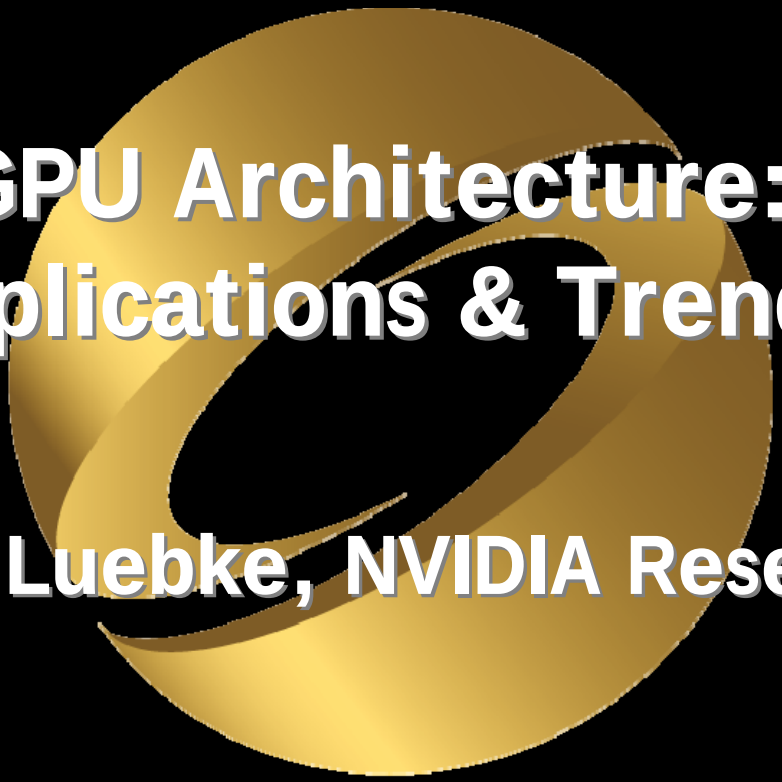
GPU Architecture: Implications & Trends

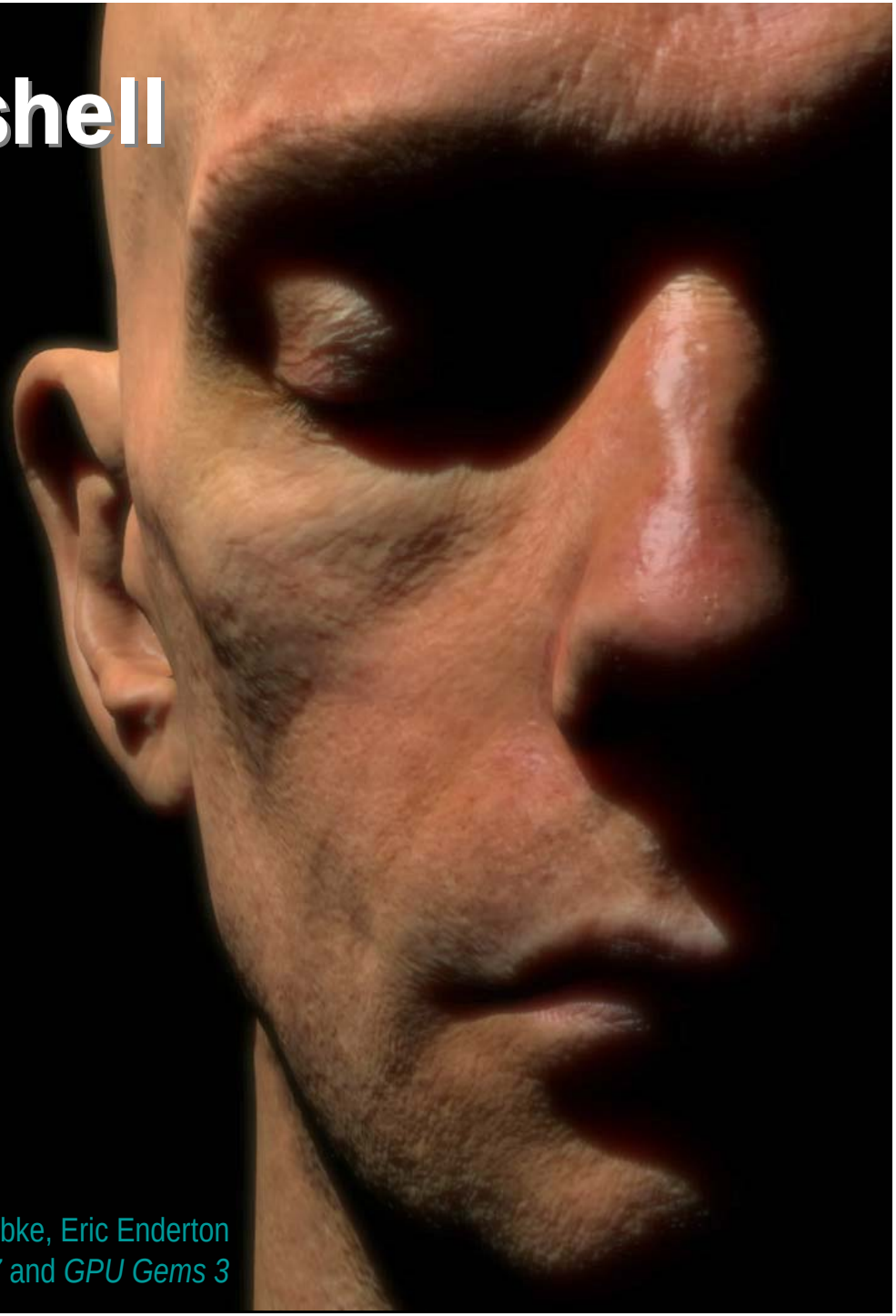
David Luebke, NVIDIA Research



SIGGRAPH2008

Beyond Programmable Shading: In Action

Graphics in a Nutshell

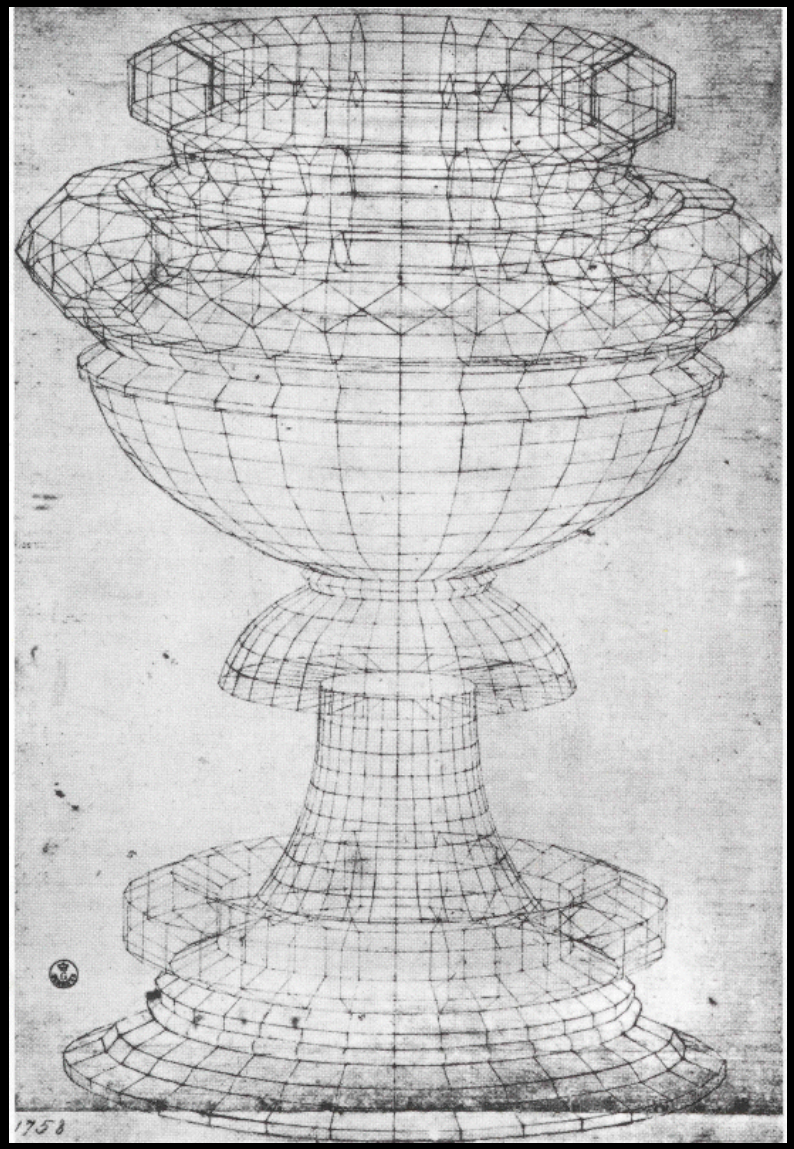


- **Make great images**
  - intricate shapes
  - complex optical effects
  - seamless motion

- **Make them fast**
  - invent clever techniques
  - use every trick imaginable
  - build monster hardware

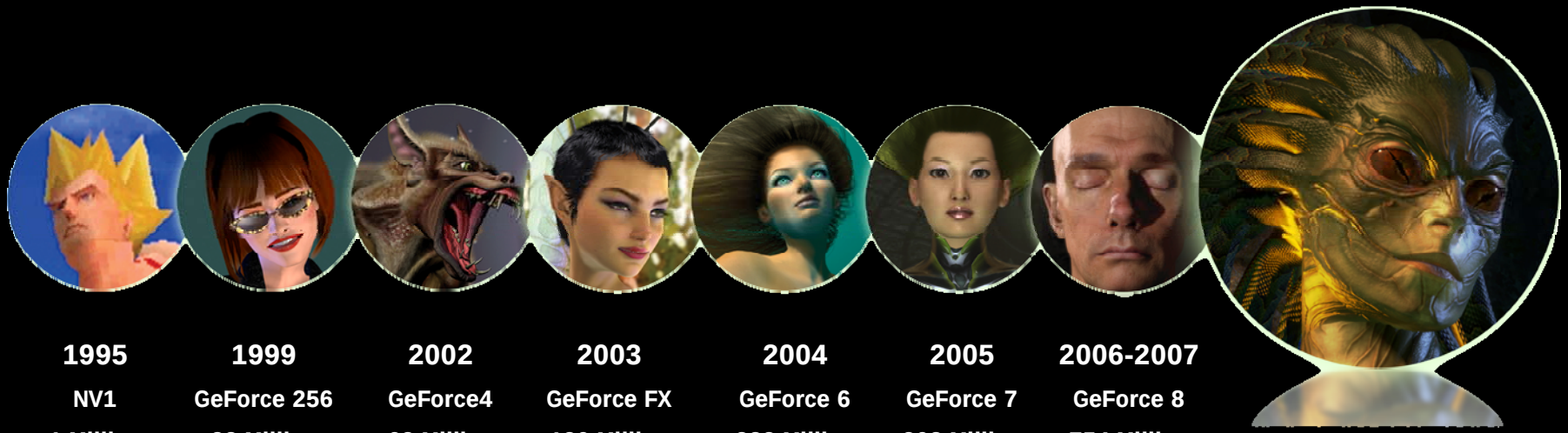
Eugene d'Eon, David Luebke, Eric Enderton
In *Proc. EGSR 2007* and *GPU Gems 3*

... or we could just do it by hand



Perspective study of a chalice
Paolo Uccello, circa 1450

GPU Evolution - Hardware



1995
NV1
1 Million
Transistors

1999
GeForce 256
22 Million
Transistors

2002
GeForce4
63 Million
Transistors

2003
GeForce FX
130 Million
Transistors

2004
GeForce 6
222 Million
Transistors

2005
GeForce 7
302 Million
Transistors

2006-2007
GeForce 8
754 Million
Transistors

2008
GeForce GTX 200
1.4 Billion
Transistors

GPU Evolution - Programmability

Future: CUDA,
DX11 Compute, OpenCL

CUDA (PhysX, RT, AFSM…)
2008 - Backbreaker

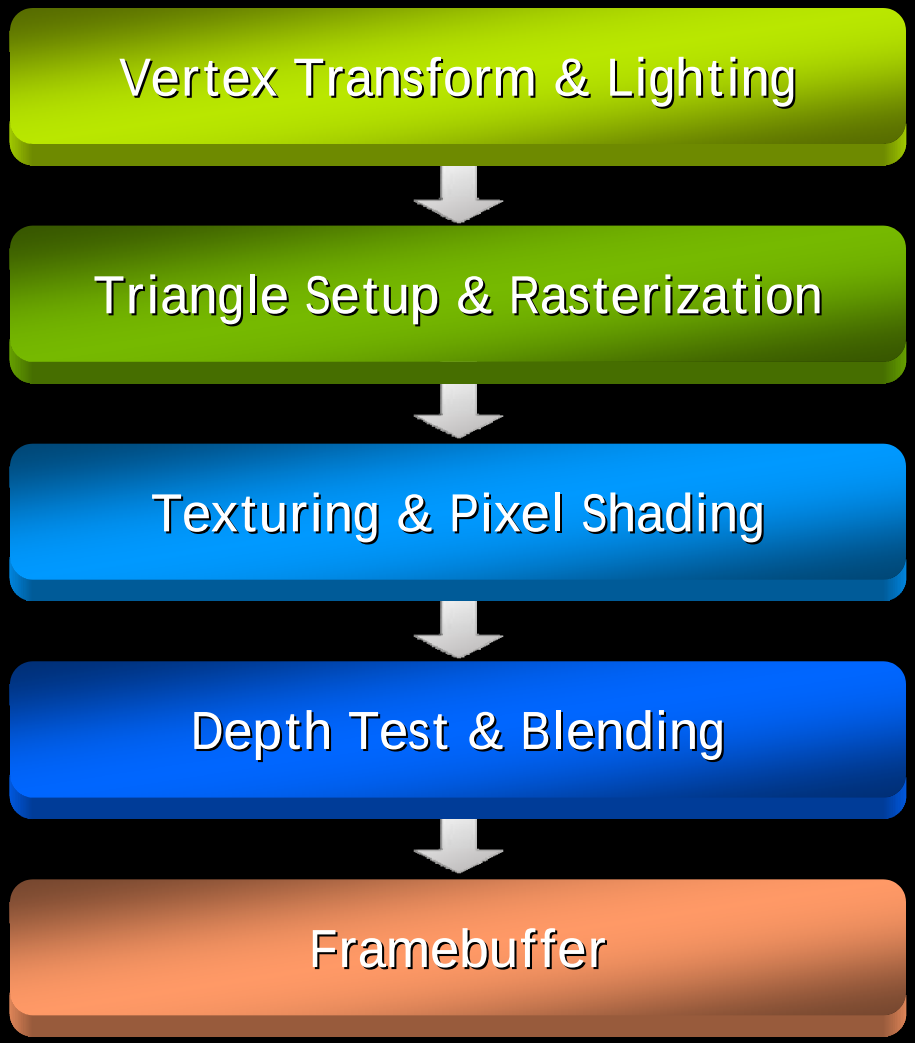
DX10 Geo Shaders
2007 - Crysis

DX9 Prog Shaders
2004 – Far Cry

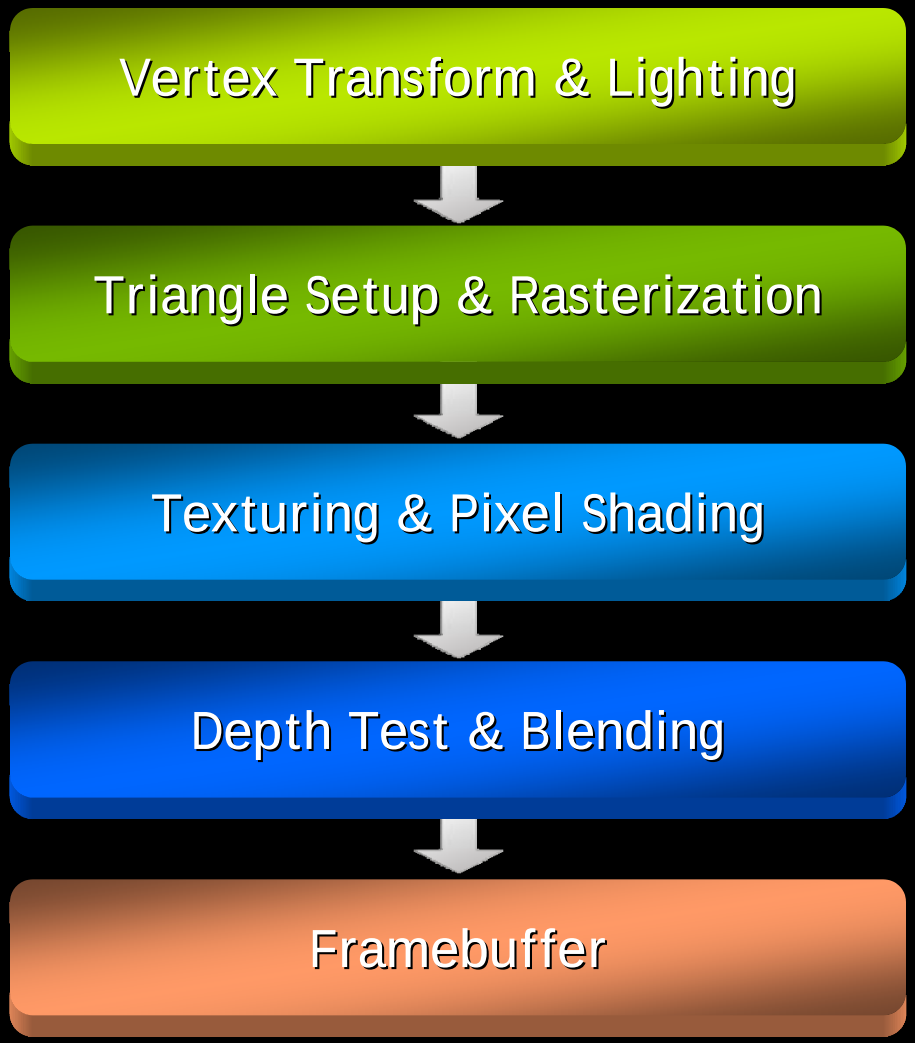
DX8 Pixel Shaders
2001 – Ballistics

DX7 HW T&L
1999 – Test Drive 6

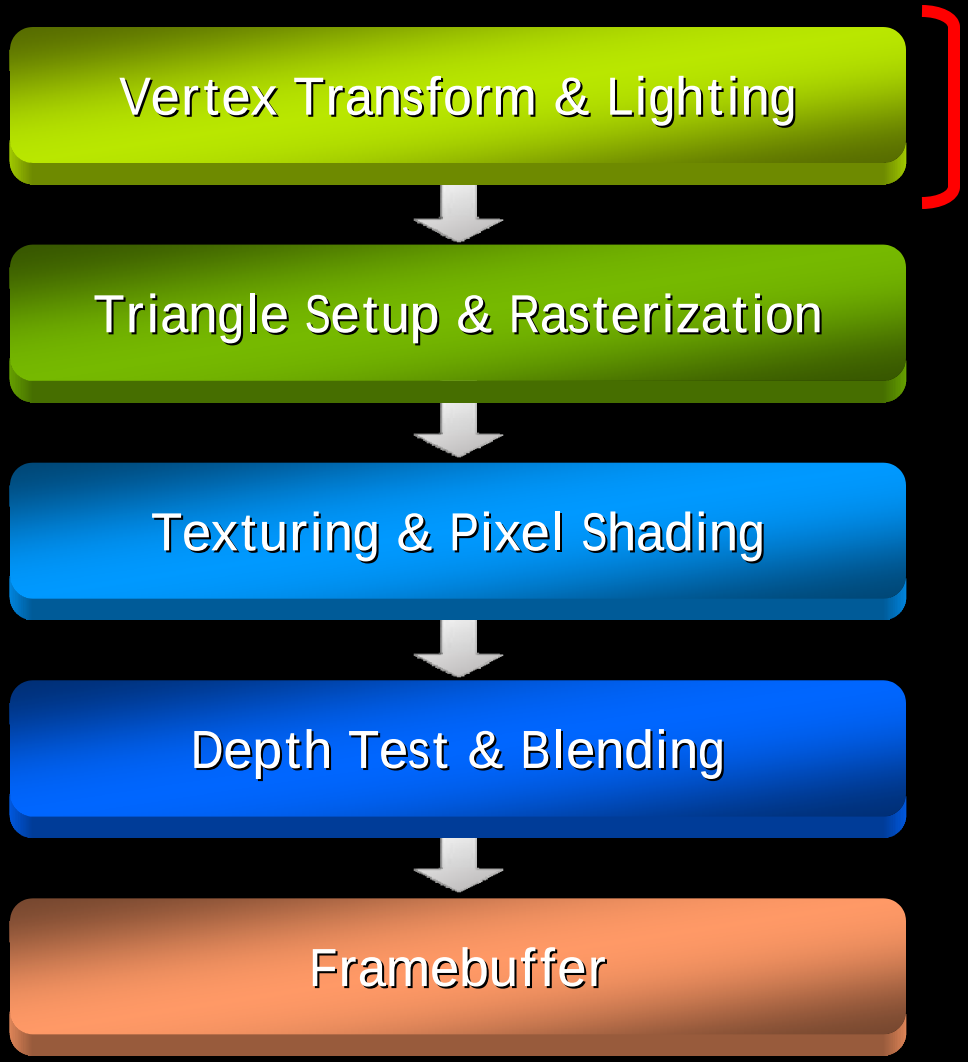
The Graphics Pipeline

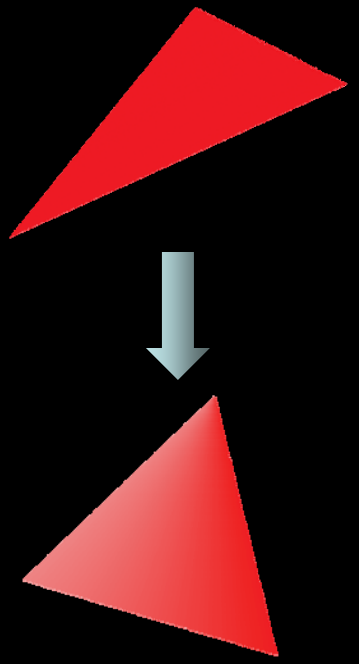


The Graphics Pipeline

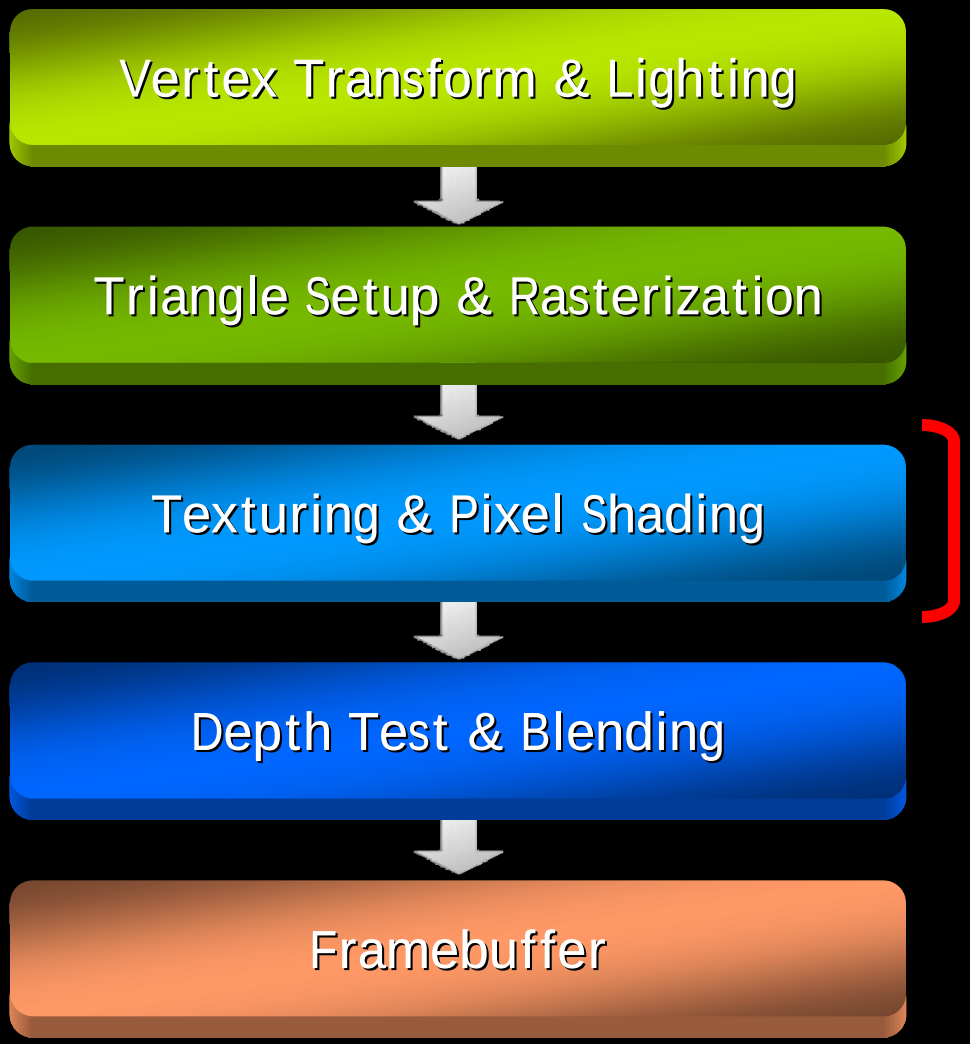


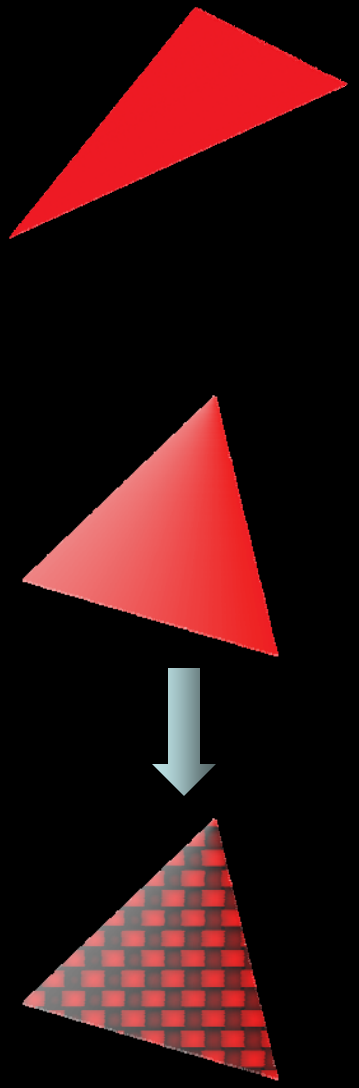
The Graphics Pipeline



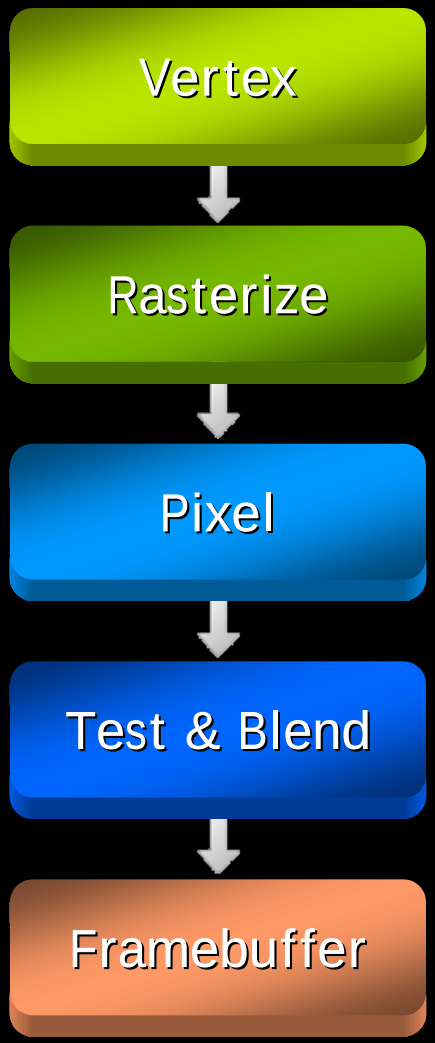


The Graphics Pipeline



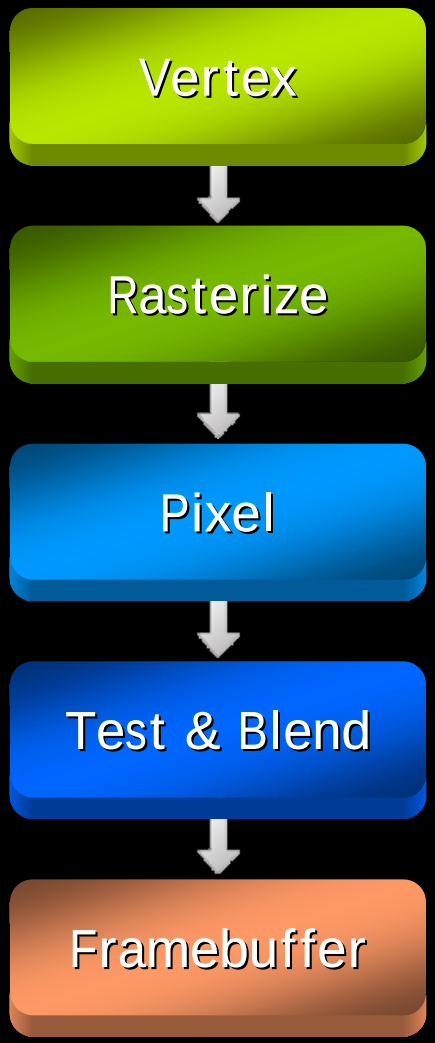


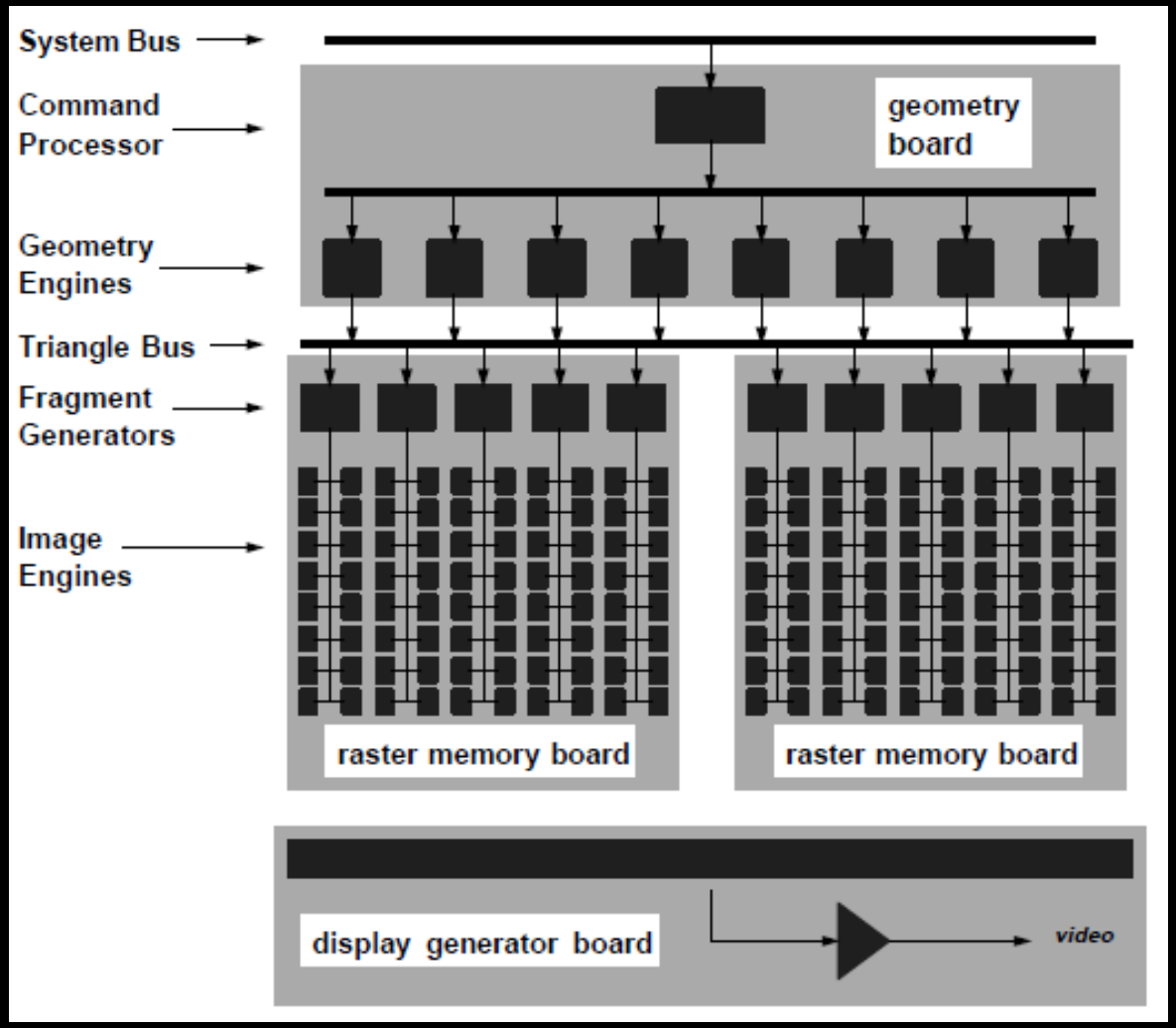
The Graphics Pipeline



- Key abstraction of real-time graphics
- Hardware used to look like this
- Distinct chips/boards per stage
- Fixed data flow through pipeline

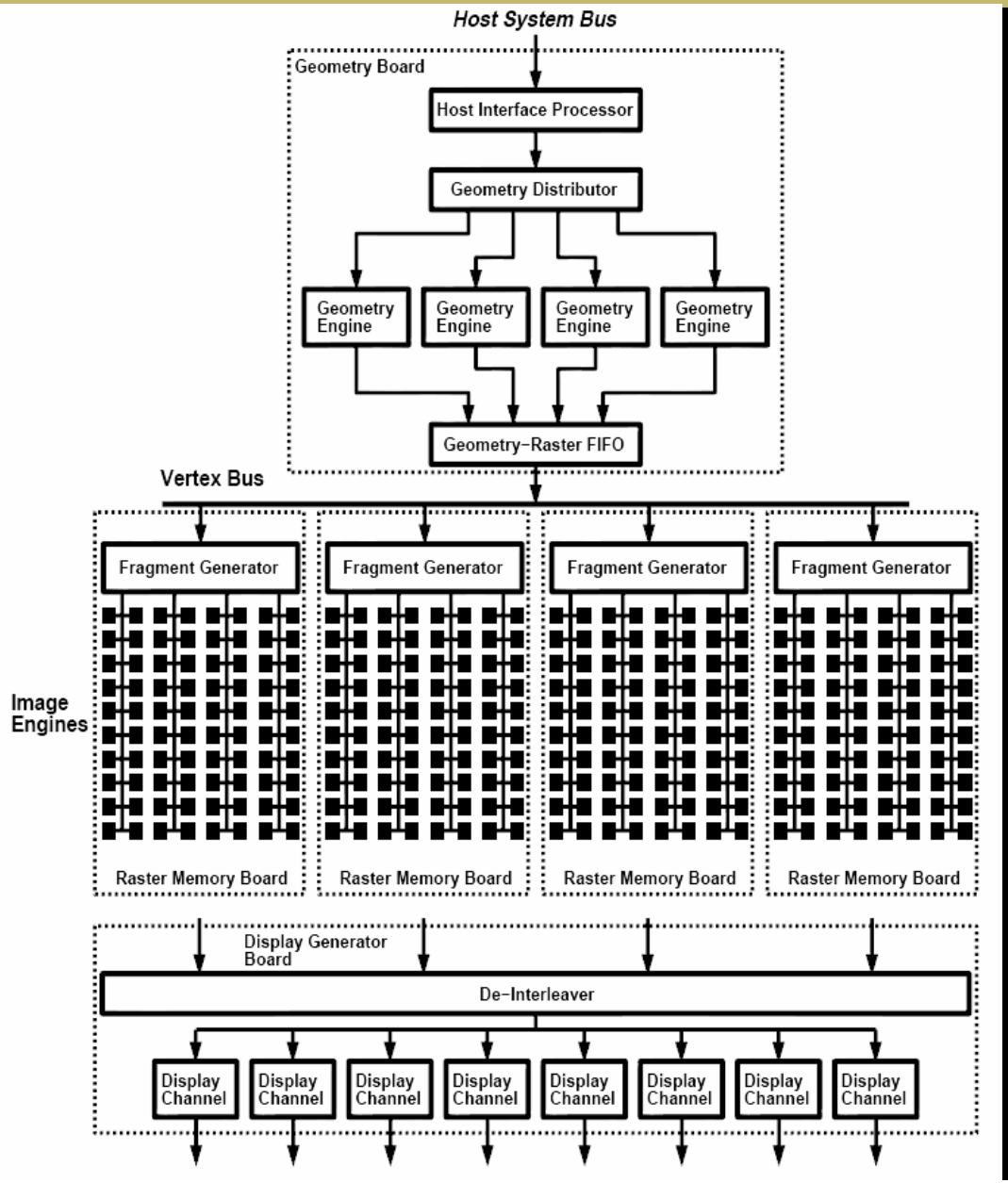
SGI RealityEngine (1993)





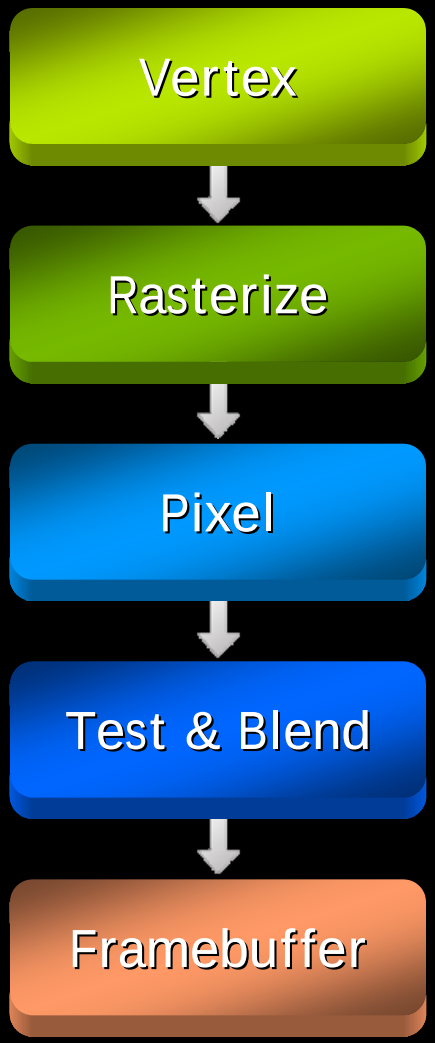
Kurt Akeley. RealityEngine Graphics. In *Proc. SIGGRAPH '93*. ACM Press, 1993.

SGI InfiniteReality (1997)



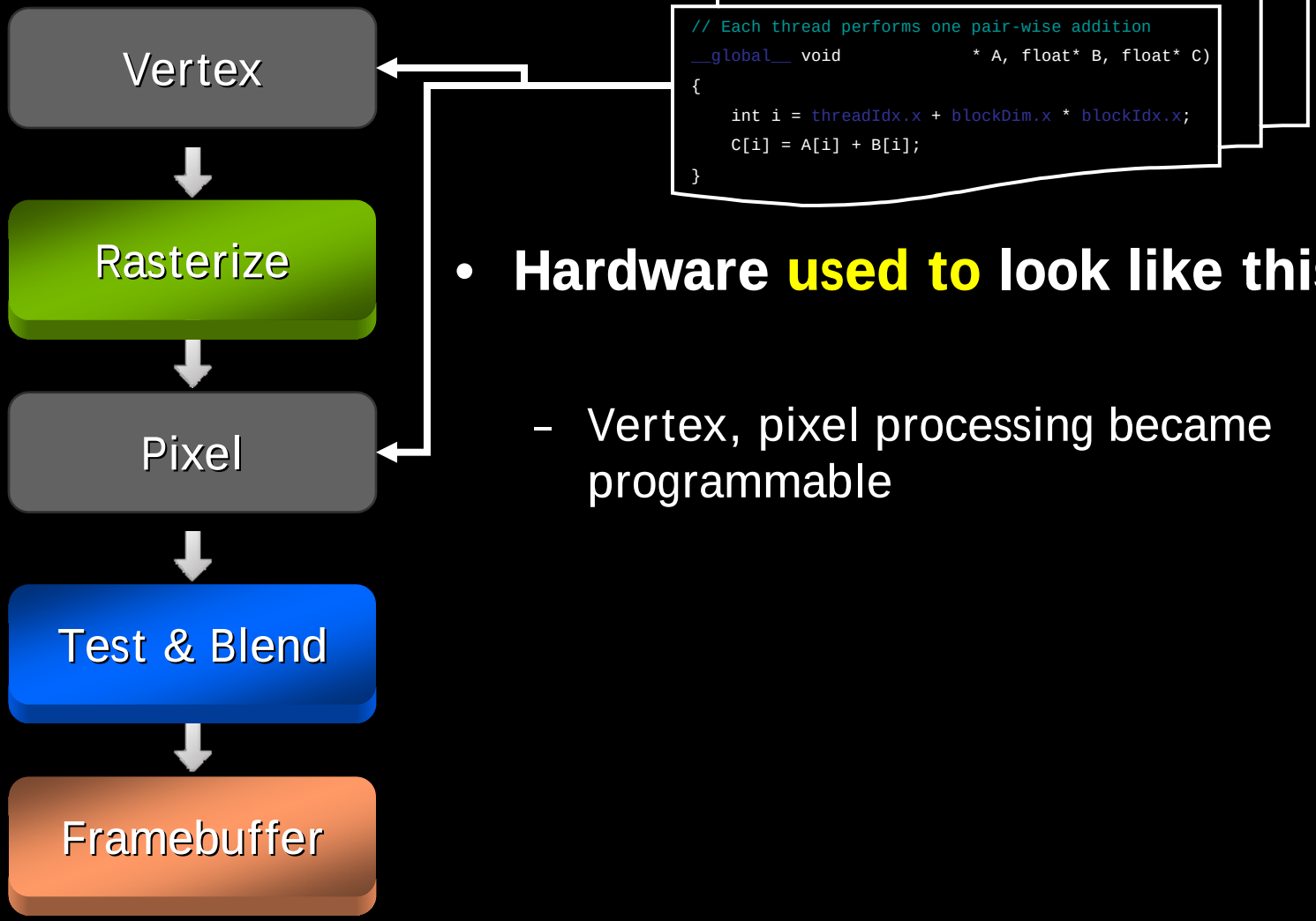
Montrym, Baum, Dignam, & Migdal. InfiniteReality: A real-time graphics system. In *Proc. SIGGRAPH '97*. ACM Press, 1997.

The Graphics Pipeline



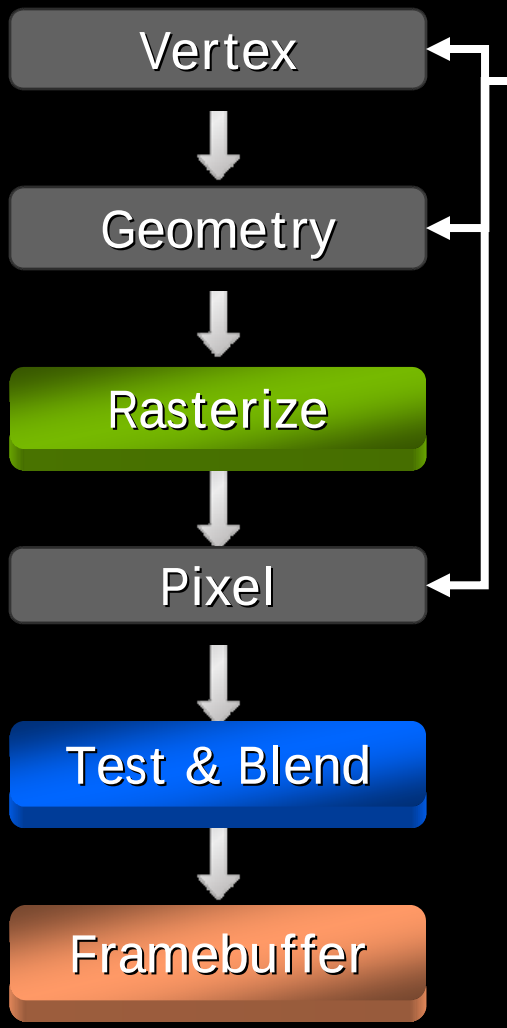
- Remains a useful abstraction
- Hardware used to look like this

The Graphics Pipeline



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

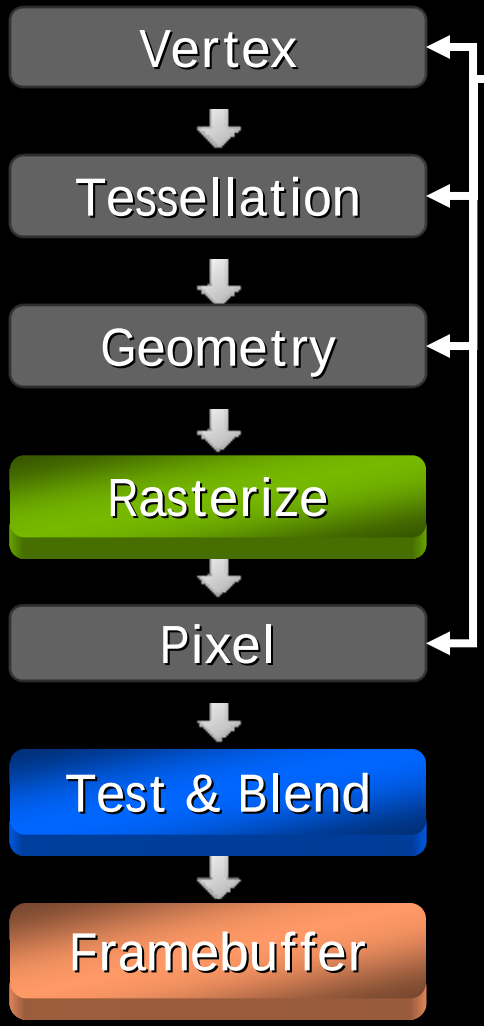
The Graphics Pipeline




```
// Each thread performs one pair-wise addition
__global__ void              * A, float* B, float* C)
{
    int i = threadIdx.x + blockDim.x * blockIdx.x;
    C[i] = A[i] + B[i];
}
```

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

The Graphics Pipeline

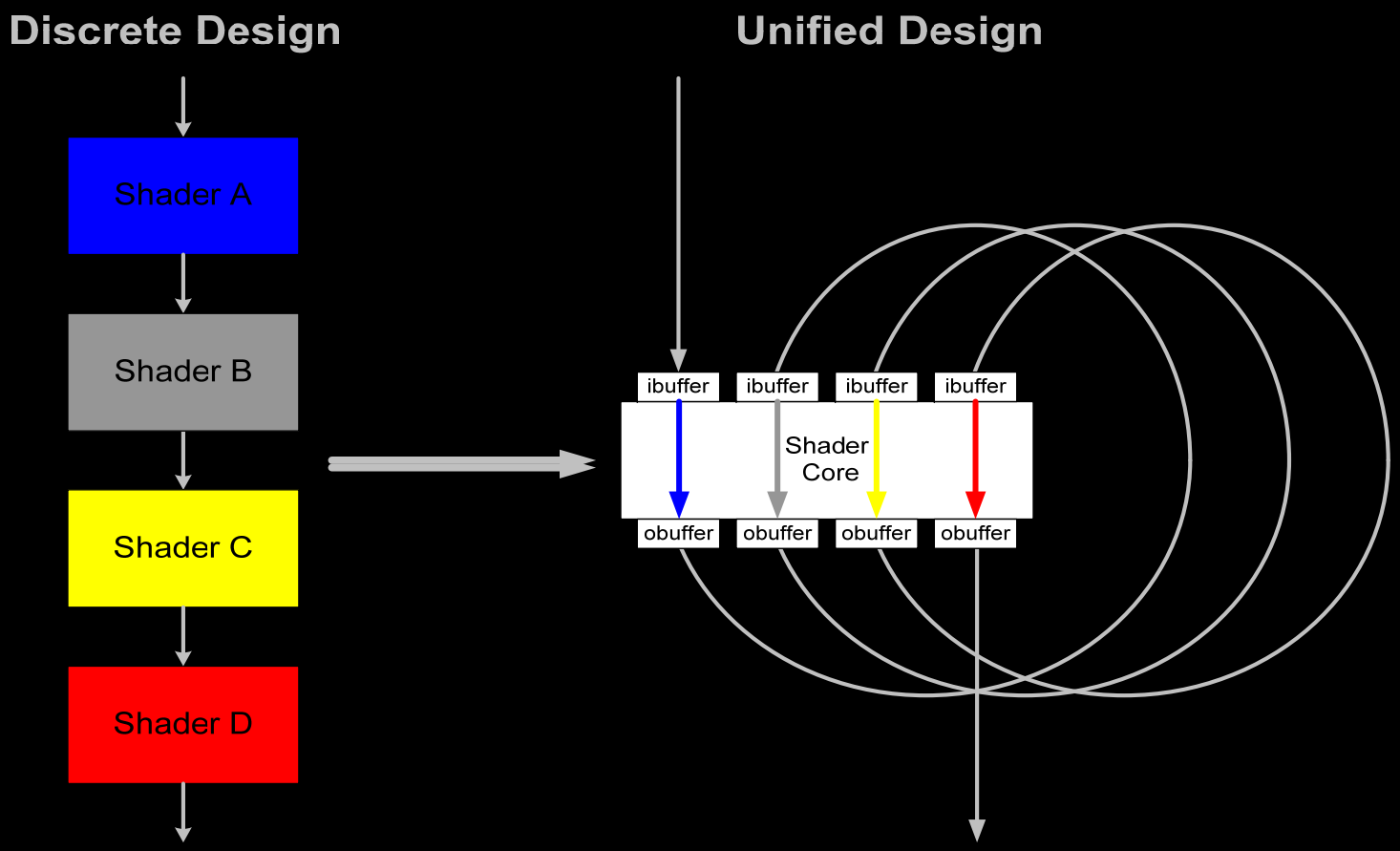



```
// Each thread performs one pair-wise addition
__global__ void             * A, float* B, float* C)
{
    int i = threadIdx.x + blockDim.x * blockIdx.x;
    C[i] = A[i] + B[i];
}
```

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

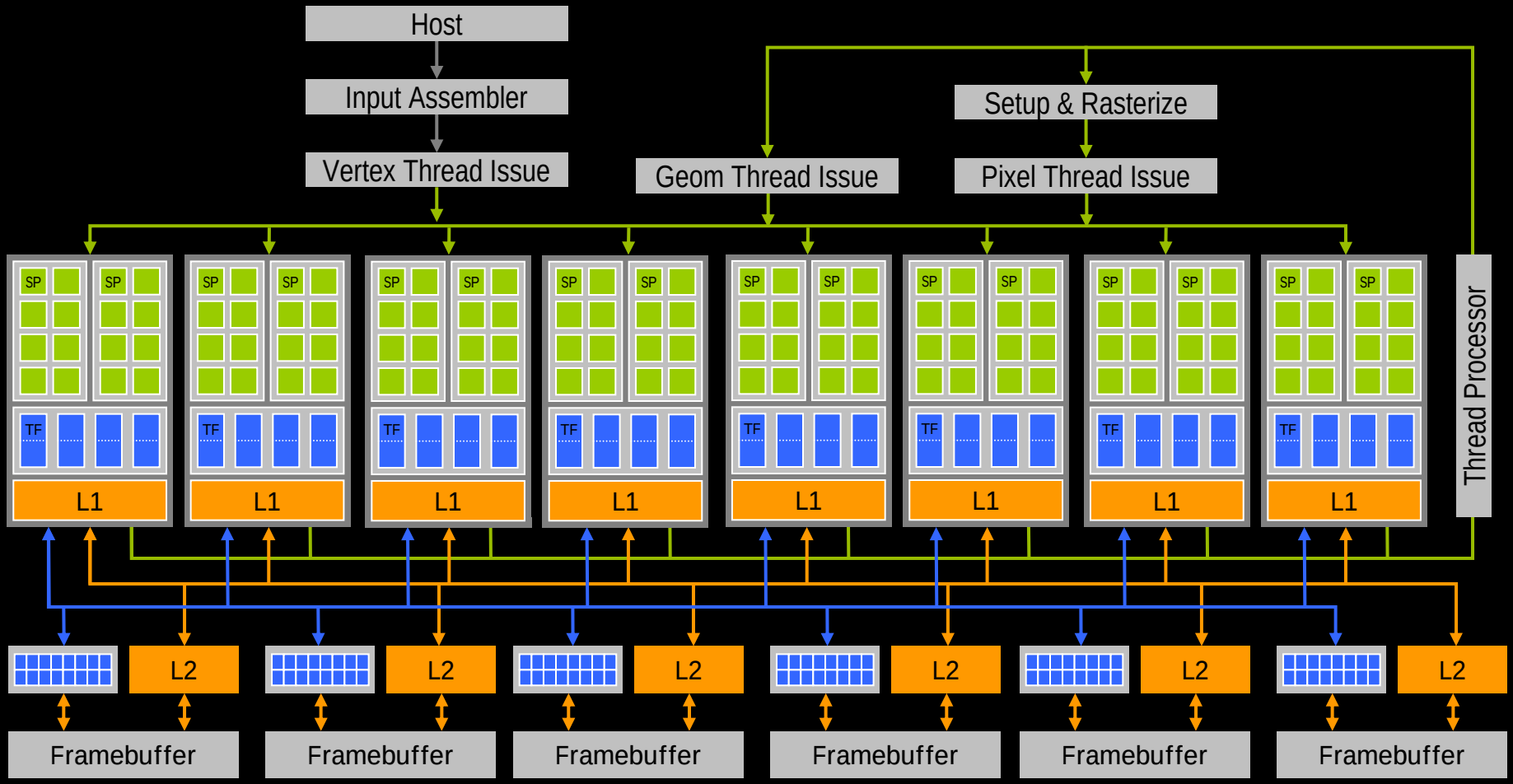
GPU architecture increasingly centers around shader execution

Modern GPUs: Unified Design

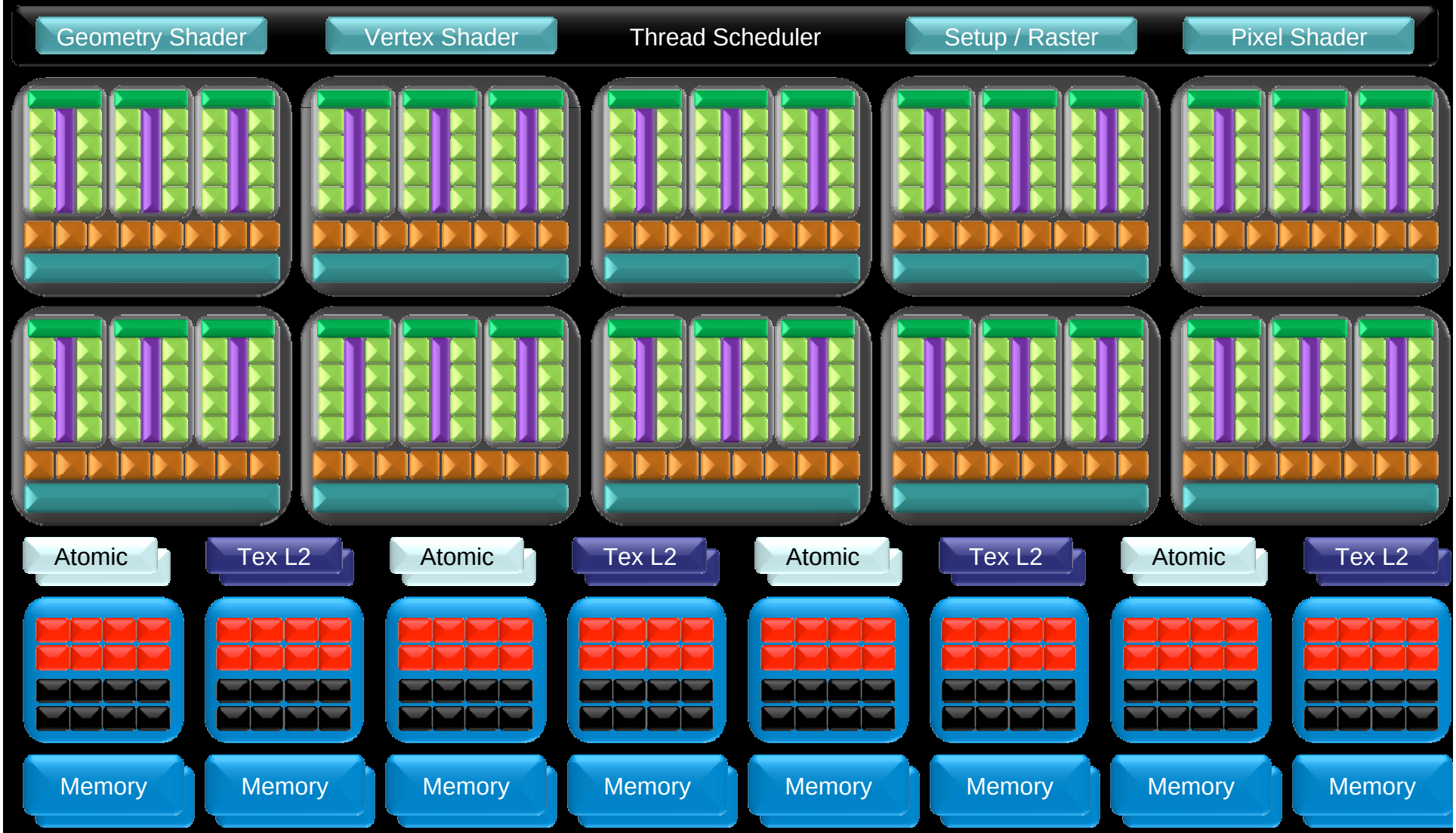


Vertex shaders, pixel shaders, etc. become *threads*
running different programs on a flexible core

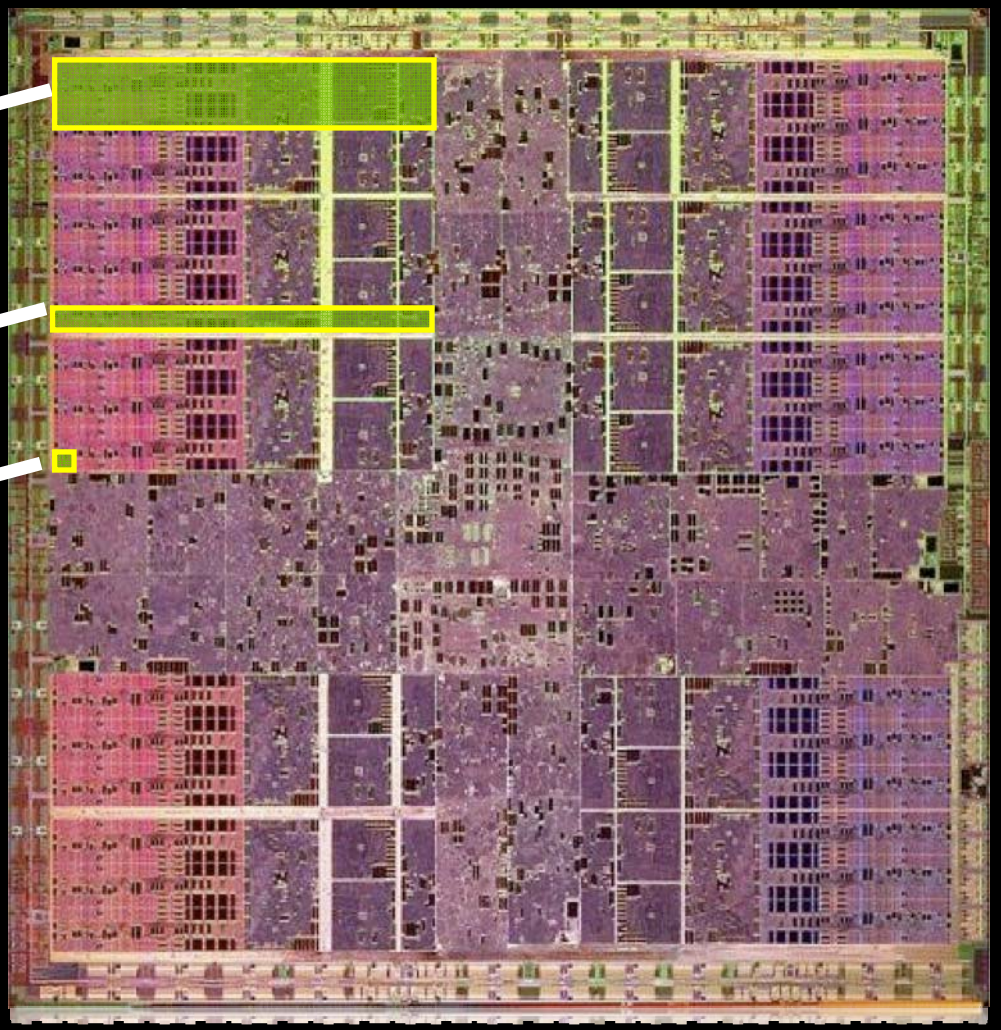
GeForce 8: Modern GPU Architecture



GeForce GTX 200 Architecture



Tesla GPU Architecture



Goal: Performance per millimeter

- **For GPUs, perfomance == throughput**
- **Strategy: hide latency with computation not cache**
  ➔ Heavy multithreading!
- **Implication: need many threads to hide latency**
  - Occupancy – typically prefer 128 or more threads/TPA
  - Multiple thread blocks/TPA help minimize effect of barriers
- **Strategy: Single Instruction Multiple Thread (*SIMT*)**
  - Support SPMD programming model
  - Balance performance with ease of programming

SIMT Thread Execution

- **High-level description of SIMT:**
  - Launch zillions of threads
  - When they do the same thing, hardware makes them go fast
  - When they do different things, hardware handles it gracefully

SIMT Thread Execution

- **Groups of 32 threads formed into warps**

  Warp: a set of parallel threads that execute a single instruction

SIMT Thread Execution

- **Groups of 32 threads formed into warps**
  - always executing same instruction
  - some become inactive when code path diverges
  - hardware automatically handles divergence
- **Warps are the primitive unit of scheduling**
  - pick 1 of 32 warps for each instruction slot
  - Note warps may be running different programs/shaders!
- **SIMT execution is an implementation choice**
  - sharing control logic leaves more space for ALUs
  - largely invisible to programmer
  - must understand for performance, not correctness

GPU Architecture: Summary

- From fixed function to configurable to programmable
  → architecture now centers on flexible processor core

- Goal: performance / mm^2 (perf == throughput)
  → architecture uses heavy multithreading

- Goal: balance performance with ease of use
  → SIMT: hardware-managed parallel thread execution

GPU Architecture: Trends

- **Long history of ever-increasing programmability**
  - Culminating today in CUDA: program GPU directly in C
- **Graphics pipeline, APIs are abstractions**
  - CUDA + graphics enable "replumbing" the pipeline
- **Future: continue adding expressiveness, flexibility**
  - CUDA, OpenCL, DX11 Compute Shader, …
  - Lower barrier further between compute and graphics

Questions?

David Luebke
dluebke@nvidia.com

SIGGRAPH2008

Beyond Programmable Shading: In Action

GPU Design

CPU/GPU Parallelism

- **Moore's Law gives you more and more transistors**
  - **What do you want to do with them?**
  - **CPU strategy: make the workload (one compute thread) run as fast as possible**
    - **Tactics:**
      - Cache (area limiting)
      - Instruction/Data prefetch
      - Speculative execution

      →limited by "perimeter" – communication bandwidth

      …then add task parallelism…multi-core
  - **GPU strategy: make the workload (as many threads as possible) run as fast as possible**
    - **Tactics:**
      - Parallelism (1000s of threads)
      - Pipelining

      → limited by "area" – compute capability

© NVIDIA Corporation 2007

GPU Architecture

- Massively Parallel
  - 1000s of processors (today)
- Power Efficient
  - Fixed Function Hardware = area & power efficient
  - Lack of speculation. More processing, less leaky cache
- Latency Tolerant from Day 1
- Memory Bandwidth
  - Saturate 512 Bits of Exotic DRAMs All Day Long (140 GB/sec today)
  - No end in sight for Effective Memory Bandwidth
- Commercially Viable Parallelism
  - Largest installed base of Massively Parallel (N>4) Processors
    - Using CUDA!!! Not just as graphics
- Not dependent on large caches for performance
  - Computing power = Freq * Transistors
  - Moore's law ^2

© NVIDIA Corporation 2007

What is a game?

1. AI
   - Terascale GPU
2. Physics
   - Terascale GPU
3. Graphics
   - Terascale GPU
4. Perl Script
   - 5 sq mm^2 (1% of die) serial CPU
     - Important to have, along with
       - Video processing, dedicated display, DMA engines, etc.

© NVIDIA Corporation 2007

GPU Architectures: Past/Present/Future

- 1995: Z-Buffered Triangles
- Riva 128: 1998: Textured Tris
- NV10: 1999: Fixed Function X-Formed Shaded Triangles
- NV20: 2001: FFX Triangles with Combiners at Pixels
- NV30: 2002: Programmable Vertex and Pixel Shaders (!)
- NV50: 2006: Unified shaders, CUDA
  - Global Illumination, Physics, Ray tracing, AI
- future???: extrapolate trajectory
  - Trajectory == Extension + Unification

© NVIDIA Corporation 2007

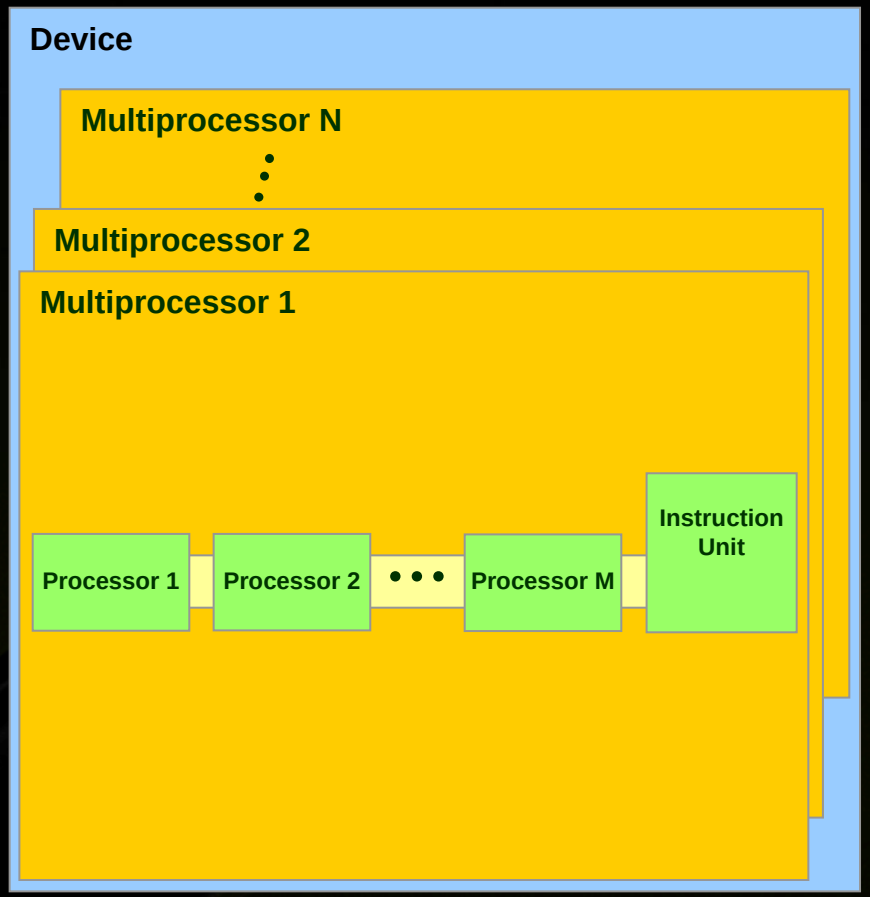
Dedicated Fixed Function Hardware area efficient, power efficient

- Rasterizer: 256pix/clock (teeny tiny silicon)
- Z compare & update with compression:
  - 256 samples / clock (in parallel with shaders)
  - Useful for primary rays, shadows, GI
  - Latency hidden
  - -tightly- coupled to frame-buffer
- Compressed color with blending
  - 32 Samples per clock (in parallel with raster & shaders & Z)
  - Latency hidden, etc.
- Primitive assembly: latency hidden serial datastructures
  - in parallel with shaders, & color & Z & raster
- HW Clipping, Setup, Planes, Microscheduling & Loadbalancing
  - In parallel with shaders & color & Z & raster & prim assembly, etc.

© NVIDIA Corporation 2007

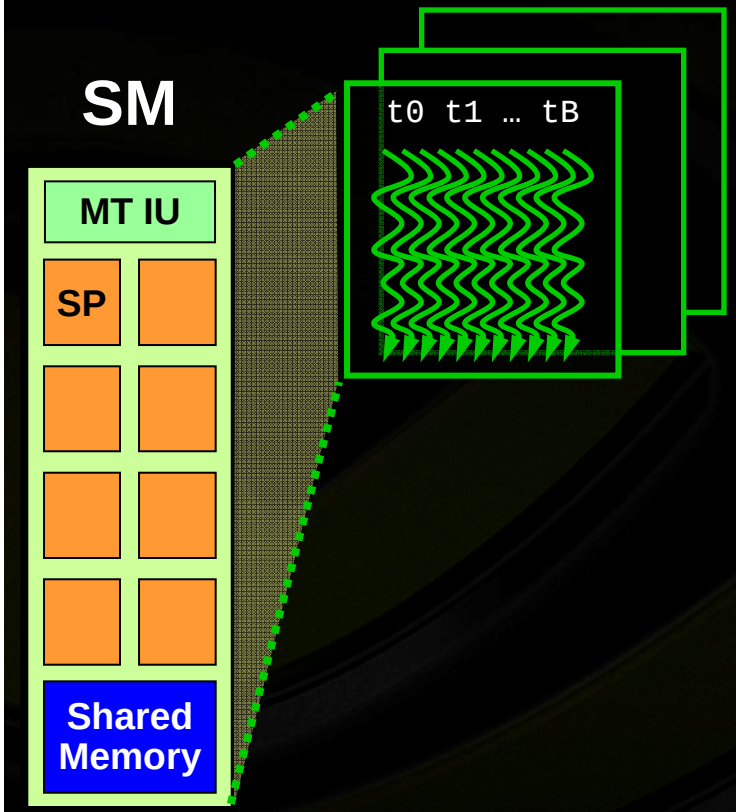
Hardware Implementation:
A Set of SIMD Multiprocessors

- The device is a set of multiprocessors
- Each multiprocessor is a set of 32-bit processors with a Single Instruction Multiple Data architecture
- At each clock cycle, a multiprocessor executes the same instruction on a group of threads called a warp
- The number of threads in a warp is the warp size



© NVIDIA Corporation 2007

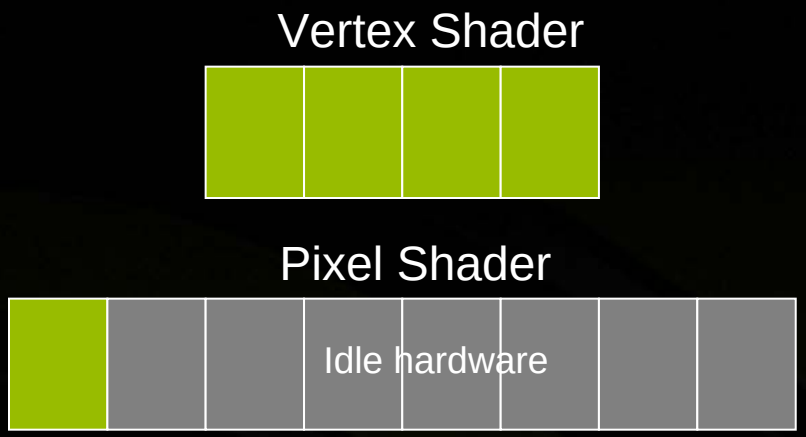
Streaming Multiprocessor (SM)



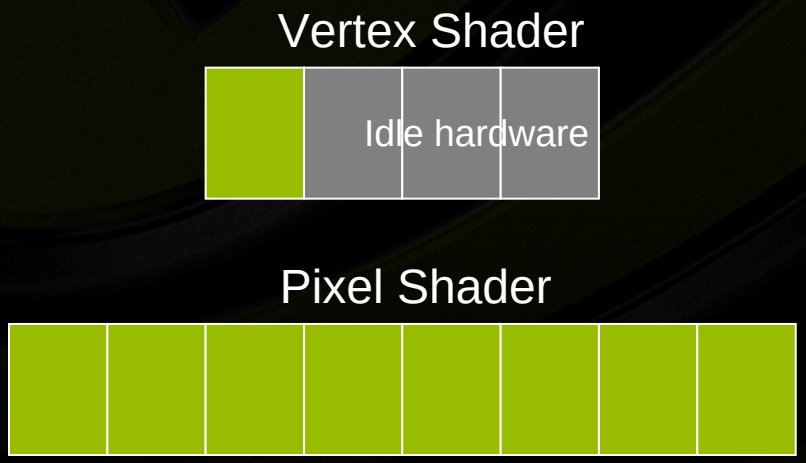
- Processing elements
  - 8 scalar thread processors (SP)
  - 32 GFLOPS peak at 1.35 GHz
  - 8192 32-bit registers (32KB)
    - ½ MB total register file space!
  - usual ops: float, int, branch, …
- Hardware multithreading
  - up to 8 blocks resident at once
  - up to 768 active threads in total
- 16KB on-chip memory
  - low latency storage
  - shared amongst threads of a block
  - supports thread communication

© NVIDIA Corporation 2007

Why unify?



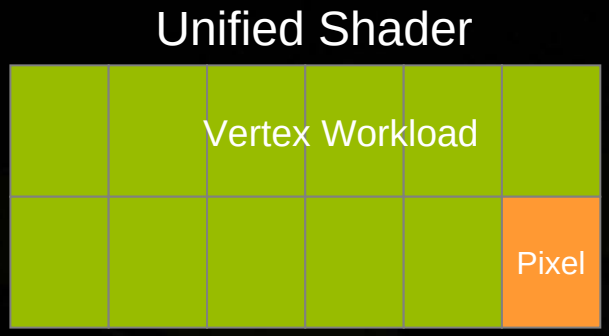
Heavy Geometry
Workload Perf = 4



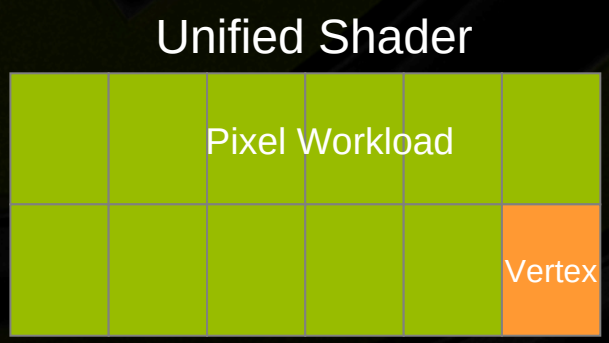
Heavy Pixel
Workload Perf = 8

© NVIDIA Corporation 2007

Why unify?



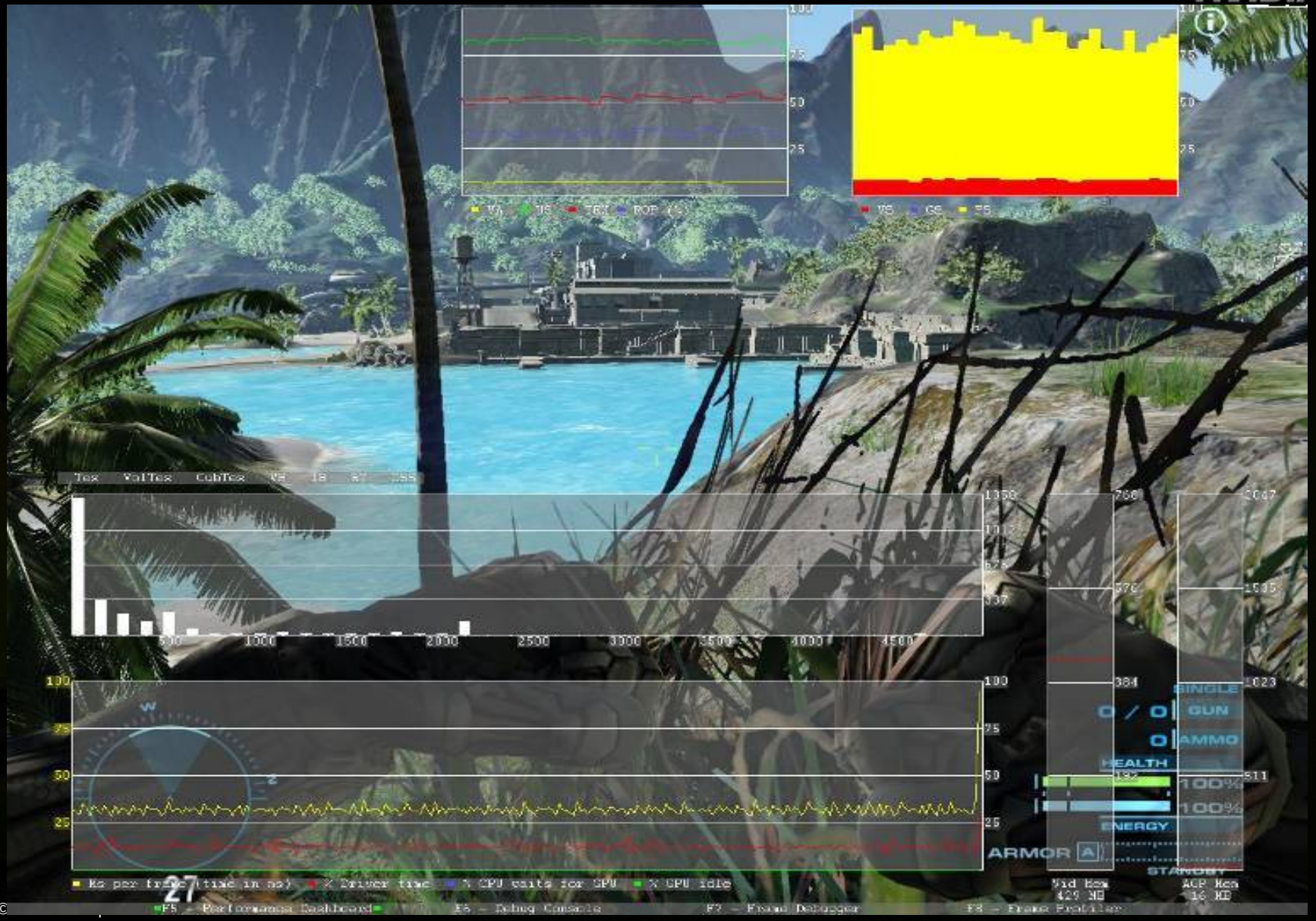
Heavy Geometry
Workload Perf = 11



Heavy Pixel
Workload Perf = 11

© NVIDIA Corporation 2007

Unified


Tex
VolTex
CubTex
SINGLE
GUN
AMMO
HEALTH
100%
100%
ENERGY
ARMOR
STANDBY
F5 - Performance Dashboard
F6 - Debug Console
F7 - Frame Debugger
F8 - Frame Profiler

©

Dynamic Load Balancing – Company of Heroes

Less Geometry

More Geometry

High **pixel shader** use

Low **vertex shader** use

Balanced use of **pixel shader** and **vertex shader**

Unified Shader

© NVIDIA Corporation 2007