SIGGRAPH2006

GPU Shading and Rendering
Course 3
July 30, 2006

NVIDIA Graphics, Cg, and Transparency

Mark Kilgard

Graphics Software Engineer

NVIDIA Corporation

SIGGRAPH2006

Outline

- NVIDIA graphics hardware
  - seven years for GeForce + the future
- Cg—C for Graphics
  - the cross-platform GPU programming language
- Depth peeling
  - out-of-order transparency now practical

Seven Years of GeForce

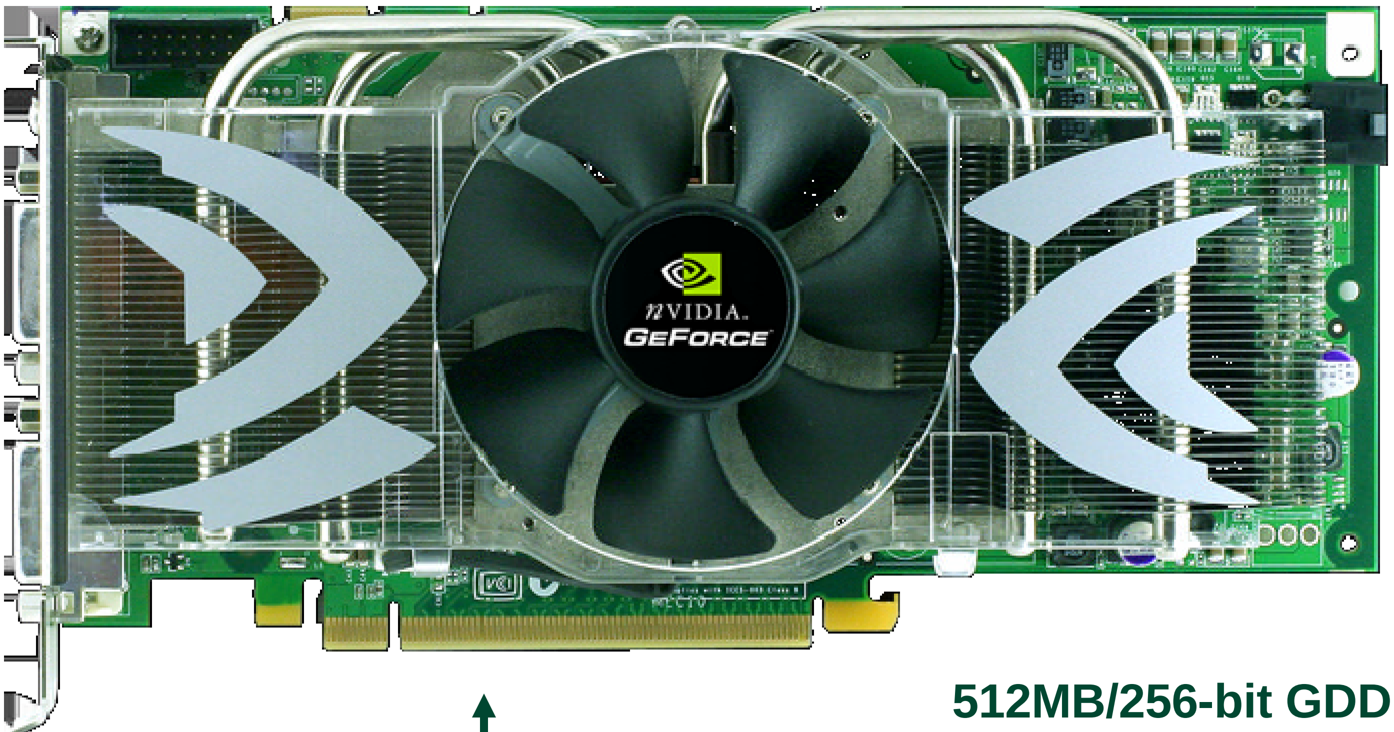
	Product	New Features	OpenGL Version	Direct3D Version
2000	GeForce 256	Hardware transform & lighting, configurable fixed-point shading, cube maps, texture compression, anisotropic texture filtering	1.3	DX7
2001	GeForce3	Programmable vertex transformation, 4 texture units, dependent textures, 3D textures, shadow maps, multisampling, occlusion queries	1.4	DX8
2002	GeForce4 Ti 4600	Early Z culling, dual-monitor	1.4	DX8.1
2003	GeForce FX	Vertex program branching, floating-point fragment programs, 16 texture units, limited floating-point textures, color & depth compression	1.5	DX9
2004	GeForce 6800 Ultra	Vertex textures, structured fragment branching, non-power-of-two textures, generalized floating-point textures, floating-point texture filtering and blending, dual-GPU	2.0	DX9c
2005	GeForce 7800 GTX	Transparency antialiasing, quad-GPU	2.0	DX9c
2006	GeForce 7900 GTX	Single-board dual-GPU, process efficiency	2.1	DX9c

2006: the GeForce 7900 GTX board

SLI Connector

sVideo
TV Out

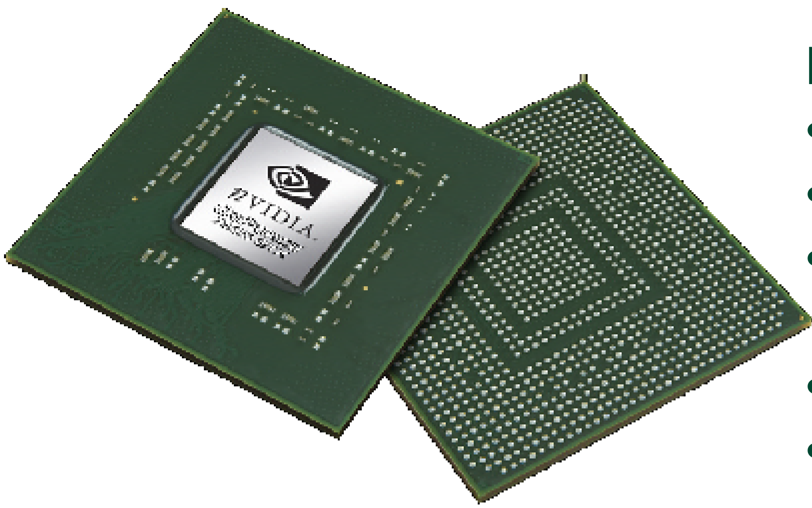
DVI x 2



16x PCI-Express

512MB/256-bit GDDR3
1600 MHz effective
8 pieces of 8Mx32

2006: the GeForce 7900 GTX GPU



278 million transistors
650 MHz core clock
1,600 MHz GDDR3 effective memory clock
256-bit memory interface

Notable Functionality

- Non-power-of-two textures with mipmaps
- Floating-point (fp16) blending and filtering
- sRGB color space texture filtering and frame buffer blending
- Vertex textures
- 16x anisotropic texture filtering
- Dynamic vertex *and* fragment branching
- Double-rate depth/stencil-only rendering
- Early depth/stencil culling
- Transparency antialiasing

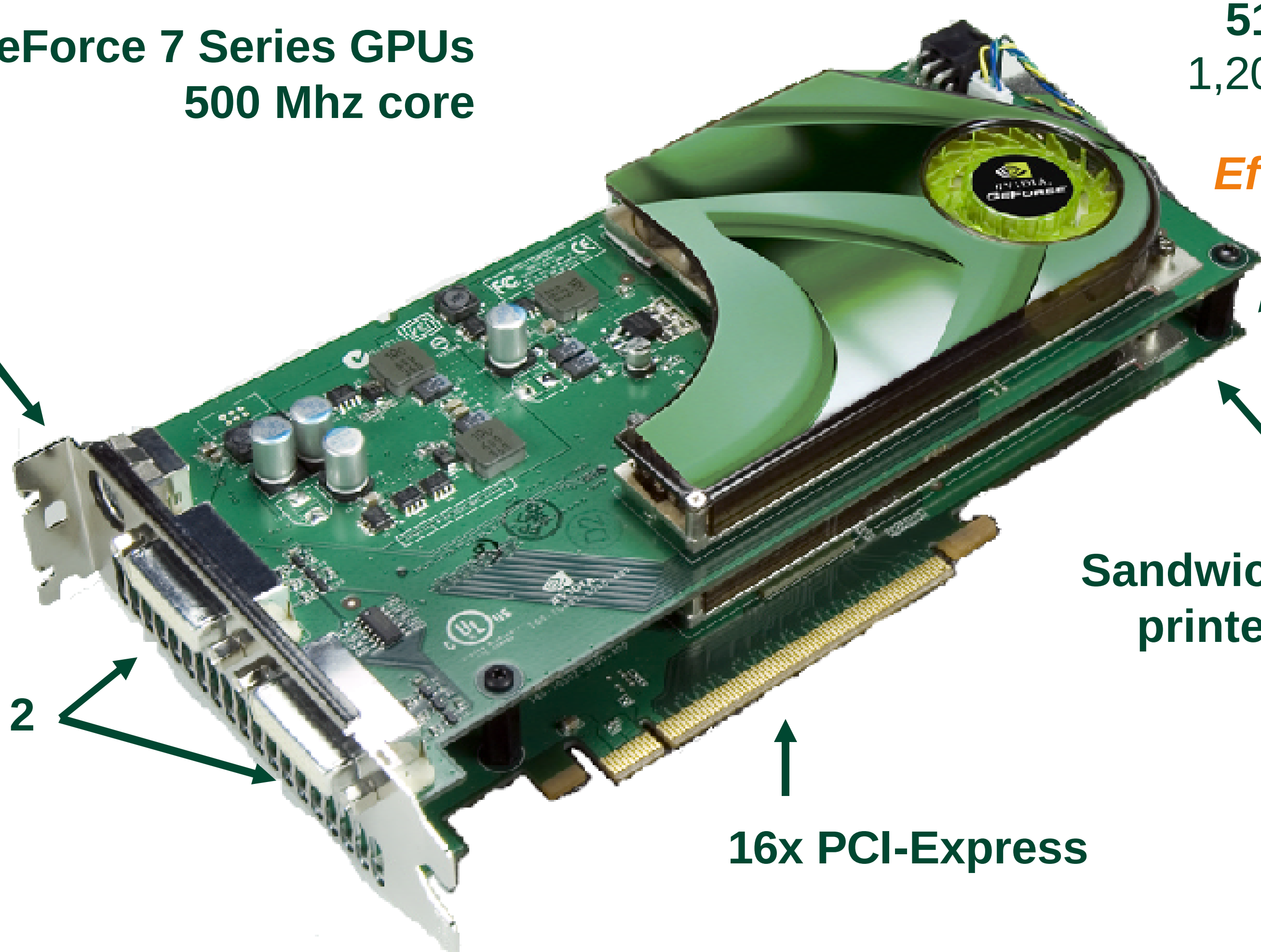
2006: GeForce 7950 GX2, SLI-on-a-card

1 GB video memory
512 MB per GPU
1,200 Mhz effective

Two GeForce 7 Series GPUs
500 Mhz core

Effective 512-bit memory interface!

sVideo TV Out



Sandwich of two printed circuit boards

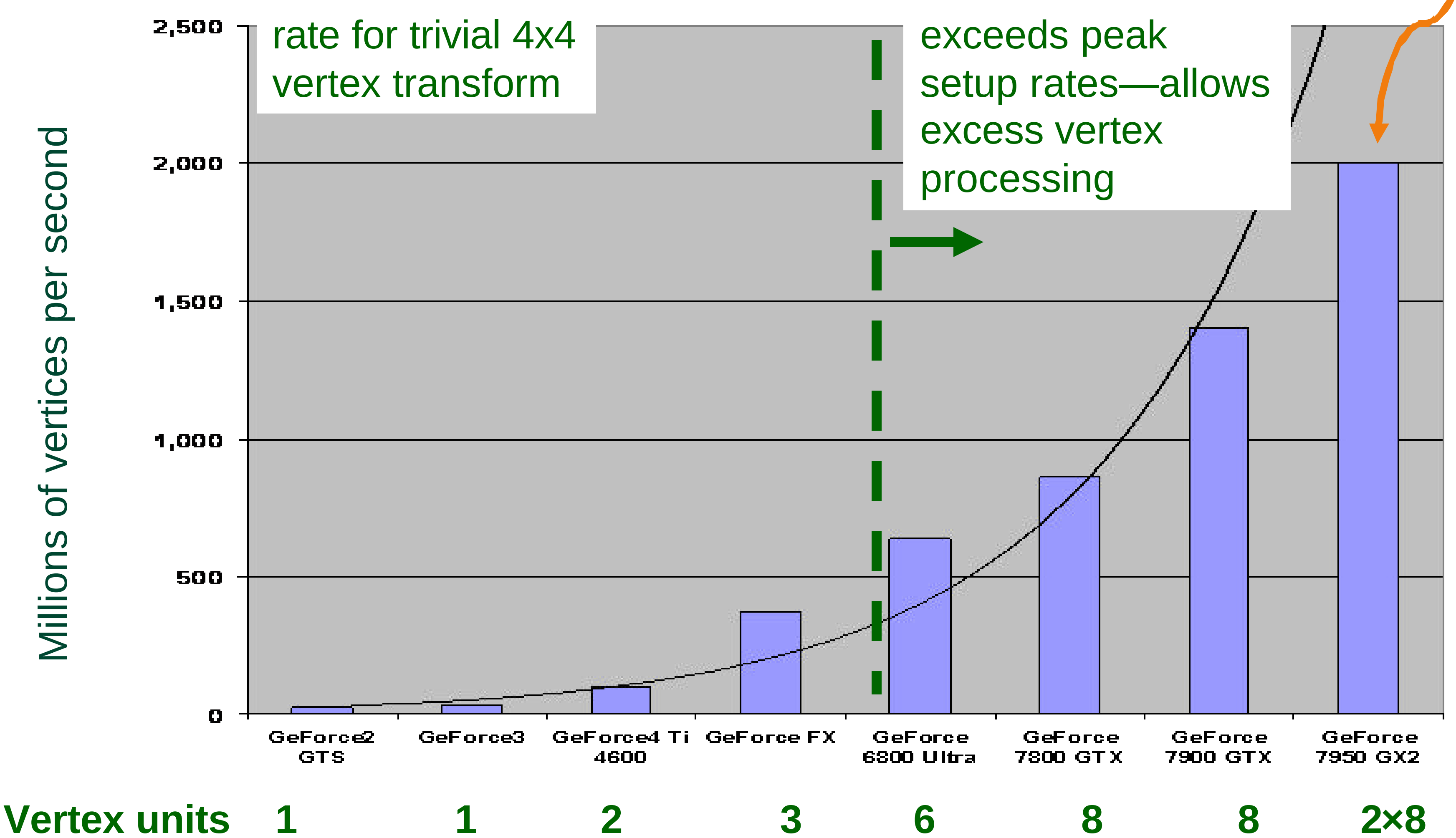
DVI x 2

16x PCI-Express

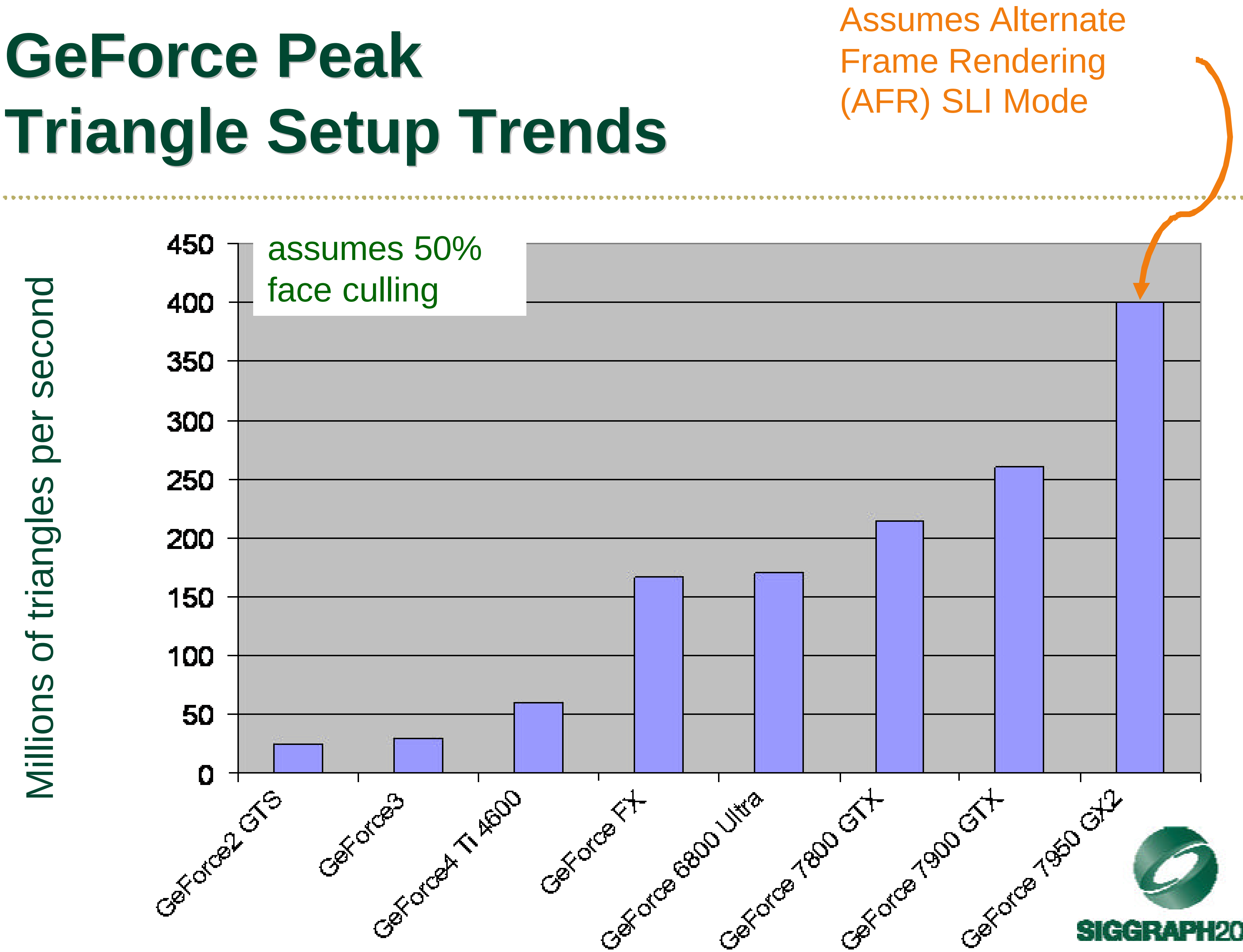
SIGGRAPH2006

GeForce Peak Vertex Processing Trends

Assumes Alternate Frame Rendering (AFR) SLI Mode

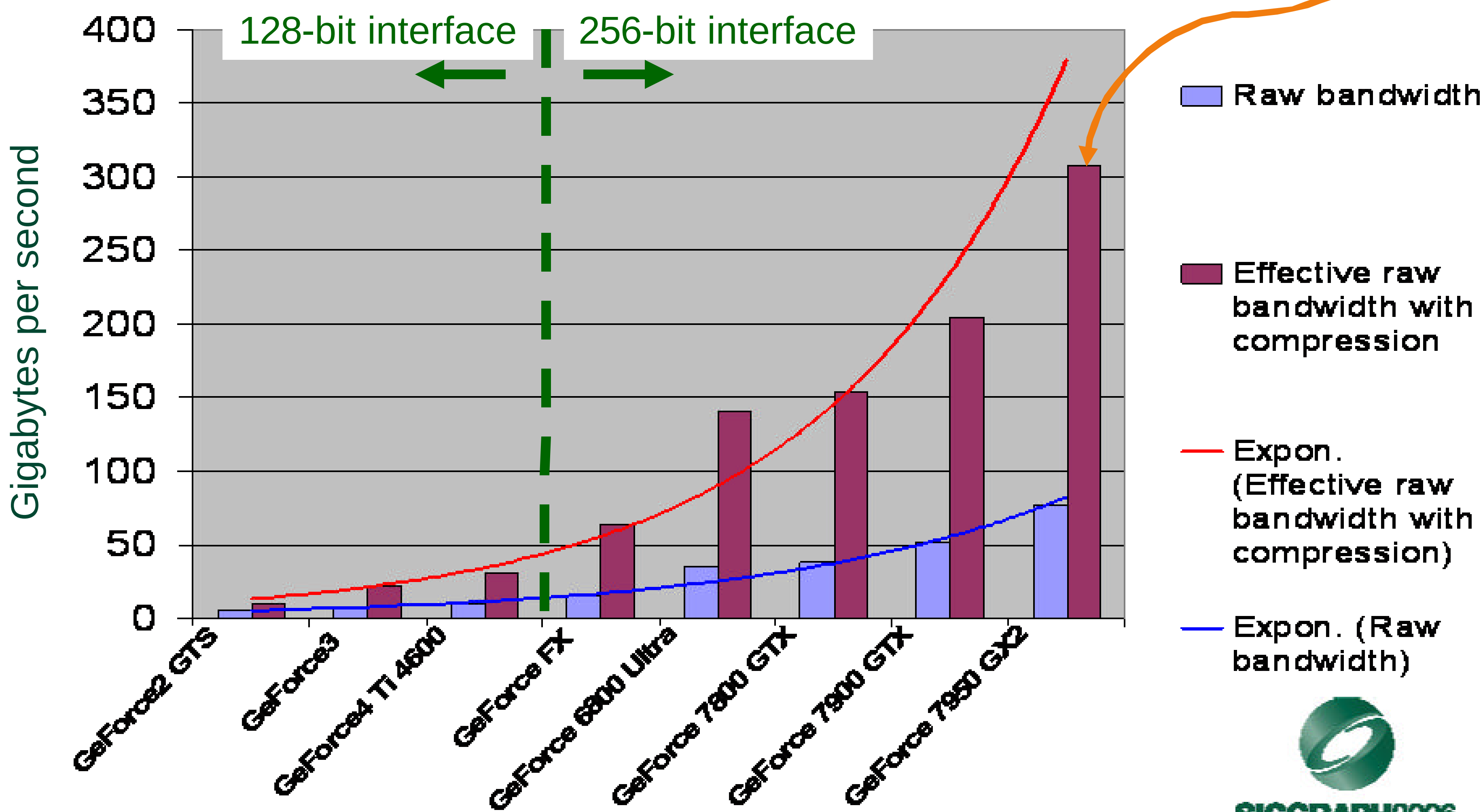


GeForce Peak Triangle Setup Trends



GeForce Peak Memory Bandwidth Trends

Two physical 256-bit memory interfaces



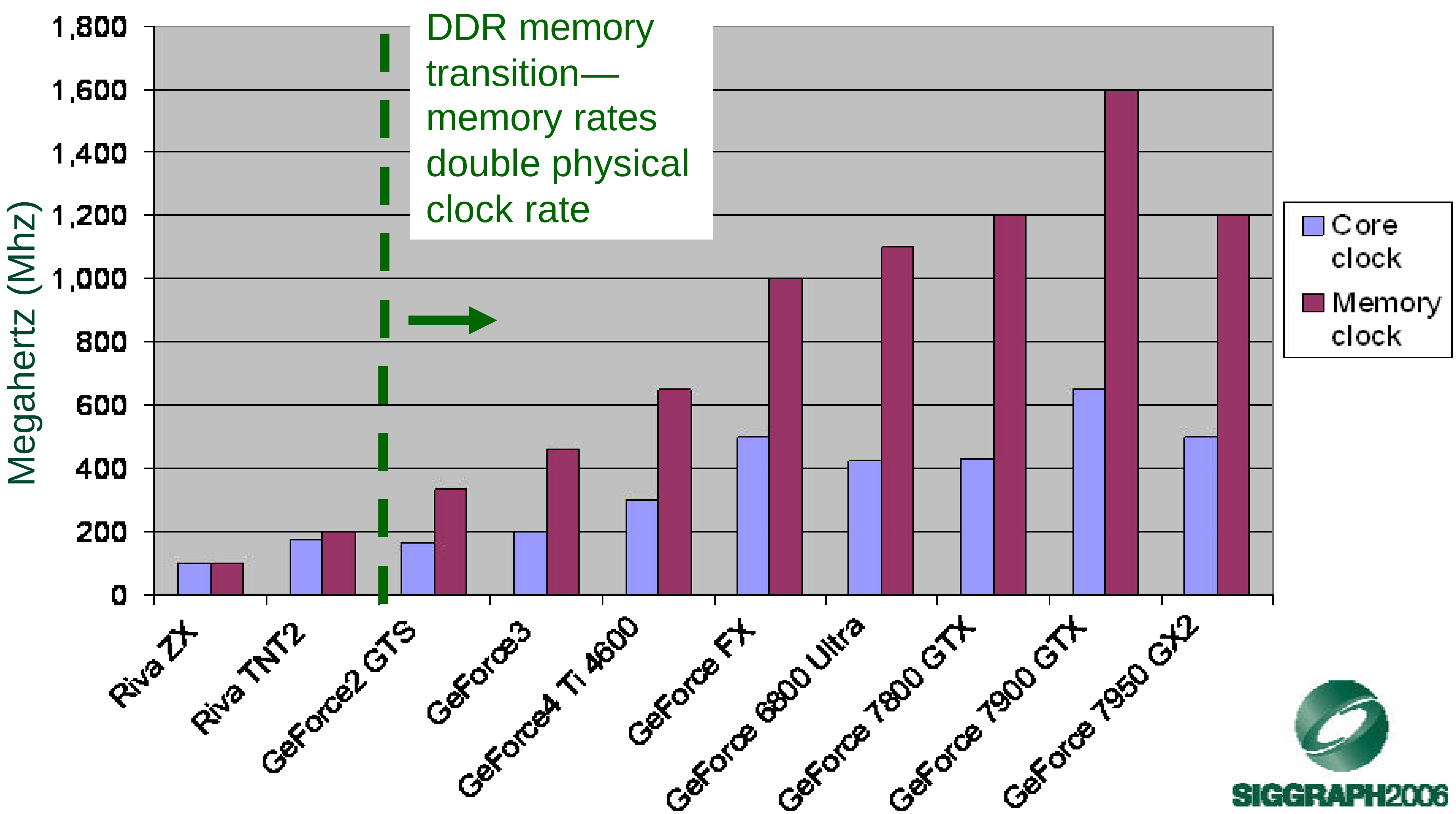
SIGGRAPH2006

Effective GPU Memory Bandwidth

- **Compression schemes**
  - Lossless depth and color (when multisampling) compression
  - Lossy texture compression (S3TC / DXTC)
  - Typically assumes 4:1 compression
- **Avoid useless work**
  - Early killing of fragments (Z cull)
  - Avoid useless blending and texture fetches
- **Very clever memory controller designs**
  - Combining memory accesses for improved coherency
  - Caches for texture fetches

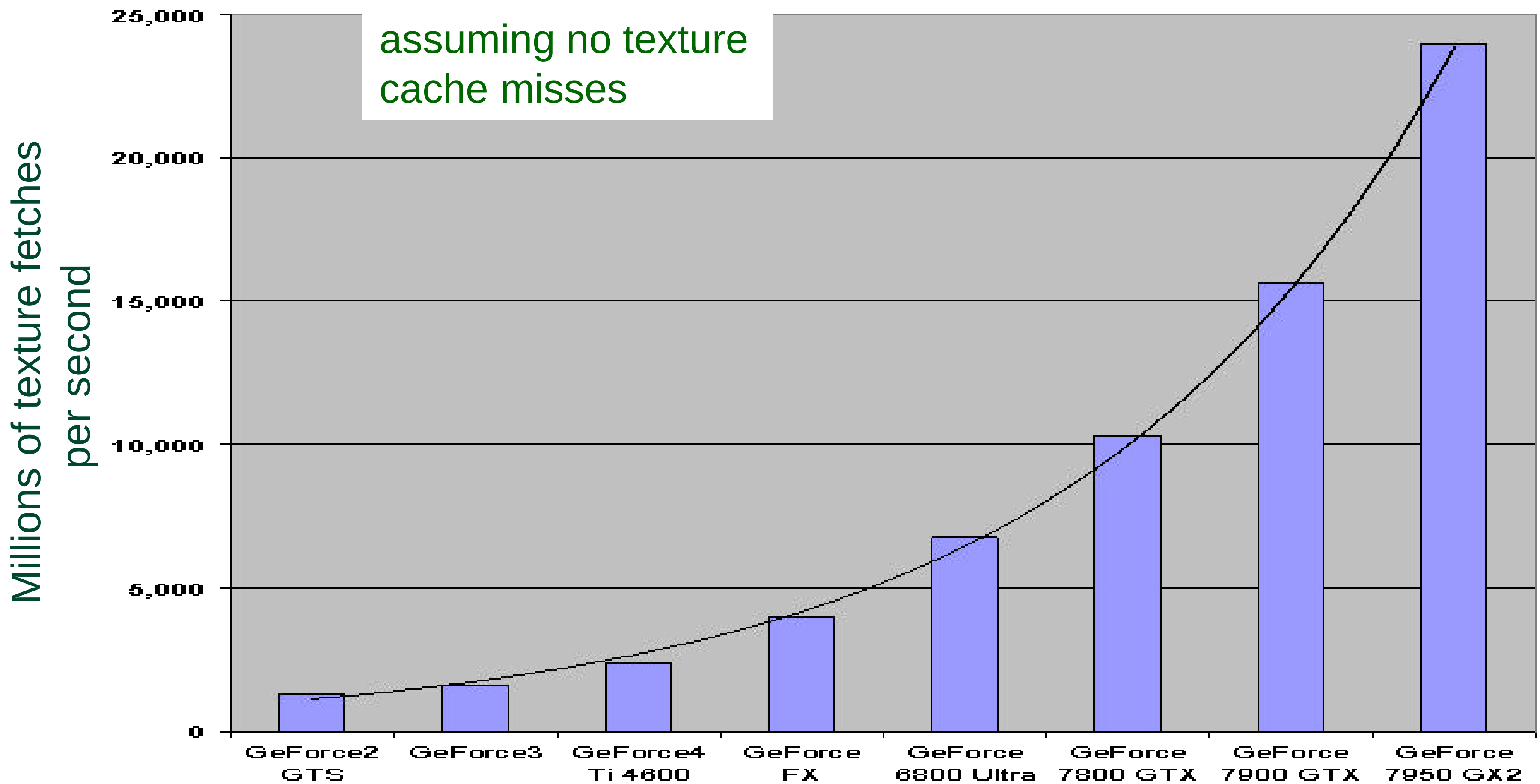
SIGGRAPH2006

NVIDIA Graphics Core and Memory Clock Rates



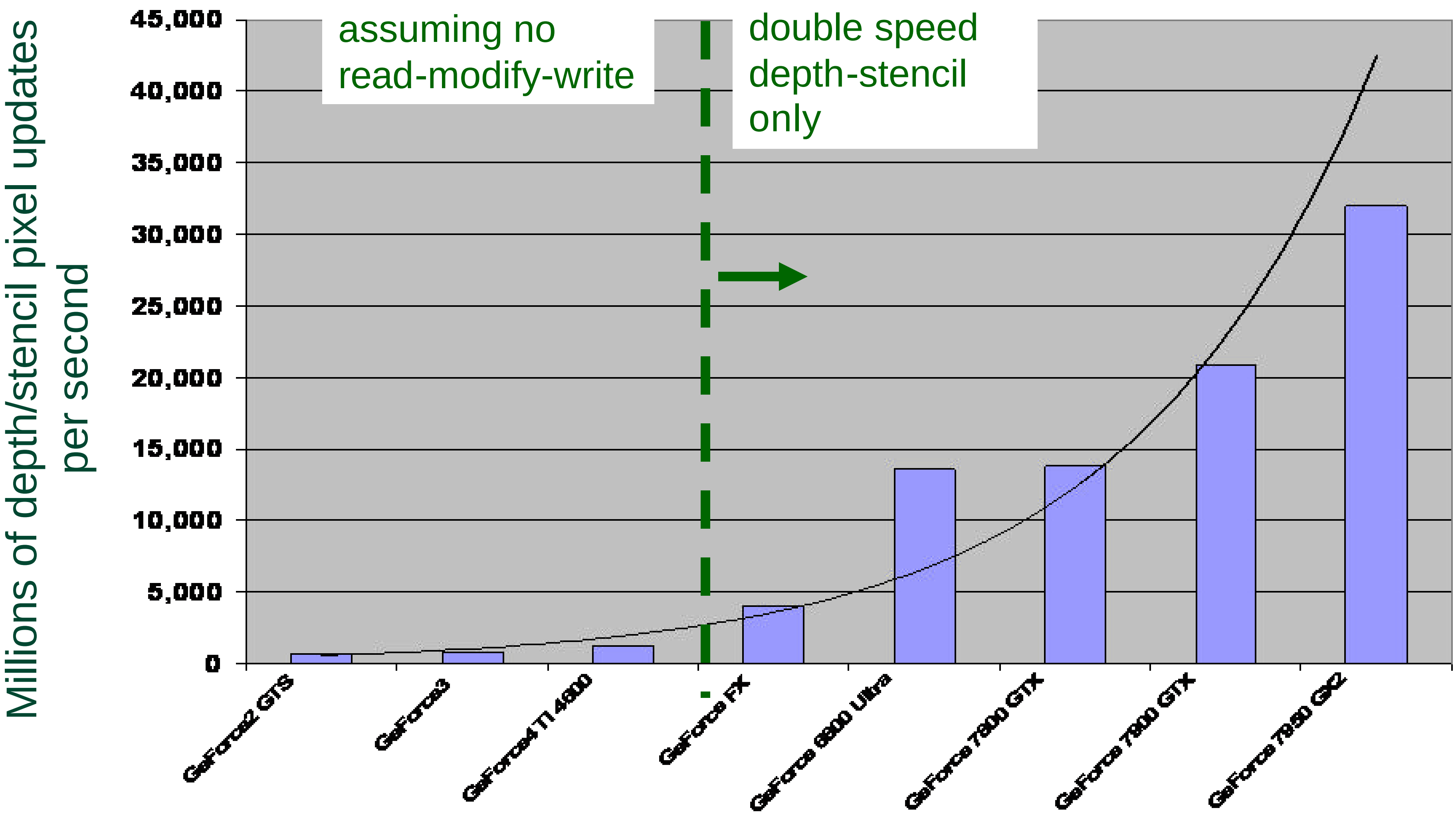
SIGGRAPH2006

GeForce Peak Texture Fetch Trends

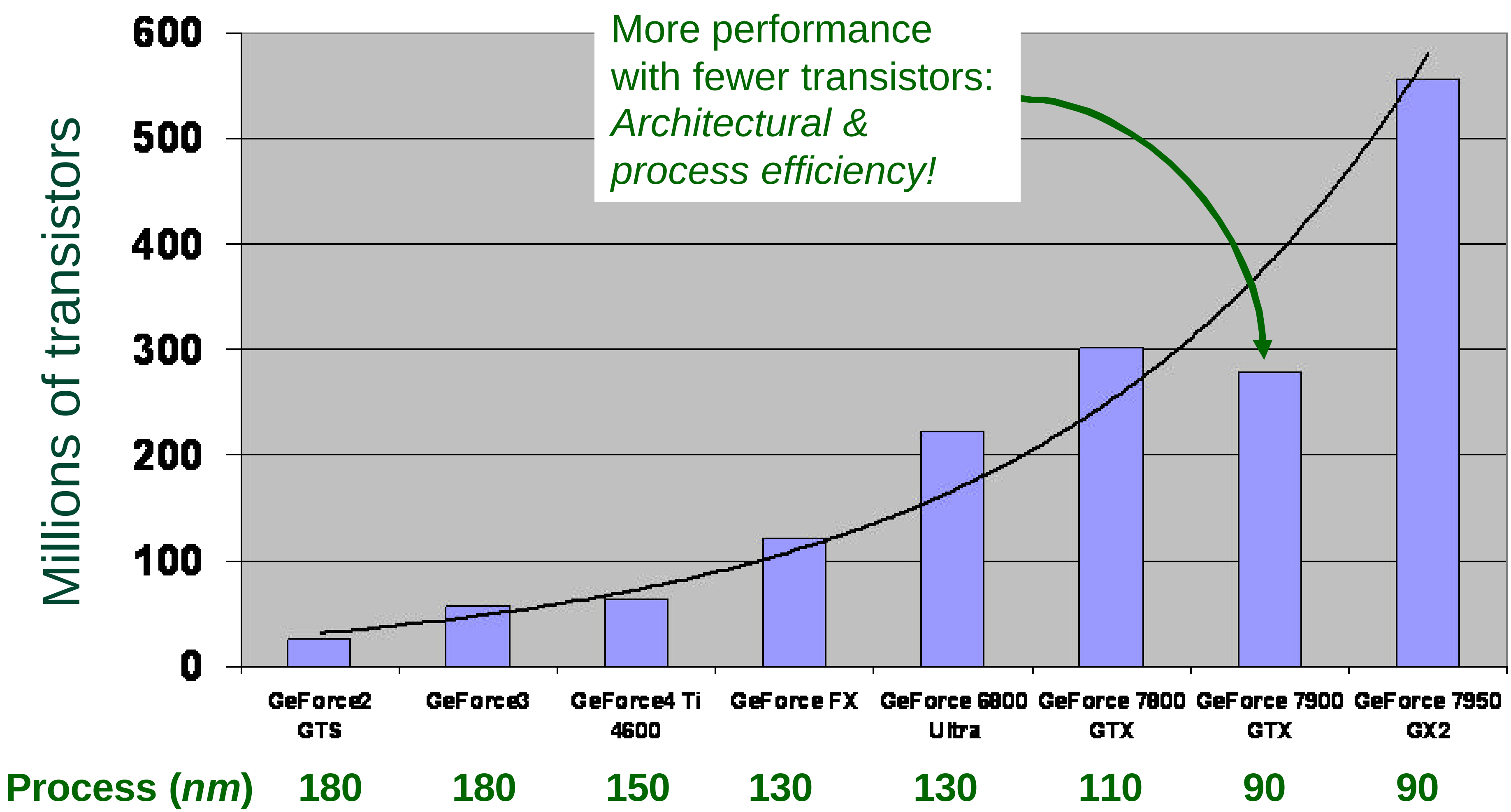


Texture units	2×4	2×4	2×4	2×4	16	24	24	2×24

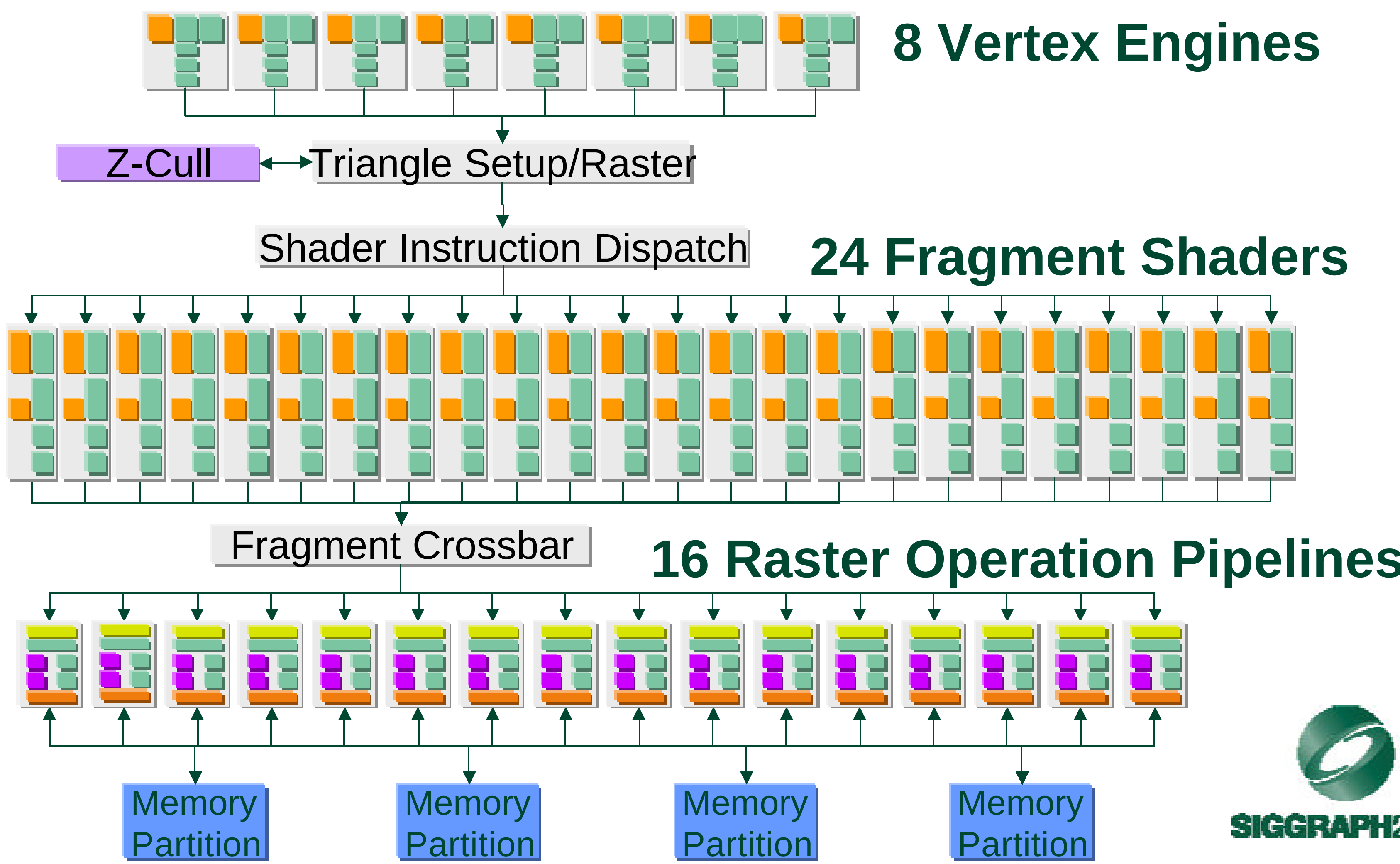
GeForce Peak Depth/Stencil-only Fill



GeForce Transistor Count and Semiconductor Process



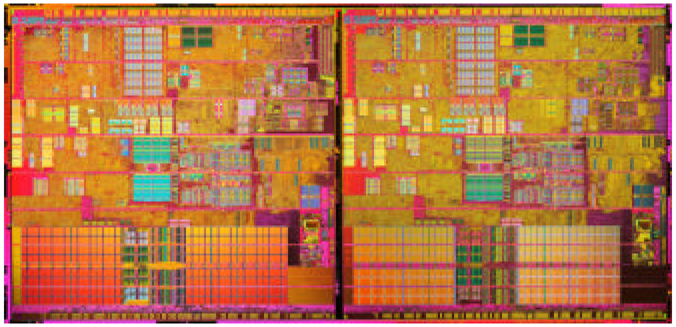
GeForce 7900 GTX Parallelism

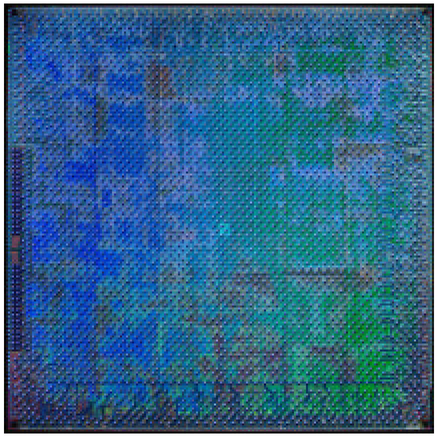


SIGGRAPH2006

Hardware Unit	GeForce FX 5900	GeForce 6800 Ultra	GeForce 7900 GTX
Vertex	3	6	8
Fragment / 2nd Texture Fetch	4+4	16	24
Raster Color / Raster Depth	4+4	16+16	16+16

2005: Comparison to CPU





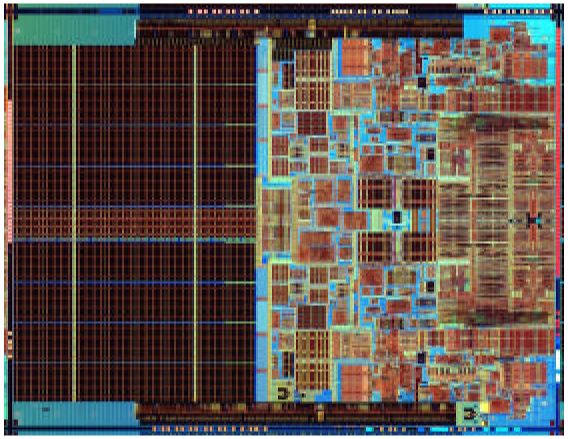
Pentium Extreme Edition 840

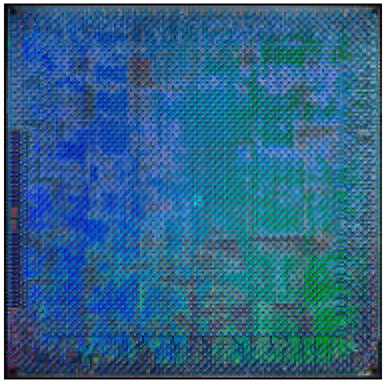
- 3.2 GHz Dual Core
- 230M Transistors
- 90nm process
- 206 mm^2
- 2 x 1MB Cache
- 25.6 GFlops

GeForce 7800 GTX

- 430 MHz
- 302M Transistors
- 110nm process
- 326 mm^2
- 313 GFlops (shader)
- 1.3 TFlops (total)

2006: Comparison to CPU





Intel Core 2 Extreme X6800

- 2.93 GHz Dual Core
- 291M Transistors
- 65nm process
- 143 mm^2
- 4MB Cache
- 23.2 GFlops

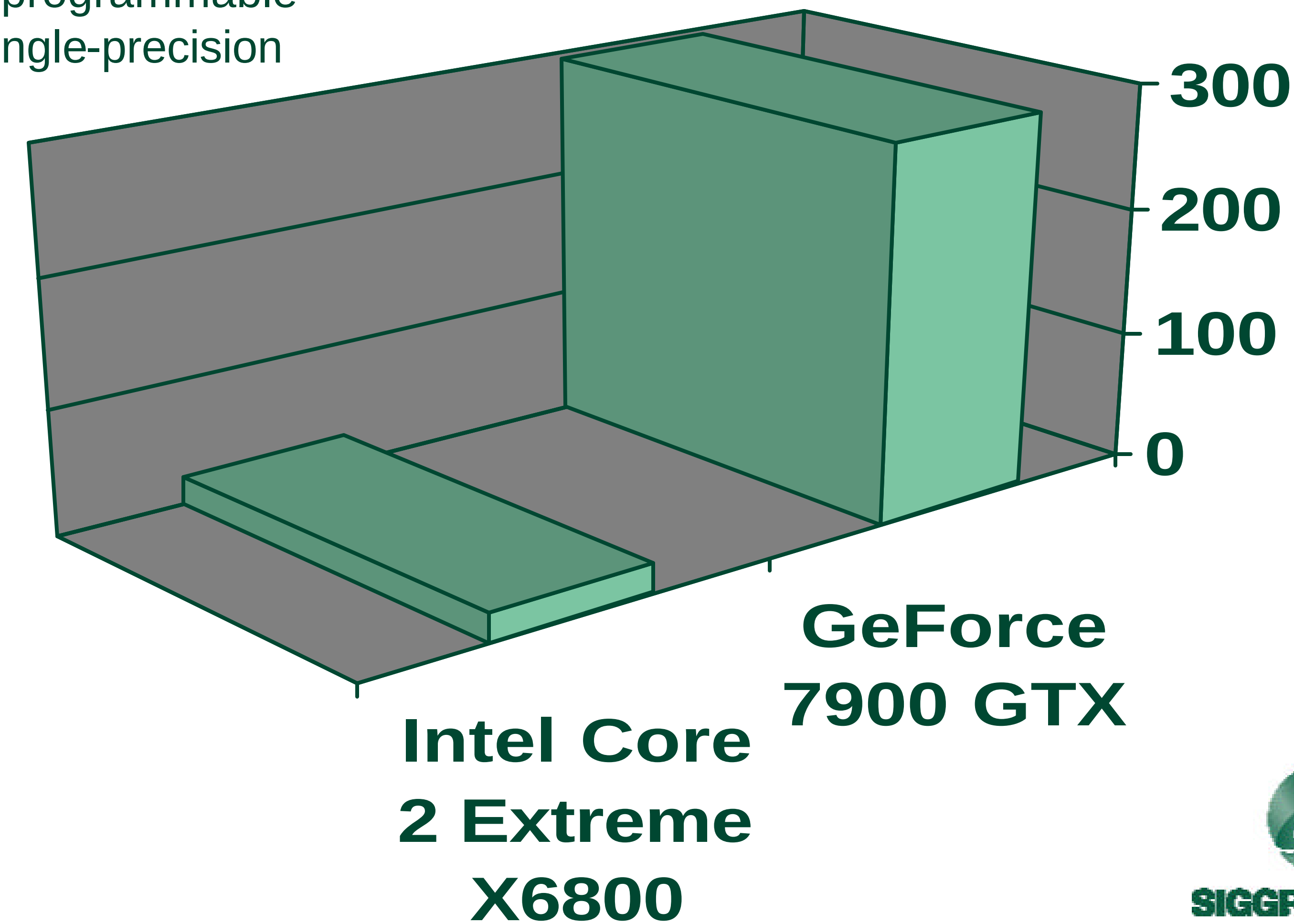
GeForce 7900 GTX

- 650 MHz
- 278M Transistors
- 90nm process
- 196 mm^2
- 477 GFlops (shader)
- 2.1 TFlops (total)

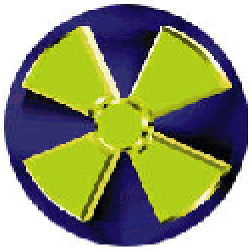
SIGGRAPH2006

Giga Flops Imbalance

Theoretical programmable
IEEE 754 single-precision
Giga Flops



Future NVIDIA GPU directions



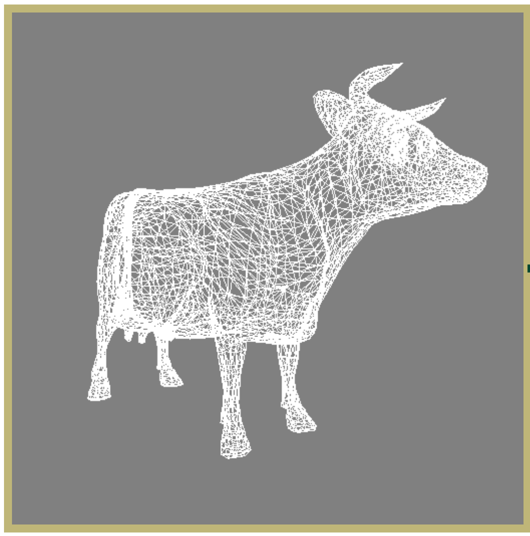
- DirectX 10 feature set
  - Massive graphics functionality upgrade
- Language and tool support
  - Performance tuning and content development
- Improved GPGPU
  - Harness the bandwidth & Gflops for non-graphics
- Multi-GPU systems innovation
  - Next-generation SLI

DirectX 10-class GPU functionality

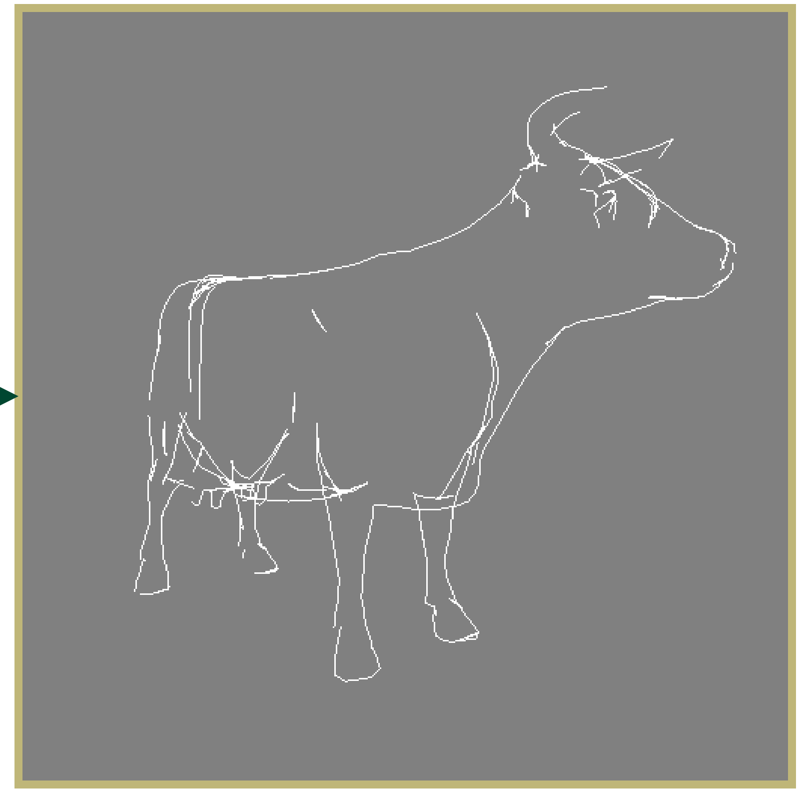
- Generalized programmability, including
  - Integer instructions
  - Efficient branching
  - Texture size queries, unfiltered texel fetches, & offset fetches
  - Shadow cube maps for omni-directional shadowing
  - Sourcing constants from bind-able buffer objects
- Per-primitive programmable processing
  - Emits zero or more strips of triangles/points/lines
  - New line and triangle adjacency primitives
  - Output to multiple viewports and buffers

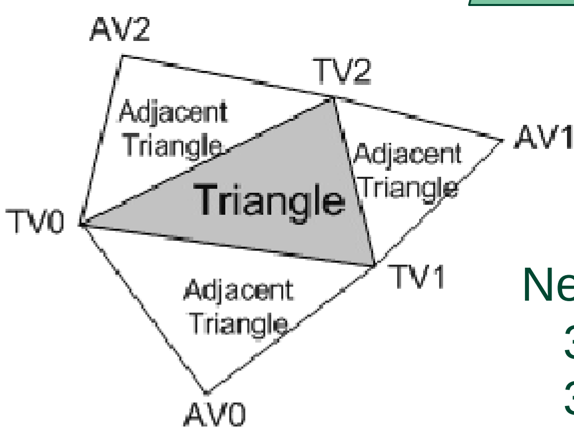
SIGGRAPH2006

Per-primitive processing example: Automatic silhouette edge rendering



emit edge of adjacent triangles that face opposite directions





New triangle adjacency primitive =
- 3 conventional vertices +
- 3 vertices for adjacent triangles

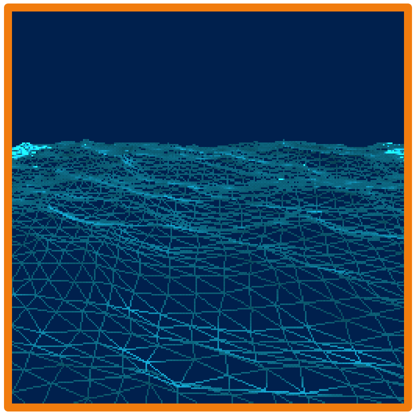
SIGGRAPH2006

More DirectX 10-class GPU functionality

- Better blending
  - Improved blending control for multiple draw buffers
  - sRGB and 32-bit floating-point framebuffer blending
- Streamed output of vertex processing to buffers
  - Render to vertex array
- Texture improvements
  - Indexing into an “array” of 2D textures
  - Improved render-to-texture
  - Luminance-alpha compressed formats
  - Compact High Dynamic Range texture formats
  - Integer texture formats
  - 32-bit floating-point texture filtering

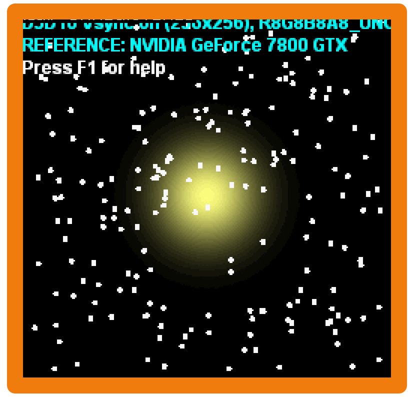
SIGGRAPH2006

Uses of DirectX 10 functionality

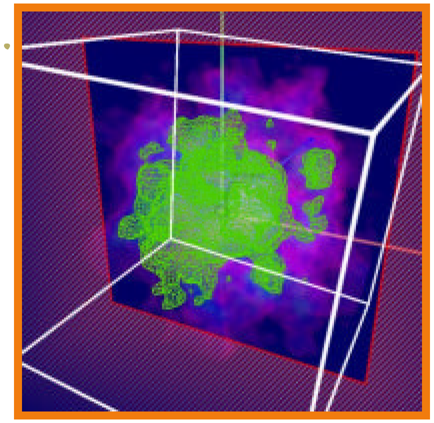


Deep Waves

Sparkling Sprites



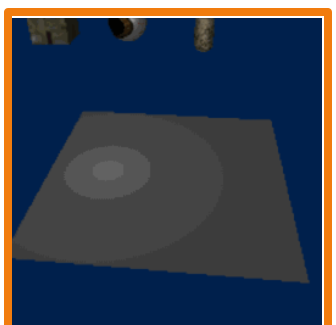
GPU Fluid Simulation



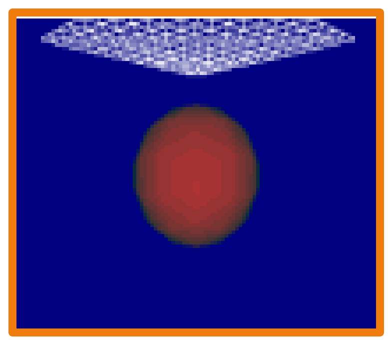
GPU Marching Cubes

Table-free Noise

Styled Line Drawing



Deformable Collisions

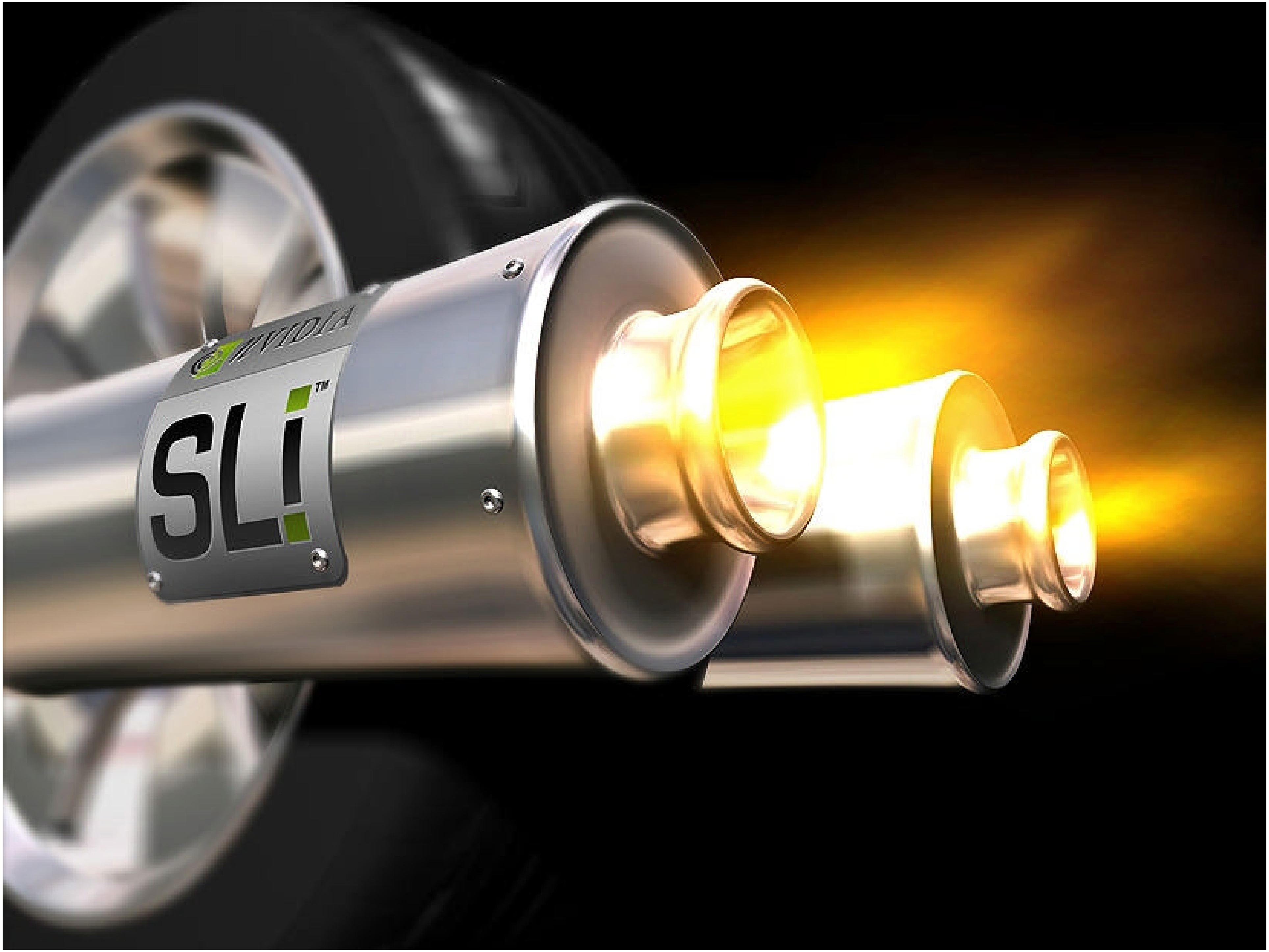


GPU Cloth

SIGGRAPH2006

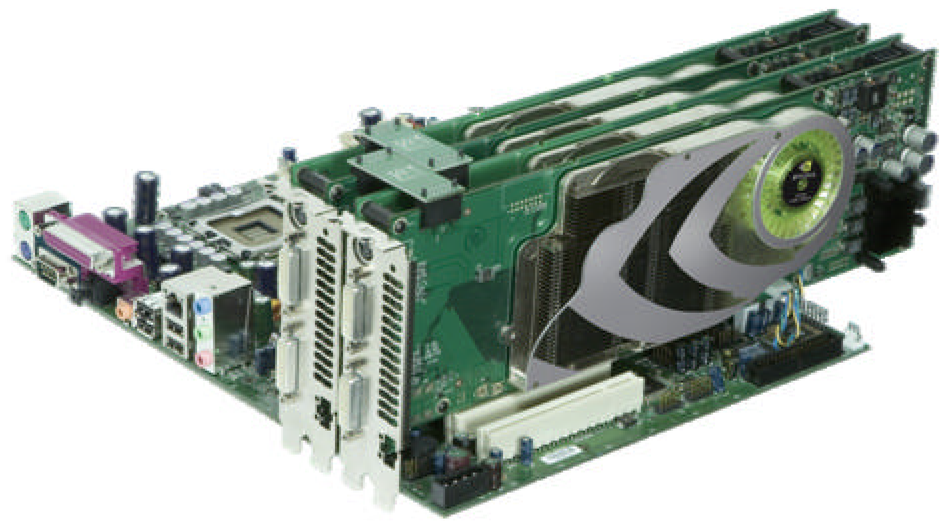
DirectX 10-class functionality parity

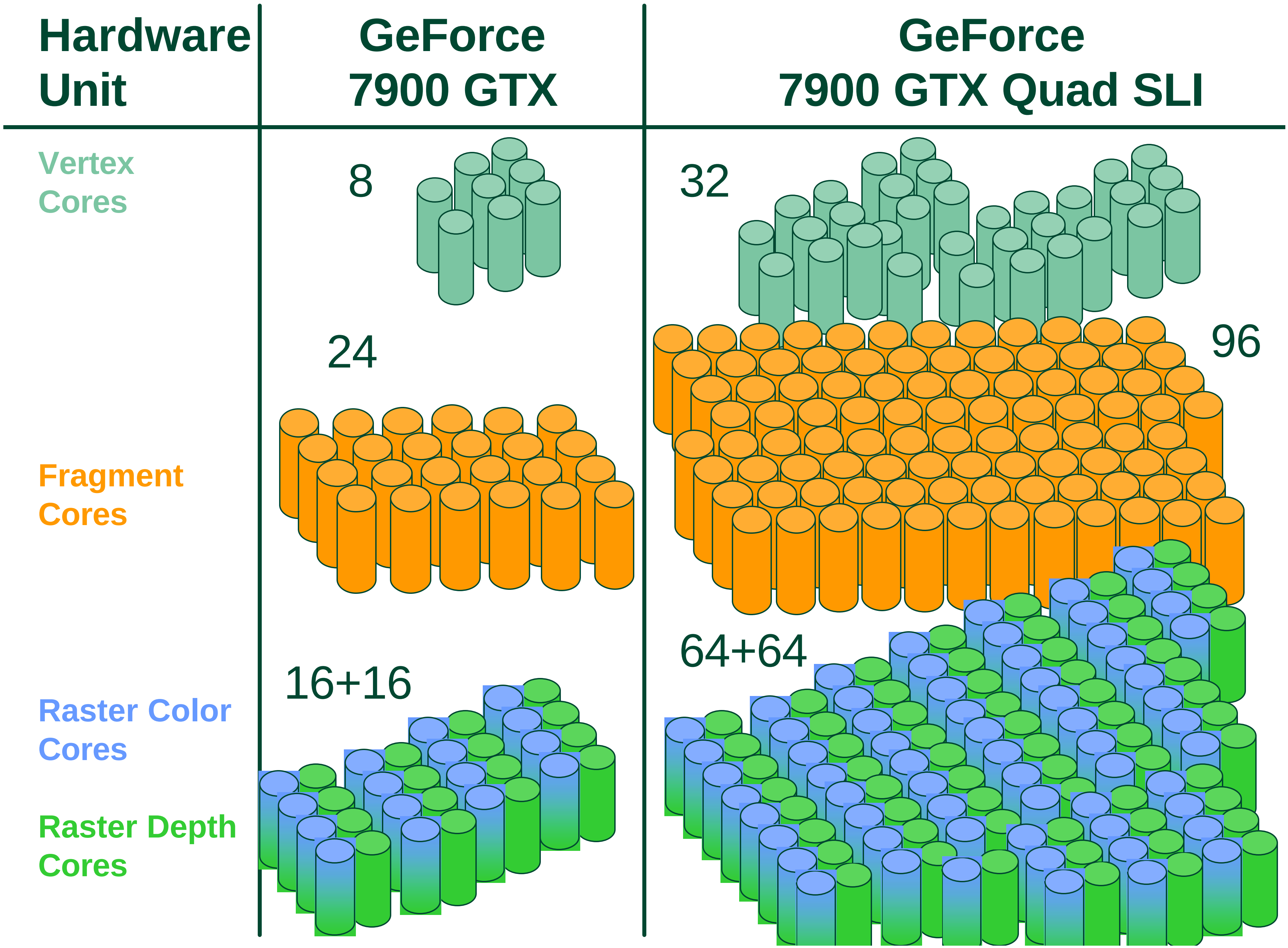
- Feature parity
  - DirectX 10-class features available via OpenGL
  - Cross API portability of programmable shading content through Cg
- Performance parity
  - 3D API agnostic performance parity on all Windows operating systems
- System support parity
  - Linux, Mac, FreeBSD, Solaris
  - Shared code base for drivers


SLI™

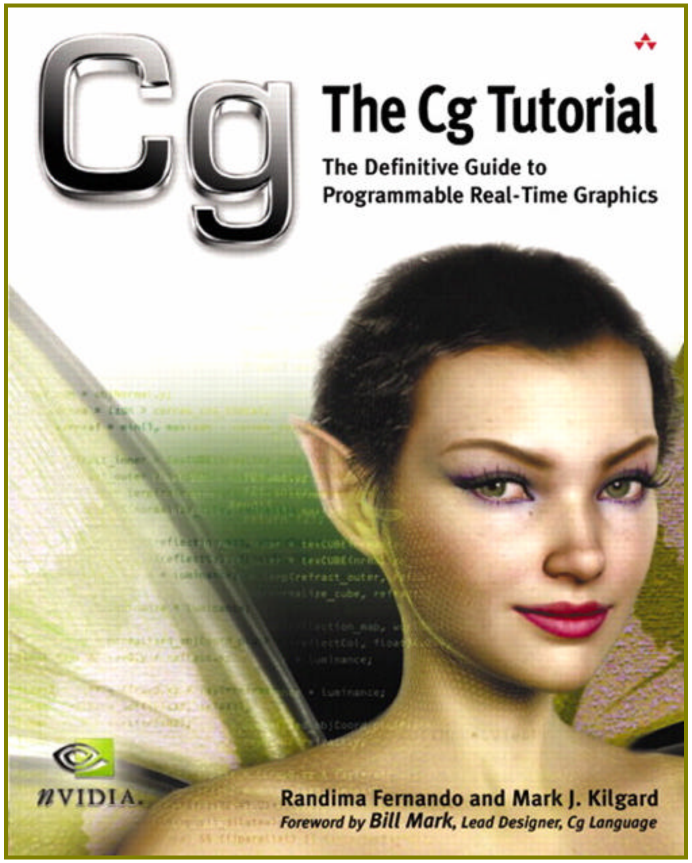
Multi-GPU Support

- Original SLI was just the beginning
  - Quad-SLI
  - SLI support infuses all NVIDIA product design and development
- New SLI APIs for application-control of multiple GPUs
- SLI for notebooks
  - Better thermals and power




Hardware Unit
GeForce 7900 GTX
GeForce 7900 GTX Quad SLI
Vertex Cores
8
32
Fragment Cores
24
96
Raster Color Cores
Raster Depth Cores
16+16
64+64

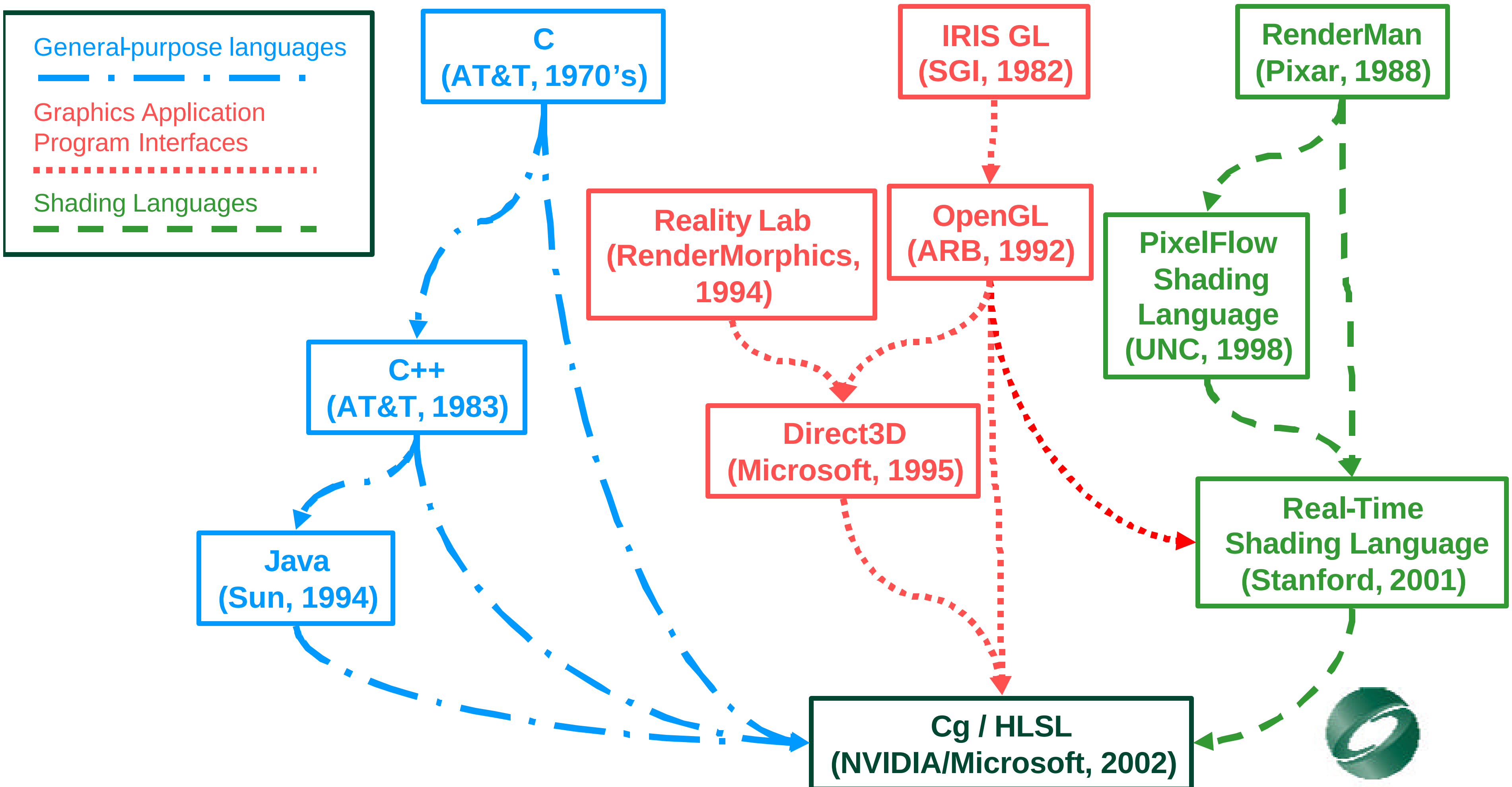
Cg: C for Graphics



Cg: C for Graphics

- Cg as it exists today
  - High-level, inspired mostly by C
  - Graphics focused
    - API-independent
      - GLSL tied to OpenGL; HLSL tied to Direct3D; Cg works for both
    - Platform-independent
      - Cg works on PlayStation 3, ATI, NVIDIA, Linux, Solaris, Mac OS X, Windows, etc.
- Production language and system
  - Cg 1.5 is part of 3D content creation tool chains
  - Portability of Cg shaders is important

Evolution of Cg

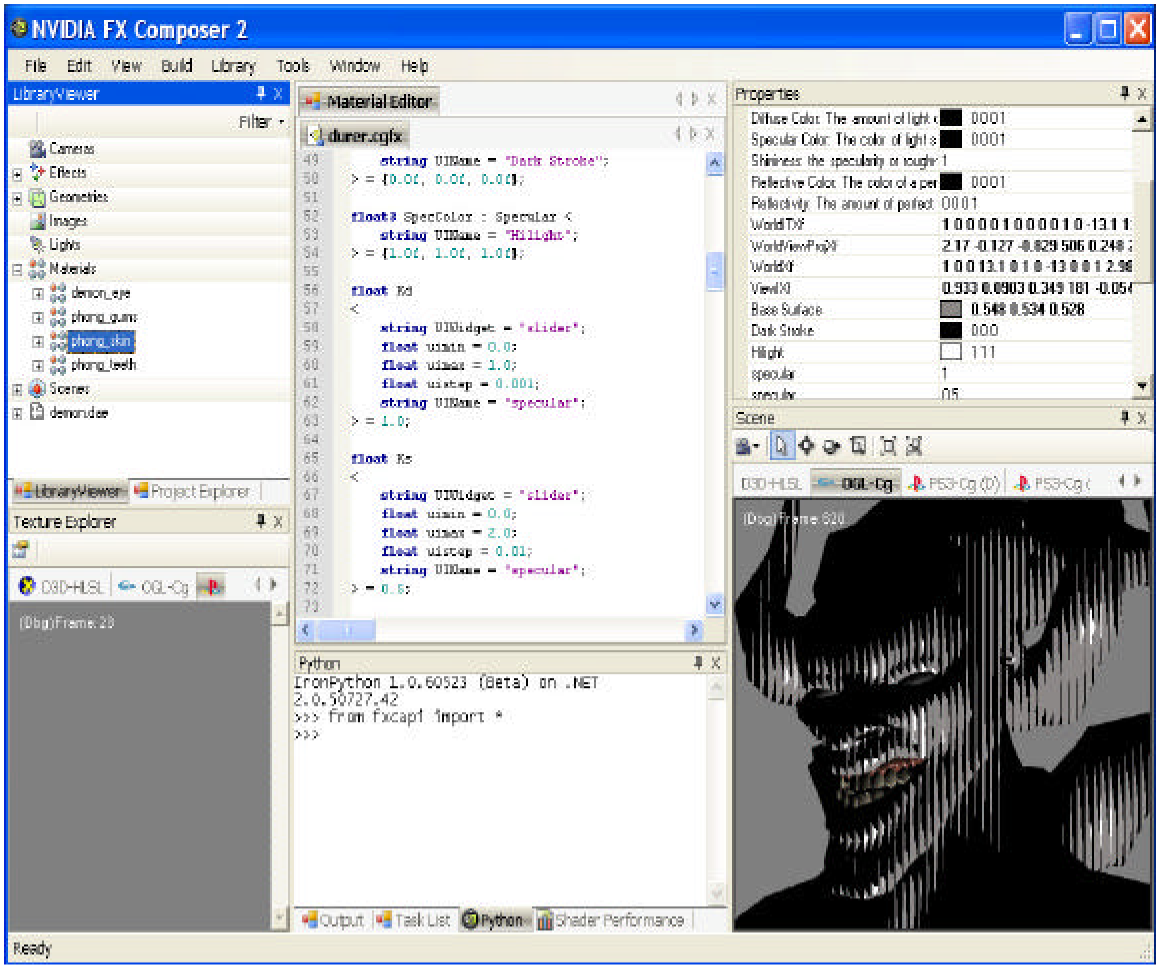


SIGGRAPH2006

Cg 1.5

- Current release of Cg
  - Supports Windows, Linux, Mac (including x86 Macs) + now Solaris
  - Shader Model 3.0 profiles for Direct3D 9.0c
  - Matches Sony's PlayStation 3 Cg support
  - Tool chain support: FX Composer 2.0
- New functionality
  - Procedural effects generation
  - Combined programs for multiple domains
  - New GLSL profiles to compile Cg to GLSL
- Improved compiler optimization

FX Composer for Cg shader authoring



- Shaders are assets
  - Portability matters
- So express shaders in a multi-platform, multi-API language
  - That's Cg

SIGGRAPH2006

Future: Modernizing Cg

- Opportunity to re-think the Cg language
  - Experience-driven
  - Shader writing was programming-in-the-small
    - But not anymore!
    - Provide better abstraction mechanisms
  - Must be backward compatible
- **Challenge**: Instead of inventing yet-another shading language-specific keyword, think how a C++ programmer express the feature
  - Think templates and classes

Cg Directions

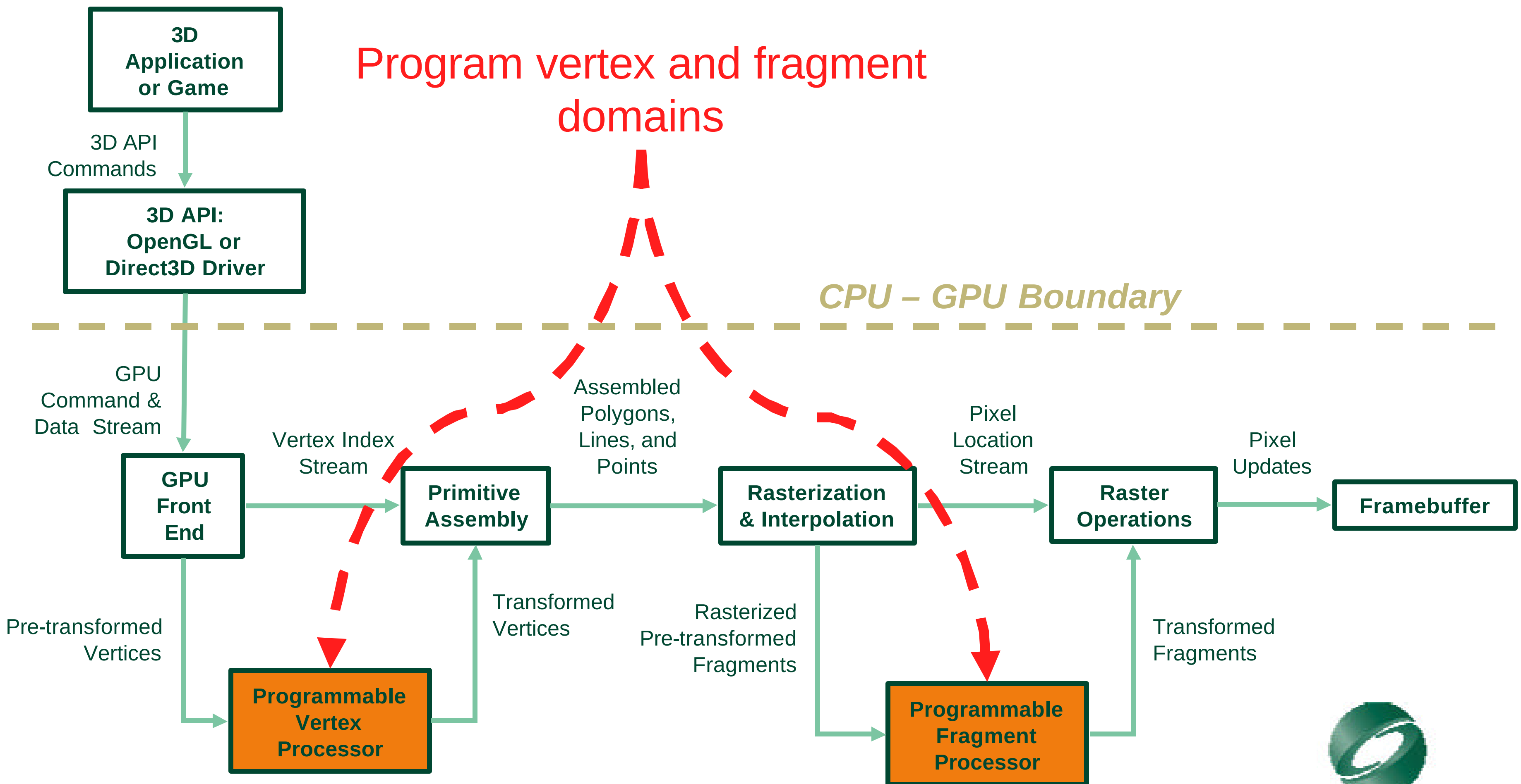
- DirectX 10-class feature support
  - Primitive (geometry) programs
  - Constant buffers
  - Interpolation modes
  - Read-write index-able temporaries
  - New texture targets: texture arrays, shadow cube maps
- Incorporate established C++ features, examples:
  - Classes
  - Templates
  - Operator overloading
  - But <u>not</u> runtime features like new/delete, RTTI, or exceptions

Why C++?

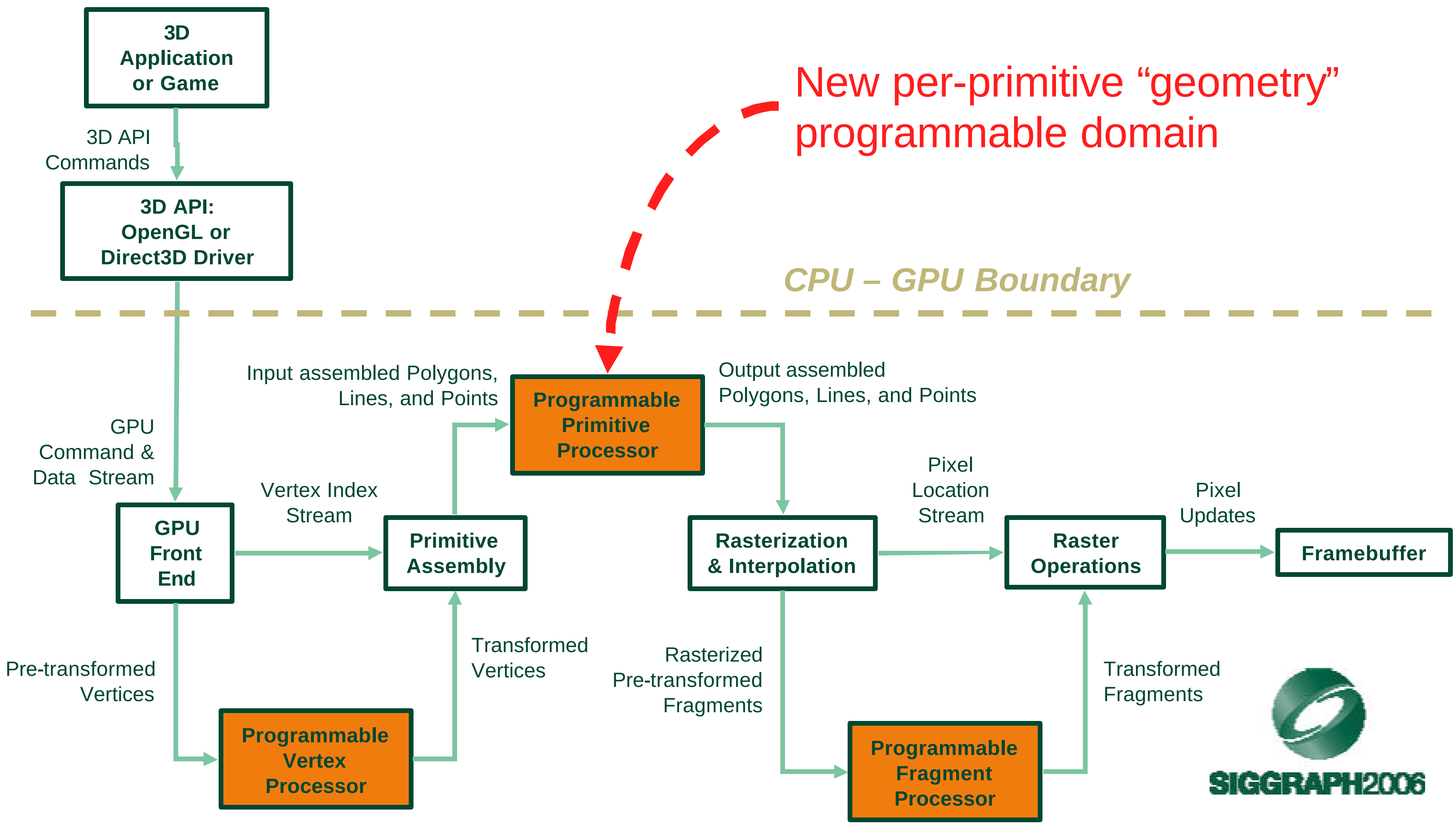
- Already inspiration for much of Cg
  - Think of Cg’s first-class vectors simply as classes
- Functionality in C++ is well-understood and popular
- C++ is biased towards compile-time abstraction
  - Rather than more run-time focus of Java and C#
  - Compile-time abstraction is good since GPUs lack the run-time support for heaps, garbage collection, exceptions, and run-time polymorphism

SIGGRAPH2006

Logical Programmable Graphics Pipeline



Future Logical Programmable Graphics Pipeline



SIGGRAPH2006

Pass Through Geometry Program Example

flatColor initialized from constant buffer 6

Primitive's attributes arrive as "templated" attribute arrays

```
BufferInit<float4,6> flatColor;

TRIANGLE void passthru(AttribArray<float4> position : POSITION,
                       AttribArray<float4> texCoord : TEXCOORD0)
{
  flatAttrib(flatColor:COLOR);
  for (int i=0; i<position.length; i++) {
    emitVertex(position[i], texCoord[i]);
  }
}
```

Makes sure flat attributes are associated with the proper provoking vertex convention

Length of attribute arrays depends on the input primitive mode, 3 for TRIANGLE

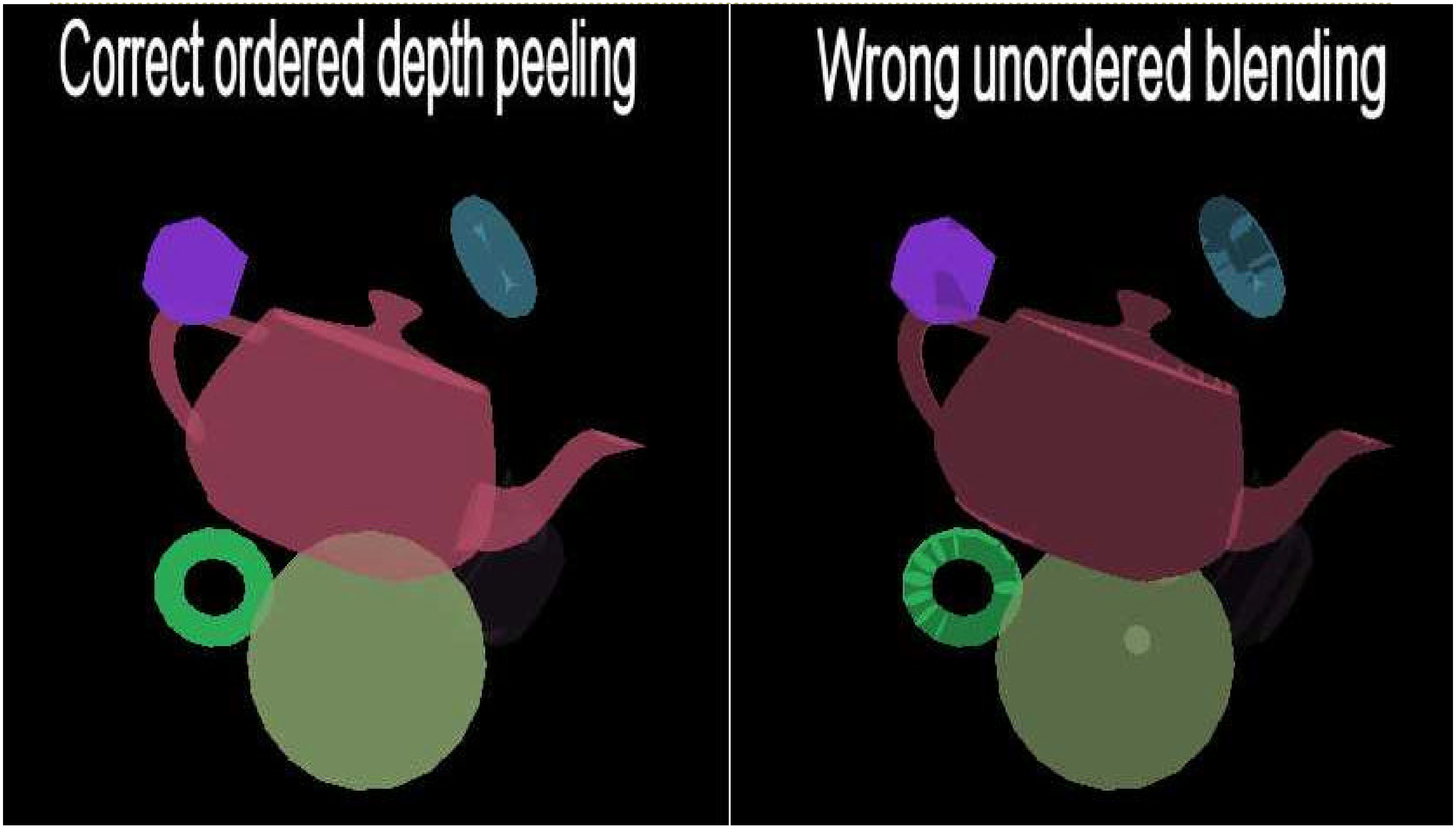
Bundles a vertex based on parameter values and semantics

SIGGRAPH2006

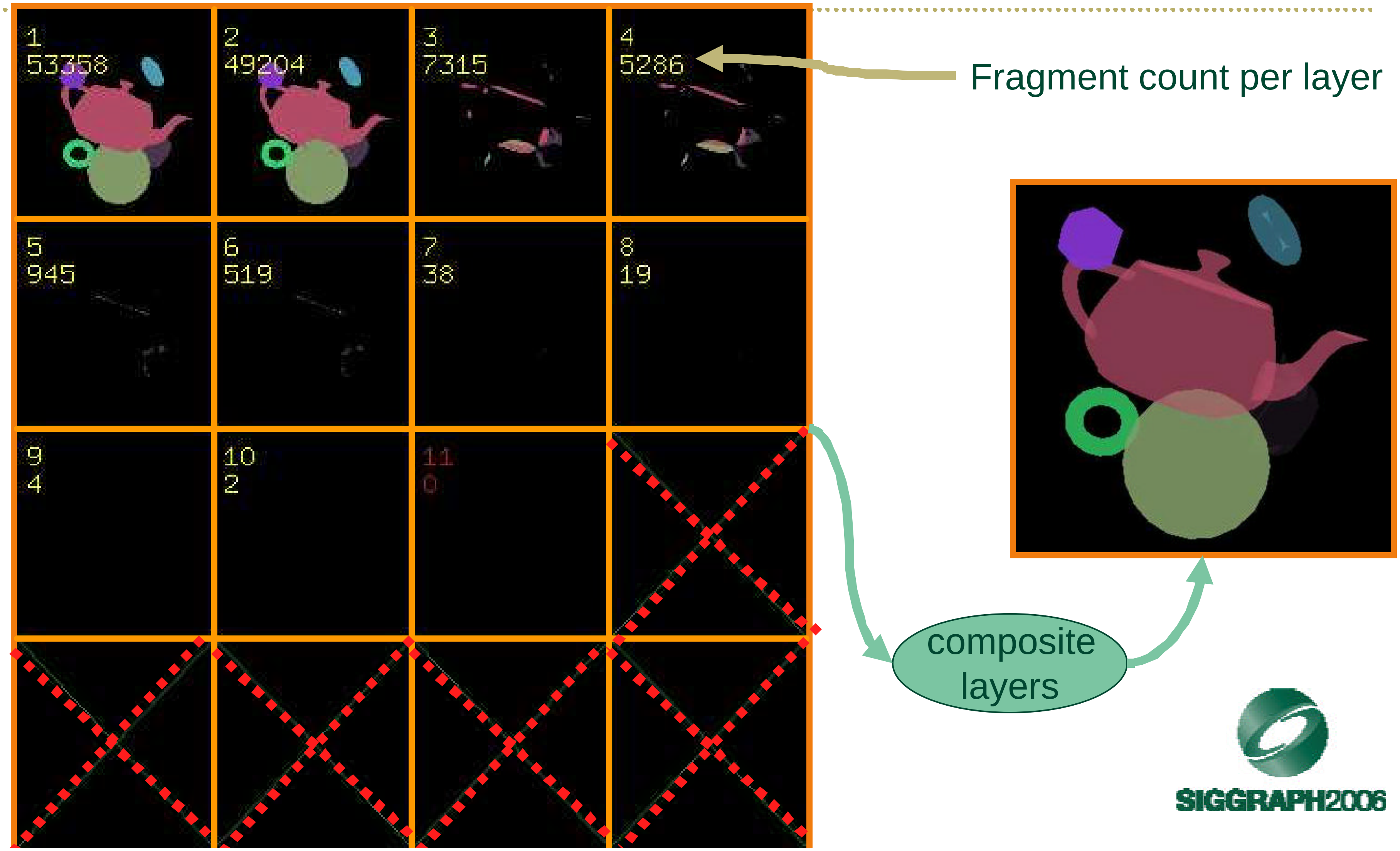
Depth peeling

- Brute force order-independent transparency algorithm [Everitt 2001]
- Approach
  - Render transparent objects repeatedly
    - Each pass peels successive color layer using dual-depth buffers
  - Composite peeled layers in order
- Caveats
  - Typically makes “thin film” assumption
  - No refraction or scattering

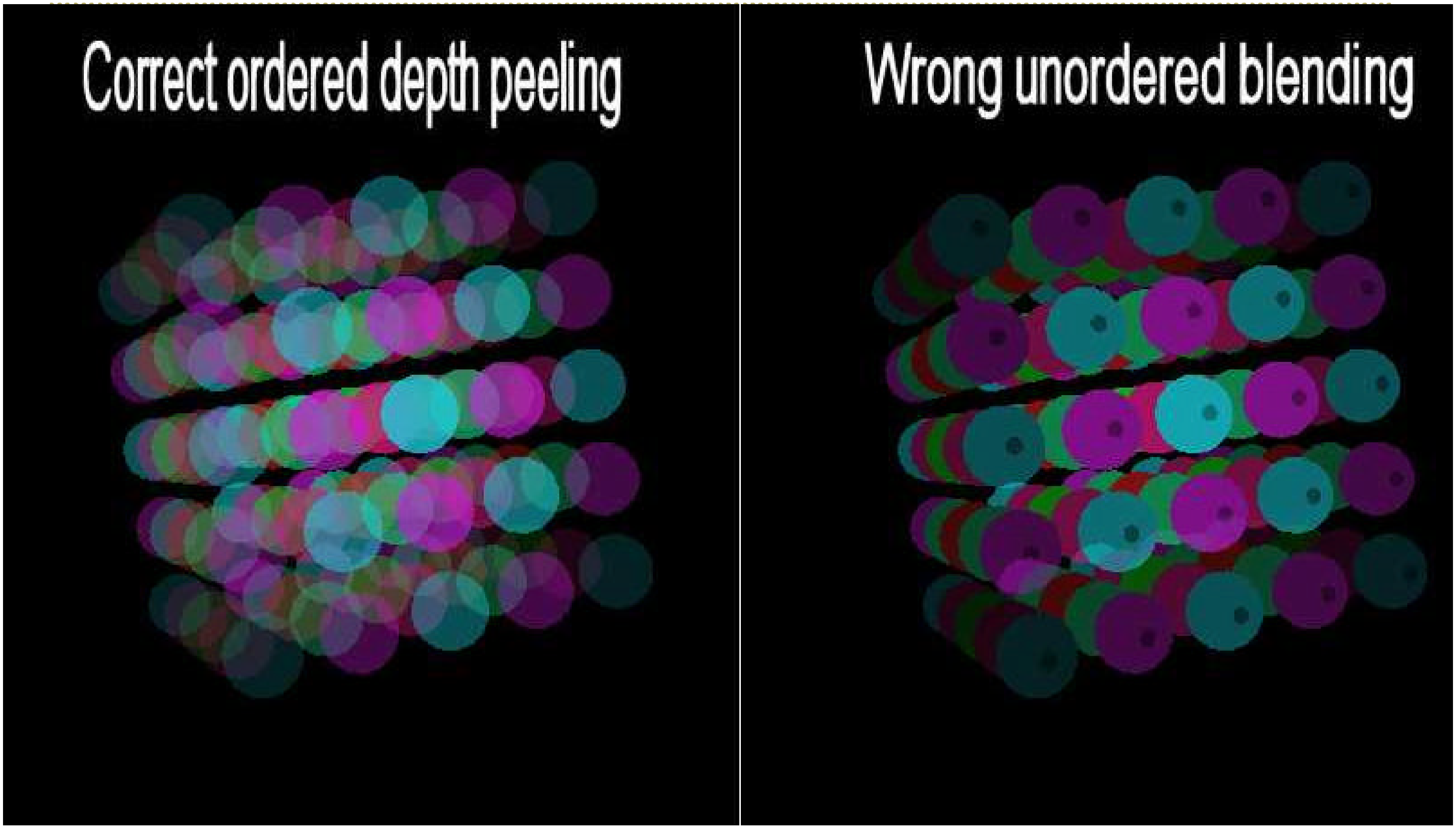
Transparency: The Good vs. the Ugly



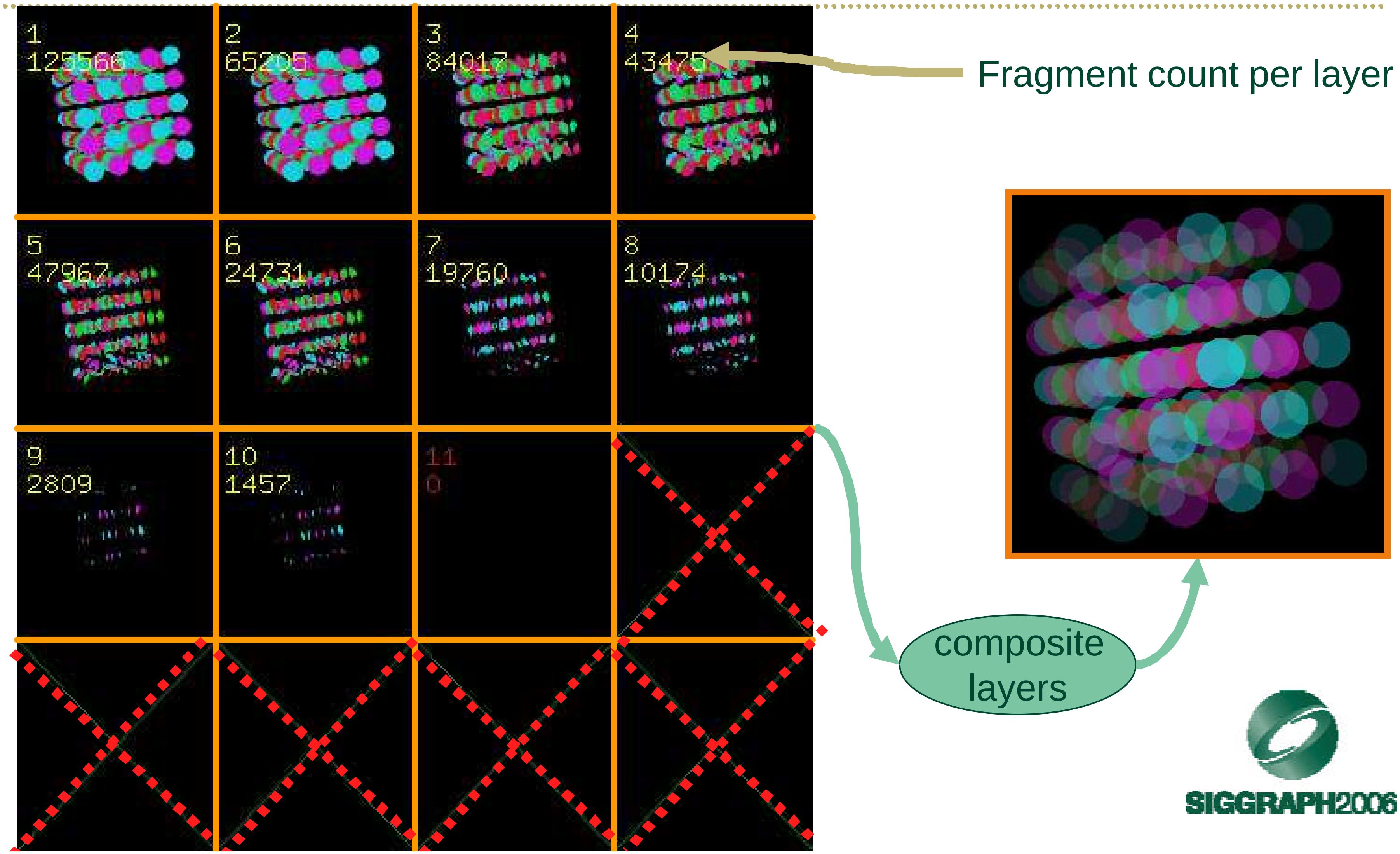
Peeled layer visualization



Another example: The Good vs. the Ugly



Another peeled layer visualization



SIGGRAPH2006

Real-time transparency demo

Depth peeling: How it works

- Conventional depth buffer after rendering
  - Color buffer has color of closest fragment
  - Depth buffer has depth of closest fragment
- Re-use the depth buffer!
  - Make depth buffer into a shadow map
  - Clear a 2nd depth buffer
  - Discard fragments if fragment depth is closer than corresponding pixel's depth in shadow map
  - Save color buffer for compositing
  - Repeat this with current depth buffer to peel another layer
- Prior depth buffer works as "back stop" for next pass
  - Discard fragments closer or as close as last pass for every pixel

SIGGRAPH2006

Optimizations for real-time depth peeling

- Optimizations
  - Render-to-texture to ping-pong between 2 back stop depth buffers (no depth buffer copies)
  - Shadow mapping for 2nd read-only "back stop" depth buffer
  - Asynchronous occlusion queries to determine fragments still being peeled
  - Threshold to stop peeling
  - Smart front-to-back ("under") compositing
- **Result:** 120+ fps depth peeling for peeling and composting up to 14 layers as needed

So is transparency a solved problem?

- Bounding the error
  - Assume a lower bound on opacity of objects
    - …and an upper bound on layers peeled
  - $worstCaseError = (1\text{-}minOpacity)^{maxLayers}$
    - **Example:** 20% min. opacity with 15 peeled layers
    - Remaining potential transparency could be off by just 3.5% if looking through 15 layers of 20% opacity (worst possible case)
  - Typical cases are much, much better than that
    - As occlusion query can provide a count of mis-ordered pixels
- Arguably **could be** for a certain class of transparency
  - Mostly opaque scenes with thin film transparency like windows
  - CAD models made of virtual Jell-O®

SIGGRAPH2006

Conclusions

- NVIDIA GPUs
  - Expect more compute and bandwidth increases >> CPUs
  - DirectX 10 = large functionality upgrade for graphics
- Cg, the only cross-API, multi-platform language for programmable shading
  - Think shaders as content, not GPU programs trapped inside applications
- Depth peeling
  - Harnessing the GPU’s brute force for transparency