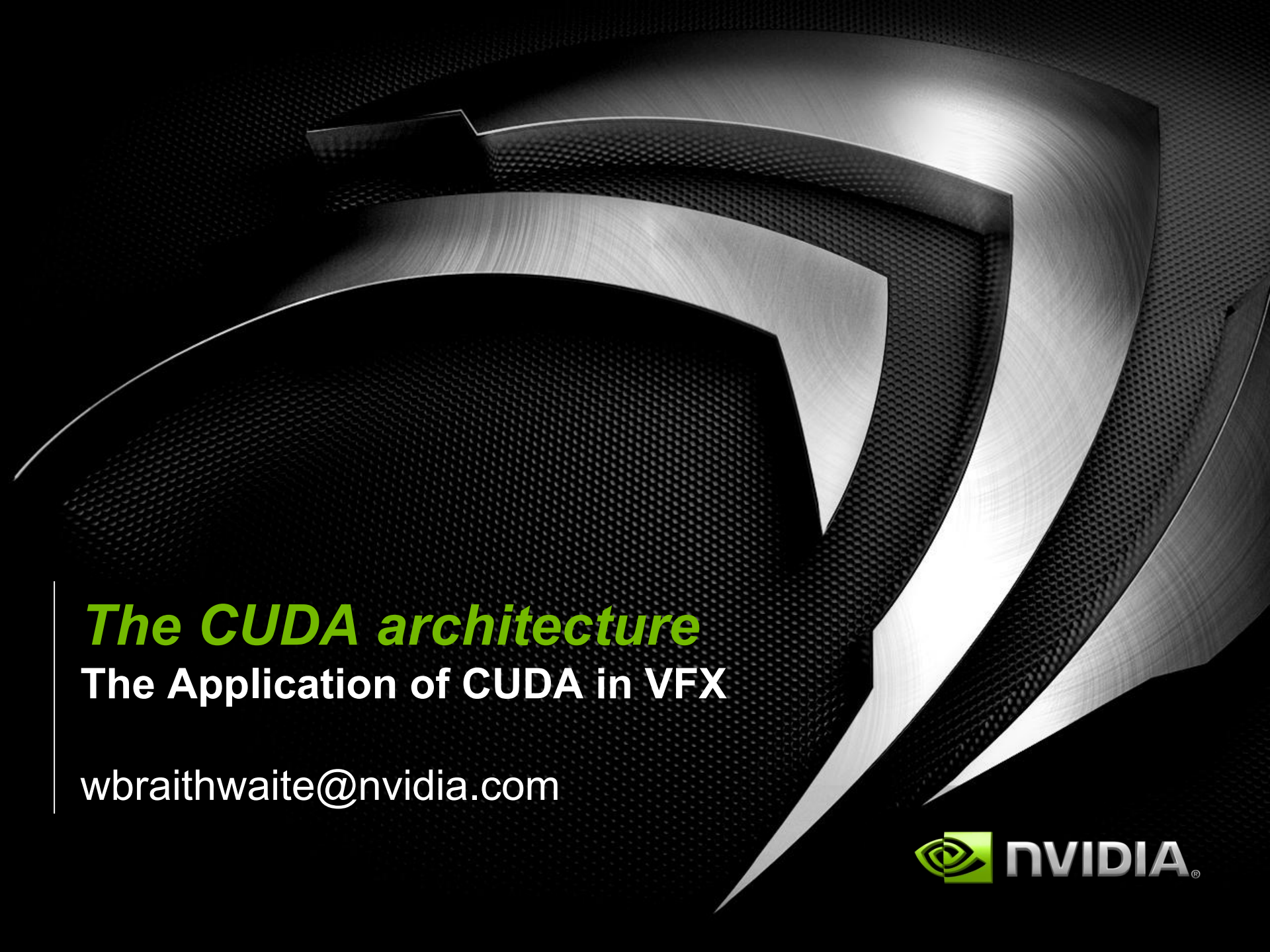

The CUDA architecture
The Application of CUDA in VFX
wbraithwaite@nvidia.com
NVIDIA

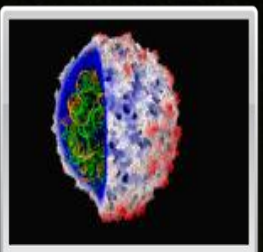
Motivation

- Pros ☺
  - Shallow learning curve – C/C++, language integration
  - People have achieved impressive performance
  - High-end NVIDIA cards have lots of RAM = complex tasks
    - 4GB, and more in future architectures...
    - CUDA programming model lets YOU control it
  - Scalable architecture
    - write once... and run for generations. (CUDA produced since 2007)
  - Large CUDA community... that is growing
- Potential cons? ☹
  - Data-parallel computing is a new craze. How long before something else comes along?
  - Parallel code looks really complicated. Why go parallel?
  - CUDA is another API to learn. Is it worth it?

Wide Developer Acceptance & Success

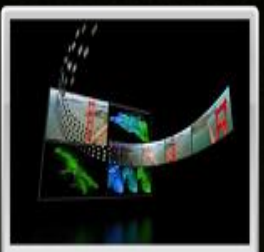
146X

Interactive visualization of volumetric white matter connectivity



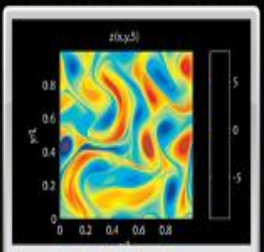
36X

Ion placement for molecular dynamics simulation



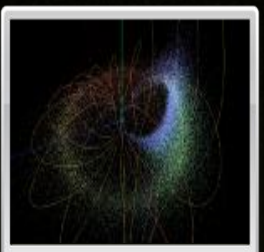
19X

Transcoding HD video stream to H.264



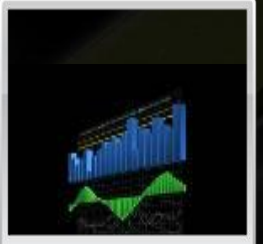
17X

Simulation in Matlab using .mex file CUDA function



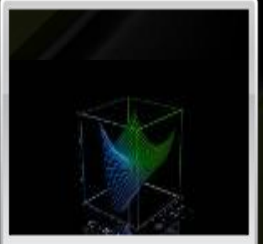
100X

Astrophysics N-body simulation



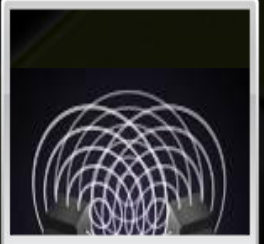
149X

Financial simulation of LIBOR model with swaptions



47X

GLAME@lab: An M-script API for linear Algebra operations on GPU

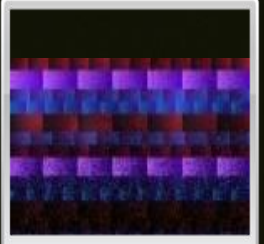


20X

Ultrasound medical imaging for cancer diagnostics

24X

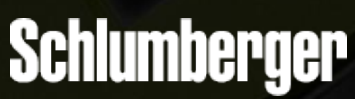
Highly optimized object oriented molecular dynamics



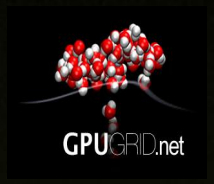
30X

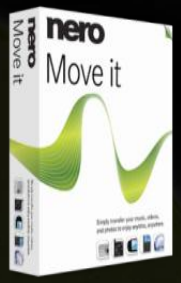
Cmatch exact string matching to find similar proteins and gene sequences

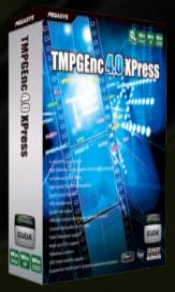
Commercial Adoption

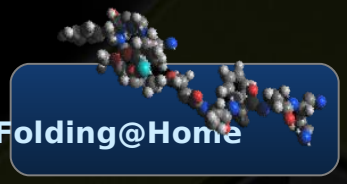


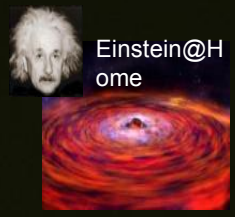
GPU Accelerated Consumer Apps





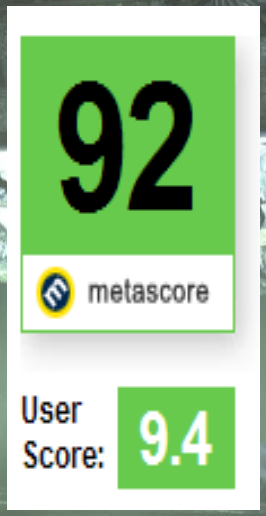






Batman: Arkham Asylum with PhysX

- Highest rated PC game since 2007!
  - Smoke reacts with Batman
  - Walls explode
  - Glass shatters
  - Tattered curtains react with characters



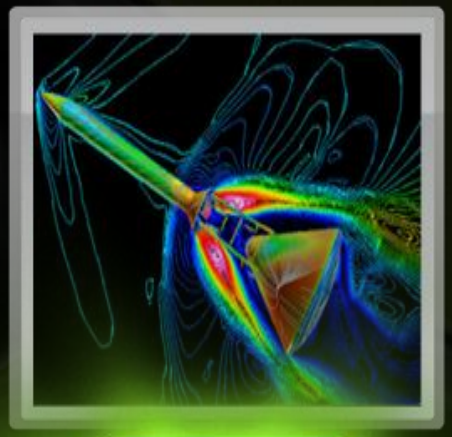
Parallel Computing on All GPUs

Over 100 Million CUDA Capable GPUs Deployed

GeForce®
Entertainment

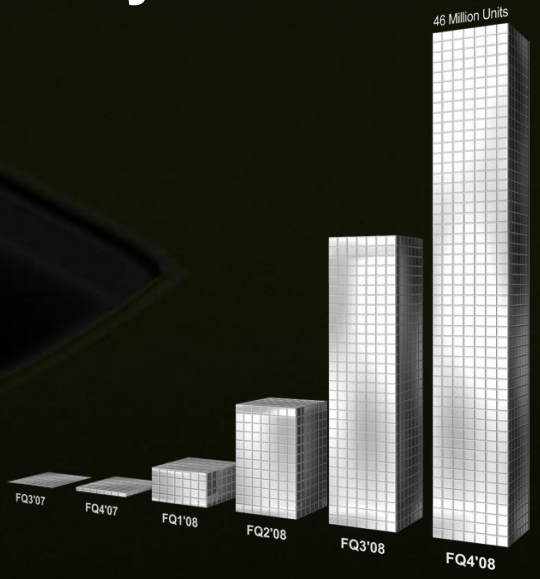
Tesla™
High-Performance Computing

Quadro®
Design & Creation

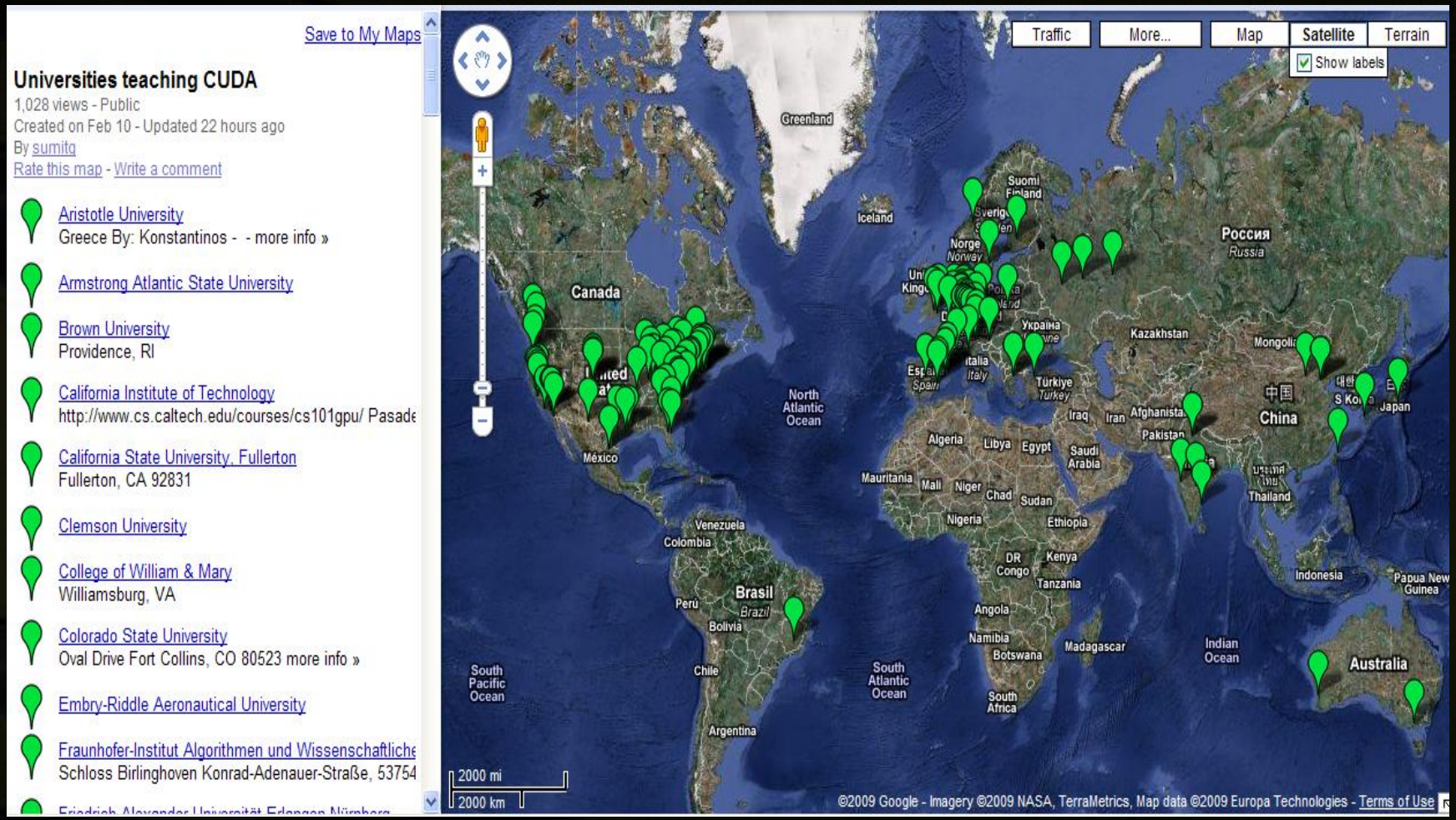


Democratization of Parallel Computing

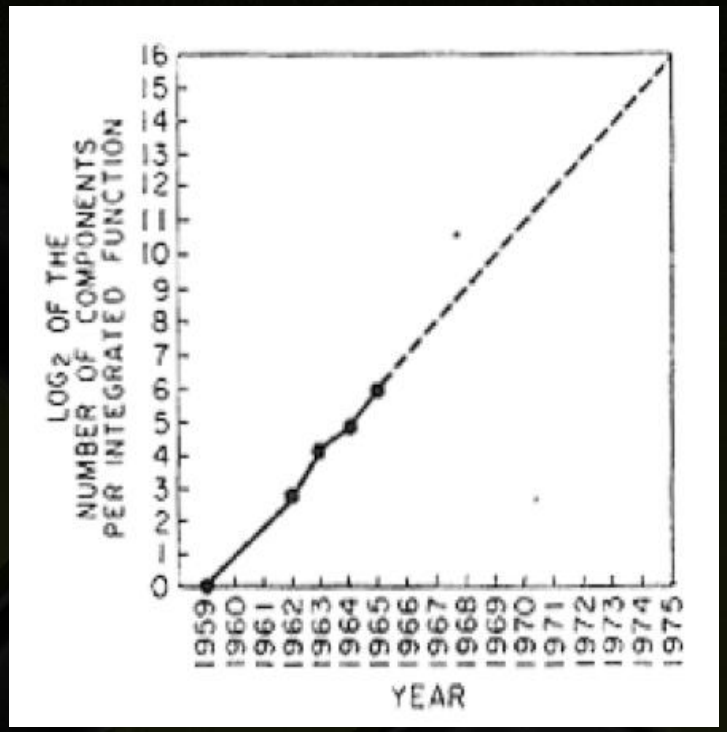
- GPUs and CUDA bring parallel computing to the masses
  - > 1,000,000 CUDA-capable GPUs sold to date
  - > 100,000 CUDA developer downloads
  - Spend only ~$200 for 500 GFLOPS!
- Data-parallel supercomputers are everywhere
  - CUDA makes this power accessible
  - We're already seeing innovations in data-parallel computing
  - Massive multiprocessors are a commodity technology



200+ Universities Teaching GPU Computing on CUDA



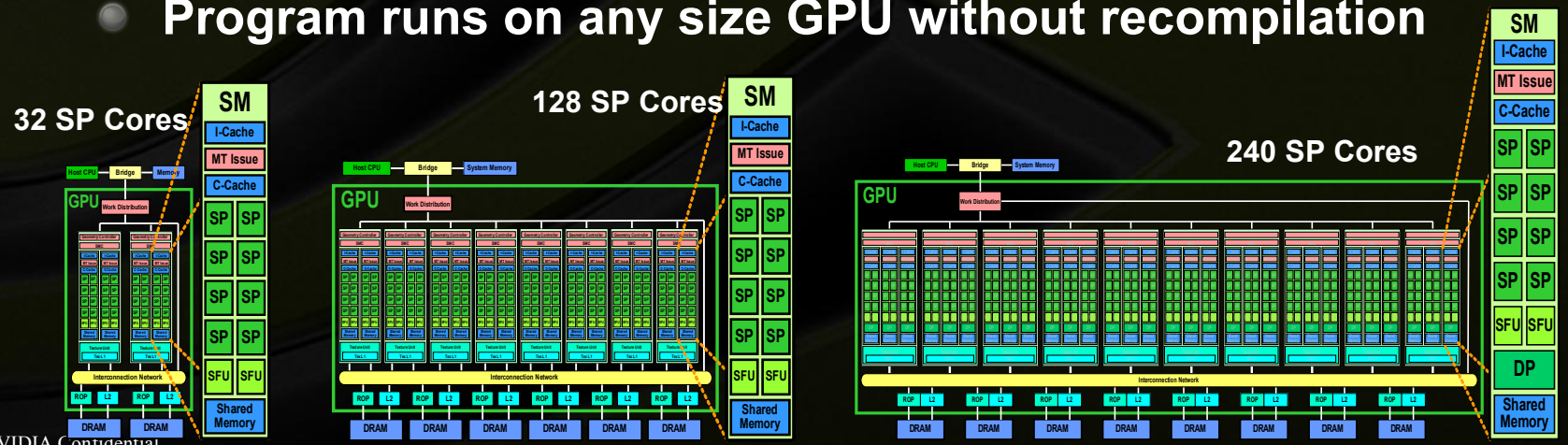
Moore's Law



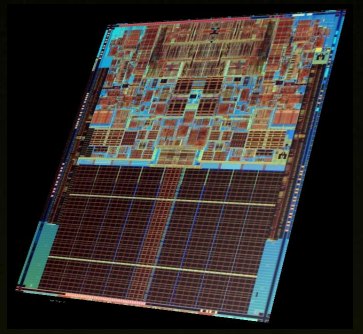
- Moore’s law gives us more transistors
  - No prediction of processor performance
  - Advances in architecture turn device scaling into performance scaling
- Architects turn transistors into more performance
- Applications turn performance into value for the user

Parallelism is increasing rapidly

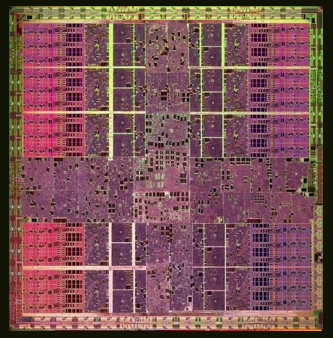
- CPUs and GPUs are both parallel processors, but...
  - CPUs now have 2, 4, 8, … processors
  - GPUs now have 32, 64, 128, 240, … processors
- Parallelism is increasing rapidly with Moore's Law
  - Processor count doubles every 18 – 24 months
  - Individual processor cores no longer getting faster
- CUDA is a *scalable* parallel architecture
  - Program runs on any size GPU without recompilation



Throughput processors



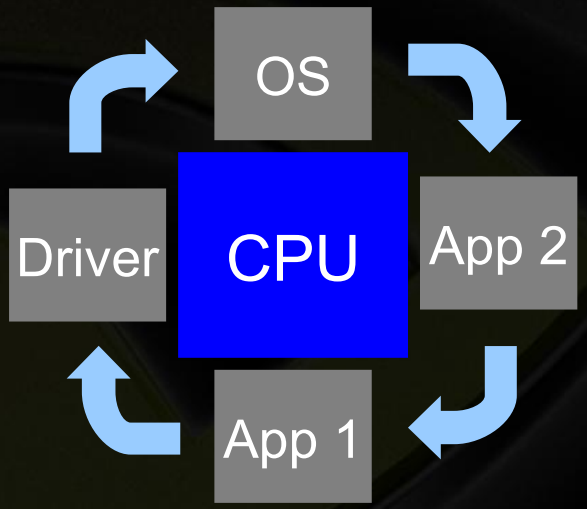
- Latency optimized processors are improving very slowly
  - Little value is being delivered from the evolution of these processors



- Throughput optimized processors are still improving at >70% per year
  - This drives new throughput applications that convert this performance to value

- Going forward, throughput processors matter, not latency processors

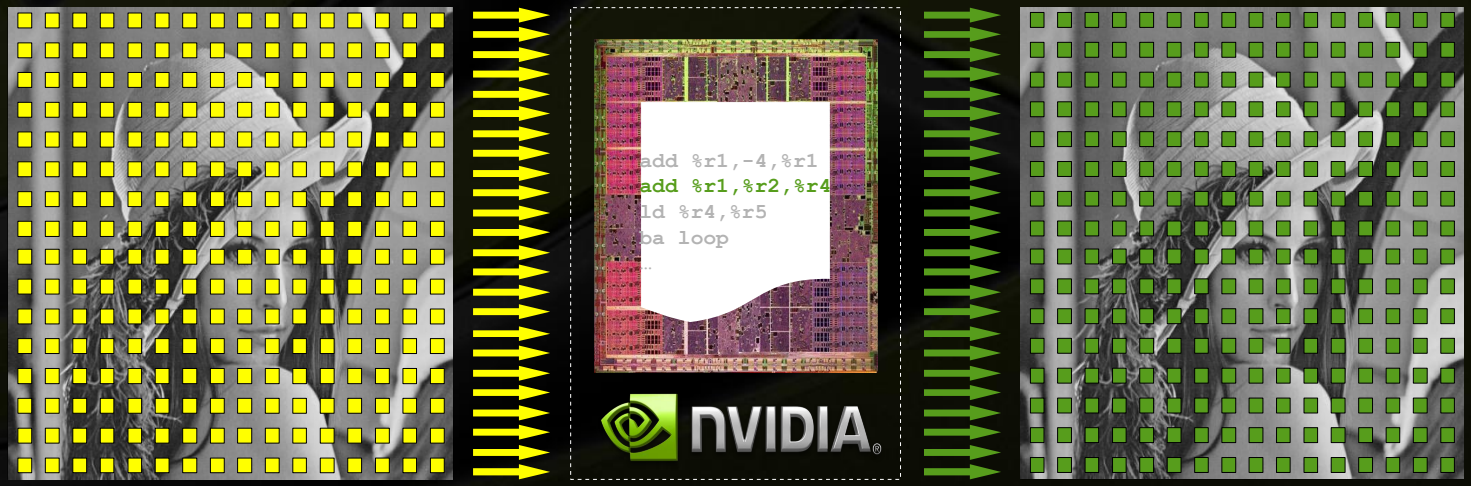
Task-Parallel processing



- Conventional Task-Parallel approach:
  - Different threads perform different tasks
  - independent processes with little communication
  - Simple to implement
  - Can have arbitrary program flow
    - lots of conditional logic and decisions
  - Applications must be re-engineered to scale with multiple cores

Data-Parallel programming

- Every data-item is represented by a thread
  - Vast amounts of data
  - The same computation on all data
  - Minimal dependencies between data items in each step
- Saturates many ALUs
  - maximal arithmetic intensity
- Generally requires redesign of algorithms



The golden age of supercomputing

- 1980s, early 1990s
  - Particularly data-parallel computing

Cray X-MP (1982)

- Architectures
  - Connection Machine, MasPar, Cray
  - True Supercomputers: incredibly exotic, powerful, expensive

Thinking Machines CM-1 (1984)

- Algorithms, languages, & programming models
  - Solved a wide variety of problems
  - Varius parallel algorithmic models developed
  - P-RAM, V-RAM, circuit, hypercube, etc.

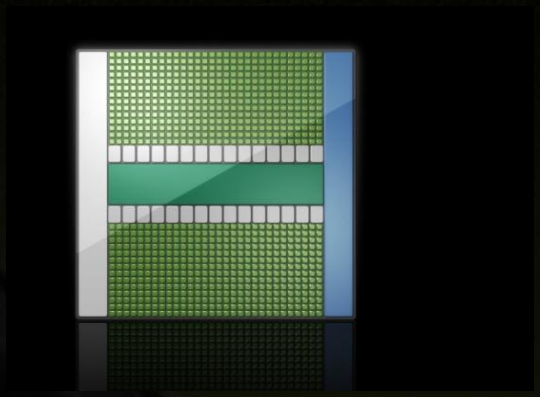
MasPar MP-1 (1990)

- Disappeared due to advance of commodity hardware

Heterogeneous Processing

- CPU:
  - fast caches (data reuse)
  - fine branching granularity
  - lots of different processes running
  - high performance on a single thread
  - Threads are heavyweight
  - CPU with 4 x Quad cores have up to 16 active threads at a time
- GPU:
  - lots of math units
  - fast access to onboard memory
  - runs parallel program on graphics elements (pixels vertices)
  - high throughput on parallel tasks
  - Threads are lightweight
  - Current GPUs have 10000s of active threads

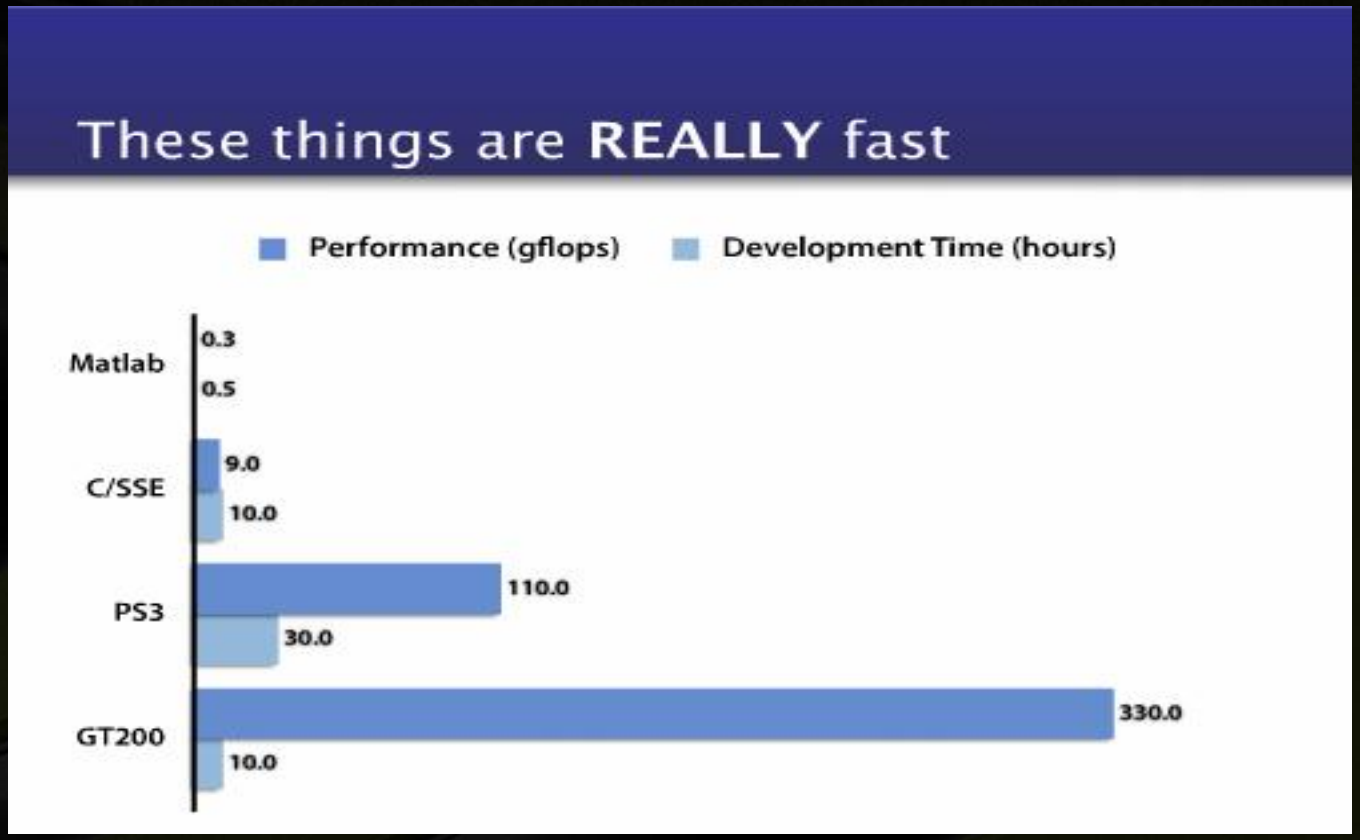
CUDA Programming Model Highlights



- Let programmers focus on parallel algorithms
  - Not mechanics of a parallel programming language
- Scale to 100's of cores, 1000's of parallel threads
  - Transparently, with one source and same binary
- Enable heterogeneous systems (CPU+GPU)
  - CPU good at serial tasks, GPU at parallel data computation

Higher Productivity with CUDA C

Performance vs. Programming Effort (!/$)

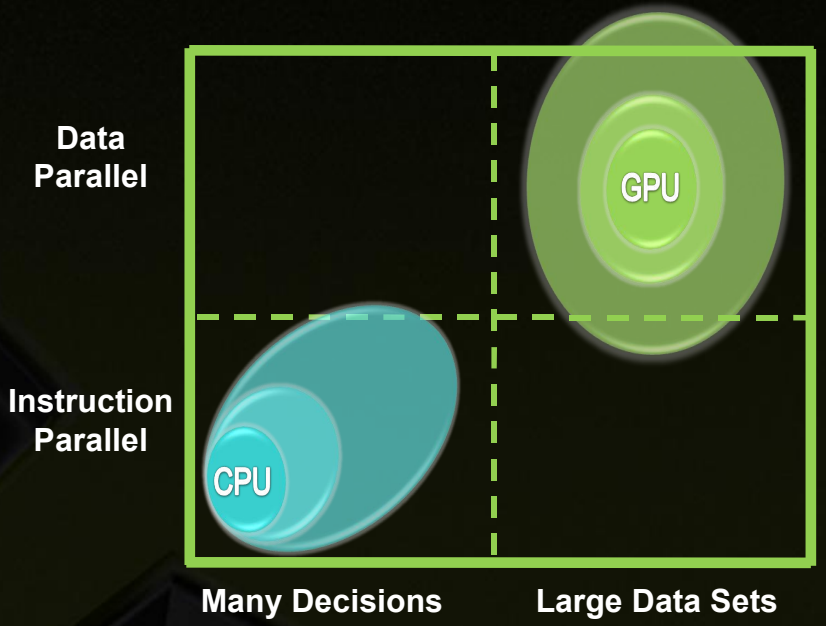


- From a Harvard / MIT presentation

http://www.slideshare.net/npinto/iap09-cudamit-6963-guest-lecture-unlocking-biologicallyinspired-computer-vision-a-highthroughput-approach-david-cox-harvard-jim-dicarlo-nicolas-pinto-mit-presentation

Performance Benefit

- GPUs are ideally suited for data parallel code



- Look for "exploitable concurrency"
  - Find fundamental parts of the algorithm that are separable

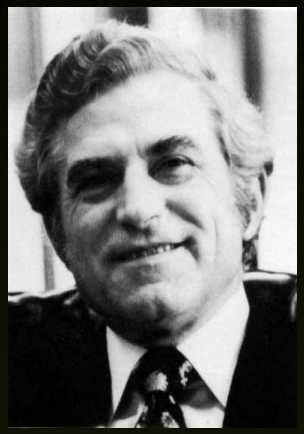
Hardware constraints

- PCI bus traffic should be minimized
  - You must justify transferring code to the device
  - Data wants to be kept on device for as long as possible
- Minimal dynamic memory allocation
- Data access coherency
  - adjacent threads access adjacent memory
- Predictable program flow
  - similar workload on threads and minimal synchronization
- Maximal arithmetic intensity
  - hide memory latency
  - keep threads occupied

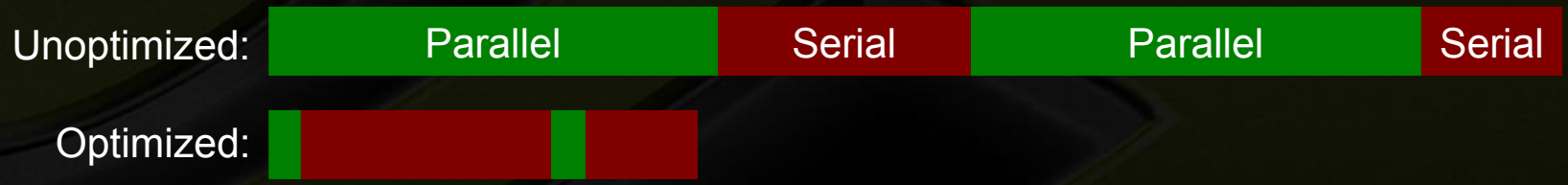
Amdahl's Law – Example

- P = parallel proportion
- N = number of procs

$$S = \frac{1}{(1-P) + \frac{P}{N}}.$$



- Assume N → infinity
- Only ¾ of program can be parallelized
- S = 4



- The maximum speedup can only be 4x

Performance benefit

- Does massively parallel hardware has limited benefit?
  - NO, because...
- More compute power can mean more possibilities
  - Sometimes only 4x can be the difference between interactivity and offline processing

Parallelization → Interactivity (→ Fun)

Image from Harry Potter and the Half Blood Prince, courtesy of Warner Brothers

Embarrassingly parallel

- VFX has many algorithms to target for parallelization
- Some low-hanging fruit:
  - Rendering
    - Rasterization / shading already runs on GPUs
    - Visibility culling
    - Procedural geometry
  - Image processing
  - Compression / Decompression
    - Animation, audio, video, etc.
  - Animation blending
  - Rigid-body physics
  - Some collision calculations
  - Etc.

Graphics or Compute?

- Use GRAPHICS if...
  - you require only “gather”
  - you can take advantage of
    - framebuffer operations
    - hardware mipmapping
    - rasterization, etc.
- Use CUDA if...
  - you can take advantage of on-chip shared memory
  - you want to write the code in C/C++
  - you have complex data structures
  - you want scattered writes
  - you want to be sure of scalability for future devices

CUDA – Image processing

- Image processing is a natural fit for data-parallel
  - pixels → threads
  - Lots of data reuse
- CUDA offers advantages over shaders
  - Shared memory
  - Scattered writes
  - More flexible programming model
- CUDA includes functionality for sharing data with:
  - OpenGL, Direct3D, Video decoder hardware
- Many SDK examples available...

Image processing

- Complex convolutions
  - Fast box and recursive Gaussian filters, BRDFs
- Transforms
  - FFT, DCT, Wavelet
- Histograms
- Noise reduction
- Filters
  - Bicubic, median
- Optical flow
  - Tracking
  - Stabilization
  - Motion estimation

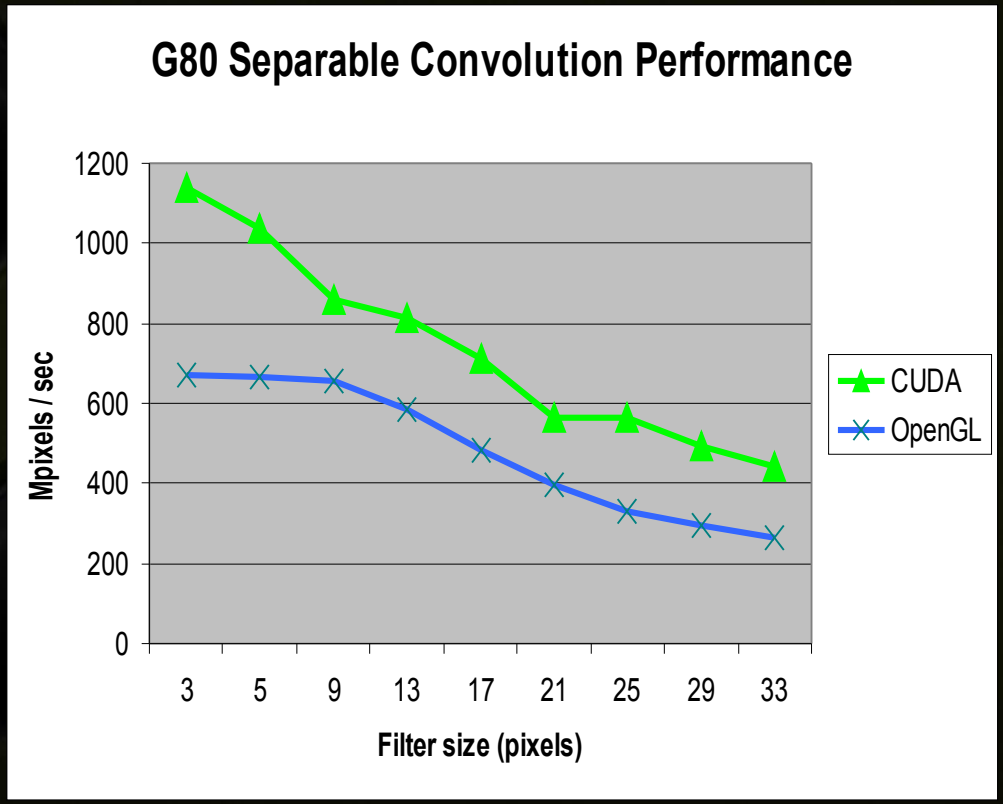
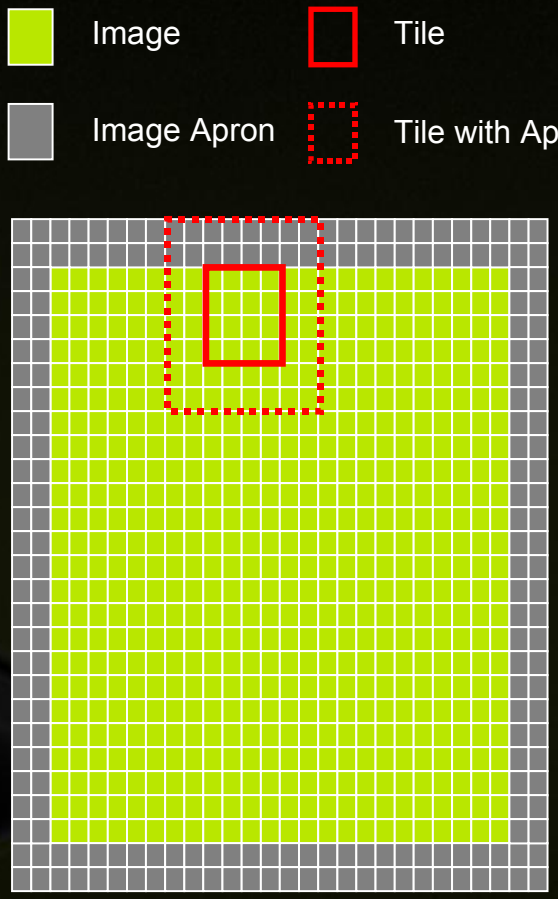


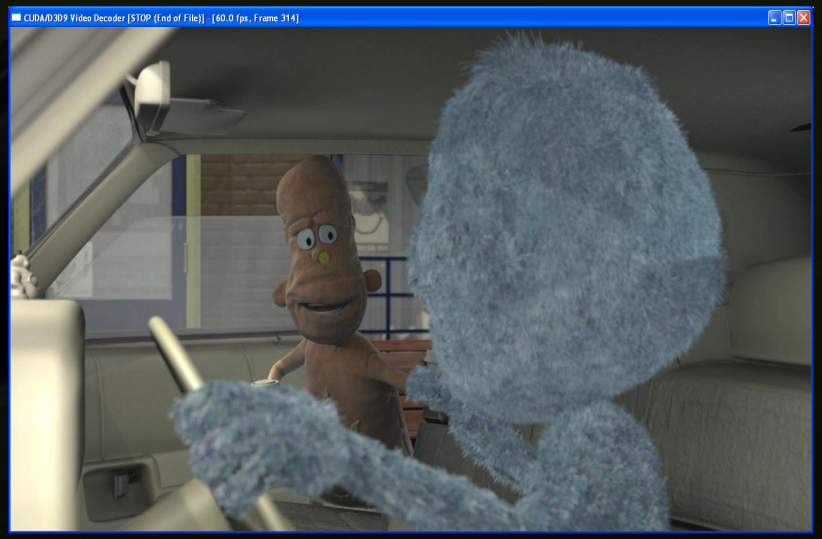
Image processing – Implementation

- Efficient memory usage
  - Store filter coefficients in constant memory
  - Cache image tile in shared memory
- Each output pixel needs to read adjacent pixels within radius
  - Tiles in shared memory must be expanded with an apron that contains neighboring pixels
- Pixels within apron write results
  - Some threads are wasted



CUDA – Video decoding

- NVCUVID: video extension for CUDA
  - Enables direct access to output of GPU
  - Supports MPEG(1/2) and H.264
  - Similar to DXVA API, but platform OS independent.
  - Decodes directly to 4:2:0 surface & can be mapped in CUDA
  - See “cudaVideoDecode” Windows sample in CUDA SDK 2.0



Video decoding – Commercial example

- **Badaboom Media Converter**

  Developed by Elemental Technologies

  - CUDA accelerated video transcoder
  - Converts almost anything to H.264 format
  - Converts HD video at more than 100 fps

CUDA – Panorama stitching

Work by James Fung at NVIDIA

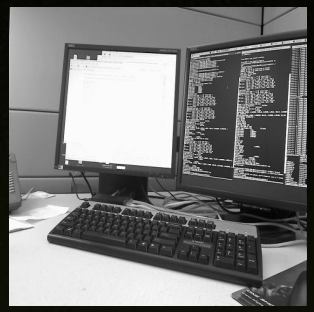
- **Panorama generated from three 3840 x 2880 images**
  - **completed in 0.577 seconds on a GTX-280 NVIDIA GPU**

CPU	GPU
Xeon E5440 2.83 GHz Quad-Core 4 GB RAM	Tesla C1060 4 GB Framebuffer
3.3 s	0.668 s

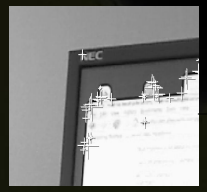
Comparison of GPU vs. Photoshop CS3 Photomerge Tool

Panaroma stitching algorithm

Left image



Right image

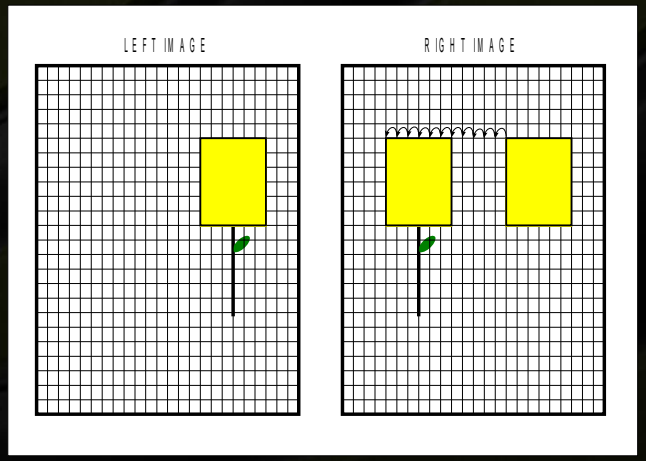


Keypoint Detection & Extraction (SIFT) → Keypoint Matching → Recover Homography (RANSAC) → Create Laplacian Pyramid → Projective Transform → Multi-Band Blend

CUDA – Stereo vision

Work by Joe Stam at NVIDIA

- **Find correspondence between left and right image for every pixel**
  - **640x480 pixels, 11x11 block match → 2 Billion computations for 50-pixel search (with no optimization)**



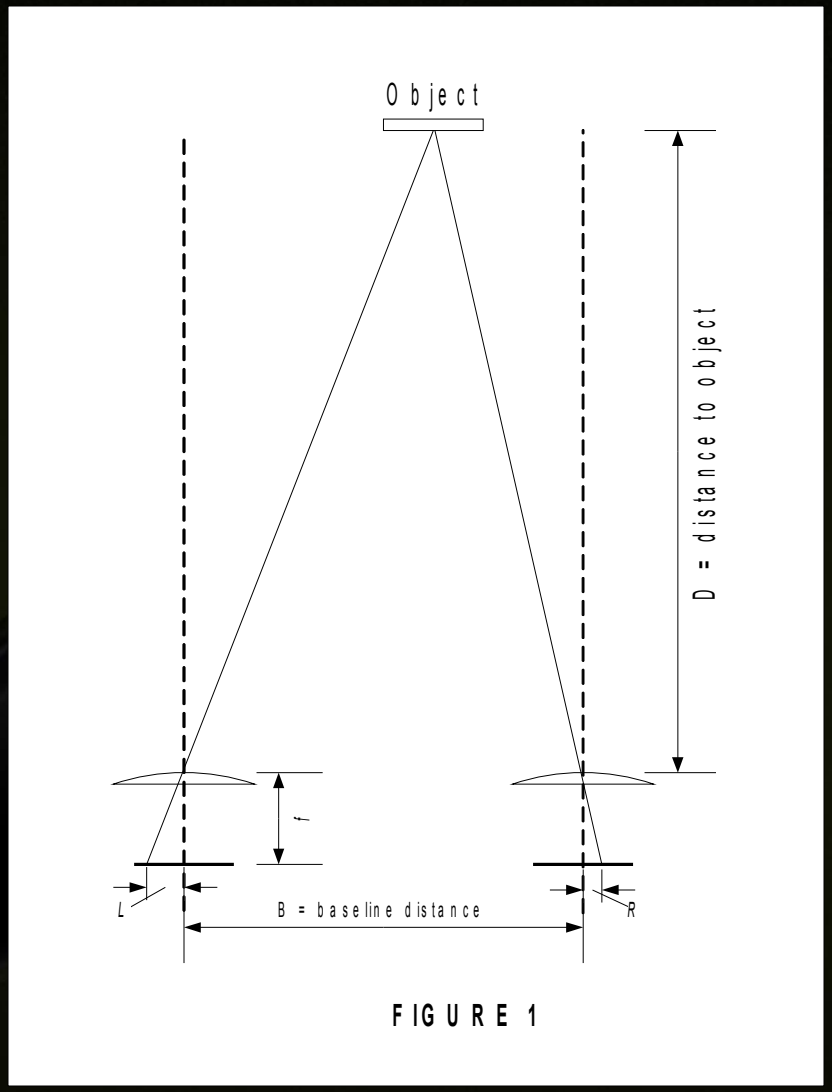
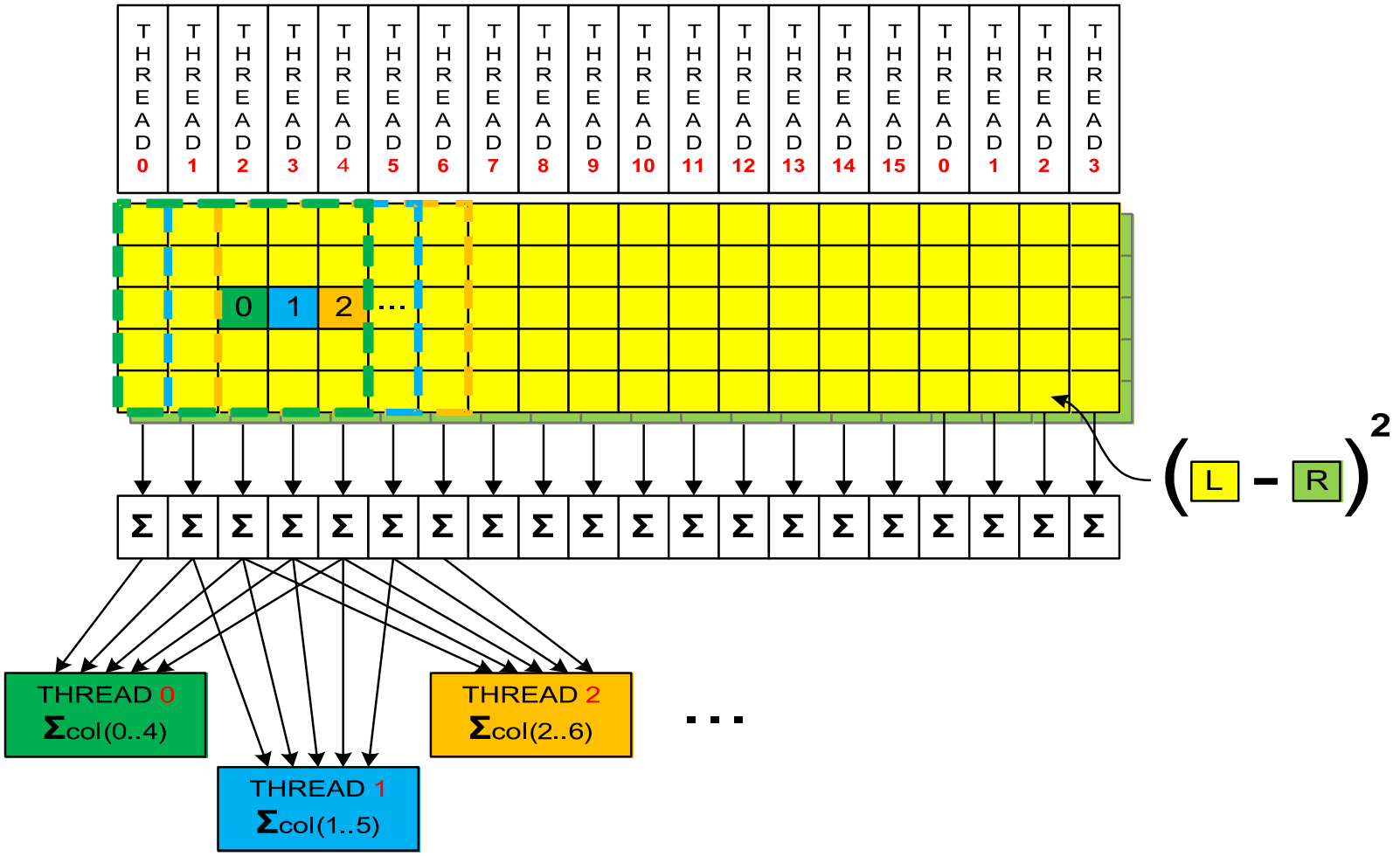
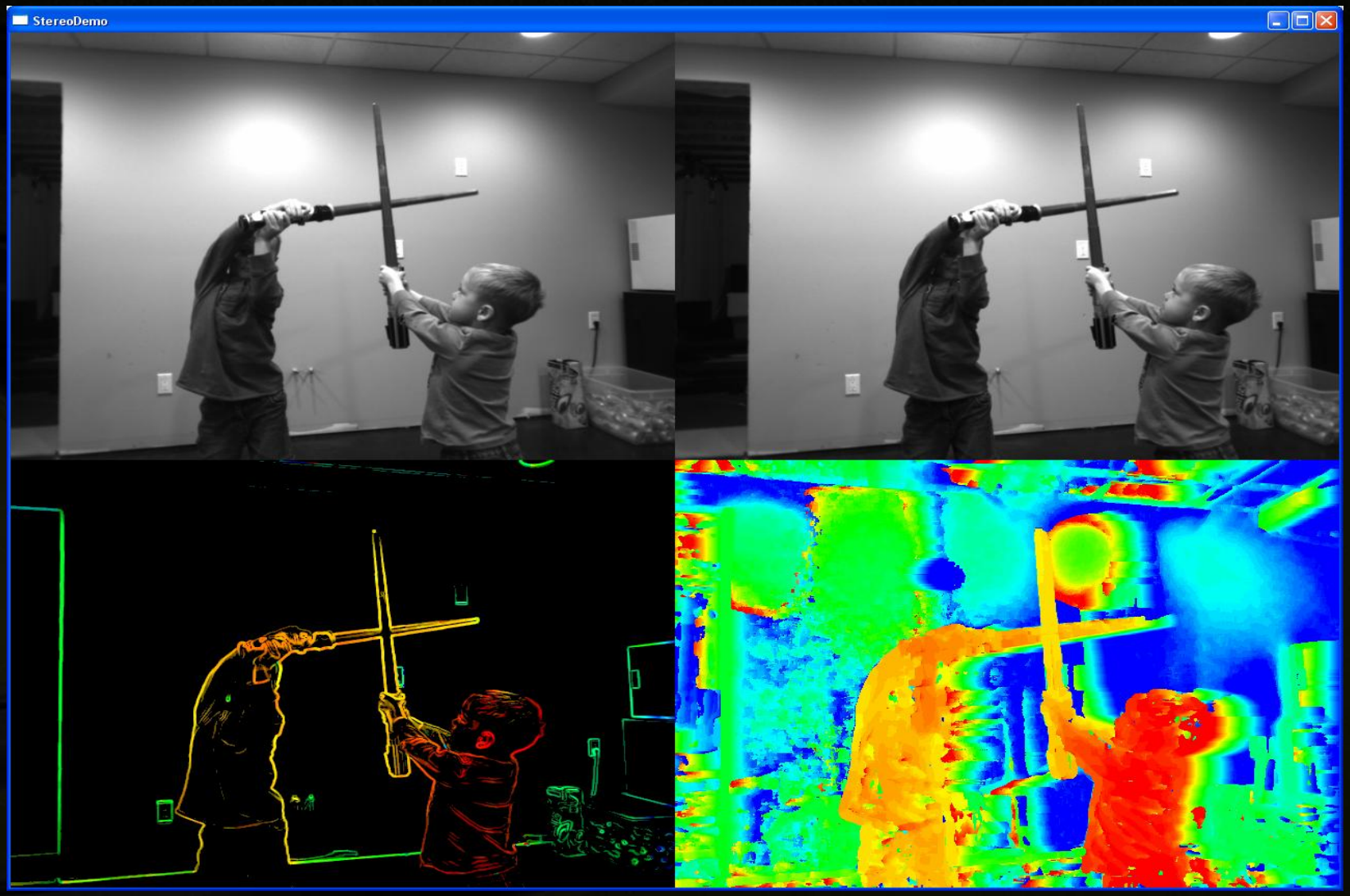


FIGURE 1

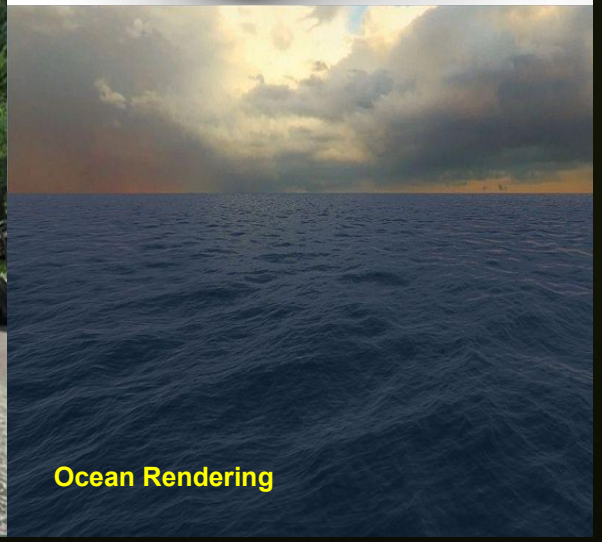
Stereo vision - Implementation



Stereo vision – Results



CUDA – Compositing



Some images courtesy Crytek GmBH

Compositing – Commercial example

- Davinci R4K
  - compositing & color grading
  - 4 x NVIDIA Quadroplexes

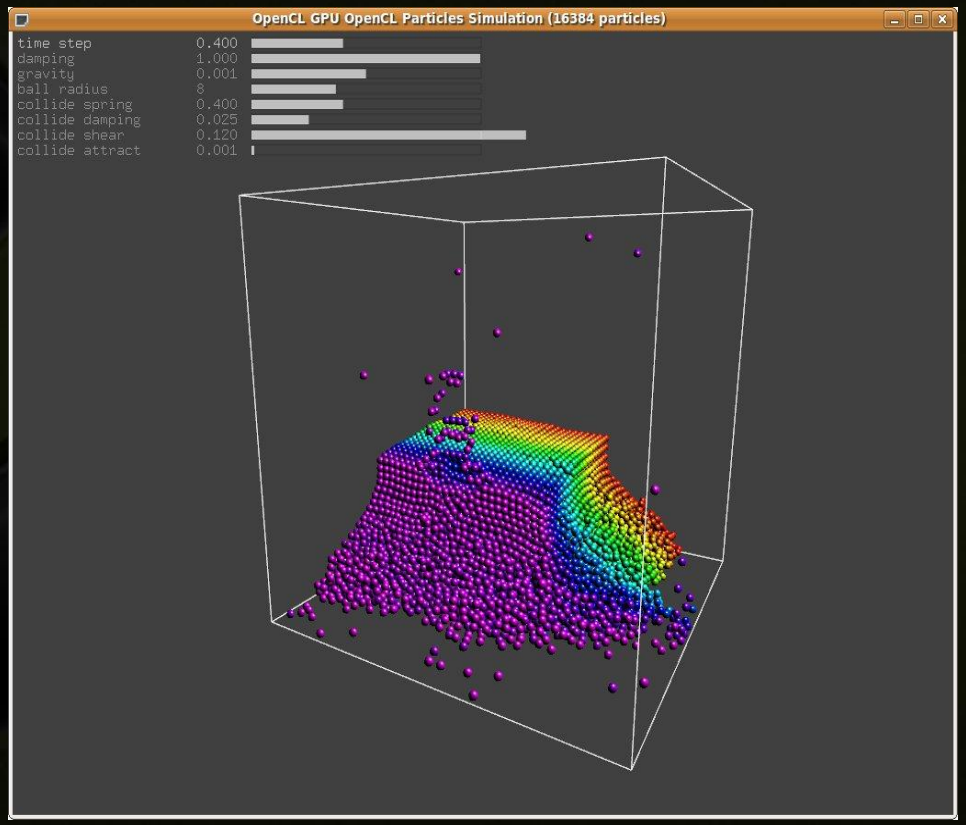
CUDA – Particles

- Very easy to parallelize
- Small memory footprint
  - Attributes are stored per particle
  - Modern GPUs can handle millions of particles
- Interacting particles
  - No particle-particle interaction is easy
  - Can be brute force (n-body simulation using CUDA in SDK)
  - Spatial subdivision and sorting on GPU

CUDA – Particle simulation

Work by Simon Green at NVIDIA

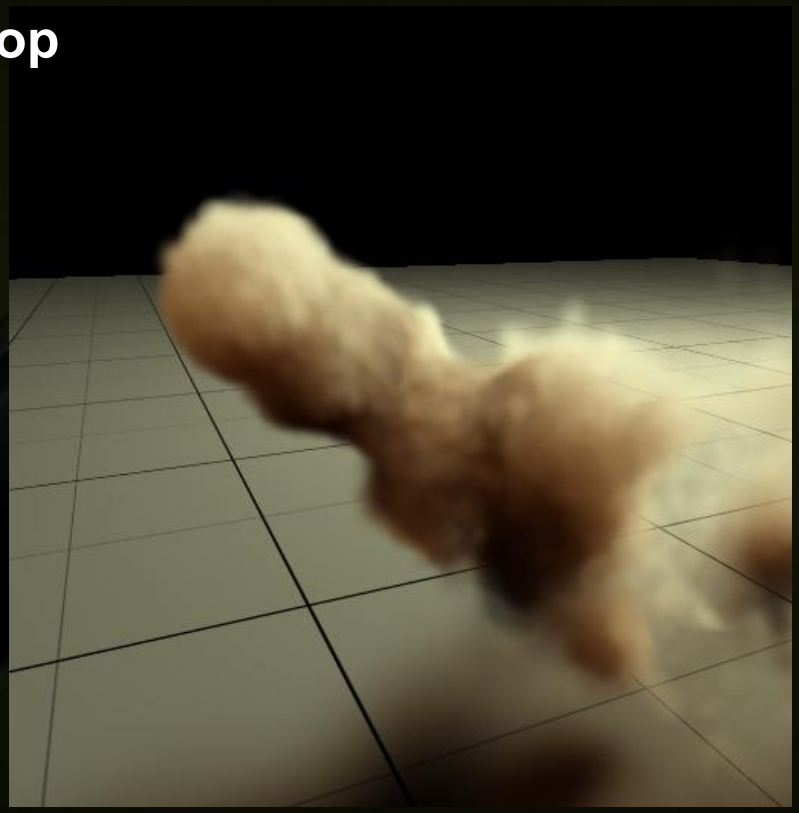
- Particle simulation
  - Spatially hashed into uniform grid
  - Sorted with radix-sort
  - Simulated using DEM
  - rendered using graphics interop
- Available in SDK



CUDA – Particle rendering

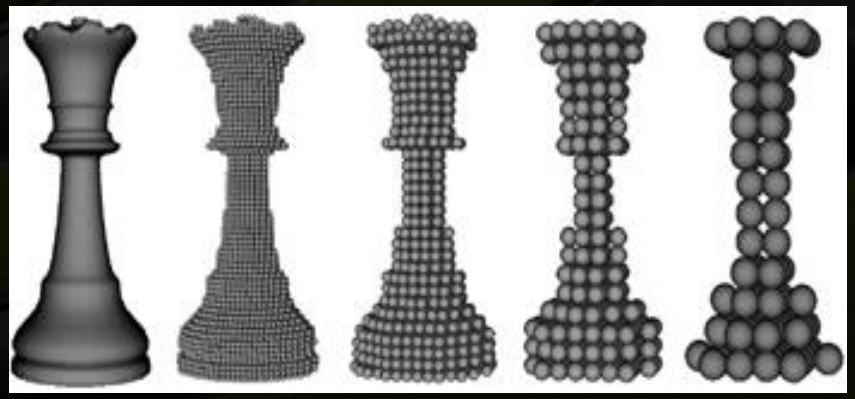
Work by Simon Green at NVIDIA

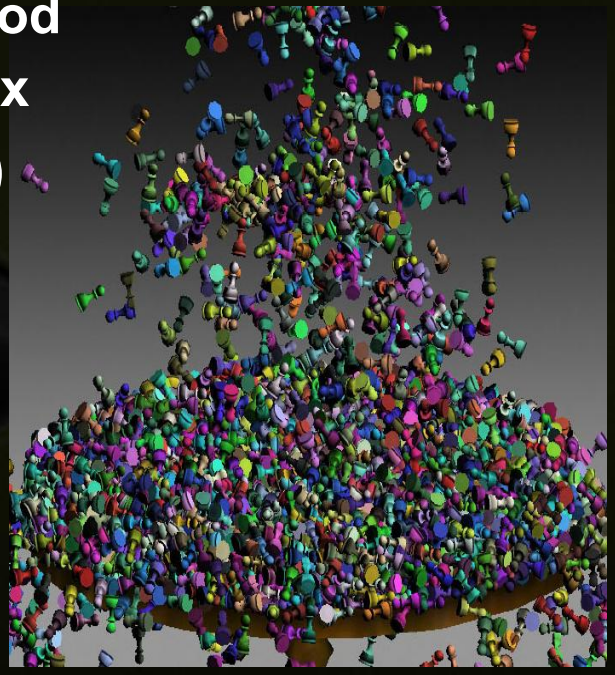
- Volumetric particle shadows
  - Simulated using noise functions
  - Sorted with radix-sort
  - rendered using graphics interop
- Available in SDK



CUDA – Physics

- Particle-based Rigid-Body Simulation
  - Host → device transfer of proxy mesh
  - Dynamic meshes voxelized
    - OPENGL methods: using depth peeling / clipping planes
    - CUDA methods (physX)
  - Simulate using Distinct Element Method
  - Device → host transfer of xform matrix
- Example in GPU gems 3 (Harada07)

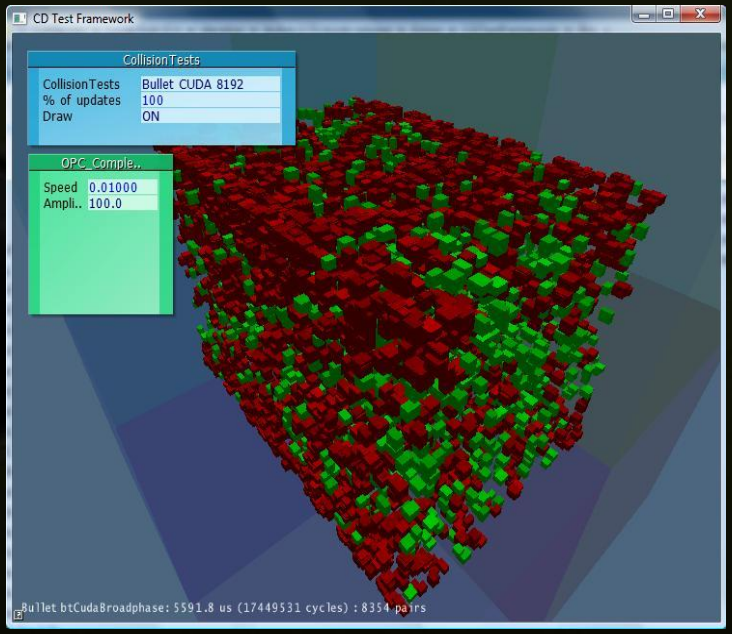




NVIDIA PhysX Creates Truly Dynamic Worlds Where Games Come Alive!

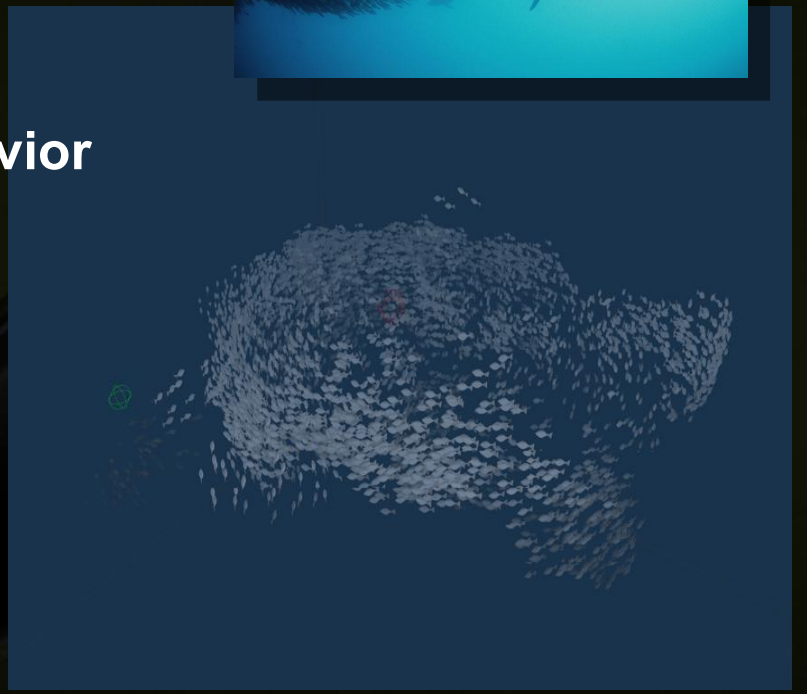
Physics – Commercial example

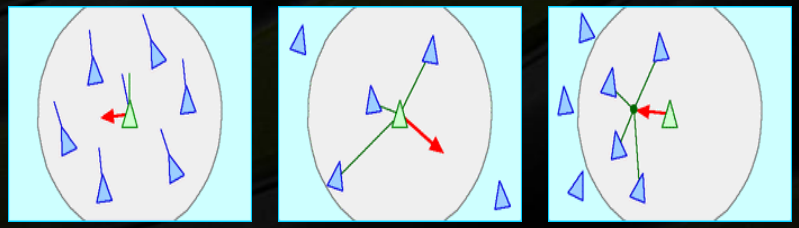
- Bullet Physics Library
  - Open-source Physics engine
  - Plugins
    - Maya
    - Houdini
    - Cinema4D
    - Blender...
  - rigid-body FX in the movie 2012
  - CUDA being implemented into the engine!
- Check it out:
  - http://bulletphysics.org



CUDA – Crowd simulation

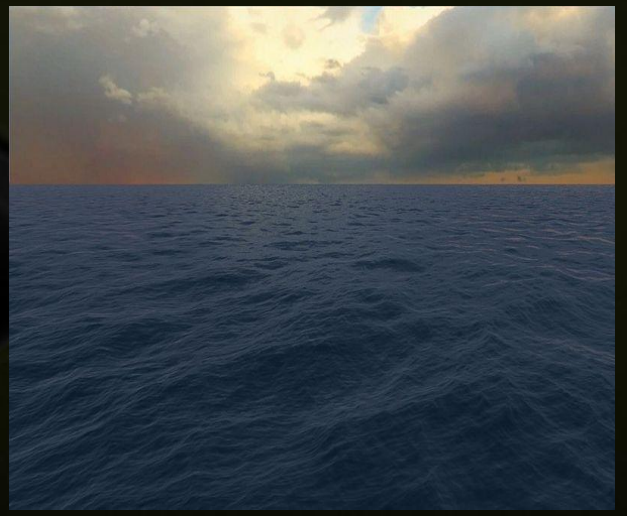
- Minimal host ↔ device transfer
  - Graphics interop for visualization
  - host → device transfer of controllers only
    - Prey, Predators, Obstacles, etc.
- Agent behavior scripts in C
  - easier to code complex behavior
- Very Interactive!





CUDA – Ocean simulation

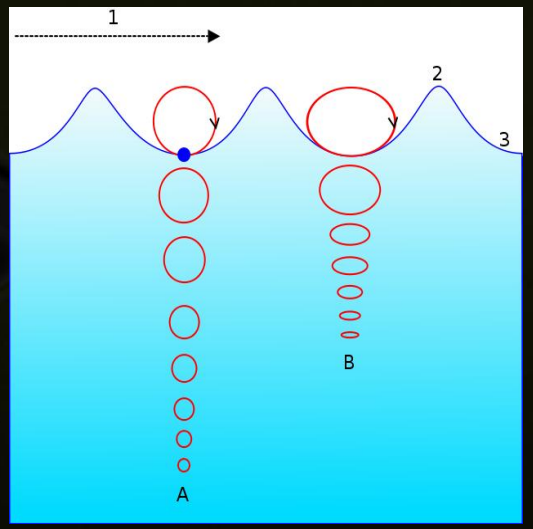
- Based on Jerry Tessendorf's paper: "Simulating Ocean Water"
  - Statistic based, not physics based
  - Generate wave distribution in frequency domain, then perform inverse FFT
- Widely used in VFX, and games
  - In VFX the height map is large
    - Titanic, 2048x2048
    - Water World, 2048x2048



Ocean simulation – Implementation

- FFT on CPU becomes bottleneck when map is big
  - Large textures also take longer on CPU → GPU transfer
  - Large displacement map is required for detailed waves
- GPU is really good at FFT
  - Multiple 512x512 transforms can be performed in trivial time on high-end GPUs
- See CUDA SDK sample

- The ocean surface is composed by enormous simple waves
- Each sine wave is a hybrid sine wave (Gerstner wave)
  - A mass point on the surface is doing vertical circular motion



More potential applications...

- Model deformation (Deformers, Muscle simulations)
  - Inherently spatial algorithms
  - Mass-spring systems
  - Potential use of GLinterop
- Tracking
  - Image processing, optical flow, sorting, matrix solving
- Relighting
  - Ability to reproduce shaders, difficult to represent in GLSL
  - General raytracing is difficult to implement in GPGPU
  - OptiX...

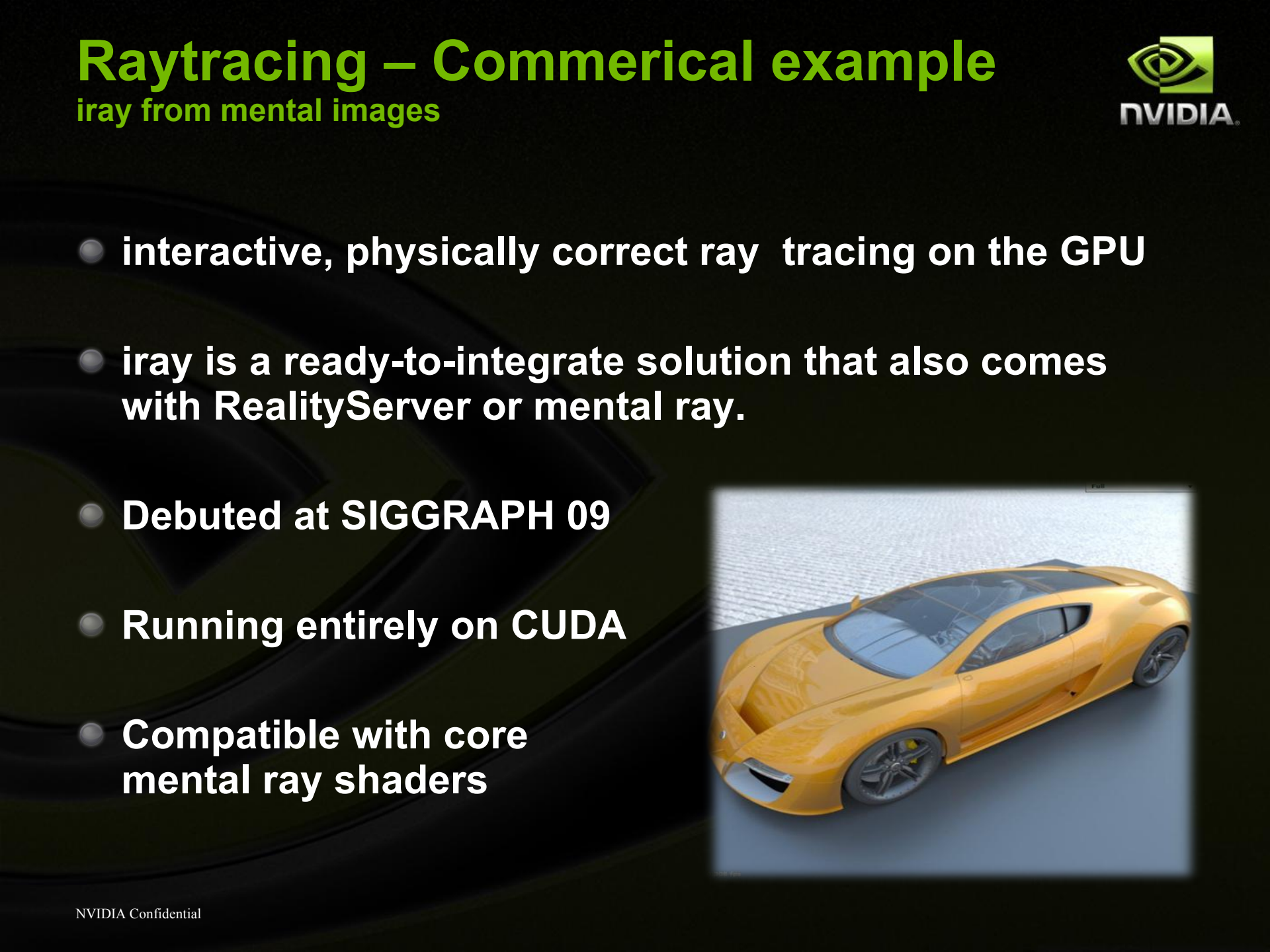
The OptiX ray tracing engine - overview

- A General Purpose Ray Tracing API
  - Rendering, baking, collision detection, A.I. queries, etc.
  - Modern shader-centric, stateless and bindless design
  - Not a renderer but can implement many types of renderers
- Highly Programmable
  - Shading with arbitrary ray payloads
  - Ray generation/framebuffer operations (cameras, data unpacking, etc.)
  - Programmable intersection (triangles, NURBS, implicit surfaces, etc.)
- Easy to Program
  - Write single ray code (no exposed ray packets)
  - No need to rewrite shaders to target different hardware

OptiX SDK Release

Raytracing – Commerical example

iray from mental images

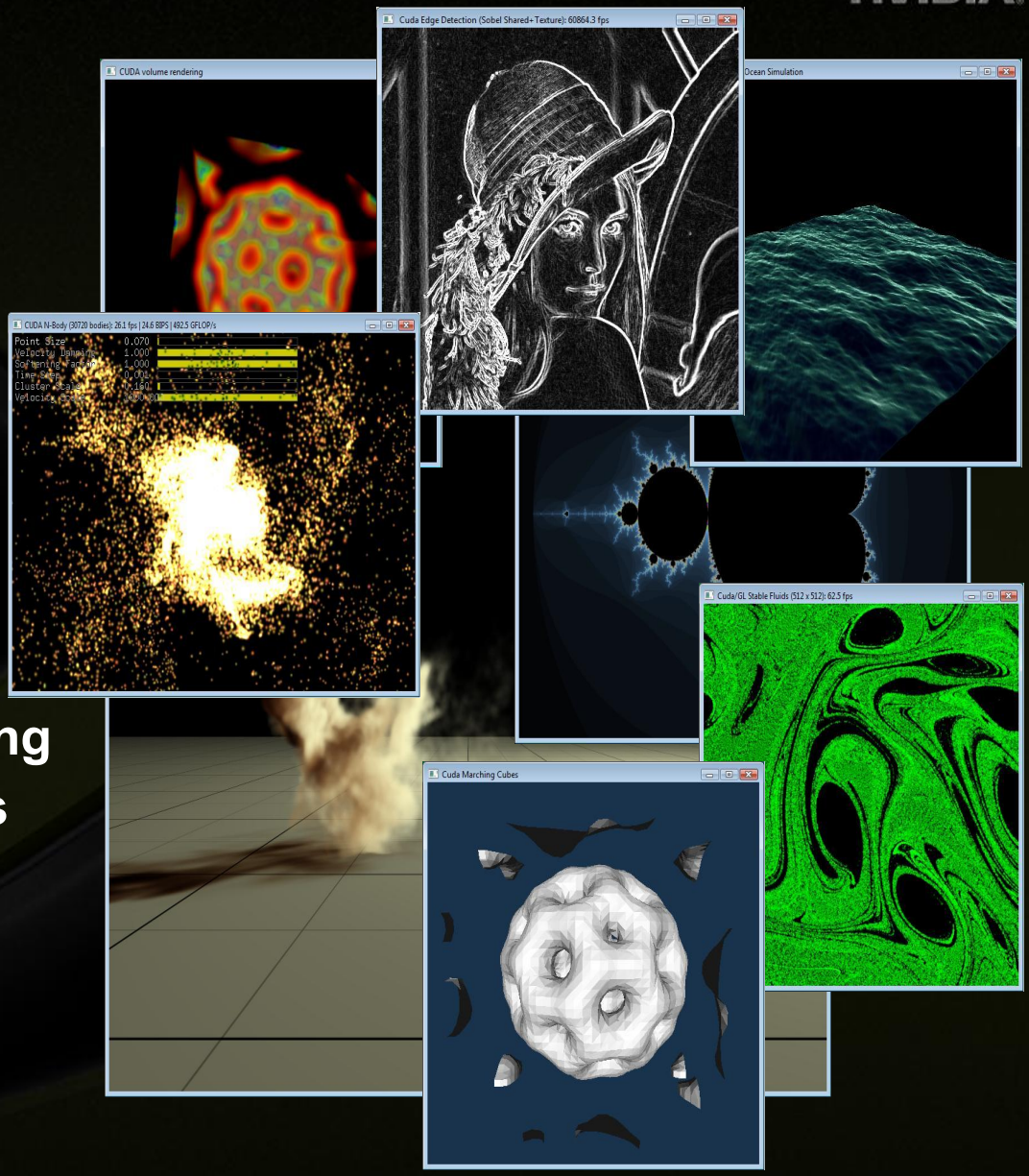


- interactive, physically correct ray tracing on the GPU
- iray is a ready-to-integrate solution that also comes with RealityServer or mental ray.
- Debuted at SIGGRAPH 09
- Running entirely on CUDA
- Compatible with core mental ray shaders

NVIDIA SDKs

Hundreds of code samples for CUDA-C, DirectCompute, and OpenCL

- Finance
- Oil & Gas
- Video/Image Processing
- 3D Volume Rendering
- Particle Simulations
- Fluid Simulations
- Math Functions



CUDA – Open Source Projects

- Thrust
  - Library of parallel algorithms with high-level STL-like interface
  - http://code.google.com/p/thrust
- OpenCurrent:
  - C++ library for solving PDE's over regular grids
  - http://code.google.com/p/opencurrent
- CUDPP
  - CUDA Parallel Primitives: Scan, Reduce, Sort, etc.
  - http://code.google.com/p/cudpp
- 200+ projects on Google Code & SourceForge
  - Search for CUDA, OpenCL, GPGPU

Language & APIs for GPU Computing

Solution	Approach
CUDA C	Language Integration Device-Level API
PGI Accelerator PGI CUDA Fortran	Auto Parallelizing Compiler Language Integration
CAPS HMPP	Auto Parallelizing Compiler
OpenCL	Device-Level API
DirectCompute	Device-Level API
PyCUDA	API Bindings
CUDA.NET, OpenCL.NET, jCUDA	API Bindings
…	…

OpenCL

- Cross-vendor open standard
  - Managed by the Khronos Group
- Low-level API for device management and launching kernels
  - Close-to-the-metal programming interface
  - JIT compilation of kernel programs
- C-based language for compute kernels
  - Kernels must be optimized for each processor architecture
- NVIDIA released the first OpenCL v1.0 conformant driver for Windows and Linux to thousands of developers in June 2009

CUDA-C or OpenCL – It's your choice

- OpenCL is another API for the CUDA architecture
  - NVIDIA is leading the pack with its OpenCL implementation
  - Code can be ported relatively easily
    - "CUDA-C Driver API" and "OpenCL API" are very similar
- Things to consider:
  - CUDA delivers new hardware features faster than OpenCL
  - CUDA is quite mature, but openCL is still in its infancy
  - CUDA-C code is usually smaller than OpenCL code
  - CUDA-C offers some C++ functionality
    - OpenCL only uses a C99-based language (similar to Cg or HLSL)
  - Most 1st and 3rd party libraries are currently for CUDA-C only

OpenCL Performance Caveat

- OpenCL is about performance
  - Gives developers access to the massive compute power of parallel processors like GPUs
- But performance is not usually portable across devices:
  - There are multiple ways of implementing a given algorithm in OpenCL and each implementation can have vastly different performance characteristics for a given compute device
- Achieving good performance on GPUs requires a basic understanding of GPU architecture

CUDA tips

Whichever API you decide to use...

- Simplify CUDA coding further
  - Use language abstractions, e.g. C++ classes
    - Object cleanup tied to lifetime of objects.
    - Makes it much easier to write error-free and leak-free code
  - Try automatic error checking with exceptions
- Use the development tools
  - profiler, debugger
- Use CUDA libraries where possible
  - take advantage of free feature improvement and knowledge of the driver team

CUDA tips

- Parameterization
  - Makes it easier to fine tune
  - Helps adaptation to different GPUs
  - GPUs vary in many ways: # of multiprocessors, Memory bandwidth, Shared memory size, Register file size, Max. threads per block
- You can even make apps self-tuning (like FFTW and ATLAS)
  - “Experiment” mode discovers and saves optimal configuration

Can GPUs do real IEEE FP?

- G8x GPU floating-point is IEEE 754
  - Comparable to other processors / accelerators.
  - More precise / usable in some ways.
  - Less precise in other ways.
- GPU FP is getting better every generation.
  - IEEE compliant double-precision support in modern cards.

G8x Deviations from IEEE-754

- Addition and Multiplication are IEEE compliant
  - Maximum 0.5 ulp error
- However, often combined into multiply-add (FMAD)
  - Intermediate result is truncated

- Division is non-compliant (2 ulp)
- Not all rounding modes are supported
- Denormalized numbers are not supported
- No mechanism to detect floating-point exceptions
  - behaves as if the exception is always masked
- See the documentation for more info

GPU results may not match CPU

- Many variables
  - hardware, compiler, optimization settings
- CPU operations aren't strictly limited to 0.5 ulp
  - Sequences of operations can be more accurate due to 80-bit extended precision ALUs
  - Even in reduced precision mode, x86 uses 15-bit exponents internally (instead of 8 or 10-bit)
- CPUs typically support denormalized numbers
- Check out:
  - http://math.nist.gov/javanumerics/reports/jgfnwg-01.html
  - Search for: "What every computer scientist should know about Floating-Point arithmetic"
- Fixed point? :-)

x86 80-bit Extended double precision

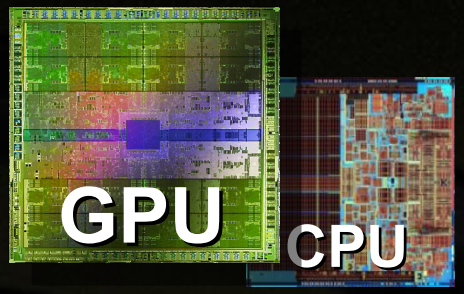
- Programmers can try the following methods:
  - Compiler options:
    - gcc : `-ffloat_store`
  - Force single precision on a portion of the code with:
    - `_FPU_GETCW(...)` and `_FPU_SETCW(...)`

```
unsigned int originalCW;
_FPU_GETCW(originalCW);
unsigned int cw = (originalCW & ~0x300) | 0x000;
_FPU_SETCW(cw);

// do 24bit mantissa coding here...

_FPU_SETCW(originalCW);
```

FP Math is Not Associative!

- In symbolic math
  - (x + y) + z == x + (y + z)
- This is not necessarily true in floating-point
  - Try ($x = 10^{30}$, $y = -10^{30}$, $z = 1$) in the above equation
- When you parallelize computations, you potentially change the order of operations
  - Parallel results may not exactly match sequential results
- This is not specific to GPUs or CUDA.
  It is an inherent part of parallel execution

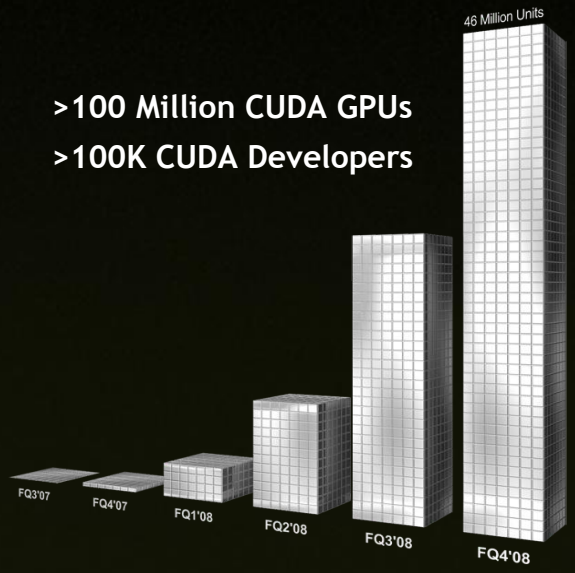
Questions?



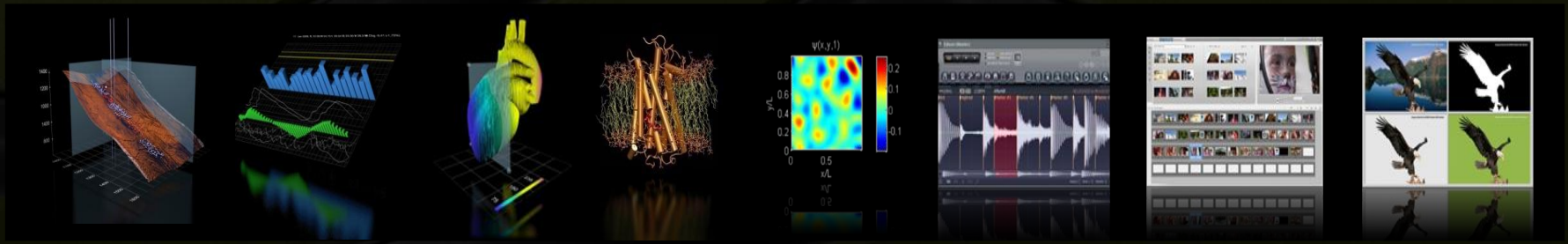
Heterogeneous Computing

CUDA

>100 Million CUDA GPUs
>100K CUDA Developers



www.nvidia.com/CUDA



Oil & Gas | Finance | Medical | Biophysics | Numerics | Audio | Video | Imaging