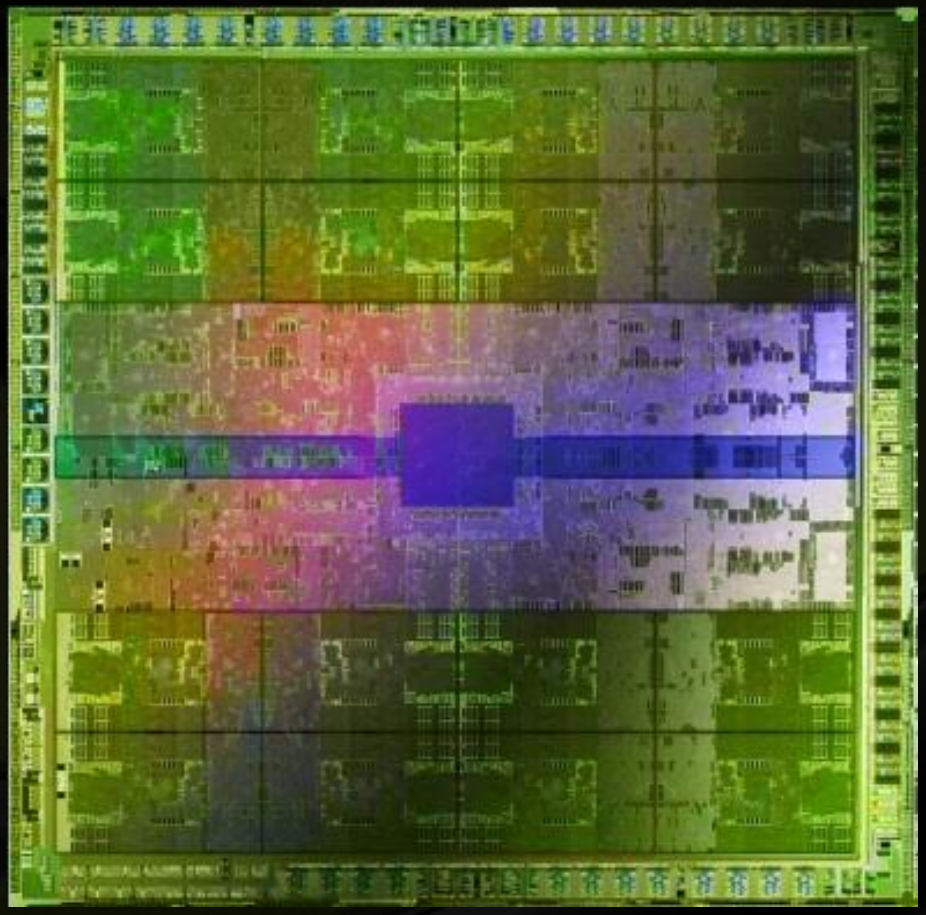
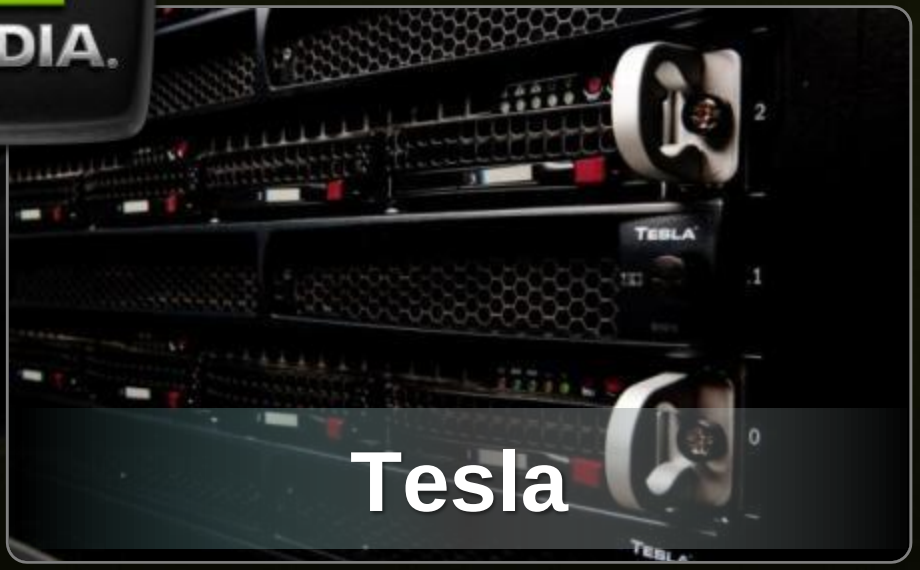
GPU Computing
Past, Present, and Future
David Luebke
NVIDIA

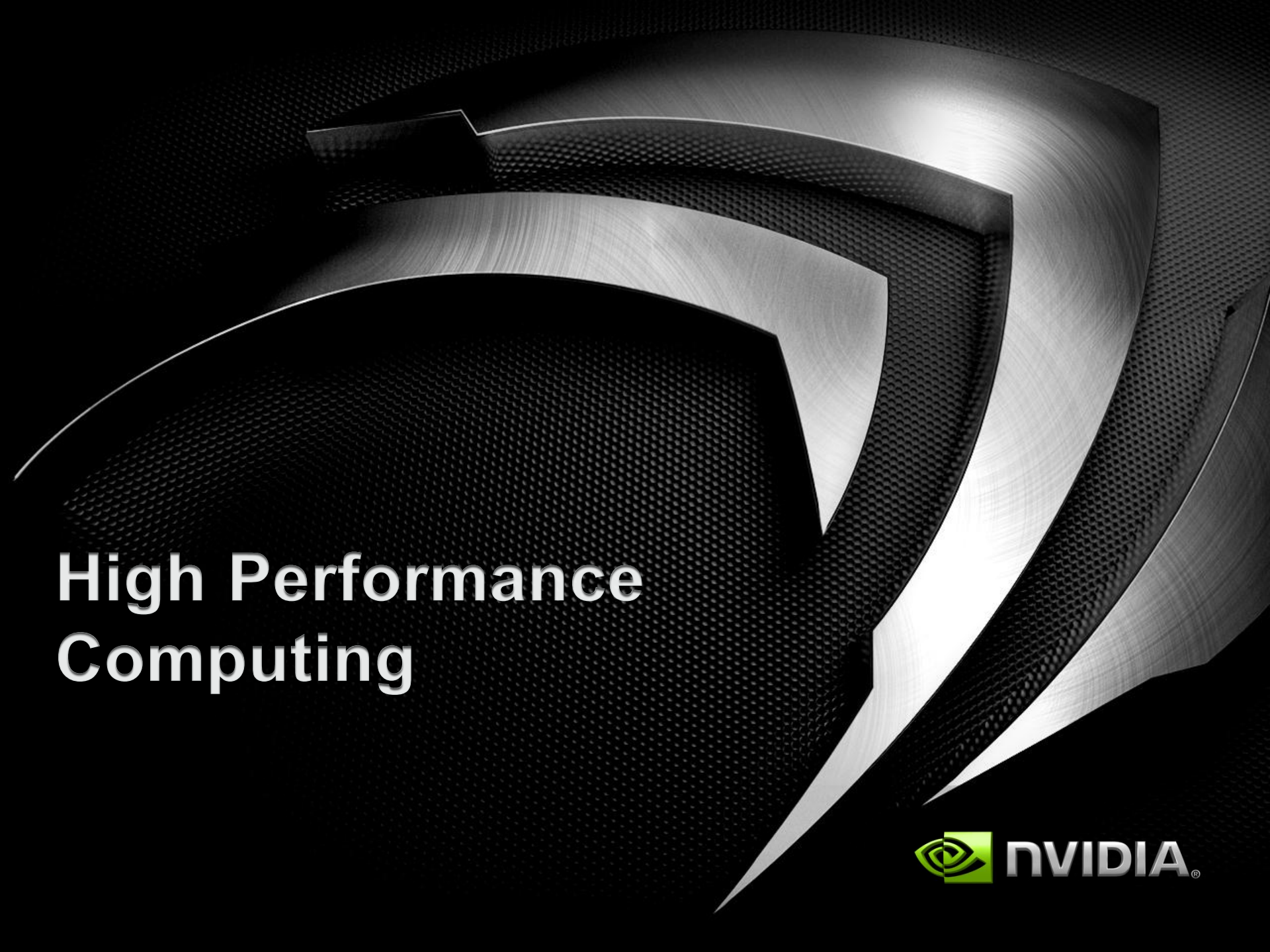
Graphics Processing Unit (GPU)



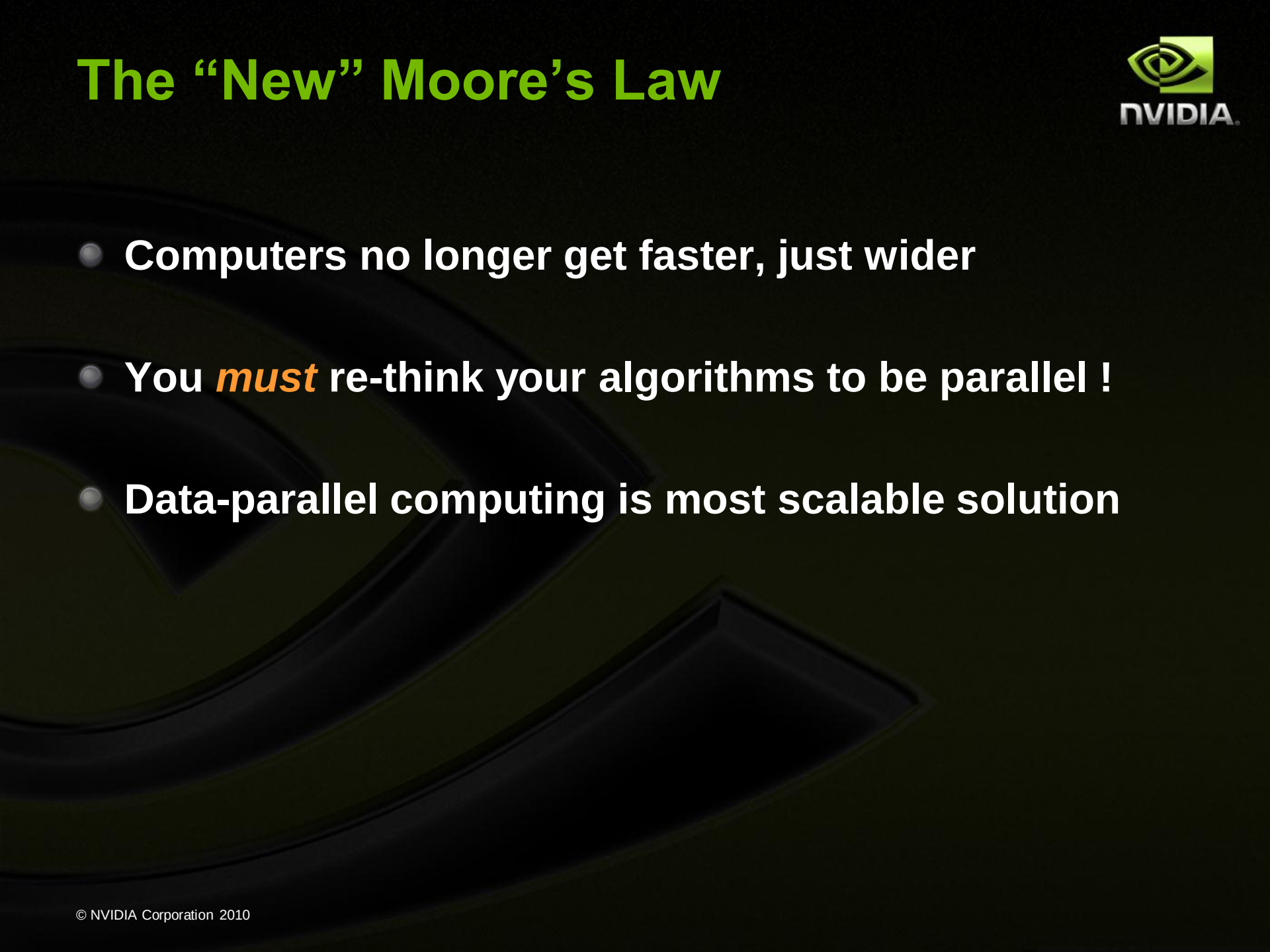
© NVIDIA Corporation 2010

What GPUs Do




High Performance
Computing
NVIDIA

The “New” Moore’s Law

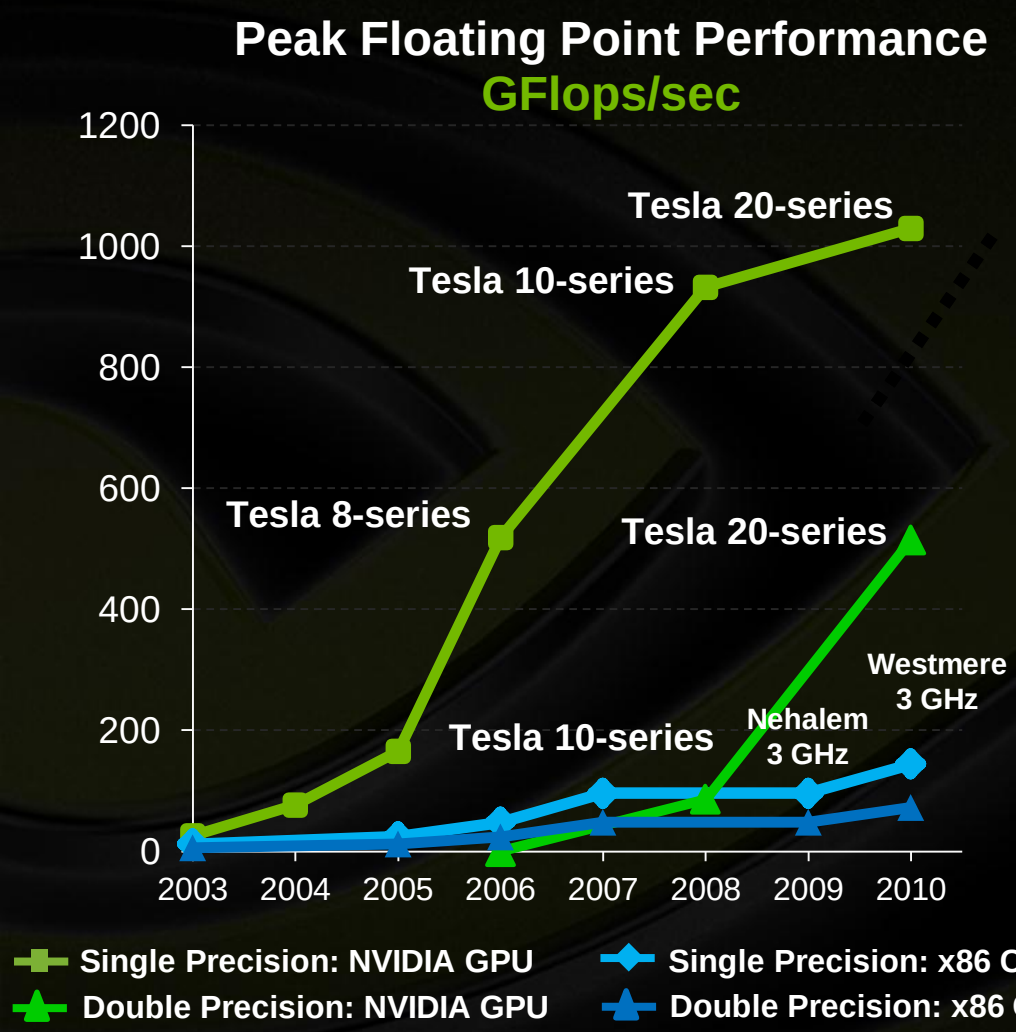


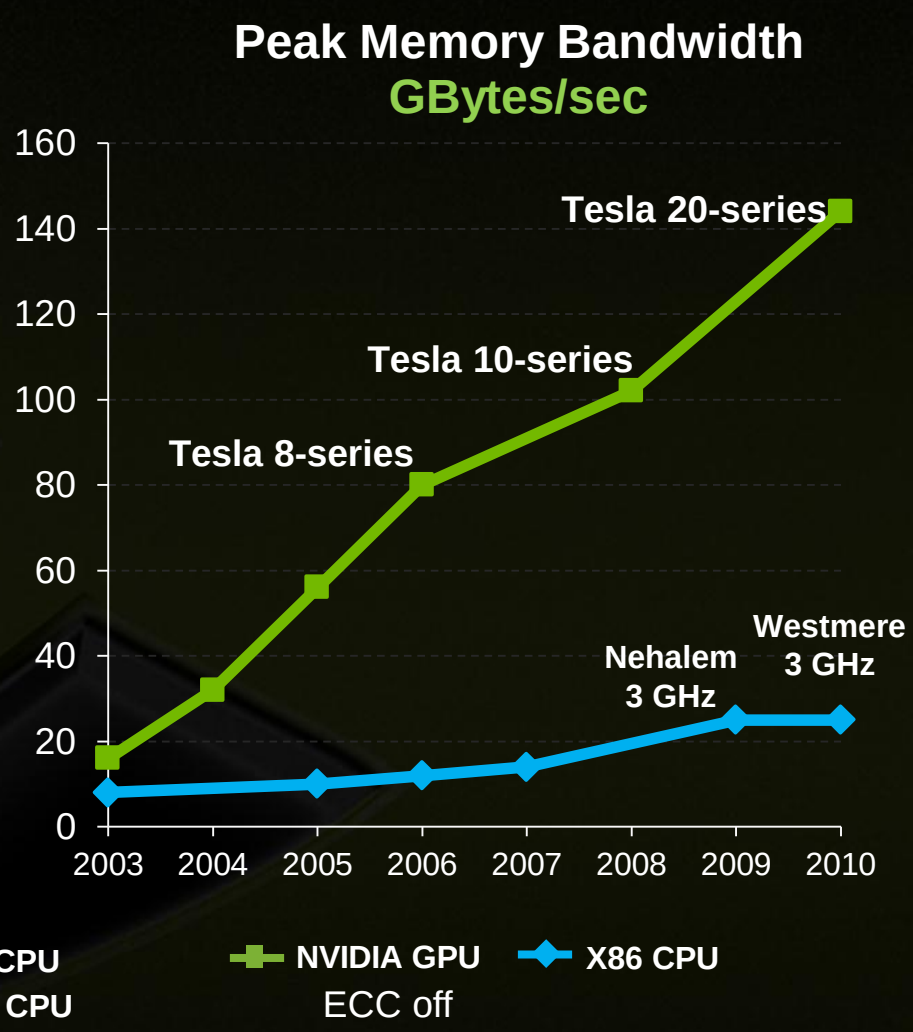
- Computers no longer get faster, just wider
- You *must* re-think your algorithms to be parallel !
- Data-parallel computing is most scalable solution

© NVIDIA Corporation 2010

The World's Programmers

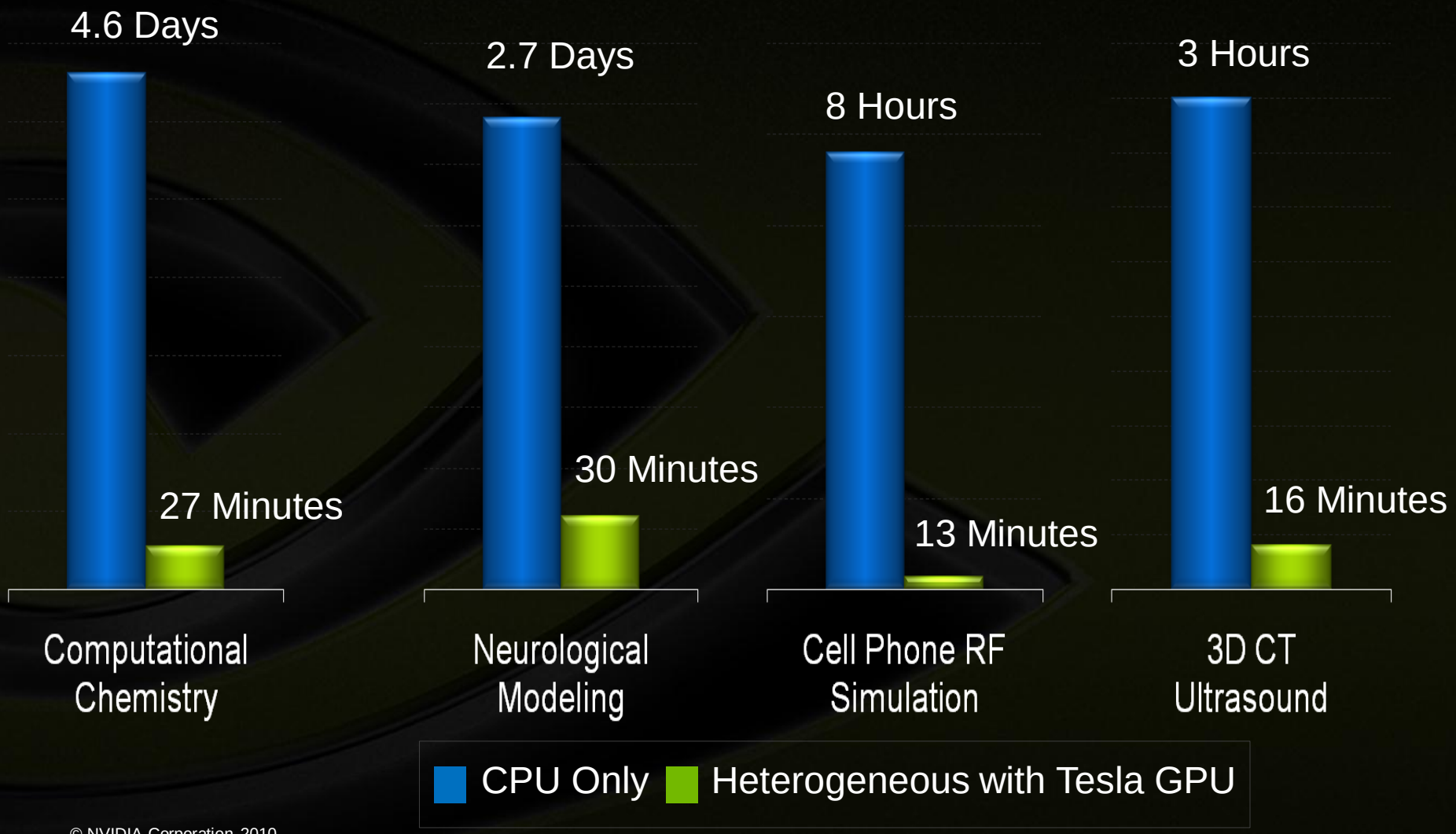
Why GPU Computing?





© NVIDIA Corporation 2010

Accelerating Insight



© NVIDIA Corporation 2010

GPU Computing
at the Tipping Point
A sampling of “killer” apps
NVIDIA

Detecting IEDs

12 mph
CPU

77 mph
GPU

© NVIDIA Corporation 2010

Reducing Radiation from CT Scans

28,000 people/year develop cancer from CT scans

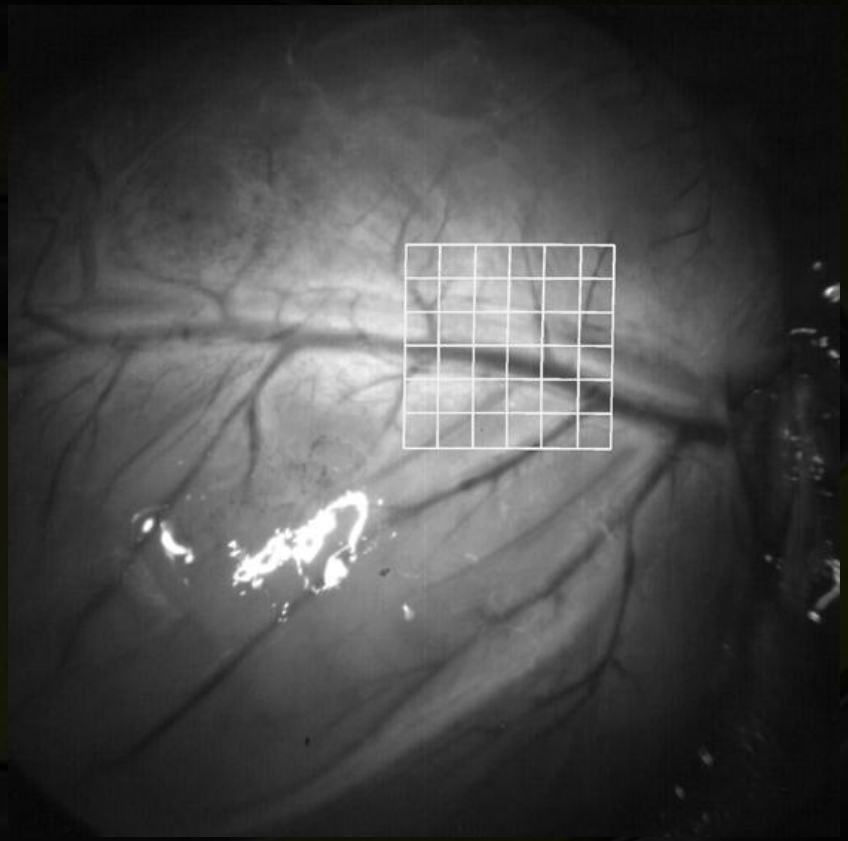
UCSD: advanced CT reconstruction reduces radiation by **35-70x**

CPUs: 2 hours (unusable)

CUDA: 2 minutes (clinically practical)

© NVIDIA Corporation 2010

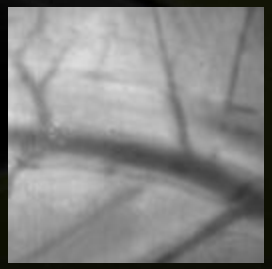
Surgery on a Beating Heart



Only 2% of surgeons can operate on a beating heart

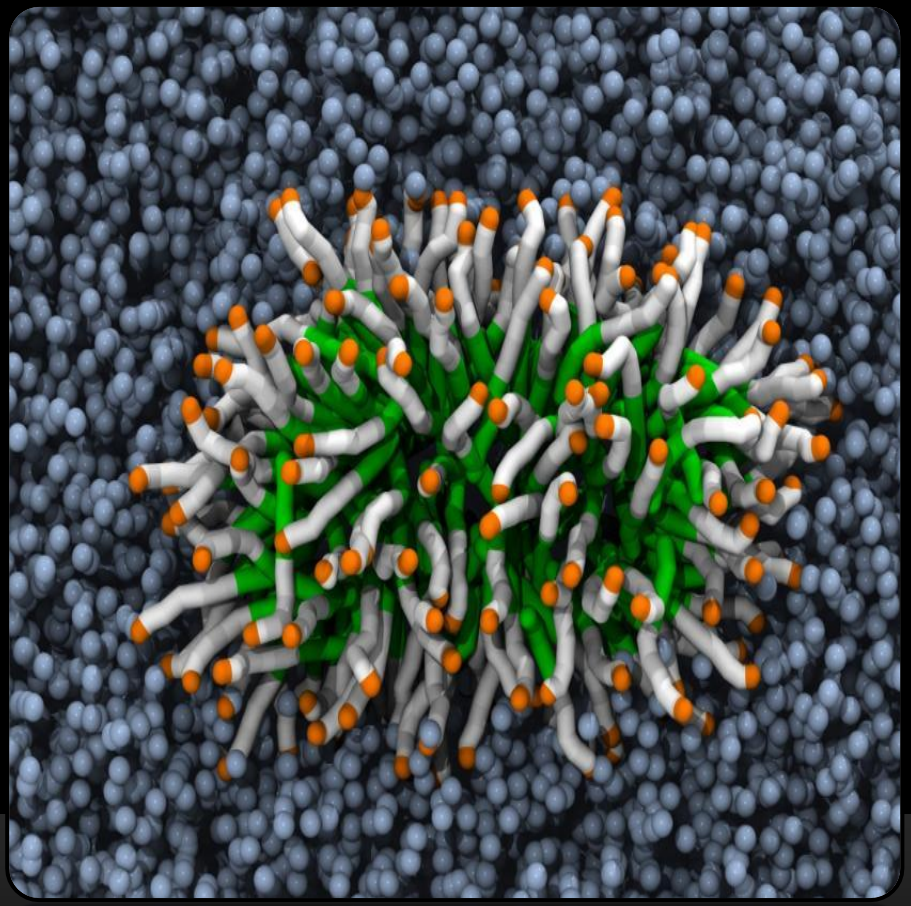
Patient stands to lose 1 point of IQ every10 min with heart stopped

GPU enables real-time motion compensation to virtually stop beating heart for surgeons:



Courtesy Laboratoire d'Informatique de Robotique et de Microelectronique de Montpellier

Simulating Shampoo



Surfactant Simulation

"The outcome is quite spectacular…with two GPUs we can run a single simulation as fast as on 128 CPUs of a Cray XT3 or on 1024 CPUs of an IBM BlueGene/L machine."

"We can try things that were undoable before. It still blows my mind."

Axel Kohlmeyer
Temple University

Cotton

Detect contaminants in cotton lint

Cotton is over cleaned

Results in Fiber Damage

GPUs enables real-time feedback while cleaning cotton

96% lower fiber damage

$100M additional potential revenue

"Parallel Algorithm for GPU Processing; for use in High Speed Machine Vision Sensing of Cotton Lint Trash", Mathew G. Pelletier, February 8, 2008

Tianhe-1A

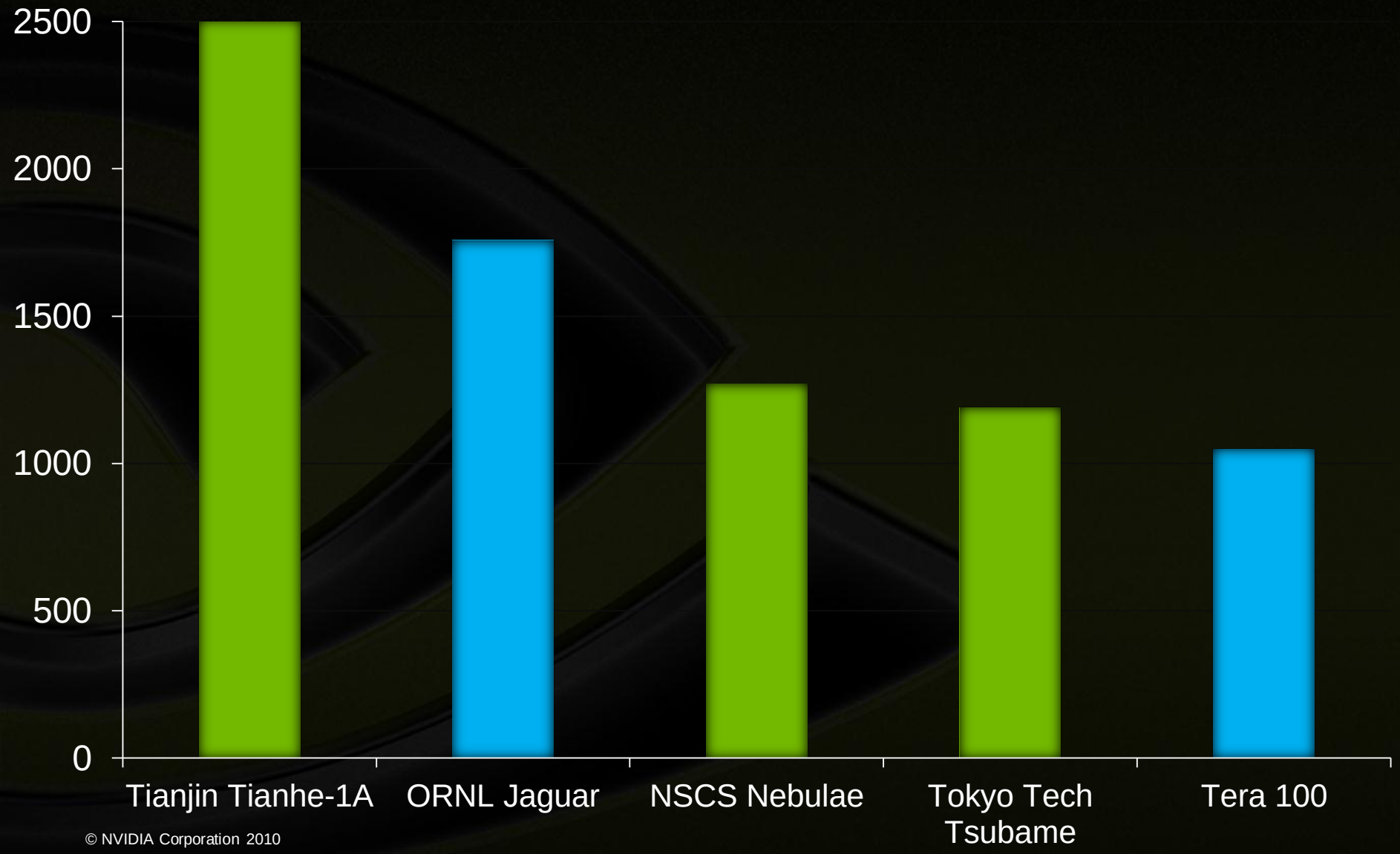
Fastest Supercomputer in the World

2.507 PetaFLOP/s

7,168 Tesla GPUs

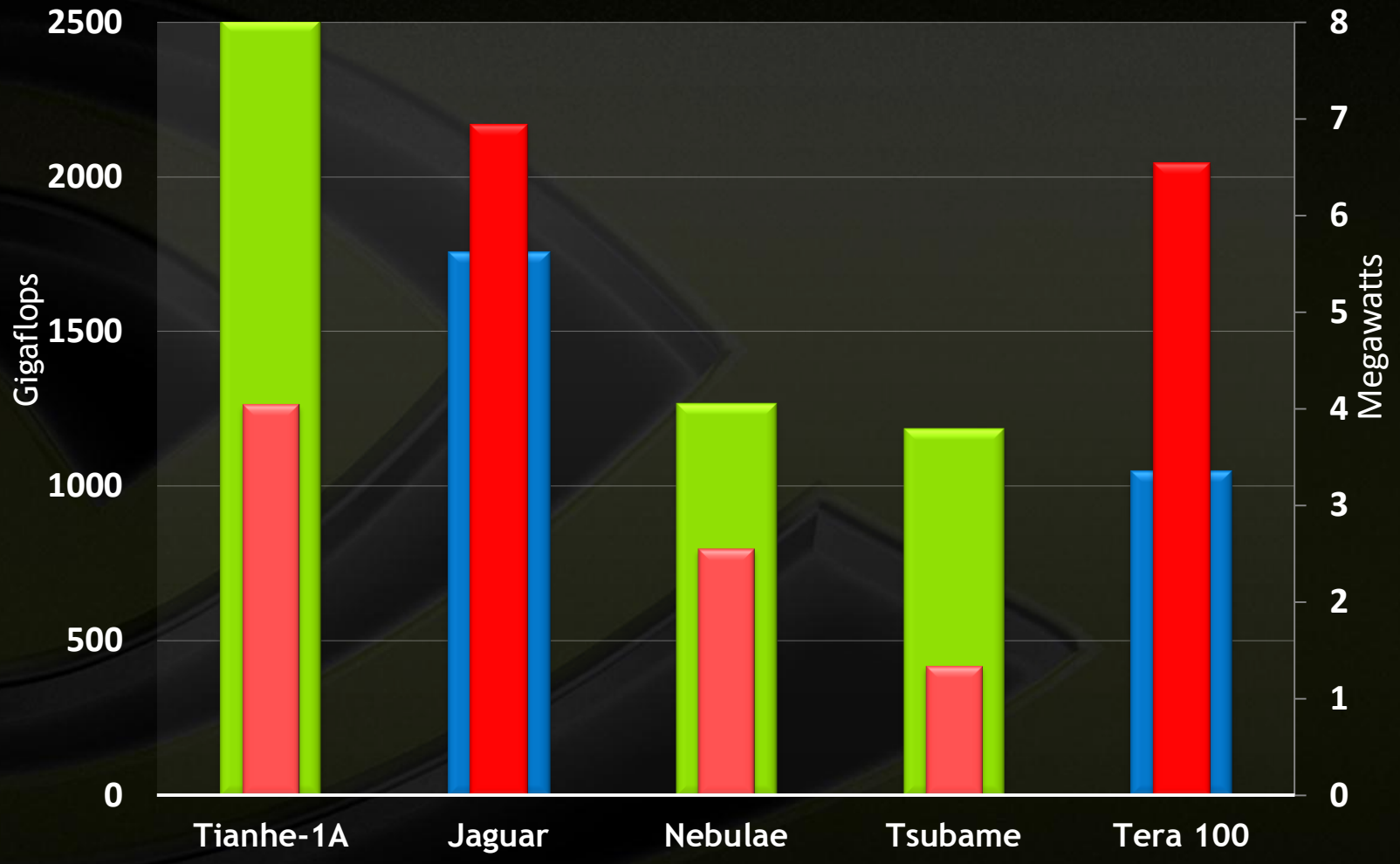
© NVIDIA Corporation 2010

3 of Top5 are Tesla GPU Supercomputers



© NVIDIA Corporation 2010

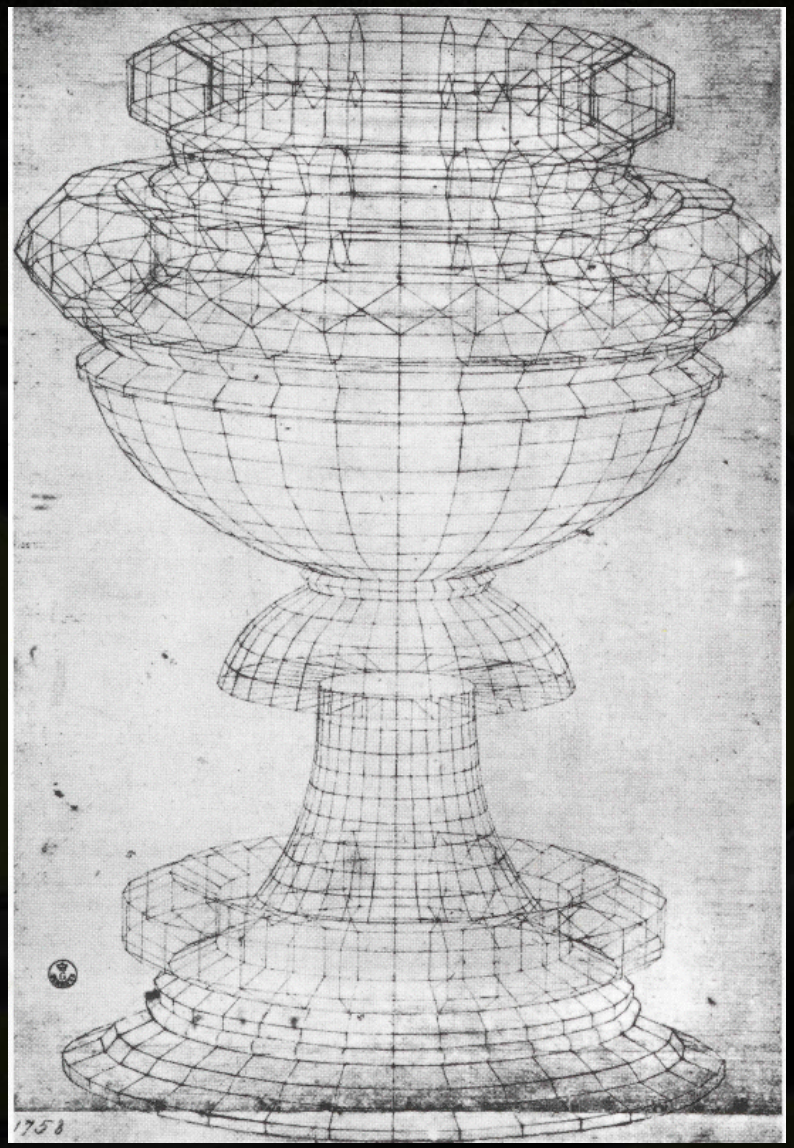
And More Efficient



© NVIDIA Corporation 2010

Heritage:
Graphics Hardware
NVIDIA

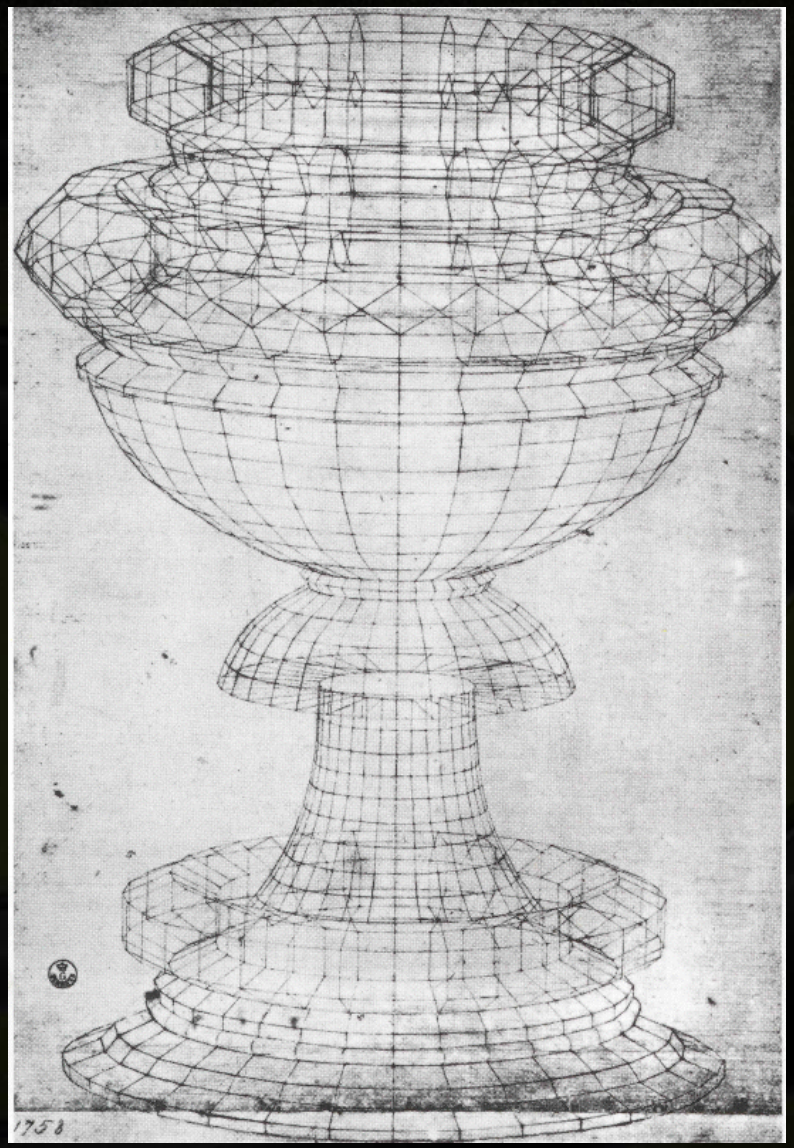
Early 3D Graphics

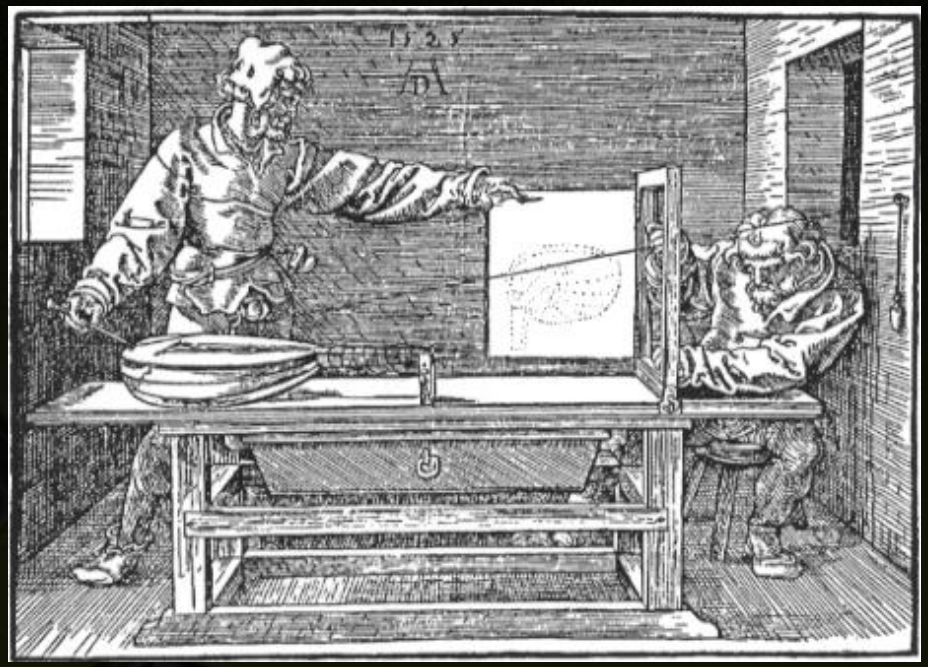


Perspective study of a chalice
Paolo Uccello, circa 1450

© NVIDIA Corporation 2010

Early Graphics Hardware





Artist using a perspective machine
Albrecht Dürer, 1525

Perspective study of a chalice
Paolo Uccello, circa 1450

© NVIDIA Corporation 2010

Early *Electronic* Graphics Hardware

SKETCHPAD: A Man-Machine Graphical Communication System
Ivan Sutherland, 1963

© NVIDIA Corporation 2010

The Graphics Pipeline

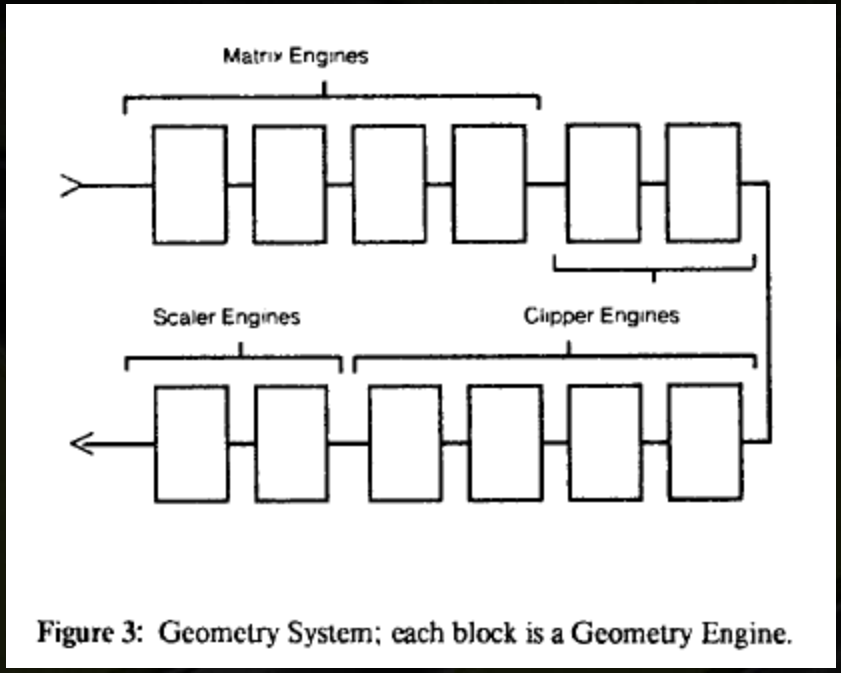


Figure 3: Geometry System; each block is a Geometry Engine.

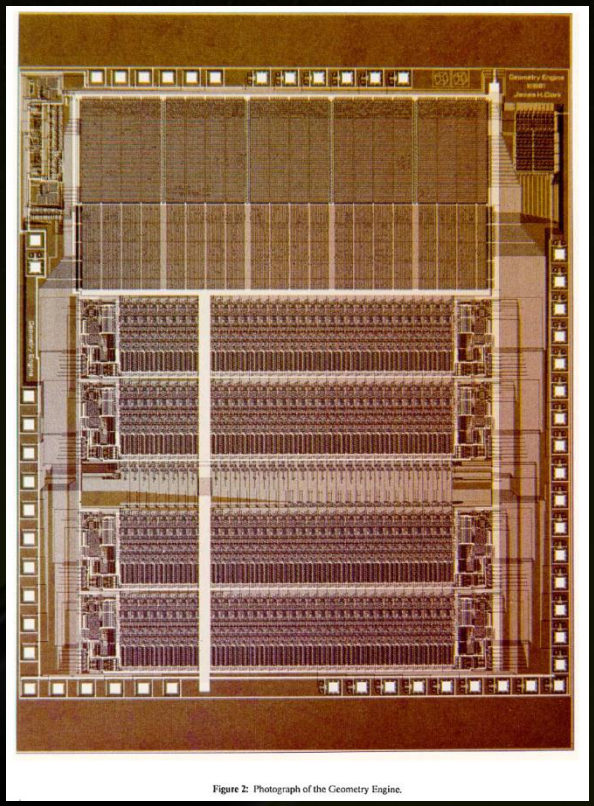
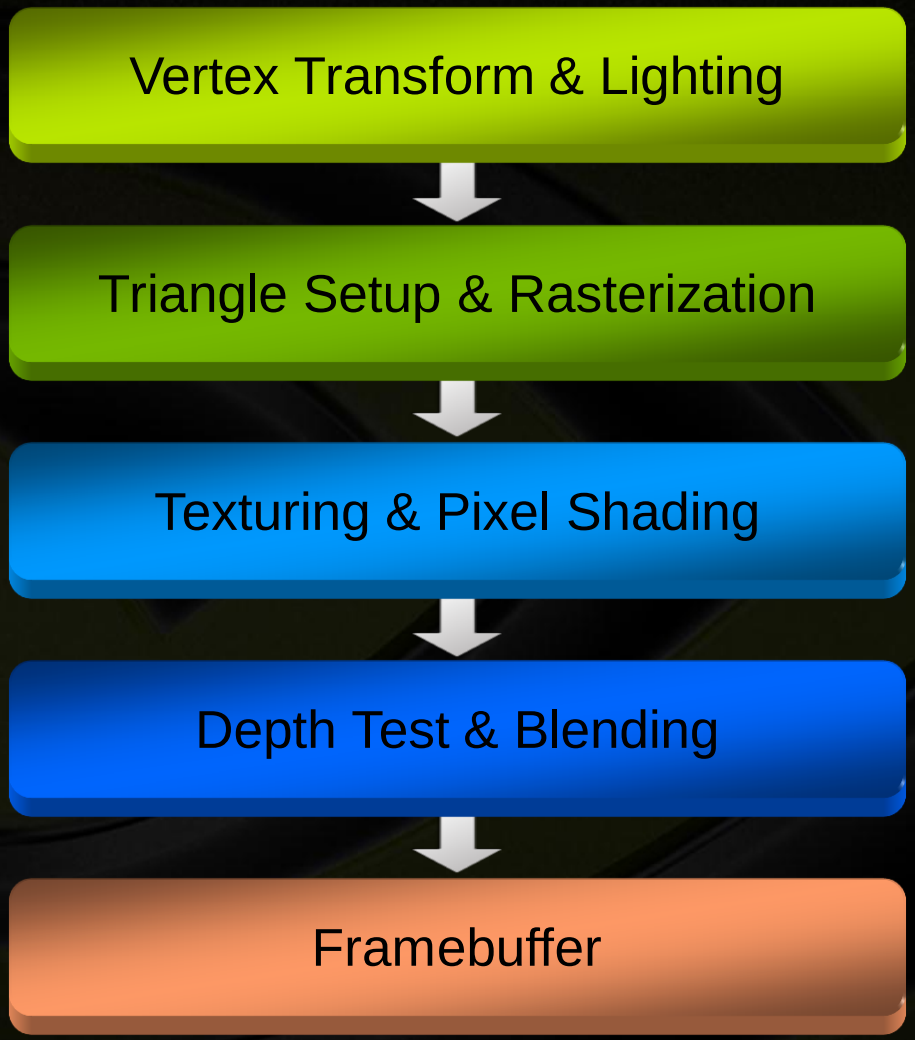


Figure 2: Photograph of the Geometry Engine.

The Geometry Engine: A VLSI Geometry System for Graphics
Jim Clark, 1982

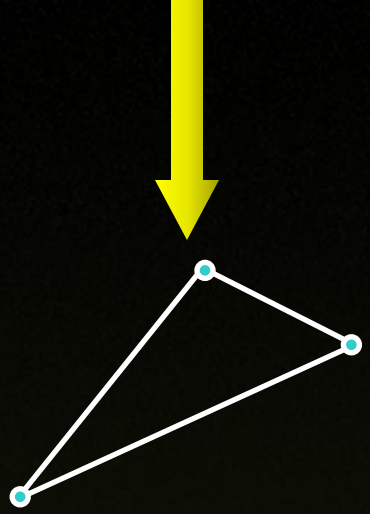
© NVIDIA Corporation 2010

The Graphics Pipeline



© NVIDIA Corporation 2010

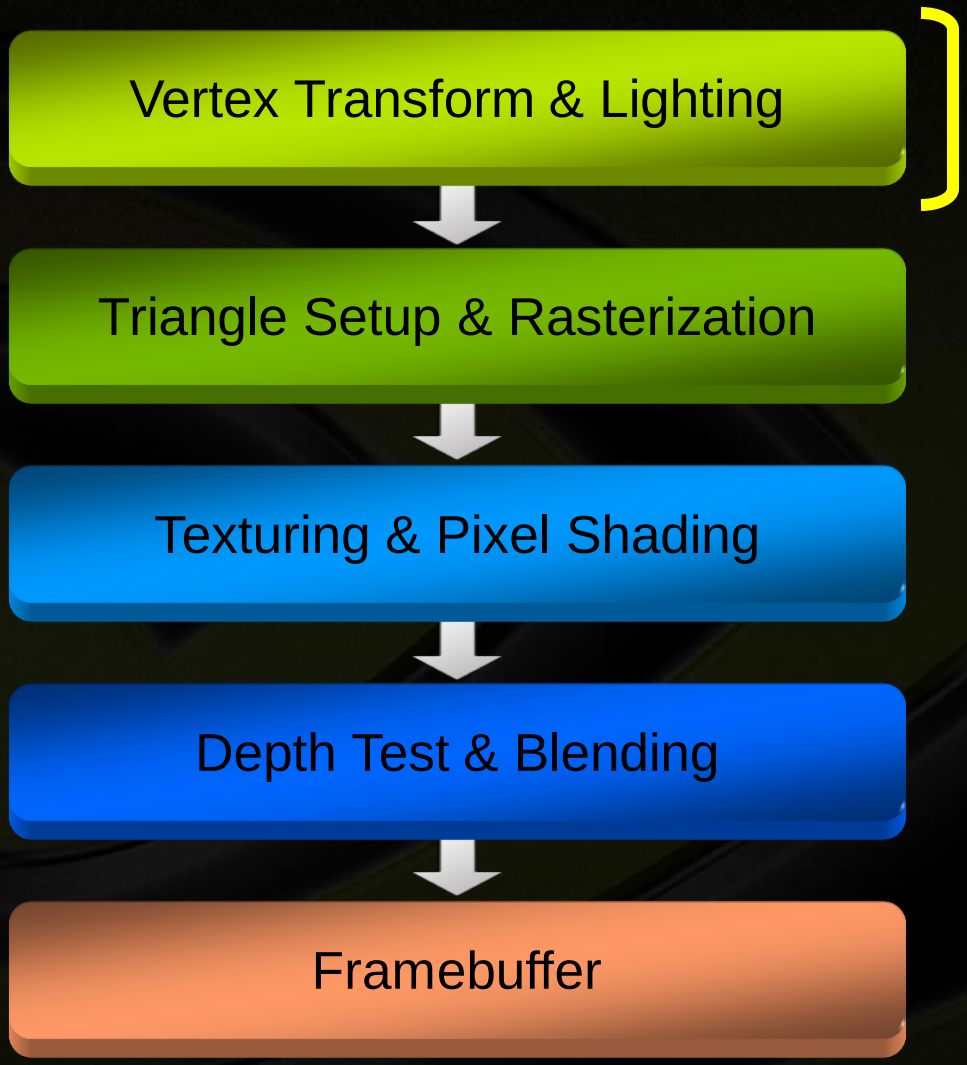
The Graphics Pipeline

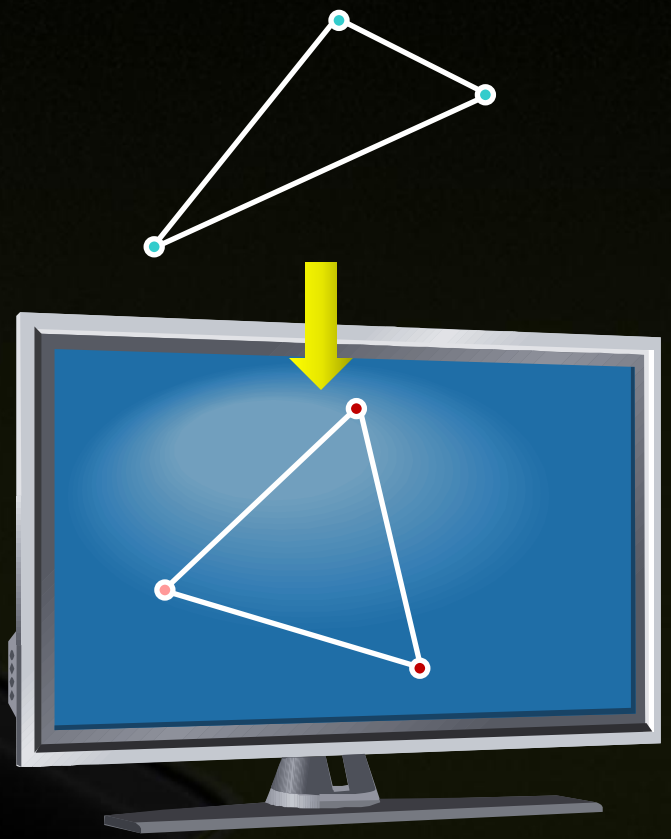


- Vertex Transform & Lighting
- Triangle Setup & Rasterization
- Texturing & Pixel Shading
- Depth Test & Blending
- Framebuffer

© NVIDIA Corporation 2010

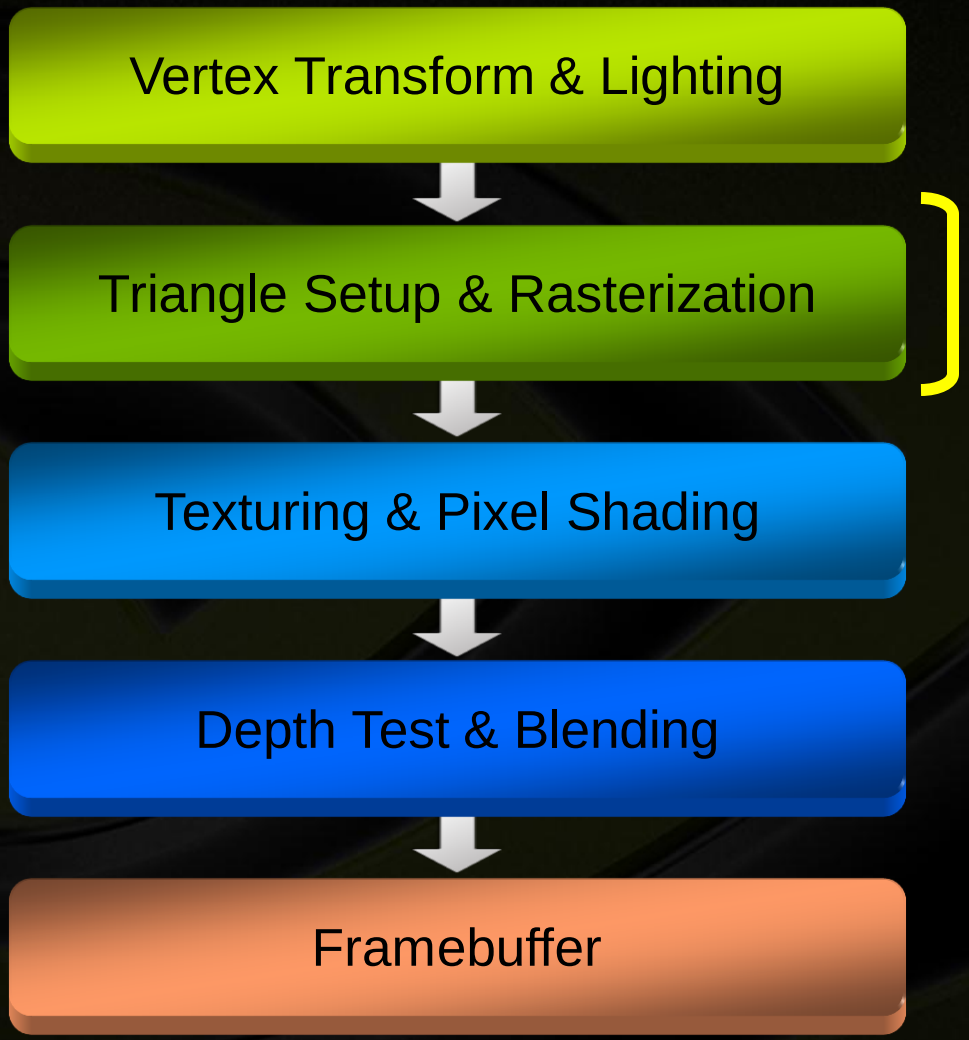
The Graphics Pipeline

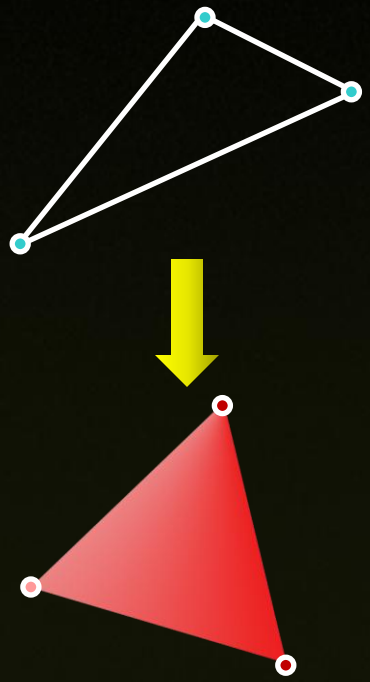




© NVIDIA Corporation 2010

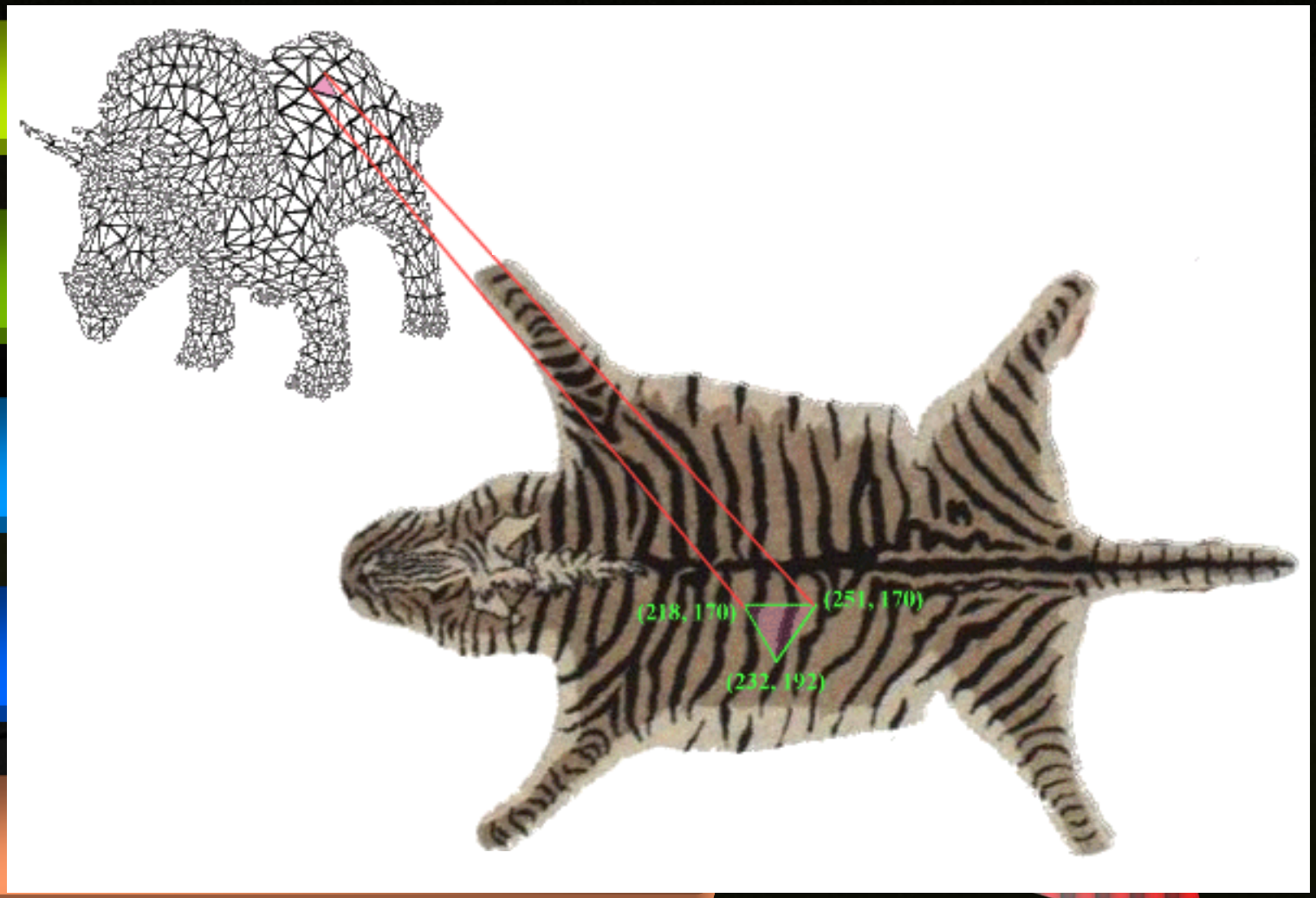
The Graphics Pipeline





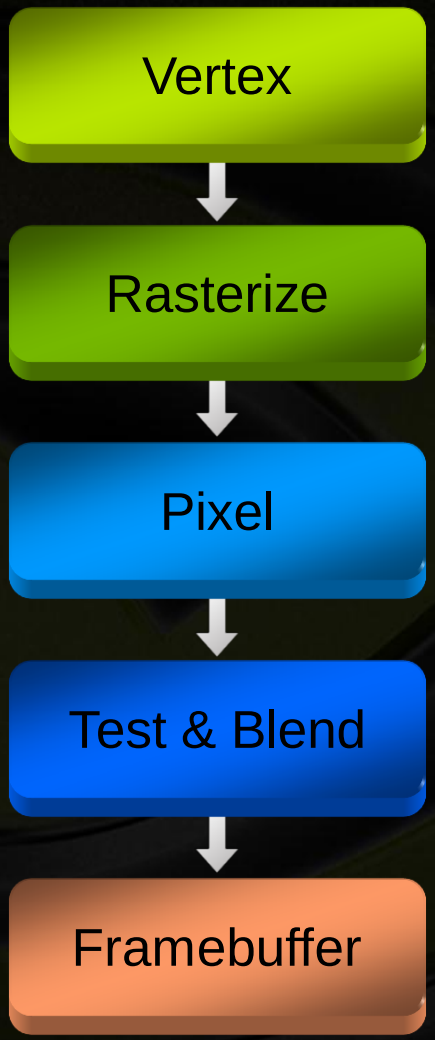
© NVIDIA Corporation 2010

The Graphics Pipeline



© NVIDIA Corporation 2010

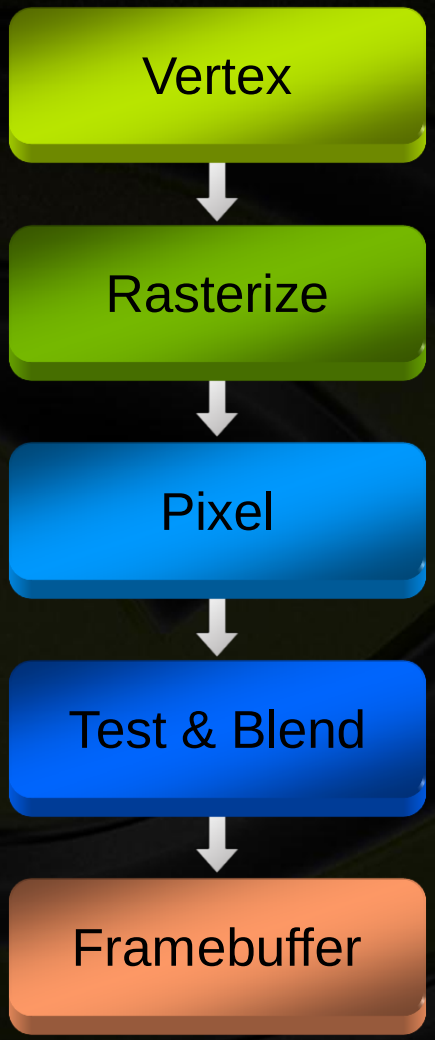
The Graphics Pipeline

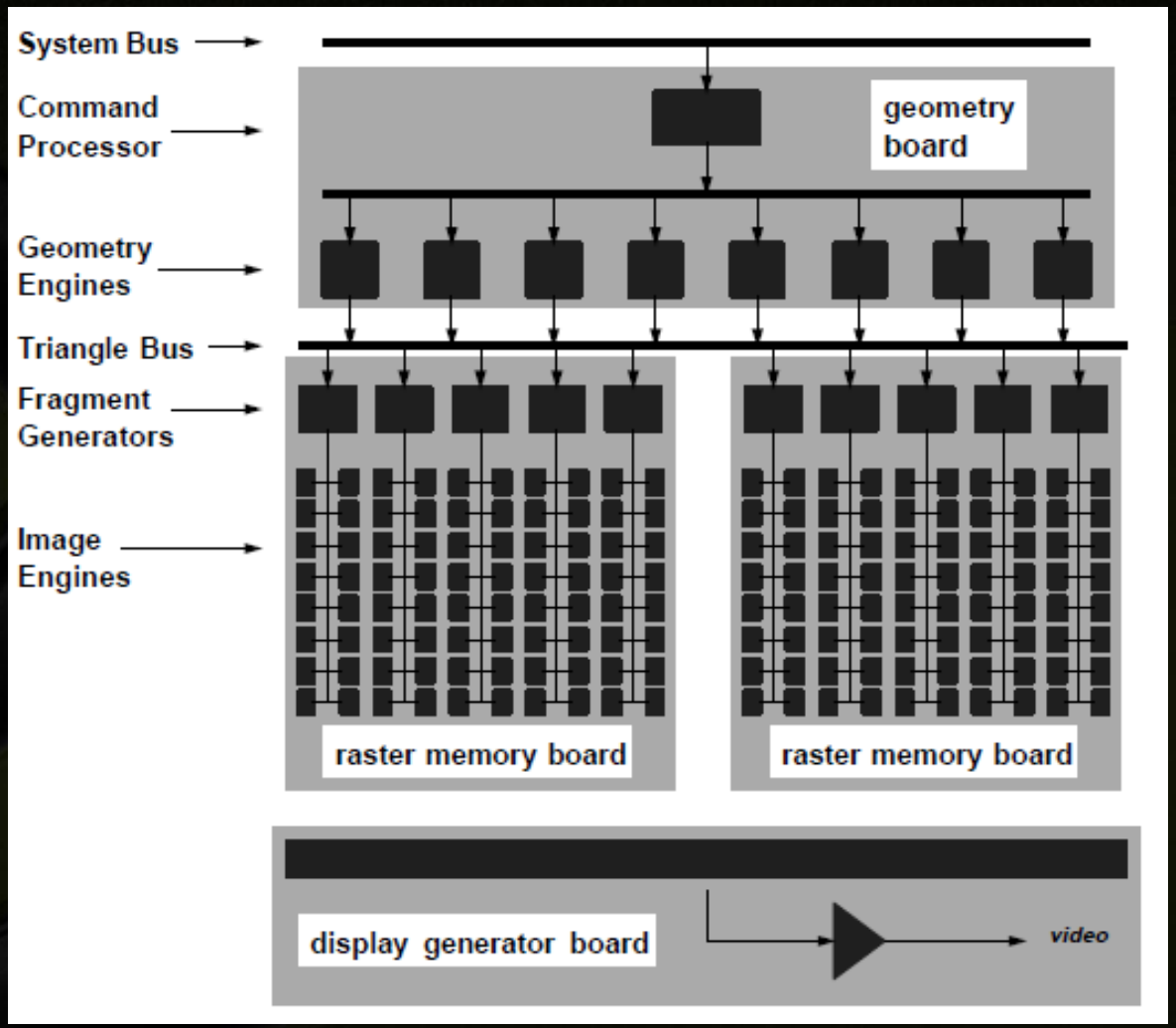


- Key abstraction of real-time graphics
- Hardware used to look like this
- One chip/board per stage
- Fixed data flow through pipeline

© NVIDIA Corporation 2010

SGI RealityEngine (1993)

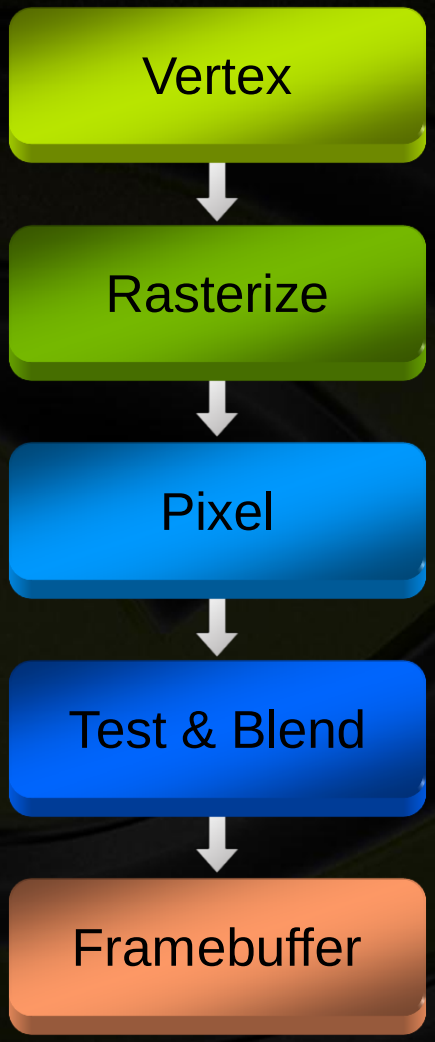


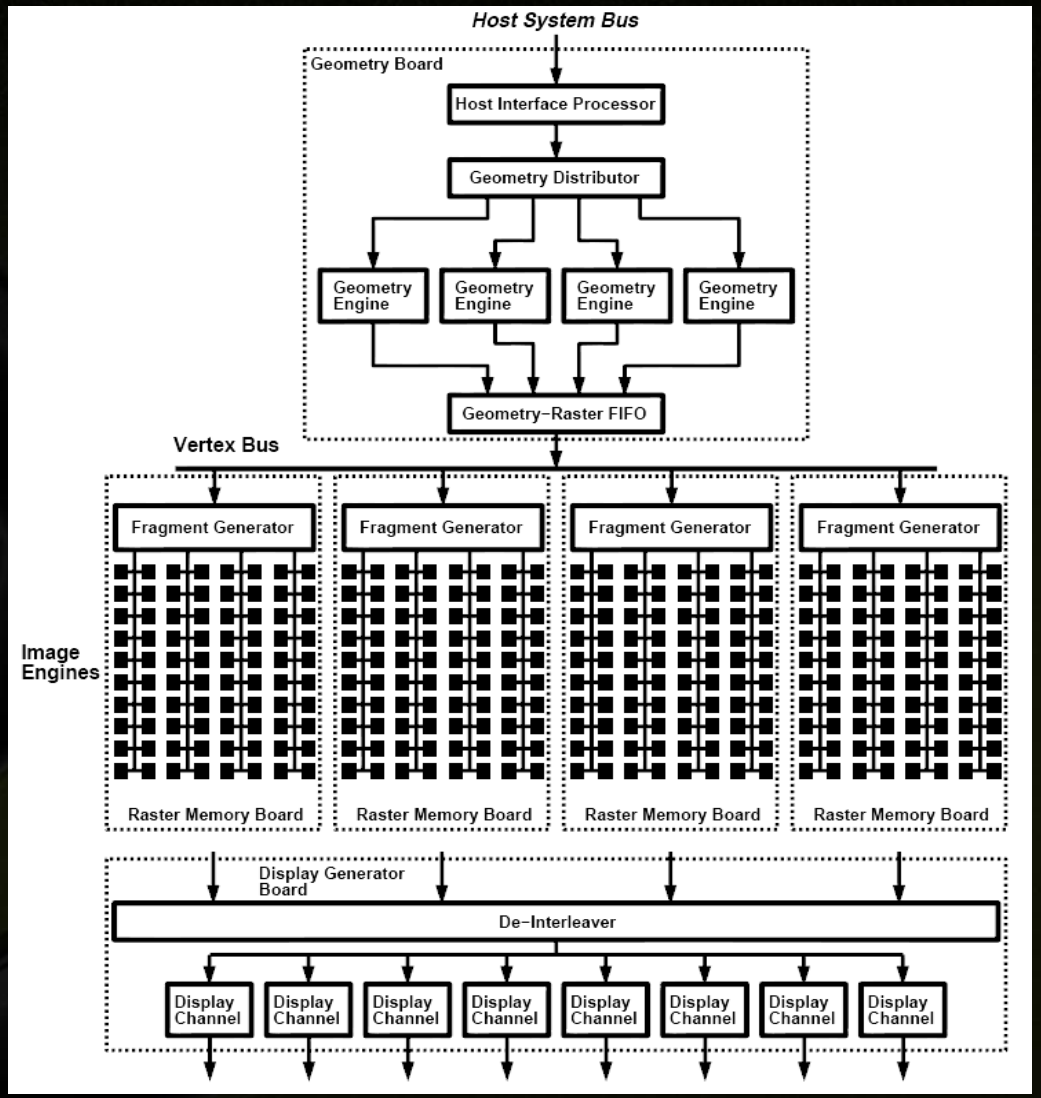


RealityEngine Graphics
Kurt Akeley , SIGGRAPH 93

© NVIDIA Corporation 2010

SGI InfiniteReality (1997)

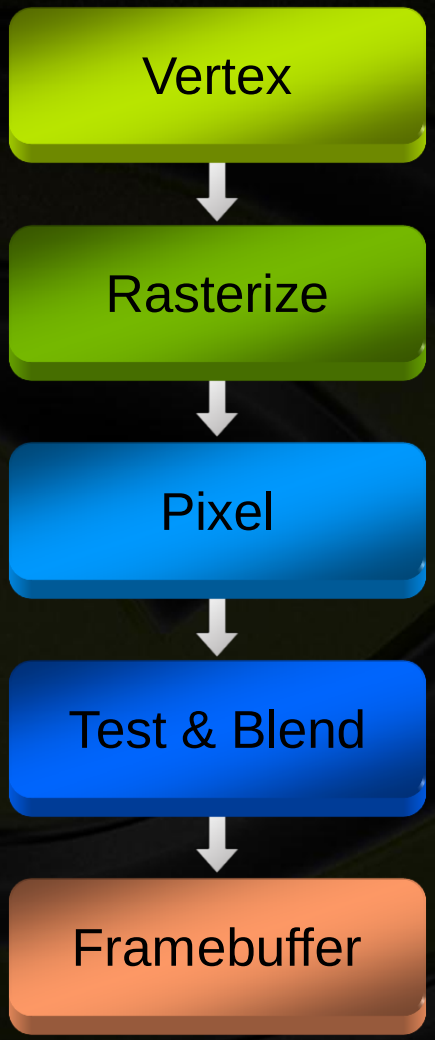




InfiniteReality: A real-time graphics system
Montrym et al., SIGGRAPH 97

© NVIDIA Corporation 2010

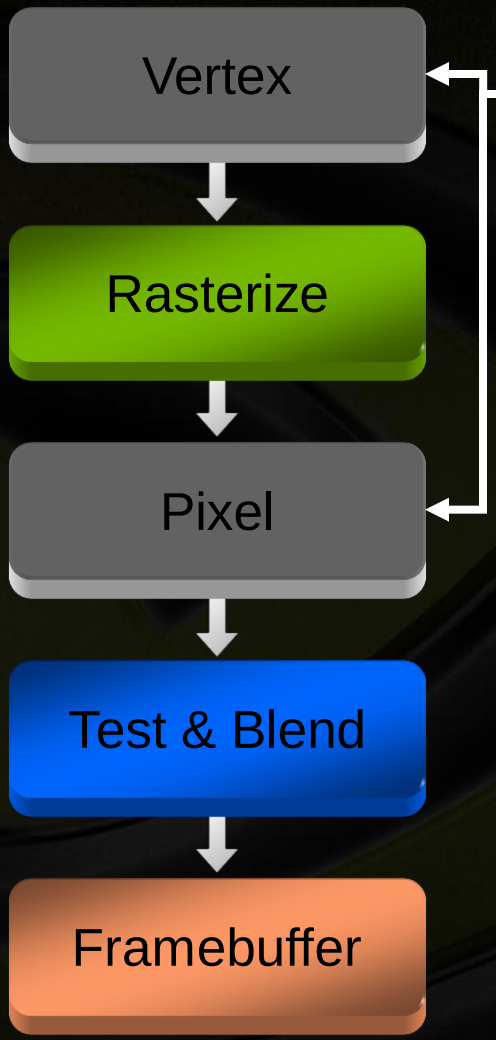
The Graphics Pipeline



- Remains a useful abstraction
- Hardware used to look like this

© NVIDIA Corporation 2010

The Graphics Pipeline

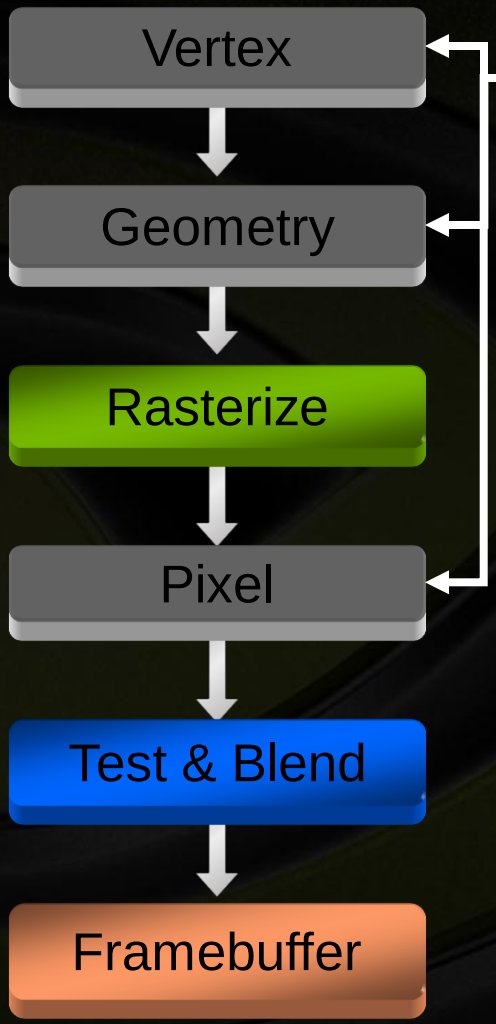



```
pixel_out
main(uniform sampler2D texture : TEXUNIT 0, pixel_in
{
  pixel_out OUT;
  float d= clamp(1.0 - pow(dot(IN.lightdist, IN.light
  float3 color = tex2D(texture, IN.texcoord).rgb;
  OUT.color = color * (d + 0.4);
  return OUT;
}
```

- **Hardware used to look like this:**
  - **Vertex, pixel processing became programmable**

© NVIDIA Corporation 2010

The Graphics Pipeline

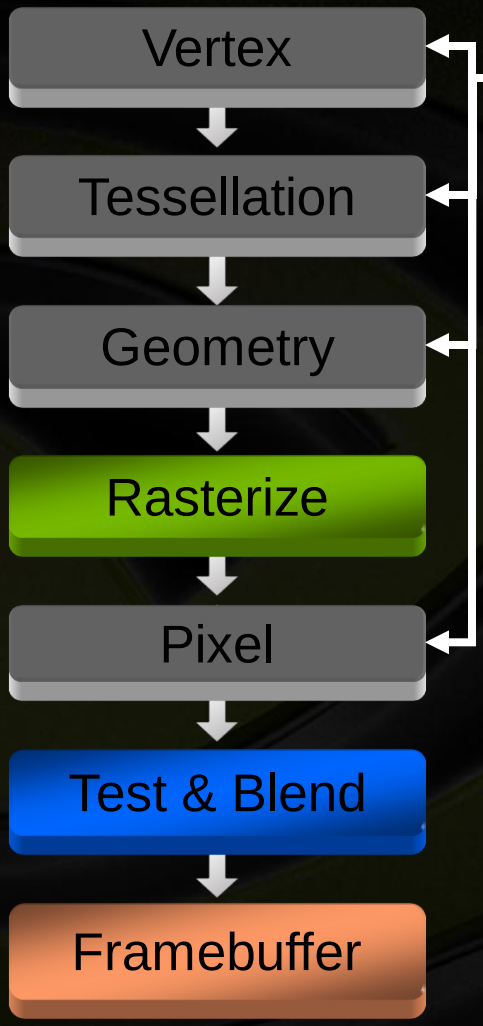



```
pixel_out
main(uniform sampler2D texture : TEXUNIT 0, pixel_in
{
  pixel_out OUT;
  float d= clamp(1.0 - pow(dot(IN.lightdist, IN.light
  float3 color = tex2D(texture, IN.texcoord).rgb;
  OUT.color = color * (d + 0.4);
  return OUT;
}
```

- **Hardware used to look like this**
  - **Vertex, pixel processing became programmable**
  - **New stages added**

© NVIDIA Corporation 2010

The Graphics Pipeline



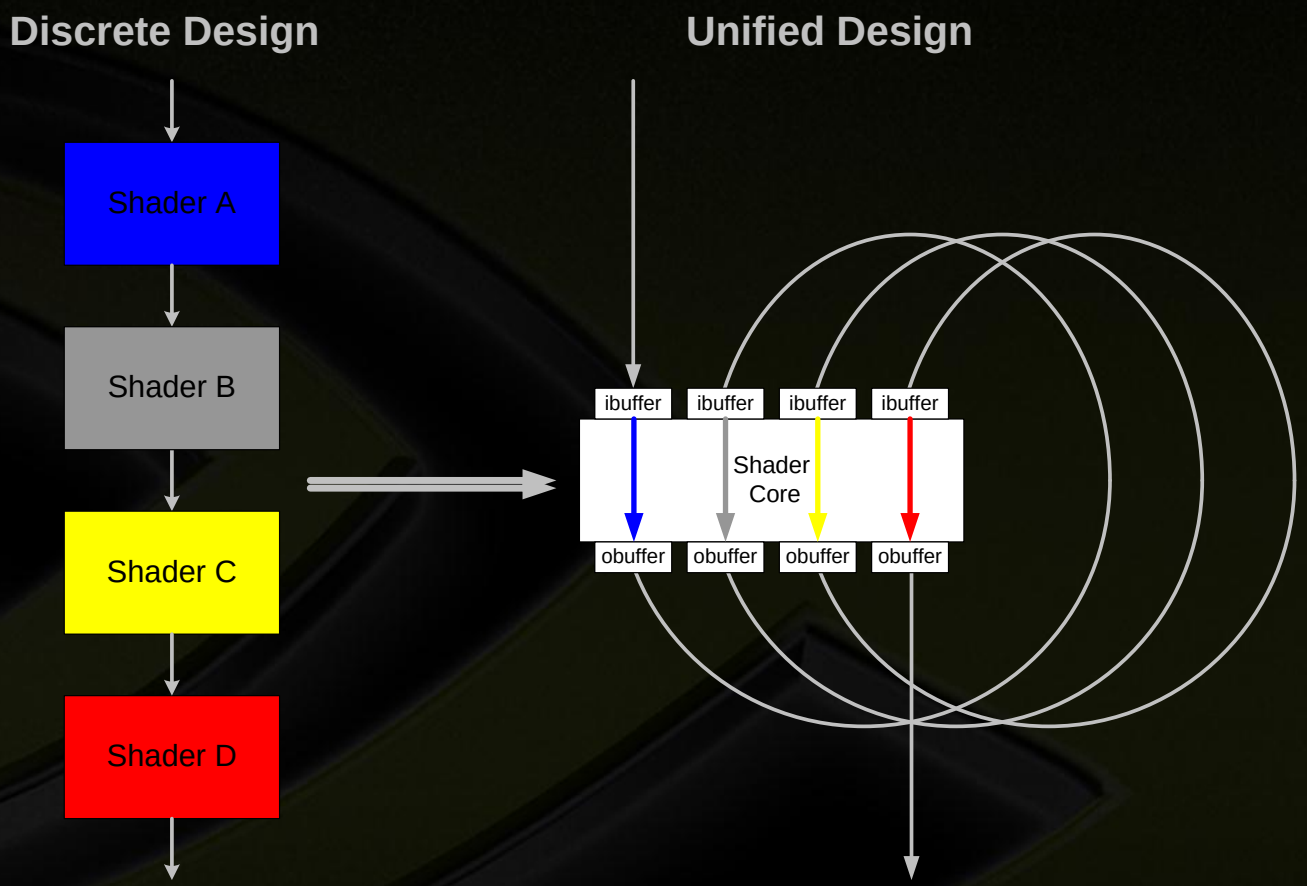

```
pixel_out
main(uniform sampler2D texture : TEXUNIT 0, pixel_in
{
  pixel_out OUT;
  float d= clamp(1.0 - pow(dot(IN.lightdist, IN.light
  float3 color = tex2D(texture, IN.texcoord).rgb;
  OUT.color = color * (d + 0.4);
  return OUT;
}
```

- **Hardware used to look like this**
  - **Vertex, pixel processing became programmable**
  - **New stages added**

GPU architecture increasingly centers around shader execution

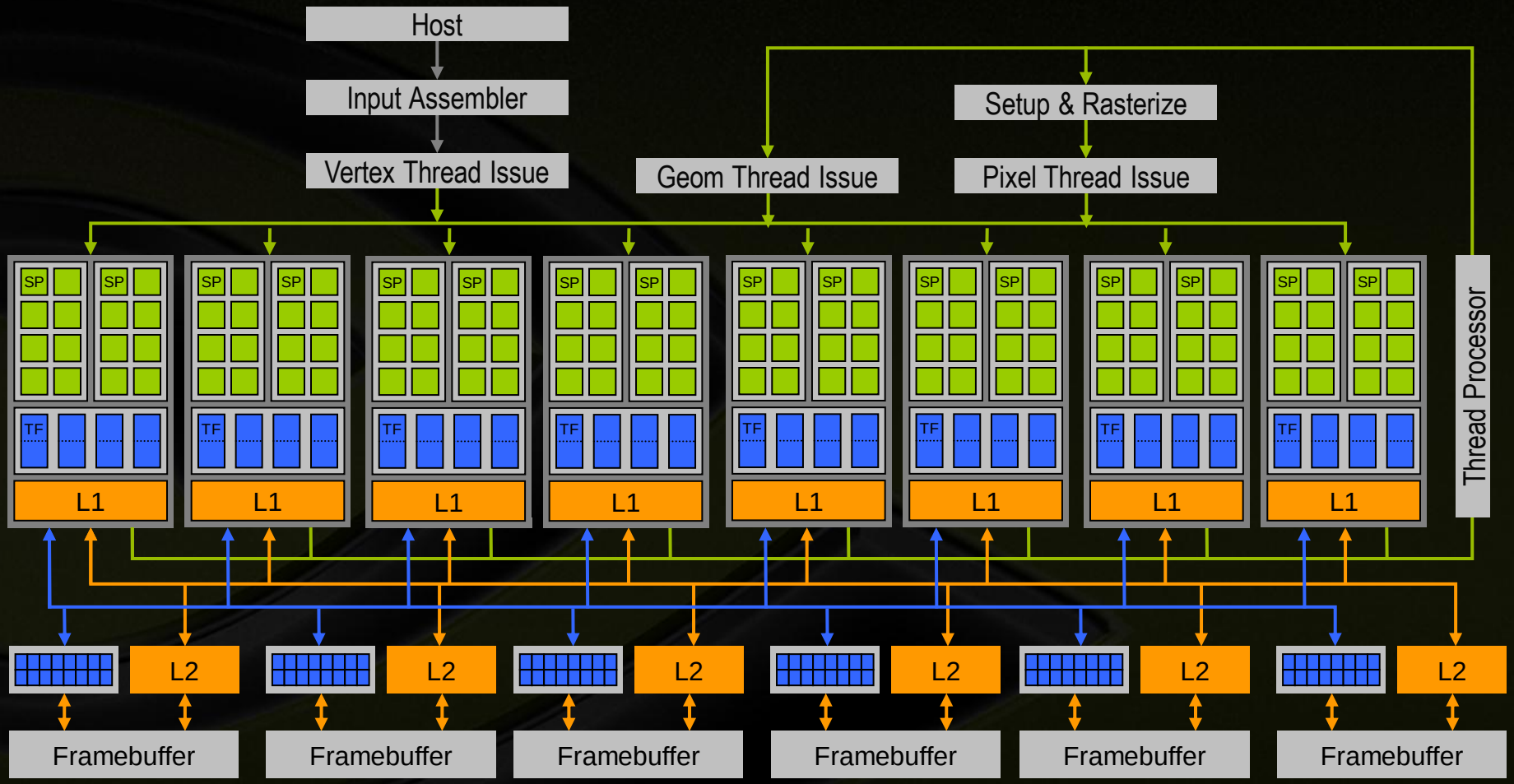
© NVIDIA Corporation 2010

Modern GPUs: Unified Design



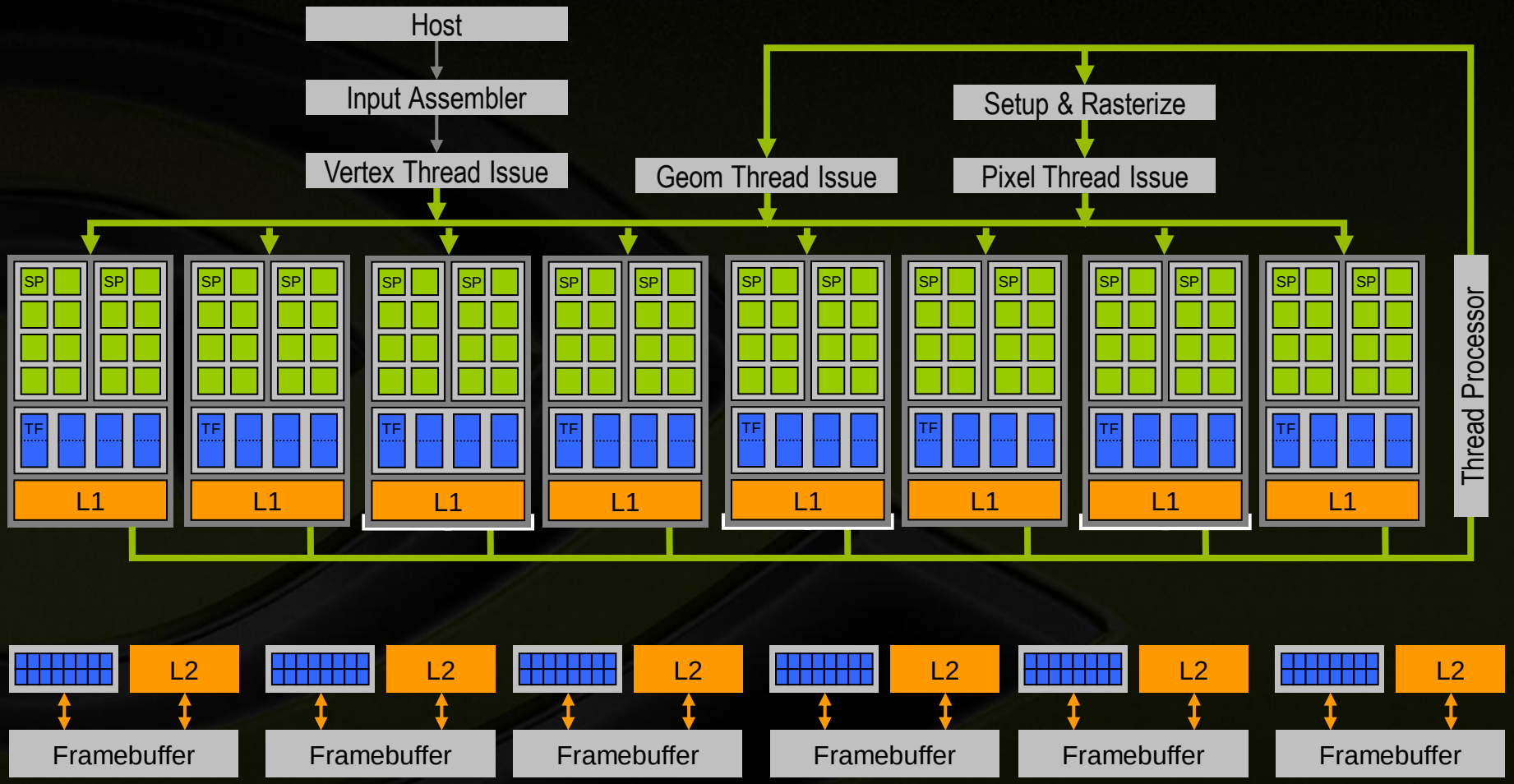
Vertex shaders, pixel shaders, etc. become *threads* running different programs on a flexible core

GeForce 8: Modern GPU Architecture



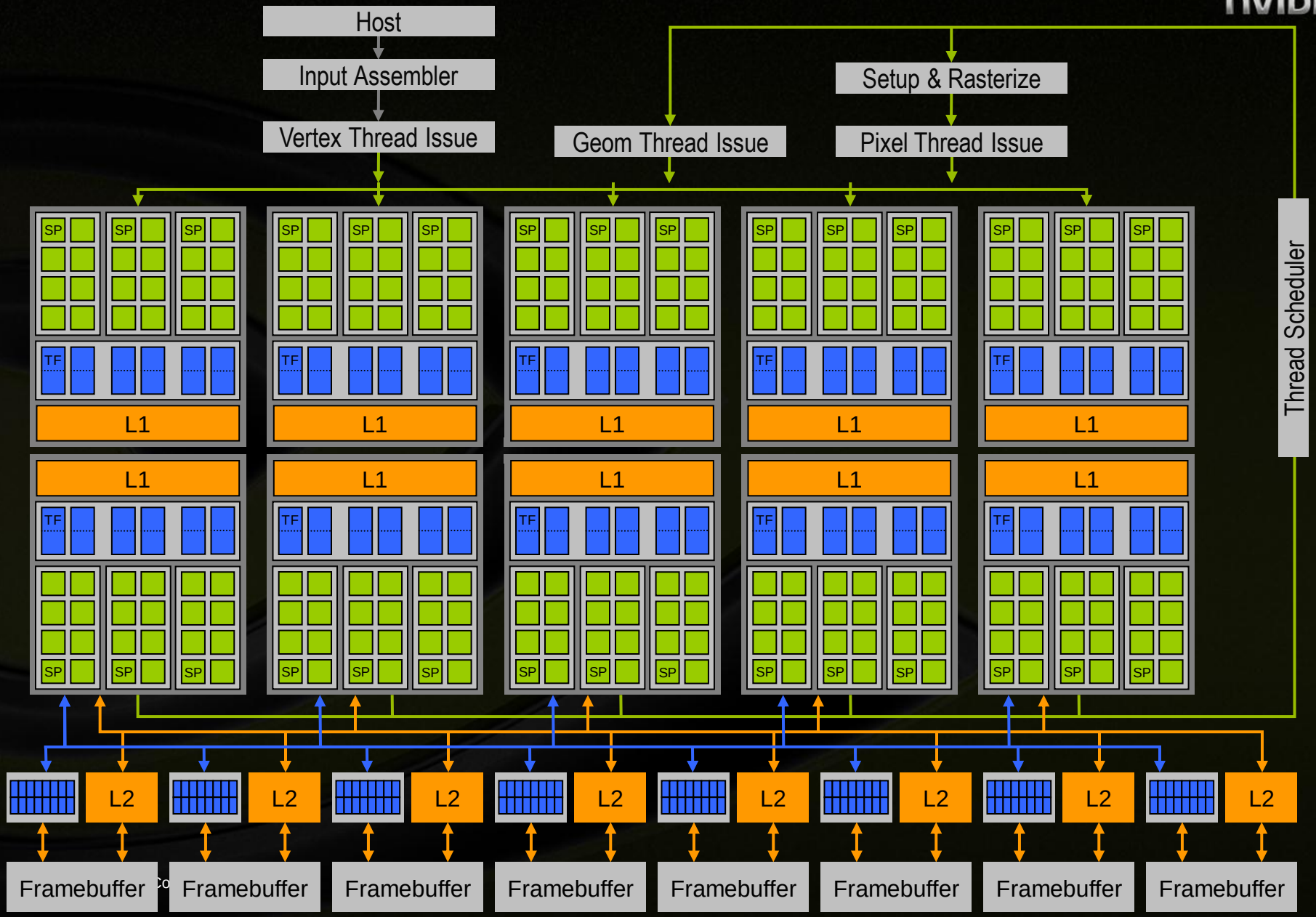
© NVIDIA Corporation 2010

GeForce 8: Modern GPU Architecture

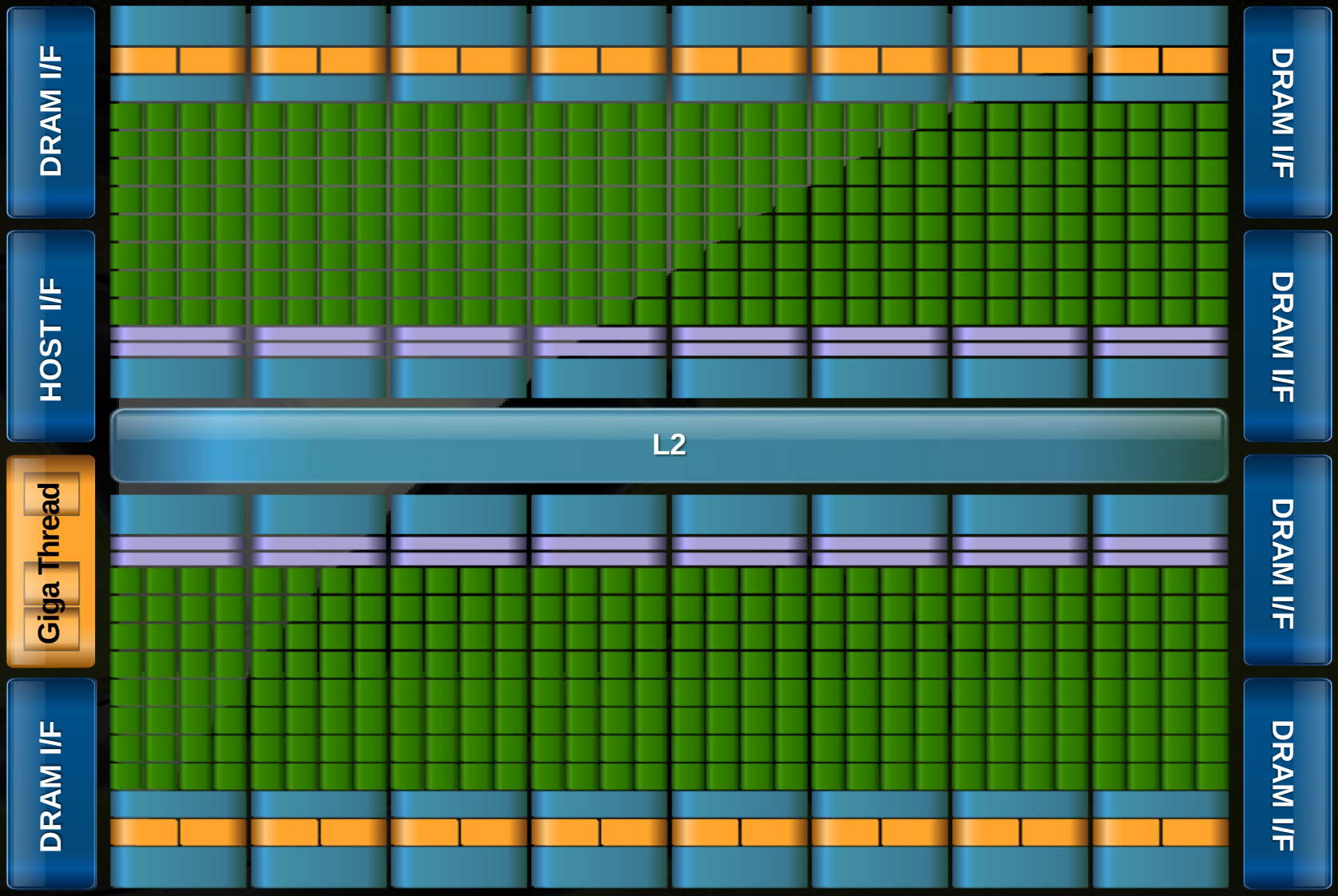


© NVIDIA Corporation 2010

Modern GPU Architecture: GT200



Current GPU Architecture: *Fermi*



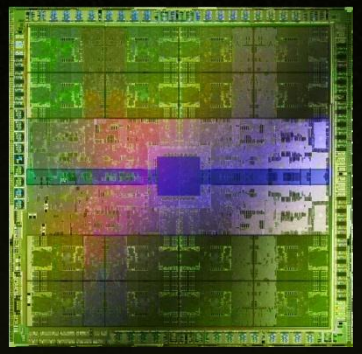
© NVIDIA Corporation 2010

NVIDIA next-gen "Fermi" architecture

GPUs Today

Lessons from Graphics Pipeline

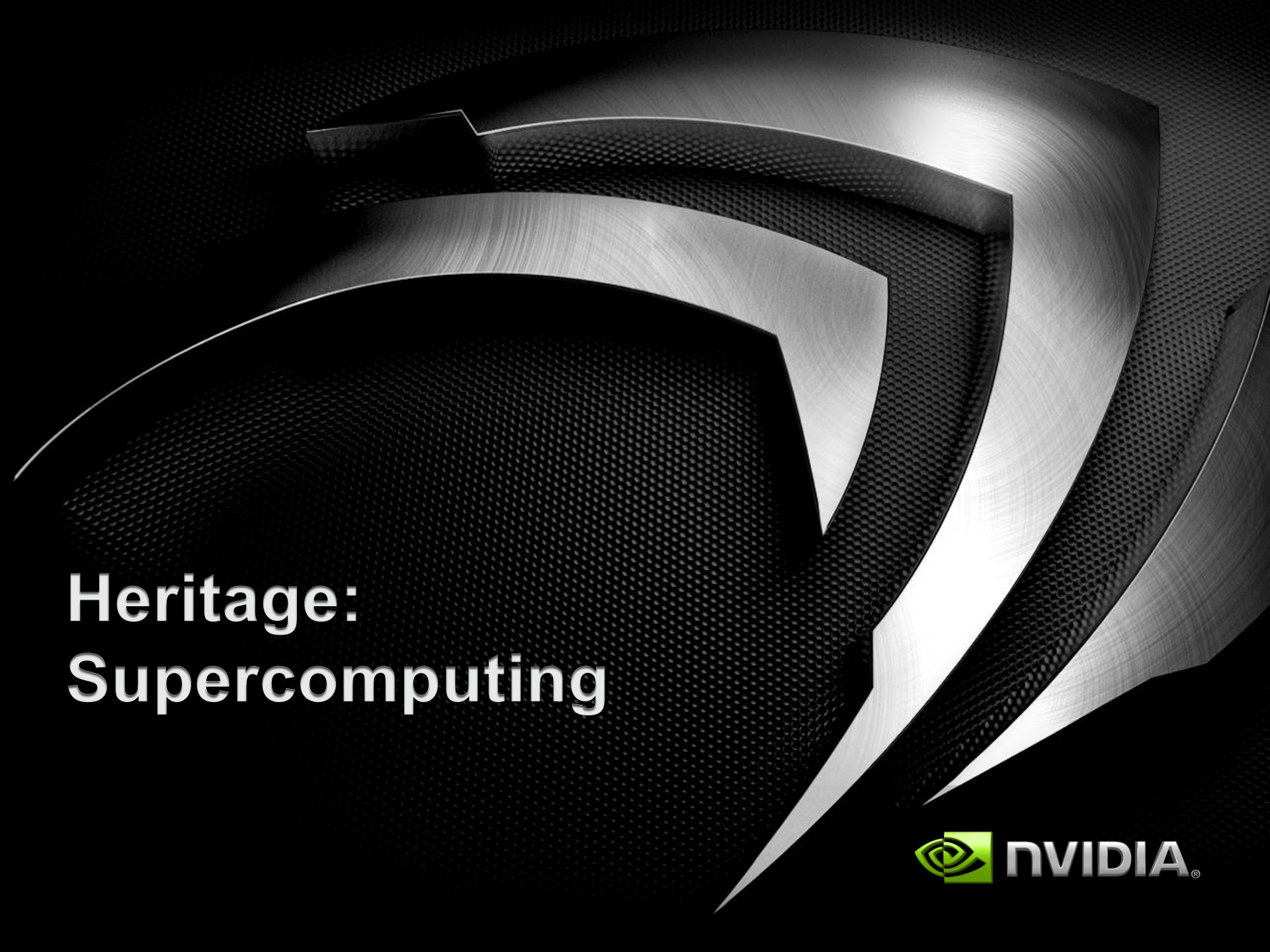
- **Throughput** is paramount
- Create, run, & retire **lots of threads** very rapidly
- Use **multithreading** to hide latency



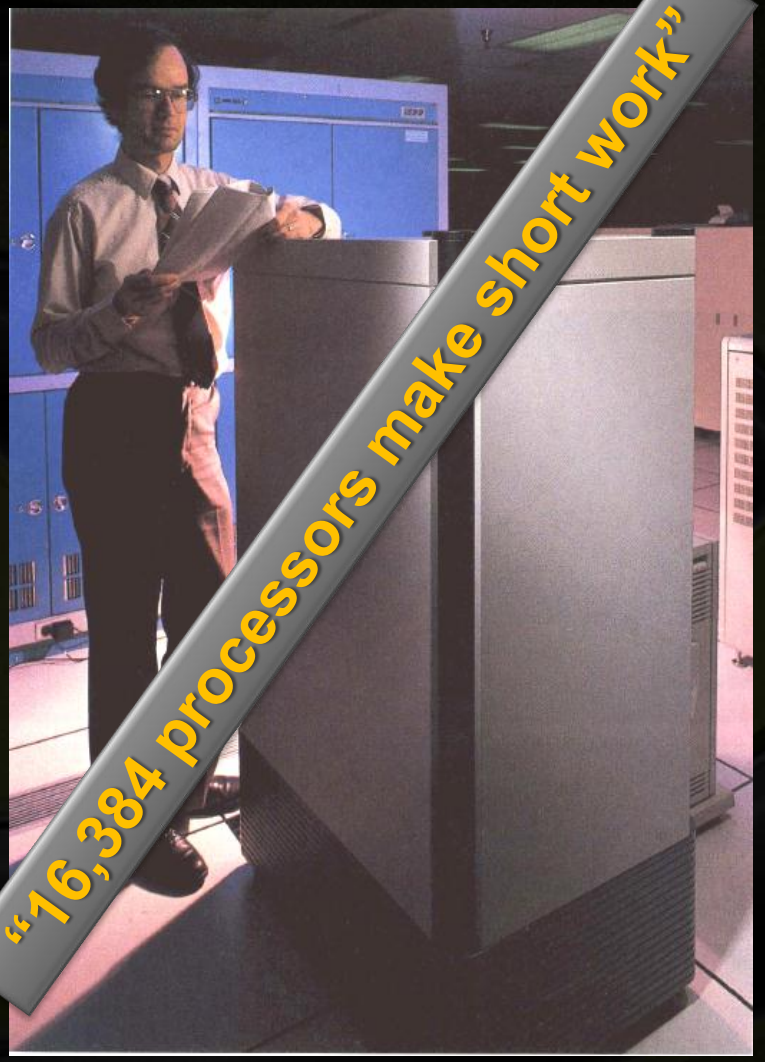
"Fermi"
3B xtors

© NVIDIA Corporation 2010

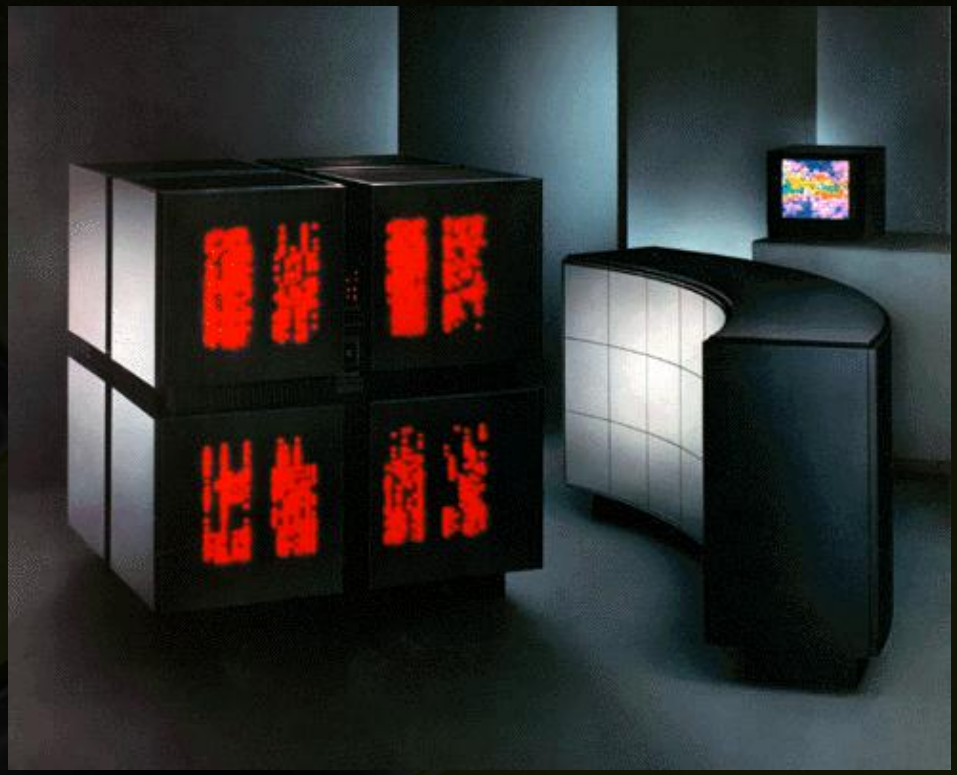
1995 2000 2005 2010


Heritage:
Supercomputing
NVIDIA

How to build a parallel machine: SIMD



MasPar MP1 (front), Goddard MPP (back)



Thinking Machines CM-2

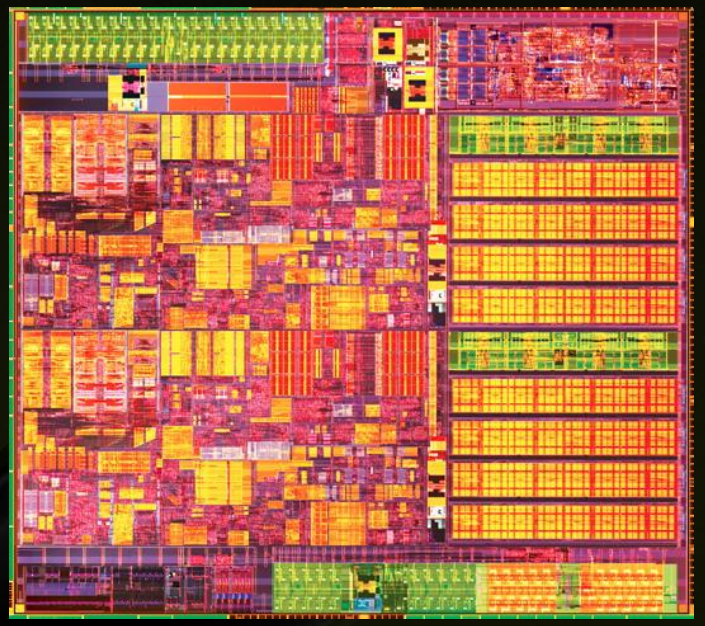
How to build a parallel machine: Hardware Multithreading

Tera MTA

© NVIDIA Corporation 2010

How to build a parallel machine:
Symmetric Multiprocessing

SGI Challenge



Intel Core2 Duo

Fermi, Oversimplified

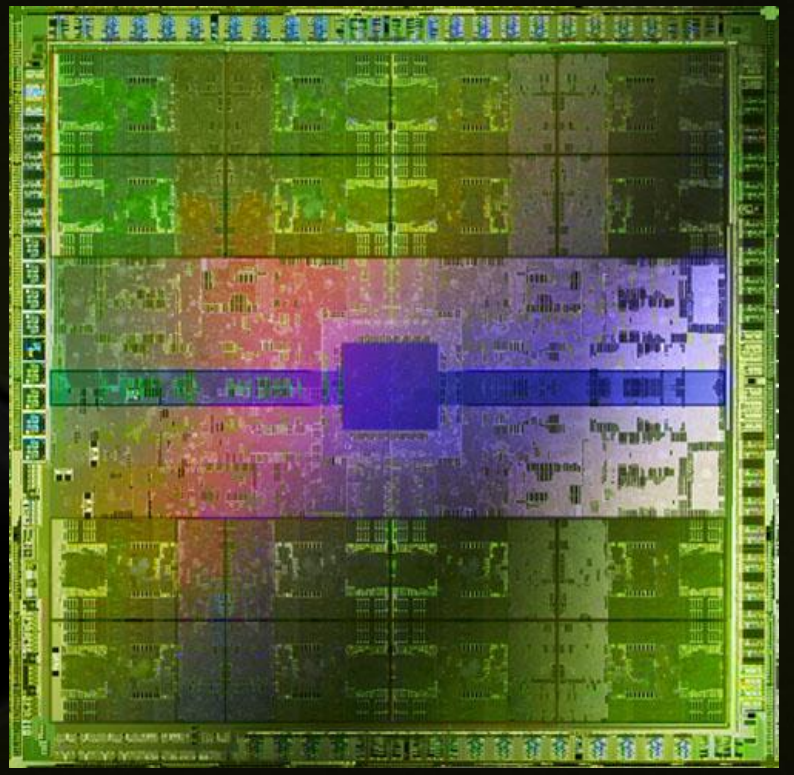
32-wide SIMD (two 16-wide datapaths)

48-way hardware multithreading

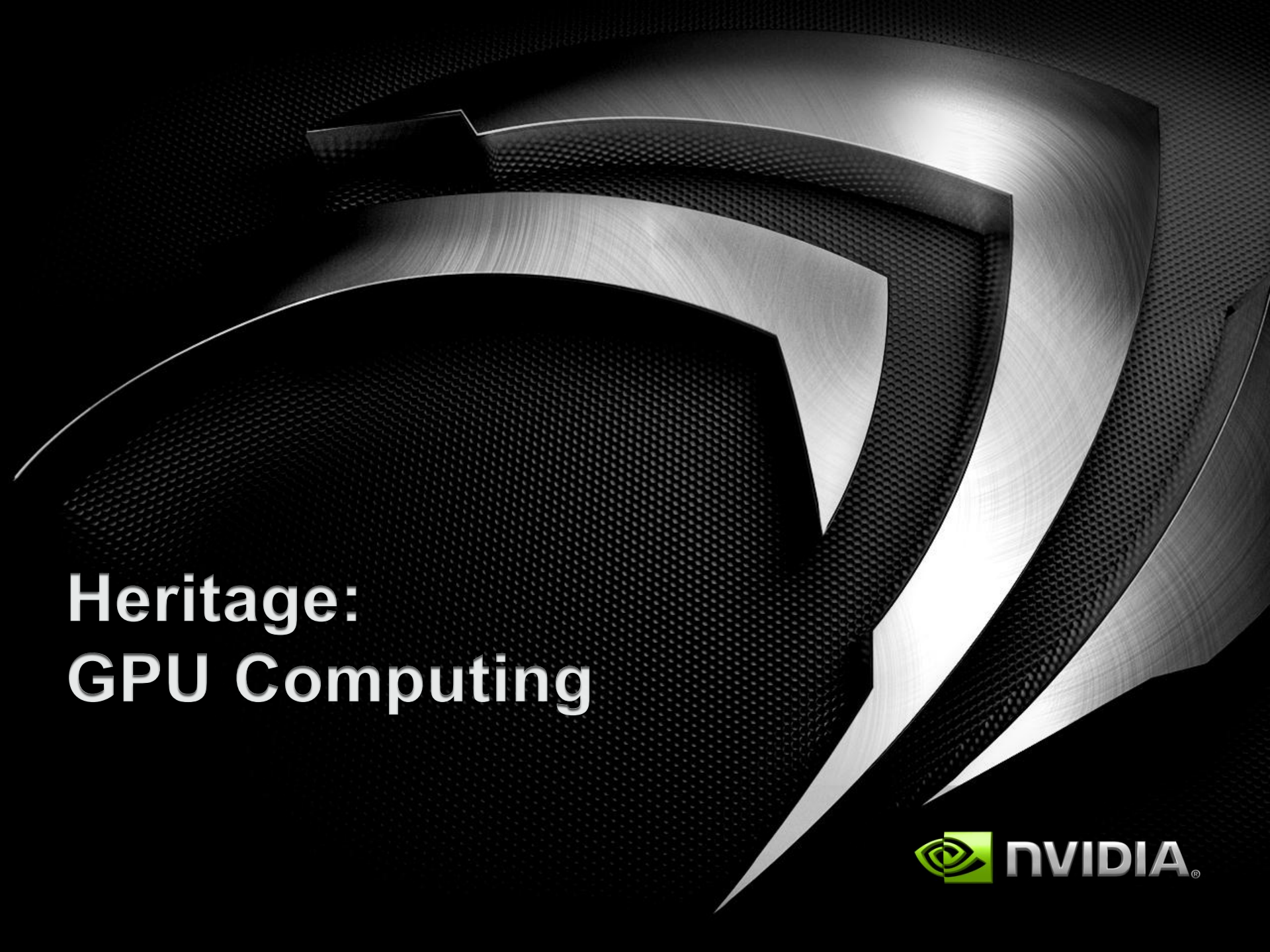
× 16-way SMP

24576 threads in flight

@ 512 FMA ops per clock



© NVIDIA Corporation 2010


Heritage:
GPU Computing
NVIDIA

GPU Computing 1.0: *GPGPU*

(Ignoring prehistory: Ikonas, Pixel Machine, Pixel-Planes…)

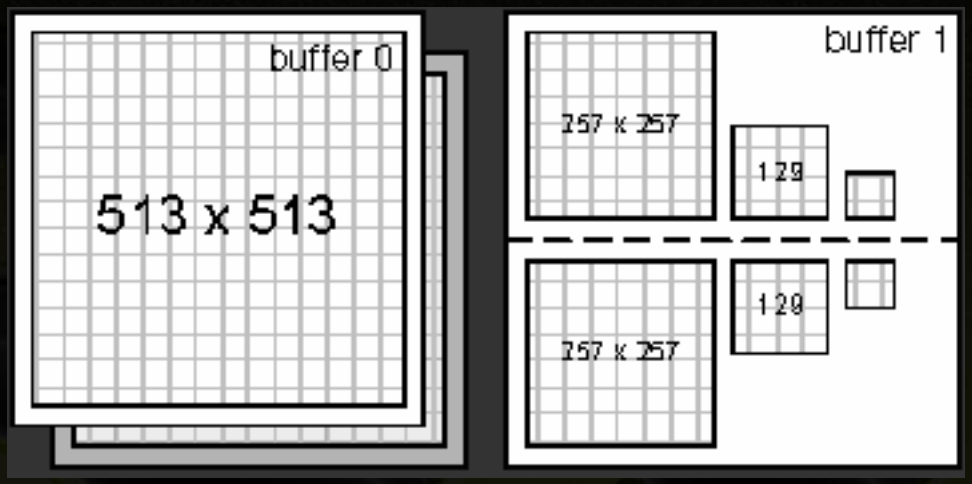
Compute pretending to be graphics

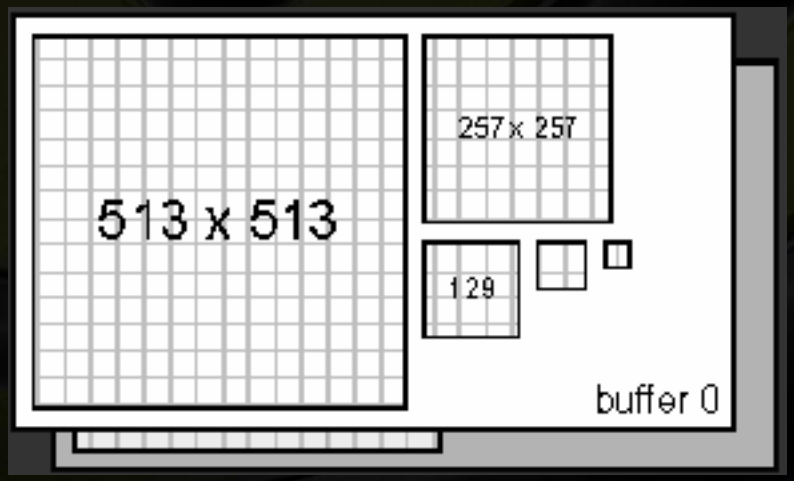
- **Disguise data as triangles or textures**
- **Disguise algorithm as render passes & shaders**

→Trick graphics pipeline into doing your computation!

© NVIDIA Corporation 2010

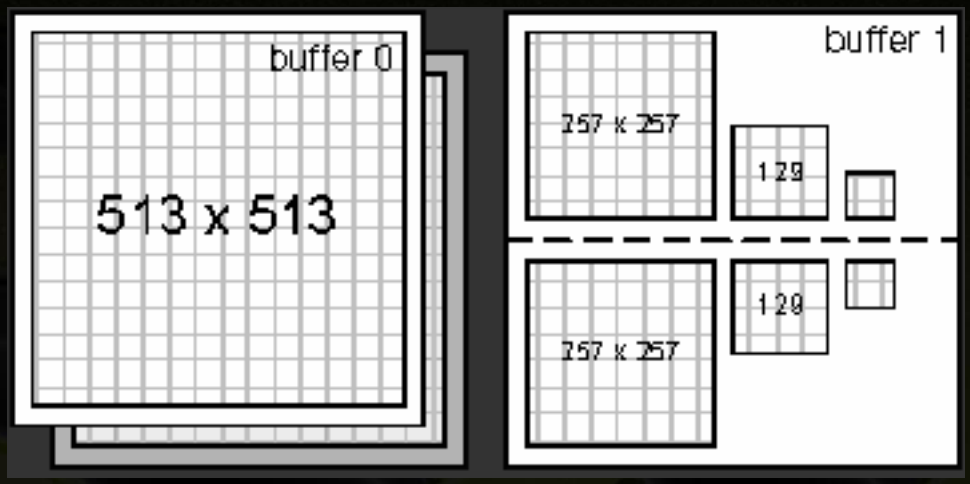
Typical GPGPU Constructs

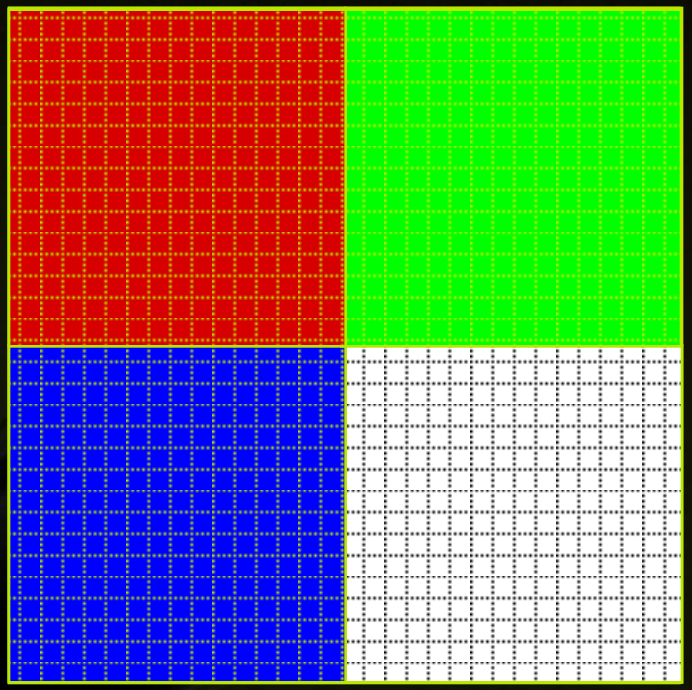


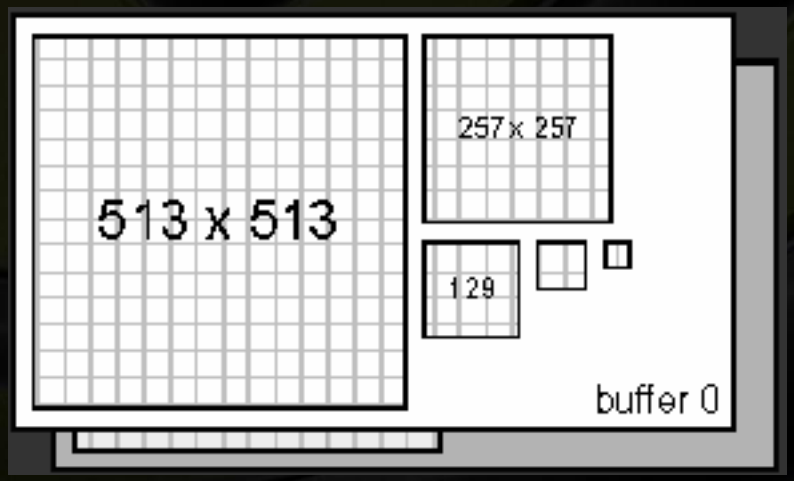


© NVIDIA Corporation 2010

Typical GPGPU Constructs



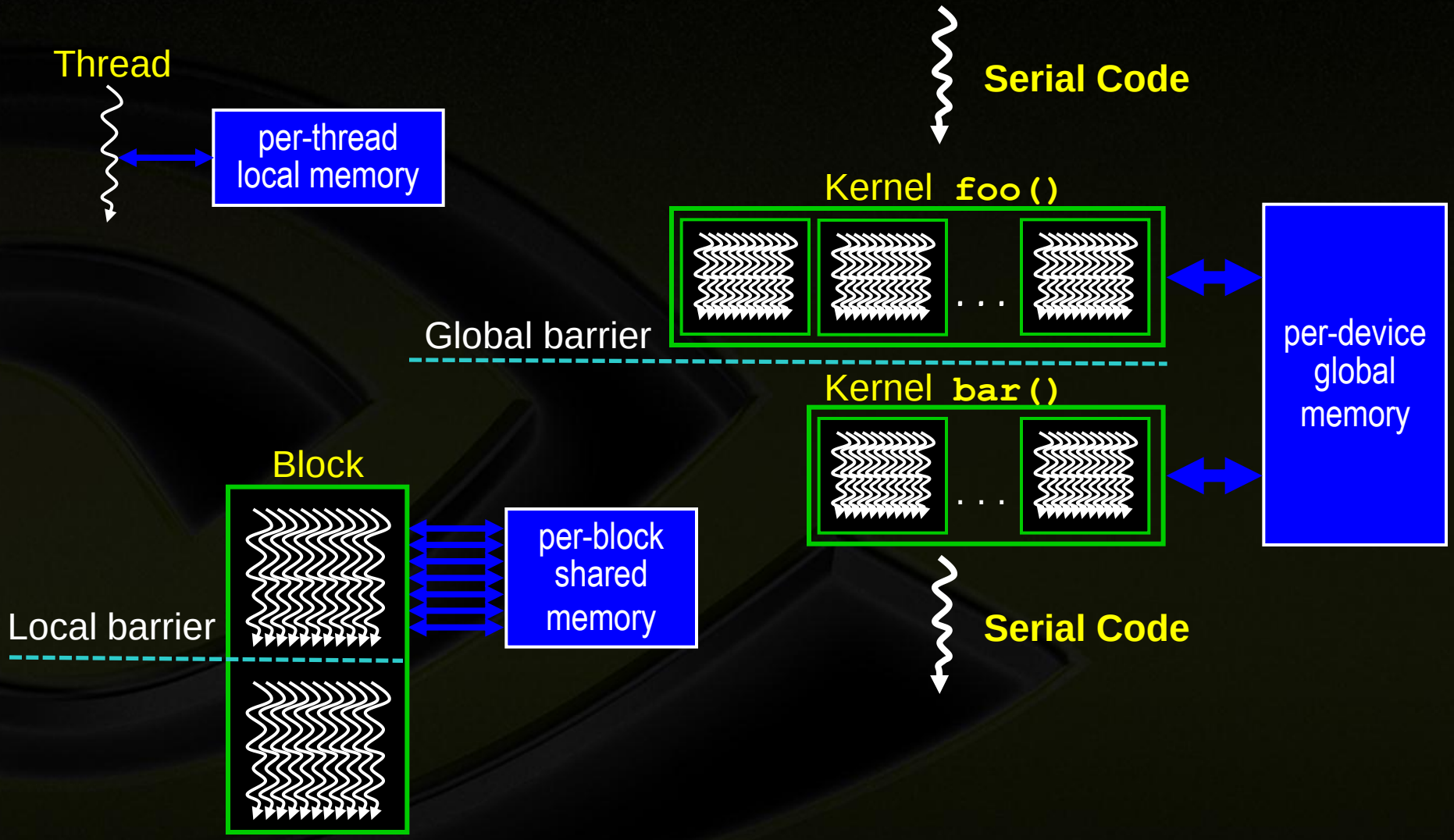




A Multigrid Solver for Boundary Value Problems Using Programmable Graphics Hardware
Nolan Goodnight et al., 2003

© NVIDIA Corporation 2010

GPU Computing 2.0: *CUDA*



GPU Computing:
State of the Union
NVIDIA

GPU Computing 3.0: *An Ecosystem*

NVIDIA

Languages & API's

CUDA C/C++

Microsoft® DirectX 11

Hardware & Product Lines

Research & Education

Algorithmic Sophistication

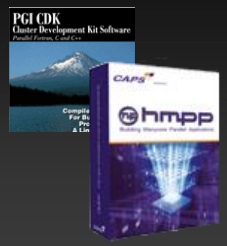
Cloud Services

amazon web services™

Libraries

$$\oint \mathbf{E}\cdot d\mathbf{A} = \frac{q_{enc}}{\varepsilon_0}$$
$$\oint \mathbf{B}\cdot d\mathbf{A} = 0$$
$$\oint \mathbf{E}\cdot d\mathbf{s} = -\frac{d\Phi_B}{dt}$$
$$\oint \mathbf{B}\cdot d\mathbf{s} = \mu_0\varepsilon_0\frac{d\Phi_E}{dt} + \mu_0 i_{enc}$$

Tools & Partners

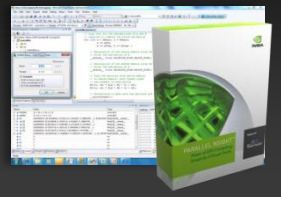


TOTALVIEW TECHNOLOGIES
A Rogue Wave Software Company

allinea ddt
the distributed debugging tool

Mathematical Packages

Integrated Development Environment



Microsoft® Visual Studio
PARTNER

GPU Computing Today

By the numbers:

- **200 Million CUDA capable GPU's**
- **600,000 CUDA Toolkit downloads**
- **100,000 active GPU Computing Developers**
- **8,000 members in Parallel Nsight developer program**
- **362 Universities teaching CUDA worldwide**
- **11 CUDA Centers of Excellence worldwide**

© NVIDIA Corporation 2010

CUDA Today

CUDA Toolkit 1.x

- C Compiler
- C Extensions
- Single Precision
- BLAS
- FFT
- SDK w/ 40 samples
- Win XP 64
- Atomics support
- Multi-GPU support

CUDA Toolkit 2.x

- Double Precision
- cuda-gdb
- Visual Profiler
- Compiler Optimizations
- Vista 32/64
- Mac OSX
- 3D Textures
- HW Interpolation
- DP FFT
- Parallel Nsight (beta)
- 16-32 conversion intrinsics
- Performance enhancements

CUDA Toolkit 3.x

- Fermi arch support
- C++ Class Templates
- C++ Class Inheritance
- Tools updates
- cuda-memcheck
- GPUDirect™
- 16-way concurrency
- Function pointers & recursion

New in 3.2

- New CUSPARSE Library
- New CURAND Library (Sobol)
- Support for 6GB Tesla, Quadro
- Multi-GPU Debugging
- Math Library Perf Increases
- Cluster Management Features
- Integrated TCC Mode

© NVIDIA Corporation 2010

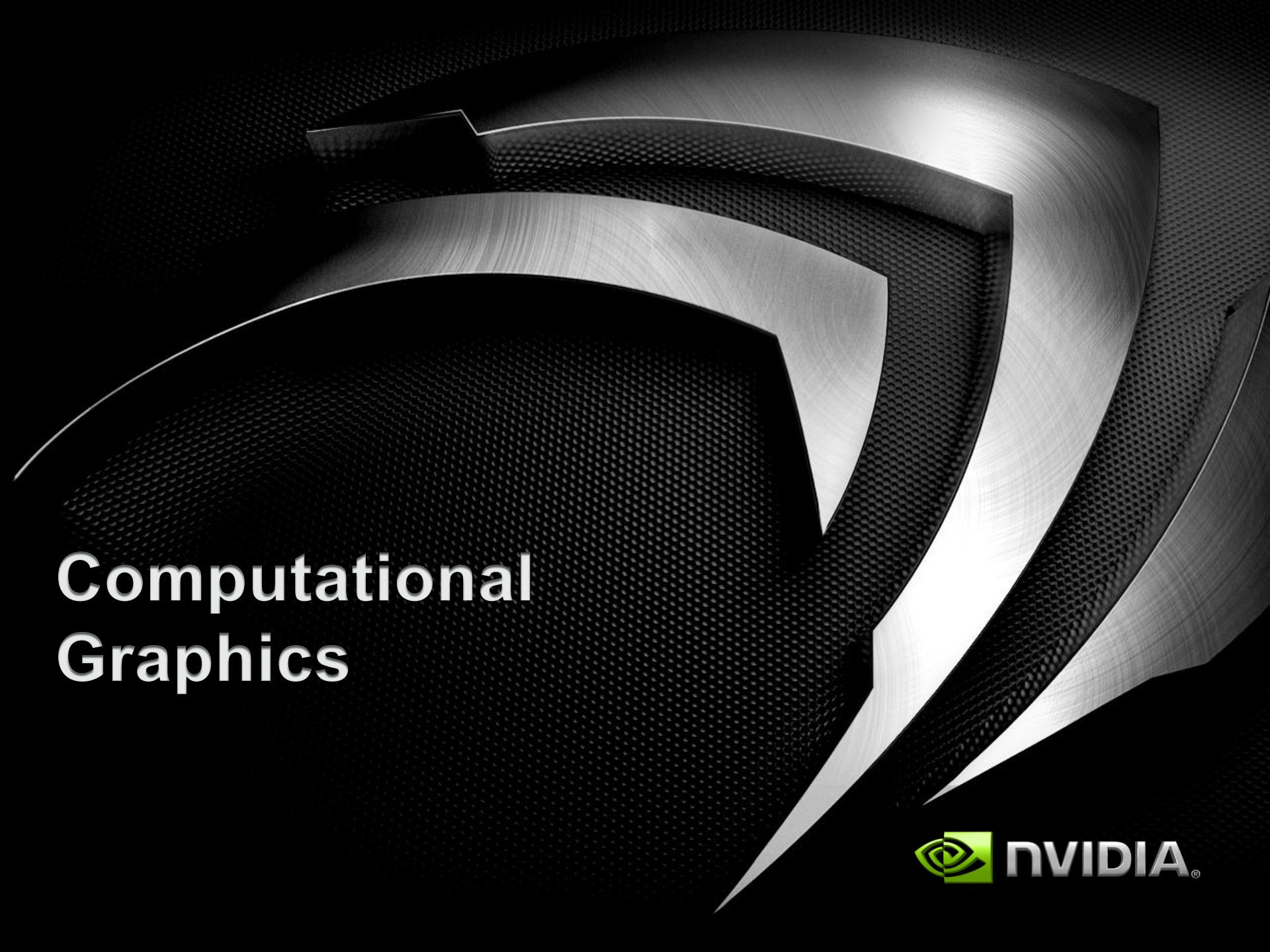
thrust::sort

sort.cu

```
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>
#include <thrust/generate.h>
#include <thrust/sort.h>
#include <cstdlib>

int main(void)
{
  // generate random data on the host
  thrust::host_vector<int> h_vec(1000000);
  thrust::generate(h_vec.begin(), h_vec.end(), rand);

  // transfer to device and sort
  thrust::device_vector<int> d_vec = h_vec;
  // sort 1B 32b keys/sec on Fermi
  thrust::sort(d_vec.begin(), d_vec.end());

  return 0;
}
```

http://thrust.googlecode.com

© NVIDIA Corporation 2010


Computational
Graphics
NVIDIA

Workloads

- Each GPU is designed to target a mix of known and speculative workloads
- The art of GPU design is choosing these workloads
  (and shipping on schedule!)

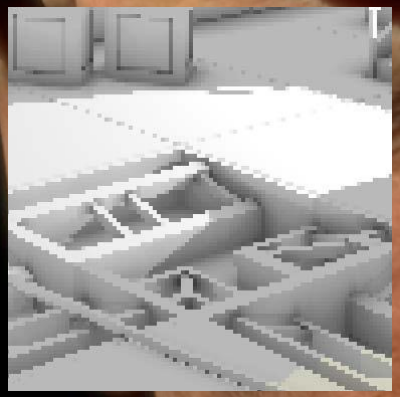
What workloads will drive future GPUs?

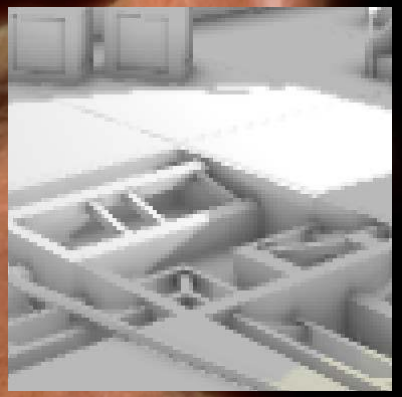
- High performance computing
- Graphics
- *Computational graphics*

© NVIDIA Corporation 2010

Filtering

- Separable filters
  - Depth of field
  - Film bloom
  - Subsurface scattering





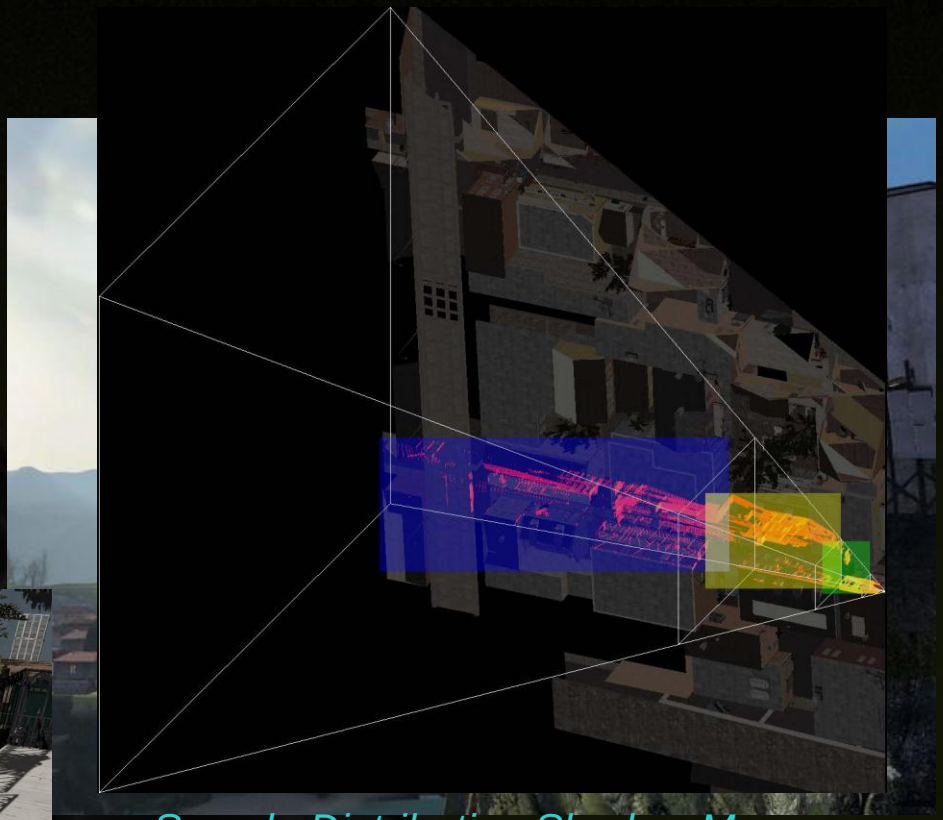
Subpixel Reconstruction Anti-Aliasing
Chajdas, McGuire, Luebke I3D 2011

Realistic Skin Rendering
d'Eon, Luebke, Enderton EGSR 2003

© NVIDIA Corporation 2010

Histogram

- Luminance values for tone mapping
- Sample distribution for shadow map creation

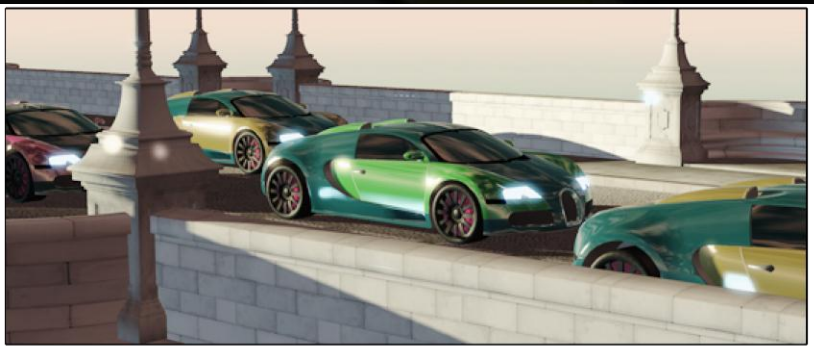


Sample Distribution Shadow Maps
Lauritzen, Salvi, Lefohn, I3D 2011

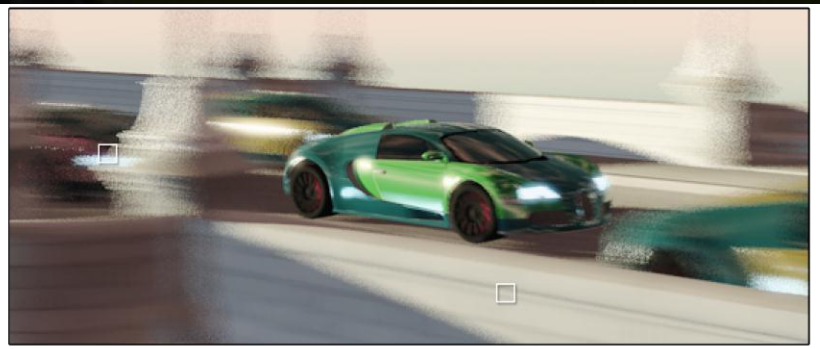
© NVIDIA Corporation 2010

Rasterization as Iteration

- Rasterize convex hull of moving triangle
- Ray trace against triangle at each pixel

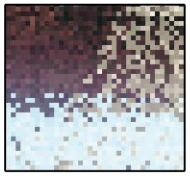


a) Conventional Rasterization

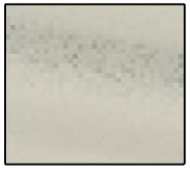


b) Stochastic Rasterization

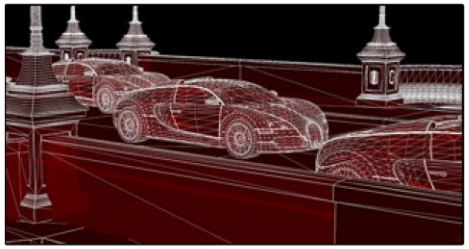
c)



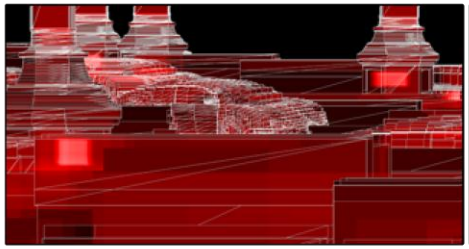
d)



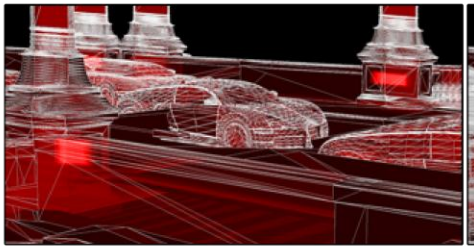
10x Zoom



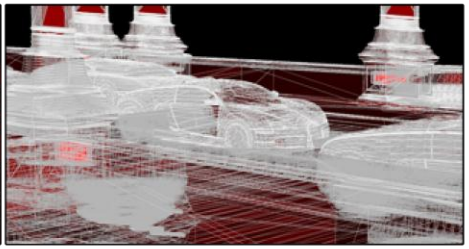
e) t=1 Geometry



f) 2D AABB Geometry



g) 2D Convex Hull Geometry

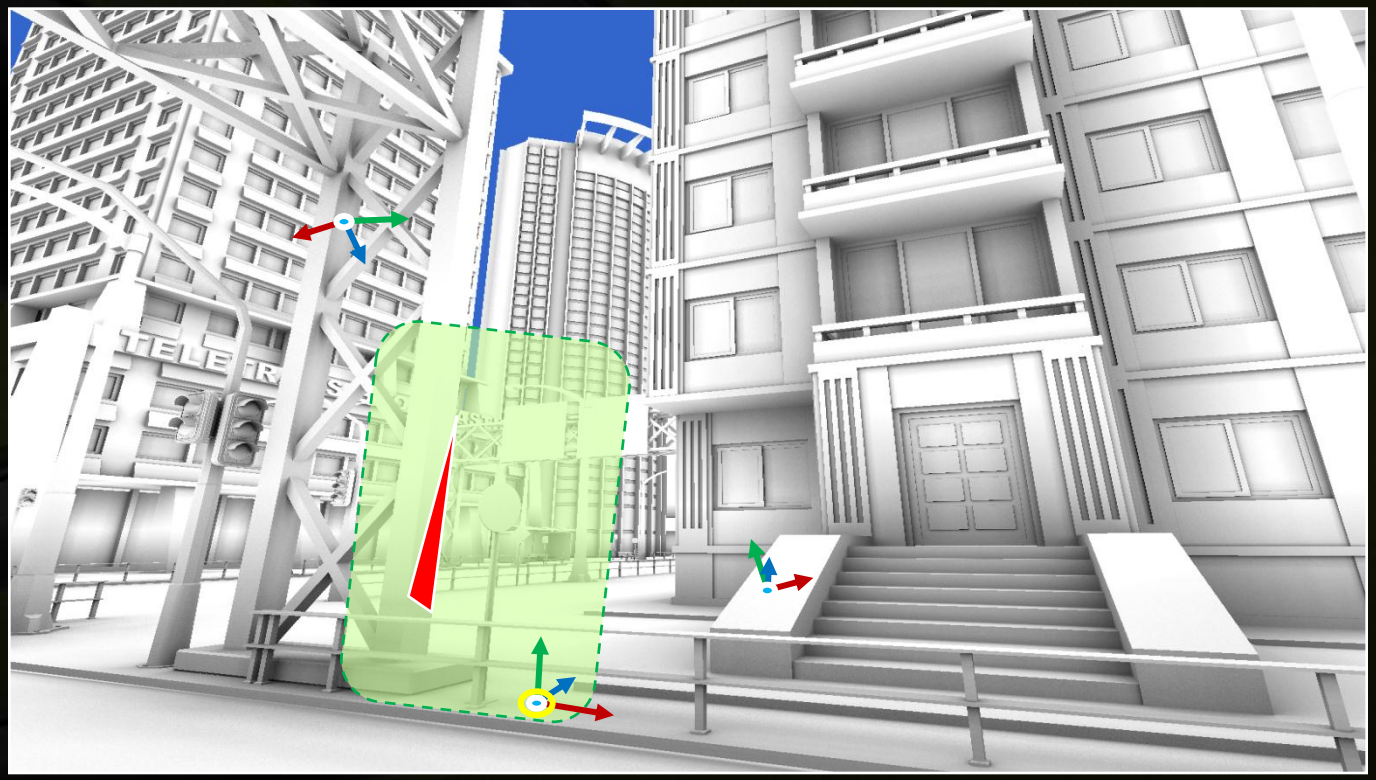


h) 2D C.H. Full Wireframe

© NVIDIA Corporation 2010

Rasterization as Iteration

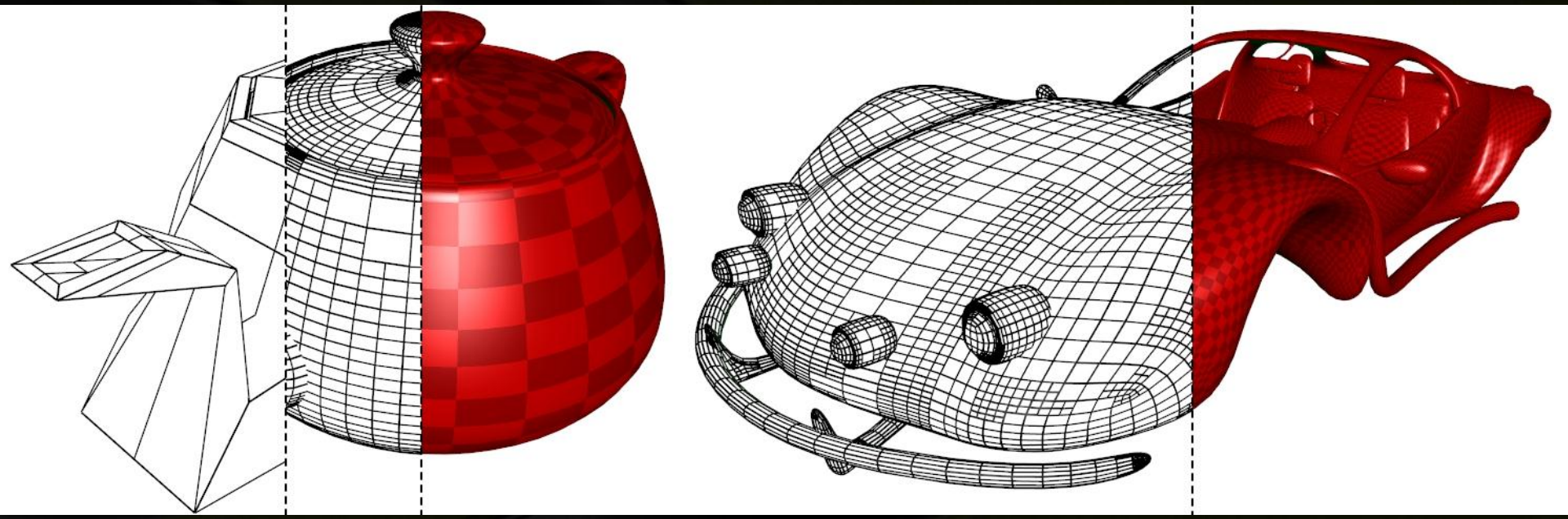
- Darken pixels by % of hemisphere blocked by nearby triangles
- Compute triangle *regions of influence* to find affected pixels



Two Methods for Fast Ray-Cast Ambient Occlusion, Samuli Laine & Tero Karras, EGSR 2010

CUDA Tessellation

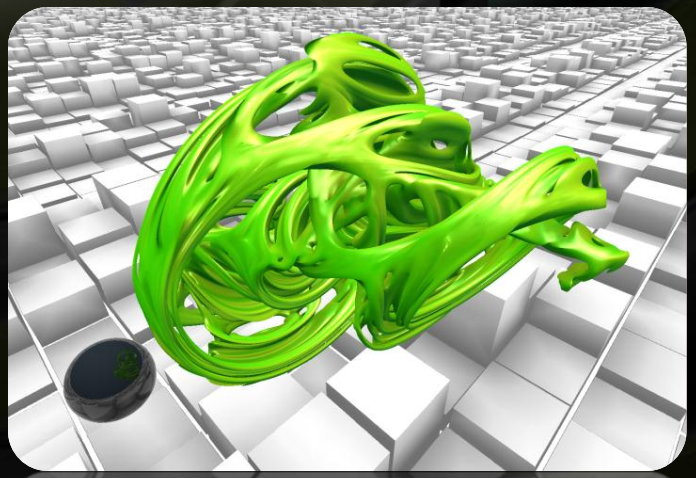
- Flexible adaptive geometry generation
- Recursive subdivision



Real-Time View-Dependent Rendering of Parametric Surfaces
Eisenacher, Meyer, Loop 2009

© NVIDIA Corporation 2010

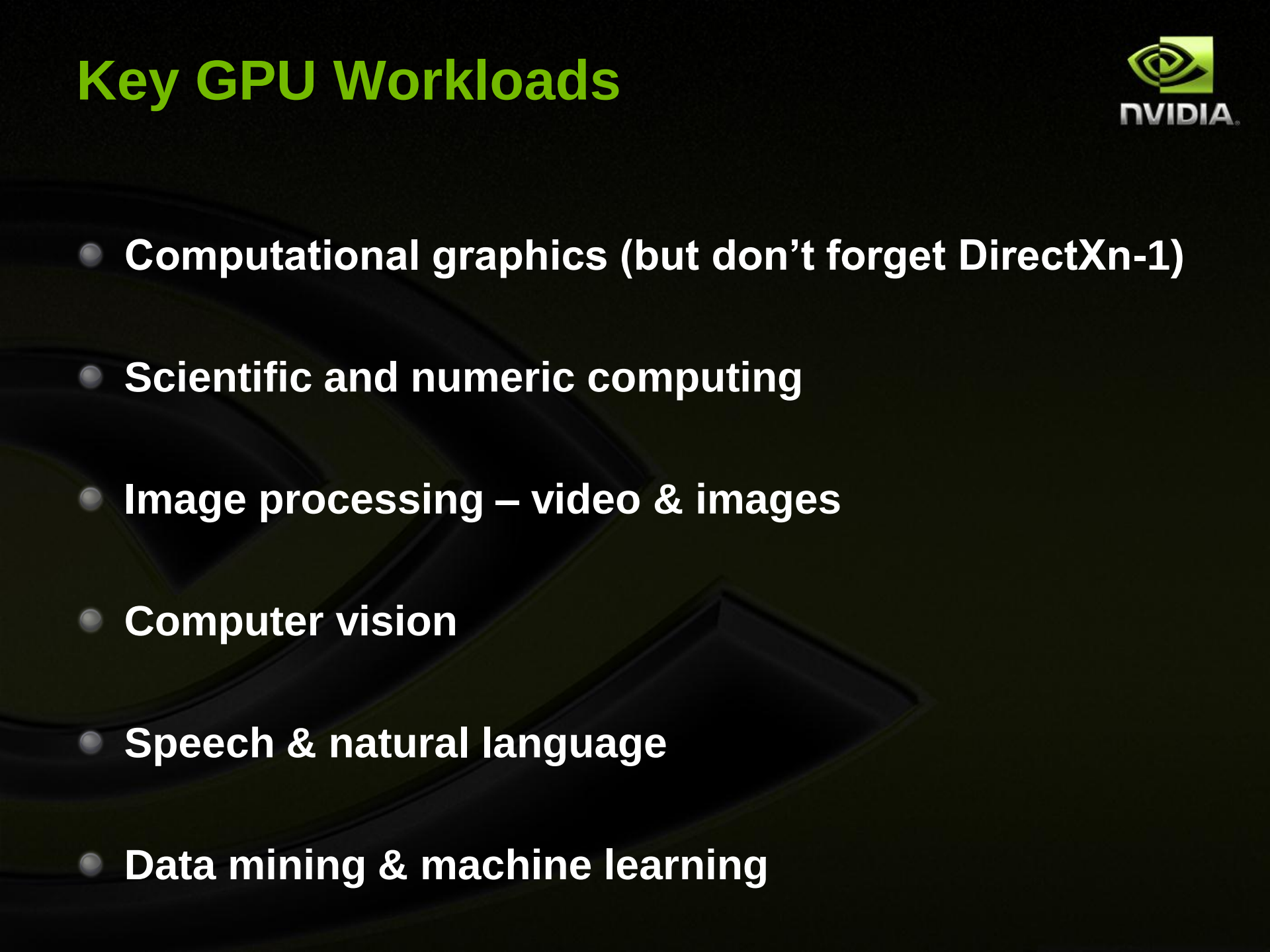
Ray Tracing



OptiX: A General-Purpose Ray Tracing Engine, Parker *et al.*, *SIGGRAPH 2010*

GPU Computing 4.0?

Key GPU Workloads



- Computational graphics (but don’t forget DirectXn-1)
- Scientific and numeric computing
- Image processing – video & images
- Computer vision
- Speech & natural language
- Data mining & machine learning

Key CUDA Challenges

- Express other programming models elegantly
  - Persistent thread blocks: fill machine, fetch work, repeat
  - Producer-consumer: work queues, work stealing, …
  - Nested & irregular parallelism: divide&conquer, BFS, …
  - Task-parallel: kernel, thread block or warp as parallel task
- Express locality: deep memories, compute “places”
- Improve & mature development environment

Key GPGPU Researcher Challenges

- Foster high-level libraries, languages, platforms
  - Domain-specific tools & packages
  - “Horizontal” programming layers & patterns
- Rethink algorithms, numerics, approaches
  - Computation is cheap
  - Data movement is costly
  - *Think parallel !*

Final Thoughts – Education

- We should teach parallel computing in CS 1 or CS 2
  - Computers don't get faster, just wider
  - Manycore is the ~~future~~ now of computing

Insertion Sort Heap Sort Merge Sort

Which goes faster on large data?

ALL Students need to understand this! Early!

Questions?
dluebke@nvidia.com
NVIDIA

Computational Challenge

© NVIDIA Corporation 2010

Fermi Features, Spoken in HPC

- Full scatter-gather and automatic predication to simplify SIMD programming (SIMT)
- Hardware accelerated task distributor for dynamic load balancing
- Dynamically partitionable register file
- High performance atomic operations
- On-chip crossbar network
- Local scratchpad per core for fine-grained thread coop
- IEEE 754-2008 floating point with high-speed fp64
- High-speed GDDR memory interface
- Optional ECC protection on DRAM, L2, L1, ShMem, RF
- Mature programming models based on C, C++, Fortran

© NVIDIA Corporation 2010

CUDA Examples

CUDA C Example

```
void saxpy_serial(int n, float a, float *x, float *y)
{
    for (int i = 0; i < n; ++i)
        y[i] = a*x[i] + y[i];
}

// Invoke serial SAXPY kernel
saxpy_serial(n, 2.0, x, y);
```

Serial C Code

```
__global__ void saxpy_parallel(int n, float a, float *x, float *y)
{
    int i = blockIdx.x*blockDim.x + threadIdx.x;
    if (i < n)  y[i] = a*x[i] + y[i];
}

// Invoke parallel SAXPY kernel with 256 threads/block
int nblocks = (n + 255) / 256;
saxpy_parallel<<<nblocks, 256>>>(n, 2.0, x, y);
```

Parallel C Code

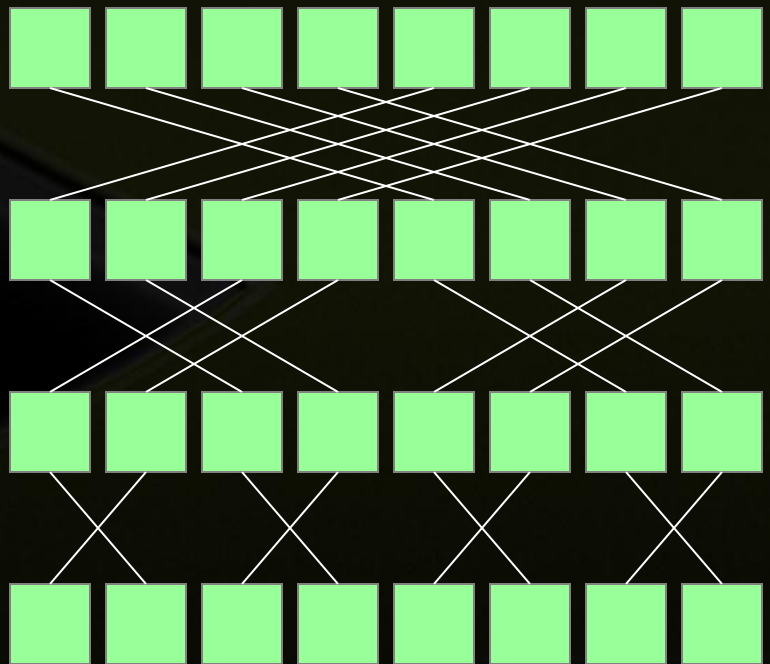
Example: Parallel Reduction

- Summing up a sequence with 1 thread:

```
int sum = 0;
for(int i=0; i<N; ++i)   sum += x[i];
```

- Parallel reduction builds a summation tree
  - each thread holds 1 element
  - stepwise partial sums
  - N threads need log N steps

  - one possible approach:
    Butterfly pattern



© NVIDIA Corporation 2009

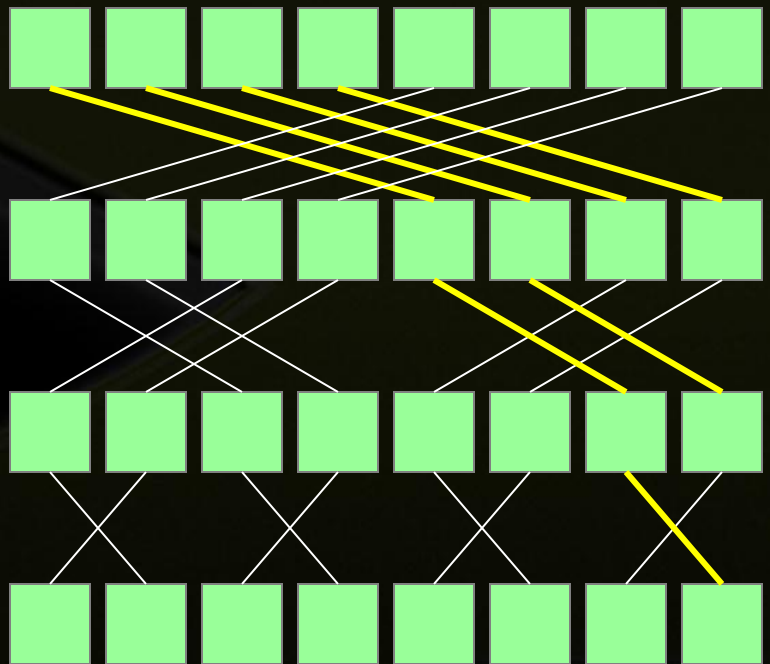
Example: Parallel Reduction

- Summing up a sequence with 1 thread:

```
int sum = 0;
for(int i=0; i<N; ++i)   sum += x[i];
```

- Parallel reduction builds a summation tree
  - each thread holds 1 element
  - stepwise partial sums
  - N threads need log N steps

  - one possible approach:
    Butterfly pattern



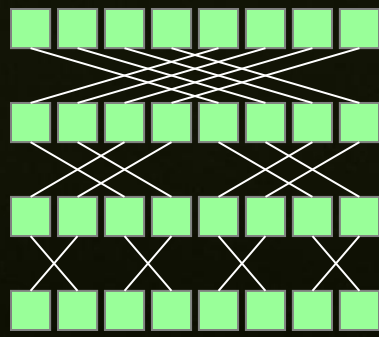
© NVIDIA Corporation 2009

Parallel Reduction for 1 Block

```
// INPUT: Thread i holds value x_i
int i = threadIdx.x;
__shared__ int sum[blocksize];
```

```
// One thread per element
sum[i] = x_i; __syncthreads();
```

```
for(int bit=blocksize/2; bit>0; bit/=2)
{
    int t=sum[i]+sum[i^bit]; __syncthreads();
    sum[i]=t;                __syncthreads();
}
```



```
// OUTPUT: Every thread now holds sum in sum[i]
```

© NVIDIA Corporation 2009