
Then the CUDA Samples can be installed by running the following command, where \(<target\_path>\) is the location where to install the samples:

\[ $\ cuda-install-samples-7.0.sh <target\_path> \]
2.2. Building Samples

Windows

The Windows samples are built using the Visual Studio IDE. Solution files (.sln) are provided for each supported version of Visual Studio, using the format:

*_vs<version>.sln* - for Visual Studio <version>

Complete samples solution files exist at:

C:\ProgramData\NVIDIA Corporation\CUDA Samples\v7.0\

Each individual sample has its own set of solution files at:

C:\ProgramData\NVIDIA Corporation\CUDA Samples\v7.0\<sample_dir>\

To build/examine all the samples at once, the complete solution files should be used. To build/examine a single sample, the individual sample solution files should be used.

Some samples require that the Microsoft DirectX SDK (June 2010 or newer) be installed and that the VC++ directory paths are properly set up (Tools > Options...). Check DirectX Dependencies section for details.

Linux

The Linux samples are built using makefiles. To use the makefiles, change the current directory to the sample directory you wish to build, and run `make`:

```
$ cd <sample_dir>
$ make
```

The samples makefiles can take advantage of certain options:

- **TARGET_ARCH=<arch>** - cross-compile targeting a specific architecture. Allowed architectures are x86_64, armv7l, aarch64, and ppc64le.

  By default, TARGET_ARCH is set to HOST_ARCH. On a x86_64 machine, not setting TARGET_ARCH is the equivalent of setting TARGET_ARCH=x86_64.

  ```
  $ make TARGET_ARCH=x86_64
  $ make TARGET_ARCH=armv7l
  $ make TARGET_ARCH=aarch64
  $ make TARGET_ARCH=ppc64le
  ```

  See here for more details.

- **dbg=1** - build with debug symbols

  ```
  $ make dbg=1
  ```

- **SMS="A B ..."** - override the SM architectures for which the sample will be built, where "A B ..." is a space-delimited list of SM architectures. For example, to generate SASS for SM 20 and SM 30, use **SMS="20 30"**.
$ make SMS="20 30"

- **HOST_COMPILER=<host_compiler>** - override the default g++ host compiler. See the Linux Getting Started Guide for a list of supported host compilers.

$ make HOST_COMPILER=g++

### Mac

The Mac samples are built using makefiles. To use the makefiles, change directory into the sample directory you wish to build, and run `make`:

```bash
$ cd <sample_dir>
$ make
```

The samples makefiles can take advantage of certain options:

- **dbg=1** - build with debug symbols
  ```bash
  $ make dbg=1
  ```

- **SMS="A B ..."** - override the SM architectures for which the sample will be built, where "A B ..." is a space-delimited list of SM architectures. For example, to generate SASS for SM 20 and SM 30, use `SMS="20 30"`.
  ```bash
  $ make SMS="A B ...
  ```

- **HOST_COMPILER=<host_compiler>** - override the default clang host compiler. See the Mac Getting Started Guide for a list of supported host compilers.
  ```bash
  $ make HOST_COMPILER=clang
  ```

### 2.3. CUDA Cross-Platform Samples

This section describes the options used to build cross-platform samples. **TARGET_ARCH=<arch>** and **TARGET_OS=<os>** should be chosen based on the supported targets shown below. **TARGET_FS=<path>** can be used to point nvcc to libraries and headers used by the sample.

<table>
<thead>
<tr>
<th>TARGET ARCH</th>
<th>TARGET OS</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>linux</td>
</tr>
<tr>
<td>x86_64</td>
<td>YES</td>
</tr>
<tr>
<td>armv7l</td>
<td>YES</td>
</tr>
<tr>
<td>aarch64</td>
<td>NO</td>
</tr>
<tr>
<td>ppc64le</td>
<td>YES</td>
</tr>
</tbody>
</table>
TARGET_ARCH

The target architecture must be specified when cross-compiling applications. If not specified, it defaults to the host architecture. Allowed architectures are:

- **x86_64** - 64-bit x86 CPU architecture
- **armv7l** - 32-bit ARM CPU architecture, like that found on Jetson TK1
- **aarch64** - 64-bit ARM CPU architecture, found on certain Android systems
- **ppc64le** - 64-bit little-endian IBM POWER8 architecture

TARGET_OS

The target OS must be specified when cross-compiling applications. If not specified, it defaults to the host OS. Allowed OSes are:

- **linux** - for any Linux distributions
- **darwin** - for Mac OS X
- **android** - for any supported device running Android
- **qnx** - for any supported device running QNX

TARGET_FS

The most reliable method to cross-compile the CUDA Samples is to use the TARGET_FS variable. To do so, mount the target’s filesystem on the host, say at `/mnt/target`. This is typically done using `exportfs`. In cases where `exportfs` is unavailable, it is sufficient to copy the target’s filesystem to `/mnt/target`. To cross-compile a sample, execute:

```
$ make TARGET_ARCH=<arch> TARGET_OS=<os> TARGET_FS=/mnt/target
```

Copying Libraries

If the TARGET_FS option is not available, the libraries used should be copied from the target system to the host system, say at `/opt/target/libs`. If the sample uses GL, the GL headers must also be copied, say at `/opt/target/include`. The linker must then be told where the libraries are with the `-rpath-link` and/or `-L` options. To ignore unresolved symbols from some libraries, use the `--unresolved-symbols` option as shown below. `SAMPLE_ENABLED` should be used to force the sample to build. For example, to cross-compile a sample which uses such libraries, execute:

```
$ make TARGET_ARCH=<arch> TARGET_OS=<os> \
   EXTRA_LDFLAGS="-rpath-link=/opt/target/libs -L/opt/target/libs \
   --unresolved-symbols=ignore-in-shared-libs" \
   EXTRA_CFLAGS="-I /opt/target/include" \
   SAMPLE_ENABLED=1
```
2.4. Using CUDA Samples to Create Your Own CUDA Projects

2.4.1. 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_Utility, 2_Graphics, 3_Imaging, 4_Finance, 5_Simulations, 6_Advanced, 7_CUDALibraries.)

1. Copy the content of:
   C:\ProgramData\NVIDIA Corporation\CUDA Samples\v7.0\<category>\template
   or
   C:\ProgramData\NVIDIA Corporation\CUDA Samples\v7.0\<category>\template_runtime
   to a directory of your own:
   C:\ProgramData\NVIDIA Corporation\CUDA Samples\v7.0\<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_vs<version>.sln
5. Run myproject.exe from the release or debug directories located in:
   C:\ProgramData\NVIDIA Corporation\CUDA Samples\v7.0\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.

2.4.2. Creating CUDA Projects for Linux

The default installation folder <SAMPLES_INSTALL_PATH> is NVIDIA_CUDA_7.0_Samples and <category> is one of the following: 0_Simple, 1_Utility, 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":
   ```make dbg=1```

5. Run the program:
   ```../../bin/x86_64/linux/release/myproject```

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

2.4.3. Creating CUDA Projects for Mac OS X

The default installation folder <SAMPLES_INSTALL_PATH> is: /Developer/NVIDIA/CUDA-7.0/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, 1Utilities, 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"
   make dbg=1

5. Run the program:

   ../../bin/x86_64/darwin/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

asyncAPI

This sample uses CUDA streams and events to overlap execution on CPU and GPU.
### Supported SM Architecture
<table>
<thead>
<tr>
<th>Supported SM Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3</td>
</tr>
</tbody>
</table>

### CUDA API
- cudaEventCreate, cudaEventRecord, cudaEventQuery, cudaEventDestroy,
- cudaEventElapsedTime, cudaMemcpyAsync

### Key Concepts
Asynchronous Data Transfers, CUDA Streams and Events

### Supported OSes
- Linux, Windows, OS X

---

**cdpSimplePrint - 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.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

**Dependencies**
- CDP

**Supported SM Architecture**
- SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**Key Concepts**
- CUDA Dynamic Parallelism

**Supported OSes**
- Linux, Windows, OS X

---

**cdpSimpleQuicksort - 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.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

**Dependencies**
- CDP

**Supported SM Architecture**
- SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**Key Concepts**
- CUDA Dynamic Parallelism

**Supported OSes**
- Linux, Windows, OS X
clock - Clock

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

**Supported SM Architecture**
SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**CUDA API**
cudaMalloc, cudaFree, cudaMemcpy

**Key Concepts**
Performance Strategies

**Supported OSes**
Linux, Windows, OS X

clock_nvrtc - Clock libNVRTC

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

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

**Dependencies**
NVRTC

**Supported SM Architecture**
SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**CUDA API**
cudaMalloc, cudaFree, cudaMemcpy

**Key Concepts**
Performance Strategies, Runtime Compilation

**Supported OSes**
Linux, Windows, OS X

cppIntegration - 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.

**Supported SM Architecture**
SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**CUDA API**
cudaMalloc, cudaFree, cudaMemcpy

**Supported OSes**
Linux, Windows, OS X
cppOverload
This sample demonstrates how to use C++ function overloading on the GPU.

<table>
<thead>
<tr>
<th>Supported SM Architecture</th>
<th>SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA API</td>
<td>cudaFuncSetCacheConfig, cudaFuncGetAttributes</td>
</tr>
<tr>
<td>Key Concepts</td>
<td>C++ Function Overloading, CUDA Streams and Events</td>
</tr>
<tr>
<td>Supported OSes</td>
<td>Linux, Windows, OS X</td>
</tr>
</tbody>
</table>

cudaOpenMP
This sample demonstrates how to use OpenMP API to write an application for multiple GPUs.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

<table>
<thead>
<tr>
<th>Dependencies</th>
<th>OpenMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported SM Architecture</td>
<td>SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3</td>
</tr>
<tr>
<td>CUDA API</td>
<td>cudaMalloc, cudaMemcpy</td>
</tr>
<tr>
<td>Key Concepts</td>
<td>CUDA Systems Integration, OpenMP, Multithreading</td>
</tr>
<tr>
<td>Supported OSes</td>
<td>Linux, Windows, OS X</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Supported SM Architecture</th>
<th>SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA API</td>
<td>cudaMalloc, cudaMemcpy, cudaMemcpy, cudaMemcpy, cudaMemcpy</td>
</tr>
<tr>
<td>Key Concepts</td>
<td>Performance Strategies, PTX Assembly, CUDA Driver API</td>
</tr>
<tr>
<td>Supported OSes</td>
<td>Linux, Windows, OS X</td>
</tr>
</tbody>
</table>
**inlinePTX_nvrtc - Using Inline PTX with libNVRTC**

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

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

**Dependencies**
- NVRTC

**Supported SM Architecture**
- SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**CUDA API**
- cudaMalloc, cudaMemcpy, cudaMemcpyHost, cudaFree, cudaMemcpyHost

**Key Concepts**
- Performance Strategies, PTX Assembly, CUDA Driver API, Runtime Compilation

**Supported OSes**
- Linux, Windows, OS X

---

**matrixMul - 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.

**Supported SM Architecture**
- SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**CUDA API**
- cudaEventCreate, cudaEventRecord, cudaEventQuery, cudaEventDestroy, cudaEventElapsedTime, cudaMemcpy, cudaMemcpyHost, cudaMemcpyHost, cudaMemcpyHost

**Key Concepts**
- CUDA Runtime API, Linear Algebra

**Supported OSes**
- Linux, Windows, OS X

---

**matrixMul_nvrtc - Matrix Multiplication with libNVRTC**

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.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

<table>
<thead>
<tr>
<th>Dependencies</th>
<th>NVRTC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported SM Architecture</td>
<td>SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3</td>
</tr>
<tr>
<td>CUDA API</td>
<td>cudaEventCreate, cudaEventRecord, cudaEventQuery, cudaEventDestroy, cudaEventElapsedTime, cudaEventSynchronize, cudaMalloc, cudaFree, cudaMemcpy</td>
</tr>
<tr>
<td>Key Concepts</td>
<td>CUDA Runtime API, Linear Algebra, Runtime Compilation</td>
</tr>
<tr>
<td>Supported OSes</td>
<td>Linux, Windows, OS X</td>
</tr>
</tbody>
</table>

matrixMulCUBLAS - 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.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

<table>
<thead>
<tr>
<th>Dependencies</th>
<th>CUBLAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported SM Architecture</td>
<td>SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3</td>
</tr>
<tr>
<td>CUDA API</td>
<td>cudaEventCreate, cudaEventRecord, cudaEventQuery, cudaEventDestroy, cudaEventElapsedTime, cudaMalloc, cudaMemcpy, cublasCreate, cublasSgemm</td>
</tr>
<tr>
<td>Key Concepts</td>
<td>CUDA Runtime API, Performance Strategies, Linear Algebra, CUBLAS</td>
</tr>
<tr>
<td>Supported OSes</td>
<td>Linux, Windows, OS X</td>
</tr>
</tbody>
</table>
matrixMulDrv - 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.


CUDA API: cuModuleLoad, cuModuleLoadDataEx, cuModuleGetFunction, cuMemAlloc, cuMemFree, cudaMemcpyHtoD, cudaMemcpyDtoH, cuLaunchKernel

Key Concepts: CUDA Driver API, Matrix Multiply

Supported OSes: Linux, Windows, OS X

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.


CUDA API: cudaMemcpy, cudaMemcpyHost, cudaFree, cudaMemcpyHost, cudaMemcpy

Key Concepts: Assert

Supported OSes: Linux, Windows

simpleAssert_nvrtc - simpleAssert with libNVRTC

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.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

Dependencies: NVRTC

<table>
<thead>
<tr>
<th>Sample</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>simpleAtomicIntrinsics</strong> - Simple Atomic Intrinsics</td>
<td>A simple demonstration of global memory atomic instructions. Requires Compute Capability 2.0 or higher.</td>
</tr>
<tr>
<td><strong>simpleAtomicIntrinsics_nvrtc</strong> - Simple Atomic Intrinsics with libNVRTC</td>
<td>A simple demonstration of global memory atomic instructions. This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed.</td>
</tr>
<tr>
<td><strong>simpleCallback</strong> - Simple CUDA Callbacks</td>
<td>This sample implements multi-threaded heterogeneous computing workloads with the new CPU callbacks for CUDA streams and events introduced with CUDA 5.0.</td>
</tr>
</tbody>
</table>

### simpleAtomicIntrinsics

**CUDA API**: cudaMalloc, cudaMallocHost, cudaFree, cudaFreeHost, cudaMemcpy  
**Key Concepts**: Assert, Runtime Compilation  
**Supported OSes**: Linux, Windows

### simpleAtomicIntrinsics_nvrtc

**CUDA API**: cudaMalloc, cudaFree, cudaMemcpy, cudaFreeHost  
**Key Concepts**: Atomic Intrinsics, Runtime Compilation  
**Supported OSes**: Linux, Windows, OS X

### simpleCallback

**Supported SM Architecture**: SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3
simpleCubemapTexture - Simple Cubemap Texture

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

**Supported SM Architecture**
- SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**CUDA API**
- cudaMalloc, cudaMemcpy3DArray, cudaMemcpy3D, cudaMemcpy
- cudaCreateChannelDesc, cudaMemcpy
- cudaBindTextureToArray, cudaMemcpy
- cudaMalloc, cudaFree, cudaFreeArray, cudaMemcpy

**Key Concepts**
- Texture, Volume Processing

**Supported OSes**
- Linux, Windows, OS X

---

cudaStreamCreate, cudaMemcpyAsync, cudaStreamAddCallback, cudaStreamDestroy

---

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.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

**Dependencies**
- IPC

**Supported SM Architecture**
- SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**CUDA API**
- cudaIpcGetEventHandle, cudaIpcOpenMemHandle, cudaIpcCloseMemHandle, cudaMemcpy
- cudaFreeHost, cudaMemcpy

**Key Concepts**
- CUDA Systems Integration, Peer to Peer, InterProcess Communication

**Supported OSes**
- Linux
**simpleLayeredTexture - Simple Layered Texture**

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

**Supported SM Architecture**
- SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**CUDA API**
- cudaMalloc, cudaMalloc3DArray, cudaMemcpy3D, cudaMemcpy
- cudaMemcpy

**Key Concepts**
- Texture, Volume Processing

**Supported OSes**
- Linux, Windows, OS X

**simpleMPI**

Simple example demonstrating how to use MPI in combination with CUDA.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

**Dependencies**
- MPI

**Supported SM Architecture**
- SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**CUDA API**
- cudaMemcpy

**Key Concepts**
- CUDA Systems Integration, MPI, Multithreading

**Supported OSes**
- Linux, Windows, OS X

**simpleMultiCopy - Simple Multi Copy and Compute**

Supported in GPUs with Compute Capability 1.1, overlapping compute with one memcpy 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.

**Supported SM Architecture**
- SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**CUDA API**
- cudaMemcpy
- cudaEventCreate, cudaEventRecord, cudaEventQuery, cudaEventDestroy, cudaMemcpyAsync
simpleMultiGPU - 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.

**Supported SM Architecture**

SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**CUDA API**

cudaEventCreate, cudaEventRecord, cudaEventQuery, cudaEventDestroy, cudaMemcpyAsync

**Key Concepts**

Asynchronous Data Transfers, CUDA Streams and Events, Multithreading, Multi-GPU

**Supported OSes**

Linux, Windows, OS X

---

simpleOccupancy

This sample demonstrates the basic usage of the CUDA occupancy calculator and occupancy-based launch configurator APIs by launching a kernel with the launch configurator, and measures the utilization difference against a manually configured launch.

**Supported SM Architecture**

SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**Key Concepts**

Occupancy Calculator

**Supported OSes**

Linux, Windows, OS X

---

simpleP2P - 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.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.
simplePitchLinearTexture - Pitch Linear Texture

Use of Pitch Linear Textures

Supported SM
SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

CUDA API
cudaMallocPitch, cudaMemcpy2D, cudaMemcpyToArray, cudaMemcpyToHost, cudaMemcpyToDevice, cudaMemcpyFromArray, cudaMemcpyFromHost, cudaMemcpyFromDevice, cudaMemcpyToArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyFromHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToHost, cudaMemcpyFromArray, cudaMemcpyToH...
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.

**Samples Reference**

<table>
<thead>
<tr>
<th>Supported SM Architecture</th>
<th>SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Key Concepts</td>
<td>Separate Compilation</td>
</tr>
<tr>
<td>Supported OSes</td>
<td>Linux, Windows, OS X</td>
</tr>
</tbody>
</table>

**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 2.0 or higher.

<table>
<thead>
<tr>
<th>Supported SM Architecture</th>
<th>SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA API</td>
<td>cudaEventCreate, cudaEventRecord, cudaEventQuery, cudaEventDestroy, cudaEventElapsedTime, cudaMemcpyAsync</td>
</tr>
<tr>
<td>Key Concepts</td>
<td>Asynchronous Data Transfers, CUDA Streams and Events</td>
</tr>
<tr>
<td>Supported OSes</td>
<td>Linux, Windows, OS X</td>
</tr>
</tbody>
</table>

**simpleSurfaceWrite - Simple Surface Write**

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

<table>
<thead>
<tr>
<th>Supported SM Architecture</th>
<th>SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA API</td>
<td>cudaMalloc, cudaMemcpy, cudaMemcpyAsync, cudaMallocArray, cudaBindTextureToArray, cudaCreateChannelDesc, cudaMemcpyAsync, cudaFreeArray, cudaMemcpy</td>
</tr>
<tr>
<td>Key Concepts</td>
<td>Texture, Surface Writes, Image Processing</td>
</tr>
<tr>
<td>Supported OSes</td>
<td>Linux, Windows, OS X</td>
</tr>
</tbody>
</table>

**simpleTemplates - Simple Templates**

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

| Supported SM Architecture | SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3 |
simpleTemplates_nvrtc - Simple Templates with libNVRTC

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

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

Dependencies
- NVRTC

Supported SM Architecture
- SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

Key Concepts
- C++ Templates, Runtime Compilation

Supported OSes
- Linux, Windows, OS X

simpleTexture - Simple Texture

Simple example that demonstrates use of Textures in CUDA.

Supported SM Architecture
- SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

CUDA API
- cudaMemcpy, cudaMemcpyToArray, cudaMemcpyToSymbol, cudaMemcpyToSymbol, cudaBindTexture, cudaBindTextureToArray, cudaMalloc, cudaFree, cudaFreeArray, cudaMemcpy

Key Concepts
- CUDA Runtime API, Texture, Image Processing

Supported OSes
- Linux, Windows, OS X

simpleTextureDrv - 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.

Supported SM Architecture
- SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

CUDA API

Supported OSes
- Linux, Windows, OS X
cuArrayDestroy, cuCtxDetach, cuMemcpy2D, cuModuleGetTexRef, cu TexRefSetArray, cuTexRefSetAddressMode, cuTexRefSetFilterMode, cuTexRefSetFlags, cuTexRefSetFormat, cuParamSetTexRef

Key Concepts
CUDA Driver API, Texture, Image Processing

Supported OSes
Linux, Windows, OS X

simpleVoteIntrinsics - Simple Vote Intrinsics
Simple program which demonstrates how to use the Vote (any, all) intrinsic instruction in a CUDA kernel. Requires Compute Capability 2.0 or higher.

Supported SM
SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

CUDA API
cudaMalloc, cudaFree, cuda Memcpy, cudaMemcpy, cudaMemcpyHost

Key Concepts
Vote Intrinsics

Supported OSes
Linux, Windows, OS X

simpleVoteIntrinsics_nvrtc - Simple Vote Intrinsics with libNVRTC
Simple program which demonstrates how to use the Vote (any, all) intrinsic instruction in a CUDA kernel with runtime compilation using NVRTC APIs. Requires Compute Capability 2.0 or higher.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

Dependencies
NVRTC

Supported SM
SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

CUDA API
cuMemAlloc, cuMemFree, cuMemcpyHtoD, cuMemFree

Key Concepts
Vote Intrinsics, CUDA Driver API, Runtime Compilation

Supported OSes
Linux, Windows, OS X
simpleZeroCopy

This sample illustrates how to use Zero MemCopy, kernels can read and write directly to pinned system memory.

<table>
<thead>
<tr>
<th>Supported SM Architecture</th>
<th>SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA API</td>
<td>cudaEventCreate, cudaEventRecord, cudaEventQuery, cudaEventDestroy, cudaEventElapsedTime, cudaHostAlloc, cudaHostGetDevicePointer, cudaHostRegister, cudaHostUnregister, cudaFreeHost</td>
</tr>
<tr>
<td>Key Concepts</td>
<td>Performance Strategies, Pinned System Paged Memory, Vector Addition</td>
</tr>
<tr>
<td>Supported OSes</td>
<td>Linux, Windows, OS X</td>
</tr>
<tr>
<td>Whitepaper</td>
<td>CUDA2.2PinnedMemoryAPIs.pdf</td>
</tr>
</tbody>
</table>

template - Template

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

<table>
<thead>
<tr>
<th>Supported SM Architecture</th>
<th>SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA API</td>
<td>cudaMalloc, cudaFree, cudaDeviceSynchronize, cudaMemcpy</td>
</tr>
<tr>
<td>Key Concepts</td>
<td>Device Memory Allocation</td>
</tr>
<tr>
<td>Supported OSes</td>
<td>Linux, Windows, OS X</td>
</tr>
</tbody>
</table>

template_runtime - Template using CUDA Runtime

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

<table>
<thead>
<tr>
<th>Supported SM Architecture</th>
<th>SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3</th>
</tr>
</thead>
<tbody>
<tr>
<td>CUDA API</td>
<td>cudaMalloc, cudaMallocHost, cudaFree, cudaFreeHost, cudaDeviceSynchronize, cudaMemcpy</td>
</tr>
<tr>
<td>Key Concepts</td>
<td>CUDA Data Transfers, Device Memory Allocation</td>
</tr>
<tr>
<td>Supported OSes</td>
<td>Linux, Windows, OS X</td>
</tr>
</tbody>
</table>
UnifiedMemoryStreams - Unified Memory Streams

This sample demonstrates the use of OpenMP and streams with Unified Memory on a single GPU.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

<table>
<thead>
<tr>
<th>Dependencies</th>
<th>UVM, CUBLAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported SM Arch.</td>
<td>SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3</td>
</tr>
<tr>
<td>CUDA API</td>
<td>cudaMallocManaged, cudaStreamAttachManagedMem</td>
</tr>
<tr>
<td>Key Concepts</td>
<td>CUDA Systems Integration, OpenMP, CUBLAS, Multithreading, Unified Memory, CUDA Streams and Events</td>
</tr>
<tr>
<td>Supported OSes</td>
<td>Linux, Windows</td>
</tr>
</tbody>
</table>

vectorAdd - 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.

| Supported SM Arch. | SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3 |
| CUDA API           | cudaEventCreate, cudaEventRecord, cudaEventQuery, cudaEventDestroy, cudaEventElapsedTime, cudaEventSynchronize, cudaMalloc, cudaFree, cudaMemcpy |
| Key Concepts       | CUDA Runtime API, Vector Addition |
| Supported OSes     | Linux, Windows, OS X |

vectorAdd_nvrtc - Vector Addition with libNVRTC

This CUDA Driver API sample uses NVRTC for runtime compilation of vector addition kernel. Vector addition kernel demonstrated is the same as the sample illustrating Chapter 3 of the programming guide.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample
will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

### Dependencies
- **NVRTC**

### Supported SM Architecture
- SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

### CUDA API
- cuMemAlloc, cuMemFree, cuMemcpyhToD, cuMemcpyDtoH

### Key Concepts
- CUDA Driver API, Vector Addition, Runtime Compilation

### Supported OSes
- Linux, Windows, OS X

---

**vectorAddDrv - 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.

### Supported SM Architecture
- SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

### CUDA API
- cuModuleLoad, cuModuleLoadDataEx, cuModuleGetFunction, cuMemAlloc, cuMemFree, cuMemcpyhToD, cuMemcpyDtoH, cuLaunchKernel

### Key Concepts
- CUDA Driver API, Vector Addition

### Supported OSes
- Linux, Windows, OS X

---

### 3.2. Utilities Reference

**bandwidthTest - 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.

### Supported SM Architecture
- SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

### CUDA API
- cudaSetDevice, cudaHostAlloc, cudaFree, cudaMemcpy, cudaMemcpyAsync, cudaEventCreate, cudaEventRecord, cudaEventDestroy, cudaDeviceSynchronize, cudaEventElapsedTime

### Key Concepts
- CUDA Streams and Events, Performance Strategies
Supported OSes: Linux, Windows, OS X

**deviceQuery - Device Query**

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

- **Supported SM Architecture**: SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3
- **CUDA API**: cudaSetDevice, cudaGetDeviceCount, cudaGetDeviceProperties, cudaDriverGetVersion, cudaRuntimeGetVersion
- **Key Concepts**: CUDA Runtime API, Device Query
- **Supported OSes**: Linux, Windows, OS X

**deviceQueryDrv - Device Query Driver API**

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

- **Supported SM Architecture**: SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3
- **CUDA API**: cuInit, cuDeviceGetCount, cuDeviceComputeCapability, cuDriverGetVersion, cuDeviceTotalMem, cuDeviceGetAttribute
- **Key Concepts**: CUDA Driver API, Device Query
- **Supported OSes**: Linux, Windows, OS X

**p2pBandwidthLatencyTest - Peer-to-Peer Bandwidth Latency Test with Multi-GPUs**

This application demonstrates the CUDA Peer-To-Peer (P2P) data transfers between pairs of GPUs and computes latency and bandwidth. Tests on GPU pairs using P2P and without P2P are tested.

- **Supported SM Architecture**: SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3
- **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
3.3. Graphics Reference

bindlessTexture - 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.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

**Dependencies**
- X11, GL

**Supported SM Architecture**
- SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**CUDA API**
- cudaGLSetGLDevice, cudaGraphicsMapResources,
- cudaGraphicsUnmapResources, cudaGraphicsResourceGetMappedPointer,
- cudaGraphicsRegisterResource, cudaGraphicsGLRegisterBuffer,
- cudaGraphicsUnregisterResource

**Key Concepts**
- Graphics Interop, Texture

**Supported OSes**
- Linux, Windows, OS X

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 uses double precision. Thanks to Mark Granger of NewTek who submitted this code sample.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

**Dependencies**
- X11, GL

**Supported SM Architecture**
- SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3
marchingCubes - 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.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

**Dependencies**
- X11, GL

**Supported SM Architecture**
- SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**CUDA API**

**Key Concepts**
- OpenGL Graphics Interop, Vertex Buffers, 3D Graphics, Physically Based Simulation

**Supported OSes**
- Linux, Windows, OS X

simpleD3D10 - 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.

**Supported SM Architecture**
- SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**CUDA API**
Key Concepts: Graphics Interop, 3D Graphics

Supported OSes: Windows

**simpleD3D10RenderTarget - Simple Direct3D10 Render Target**

Simple program which demonstrates interop of rendertargets between Direct3D10 and CUDA. The program uses RenderTarget positions with CUDA and generates a histogram with visualization. A Direct3D10 Capable device is required.

- **Supported SM Architecture**: SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3
- **CUDA API**:
  - cudaD3D10GetDevice, cudaD3D10SetDirect3DDevice,
  - cudaGraphicsD3D10RegisterResource, cudaGraphicsResourceSetMapFlags,
  - cudaGraphicsSubResourceGetMappedArray, cudaMemcpy2DToArray,
  - cudaGraphicsUnregisterResource

- **Key Concepts**: Graphics Interop, Texture
- **Supported OSes**: Windows

**simpleD3D10Texture - 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.

- **Supported SM Architecture**: SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3
- **CUDA API**:
  - cudaD3D10GetDevice, cudaD3D10SetDirect3DDevice,
  - cudaGraphicsD3D10RegisterResource, cudaGraphicsResourceSetMapFlags,
  - cudaGraphicsSubResourceGetMappedArray, cudaMemcpy2DToArray,
  - cudaGraphicsUnregisterResource

- **Key Concepts**: Graphics Interop, Texture
- **Supported OSes**: Windows

**simpleD3D11Texture - 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.

**Supported SM Architecture**

SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**CUDA API**
cudaD3D11GetDevice, cudaD3D11SetDirect3DDevice,
cudaGraphicsD3D11RegisterResource, cudaGraphicsResourceSetMapFlags,
cudaGraphicsSubResourceGetMappedArray, cudaMemcpy2DToArray,
cudaGraphicsUnregisterResource

**Key Concepts**
Graphics Interop, Image Processing

**Supported OSes**
Windows

---

**simpleD3D9 - 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.

**Supported SM Architecture**

SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**CUDA API**
cudaD3D9GetDevice, cudaD3D9SetDirect3DDevice,

**Key Concepts**
Graphics Interop

**Supported OSes**
Windows

---

**simpleD3D9Texture - 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.

**Supported SM Architecture**

SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**CUDA API**
cudaD3D9GetDevice, cudaD3D9SetDirect3DDevice,
cudaGraphicsD3D9RegisterResource, cudaGraphicsResourceSetMapFlags,
cudaGraphicsSubResourceGetMappedArray, cudaMemcpy2DToArray,
cudaMemcpy3D, cudaGraphicsUnregisterResource

**Key Concepts**
Graphics Interop, Texture

**Supported OSes**
Windows
simpleGL - 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.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

**Dependencies**
- X11, GL

**Supported SM Architecture**
- SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**CUDA API**

**Key Concepts**
- Graphics Interop, Vertex Buffers, 3D Graphics

**Supported OSes**
- Linux, Windows, OS X

simpleGLES - Simple OpenGLES
Demonstrates data exchange between CUDA and OpenGL ES (aka Graphics interop). The program modifies vertex positions with CUDA and uses OpenGL ES to render the geometry.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

**Dependencies**
- X11, GLES

**Supported SM Architecture**
- SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**CUDA API**

**Key Concepts**
- Graphics Interop, Vertex Buffers, 3D Graphics

**Supported OSes**
- Linux
simpleGLES_screen - Simple OpenGLES on Screen

Demonstrates data exchange between CUDA and OpenGL ES (aka Graphics interop). The program modifies vertex positions with CUDA and uses OpenGL ES to render the geometry.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

Dependencies
screen, GLES

Supported SM
SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

CUDA API
cudaGraphicsMapResources, cudaGraphicsUnmapResources,
cudaGraphicsResourceGetMappedPointer, cudaGraphicsRegisterResource,
cudaGraphicsGLRegisterBuffer, cudaGraphicsUnregisterResource

Key Concepts
Graphics Interop, Vertex Buffers, 3D Graphics

Supported OSes
Linux

simpleTexture3D - Simple Texture 3D

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

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

Dependencies
X11, GL

Supported SM
SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

CUDA API
cudaGLSetGLDevice, cudaGraphicsMapResources,
cudaGraphicsUnmapResources, cudaGraphicsResourceGetMappedPointer,
cudaGraphicsRegisterResource, cudaGraphicsGLRegisterBuffer,
cudaGraphicsUnregisterResource

Key Concepts
Graphics Interop, Image Processing, 3D Textures, Surface Writes

Supported OSes
Linux, Windows, OS X
SLID3D10Texture - 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.

**Supported SM Architecture**
SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**CUDA API**
cudaD3D10GetDevice, cudaD3D10SetDirect3DDevice,
cudaGraphicsD3D10RegisterResource, cudaGraphicsResourceSetMapFlags,
cudaGraphicsSubResourceGetMappedArray, cudaMemcpy2DToArray,
cudaGraphicsUnregisterResource

**Key Concepts**
Performance Strategies, Graphics Interop, Image Processing, 2D Textures

**Supported OSes**
Windows

volumeFiltering - Volumetric Filtering with 3D Textures and Surface Writes

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

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

**Dependencies**
X11, GL

**Supported SM Architecture**
SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**CUDA API**
cudaGLSetGLDevice, cudaGraphicsMapResources,
cudaGraphicsUnmapResources, cudaGraphicsResourceGetMappedPointer,
cudaGraphicsRegisterResource, cudaGraphicsGLRegisterBuffer,
cudaGraphicsUnregisterResource

**Key Concepts**
Graphics Interop, Image Processing, 3D Textures, Surface Writes

**Supported OSes**
Linux, Windows, OS X

volumeRender - Volume Rendering with 3D Textures

This sample demonstrates basic volume rendering using 3D Textures.
This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

**Dependencies**

X11, GL

**Supported SM Architecture**

SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**CUDA API**

cudaGLSetGLDevice, cudaGraphicsMapResources,
cudaGraphicsUnmapResources, cudaGraphicsResourceGetMappedPointer,
cudaGraphicsRegisterResource, cudaGraphicsGLRegisterBuffer,
cudaGraphicsUnregisterResource

**Key Concepts**

Graphics Interop, Image Processing, 3D Textures

**Supported OSes**

Linux, Windows, OS X

### 3.4. Imaging Reference

**bicubicTexture - Bicubic B-spline Interpolation**

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

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

**Dependencies**

X11, GL

**Supported SM Architecture**

SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**CUDA API**

cudaGLSetGLDevice, cudaGraphicsMapResources,
cudaGraphicsUnmapResources, cudaGraphicsResourceGetMappedPointer,
cudaGraphicsRegisterResource, cudaGraphicsGLRegisterBuffer,
cudaGraphicsUnregisterResource

**Key Concepts**

Graphics Interop, Image Processing

**Supported OSes**

Linux, Windows, OS X
bilateralFilter - 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 weighted by considering both the spatial distance and color distance between its neighbors. 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"

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

**Dependencies**
- X11, GL

**Supported SM Architecture**
- SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**CUDA API**

**Key Concepts**
- Graphics Interop, Image Processing

**Supported OSes**
- Linux, Windows, OS X

boxFilter - Box Filter

Fast image box filter using CUDA with OpenGL rendering.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

**Dependencies**
- X11, GL

**Supported SM Architecture**
- SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**CUDA API**

**Key Concepts**
- Graphics Interop, Image Processing
**Supported OSes**
Linux, Windows, OS X

---

**convolutionFFT2D - FFT-Based 2D Convolution**

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

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

**Dependencies**
CUFFT

**Supported SM Architecture**
SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**CUDA API**
cufftPlan2d, cufftExecR2C, cufftExecC2R, cufftDestroy

**Key Concepts**
Image Processing, CUFFT Library

**Supported OSes**
Linux, Windows, OS X

---

**convolutionSeparable - CUDA Separable Convolution**

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

**Supported SM Architecture**
SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**Key Concepts**
Image Processing, Data Parallel Algorithms

**Supported OSes**
Linux, Windows, OS X

**Whitepaper**
convolutionSeparable.pdf

---

**convolutionTexture - Texture-based Separable Convolution**

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

**Supported SM Architecture**
SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**Key Concepts**
Image Processing, Texture, Data Parallel Algorithms
cudaDecodeD3D9 - 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 2.0 or higher.

Supported SM Architecture

Supported OSes

CUDA API

Key Concepts

Supported OSes

Whitepaper

cudaDecodeGL - 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 2.0 or higher.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.
dct8x8 - 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.

Supported SM Architecture

SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

Key Concepts

Image Processing, Video Compression

Supported OSes

Linux, Windows, OS X

Whitepaper

dct8x8.pdf

dwtHaar1D - 1D Discrete Haar Wavelet Decomposition

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

Supported SM Architecture

SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3
Key Concepts: Image Processing, Video Compression
Supported OSes: Linux, Windows, OS X

**dxtc - 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.

- **Supported SM Architecture**: SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3
- **Key Concepts**: Image Processing, Image Compression
- **Supported OSes**: Linux, Windows, OS X
- **Whitepaper**: cuda_dxtc.pdf

**histogram - CUDA Histogram**

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

- **Supported SM Architecture**: SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3
- **Key Concepts**: Image Processing, Data Parallel Algorithms
- **Supported OSes**: Linux, Windows, OS X
- **Whitepaper**: histogram.pdf

**HSOpticalFlow - Optical Flow**

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

- **Supported SM Architecture**: SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3
- **Key Concepts**: Image Processing, Data Parallel Algorithms
- **Supported OSes**: Linux, Windows, OS X
- **Whitepaper**: OpticalFlow.pdf
imageDenoising - Image denoising

This sample demonstrates two adaptive image denoising techniques: 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 technique, taking advantage of shared memory, is implemented in addition to DirectX counterparts.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

**Dependencies**
- X11, GL

**Supported SM Architecture**
- SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**Key Concepts**
- Image Processing

**Supported OSes**
- Linux, Windows, OS X

**Whitepaper**
- imageDenoising.pdf

postProcessGL - Post-Process in OpenGL

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

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

**Dependencies**
- X11, GL

**Supported SM Architecture**
- SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**CUDA API**

**Key Concepts**
- Graphics Interop, Image Processing

**Supported OSes**
- Linux, Windows, OS X
recursiveGaussian - 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.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

**Dependencies**
- X11, GL

**Supported SM Architecture**
- SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**CUDA API**
- cudaGLSetGLDevice, cudaGraphicsMapResources,
- cudaGraphicsUnmapResources, cudaGraphicsResourceGetMappedPointer,
- cudaGraphicsRegisterResource, cudaGraphicsGLRegisterBuffer,
- cudaGraphicsUnregisterResource

**Key Concepts**
- Graphics Interop, Image Processing

**Supported OSes**
- Linux, Windows, OS X

simpleCUDA2GL - CUDA and OpenGL Interop of Images

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

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

**Dependencies**
- X11, GL

**Supported SM Architecture**
- SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**CUDA API**
- cudaGLSetGLDevice, cudaGraphicsMapResources,
- cudaGraphicsUnmapResources, cudaGraphicsResourceGetMappedPointer,
- cudaGraphicsRegisterResource, cudaGraphicsGLRegisterBuffer,
- cudaGraphicsUnregisterResource

**Key Concepts**
- Graphics Interop, Image Processing, Performance Strategies

**Supported OSes**
- Linux, Windows, OS X
SobelFilter - Sobel Filter

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

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

**Dependencies**
- X11, GL

**Supported SM Architecture**
- SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**CUDA API**
- cudaGLSetGLDevice, cudaGraphicsMapResources,
  - cudaGraphicsUnmapResources, cudaGraphicsResourceGetMappedPointer,
  - cudaGraphicsRegisterResource, cudaGraphicsGLRegisterBuffer,
  - cudaGraphicsUnregisterResource

**Key Concepts**
- Graphics Interop, Image Processing

**Supported OSes**
- Linux, Windows, OS X

stereoDisparity - 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.

**Supported SM Architecture**
- SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**Key Concepts**
- Image Processing, Video Intrinsics

**Supported OSes**
- Linux, Windows, OS X

3.5. Finance Reference

binomialOptions - Binomial Option Pricing

This sample evaluates fair call price for a given set of European options under binomial model.
binomialOptions_nvrtc - Binomial Option Pricing with libNVRTC

This sample evaluates fair call price for a given set of European options under binomial model. This sample makes use of NVRTC for Runtime Compilation.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

Dependencies  NVRTC
Supported SM  SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3
Architecture  Computational Finance
Supported OSes  Linux, Windows, OS X
Whitepaper  binomialOptions.pdf

BlackScholes - Black-Scholes Option Pricing

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

Supported SM  SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3
Architecture  Computational Finance
Supported OSes  Linux, Windows, OS X
Whitepaper  BlackScholes.pdf
BlackScholes_nvrtc - Black-Scholes Option Pricing with libNVRTC

This sample evaluates fair call and put prices for a given set of European options by Black-Scholes formula, compiling the CUDA kernels involved at runtime using NVRTC. This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

Dependencies              | NVRTC
Supported SM Architecture  | SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3
Key Concepts               | Computational Finance
Supported OSes             | Linux, Windows, OS X
Whitepaper                 | BlackScholes.pdf

MonteCarloMultiGPU - 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. This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

Dependencies              | CURAND
Supported SM Architecture  | SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3
Supported OSes             | Linux, Windows, OS X
Whitepaper                 | MonteCarlo.pdf
quasirandomGenerator - Niederreiter Quasirandom Sequence Generator

This sample implements Niederreiter Quasirandom Sequence Generator and Inverse Cumulative Normal Distribution functions for the generation of Standard Normal Distributions.


Key Concepts: Computational Finance

Supported OSes: Linux, Windows, OS X

quasirandomGenerator_nvrtc - Niederreiter Quasirandom Sequence Generator with libNVTC

This sample implements Niederreiter Quasirandom Sequence Generator and Inverse Cumulative Normal Distribution functions for the generation of Standard Normal Distributions, compiling the CUDA kernels involved at runtime using NVRTC.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

Dependencies: NVRTC


Key Concepts: Computational Finance, Runtime Compilation

Supported OSes: Linux, Windows, OS X

SobolQRNG - Sobol Quasirandom Number Generator

This sample implements Sobol Quasirandom Sequence Generator.


Key Concepts: Computational Finance

Supported OSes: Linux, Windows, OS X
3.6. Simulations Reference

**fluidsD3D9 - Fluids (Direct3D Version)**

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

- **Supported SM Architecture**: SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3
- **Key Concepts**: Graphics Interop, CUFFT Library, Physically-Based Simulation
- **Supported OSes**: Windows

**fluidsGL - Fluids (OpenGL Version)**

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

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

- **Dependencies**: X11, GL, CUFFT
- **Supported SM Architecture**: SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3
- **Key Concepts**: Graphics Interop, CUFFT Library, Physically-Based Simulation
- **Supported OSes**: Linux, Windows, OS X
- **Whitepaper**: fluidsGL.pdf
nbody - 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.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

Dependencies
X11, GL

Supported SM
SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

CUDA API

Key Concepts
Graphics Interop, Data Parallel Algorithms, Physically-Based Simulation

Supported OSes
Linux, Windows, OS X

Whitepaper
nbody_gems3_ch31.pdf

nbody_screen - CUDA N-Body Simulation on Screen

This sample demonstrates efficient all-pairs simulation of a gravitational n-body simulation in CUDA. Unlike the OpenGL nbody sample, there is no user interaction. This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.
oceanFFT - CUDA FFT Ocean Simulation

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

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

particles - 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

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample
will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

<table>
<thead>
<tr>
<th>Dependencies</th>
<th>X11, GL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported SM</td>
<td>SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3</td>
</tr>
<tr>
<td>Architecture</td>
<td></td>
</tr>
<tr>
<td>Key Concepts</td>
<td>Graphics Interop, Data Parallel Algorithms, Physically-Based Simulation, Performance Strategies</td>
</tr>
<tr>
<td>Supported OSes</td>
<td>Linux, Windows, OS X</td>
</tr>
<tr>
<td>Whitepaper</td>
<td>particles.pdf</td>
</tr>
</tbody>
</table>

**smokeParticles - 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.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

<table>
<thead>
<tr>
<th>Dependencies</th>
<th>X11, GL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported SM</td>
<td>SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3</td>
</tr>
<tr>
<td>Architecture</td>
<td></td>
</tr>
<tr>
<td>Key Concepts</td>
<td>Graphics Interop, Data Parallel Algorithms, Physically-Based Simulation</td>
</tr>
<tr>
<td>Supported OSes</td>
<td>Linux, Windows, OS X</td>
</tr>
<tr>
<td>Whitepaper</td>
<td>smokeParticles.pdf</td>
</tr>
</tbody>
</table>
VFlockingD3D10

The sample models formation of V-shaped flocks by big birds, such as geese and cranes. The algorithms of such flocking are borrowed from the paper "V-like formations in flocks of artificial birds" from Artificial Life, Vol. 14, No. 2, 2008. The sample has CPU- and GPU-based implementations. Press 'g' to toggle between them. The GPU-based simulation works many times faster than the CPU-based one. The printout in the console window reports the simulation time per step. Press 'r' to reset the initial distribution of birds.

Supported SM Architecture
SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

CUDA API
cudaD3D10SetGLDevice, cudaGraphicsMapResources,
cudaGraphicsUnmapResources, cudaGraphicsResourceGetMappedPointer,
cudaGraphicsRegisterResource, cudaGraphicsGLRegisterBuffer,
cudaGraphicsUnregisterResource

Key Concepts
Graphics Interop, Data Parallel Algorithms, Physically-Based Simulation,
Performance Strategies

Supported OSes
Windows

3.7. Advanced Reference
alignedTypes - Aligned Types

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

Supported SM Architecture
SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

Key Concepts
Performance Strategies

Supported OSes
Linux, Windows, OS X

cdpAdvancedQuicksort - 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.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample
will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

<table>
<thead>
<tr>
<th>Dependencies</th>
<th>CDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported SM</td>
<td>SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3</td>
</tr>
<tr>
<td>Architecture</td>
<td>CUDA Dynamic Parallelism</td>
</tr>
<tr>
<td>Supported OSes</td>
<td>Linux, Windows, OS X</td>
</tr>
</tbody>
</table>

**cdpBezierTessellation - Bezier Line Tessellation (CUDA Dynamic Parallelism)**

This sample demonstrates bezier tessellation of lines implemented using CUDA Dynamic Parallelism. This sample requires devices with compute capability 3.5 or higher.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

<table>
<thead>
<tr>
<th>Dependencies</th>
<th>CDP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported SM</td>
<td>SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3</td>
</tr>
<tr>
<td>Architecture</td>
<td>CUDA Dynamic Parallelism</td>
</tr>
<tr>
<td>Supported OSes</td>
<td>Linux, Windows, OS X</td>
</tr>
</tbody>
</table>

**cdpLUDecomposition - 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.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

<table>
<thead>
<tr>
<th>Dependencies</th>
<th>CDP, CUBLAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported SM</td>
<td>SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3</td>
</tr>
</tbody>
</table>
cdpQuadtree - 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.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

Dependencies  CDP
Supported SM  SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3
Architecture
Key Concepts  CUDA Dynamic Parallelism
Supported OSes  Linux, Windows, OS X

concurrentKernels - 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

Supported SM  SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3
Architecture
Key Concepts  Performance Strategies
Supported OSes  Linux, Windows, OS X

eigenvalues - 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.

Supported SM  SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3
Architecture
fastWalshTransform - Fast Walsh Transform

Naturally(Hadamard)-ordered Fast Walsh Transform for batching vectors of arbitrary eligible lengths that are power of two in size.

Supported SM Architecture SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

Key Concepts Linear Algebra, Data-Parallel Algorithms, Video Compression

Supported OSes Linux, Windows, OS X

FDTD3d - CUDA C 3D FDTD

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

Supported SM Architecture SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

Key Concepts Performance Strategies

Supported OSes Linux, Windows, OS X

FunctionPointers - Function Pointers

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

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

Dependencies X11, GL

Supported SM Architecture SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

Key Concepts Graphics Interop, Image Processing

Supported OSes Linux, Windows, OS X
interval - Interval Computing

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

Supported SM Architecture
SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

Key Concepts
Recursion, Templates

Supported OSes
Linux, Windows, OS X

lineOfSight - 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/).

Supported SM Architecture
SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

Supported OSes
Linux, Windows, OS X

matrixMulDynlinkJIT - 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.

Supported SM Architecture
SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

CUDA API
cuModuleLoad, cuModuleLoadDataEx, cuModuleGetFunction, cuMemAlloc, cuMemFree, cudaMemcpyHtoD, cudaMemcpyDtoH, cuLaunchKernel

Key Concepts
CUDA Driver API, CUDA Dynamically Linked Library

Supported OSes
Linux, Windows, OS X
mergeSort - 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

Supported SM Architecture
SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

Key Concepts
Data-Parallel Algorithms

Supported OSes
Linux, Windows, OS X

newdelete - NewDelete

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

Supported SM Architecture
SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

Supported OSes
Linux, Windows, OS X

ptxjit - 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 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.

Supported SM Architecture
SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

Key Concepts
CUDA Driver API

Supported OSes
Linux, Windows, OS X

radixSortThrust - 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.

**reduction - 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 Data-Parallel Algorithms like reduction.

**scalarProd - Scalar Product**

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

- **Supported SM Architecture**: SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3
- **Key Concepts**: Data-Parallel Algorithms, Performance Strategies
- **Supported OSes**: Linux, Windows, OS X

segmentationTreeThrust - 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.

- **Supported SM Architecture**: SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3
- **Key Concepts**: Data-Parallel Algorithms, Performance Strategies
- **Supported OSes**: Linux, Windows, OS X

shfl_scan - 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.

- **Supported SM Architecture**: SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3
- **Key Concepts**: Data-Parallel Algorithms, Performance Strategies
- **Supported OSes**: Linux, Windows, OS X

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

StreamPriorities - Stream Priorities

This sample demonstrates basic use of stream priorities.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

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" CUDA Sample). Single-pass reduction requires global atomic instructions (Compute Capability 2.0 or later) and the _threadfence() intrinsic (CUDA 2.2 or later).

| Supported SM | SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3 |
| Supported Architecture | |
| Key Concepts | Data-Parallel Algorithms, Performance Strategies |
| Supported OSes | Linux, Windows, OS X |

**threadMigration - CUDA Context Thread Management**

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

| Supported SM | SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3 |
| Supported Architecture | |
| CUDA API | cuCtxCreate, cuCtxDestroy, cuModuleLoad, cuModuleLoadDataEx, cuModuleGetFunction, cuLaunchKernel, cuMemcpyDtoH, cuCtxPushCurrent, cuCtxPopCurrent |
| Key Concepts | CUDA Driver API |
| Supported OSes | Linux, Windows, OS X |

**transpose - Matrix Transpose**

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

| Supported SM | SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3 |
| Supported Architecture | |
| Key Concepts | Performance Strategies, Linear Algebra |
| Supported OSes | Linux, Windows, OS X |
| Whitepaper | MatrixTranspose.pdf |
3.8. Cudalibraries Reference

batchCUBLAS

A CUDA Sample that demonstrates how using batched CUBLAS API calls to improve overall performance.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

Dependencies
- CUBLAS

Supported SM
- SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

Architecture

Key Concepts
- Linear Algebra, CUBLAS Library

Supported OSes
- Linux, Windows, OS X

boxFilterNPP - Box Filter with NPP

A NPP CUDA Sample that demonstrates how to use NPP FilterBox function to perform a Box Filter.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

Dependencies
- FreeImage, NPP

Supported SM
- SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

Architecture

Key Concepts
- Performance Strategies, Image Processing, NPP Library

Supported OSes
- Linux, Windows, OS X

conjugateGradient - ConjugateGradient

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

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample
will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

<table>
<thead>
<tr>
<th>Dependencies</th>
<th>CUBLAS, CUSPARSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported SM</td>
<td>SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3</td>
</tr>
<tr>
<td>Architecture</td>
<td></td>
</tr>
<tr>
<td>Key Concepts</td>
<td>Linear Algebra, CUBLAS Library, CUSPARSE Library</td>
</tr>
<tr>
<td>Supported OSes</td>
<td>Linux, Windows, OS X</td>
</tr>
</tbody>
</table>

**conjugateGradientPrecond - Preconditioned Conjugate Gradient**

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

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

<table>
<thead>
<tr>
<th>Dependencies</th>
<th>CUBLAS, CUSPARSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported SM</td>
<td>SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3</td>
</tr>
<tr>
<td>Architecture</td>
<td></td>
</tr>
<tr>
<td>Key Concepts</td>
<td>Linear Algebra, CUBLAS Library, CUSPARSE Library</td>
</tr>
<tr>
<td>Supported OSes</td>
<td>Linux, Windows, OS X</td>
</tr>
</tbody>
</table>

**conjugateGradientUM - ConjugateGradientUM**

This sample implements a conjugate gradient solver on GPU using CUBLAS and CUSPARSE library, using Unified Memory

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

<table>
<thead>
<tr>
<th>Dependencies</th>
<th>UVM, CUBLAS, CUSPARSE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported SM</td>
<td>SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3</td>
</tr>
<tr>
<td>Architecture</td>
<td></td>
</tr>
<tr>
<td>Key Concepts</td>
<td>Unified Memory, Linear Algebra, CUBLAS Library, CUSPARSE Library</td>
</tr>
<tr>
<td>Supported OSes</td>
<td>Linux, Windows</td>
</tr>
</tbody>
</table>
cuHook - CUDA Interception Library

This sample demonstrates how to build and use an intercept library with CUDA.


Supported OSes: Linux

freemageInteropNPP - FreeImage and NPP Interopability

A simple CUDA Sample demonstrate how to use FreeImage library with NPP.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

Dependencies: FreeImage, NPP


Key Concepts: Performance Strategies, Image Processing, NPP Library

Supported OSes: Linux, Windows, OS X

grabcutNPP - 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)

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

Dependencies: X11, GL, FreeImage, NPP


Key Concepts: Performance Strategies, Image Processing, NPP Library

Supported OSes: Linux, Windows, OS X
histEqualizationNPP - Histogram Equalization with NPP

This CUDA Sample demonstrates how to use NPP for histogram equalization for image data.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

Dependencies: Freimage, NPP
Supported SM: SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3
Architecture
Key Concepts: Image Processing, Performance Strategies, NPP Library
Supported OSes: Linux, Windows, OS X

imageSegmentationNPP - Image Segmentation using Graphcuts with NPP

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

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

Dependencies: Freimage, NPP
Supported SM: SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3
Architecture
Key Concepts: Image Processing, Performance Strategies, NPP Library
Supported OSes: Linux, Windows, OS X

jpegNPP - JPEG encode/decode and resize with NPP

This sample demonstrates a simple image processing pipeline. First, a JPEG file is huffman decoded and inverse DCT transformed and dequantized. Then the different planes are resized. Finally, the resized image is quantized, forward DCT transformed and huffman encoded.
This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

**Dependencies**
- FreeImage, NPP

**Supported SM Architecture**
- SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**CUDA API**
- nppiGetGpuComputeCapability, nppiDCTInitAlloc,
  nppiDecodeHuffmanScanHost_JPEG_8u16s_P3R,
  nppiDCTQuantInv8x8LS_JPEG_16s8u_C1R_NEW, nppiResizeSqrPixel_8u_C1R,
  nppiEncodeHuffmanGetSize, nppiDCTFree

**Supported OSes**
- Linux, Windows, OS X

---

**MC_EstimatePiInlineP - 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.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

**Dependencies**
- CURAND

**Supported SM Architecture**
- SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3

**Key Concepts**
- Random Number Generator, Computational Finance, CURAND Library

**Supported OSes**
- Linux, Windows, OS X

---

**MC_EstimatePiInlineQ - Monte Carlo Estimation of Pi (inline QRNG)**

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

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.
MC_EstimatePiP - 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.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

 Dependencies: CURAND
 Key Concepts: Random Number Generator, Computational Finance, CURAND Library
 Supported OSes: Linux, Windows, OSX

MC_EstimatePiQ - 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.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

 Dependencies: CURAND
 Key Concepts: Random Number Generator, Computational Finance, CURAND Library
 Supported OSes: Linux, Windows, OSX
MC_SingleAsianOptionP - Monte Carlo Single Asian Option

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

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

- **Dependencies**: CURAND
- **Supported SM Architecture**: SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3
- **Key Concepts**: Random Number Generator, Computational Finance, CURAND Library
- **Supported OSes**: Linux, Windows, OS X

MersenneTwisterGP11213

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

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

- **Dependencies**: CURAND
- **Supported SM Architecture**: SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3
- **Key Concepts**: Computational Finance, CURAND Library
- **Supported OSes**: Linux, Windows, OS X

randomFog - Random Fog

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

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.
simpleCUBLAS - Simple CUBLAS

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

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

<table>
<thead>
<tr>
<th>Dependencies</th>
<th>CUBLAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported SM</td>
<td>SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3</td>
</tr>
<tr>
<td>Architecture</td>
<td></td>
</tr>
<tr>
<td>Key Concepts</td>
<td>Image Processing, CUBLAS Library</td>
</tr>
<tr>
<td>Supported OSes</td>
<td>Linux, Windows, OS X</td>
</tr>
</tbody>
</table>

simpleCUFFT - 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.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

<table>
<thead>
<tr>
<th>Dependencies</th>
<th>CUFFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported SM</td>
<td>SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3</td>
</tr>
<tr>
<td>Architecture</td>
<td></td>
</tr>
<tr>
<td>Key Concepts</td>
<td>Image Processing, CUFFT Library</td>
</tr>
<tr>
<td>Supported OSes</td>
<td>Linux, Windows, OS X</td>
</tr>
</tbody>
</table>
simpleCUFFT_2d_MGPU - SimpleCUFFT_2d_MGPU

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 on Multiple GPU.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

<table>
<thead>
<tr>
<th>Dependencies</th>
<th>CUFFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported SM</td>
<td>SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3</td>
</tr>
<tr>
<td>Architecture</td>
<td></td>
</tr>
<tr>
<td>Key Concepts</td>
<td>Image Processing, CUFFT Library</td>
</tr>
<tr>
<td>Supported OSes</td>
<td>Linux, Windows, OS X</td>
</tr>
</tbody>
</table>

simpleCUFFT_callback - Simple CUFFT Callbacks

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. The difference between this example and the Simple CUFFT example is that the multiplication step is done by the CUFFT kernel with a user-supplied CUFFT callback routine, rather than by a separate kernel call.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

<table>
<thead>
<tr>
<th>Dependencies</th>
<th>callback, CUFFT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supported SM</td>
<td>SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 5.0, SM 5.2, SM 5.3</td>
</tr>
<tr>
<td>Architecture</td>
<td></td>
</tr>
<tr>
<td>Key Concepts</td>
<td>Image Processing, CUFFT Library</td>
</tr>
<tr>
<td>Supported OSes</td>
<td>Linux</td>
</tr>
</tbody>
</table>
simpleCUFFT_MGPU - Simple CUFFT_MGPU

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 on Multiple GPU.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

Dependencies: CUFFT
Supported SM: SM 2.0, SM 3.0, SM 3.2, SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3
Key Concepts: Image Processing, CUFFT Library
Supported OSes: Linux, Windows, OS X

simpleDevLibCUBLAS - 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.

This sample depends on other applications or libraries to be present on the system to either build or run. If these dependencies are not available on the system, the sample will not be installed. If these dependencies are available, but not installed, the sample will waive itself at build time.

Dependencies: CDP, CUBLAS
Supported SM: SM 3.5, SM 3.7, SM 5.0, SM 5.2, SM 5.3
CUDA API: cublasCreate, cublasSetVector, cublasSgemm, cudaMalloc, cudaFree, cudaMemcpy
Key Concepts: CUDA Dynamic Parallelism, Linear Algebra
Supported OSes: Linux, Windows, OS X
Chapter 4.
DEPENDENCIES

Some CUDA Samples rely on third-party applications and/or libraries, or features provided by the CUDA Toolkit and Driver, to either build or execute. These dependencies are listed below.

If a sample has a dependency that is not available on the system, the sample will not be installed. If a sample has a third-party dependency that is available on the system, but is not installed, the sample will waive itself at build time.

Each sample’s dependencies are listed in the Samples Reference section.

Third-Party Dependencies

These third-party dependencies are required by some CUDA samples. If available, these dependencies are either installed on your system automatically, or are installable via your system’s package manager (Linux) or a third-party website.

FreImage

FreImage is an open source imaging library. FreImage can usually be installed on Linux using your distribution’s package manager system. FreImage can also be downloaded from the FreImage website. FreImage is also redistributed with the CUDA Samples.

Message Passing Interface

MPI (Message Passing Interface) is an API for communicating data between distributed processes. A MPI compiler can be installed using your Linux distribution’s package manager system. It is also available on some online resources, such as Open MPI.

Only 64-Bit

Some samples can only be run on a 64-bit operating system.
DirectX

DirectX is a collection of APIs designed to allow development of multimedia applications on Microsoft platforms. For Microsoft platforms, NVIDIA’s CUDA Driver supports DirectX. Several CUDA Samples for Windows demonstrates CUDA-DirectX Interoperability, for building such samples one needs to install DirectX SDK (June 2010 or newer), this is required to be installed only on Windows 7 and Windows Server 2008, Other Windows OSes do not need to explicitly install the DirectX SDK.

Samples which require the DirectX SDK include: `simpleD3D9`, `simpleD3D9Texture`, `simpleD3D10`, `simpleD3D10RenderTarget`, `simpleD3D10Texture`, `simpleD3D11Texture`, `SLID3D10Texture`, `cudaDecodeD3D9`, `fluidsD3D9`, and `VFlockingD3D10`.

OpenGL

OpenGL is a graphics library used for 2D and 3D rendering. On systems which support OpenGL, NVIDIA’s OpenGL implementation is provided with the CUDA Driver.

OpenGL ES

OpenGL ES is an embedded systems graphics library used for 2D and 3D rendering. On systems which support OpenGL ES, NVIDIA’s OpenGL ES implementation is provided with the CUDA Driver.

OpenMP

OpenMP is an API for multiprocessing programming. OpenMP can be installed using your Linux distribution’s package manager system. It usually comes preinstalled with GCC. It can also be found at the OpenMP website.

Screen

Screen is a windowing system found on the QNX operating system. Screen is usually found as part of the root filesystem.

X11

X11 is a windowing system commonly found on *nix style operating systems. X11 can be installed using your Linux distribution’s package manager, and comes preinstalled on Mac OS X systems.

CUDA Features

These CUDA features are needed by some CUDA samples. They are provided by either the CUDA Toolkit or CUDA Driver. Some features may not be available on your system.
**CUFFT Callback Routines**

CUFFT Callback Routines are user-supplied kernel routines that CUFFT will call when loading or storing data. These callback routines are only available on Linux x86_64 and ppc64le systems.

**CUDA Dynamic Parallelism**

CDP (CUDA Dynamic Parallelism) allows kernels to be launched from threads running on the GPU. CDP is only available on GPUs with SM architecture of 3.5 or above.

**CUBLAS**

CUBLAS (CUDA Basic Linear Algebra Subroutines) is a GPU-accelerated version of the BLAS library.

**CUDA Interprocess Communication**

IPC (Interprocess Communication) allows processes to share device pointers. IPC is only available on Linux x86_64 and ppc64le systems.

**CUFFT**

CUFFT (CUDA Fast Fourier Transform) is a GPU-accelerated FFT library.

**CURAND**

CURAND (CUDA Random Number Generation) is a GPU-accelerated RNG library.

**CUSPARSE**

CUSPARSE (CUDA Sparse Matrix) provides linear algebra subroutines used for sparse matrix calculations.

**NPP**

NPP (NVIDIA Performance Primitives) provides GPU-accelerated image, video, and signal processing functions.

**NVRTC**

NVRTC (CUDA RunTime Compilation) is a runtime compilation library for CUDA C++.

**NVCUVID**

NVCUVID (NVIDIA CUDA Video Decoder) provides GPU-accelerated video decoding capabilities.
Stream Priorities
Stream Priorities allows the creation of streams with specified priorities. Stream Priorities is only available on GPUs with SM architecture of 3.5 or above.

Unified Virtual Memory
UVM (Unified Virtual Memory) enables memory that can be accessed by both the CPU and GPU without explicit copying between the two. UVM is only available on Linux and Windows systems.
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 2  Basic Key Concepts and Associated Samples

<table>
<thead>
<tr>
<th>Basic Key Concept</th>
<th>Description</th>
<th>Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D Graphics</td>
<td>3D Rendering</td>
<td>Random Fog, Simple Direct3D10 (Vertex Array), Simple OpenGL, Simple OpenGLES, Simple OpenGLES on Screen</td>
</tr>
<tr>
<td>3D Textures</td>
<td>Volume Textures</td>
<td>Simple Texture 3D</td>
</tr>
<tr>
<td>Assert</td>
<td>GPU Assert</td>
<td>simpleAssert, simpleAssert with libNVRTC</td>
</tr>
<tr>
<td>Asynchronous Data Transfers</td>
<td>Overlapping I/O and Compute</td>
<td>Peer-to-Peer Bandwidth Latency Test with Multi-GPUs, Simple Multi Copy and Compute, Simple Multi-GPU, Simple Peer-to-Peer Transfers with Multi-GPU, asyncAPI, simpleStreams</td>
</tr>
<tr>
<td>Atomic Intrinsics</td>
<td>Using atomics with GPU kernels</td>
<td>Simple Atomic Intrinsics, Simple Atomic Intrinsics with libNVRTC</td>
</tr>
<tr>
<td>C++ Function Overloading</td>
<td>Use C++ overloading with GPU kernels</td>
<td>cppOverload</td>
</tr>
<tr>
<td>Basic Key Concept</td>
<td>Description</td>
<td>Samples</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>C++ Templates</td>
<td>Using Templates with GPU kernels</td>
<td>Simple Templates, Simple Templates with libNVRTC</td>
</tr>
<tr>
<td>CUBLAS</td>
<td>CUDA BLAS samples</td>
<td>Matrix Multiplication (CUBLAS), Unified Memory Streams</td>
</tr>
<tr>
<td>CUBLAS Library</td>
<td>CUDA BLAS samples</td>
<td>Simple CUBLAS, batchCUBLAS</td>
</tr>
<tr>
<td>CUDA Data Transfers</td>
<td>CUDA Data I/O</td>
<td>Template using CUDA Runtime</td>
</tr>
<tr>
<td>CUDA Driver API</td>
<td>Samples that show the CUDA Driver API</td>
<td>Device Query Driver API, Matrix Multiplication (CUDA Driver API Version), Simple Texture (Driver Version), Simple Vote Intrinsics with libNVRTC, Using Inline PTX, Using Inline PTX with libNVRTC, Vector Addition Driver API, Vector Addition with libNVRTC</td>
</tr>
<tr>
<td>CUDA Dynamic Parallelism</td>
<td>Dynamic Parallelism with GPU Kernels (SM 3.5)</td>
<td>Simple Print (CUDA Dynamic Parallelism), simpleDevLibCUBLAS GPU Device API Library Functions (CUDA Dynamic Parallelism)</td>
</tr>
<tr>
<td>CUDA Runtime API</td>
<td>Samples that use the Runtime API</td>
<td>Device Query, Matrix Multiplication (CUBLAS), Matrix Multiplication (CUDA Runtime API Version), Matrix Multiplication with libNVRTC, Simple Texture, Vector Addition</td>
</tr>
<tr>
<td>CUDA Streams</td>
<td>Stream API defines a sequence of operations that can be overlapped with I/O</td>
<td>Simple CUDA Callbacks</td>
</tr>
<tr>
<td>CUDA Streams and Events</td>
<td>Synchronizing Kernels with Event Timers and Streams</td>
<td>Bandwidth Test, Simple Multi Copy and Compute, Simple Multi-GPU, Unified Memory Streams, asyncAPI, cppOverload, simpleStreams</td>
</tr>
<tr>
<td>CUDA Systems Integration</td>
<td>Samples that integrate with Multi Process (OpenMP, IPC, and MPI)</td>
<td>Unified Memory Streams, cudaOpenMP, simpleIPC, simpleMPI</td>
</tr>
<tr>
<td>Basic Key Concept</td>
<td>Description</td>
<td>Samples</td>
</tr>
<tr>
<td>------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CUFFT Library</td>
<td>Samples that use the CUDA FFT accelerated library</td>
<td>Simple CUFFT, Simple CUFFT Callbacks, Simple CUFFT_MGPU, SimpleCUFFT_2d_MGPU</td>
</tr>
<tr>
<td>CURAND Library</td>
<td>Samples that use the CUDA random number generator</td>
<td>MersenneTwisterGP11213, Random Fog</td>
</tr>
<tr>
<td>Callback Functions</td>
<td>Creating Callback functions with GPU kernels</td>
<td>Simple CUDA Callbacks</td>
</tr>
<tr>
<td>Computational Finance</td>
<td>Finance Algorithms</td>
<td>Black-Scholes Option Pricing, Black-Scholes Option Pricing with libNVRTC, MersenneTwisterGP11213</td>
</tr>
<tr>
<td>Data Parallel Algorithms</td>
<td>Samples that show good usage of Data Parallel Algorithms</td>
<td>CUDA Separable Convolution, Texture-based Separable Convolution</td>
</tr>
<tr>
<td>Debugging</td>
<td>Samples useful for debugging</td>
<td>simplePrintf</td>
</tr>
<tr>
<td>Device Memory Allocation</td>
<td>Samples that show GPU Device side memory allocation</td>
<td>Template, Template using CUDA Runtime</td>
</tr>
<tr>
<td>Device Query</td>
<td>Sample showing simple device query of information</td>
<td>Device Query, Device Query Driver API</td>
</tr>
<tr>
<td>GPU Performance</td>
<td>Samples demonstrating high performance and data I/O</td>
<td>Simple Multi Copy and Compute</td>
</tr>
<tr>
<td>Graphics Interop</td>
<td>Samples that demonstrate interop between graphics APIs and CUDA</td>
<td>Bicubic B-spline Interpolation, Bilateral Filter, Box Filter, CUDA and OpenGL Interop of Images, Simple D3D10 Texture, Simple D3D11 Texture, Simple D3D9 Texture, Simple Direct3D10 (Vertex Arrays), Simple Direct3D10 Render Target, Simple Direct3D9 (Vertex Arrays), Simple OpenGL, Simple OpenGLES, Simple OpenGLES on Screen, Simple Texture 3D</td>
</tr>
<tr>
<td>Image Processing</td>
<td>Samples that demonstrate image processing algorithms in CUDA</td>
<td>Bicubic B-spline Interpolation, Bilateral Filter, Box Filter, Box Filter with NPP, CUDA Separable Convolution, CUDA and OpenGL Interop of Images, Freimage and NPP Interopability, GrabCut with NPP, Histogram Equalization with NPP, Image Segmentation</td>
</tr>
<tr>
<td>Basic Key Concept</td>
<td>Description</td>
<td>Samples</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td><strong>Key Concepts and Associated Samples</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Basic Key Concept</strong></td>
<td><strong>Description</strong></td>
<td><strong>Samples</strong></td>
</tr>
<tr>
<td>InterProcess Communication</td>
<td><em>Samples that demonstrate Inter Process Communication between processes</em></td>
<td>simpleIPC</td>
</tr>
<tr>
<td>Linear Algebra</td>
<td><em>Samples demonstrating linear algebra with CUDA</em></td>
<td>Matrix Multiplication (CUBLAS), Matrix Multiplication (CUDA Runtime API Version), Matrix Multiplication with libNVRTC, batchCUBLAS, simpleDevLibCUBLAS GPU Device API Library Functions (CUDA Dynamic Parallelism)</td>
</tr>
<tr>
<td>MPI</td>
<td><em>Samples demonstrating how to use CUDA with MPI programs</em></td>
<td>simpleMPI</td>
</tr>
<tr>
<td>Matrix Multiply</td>
<td><em>Samples demonstrating matrix multiply CUDA</em></td>
<td>Matrix Multiplication (CUDA Driver API Version)</td>
</tr>
<tr>
<td>Multi-GPU</td>
<td><em>Samples demonstrating how to take advantage of multiple GPUs and CUDA</em></td>
<td>Peer-to-Peer Bandwidth Latency Test with Multi-GPUs, Simple Multi-GPU, Simple Peer-to-Peer Transfers with Multi-GPU</td>
</tr>
<tr>
<td>Multithreading</td>
<td><em>Samples demonstrating how to use multithreading with CUDA</em></td>
<td>Simple CUDA Callbacks, Simple Multi-GPU, Unified Memory Streams, cudaOpenMP, simpleMPI</td>
</tr>
<tr>
<td>NPP Library</td>
<td><em>Samples demonstrating how to use NPP (NVIDIA Performance Primitives) for image processing</em></td>
<td>Box Filter with NPP, FreeImage and NPP Interopability, GrabCut with NPP, Histogram Equalization with NPP, Image Segmentation using Graphcuts with NPP</td>
</tr>
<tr>
<td>Occupancy Calculator</td>
<td></td>
<td>simpleOccupancy</td>
</tr>
<tr>
<td>OpenMP</td>
<td><em>Samples demonstrating how to use OpenMP</em></td>
<td>Unified Memory Streams, cudaOpenMP</td>
</tr>
<tr>
<td>Basic Key Concept</td>
<td>Description</td>
<td>Samples</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>Overlap Compute and Copy</td>
<td>Samples demonstrating how to overlap Compute and Data I/O</td>
<td>Simple Multi Copy and Compute</td>
</tr>
<tr>
<td>PTX Assembly</td>
<td>Samples demonstrating how to use PTX code with CUDA</td>
<td>Using Inline PTX, Using Inline PTX with libNVRTC</td>
</tr>
<tr>
<td>Peer to Peer</td>
<td>Samples demonstrating how to handle P2P data transfers between multiple GPUs</td>
<td>simpleIPC</td>
</tr>
<tr>
<td>Peer to Peer Data Transfers</td>
<td>Samples demonstrating how to handle P2P data transfers between multiple GPUs</td>
<td>Peer-to-Peer Bandwidth Latency Test with Multi-GPUs, Simple Peer-to-Peer Transfers with Multi-GPU</td>
</tr>
<tr>
<td>Performance Strategies</td>
<td>Samples demonstrating high performance with CUDA</td>
<td>Bandwidth Test, Box Filter with NPP, CUDA and OpenGL Interop of Images, Clock, Clock libNVRTC, Freemeage and NPP Interopability, GrabCut with NPP, Histogram Equalization with NPP, Image Segmentation using Graphcuts with NPP, Matrix Multiplication (CUBLAS), Peer-to-Peer Bandwidth Latency Test with Multi-GPUs, Simple Peer-to-Peer Transfers with Multi-GPU, Using Inline PTX, Using Inline PTX with libNVRTC, simpleZeroCopy</td>
</tr>
<tr>
<td>Pinned System Paged Memory</td>
<td>Samples demonstrating how to properly handle data I/O efficiently between the CPU host and GPU video memory</td>
<td>simpleZeroCopy</td>
</tr>
<tr>
<td>Separate Compilation</td>
<td>Samples demonstrating how to use CUDA library linking</td>
<td>Simple Static GPU Device Library</td>
</tr>
<tr>
<td>Surface Writes</td>
<td>Samples demonstrating how to use Surface Writes with GPU kernels</td>
<td>Simple Surface Write, Simple Texture 3D</td>
</tr>
<tr>
<td>Texture</td>
<td>Samples demonstrating how to use textures GPU kernels</td>
<td>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</td>
</tr>
</tbody>
</table>
Basic Key Concepts and Associated Samples

<table>
<thead>
<tr>
<th>Basic Key Concept</th>
<th>Description</th>
<th>Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unified Memory</td>
<td><em>Samples demonstrating how to use Unified Memory</em></td>
<td>ConjugateGradientUM, Unified Memory Streams</td>
</tr>
<tr>
<td>Unified Virtual Address Space</td>
<td><em>Samples demonstrating how to use UVA with CUDA programs</em></td>
<td>Peer-to-Peer Bandwidth Latency Test with Multi-GPUs, Simple Peer-to-Peer Transfers with Multi-GPU</td>
</tr>
<tr>
<td>Vector Addition</td>
<td><em>Samples demonstrating how to use Vector Addition with CUDA programs</em></td>
<td>Vector Addition, Vector Addition Driver API, Vector Addition with libNVRTC, simpleZeroCopy</td>
</tr>
<tr>
<td>Vertex Buffers</td>
<td><em>Samples demonstrating how to use Vertex Buffers with CUDA kernels</em></td>
<td>Simple OpenGL, Simple OpenGLES, Simple OpenGLES on Screen</td>
</tr>
<tr>
<td>Volume Processing</td>
<td><em>Samples demonstrating how to use 3D Textures for volume rendering</em></td>
<td>Simple Cubemap Texture, Simple Layered Texture</td>
</tr>
<tr>
<td>Vote Intrinsic</td>
<td><em>Samples demonstrating how to use vote intrinsics with CUDA</em></td>
<td>Simple Vote Intrinsics, Simple Vote Intrinsics with libNVRTC</td>
</tr>
</tbody>
</table>

Advanced Key Concepts

*Advanced Concepts demonstrate advanced techniques and algorithms implemented with CUDA.*

Table 3  Advanced Key Concepts and Associated Samples

<table>
<thead>
<tr>
<th>Advanced Key Concept</th>
<th>Description</th>
<th>Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>2D Textures</td>
<td><em>Texture Mapping</em></td>
<td>SLI D3D10 Texture</td>
</tr>
<tr>
<td>3D Graphics</td>
<td><em>3D Rendering</em></td>
<td>Marching Cubes Isosurfaces</td>
</tr>
<tr>
<td>3D Textures</td>
<td><em>Volume Textures</em></td>
<td>Volume Rendering with 3D Textures, Volumetric Filtering with 3D Textures and Surface Writes</td>
</tr>
<tr>
<td>CUBLAS Library</td>
<td><em>CUDA BLAS samples</em></td>
<td>ConjugateGradient, ConjugateGradientUM, Preconditioned Conjugate Gradient</td>
</tr>
<tr>
<td>CUDA Driver API</td>
<td><em>Samples that show the CUDA Driver API</em></td>
<td>CUDA Context Thread Management, Matrix Multiplication (CUDA Driver API version)</td>
</tr>
<tr>
<td>Advanced Key Concept</td>
<td>Description</td>
<td>Samples</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>CUDA Dynamic Parallelism</td>
<td>Dynamic Parallelism with GPU Kernels (SM 3.5)</td>
<td>Advanced Quicksort (CUDA Dynamic Parallelism), Bezier Line Tessellation (CUDA Dynamic Parallelism), LU Decomposition (CUDA Dynamic Parallelism), Quad Tree (CUDA Dynamic Parallelism), Simple Quicksort (CUDA Dynamic Parallelism)</td>
</tr>
<tr>
<td>CUDA Dynamically Linked Library</td>
<td>Dynamic loading of the CUDA DLL using CUDA Driver API</td>
<td>Matrix Multiplication (CUDA Driver API version with Dynamic Linking Version)</td>
</tr>
<tr>
<td>CUDA Streams and Events</td>
<td>Synchronizing Kernels with Event Timers and Streams</td>
<td>Stream Priorities</td>
</tr>
<tr>
<td>CUDA Systems Integration</td>
<td>Samples that integrate with Multi Process (OpenMP, IPC, and MPI)</td>
<td>simpleHyperQ</td>
</tr>
<tr>
<td>CUFFT Library</td>
<td>Samples that use the CUFFT accelerated library</td>
<td>CUDA FFT Ocean Simulation, FFT-Based 2D Convolution, Fluids (Direct3D Version), Fluids (OpenGL Version)</td>
</tr>
<tr>
<td>CURAND Library</td>
<td>Samples that use the CUDA random number generator</td>
<td>Monte Carlo Estimation of Pi (batch PRNG), Monte Carlo Estimation of Pi (batch QRN), Monte Carlo Estimation of Pi (inline PRNG), Monte Carlo Estimation of Pi (inline QRN), Monte Carlo Single Asian Option</td>
</tr>
<tr>
<td>CUSPARSE Library</td>
<td>Samples that use the cuSPARSE (Sparse Vector Matrix Multiply) functions</td>
<td>ConjugateGradient, ConjugateGradientUM, Preconditioned Conjugate Gradient</td>
</tr>
<tr>
<td>Computational Finance</td>
<td>Finance Algorithms</td>
<td>Binomial Option Pricing, Binomial Option Pricing with libNVRTC, Monte Carlo Estimation of Pi (batch PRNG), Monte Carlo Estimation of Pi (batch QRN), Monte Carlo Estimation of Pi (inline PRNG), Monte Carlo Estimation of Pi (inline QRN), Monte Carlo Single Asian Option</td>
</tr>
</tbody>
</table>
## Key Concepts and Associated Samples

<table>
<thead>
<tr>
<th>Advanced Key Concept</th>
<th>Description</th>
<th>Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Niederreiter Quasirandom Sequence Generator, Niederreiter Quasirandom Sequence Generator with libNVRTC, Sobol Quasirandom Number Generator</td>
</tr>
<tr>
<td><strong>Data Parallel Algorithms</strong></td>
<td><strong>Samples that show good usage of Data Parallel Algorithms</strong></td>
<td>CUDA Histogram, CUDA N-Body Simulation, CUDA N-Body Simulation on Screen, Mandelbrot, Optical Flow, Particles, Smoke Particles, VFlockingD3D10</td>
</tr>
<tr>
<td><strong>Data-Parallel Algorithms</strong></td>
<td><strong>Samples that show good usage of Data Parallel Algorithms</strong></td>
<td>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</td>
</tr>
<tr>
<td><strong>Graphics Interop</strong></td>
<td><strong>Samples that demonstrate interop between graphics APIs and CUDA</strong></td>
<td>DirectX Texture Compressor (DXTC)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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</td>
</tr>
<tr>
<td>Advanced Key Concept</td>
<td>Description</td>
<td>Samples</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>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</td>
</tr>
<tr>
<td>Linear Algebra</td>
<td>Samples demonstrating linear algebra with CUDA</td>
<td>ConjugateGradient, ConjugateGradientUM, Eigenvalues, Fast Walsh Transform, Matrix Transpose, Preconditioned Conjugate Gradient, Scalar Product</td>
</tr>
<tr>
<td>OpenGL Graphics Interop</td>
<td>Samples demonstrating how to use interoperability CUDA with OpenGL</td>
<td>Marching Cubes Isosurfaces</td>
</tr>
<tr>
<td>Performance Strategies</td>
<td>Samples demonstrating high performance with CUDA</td>
<td>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</td>
</tr>
<tr>
<td>Physically Based Simulation</td>
<td>Samples demonstrating high performance collisions and/or physical interactions</td>
<td>Marching Cubes Isosurfaces</td>
</tr>
<tr>
<td>Physically-Based Simulation</td>
<td>Samples demonstrating high performance collisions and/or physical interactions</td>
<td>CUDA N-Body Simulation, CUDA N-Body Simulation on Screen, Fluids (Direct3D Version), Fluids (OpenGL Version), Particles, Smoke Particles, VFlockingD3D10</td>
</tr>
<tr>
<td>Random Number Generator</td>
<td>Samples demonstrating how to use random number generation with CUDA</td>
<td>Monte Carlo Estimation of Pi (batch PRNG), Monte Carlo Estimation of Pi (batch QRNG), Monte Carlo Estimation of Pi (inline PRNG), Monte Carlo Estimation of Pi (inline QRNG), Monte Carlo Single Asian Option</td>
</tr>
<tr>
<td>Recursion</td>
<td>Samples demonstrating recursion on CUDA</td>
<td>Interval Computing</td>
</tr>
<tr>
<td>Advanced Key Concept</td>
<td>Description</td>
<td>Samples</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------</td>
<td>---------</td>
</tr>
<tr>
<td>Runtime Compilation</td>
<td></td>
<td>Clock libNVRTC, Matrix Multiplication with libNVRTC, Niederreiter Quasirandom Sequence Generator with libNVRTC, Simple Atomic Intrinsics with libNVRTC, Simple Templates with libNVRTC, Simple Vote Intrinsics with libNVRTC, Using Inline PTX with libNVRTC, Vector Addition with libNVRTC, simpleAssert with libNVRTC</td>
</tr>
<tr>
<td>Surface Writes</td>
<td><em>Samples demonstrating how to use Surface Writes with GPU kernels</em></td>
<td>Volumetric Filtering with 3D Textures and Surface Writes</td>
</tr>
<tr>
<td>Templates</td>
<td><em>Samples demonstrating how to use templates GPU kernels</em></td>
<td>Interval Computing</td>
</tr>
<tr>
<td>Texture</td>
<td><em>Samples demonstrating how to use textures GPU kernels</em></td>
<td>Bindless Texture</td>
</tr>
<tr>
<td>Vertex Buffers</td>
<td><em>Samples demonstrating how to use Vertex Buffers with CUDA kernels</em></td>
<td>Marching Cubes Isosurfaces</td>
</tr>
<tr>
<td>Video Compression</td>
<td><em>Samples demonstrating how to use video compression with CUDA</em></td>
<td>1D Discrete Haar Wavelet Decomposition, CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API, DCT8x8, Fast Walsh Transform</td>
</tr>
<tr>
<td>Video Intrinsics</td>
<td><em>Samples demonstrating how to use video intrinsics with CUDA</em></td>
<td>Stereo Disparity Computation (SAD SIMD Intrinsics)</td>
</tr>
</tbody>
</table>
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 4  CUDA Driver API and Associated Samples

<table>
<thead>
<tr>
<th>CUDA Driver API</th>
<th>Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>cuArrayCreate</td>
<td>Simple Texture (Driver Version)</td>
</tr>
<tr>
<td>cuArrayDestroy</td>
<td>Simple Texture (Driver Version)</td>
</tr>
<tr>
<td>cuCtxCreate</td>
<td>CUDA Context Thread Management, CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuCtxCreate</td>
<td>CUDA Context Thread Management, CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuCtxDestroy</td>
<td>Simple Texture (Driver Version)</td>
</tr>
<tr>
<td>cuCtxDetach</td>
<td>Simple Texture (Driver Version)</td>
</tr>
<tr>
<td>cuCtxPopCurrent</td>
<td>CUDA Context Thread Management, CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuCtxPushCurrent</td>
<td>CUDA Context Thread Management, CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuCtxSynchronize</td>
<td>Simple Texture (Driver Version)</td>
</tr>
<tr>
<td>cuD3D9CtxCreate</td>
<td>CUDA Video Decoder D3D9 API</td>
</tr>
<tr>
<td>cuD3D9GetDevice</td>
<td>CUDA Video Decoder D3D9 API</td>
</tr>
<tr>
<td>cuD3D9MapResources</td>
<td>CUDA Video Decoder D3D9 API</td>
</tr>
<tr>
<td>CUDA Driver API</td>
<td>Samples</td>
</tr>
<tr>
<td>------------------------------</td>
<td>------------------------------------------------------</td>
</tr>
<tr>
<td>cuD3D9RegisterResource</td>
<td>CUDA Video Decoder D3D9 API</td>
</tr>
<tr>
<td>cuD3D9ResourceGetMappedPitch</td>
<td>CUDA Video Decoder D3D9 API</td>
</tr>
<tr>
<td>cuD3D9ResourceGetMappedPointer</td>
<td>CUDA Video Decoder D3D9 API</td>
</tr>
<tr>
<td>cuD3D9ResourceSetMapFlags</td>
<td>CUDA Video Decoder D3D9 API</td>
</tr>
<tr>
<td>cuD3D9UnmapResources</td>
<td>CUDA Video Decoder D3D9 API</td>
</tr>
<tr>
<td>cuD3D9UnregisterResource</td>
<td>CUDA Video Decoder D3D9 API</td>
</tr>
<tr>
<td>cuDeviceComputeCapability</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API, Device Query Driver API</td>
</tr>
<tr>
<td>cuDeviceGet</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuDeviceGetAttribute</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API, Device Query Driver API</td>
</tr>
<tr>
<td>cuDeviceGetCount</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API, Device Query Driver API</td>
</tr>
<tr>
<td>cuDeviceGetName</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuDeviceTotalMem</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API, Device Query Driver API</td>
</tr>
<tr>
<td>cuDriverGetVersion</td>
<td>Device Query Driver API</td>
</tr>
<tr>
<td>cuGLCtxCreate</td>
<td>CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuGLGetDevice</td>
<td>CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuGLMapResources</td>
<td>CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuGLRegisterResource</td>
<td>CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuGLResourceGetMappedPitch</td>
<td>CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuGLResourceGetMappedPointer</td>
<td>CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuGLResourceSetMapFlags</td>
<td>CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuGLUnmapResources</td>
<td>CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuGLUnregisterResource</td>
<td>CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuInit</td>
<td>Device Query Driver API</td>
</tr>
<tr>
<td>CUDA Driver API</td>
<td>Samples</td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>cuLaunchGridAsync</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuLaunchKernel</td>
<td>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</td>
</tr>
<tr>
<td>cuMemAlloc</td>
<td>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</td>
</tr>
<tr>
<td>cuMemAllocHost</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuMemFree</td>
<td>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</td>
</tr>
<tr>
<td>cuMemFreeHost</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuMemcpy2D</td>
<td>Simple Texture (Driver Version)</td>
</tr>
<tr>
<td>cuMemcpyDtoH</td>
<td>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</td>
</tr>
<tr>
<td>cuMemcpyDtoHASync</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuMemcpyHtoD</td>
<td>Matrix Multiplication (CUDA Driver API Version), Matrix Multiplication (CUDA Driver API version with Dynamic Linking Version), Vector Addition Driver API</td>
</tr>
<tr>
<td>cuMemsetD8</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuModuleGetFunction</td>
<td>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</td>
</tr>
<tr>
<td>cuModuleGetGlobal</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
</tbody>
</table>
### CUDA Driver API

<table>
<thead>
<tr>
<th>Function</th>
<th>Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>cuModuleGetTexRef</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API, Simple Texture (Driver Version)</td>
</tr>
<tr>
<td>cuModuleLoad</td>
<td>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</td>
</tr>
<tr>
<td>cuModuleLoadDataEx</td>
<td>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</td>
</tr>
<tr>
<td>cuModuleUnload</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuParamSetSize</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuParamSetTexRef</td>
<td>Simple Texture (Driver Version)</td>
</tr>
<tr>
<td>cuParamSeti</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuParamSetv</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuStreamCreate</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuTexRefSetAddressMode</td>
<td>Simple Texture (Driver Version)</td>
</tr>
<tr>
<td>cuTexRefSetArray</td>
<td>Simple Texture (Driver Version)</td>
</tr>
<tr>
<td>cuTexRefSetFilterMode</td>
<td>Simple Texture (Driver Version)</td>
</tr>
<tr>
<td>cuTexRefSetFlags</td>
<td>Simple Texture (Driver Version)</td>
</tr>
<tr>
<td>cuTexRefSetFormat</td>
<td>Simple Texture (Driver Version)</td>
</tr>
<tr>
<td>cuvidCreateDecoder</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuvidCtxLockCreate</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuvidCtxLockDestroy</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuvidDecodePicture</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuvidDestroyDecoder</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuvidMapVideoFrame</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
</tbody>
</table>
## CUDA Runtime API Samples

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

<table>
<thead>
<tr>
<th>CUDA Runtime API</th>
<th>Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>cuArrayCreate</td>
<td>Simple Texture (Driver Version)</td>
</tr>
<tr>
<td>cuArrayDestroy</td>
<td>Simple Texture (Driver Version)</td>
</tr>
<tr>
<td>cuCtxCreate</td>
<td>CUDA Context Thread Management, CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuCtxDestroy</td>
<td>CUDA Context Thread Management, CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuCtxDetach</td>
<td>Simple Texture (Driver Version)</td>
</tr>
<tr>
<td>cuCtxPopCurrent</td>
<td>CUDA Context Thread Management, CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuCtxPushCurrent</td>
<td>CUDA Context Thread Management, CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuCtxSynchronize</td>
<td>Simple Texture (Driver Version)</td>
</tr>
<tr>
<td>cuD3D9CtxCreate</td>
<td>CUDA Video Decoder D3D9 API</td>
</tr>
<tr>
<td>cuD3D9GetDevice</td>
<td>CUDA Video Decoder D3D9 API</td>
</tr>
<tr>
<td>cuD3D9MapResources</td>
<td>CUDA Video Decoder D3D9 API</td>
</tr>
<tr>
<td>cuD3D9RegisterResource</td>
<td>CUDA Video Decoder D3D9 API</td>
</tr>
<tr>
<td>cuD3D9ResourceGetMappedPitch</td>
<td>CUDA Video Decoder D3D9 API</td>
</tr>
<tr>
<td>cuD3D9ResourceGetMappedPointer</td>
<td>CUDA Video Decoder D3D9 API</td>
</tr>
<tr>
<td>cuD3D9ResourceSetMapFlags</td>
<td>CUDA Video Decoder D3D9 API</td>
</tr>
<tr>
<td>cuD3D9UnmapResources</td>
<td>CUDA Video Decoder D3D9 API</td>
</tr>
<tr>
<td>cuD3D9UnregisterResource</td>
<td>CUDA Video Decoder D3D9 API</td>
</tr>
<tr>
<td>CUDA Runtime API</td>
<td>Samples</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>cuDeviceComputeCapability</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API, Device Query Driver API</td>
</tr>
<tr>
<td>cuDeviceGet</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuDeviceGetAttribute</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API, Device Query Driver API</td>
</tr>
<tr>
<td>cuDeviceGetCount</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API, Device Query Driver API</td>
</tr>
<tr>
<td>cuDeviceGetName</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuDeviceTotalMem</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API, Device Query Driver API</td>
</tr>
<tr>
<td>cuDriverGetVersion</td>
<td>Device Query Driver API</td>
</tr>
<tr>
<td>cuGLCtxCreate</td>
<td>CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuGLGetDevice</td>
<td>CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuGLMapResources</td>
<td>CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuGLRegisterResource</td>
<td>CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuGLResourceGetMappedPitch</td>
<td>CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuGLResourceGetMappedPointer</td>
<td>CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuGLResourceSetMapFlags</td>
<td>CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuGLUnmapResources</td>
<td>CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuGLUnregisterResource</td>
<td>CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuInit</td>
<td>Device Query Driver API</td>
</tr>
<tr>
<td>cuLaunchGridAsync</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuLaunchKernel</td>
<td>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</td>
</tr>
<tr>
<td>cuMemAlloc</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API, Matrix Multiplication (CUDA Driver API version with Dynamic Linking Version), Simple Texture (Driver Version), Simple Vote</td>
</tr>
<tr>
<td>CUDA Runtime API</td>
<td>Samples</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>Intrinsics with libNVRTC, Vector Addition Driver API, Vector Addition with libNVRTC</td>
</tr>
<tr>
<td>cuMemAllocHost</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuMemFree</td>
<td>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), Simple Vote Intrinsics with libNVRTC, Vector Addition Driver API, Vector Addition with libNVRTC</td>
</tr>
<tr>
<td>cuMemFreeHost</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuMemcpy2D</td>
<td>Simple Texture (Driver Version)</td>
</tr>
<tr>
<td>cuMemcpyDtoH</td>
<td>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, Vector Addition with libNVRTC</td>
</tr>
<tr>
<td>cuMemcpyDtoHAsync</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuMemcpyHtoD</td>
<td>Matrix Multiplication (CUDA Driver API Version), Matrix Multiplication (CUDA Driver API version with Dynamic Linking Version), Simple Vote Intrinsics with libNVRTC, Vector Addition Driver API, Vector Addition with libNVRTC</td>
</tr>
<tr>
<td>cuMemsetD8</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuModuleGetFunction</td>
<td>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</td>
</tr>
<tr>
<td>cuModuleGetGlobal</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuModuleGetTexRef</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API, Simple Texture (Driver Version)</td>
</tr>
<tr>
<td>cuModuleLoad</td>
<td>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</td>
</tr>
<tr>
<td>CUDA Runtime API</td>
<td>Samples</td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>cuModuleLoadDataEx</td>
<td>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</td>
</tr>
<tr>
<td>cuModuleUnload</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuParamSetSize</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuParamSetTexRef</td>
<td>Simple Texture (Driver Version)</td>
</tr>
<tr>
<td>cuParamSeti</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuParamSetv</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuStreamCreate</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuTexRefSetAddressMode</td>
<td>Simple Texture (Driver Version)</td>
</tr>
<tr>
<td>cuTexRefSetArray</td>
<td>Simple Texture (Driver Version)</td>
</tr>
<tr>
<td>cuTexRefSetFilterMode</td>
<td>Simple Texture (Driver Version)</td>
</tr>
<tr>
<td>cuTexRefSetFlags</td>
<td>Simple Texture (Driver Version)</td>
</tr>
<tr>
<td>cuTexRefSetFormat</td>
<td>Simple Texture (Driver Version)</td>
</tr>
<tr>
<td>cublasCreate</td>
<td>Matrix Multiplication (CUBLAS), simpleDevLibCUBLAS GPU Device API Library Functions (CUDA Dynamic Parallelism)</td>
</tr>
<tr>
<td>cublasSetVector</td>
<td>simpleDevLibCUBLAS GPU Device API Library Functions (CUDA Dynamic Parallelism)</td>
</tr>
<tr>
<td>cublasSgemm</td>
<td>Matrix Multiplication (CUBLAS), simpleDevLibCUBLAS GPU Device API Library Functions (CUDA Dynamic Parallelism)</td>
</tr>
<tr>
<td>cudaBindSurfaceToArray</td>
<td>Simple Surface Write</td>
</tr>
<tr>
<td>cudaBindTexture2D</td>
<td>Pitch Linear Texture</td>
</tr>
<tr>
<td>cudaBindTextureToArray</td>
<td>Pitch Linear Texture, Simple Cubemap Texture, Simple Layered Texture, Simple Surface Write, Simple Texture</td>
</tr>
<tr>
<td>cudaCreateChannelDesc</td>
<td>Pitch Linear Texture, Simple Cubemap Texture, Simple Layered Texture, Simple Surface Write, Simple Texture</td>
</tr>
<tr>
<td>cudaD3D10GetDevice</td>
<td>SLI D3D10 Texture, Simple D3D10 Texture, Simple Direct3D10 (Vertex Array), Simple Direct3D10 Render Target</td>
</tr>
<tr>
<td>CUDA Runtime API</td>
<td>Samples</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>cudaD3D10SetDirect3DDevice</td>
<td>SLI D3D10 Texture, Simple D3D10 Texture, Simple Direct3D10 (Vertex Array), Simple Direct3D10 Render Target</td>
</tr>
<tr>
<td>cudaD3D10SetGLDevice</td>
<td>VFlockingD3D10</td>
</tr>
<tr>
<td>cudaD3D11GetDevice</td>
<td>Simple D3D11 Texture</td>
</tr>
<tr>
<td>cudaD3D11SetDirect3DDevice</td>
<td>Simple D3D11 Texture</td>
</tr>
<tr>
<td>cudaD3D9GetDevice</td>
<td>Simple D3D9 Texture, Simple Direct3D9 (Vertex Arrays)</td>
</tr>
<tr>
<td>cudaD3D9SetDirect3DDevice</td>
<td>Simple D3D9 Texture, Simple Direct3D9 (Vertex Arrays)</td>
</tr>
<tr>
<td>cudaD3D9SetGLDevice</td>
<td>Fluids (Direct3D Version)</td>
</tr>
<tr>
<td>cudaDeviceCanAccessPeer</td>
<td>Peer-to-Peer Bandwidth Latency Test with Multi-GPUs, Simple Peer-to-Peer Transfers with Multi-GPU</td>
</tr>
<tr>
<td>cudaDeviceDisablePeerAccess</td>
<td>Peer-to-Peer Bandwidth Latency Test with Multi-GPUs, Simple Peer-to-Peer Transfers with Multi-GPU</td>
</tr>
<tr>
<td>cudaDeviceEnablePeerAccess</td>
<td>Peer-to-Peer Bandwidth Latency Test with Multi-GPUs, Simple Peer-to-Peer Transfers with Multi-GPU</td>
</tr>
<tr>
<td>cudaDeviceSynchronize</td>
<td>Bandwidth Test, Template, Template using CUDA Runtime</td>
</tr>
<tr>
<td>cudaDriverGetVersion</td>
<td>Device Query</td>
</tr>
<tr>
<td>cudaEventCreate</td>
<td>Bandwidth Test, Matrix Multiplication (CUBLAS), Matrix Multiplication (CUDA Runtime API Version), Matrix Multiplication with libNVRTC, Simple Multi Copy and Compute, Simple Multi-GPU, Vector Addition, asyncAPI, simpleStreams, simpleZeroCopy</td>
</tr>
<tr>
<td>cudaEventCreateWithFlags</td>
<td>Peer-to-Peer Bandwidth Latency Test with Multi-GPUs, Simple Peer-to-Peer Transfers with Multi-GPU</td>
</tr>
<tr>
<td>cudaEventDestroy</td>
<td>Bandwidth Test, Matrix Multiplication (CUBLAS), Matrix Multiplication (CUDA Runtime API Version), Matrix Multiplication with libNVRTC, Simple Multi Copy and Compute, Simple Multi-GPU, Vector Addition, asyncAPI, simpleStreams, simpleZeroCopy</td>
</tr>
<tr>
<td>cudaEventElapsedTime</td>
<td>Bandwidth Test, Matrix Multiplication (CUBLAS), Matrix Multiplication (CUDA Runtime API Version), Matrix Multiplication with libNVRTC, Peer-to-Peer Bandwidth Latency Test with Multi-GPUs, Simple Multi Copy and Compute, Simple Multi-GPU, Simple Peer-to-Peer Transfers</td>
</tr>
<tr>
<td><strong>CUDA Runtime API</strong></td>
<td><strong>Samples</strong></td>
</tr>
<tr>
<td>----------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>cudaEventQuery</td>
<td>Matrix Multiplication (CUBLAS), Matrix Multiplication (CUDA Runtime API Version), Matrix Multiplication with libNVRTC, Simple Multi Copy and Compute, Simple Multi-GPU, Vector Addition, asyncAPI, simpleStreams, simpleZeroCopy</td>
</tr>
<tr>
<td>cudaEventRecord</td>
<td>Bandwidth Test, Matrix Multiplication (CUBLAS), Matrix Multiplication (CUDA Runtime API Version), Matrix Multiplication with libNVRTC, Simple Multi Copy and Compute, Simple Multi-GPU, Vector Addition, asyncAPI, simpleStreams, simpleZeroCopy</td>
</tr>
<tr>
<td>cudaEventSynchronize</td>
<td>Matrix Multiplication (CUDA Runtime API Version), Matrix Multiplication with libNVRTC, Vector Addition</td>
</tr>
<tr>
<td>cudaFree</td>
<td>Bandwidth Test, C++ Integration, Clock, Clock libNVRTC, Matrix Multiplication (CUBLAS), Matrix Multiplication (CUDA Runtime API Version), Matrix Multiplication with libNVRTC, Pitch Linear Texture, Simple Atomic Intrinsics, Simple Atomic Intrinsics with libNVRTC, Simple Cubemap Texture, Simple Layered Texture, Simple Surface Write, Simple Texture, Simple Vote Intrinsics, Template, Template using CUDA Runtime, Using Inline PTX, Using Inline PTX with libNVRTC, Vector Addition, cudaOpenMP, simpleAssert, simpleAssert with libNVRTC, simpleDevLibCUBLAS GPU Device API Library Functions (CUDA Dynamic Parallelism), simpleMPI</td>
</tr>
<tr>
<td>cudaFreeArray</td>
<td>Pitch Linear Texture, Simple Cubemap Texture, Simple Layered Texture, Simple Surface Write, Simple Texture</td>
</tr>
<tr>
<td>cudaFreeHost</td>
<td>Bandwidth Test, Simple Atomic Intrinsics, Simple Atomic Intrinsics with libNVRTC, Simple Vote Intrinsics, Template using CUDA Runtime, Using Inline PTX, Using Inline PTX with libNVRTC, simpleAssert, simpleAssert with libNVRTC, simpleIPC, simpleZeroCopy</td>
</tr>
<tr>
<td>cudaFuncGetAttributes</td>
<td>cppOverload</td>
</tr>
<tr>
<td>cudaFuncSetCacheConfig</td>
<td>cppOverload</td>
</tr>
<tr>
<td>cudaGLSetGLDevice</td>
<td>Bicubic B-spline Interpolation, Bilateral Filter, Bindless Texture, Box Filter, CUDA FFT Ocean Simulation, CUDA N-Body Simulation, CUDA N-Body Simulation on Screen, CUDA and OpenGL Interop of Images, Fluids (OpenGL Version),...</td>
</tr>
<tr>
<td>CUDA Runtime API</td>
<td>Samples</td>
</tr>
<tr>
<td>----------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>cudaGetDeviceProperties</td>
<td>Device Query</td>
</tr>
<tr>
<td>cudaGraphicsD3D10RegisterResource</td>
<td>SLI D3D10 Texture, Simple D3D10 Texture, Simple Direct3D10 (Vertex Array), Simple Direct3D10 Render Target</td>
</tr>
<tr>
<td>cudaGraphicsD3D11RegisterResource</td>
<td>Simple D3D11 Texture</td>
</tr>
<tr>
<td>cudaGraphicsD3D9RegisterResource</td>
<td>Simple D3D9 Texture, Simple Direct3D9 (Vertex Arrays)</td>
</tr>
<tr>
<td>CUDA Runtime API</td>
<td>Samples</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CUDA API and Associated Samples</td>
<td></td>
</tr>
<tr>
<td>cudaGraphicsResourceSetMapFlags</td>
<td>SLI D3D10 Texture, Simple D3D10 Texture, Simple D3D11 Texture, Simple D3D9 Texture, Simple Direct3D10 (Vertex Array), Simple Direct3D10 Render Target</td>
</tr>
<tr>
<td>cudaGraphicsSubResourceGetMappedArray</td>
<td>SLI D3D10 Texture, Simple D3D10 Texture, Simple D3D11 Texture, Simple D3D9 Texture, Simple Direct3D10 (Vertex Array), Simple Direct3D10 Render Target</td>
</tr>
</tbody>
</table>
## CUDA Runtime API and Associated Samples

<table>
<thead>
<tr>
<th>CUDA Runtime API</th>
<th>Samples</th>
</tr>
</thead>
<tbody>
<tr>
<td>cudaHostAlloc</td>
<td>Bandwidth Test, simpleZeroCopy</td>
</tr>
<tr>
<td>cudaHostGetDevicePointer</td>
<td>simpleZeroCopy</td>
</tr>
<tr>
<td>cudaHostRegister</td>
<td>simpleZeroCopy</td>
</tr>
<tr>
<td>cudaHostUnregister</td>
<td>simpleZeroCopy</td>
</tr>
<tr>
<td>cudaMemcpyCloseMemHandle</td>
<td>simpleIPC</td>
</tr>
<tr>
<td>cudaMemcpyGetEventHandle</td>
<td>simpleIPC</td>
</tr>
<tr>
<td>cudaMemcpyOpenMemHandle</td>
<td>simpleIPC</td>
</tr>
<tr>
<td>cudaMemcpy</td>
<td>Simple Vote Intrinsics, simpleMPI</td>
</tr>
<tr>
<td>cudaMemcpy3DArray</td>
<td>C++ Integration, Clock, Clock libNVRTC, Matrix Multiplication (CUBLAS), Matrix Multiplication (CUDA Runtime API Version), Matrix Multiplication with libNVRTC, Pitch Linear Texture, Simple Atomic Intrinsics, Simple Atomic Intrinsics with libNVRTC, Simple Cubemap Texture, Simple Layered Texture, Simple Surface Write, Simple Texture, Template, Template using CUDA Runtime, Using Inline PTX, Using Inline PTX with libNVRTC, Vector Addition, cudaMemcpy, simpleAssert, simpleAssert with libNVRTC, simpleDevLibCUBLAS GPU Device API Library Functions (CUDA Dynamic Parallelism)</td>
</tr>
<tr>
<td>cudaMemcpyArray</td>
<td>Simple Cubemap Texture, Simple Layered Texture</td>
</tr>
<tr>
<td>cudaMemcpyHost</td>
<td>Pitch Linear Texture, Simple Surface Write, Simple Texture</td>
</tr>
<tr>
<td>cudaMemcpyManaged</td>
<td>Bandwidth Test, Template using CUDA Runtime, Using Inline PTX, Using Inline PTX with libNVRTC, simpleAssert, simpleAssert with libNVRTC</td>
</tr>
<tr>
<td>cudaMemcpyManaged</td>
<td>Unified Memory Streams</td>
</tr>
<tr>
<td>CUDA Runtime API</td>
<td>Samples</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------------------------------------------------------------------------</td>
</tr>
<tr>
<td>cudaMallocPitch</td>
<td>Pitch Linear Texture</td>
</tr>
<tr>
<td>cudaMemcpy</td>
<td>Bandwidth Test, C++ Integration, Clock, Clock libNVRTC, Matrix Multiplication (CUBLAS), Matrix Multiplication (CUDA Runtime API Version), Matrix Multiplication with libNVRTC, Peer-to-Peer Bandwidth Latency Test with Multi-GPUs, Simple Atomic Intrinsics, Simple Atomic Intrinsics with libNVRTC, Simple Cubemap Texture, Simple Layered Texture, Simple Peer-to-Peer Transfers with Multi-GPU, Simple Surface Write, Simple Texture, Simple Vote Intrinsics, Template, Template using CUDA Runtime, Using Inline PTX, Using Inline PTX with libNVRTC, Vector Addition, cudaOpenMP, simpleAssert, simpleAssert with libNVRTC, simpleDevLibCUBLAS GPU Device API Library Functions (CUDA Dynamic Parallelism), simpleIPC, simpleMPI</td>
</tr>
<tr>
<td>cudaMemcpy2D</td>
<td>Pitch Linear Texture</td>
</tr>
<tr>
<td>cudaMemcpy2D</td>
<td>SLI D3D10 Texture, Simple D3D10 Texture, Simple D3D11 Texture, Simple D3D9 Texture, Simple Direct3D10 (Vertex Array), Simple Direct3D10 Render Target</td>
</tr>
<tr>
<td>cudaMemcpy2DToArray</td>
<td>Simple Cubemap Texture, Simple D3D9 Texture, Simple Layered Texture</td>
</tr>
<tr>
<td>cudaMemcpy3D</td>
<td>Simple Cubemap Texture, Simple D3D9 Texture, Simple Layered Texture</td>
</tr>
<tr>
<td>cudaMemcpyAsync</td>
<td>Bandwidth Test, Simple CUDA Callbacks, Simple Multi Copy and Compute, Simple Multi-GPU, asyncAPI, simpleStreams</td>
</tr>
<tr>
<td>cudaMemcpyToArray</td>
<td>Pitch Linear Texture, Simple Texture</td>
</tr>
<tr>
<td>cudaMemcpy2D</td>
<td>Pitch Linear Texture</td>
</tr>
<tr>
<td>cudaRuntimeGetVersion</td>
<td>Device Query</td>
</tr>
<tr>
<td>cudaSetDevice</td>
<td>Bandwidth Test, Device Query</td>
</tr>
<tr>
<td>cudaStreamAddCallback</td>
<td>Simple CUDA Callbacks</td>
</tr>
<tr>
<td>cudaStreamAttachManagedMem</td>
<td>Unified Memory Streams</td>
</tr>
<tr>
<td>cudaStreamCreate</td>
<td>Simple CUDA Callbacks</td>
</tr>
<tr>
<td>cudaStreamDestroy</td>
<td>Simple CUDA Callbacks</td>
</tr>
<tr>
<td>cudaUnbindTexture</td>
<td>Pitch Linear Texture</td>
</tr>
<tr>
<td>cufftDestroy</td>
<td>CUDA FFT Ocean Simulation, FFT-Based 2D Convolution</td>
</tr>
<tr>
<td>CUDA Runtime API</td>
<td>Samples</td>
</tr>
<tr>
<td>-----------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>cufftExecC2R</td>
<td>CUDA FFT Ocean Simulation, FFT-Based 2D Convolution</td>
</tr>
<tr>
<td>cufftExecR2C</td>
<td>CUDA FFT Ocean Simulation, FFT-Based 2D Convolution</td>
</tr>
<tr>
<td>cufftPlan2d</td>
<td>CUDA FFT Ocean Simulation, FFT-Based 2D Convolution</td>
</tr>
<tr>
<td>cuvidCreateDecoder</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuvidCtxLockCreate</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuvidCtxLockDestroy</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuvidDecodePicture</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuvidDestroyDecoder</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuvidMapVideoFrame</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>cuvidUnmapVideoFrame</td>
<td>CUDA Video Decoder D3D9 API, CUDA Video Decoder GL API</td>
</tr>
<tr>
<td>nppGetGpuComputeCapability</td>
<td>JPEG encode/decode and resize with NPP</td>
</tr>
<tr>
<td>nppiDCTFree</td>
<td>JPEG encode/decode and resize with NPP</td>
</tr>
<tr>
<td>nppiDCTInitAlloc</td>
<td>JPEG encode/decode and resize with NPP</td>
</tr>
<tr>
<td>nppiDCTQuantInv8x8LS_JPEG_16s8u_C1R</td>
<td>JPEG encode/decode and resize with NPP</td>
</tr>
<tr>
<td>nppiDecodeHuffmanScanHost_JPEG_8u16s_P3R</td>
<td>JPEG encode/decode and resize with NPP</td>
</tr>
<tr>
<td>nppiEncodeHuffmanGetSize</td>
<td>JPEG encode/decode and resize with NPP</td>
</tr>
<tr>
<td>nppiResizeSqrPixel_8u_C1R</td>
<td>JPEG encode/decode and resize with NPP</td>
</tr>
</tbody>
</table>
Chapter 7.
FREQUENTLY ASKED QUESTIONS

Answers to frequently asked questions about CUDA can be found at http://developer.nvidia.com/cuda-faq and in the CUDA Toolkit Release Notes.
Notice

ALL NVIDIA DESIGN SPECIFICATIONS, REFERENCE BOARDS, FILES, DRAWINGS, DIAGNOSTICS, LISTS, AND OTHER DOCUMENTS (TOGETHER AND SEPARATELY, "MATERIALS") ARE BEING PROVIDED "AS IS." NVIDIA MAKES NO WARRANTIES, EXPRESSED, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE MATERIALS, AND EXPRESSLY DISCLAIMS ALL IMPLIED WARRANTIES OF NONINFRINGEMENT, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE.

Information furnished is believed to be accurate and reliable. However, NVIDIA Corporation assumes no responsibility for the consequences of use of such information or for any infringement of patents or other rights of third parties that may result from its use. No license is granted by implication of otherwise under any patent rights of NVIDIA Corporation. Specifications mentioned in this publication are subject to change without notice. This publication supersedes and replaces all other information previously supplied. NVIDIA Corporation products are not authorized as critical components in life support devices or systems without express written approval of NVIDIA Corporation.

Trademarks

NVIDIA and the NVIDIA logo are trademarks or registered trademarks of NVIDIA Corporation in the U.S. and other countries. Other company and product names may be trademarks of the respective companies with which they are associated.

Copyright

© 2007-2015 NVIDIA Corporation. All rights reserved.