CUDA SAMPLES

v5.0 | October 2012

TABLE OF CONTENTS

PREFACE

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

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

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

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

Finance
: Samples that demonstrate parallel algorithms for financial computing.

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

Advanced
: Samples that illustrate advanced algorithms implemented with CUDA.

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

Chapter 1. SIMPLE

1.1 asyncAPI

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

Minimum Required GPU	GeForce 8
Source	asyncAPI

1.2 C++ Integration

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

Minimum Required GPU	GeForce 8
Source	cppIntegration

1.3 Clock

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

Minimum Required GPU	GeForce 8
Source	clock

1.4 cudaOpenMP

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

Minimum Required GPU	GeForce 8
Source	cudaOpenMP

1.5 Matrix Multiplication (CUBLAS)

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

Minimum Required GPU	GeForce 8
Source	matrixMulCUBLAS

1.6 Matrix Multiplication (CUDA Driver API Version)

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

Minimum Required GPU	GeForce 8
Source	matrixMulDrv

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

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

Minimum Required GPU	GeForce 8
Source	matrixMulDynlinkJIT

1.8 Matrix Multiplication (CUDA Runtime API Version)

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

Minimum Required GPU	GeForce 8
Source	matrixMul

1.9 Pitch Linear Texture

Use of Pitch Linear Textures

Minimum Required GPU	GeForce 8
Source	simplePitchLinearTexture

1.10 Simple Atomic Intrinsics

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

Minimum Required GPU	GeForce 8
Source	simpleAtomicIntrinsics

1.11 Simple Cubemap Texture

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

Minimum Required GPU	GeForce GTX 400
Source	simpleCubemapTexture

1.12 Simple CUDA Callbacks

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

Minimum Required GPU	GeForce 8

Source	simpleCallback

1.13 Simple Layered Texture

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

Minimum Required GPU	GeForce GTX 400
Source	simpleLayeredTexture

1.14 Simple Multi Copy and Compute

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

Minimum Required GPU	GeForce 8
Source	simpleMultiCopy

1.15 Simple Multi-GPU

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

Minimum Required GPU	GeForce 8
Source	simpleMultiGPU

1.16 Simple Peer-to-Peer Transfers with Multi-GPU

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

Minimum Required GPU	GeForce GTX 400
Source	simpleP2P

1.17 Simple Print (CUDA Dynamic Parallelism)

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

Minimum Required GPU	KEPLER SM 3.5
Source	cdpSimplePrint

1.18 Simple Quicksort (CUDA Dynamic Parallelism)

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

Minimum Required GPU	KEPLER SM 3.5
Source	cdpSimpleQuicksort

1.19 Simple Static GPU Device Library

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

Minimum Required GPU	GeForce GTX 400
Source	simpleSeparateCompilation

1.20 Simple Surface Write

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

Minimum Required GPU	GeForce GTX 400
Source	simpleSurfaceWrite

1.21 Simple Templates

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

Minimum Required GPU	GeForce 8
Source	simpleTemplates

1.22 Simple Texture

Simple example that demonstrates use of Textures in CUDA.

Minimum Required GPU	GeForce 8

Source	simpleTexture

1.23 Simple Texture (Driver Version)

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

Minimum Required GPU	GeForce 8
Source	simpleTextureDrv

1.24 Simple Vote Intrinsics

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

Minimum Required GPU	GeForce 8
Source	simpleVoteIntrinsics

1.25 simpleAssert

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

Minimum Required GPU	GeForce GTX 400
Source	simpleAssert

1.26 simpleIPC

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

Minimum Required GPU	GeForce GTX 400
Source	simpleIPC

1.27 simpleMPI

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

Minimum Required GPU	GeForce 8
Source	simpleMPI

1.28 simplePrintf

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

Minimum Required GPU	GeForce 8
Source	simplePrintf

1.29 simpleStreams

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

Minimum Required GPU	GeForce 8
Source	simpleStreams

1.30 simpleZeroCopy

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

Minimum Required GPU	GeForce GTX 200
White Paper	CUDA2.2PinnedMemoryAPIs.pdf
Source	simpleZeroCopy

1.31 Template

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

Minimum Required GPU	GeForce 8
Source	template

1.32 Template using CUDA Runtime

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

Minimum Required GPU	GeForce 8

Source	template_runtime

1.33 Using Inline PTX

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

Minimum Required GPU	GeForce 8
Source	inlinePTX

1.34 Vector Addition

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

Minimum Required GPU	GeForce 8
Source	vectorAdd

1.35 Vector Addition Driver API

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

Minimum Required GPU	GeForce 8
Source	vectorAddDrv

Chapter 2. UTILITIES

2.1 Bandwidth Test

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

Minimum Required GPU	GeForce 8
Source	bandwidthTest

2.2 Device Query

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

Minimum Required GPU	GeForce 8
Source	deviceQuery

2.3 Device Query Driver API

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

Minimum Required GPU	GeForce 8
Source	deviceQueryDrv

Chapter 3. GRAPHICS

3.1 Bindless Texture

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

Minimum Required GPU	GeForce GTX 400
Source	bindlessTexture

3.2 Mandelbrot

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

Minimum Required GPU	GeForce 8
Source	Mandelbrot

3.3 Marching Cubes Isosurfaces

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

Minimum Required GPU	GeForce 8
Source	marchingCubes

3.4 Simple D3D10 Texture

Simple program which demonstrates how to interoperate CUDA with Direct3D10 Texture. The program creates a number of D3D10 Textures (2D, 3D, and CubeMap) which are generated from CUDA kernels. Direct3D then renders the results on the screen. A Direct3D10 Capable device is required.

Minimum Required GPU	GeForce 8
Source	simpleD3D10Texture

3.5 Simple D3D11 Texture

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

Minimum Required GPU	GeForce 8
Source	simpleD3D11Texture

3.6 Simple D3D9 Texture

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

Minimum Required GPU	GeForce 8
Source	simpleD3D9Texture

3.7 Simple Direct3D10 (Vertex Array)

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

Minimum Required GPU	GeForce 8
Source	simpleD3D10

3.8 Simple Direct3D10 Render Target

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

Minimum Required GPU	GeForce 8
Source	simpleD3D10RenderTarget

3.9 Simple Direct3D9 (Vertex Arrays)

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

Minimum Required GPU	GeForce 8
Source	simpleD3D9

3.10 Simple OpenGL

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

Minimum Required GPU	GeForce 8
Source	simpleGL

3.11 Simple Texture 3D

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

Minimum Required GPU	GeForce 8
Source	simpleTexture3D

3.12 SLI D3D10 Texture

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

Minimum Required GPU	GeForce 8
Source	SLID3D10Texture

3.13 Volume Rendering with 3D Textures

This sample demonstrates basic volume rendering using 3D Textures.

Minimum Required GPU	GeForce 8
Source	volumeRender

3.14 Volumetric Filtering with 3D Textures and Surface Writes

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

Minimum Required GPU	GeForce GTX 400
Source	volumeFiltering

Chapter 4. IMAGING

4.1 1D Discrete Haar Wavelet Decomposition

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

Minimum Required GPU	GeForce 8
Source	dwtHaar1D

4.2 Bicubic Texture Filtering

This sample demonstrates how to efficiently implement bicubic Texture filtering in CUDA.

Minimum Required GPU	GeForce 8
Source	bicubicTexture

4.3 Bilateral Filter

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

Minimum Required GPU	GeForce 8
Source	bilateralFilter

4.4 Box Filter

Fast image box filter using CUDA with OpenGL rendering.

Minimum Required GPU	GeForce 8
Source	boxFilter

4.5 CUDA Histogram

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

Minimum Required GPU	GeForce 8
White Paper	histogram.pdf
Source	histogram

4.6 CUDA Separable Convolution

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

Minimum Required GPU	GeForce 8
White Paper	convolutionSeparable.pdf
Source	convolutionSeparable

4.7 CUDA Video Decoder D3D9 API

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

Minimum Required GPU	GeForce 8
White Paper	nvcuvid.pdf
Source	cudaDecodeD3D9

4.8 CUDA Video Decoder GL API

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

Minimum Required GPU	GeForce 8
White Paper	nvcuvid.pdf
Source	cudaDecodeGL

4.9 CUDA Video Encode (C Library) API

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

Minimum Required GPU	GeForce 8
White Paper	nvcuvenc.pdf
Source	cudaEncode

4.10 DCT8x8

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

Minimum Required GPU	GeForce 8
White Paper	dct8x8.pdf
Source	dct8x8

4.11 DirectX Texture Compressor (DXTC)

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

Minimum Required GPU	GeForce 8
White Paper	cuda_dxtc.pdf
Source	dxtc

4.12 FFT-Based 2D Convolution

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

Minimum Required GPU	GeForce 8

White Paper	convolutionFFT2D.pdf
Source	convolutionFFT2D

4.13 Image denoising

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

Minimum Required GPU	GeForce 8
White Paper	imageDenoising.pdf
Source	imageDenoising

4.14 Optical Flow

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

Minimum Required GPU	GeForce 8
White Paper	OpticalFlow.pdf
Source	HSOpticalFlow

4.15 Post-Process in OpenGL

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

Minimum Required GPU	GeForce 8
Source	postProcessGL

4.16 Recursive Gaussian Filter

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

Minimum Required GPU	GeForce 8
Source	recursiveGaussian

4.17 Sobel Filter

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

Minimum Required GPU	GeForce 8
Source	SobelFilter

4.18 Stereo Disparity Computation (SAD SIMD Intrinsics)

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

Minimum Required GPU	GeForce GTX 400
Source	stereoDisparity

4.19 Texture-based Separable Convolution

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

Minimum Required GPU	GeForce 8
Source	convolutionTexture

Chapter 5. FINANCE

5.1 Binomial Option Pricing

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

Minimum Required GPU	GeForce 8
White Paper	binomialOptions.pdf
Source	binomialOptions

5.2 Black-Scholes Option Pricing

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

Minimum Required GPU	GeForce 8
White Paper	BlackScholes.pdf
Source	BlackScholes

5.3 Excel 2007 CUDA Integration Example

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

Minimum Required GPU	GeForce 8
Source	ExcelCUDA2007

5.4 Excel 2010 CUDA Integration Example

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

Minimum Required GPU	GeForce 8
Source	ExcelCUDA2010

5.5 Excel CUDA Integration Example

This sample Demonstrates how one could integrate Excel with CUDA using array formulas. This plug-in is not pre-built with the SDK installer.

Minimum Required GPU	GeForce 8
Source	ExcelCUDA

5.6 Monte Carlo Option Pricing with Multi-GPU support

This sample evaluates fair call price for a given set of European options using the Monte Carlo approach, taking advantage of all CUDA-capable GPUs installed in the system. This sample use double precision hardware if a GTX 200 class GPU is present. The sample also takes advantage of CUDA 4.0 capability to supporting using a single CPU thread to control multiple GPUs

Minimum Required GPU	GeForce 8
White Paper	MonteCarlo.pdf
Source	MonteCarloMultiGPU

5.7 Niederreiter Quasirandom Sequence Generator

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

Minimum Required GPU	GeForce 8
Source	quasirandomGenerator

5.8 Sobol Quasirandom Number Generator

This sample implements Sobol Quasirandom Sequence Generator.

Minimum Required GPU	GeForce 8
Source	SobolQRNG

Chapter 6. SIMULATIONS

6.1 CUDA FFT Ocean Simulation

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

Minimum Required GPU	GeForce 8
Source	oceanFFT

6.2 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". Starting in CUDA 4.0, the nBody sample has been updated to take advantage of new features to easily scale the n-body simulation across multiple GPUs in a single PC. Adding "-numbodies=<bodies>" to the command line will allow users to set # of bodies for simulation. Adding "-numdevices=<N>" to the command line option will cause the sample to use N devices (if available) for simulation. In this mode, the position and velocity data for all bodies are read from system memory using "zero copy" rather than from device memory. For a small number of devices (4 or fewer) and a large enough number of bodies, bandwidth is not a bottleneck so we can achieve strong scaling across these devices.

Minimum Required GPU	GeForce 8
White Paper	nbody_gems3_ch31.pdf
Source	nbody

6.3 Fluids (Direct3D Version)

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

Minimum Required GPU	GeForce 8
Source	fluidsD3D9

6.4 Fluids (OpenGL Version)

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

Minimum Required GPU	GeForce 8
White Paper	fluidsGL.pdf
Source	fluidsGL

6.5 Particles

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

Minimum Required GPU	GeForce 8
White Paper	particles.pdf
Source	particles

6.6 Smoke Particles

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

Minimum Required GPU	GeForce 8
White Paper	smokeParticles.pdf
Source	smokeParticles

6.7 VFlockingD3D10

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

Minimum Required GPU	GeForce 8
Source	VFlockingD3D10

Chapter 7. ADVANCED

7.1 Advanced Quicksort (CUDA Dynamic Parallelism)

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

Minimum Required GPU	KEPLER SM 3.5
Source	cdpAdvancedQuicksort

7.2 Aligned Types

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

Minimum Required GPU	GeForce 8
Source	alignedTypes

7.3 Bezier Line Tesselation (CUDA Dynamic Parallelism)

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

Minimum Required GPU	KEPLER SM 3.5
Source	cdpBezierTessellation

7.4 Concurrent Kernels

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

Minimum Required GPU	GeForce 8
Source	concurrentKernels

7.5 CUDA C 3D FDTD

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

Minimum Required GPU	GeForce 8
Source	FDTD3d

7.6 CUDA Context Thread Management

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

Minimum Required GPU	GeForce 8
Source	threadMigration

7.7 CUDA Parallel Prefix Sum (Scan)

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

Minimum Required GPU	GeForce 8
Source	scan

7.8 CUDA Parallel Prefix Sum with Shuffle Intrinsics (SHFL_Scan)

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

Minimum Required GPU	KEPLER SM 3.0
Source	shfl_scan

7.9 CUDA Parallel Reduction

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

Minimum Required GPU	GeForce 8
White Paper	reduction.pdf
Source	reduction

7.10 CUDA Radix Sort using the 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.

Minimum Required GPU	GeForce 8
White Paper	readme.txt
Source	radixSortThrust

7.11 CUDA Segmentation Tree Thrust Library

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

Minimum Required GPU	GeForce GTX 200
Source	segmentationTreeThrust

7.12 CUDA Sorting Networks

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

Minimum Required GPU	GeForce 8
Source	sortingNetworks

7.13 Eigenvalues

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

Minimum Required GPU	GeForce 8
White Paper	eigenvalues.pdf
Source	eigenvalues

7.14 Fast Walsh Transform

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

Minimum Required GPU	GeForce 8
Source	fastWalshTransform

7.15 Function Pointers

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

Minimum Required GPU	GeForce GTX 400
Source	FunctionPointers

7.16 Line of Sight

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

Minimum Required GPU	GeForce 8
Source	lineOfSight

7.17 LU Decomposition (CUDA Dynamic Parallelism)

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

Minimum Required GPU	KEPLER SM 3.5
Source	cdpLUDecomposition

7.18 Matrix Transpose

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

Minimum Required GPU	GeForce 8
White Paper	MatrixTranspose.pdf
Source	transpose

7.19 Merge Sort

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

Minimum Required GPU	GeForce 8
Source	mergeSort

7.20 NewDelete

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

Minimum Required GPU	GeForce GTX 400
Source	newdelete

7.21 PTX Just-in-Time compilation

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

Minimum Required GPU	GeForce 8
Source	ptxjit

7.22 Quad Tree (CUDA Dynamic Parallelism)

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

Minimum Required GPU	KEPLER SM 3.5
Source	cdpQuadtree

7.23 Scalar Product

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

Minimum Required GPU	GeForce 8
Source	scalarProd

7.24 simpleHyperQ

This sample demonstrates the use of CUDA streams for concurrent execution of several kernels on devices which provide HyperQ. Devices without HyperQ will run a maximum of two kernels concurrently.

Minimum Required GPU	GeForce 8
White Paper	HyperQ.pdf
Source	simpleHyperQ

7.25 threadFenceReduction

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

Minimum Required GPU	GeForce 8
Source	threadFenceReduction

Chapter 8. CUDALIBRARIES

8.1 batchCUBLAS

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

Minimum Required GPU	GeForce 8
Source	batchCUBLAS

8.2 Box Filter with NPP

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

Minimum Required GPU	GeForce 8
Source	boxFilterNPP

8.3 ConjugateGradient

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

Minimum Required GPU	GeForce 8
Source	conjugateGradient

8.4 FreeImage and NPP Interopability

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

Minimum Required GPU	GeForce 8

Source	freeImageInteropNPP

8.5 GrabCut with NPP

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

Minimum Required GPU	GeForce 8
Source	grabcutNPP

8.6 Histogram Equalization with NPP

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

Minimum Required GPU	GeForce 8
Source	histEqualizationNPP

8.7 Image Segmentation using Graphcuts with NPP

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

Minimum Required GPU	GeForce 8
Source	imageSegmentationNPP

8.8 MersenneTwisterGP11213

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

Minimum Required GPU	GeForce 8
Source	MersenneTwisterGP11213

8.9 Monte Carlo Estimation of Pi (batch inline QRNG)

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

Minimum Required GPU	GeForce 8
Source	MC_EstimatePiInlineQ

8.10 Monte Carlo Estimation of Pi (batch PRNG)

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

Minimum Required GPU	GeForce 8
Source	MC_EstimatePiP

8.11 Monte Carlo Estimation of Pi (batch QRNG)

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

Minimum Required GPU	GeForce 8
Source	MC_EstimatePiQ

8.12 Monte Carlo Estimation of Pi (inline PRNG)

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

Minimum Required GPU	GeForce 8
Source	MC_EstimatePiInlineP

8.13 Monte Carlo Single Asian Option

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

Minimum Required GPU	GeForce 8
Source	MC_SingleAsianOptionP

8.14 Preconditioned Conjugate Gradient

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

Minimum Required GPU	GeForce 8
Source	conjugateGradientPrecond

8.15 Random Fog

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

Minimum Required GPU	GeForce 8
Source	randomFog

8.16 Simple CUBLAS

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

Minimum Required GPU	GeForce 8
Source	simpleCUBLAS

8.17 Simple CUFFT

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

Minimum Required GPU	GeForce 8
Source	simpleCUFFT

8.18 simpleDevLibCUBLAS GPU Device API Library Functions (CUDA Dynamic Parallelism)

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

Minimum Required GPU	KEPLER SM 3.5
Source	simpleDevLibCUBLAS

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