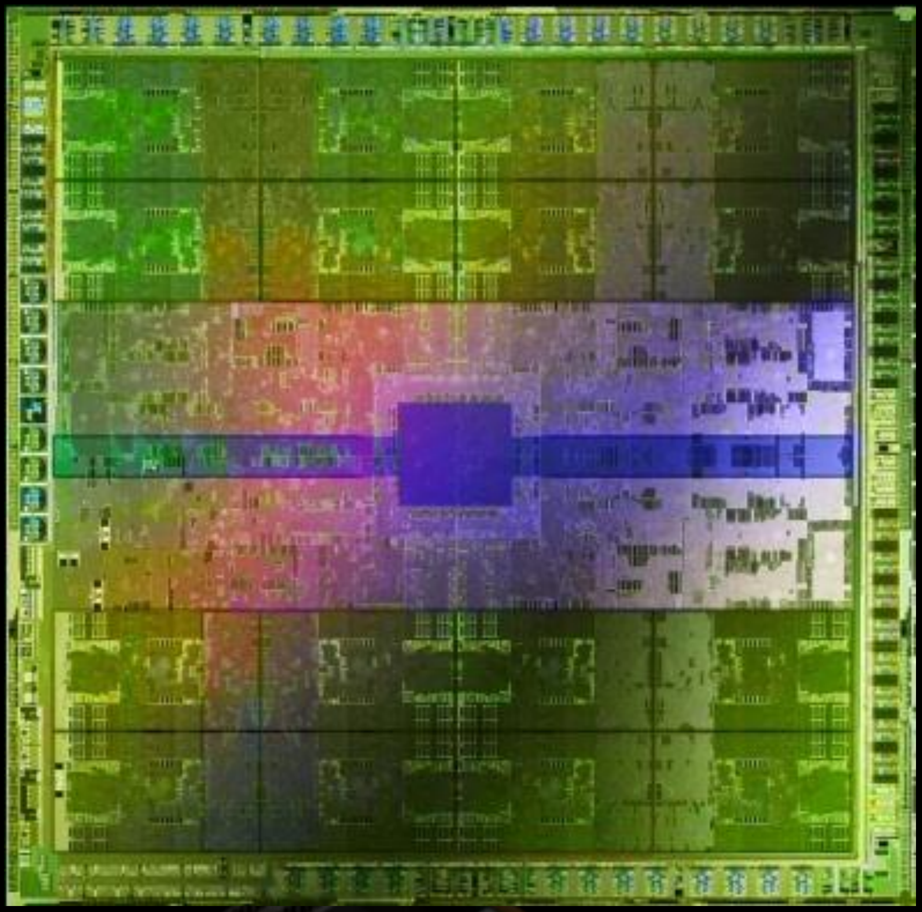
GPU TECHNOLOGY CONFERENCE
OCTOBER 11-14, 2011 | SAN JOSE, CALIFORNIA

GPU Computing

Past, Present, and Future

David Luebke
Director, NVIDIA Research

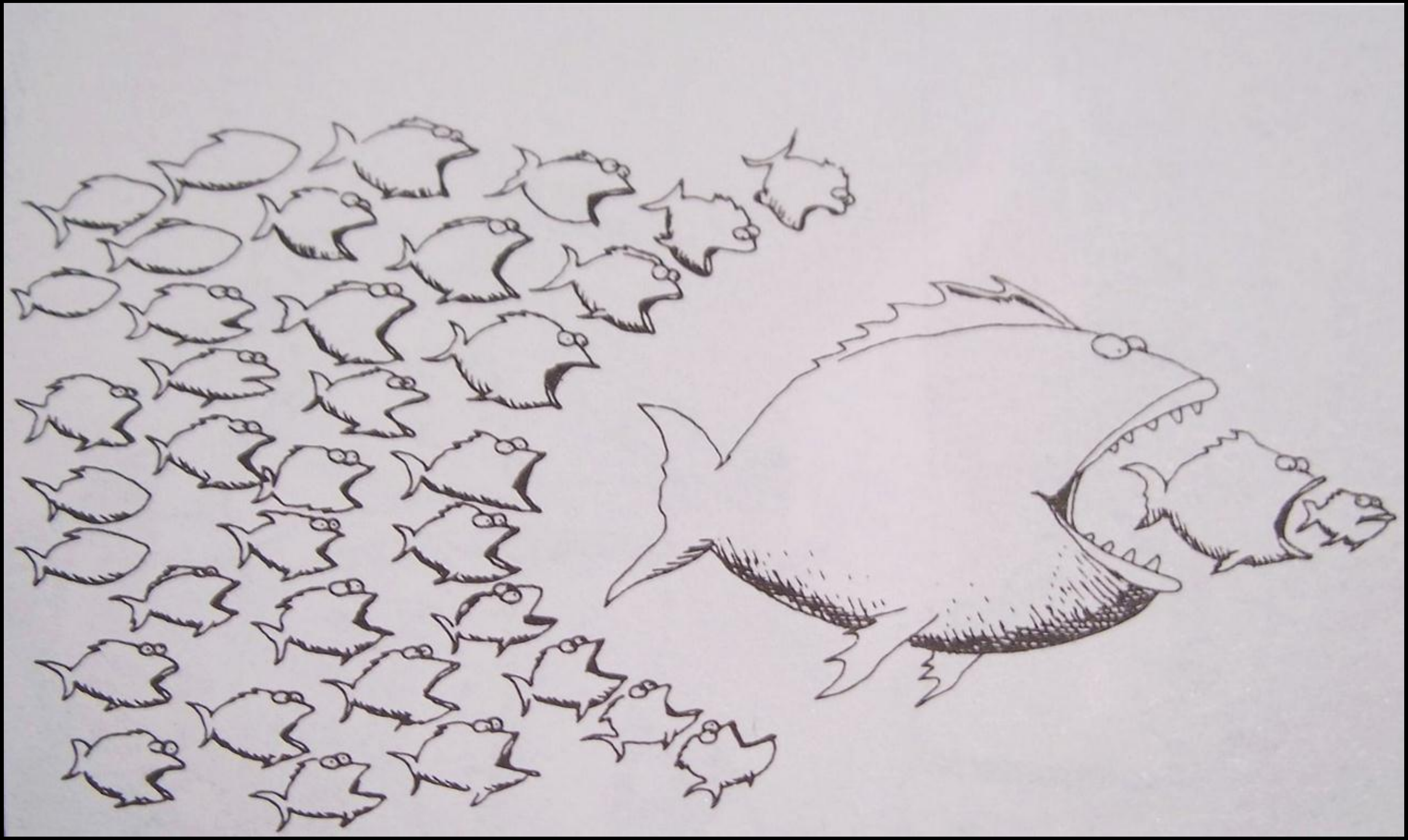
Graphics Processing Unit (GPU)



GPU TECHNOLOGY CONFERENCE

What GPUs Do

Parallel Computing, Illustrated

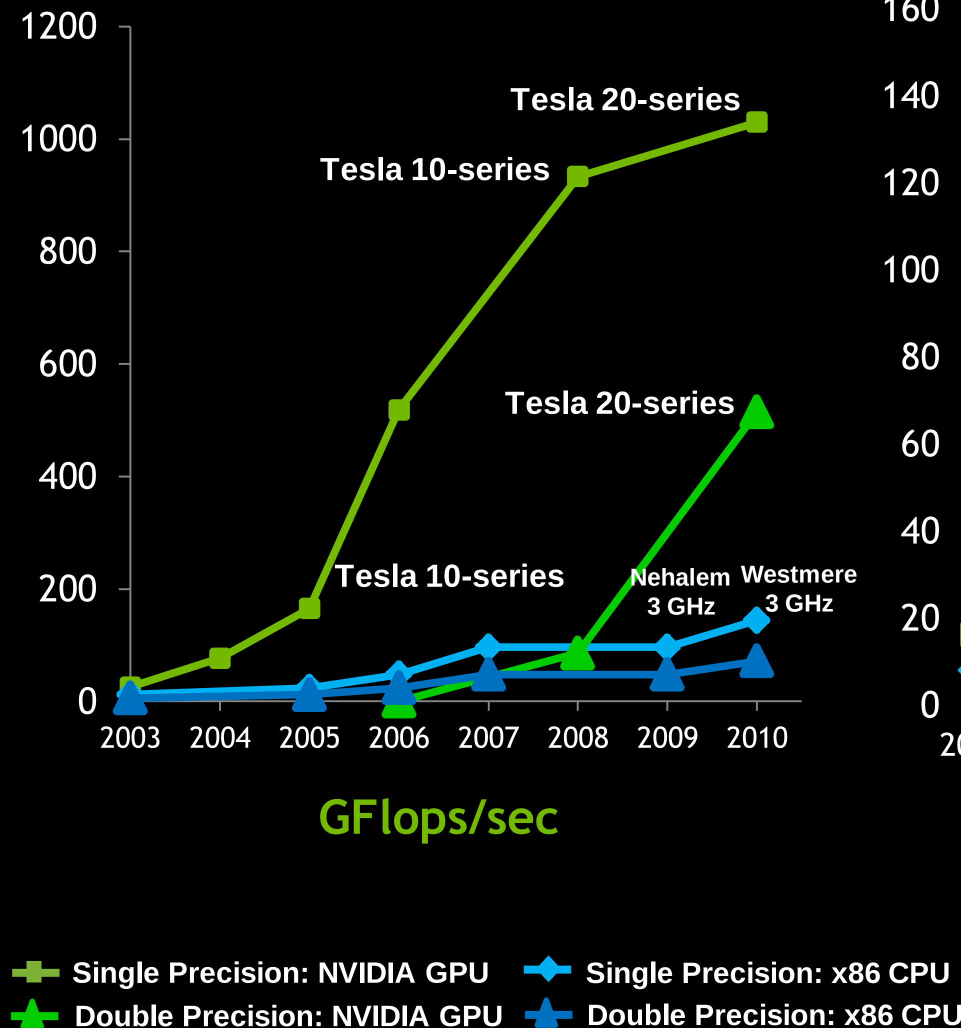


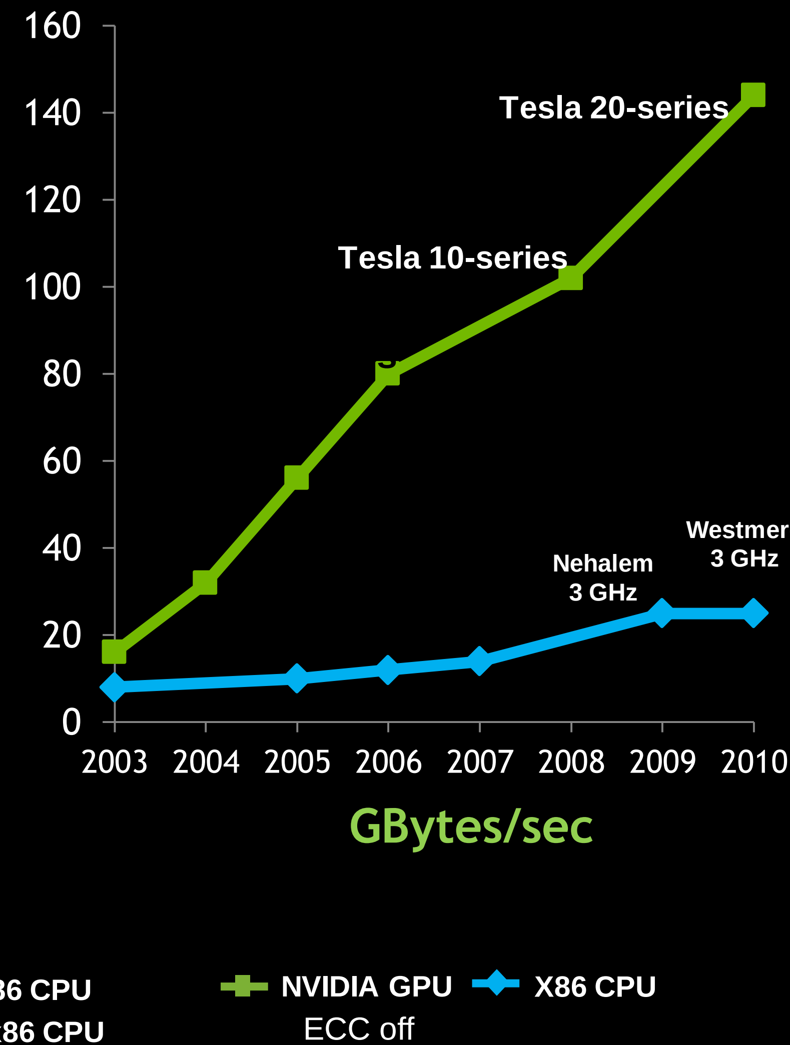
The "New" Moore's Law

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

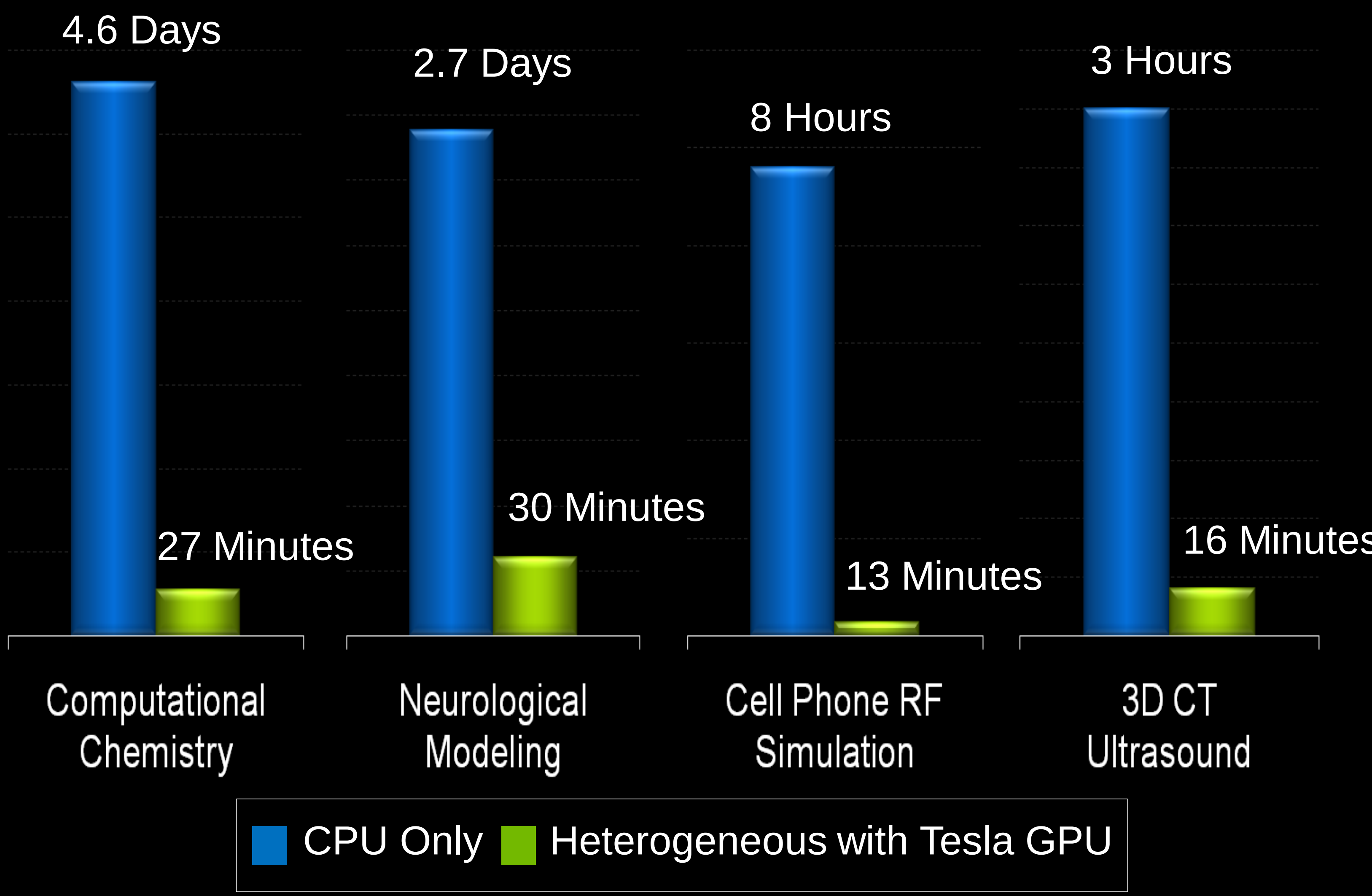
The World's Programmers

Why GPU Computing?





Accelerating Insight



GPU TECHNOLOGY CONFERENCE

OCTOBER 11-14, 2011 | SAN JOSE, CALIFORNIA

GPU Computing:

A sampling of “killer” apps

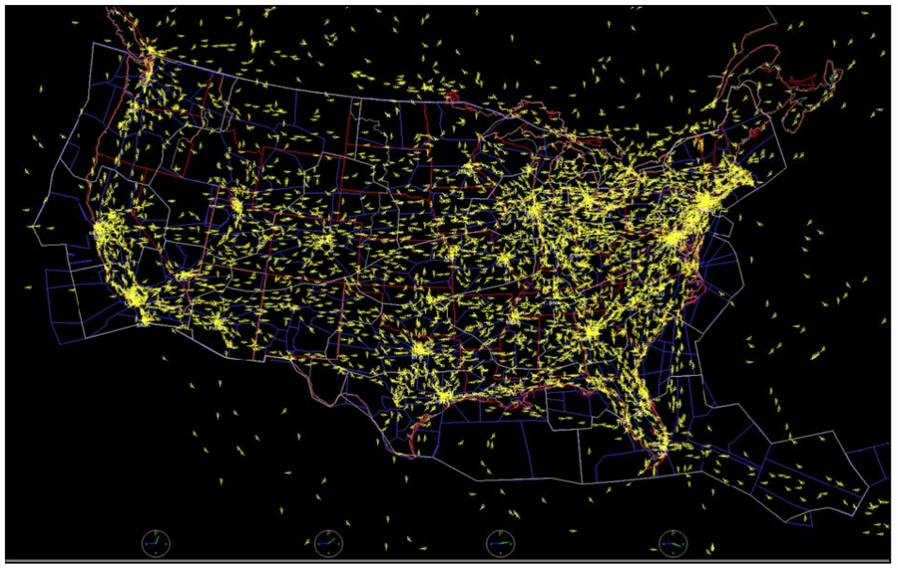
Tracking Space Junk

- Air Force monitors 19,000 pieces of space debris

- Even a paint flake can destroy spacecraft
- 21x CUDA speedup – narrow uncertainty bands and reduce false alarms

Modeling Air Traffic

- Air traffic is increasing
- Predictive modeling can avoid airport overloading
- Variables: flight paths, air speed, altitude, descent rates
- NASA ported their model to CUDA
- 10 minute process reduced to 3 second



Detecting IEDs

12 mph
CPU

77 mph
GPU

Reducing Radiation from CT Scans

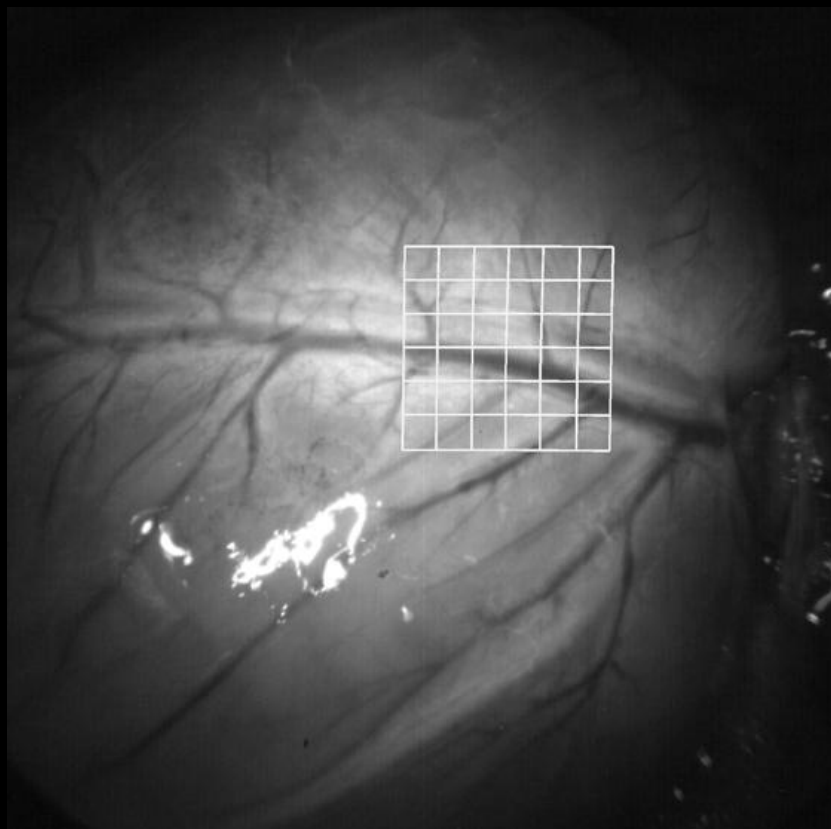
28,000 people/year develop cancer from CT scans

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

CPUs: 2 hours (unusable)

CUDA: 2 minutes (clinically practical)

Operating on a Beating Heart



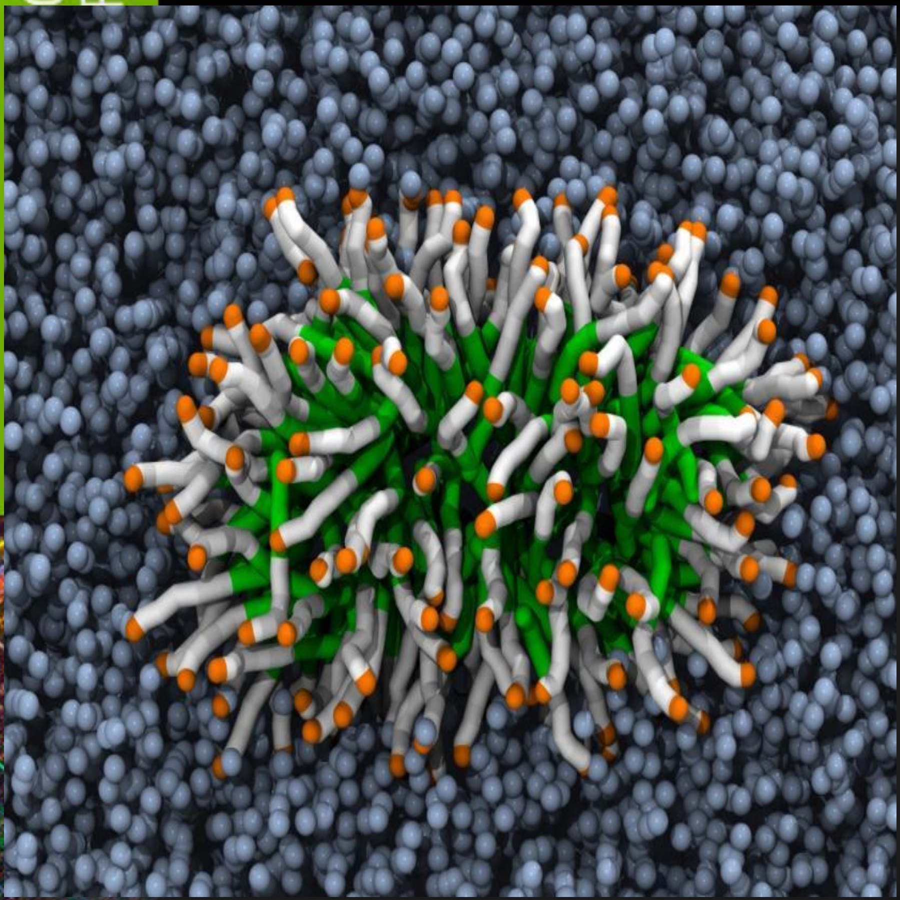
Only 2% of surgeons can operate on a beating heart

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

GPU TECHNOLOGY CONFERENCE

Cleaning Cotton

USDA
United States Department of Agriculture

Problem: Cotton is over-cleaned, causing fiber damage

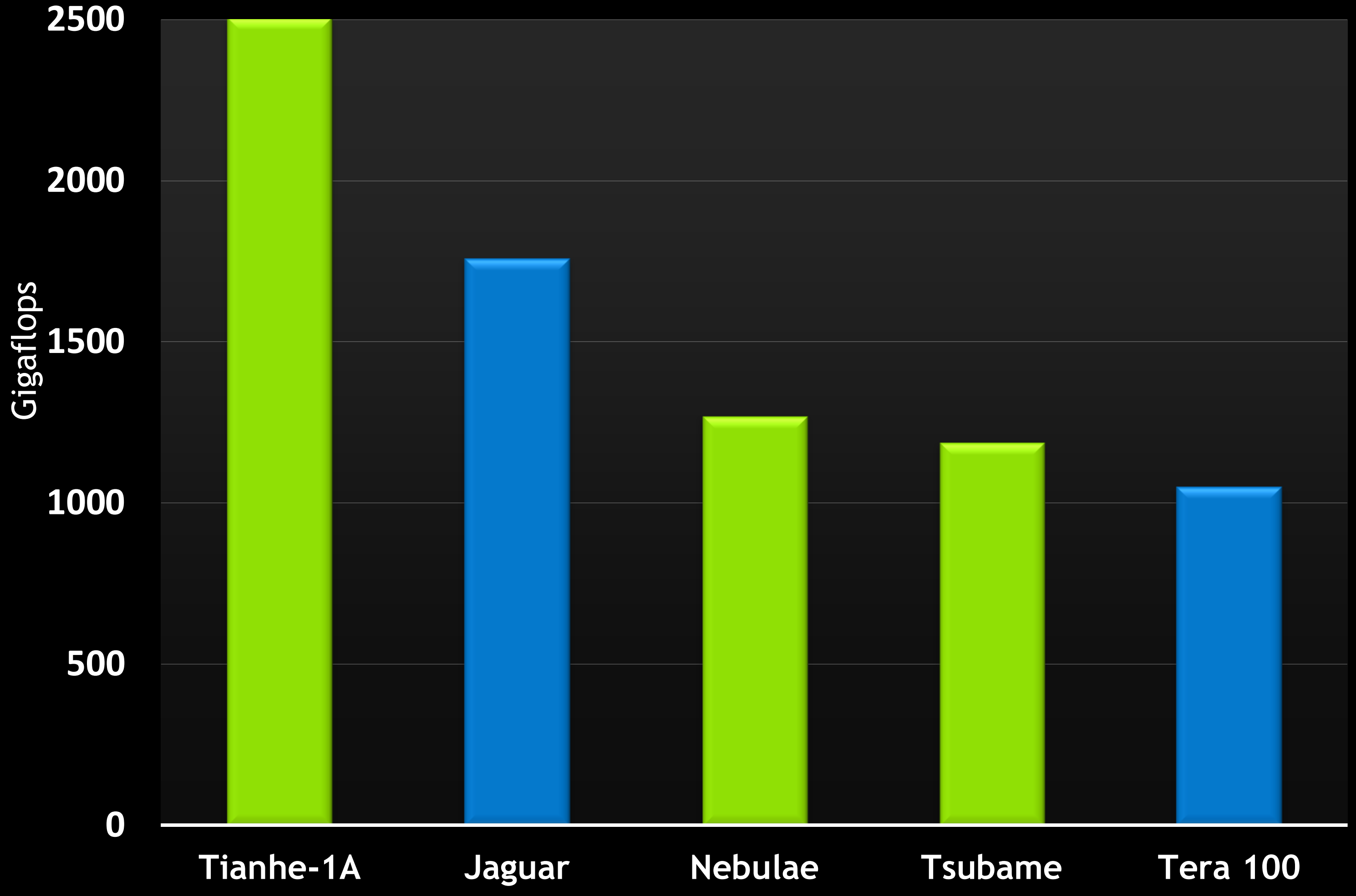
GPU-based machine vision enables real-time feedback during cleaning

96% lower fiber damage

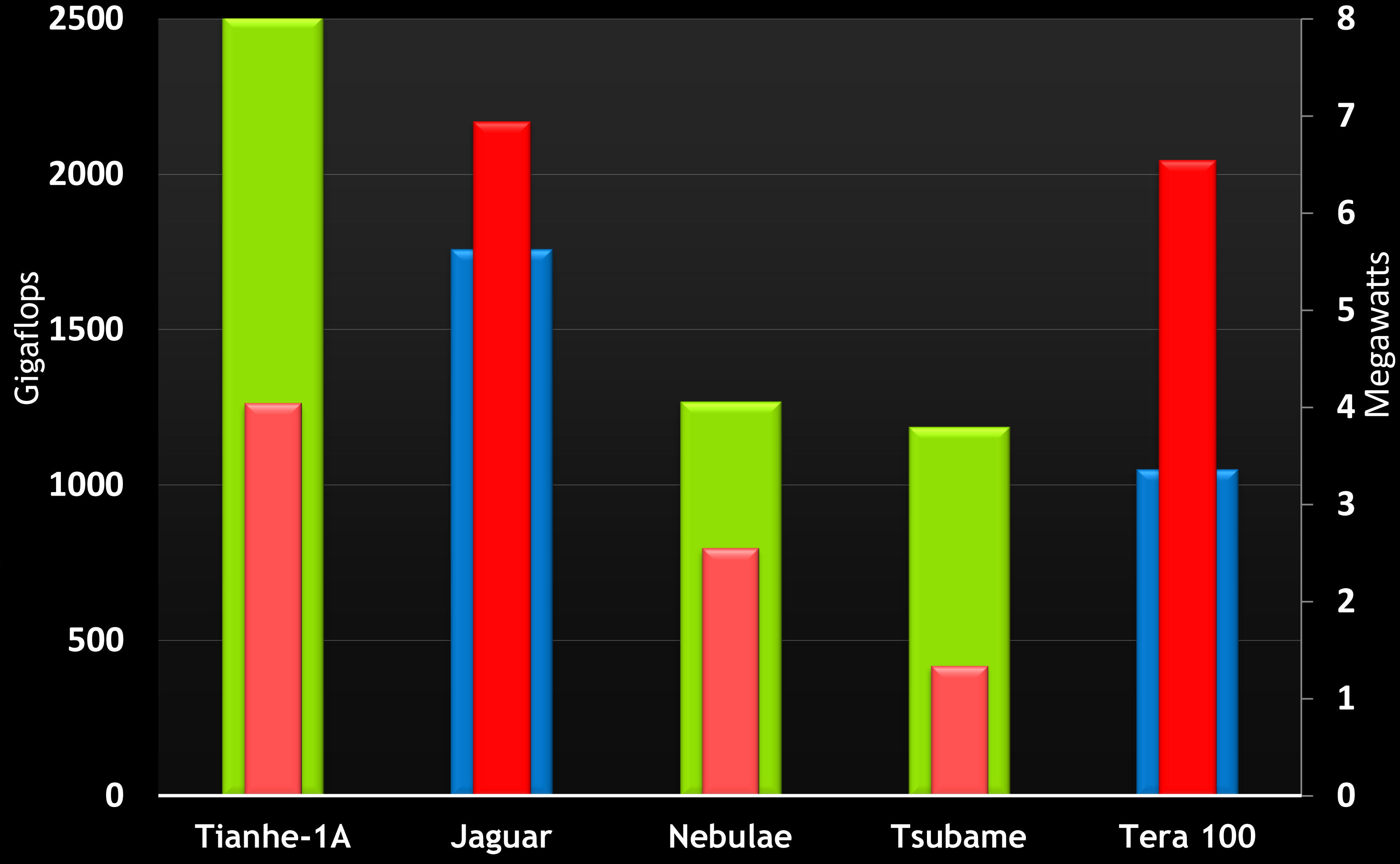
$100M additional potential revenue

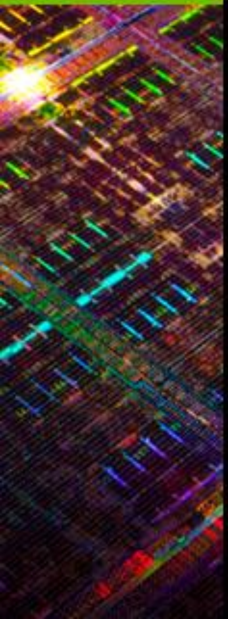
“Parallel Algorithm for GPU Processing; for use in High Speed Machine Vision Sensing of Cotton Lint Trash”, Mathew G. Pelletier, February 8, 2008

GPUs Power 3 of the Top 5…



…Using Less Power

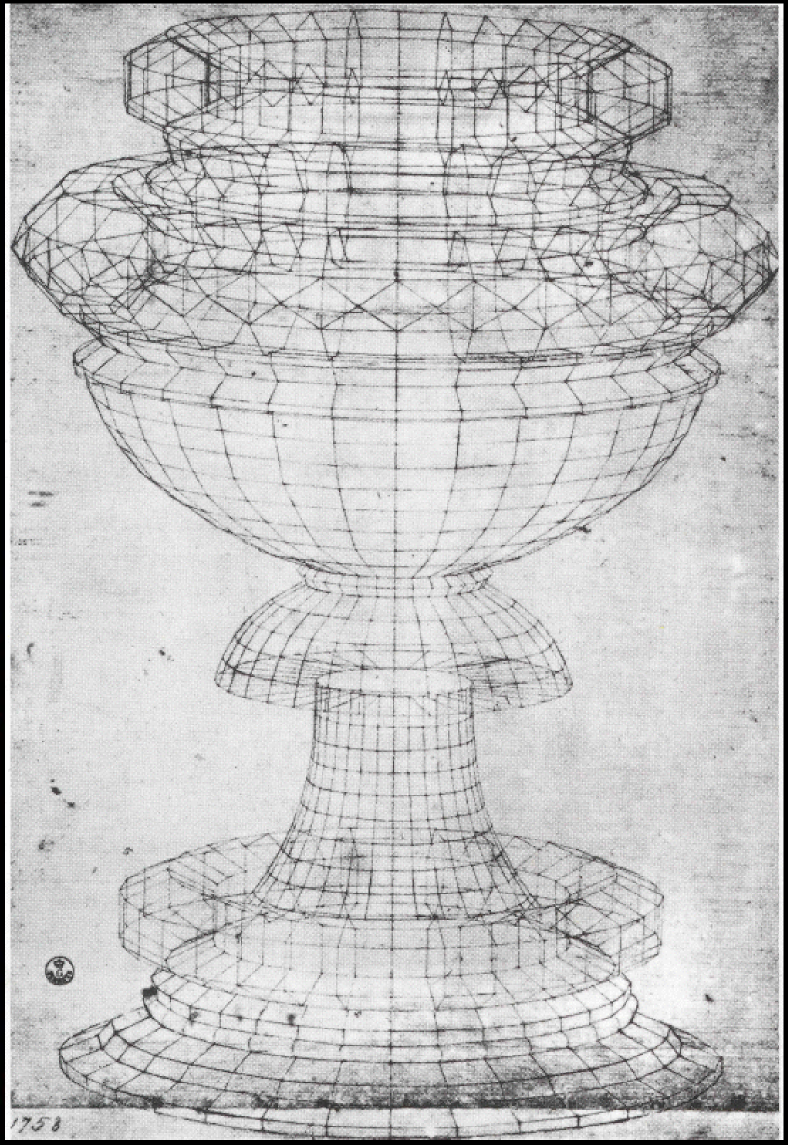




Heritage:

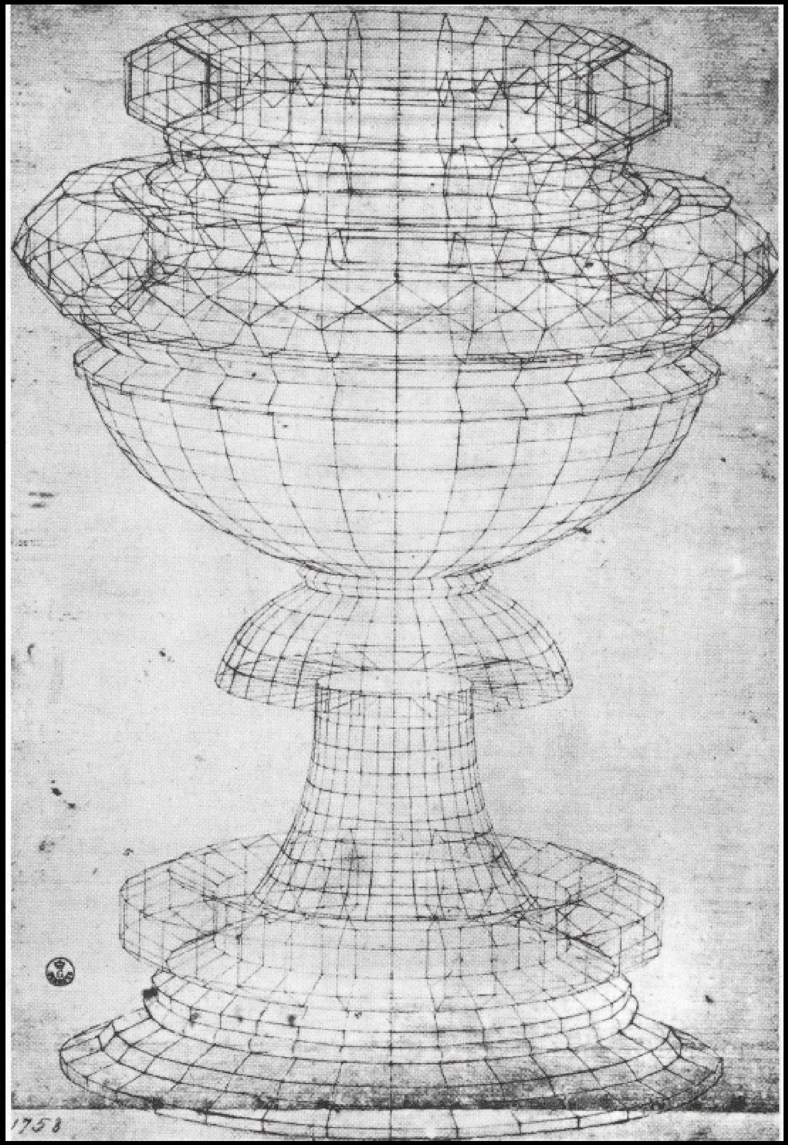
Graphics Hardware

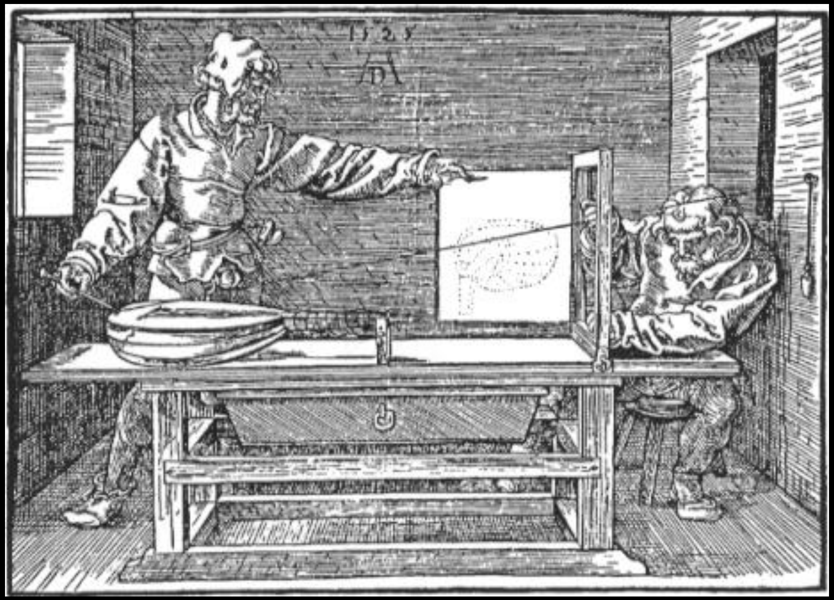
Early 3D Graphics



Perspective study of a chalice
Paolo Uccello, circa 1450

Early Graphics Hardware





Artist using a perspective machine
Albrecht Dürer, 1525

Perspective study of a chalice
Paolo Uccello, circa 1450

Early *Electronic* Graphics Hardware

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

The Graphics Pipeline

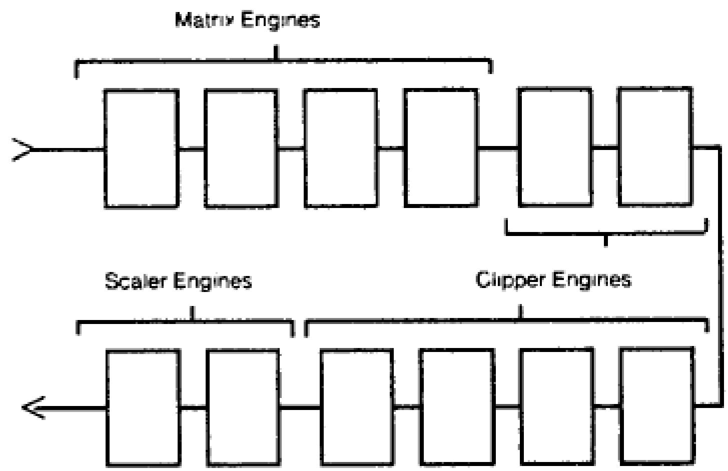


Figure 3: Geometry System; each block is a Geometry Engine.

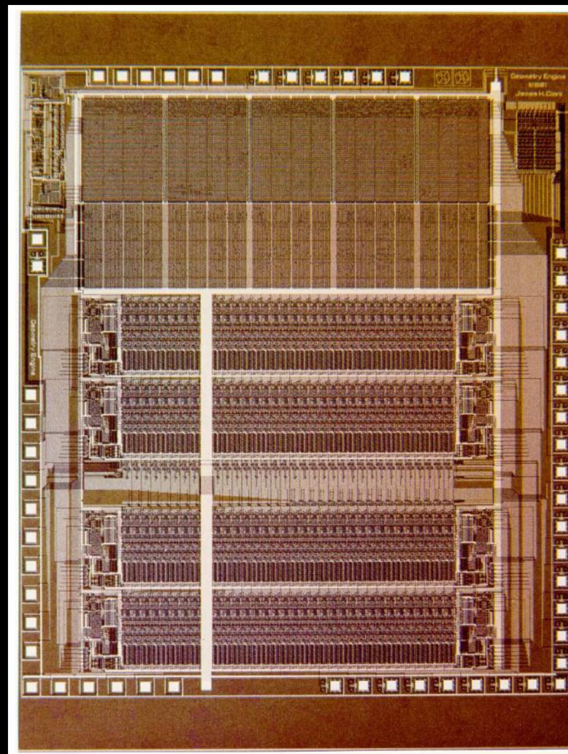
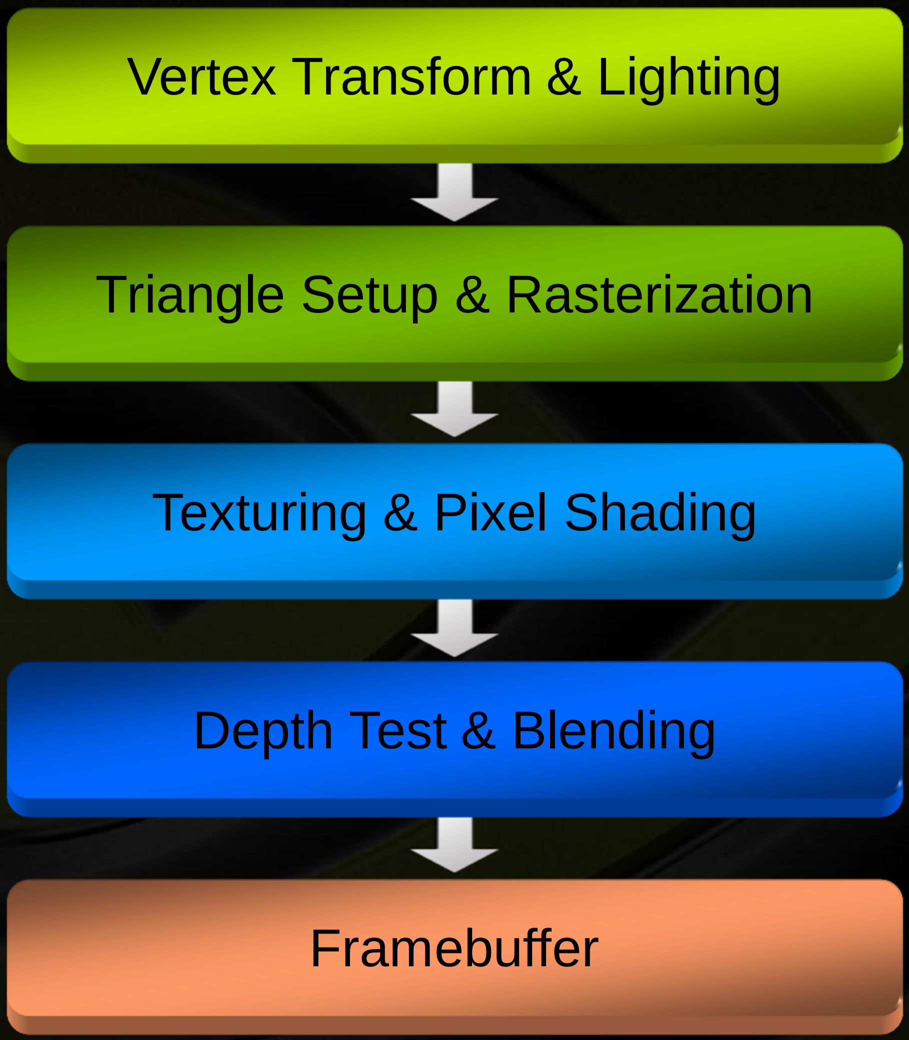


Figure 2: Photograph of the Geometry Engine.

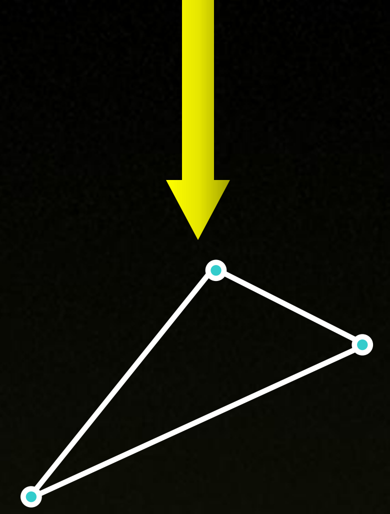
The Geometry Engine: A VLSI Geometry System for Graphics
Jim Clark, 1982

The Graphics Pipeline

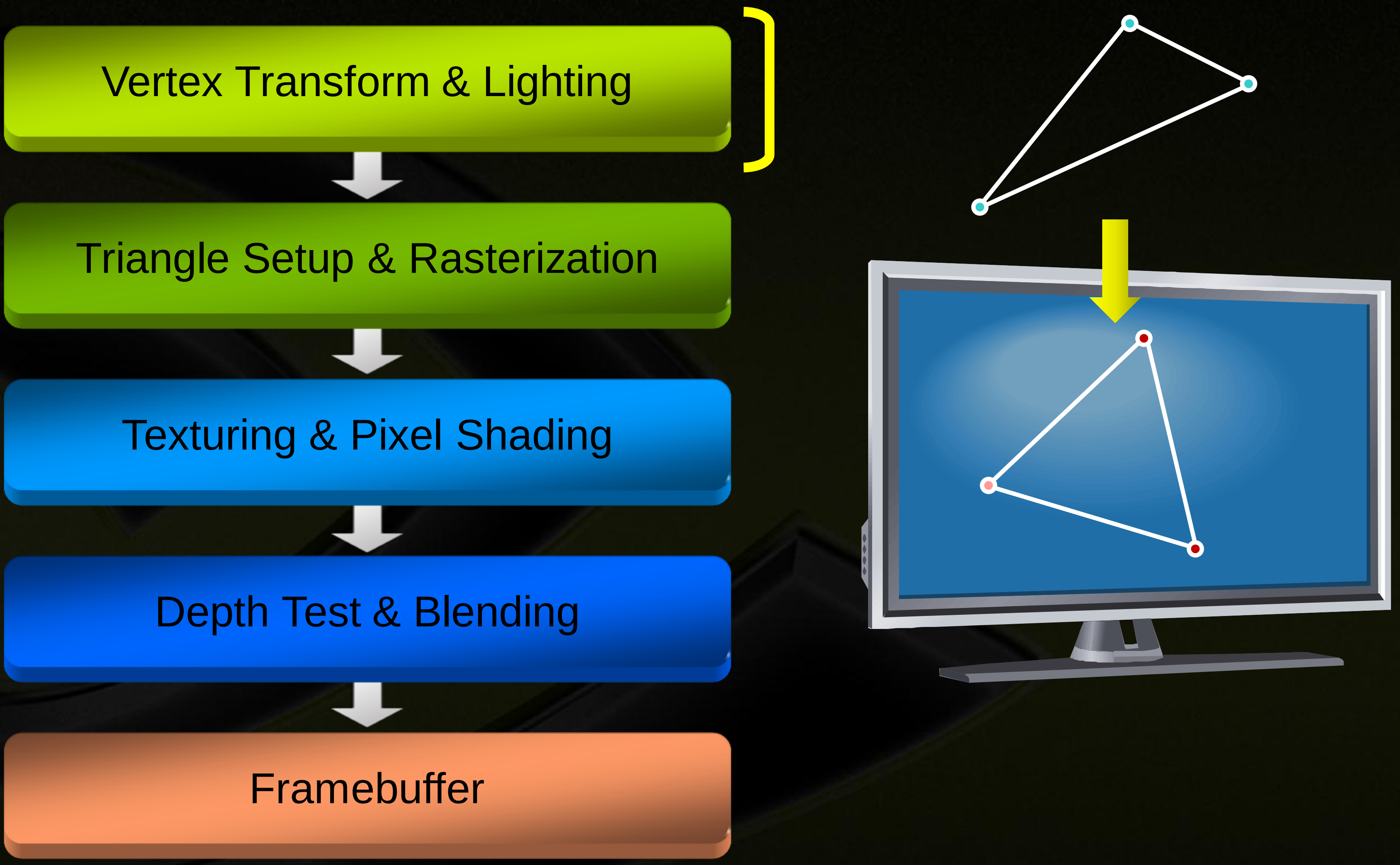


The Graphics Pipeline

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



The Graphics Pipeline



The Graphics Pipeline

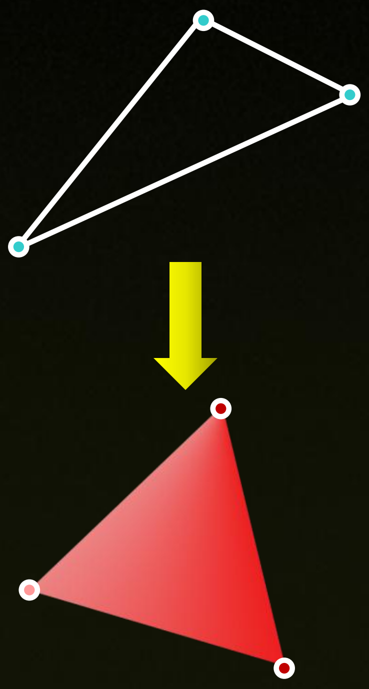
Vertex Transform & Lighting

Triangle Setup & Rasterization

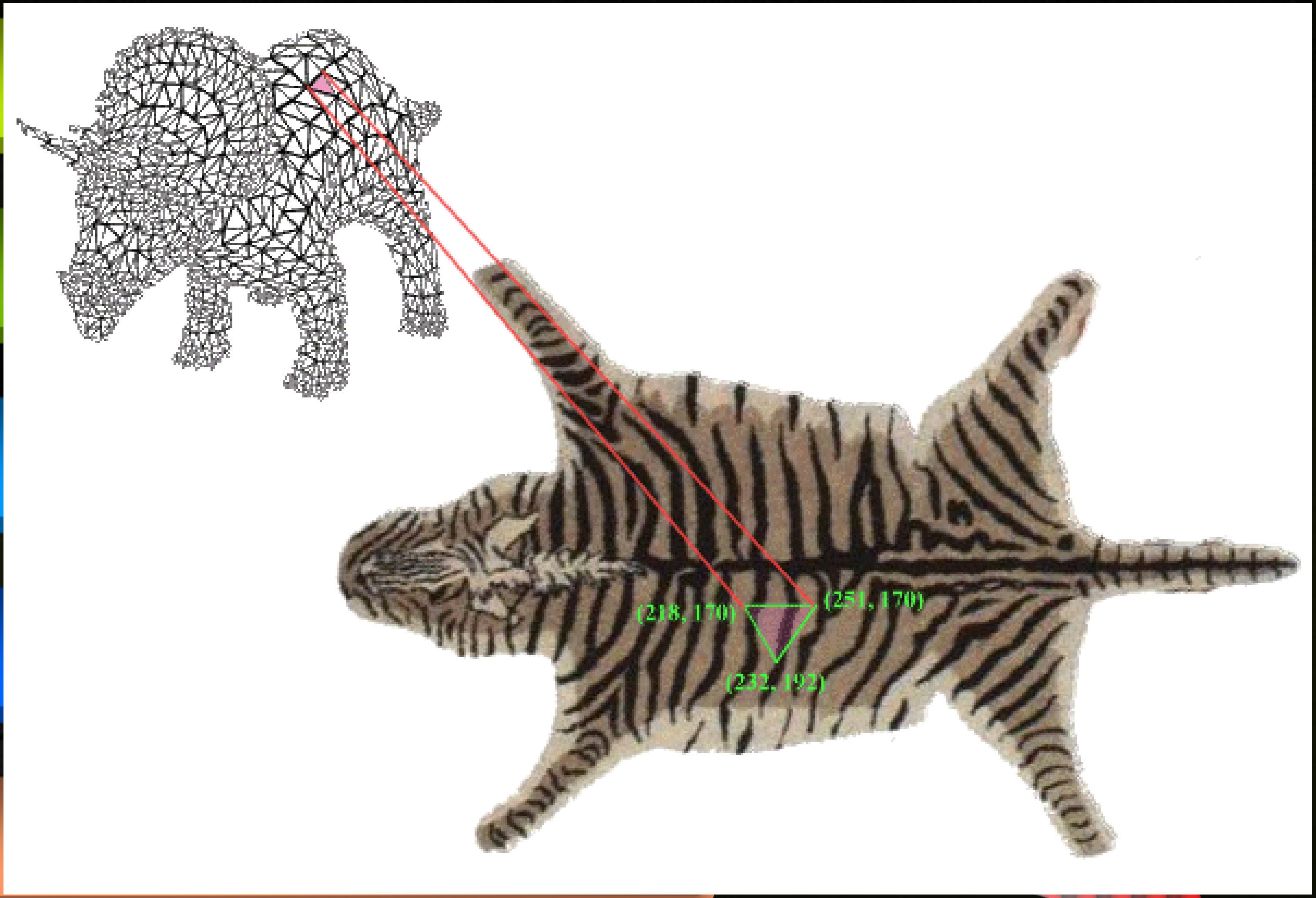
Texturing & Pixel Shading

Depth Test & Blending

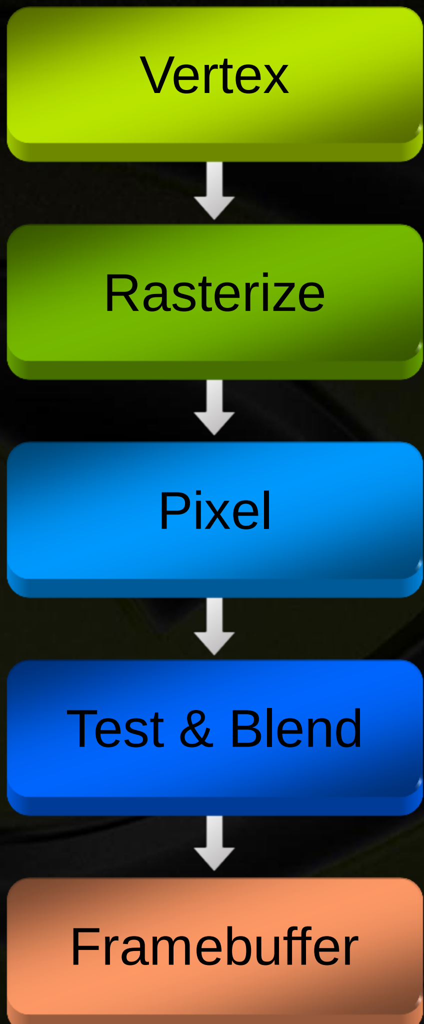
Framebuffer



The Graphics Pipeline

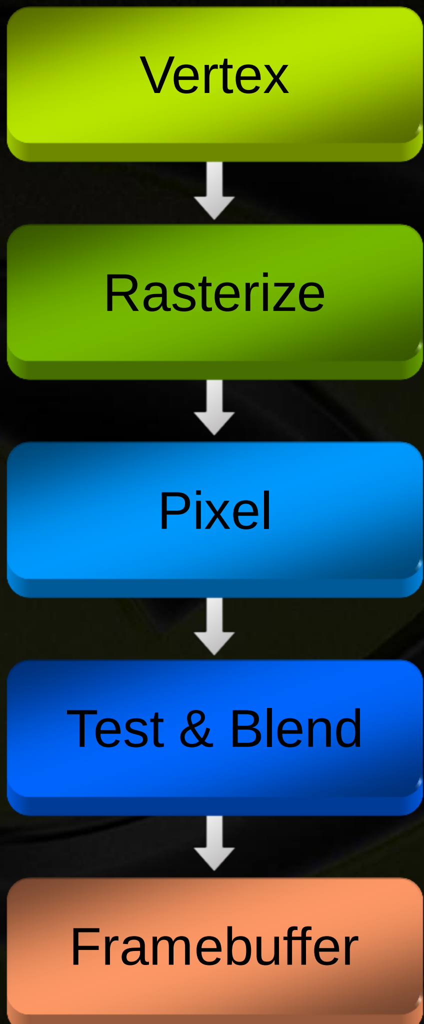


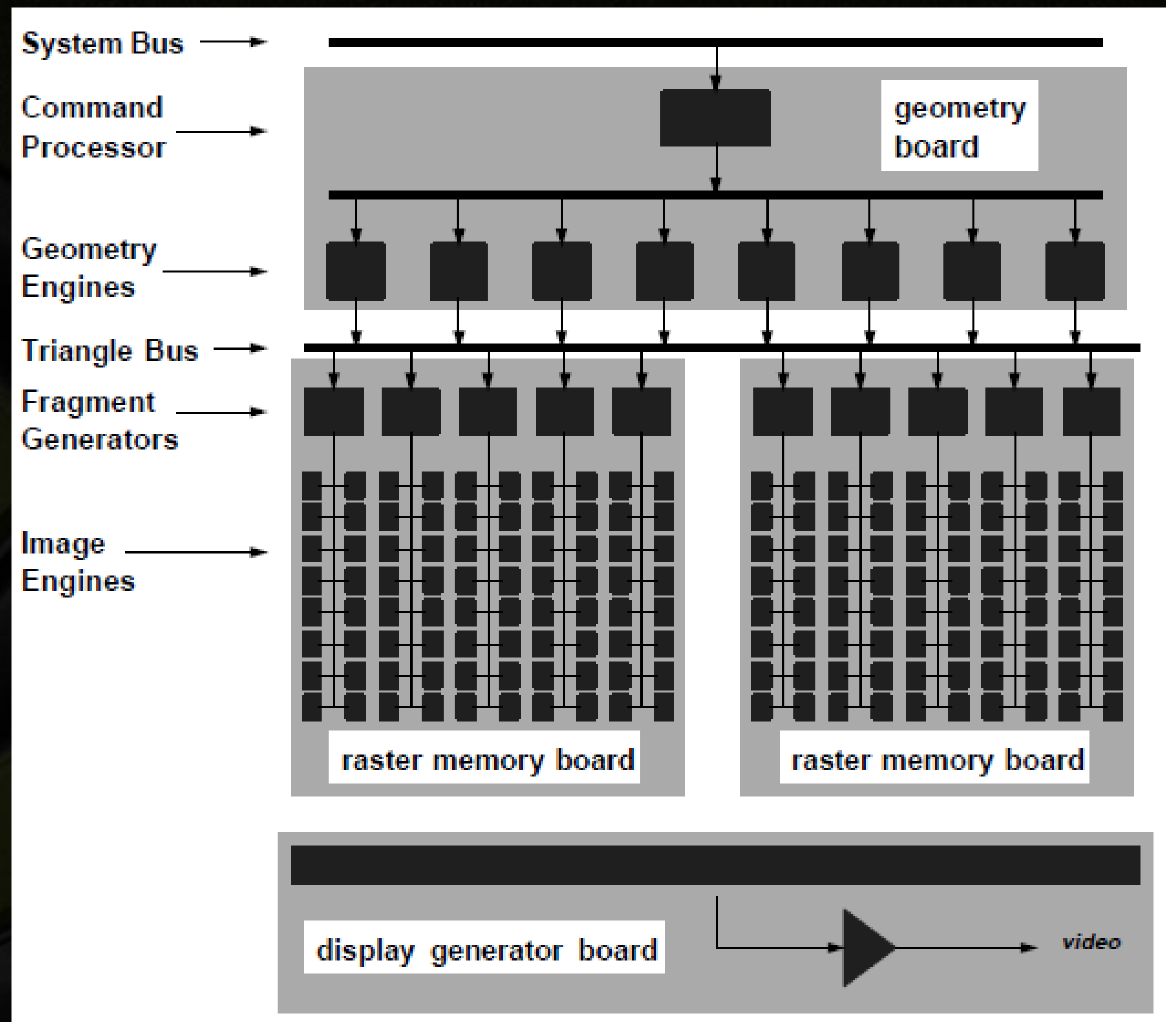
The Graphics Pipeline



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

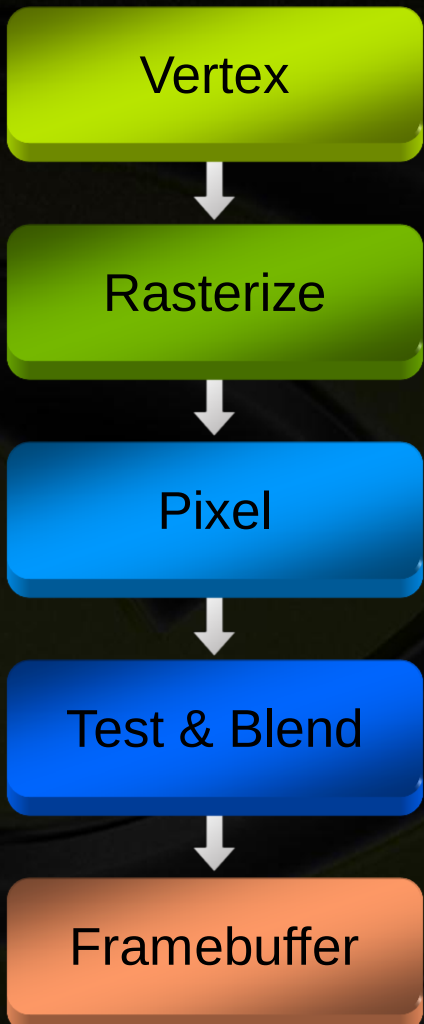
SGI RealityEngine (1993)

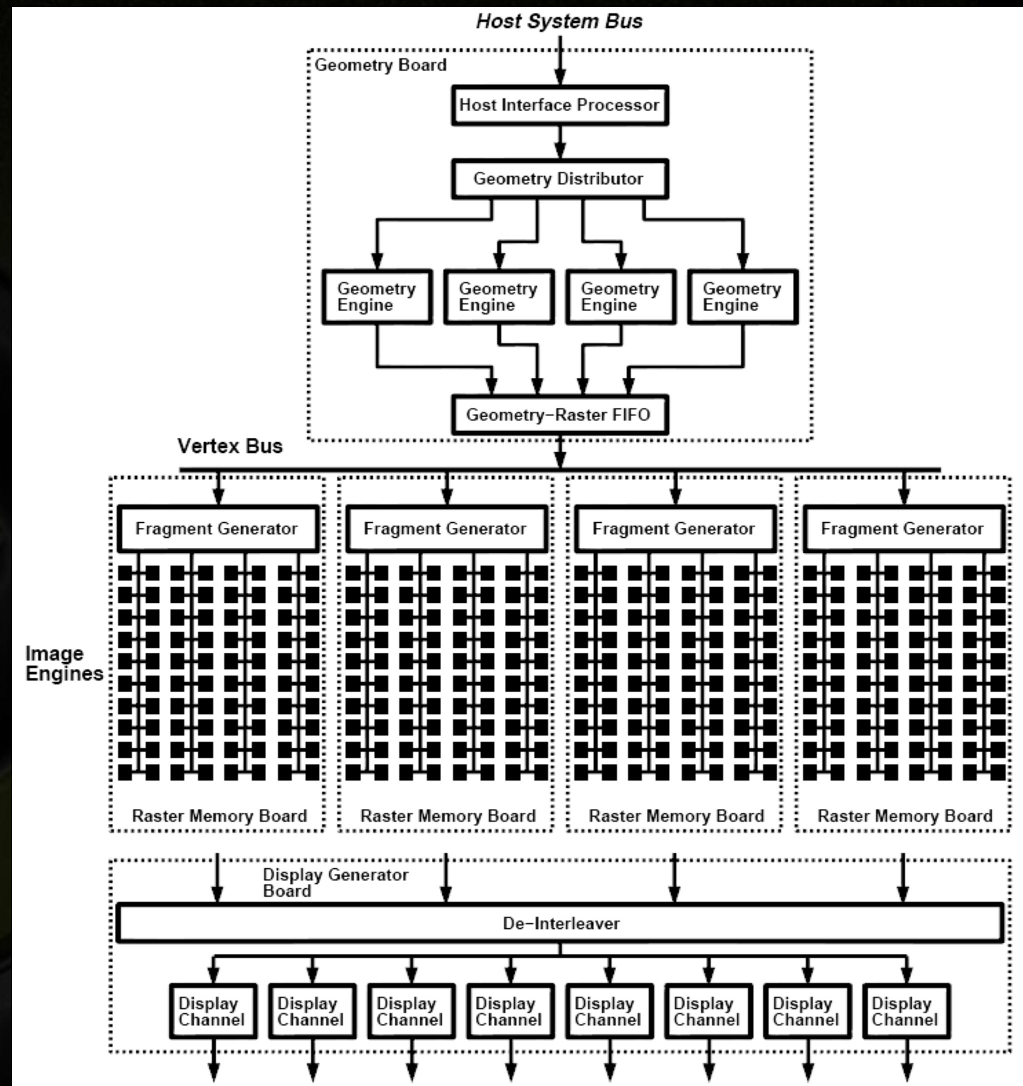




RealityEngine Graphics
Kurt Akeley , SIGGRAPH 93

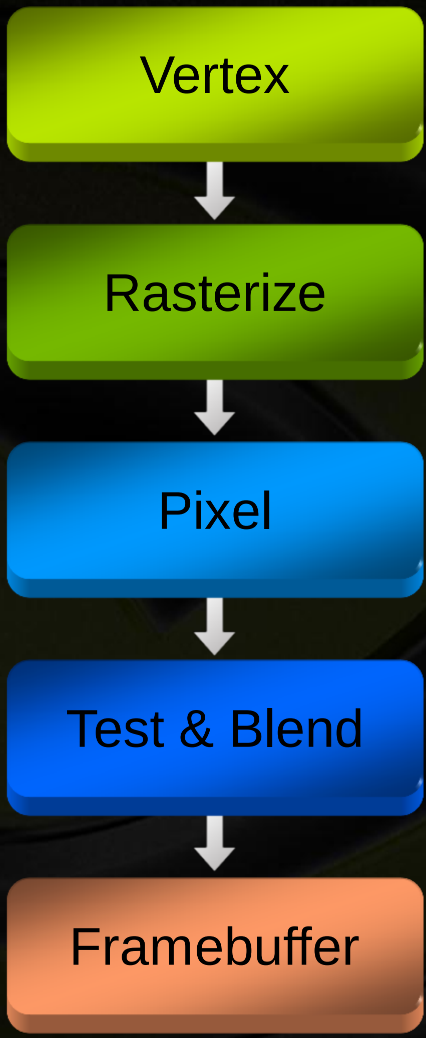
SGI InfiniteReality (1997)





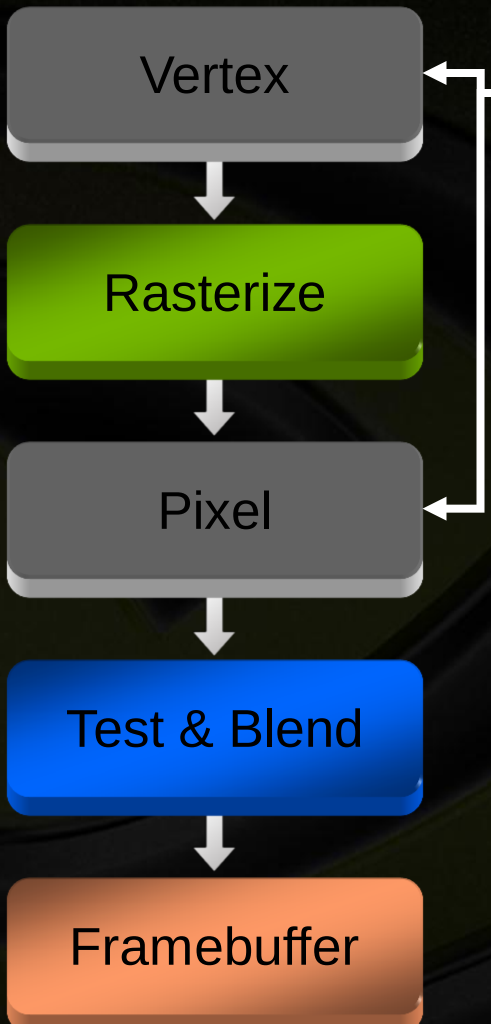
InfiniteReality: A real-time graphics system
Montrym et al., SIGGRAPH 97

The Graphics Pipeline



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

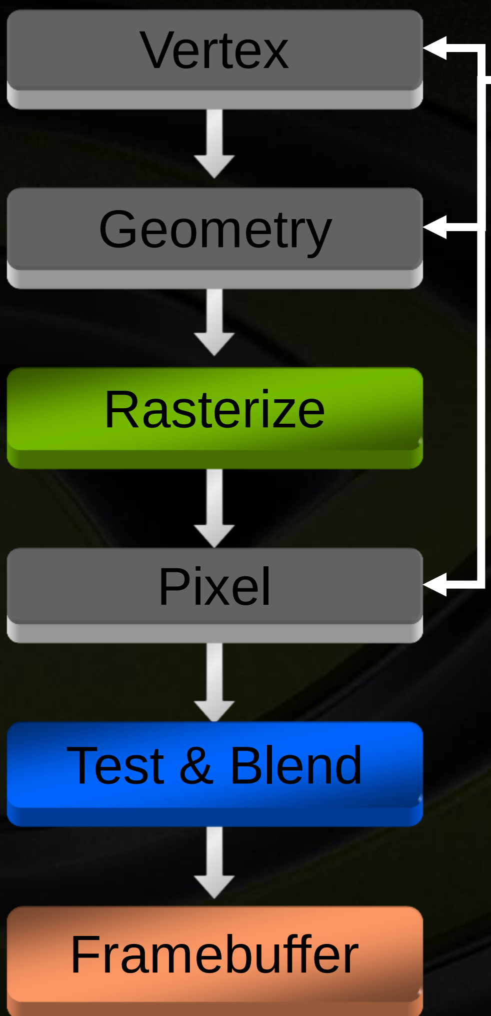
The Graphics Pipeline




```
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**

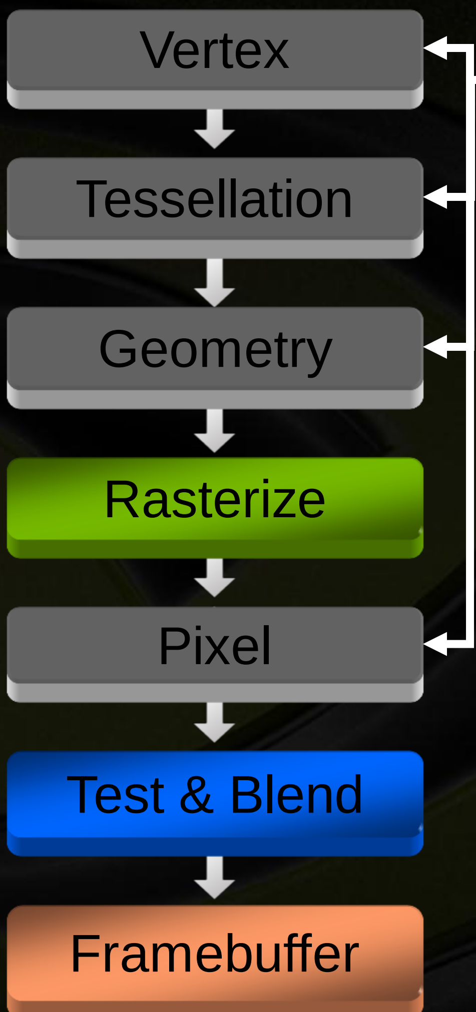
The Graphics Pipeline




```
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**

The Graphics Pipeline

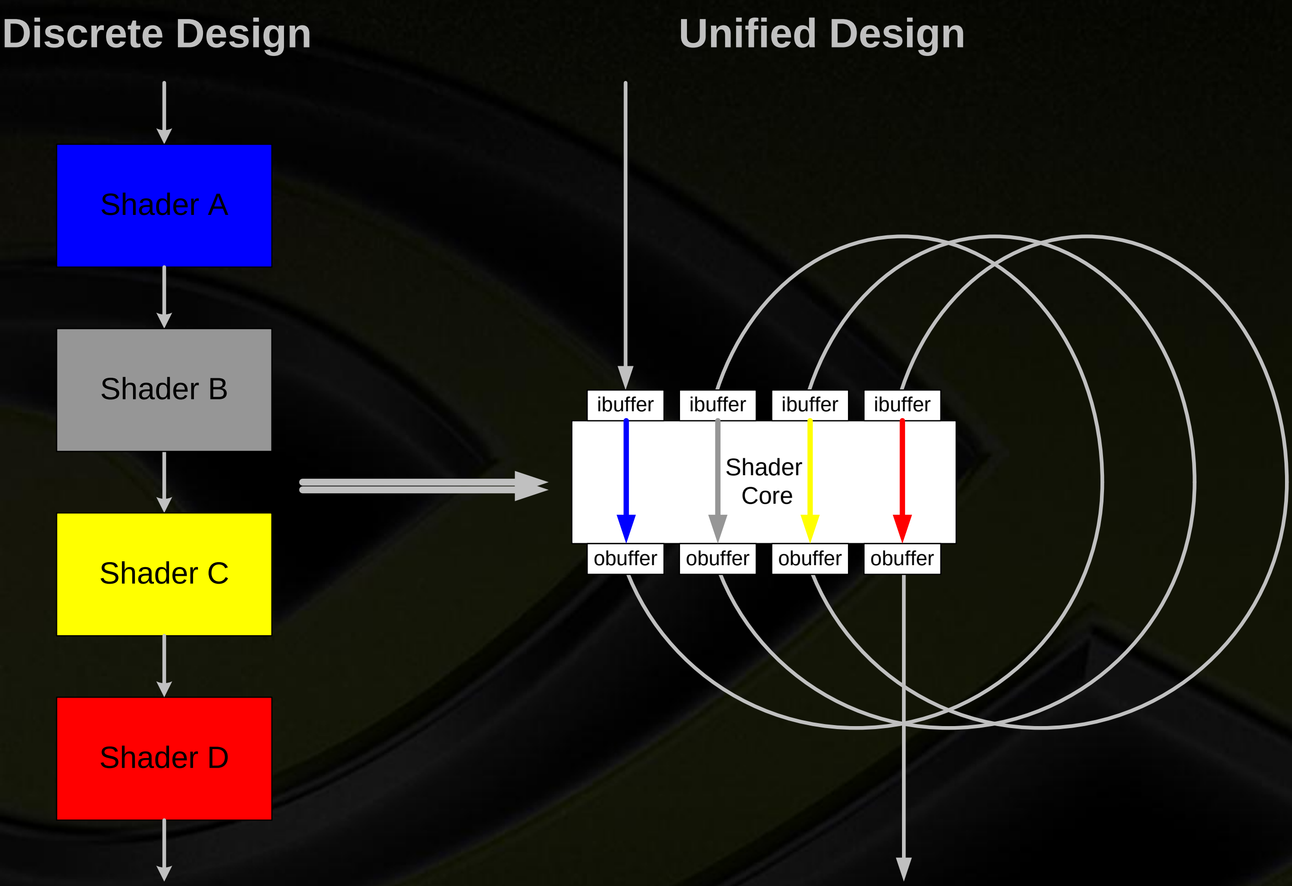



```
pixel_out
main(uniform sampler2D texture : TEXUNIT 0, pixel_in
{
  pixel_out OUT;
  float d= clamp(1.0 - pow(dot(IN.lightdist, IN.light
  float3 color = tex2D(texture, IN.texcoord).rgb;
  OUT.color = color * (d + 0.4);
  return OUT;
}
```

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

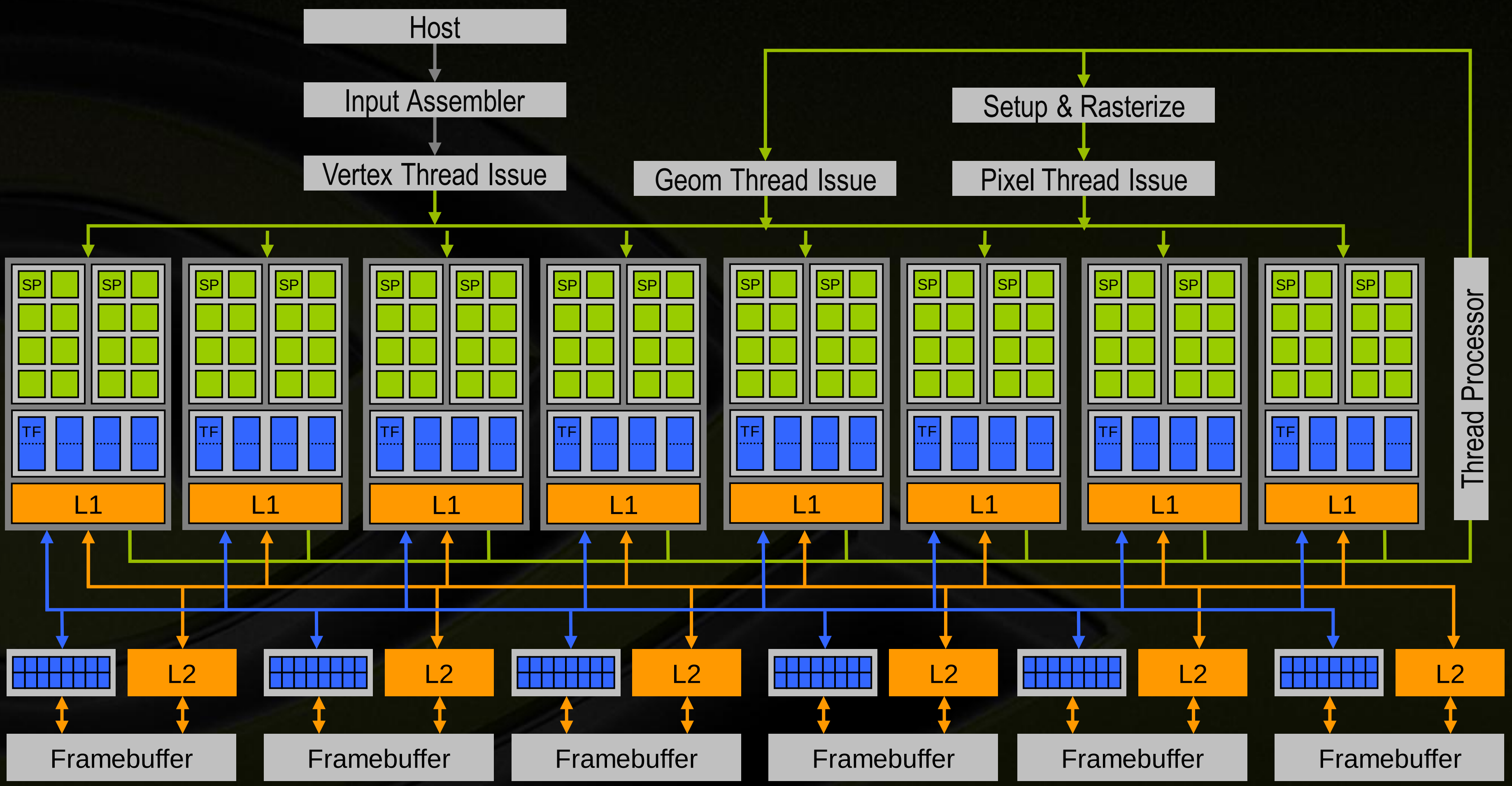
GPU architecture increasingly centers around shader execution

Modern GPUs: Unified Design

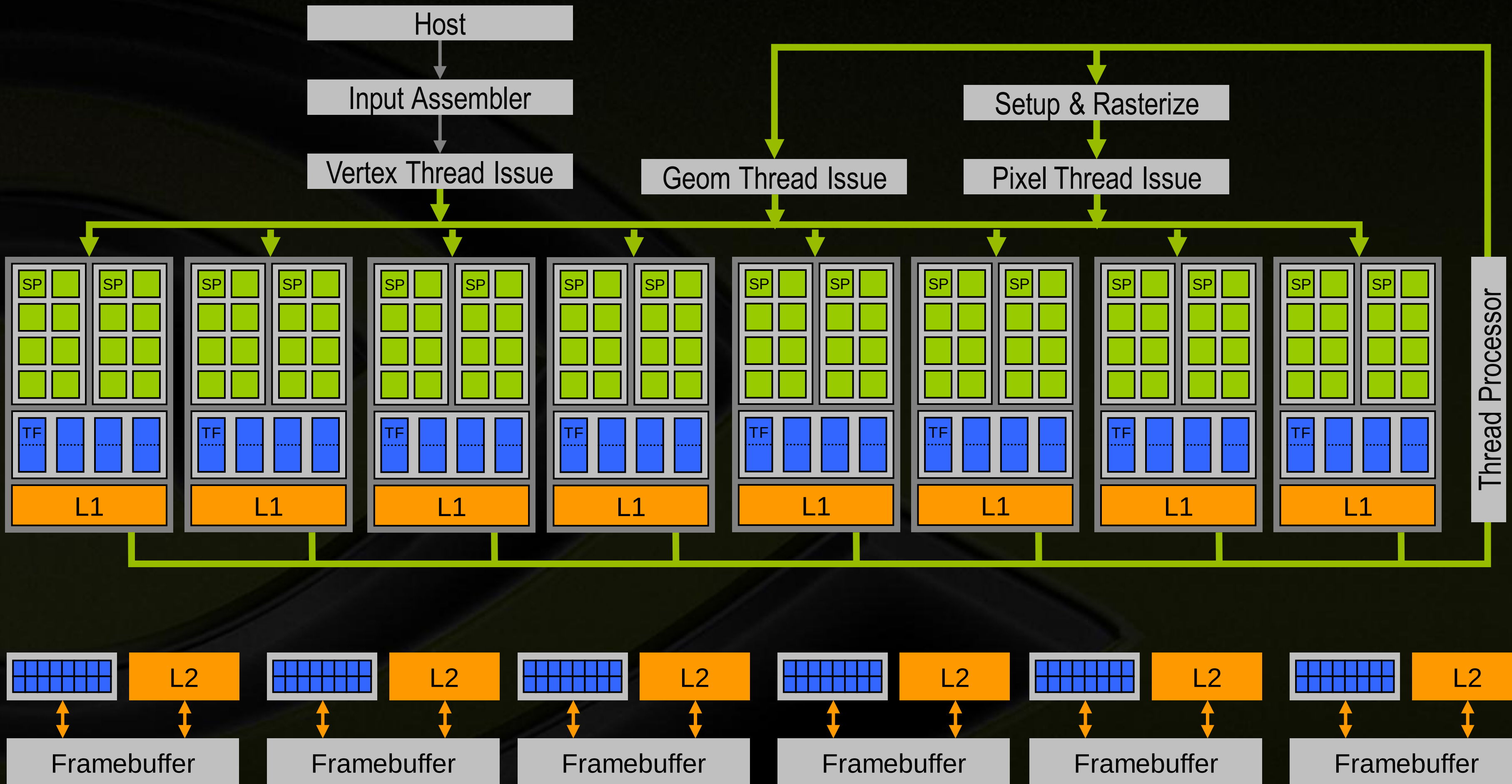


Vertex shaders, pixel shaders, etc. become *threads* running different programs on a flexible core

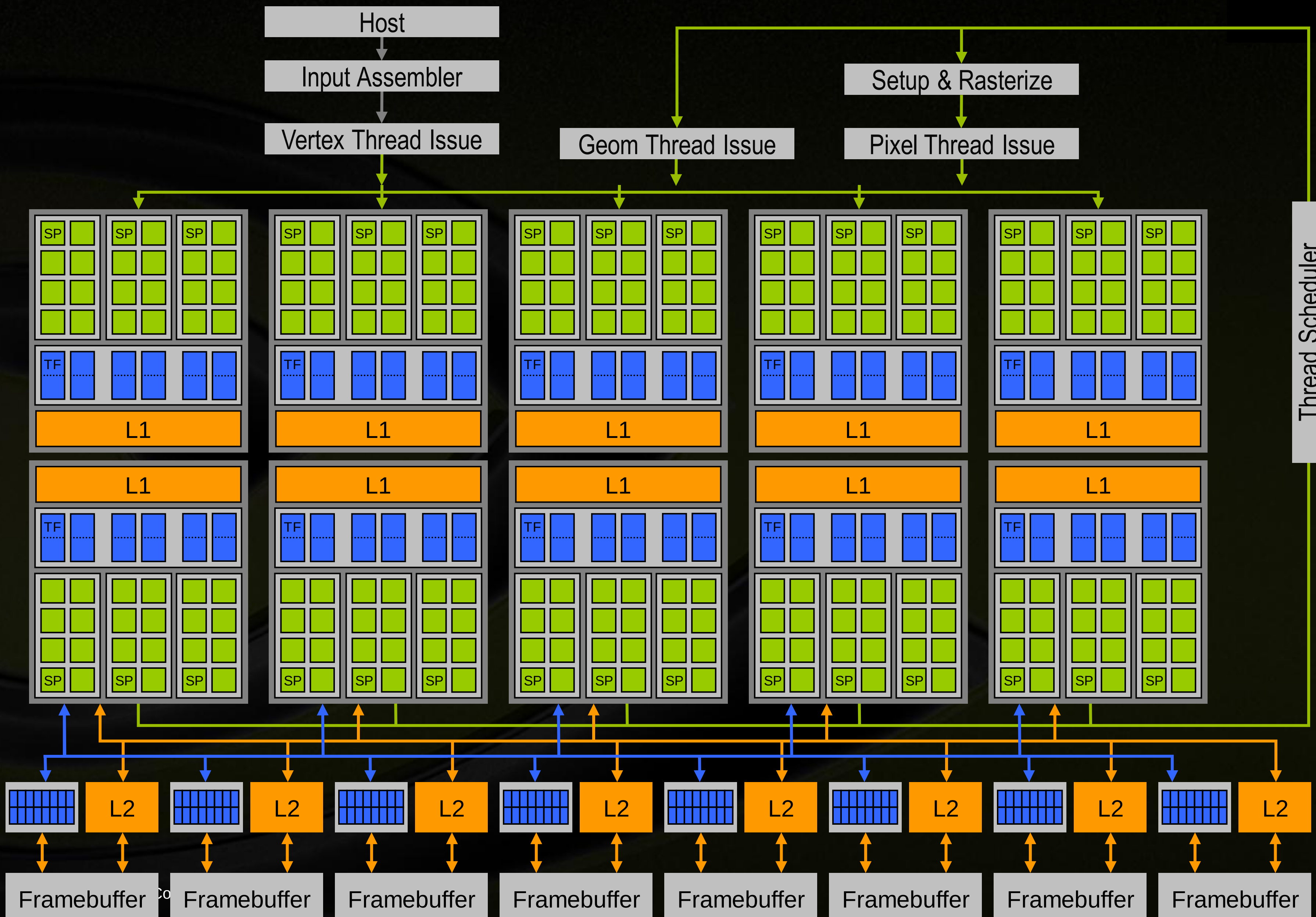
GeForce 8: Modern GPU Architecture



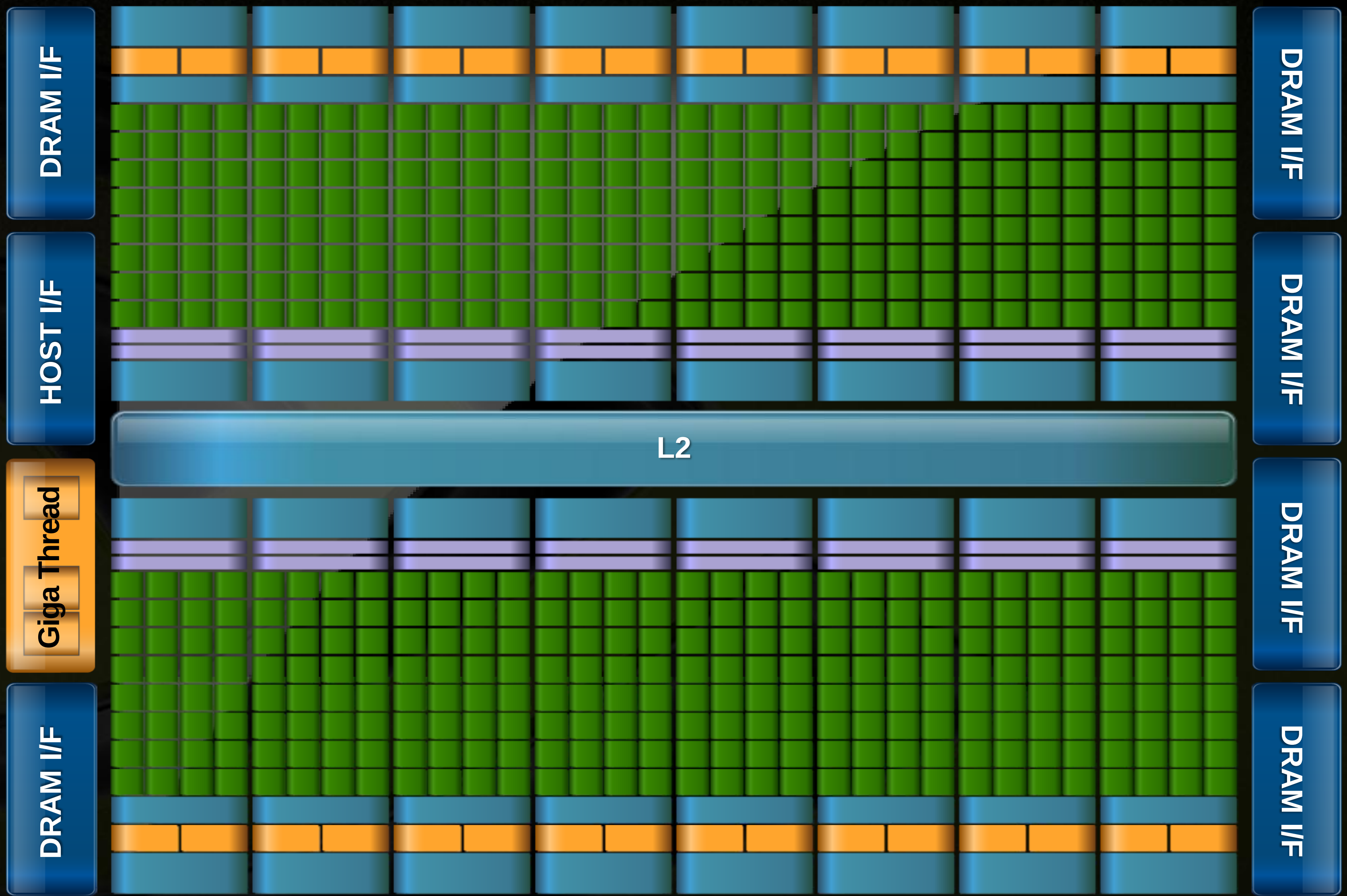
GeForce 8: Modern GPU Architecture



Modern GPU Architecture: GT200



Current GPU Architecture: *Fermi*

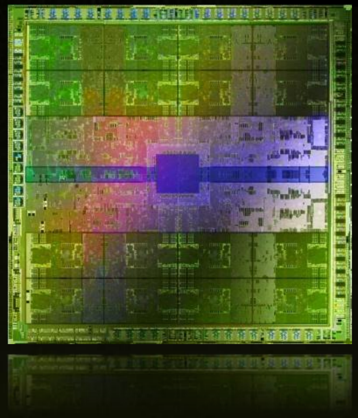


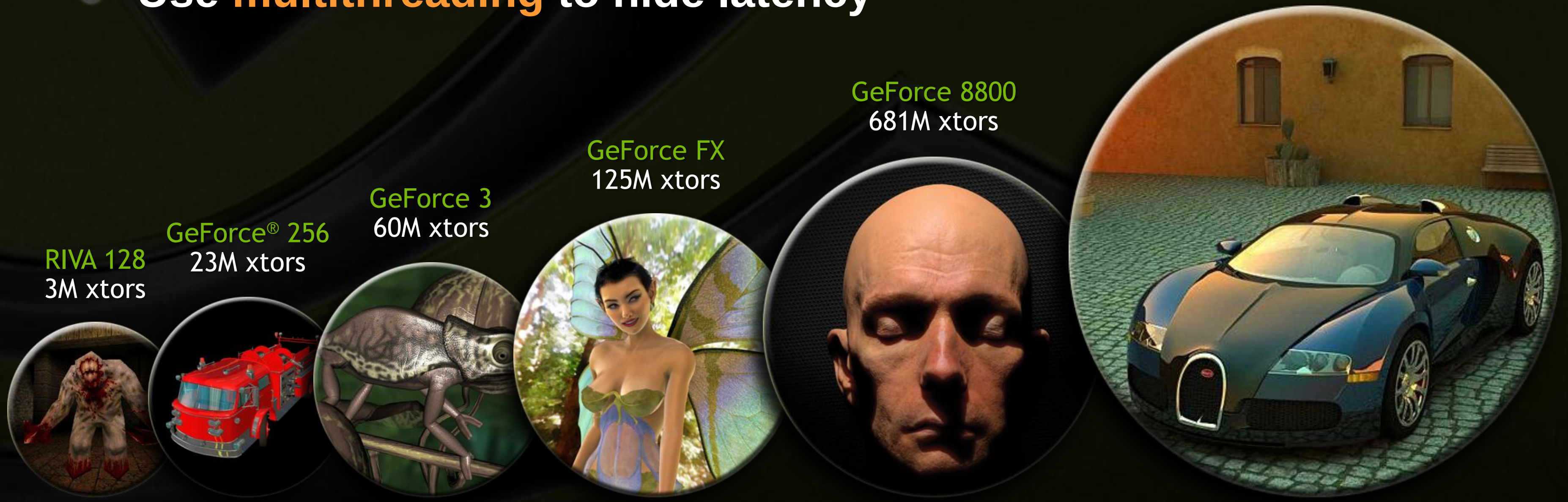
NVIDIA "Fermi" architecture

GPUs Today

Lessons from Graphics Pipeline

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



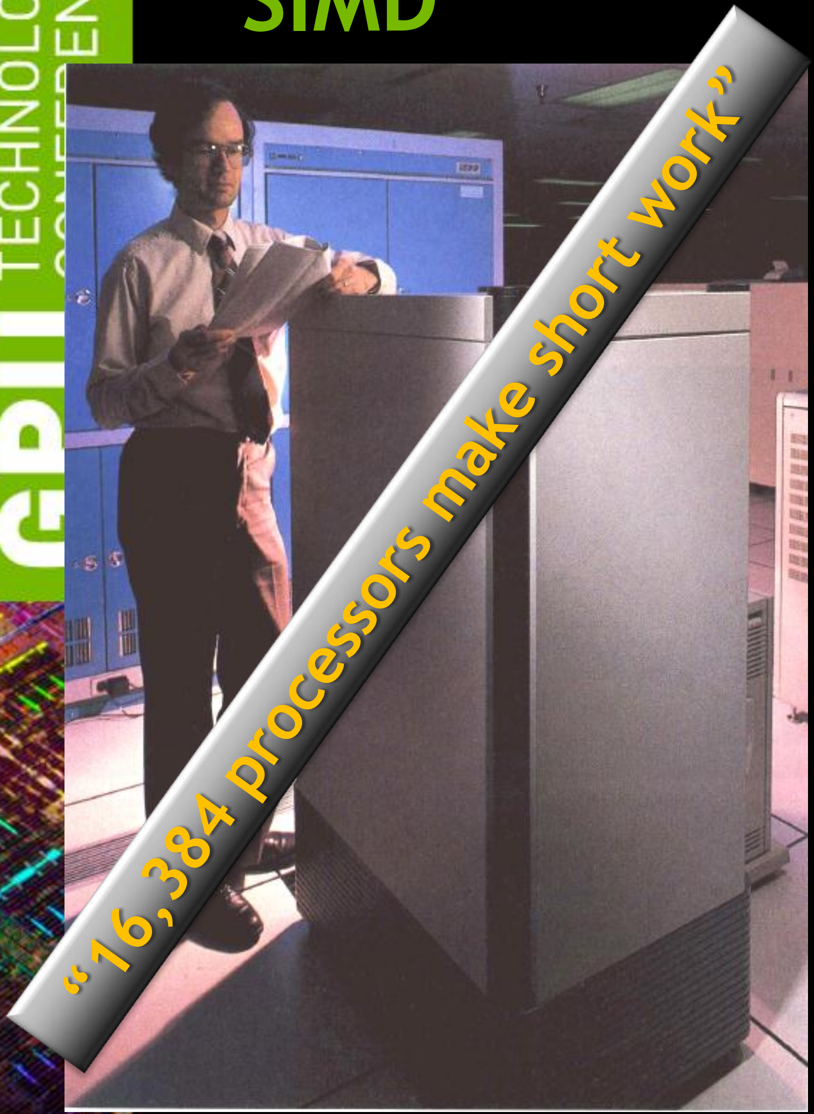


GPU TECHNOLOGY CONFERENCE

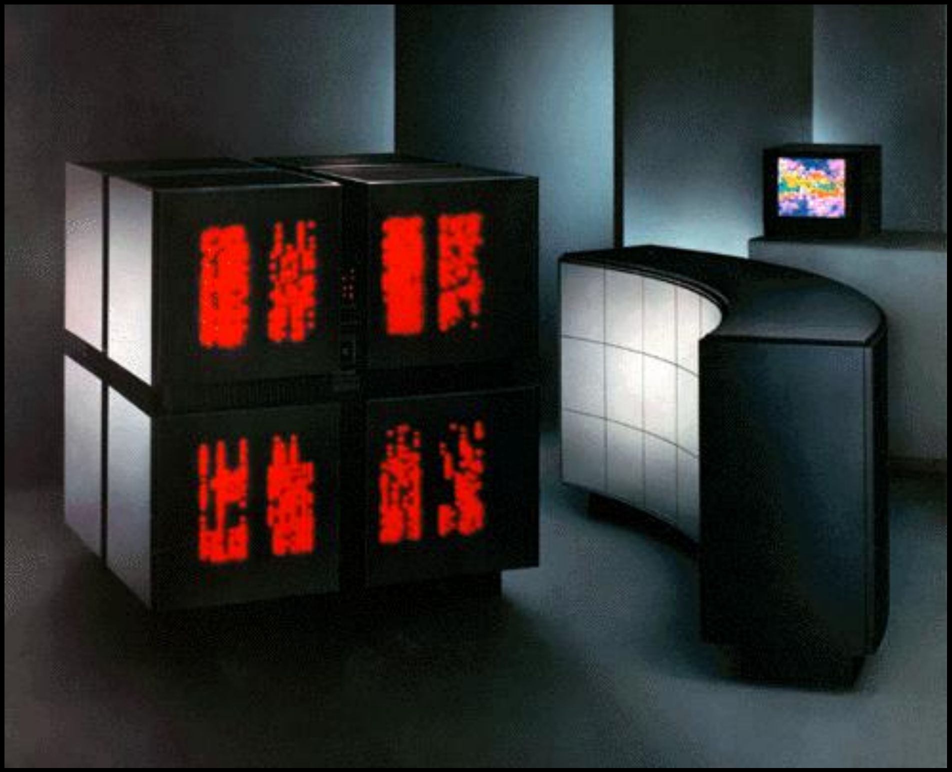
OCTOBER 11-14, 2011 | SAN JOSE, CALIFORNIA

Heritage: Supercomputing

How to build a parallel machine: SIMD



MasPar MP1 (front), Goddard MPP (back)



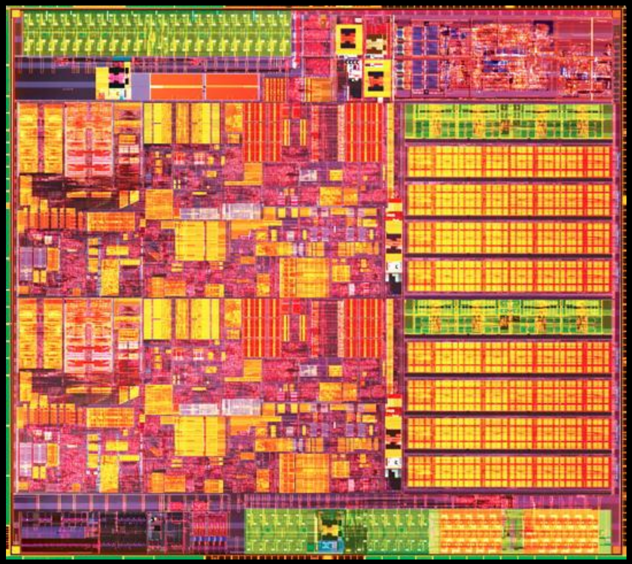
Thinking Machines CM-2

How to build a parallel machine: Hardware Multithreading

Tera MTA

How to build a parallel machine: Symmetric Multiprocessing

SGI Challenge



Intel Core2 Duo

Fermi, Oversimplified

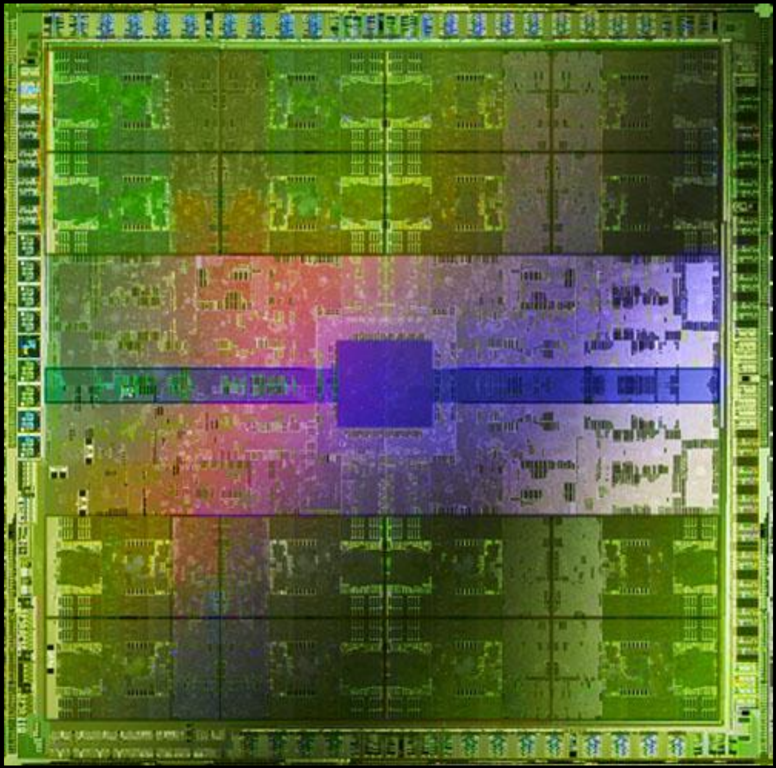
32-wide SIMD (two 16-wide datapaths)

48-way hardware multithreading

× 16-way SMP

24576 threads in flight

@ 512 FMA ops per clock



GPU TECHNOLOGY CONFERENCE
OCTOBER 11-14, 2011 | SAN JOSE, CALIFORNIA

Heritage:

GPU Computing

GPU Computing 1.0: *GPGPU*

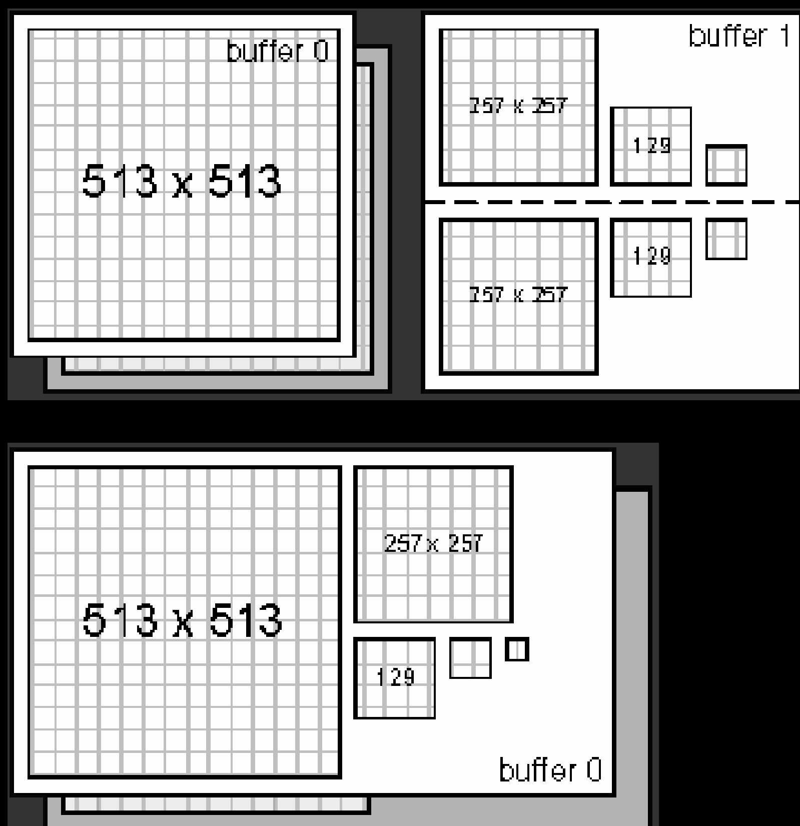
(Ignoring prehistory: Ikonas, Pixel Machine, Pixel-Planes…)

Compute pretending to be graphics

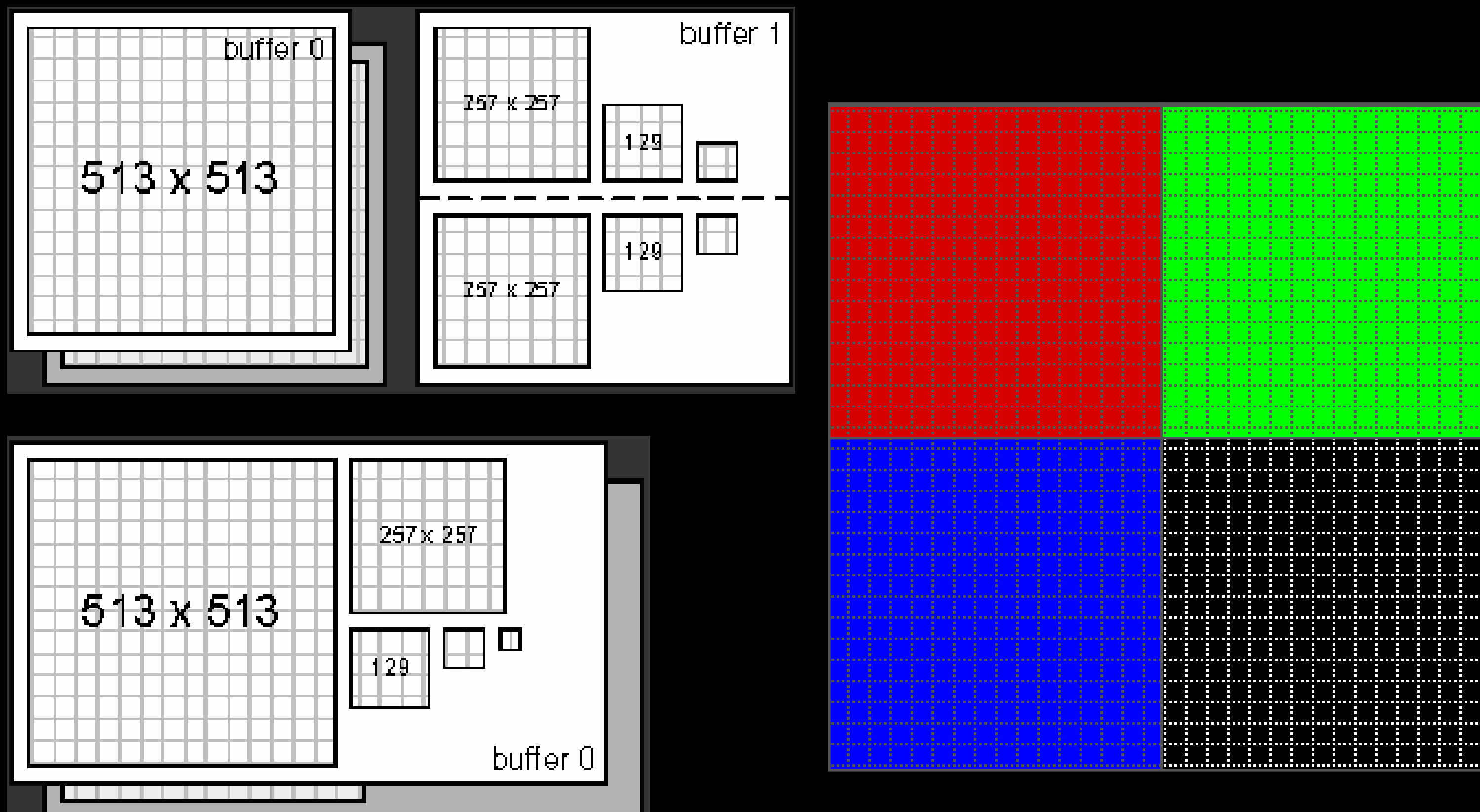
- Disguise data as triangles or textures
- Disguise algorithm as render passes & shaders

→Trick graphics pipeline into doing your computation!

Typical GPGPU Constructs

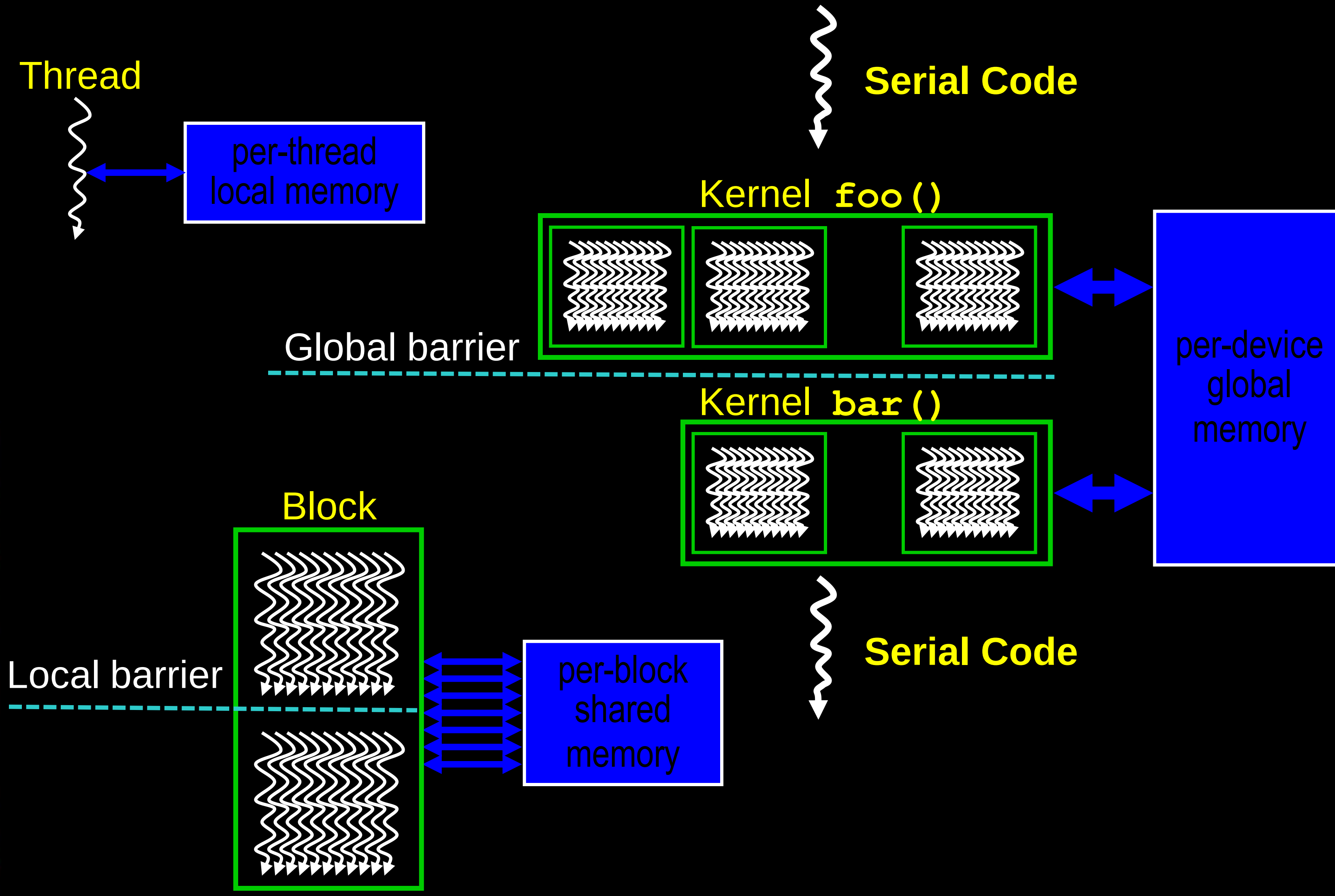


Typical GPGPU Constructs



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

GPU Computing 2.0: *CUDA*



GPU TECHNOLOGY CONFERENCE

OCTOBER 11-14, 2011 | SAN JOSE, CALIFORNIA

GPU Computing: State of the Union

GPU TECHNOLOGY CONFERENCE

GPU Computing 3.0: *An Ecosystem*

Languages & API's

CUDA C/C++

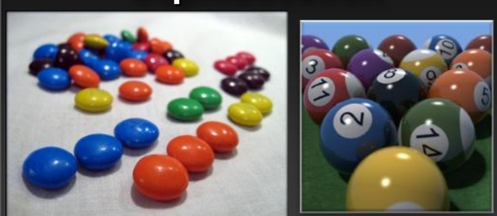
Microsoft DirectX 11

OpenCL

Hardware & Product Lines

Research & Education

Algorithmic Sophistication



Cloud Services

amazon web services™

Libraries

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

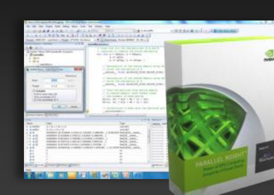
Tools & Partners

TOTALVIEW TECHNOLOGIES
A Rogue Wave Software Company

allinea ddt
the distributed debugging tool

Mathematical Packages

Integrated Development Environment



Microsoft Visual Studio PARTNER

GPU Computing by the numbers

300,000,000	CUDA Capable GPUs
500,000	CUDA Toolkit Downloads
100,000	Active CUDA Developers
400	Universities Teaching CUDA
12	CUDA Centers of Excellence

Computational Graphics

Workloads

- Each GPU is designed to target a mix of known and speculative workloads

- The art of GPU design is choosing these workloads

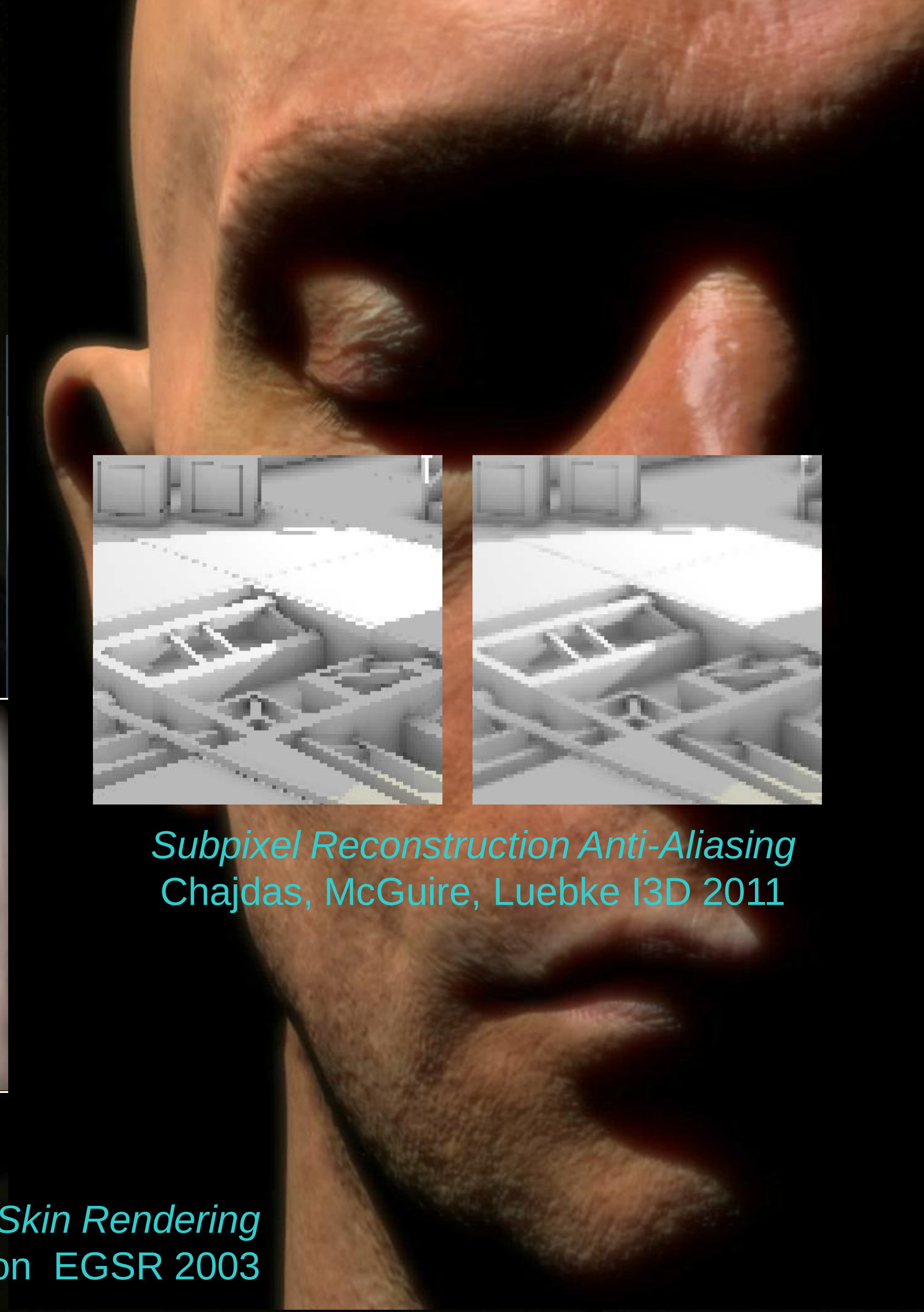
  (and shipping on schedule!)

What workloads will drive future GPUs?

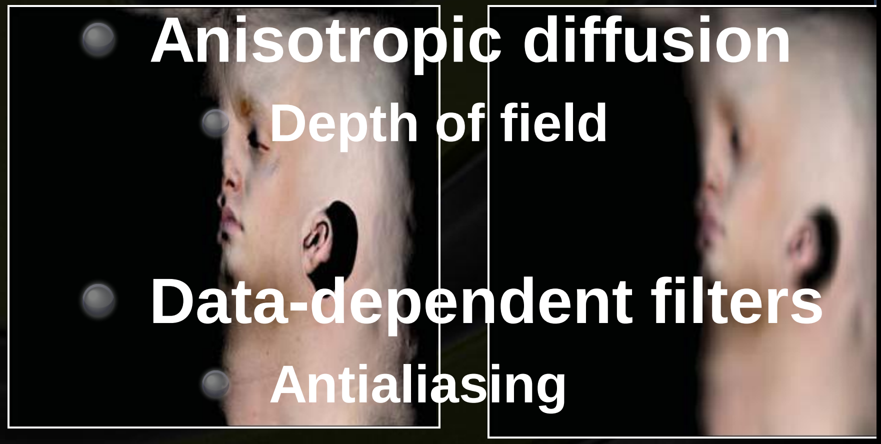
- High performance computing
- Graphics
- *Computational graphics*

Filtering

- Separable filters
  - Depth of field
  - Film bloom
  - Subsurface scattering
- Anisotropic diffusion
  - Depth of field
- Data-dependent filters
  - Antialiasing



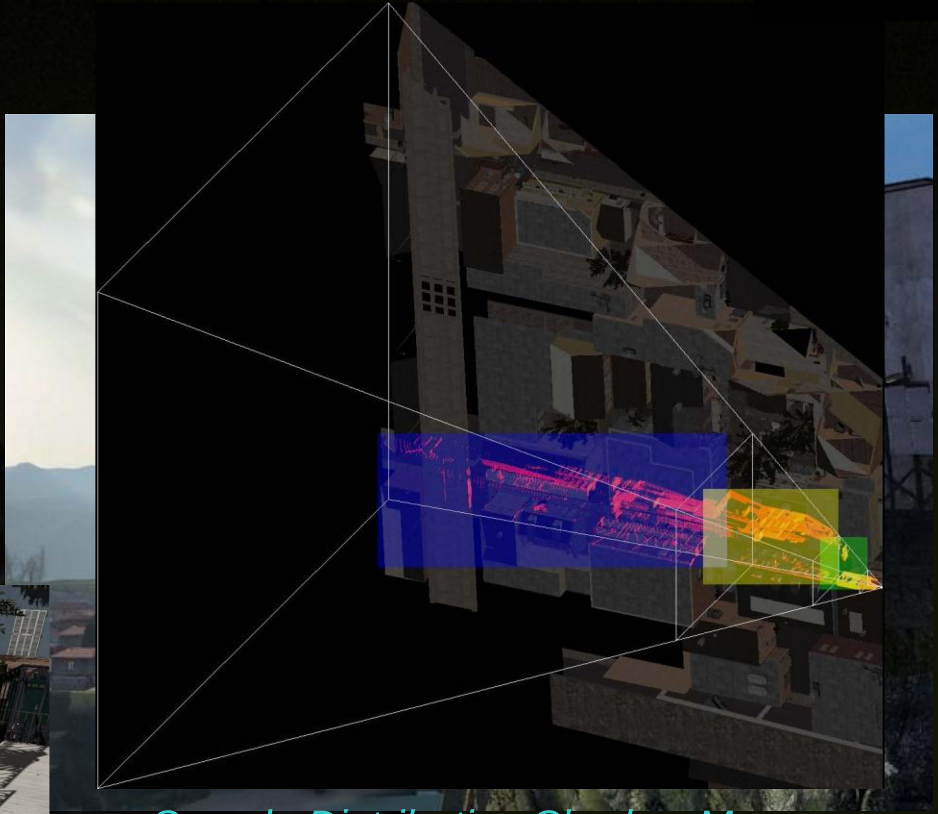
Subpixel Reconstruction Anti-Aliasing
Chajdas, McGuire, Luebke I3D 2011



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

Histogram

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



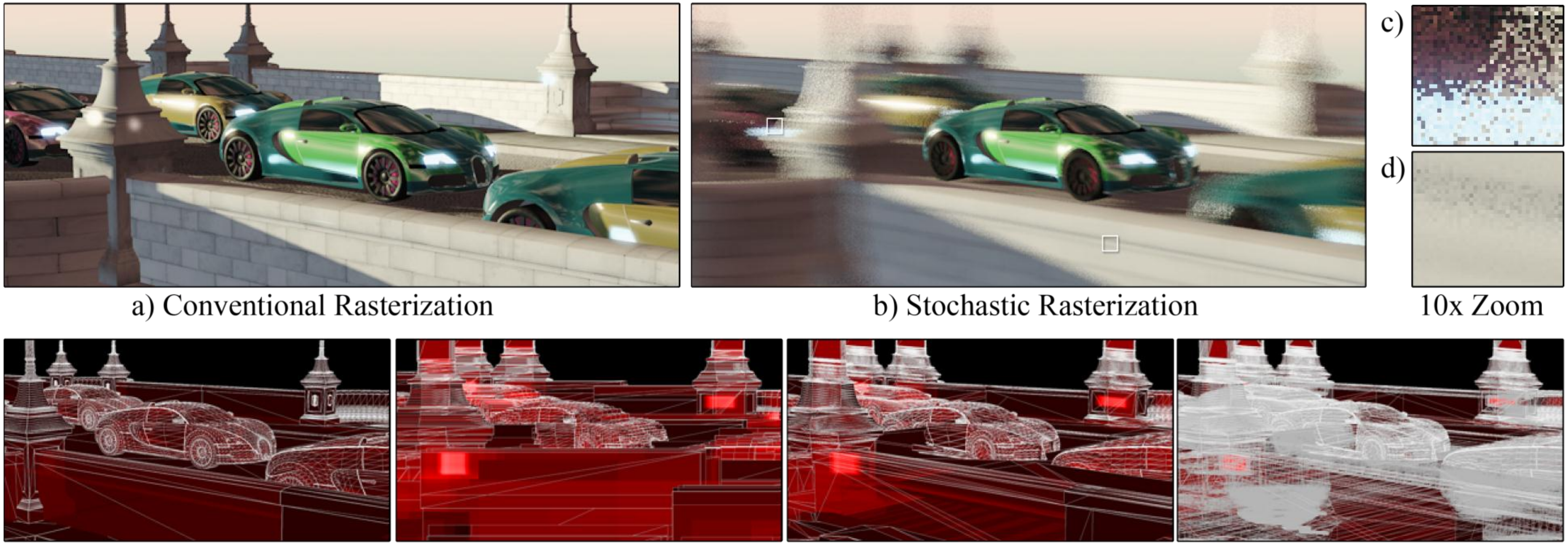
Sample Distribution Shadow Maps

Lauritzen, Salvi, Lefohn, I3D 2011

HDR in Valve's source engine

Rasterization as Iteration

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



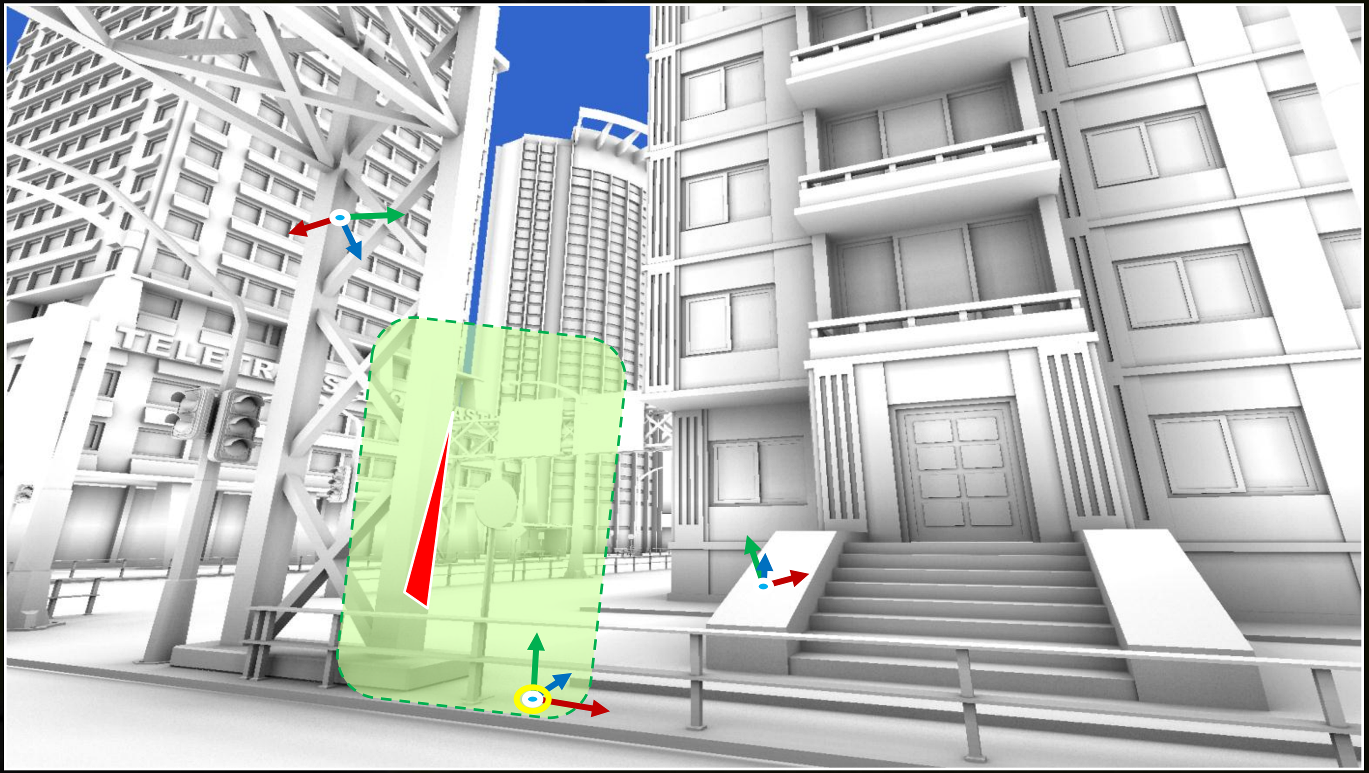
a) Conventional Rasterization b) Stochastic Rasterization 10x Zoom

e) t=1 Geometry f) 2D AABB Geometry g) 2D Convex Hull Geometry h) 2D C.H. Full Wireframe

Real-Time Stochastic Rasterization on Conventional GPU Architectures
McGuire, Enderton, Shirley, Luebke, HPG 2010

Rasterization as Iteration

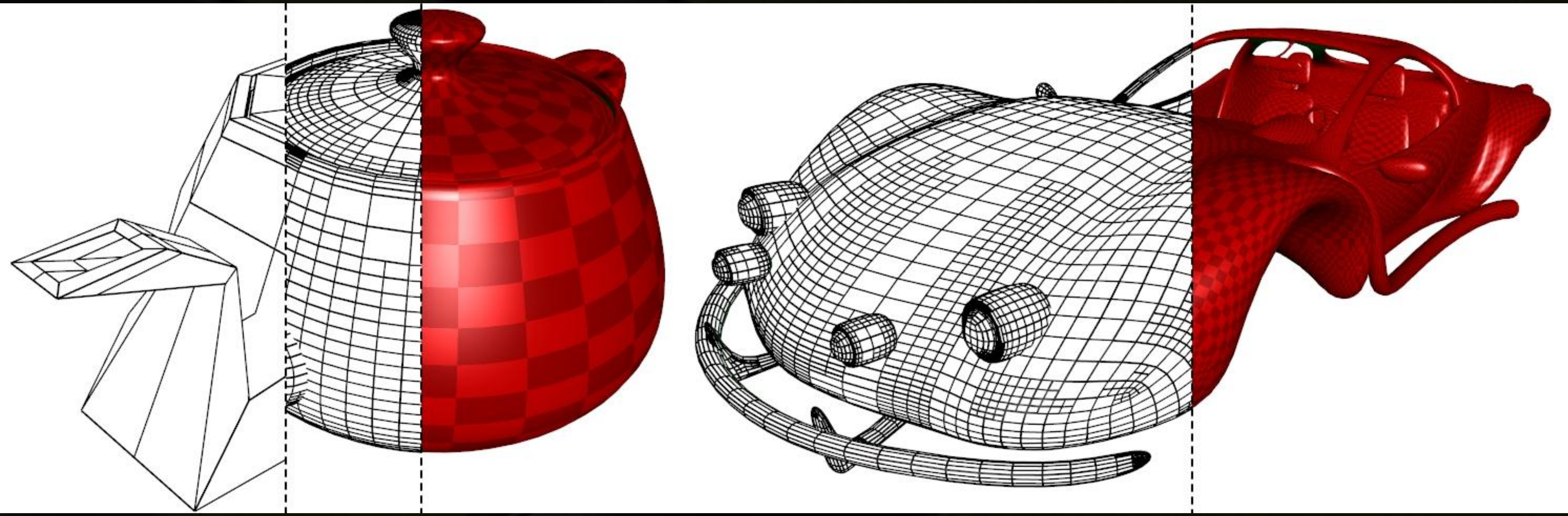
- **Darken pixels by % of hemisphere blocked by nearby triangles**
- **Compute triangle *regions of influence* to find affected pixels**



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

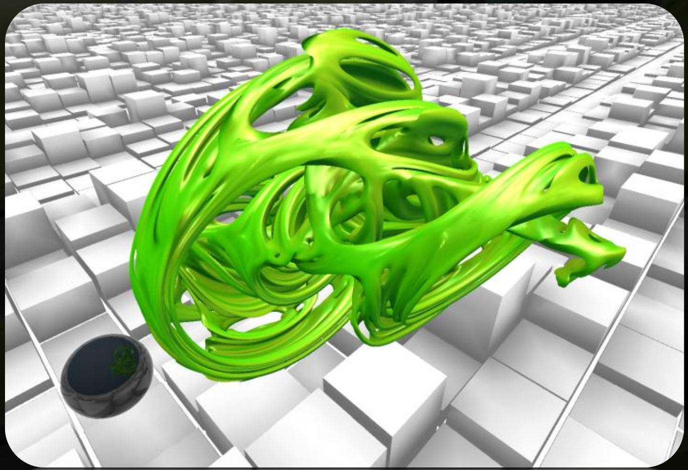
CUDA Tessellation

- **Flexible adaptive geometry generation**
- **Recursive subdivision**



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

Ray Tracing



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

GPU Computing 4.0?

Key GPU Workloads

- Computational graphics
- Scientific and numeric computing
- Image processing – video & images
- Computer vision
- Speech & natural language
- Data mining & machine learning

GPU TECHNOLOGY CONFERENCE

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!**

GPU TECHNOLOGY CONFERENCE
OCTOBER 11-14, 2011 | SAN JOSE, CALIFORNIA

Questions?

dluebke@nvidia.com

Computational Challenge

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

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

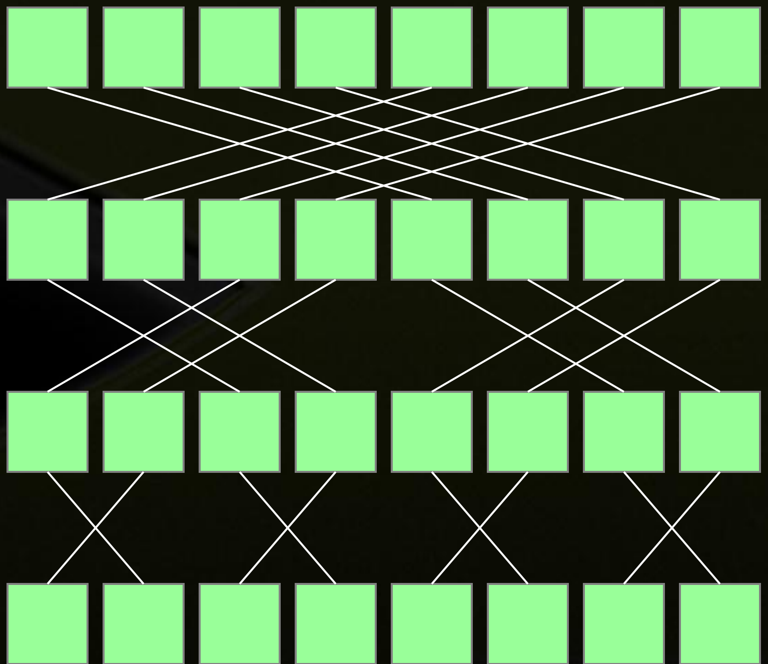
Example: Parallel Reduction

- **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**



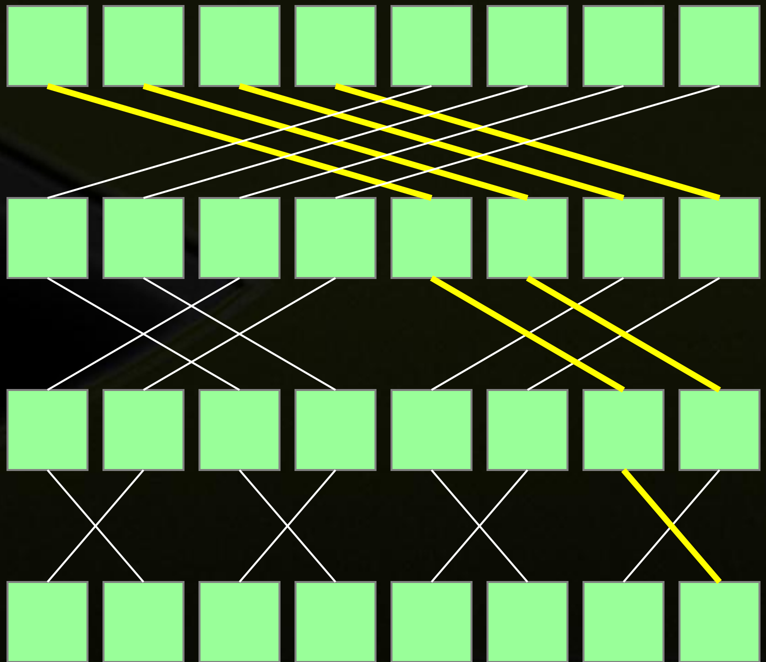
Example: Parallel Reduction

- **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**

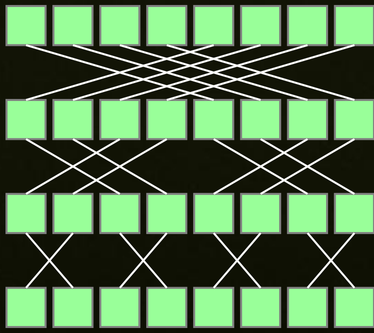


Parallel Reduction for 1 Block

```
// 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]
```



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