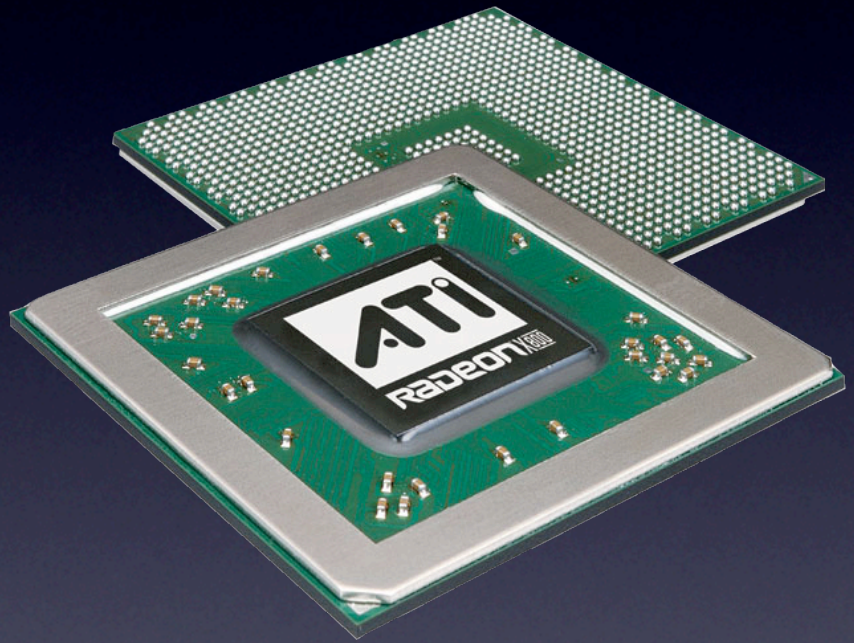
GPGPU



Peter Laurens
1st-year PhD Student, NSC

Presentation Overview

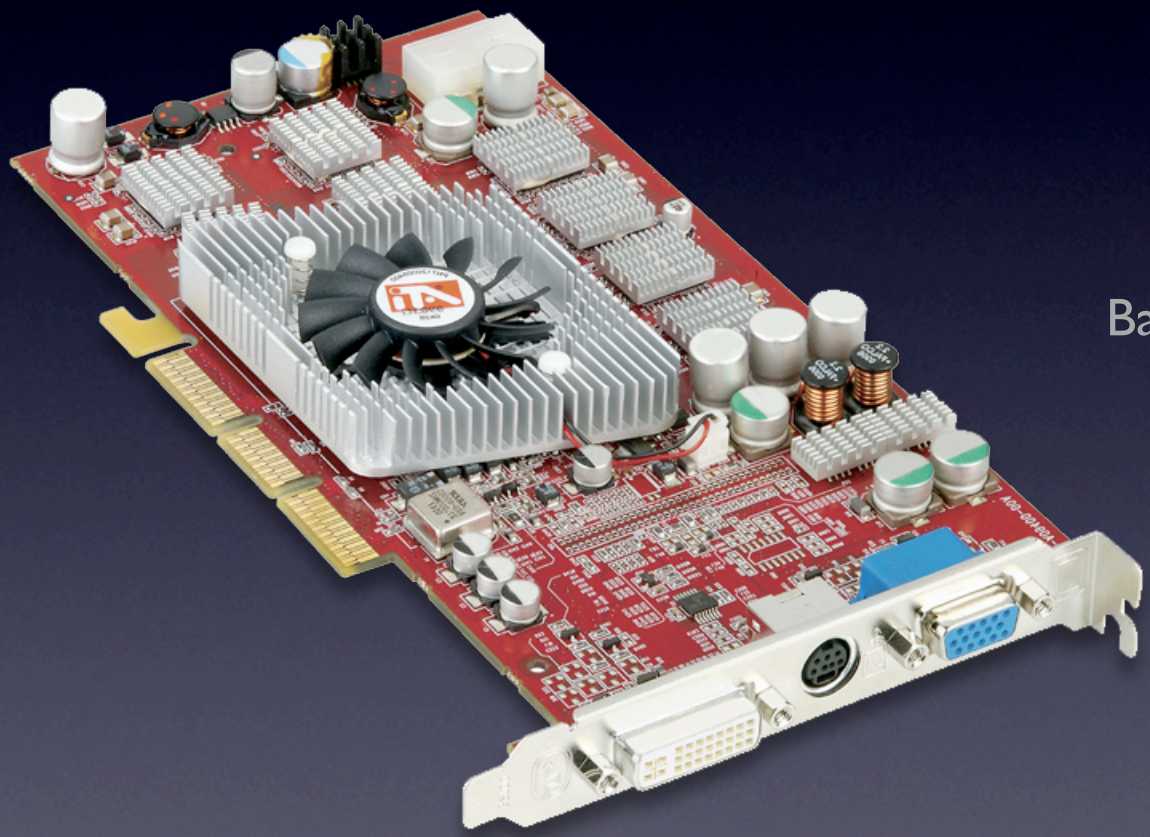
1. What is it?
2. What can it do for me?
3. How can I get it to do that?
4. What's the catch?
5. What's the future?

What is it?

Introducing the GPU

The GPU

A Specialised Hardware Component for Graphical Rendering



Core	250 - 500MHz
Memory	128MB - 512MB
Bandwidth	10 - 40GB/s
Price	~ £150 GBP

“CryENGINE” Crytek A.G.

Fixed Function Pipeline

Pixel and Vertex Shaders

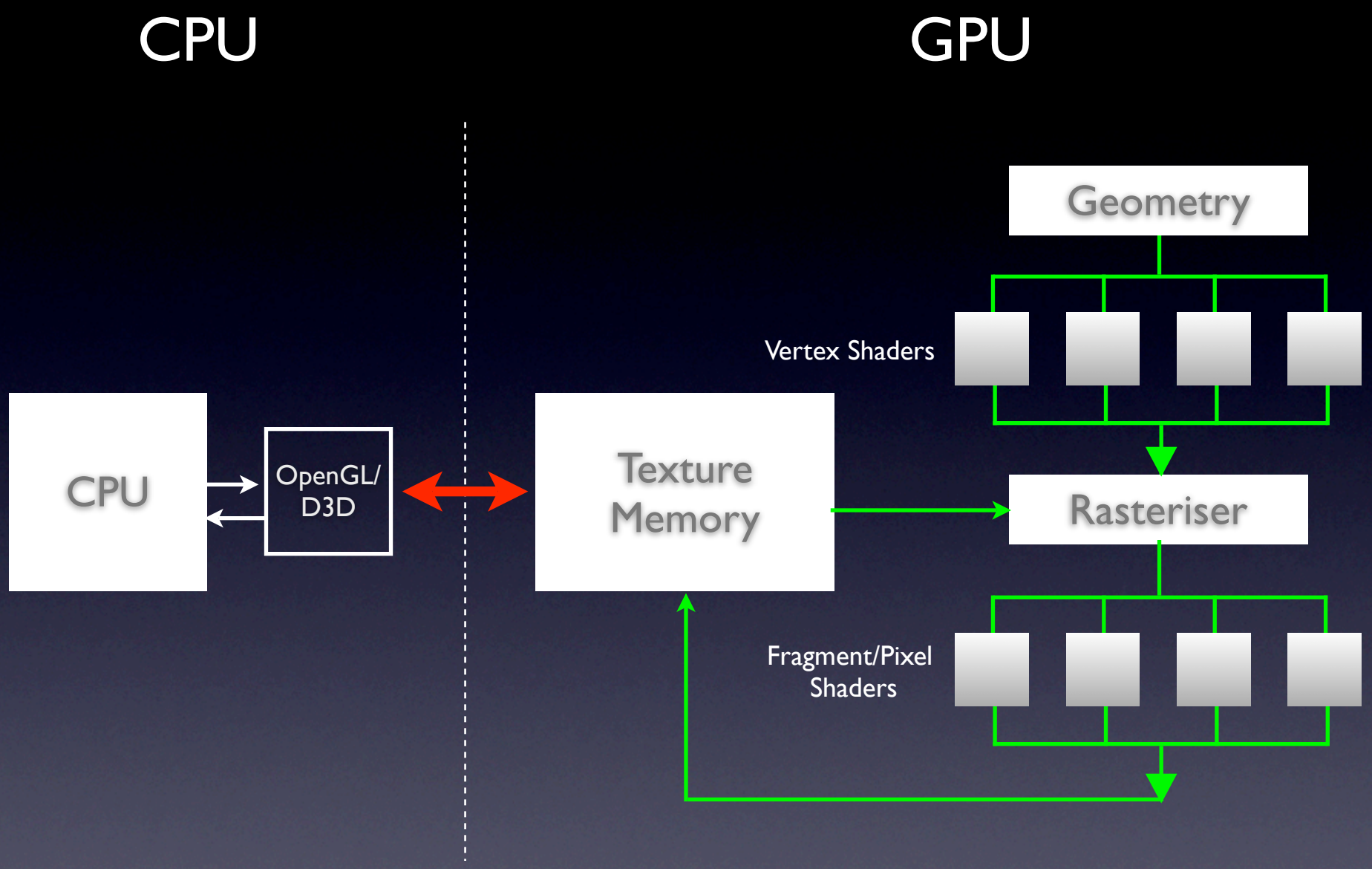
3D Geometry can be passed to the GPU,

The GPU is now programmable, arbitrary code may be run on the GPU which is executed on every pixel of a surface.

rasterised, coloured and textured.

Effects such as mip-mapping, multi-texturing, bump-mapping, etc. are possible, but you only get what you are given.

These shaders are applied in parallel to the pixels.


CPU
GPU
Geometry
Vertex Shaders
CPU
OpenGL/
D3D
Texture
Memory
Rasteriser
Fragment/Pixel
Shaders
Bandwidth is
Expensive
But Computation
is very, very, cheap

Shaders

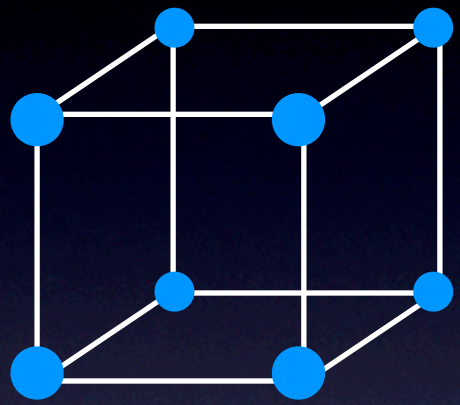
The Core of the Programmable Pipeline

Vertex Processor

Fully programmable

Runs shader programs on world geometry

Write to vertices only (no read)



Fragment Processor

Fully programmable

Takes rasterised pixel 'fragment' data from the vertex stage

Affects the output color

Random read from across the texture data, no RA writes

Direct output to texture

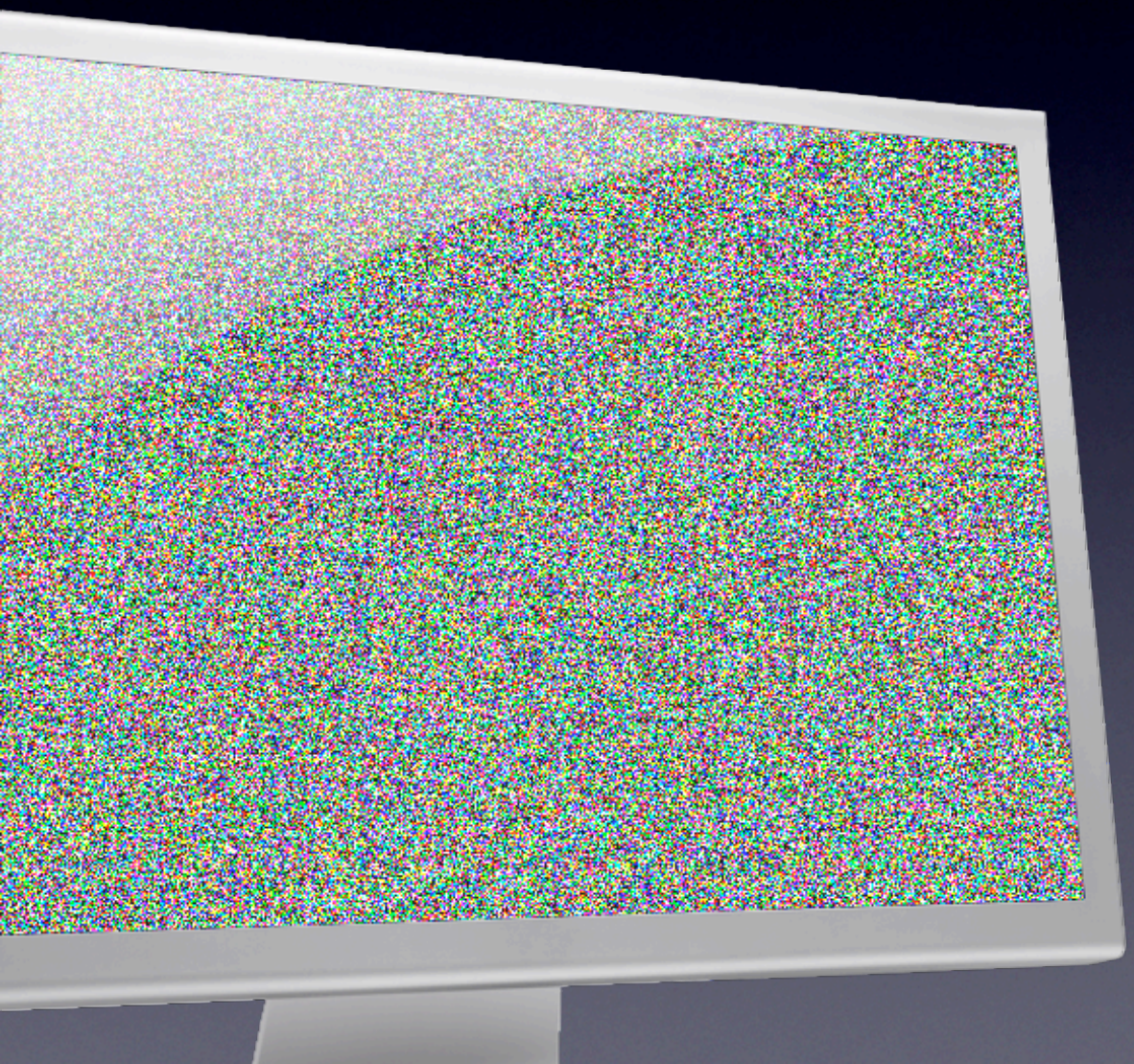
Much more applicable to GPGPU

GPGPU

What can the GPU do for me?

General Purpose GPU

Using the GPU for General Purpose Computing, taking advantage of the massive parallelism available for scientific processing



Instead of rendering geometry, render a flat quad with your data on it, use the pixel shaders to process it.

Pixel shader is applied in parallel to every pixel, independently.

Several pixel shaders in sequence.

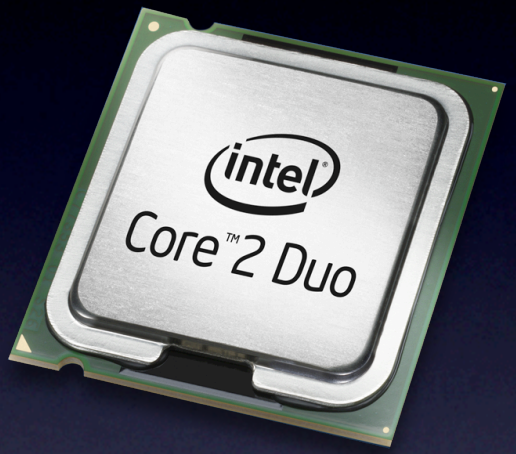
Max 4096x4096 matrix size.

Performance

CPU vs. GPU

Intel Core 2 Duo @ 3.0GHz

Computation	48 GFLOPS (peak)
Memory Bandwidth	21GB/s (peak)
Price	$851 (Apr 16th)

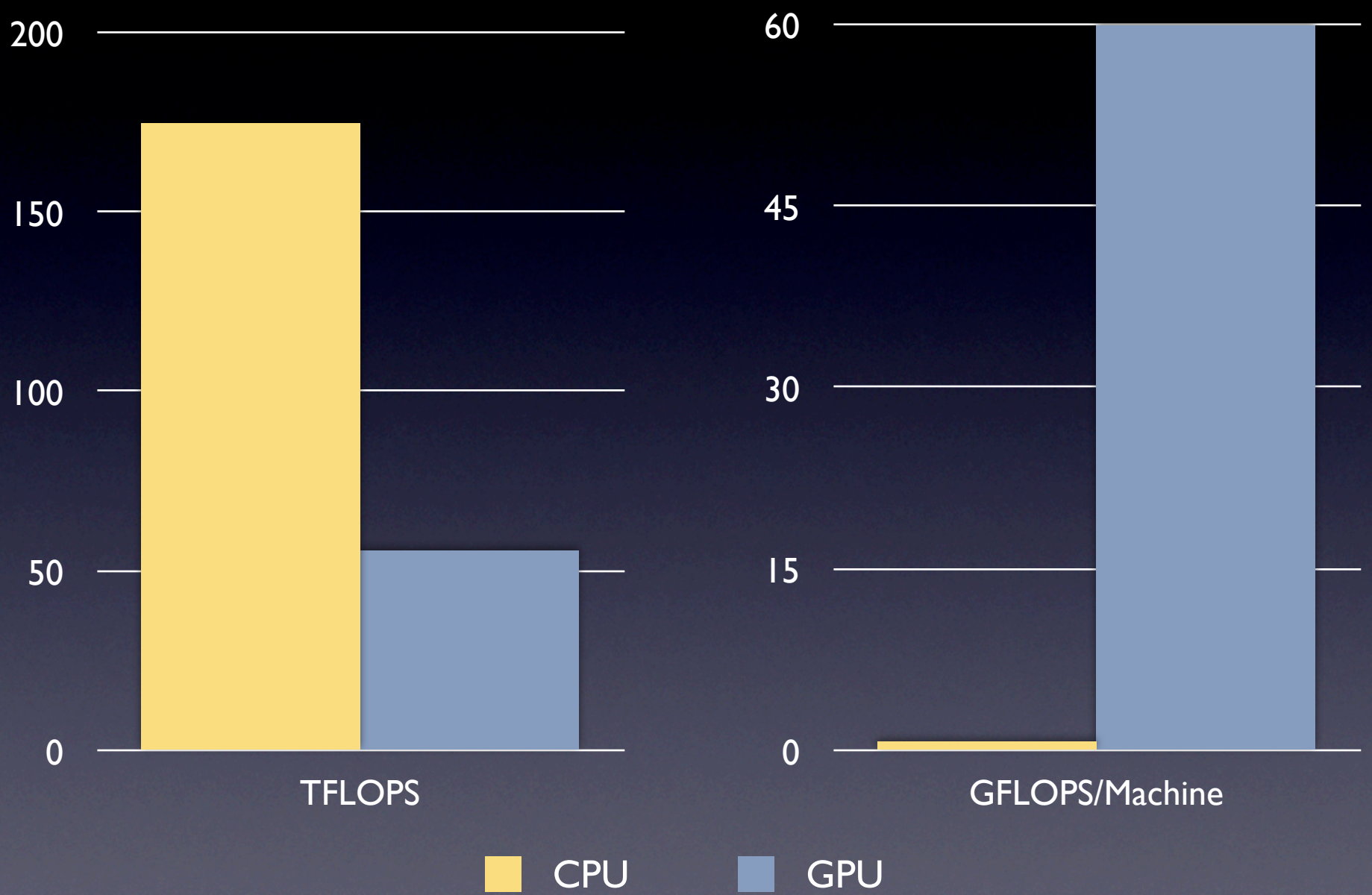


nVidia GeForce 8800GTX

Computation	330 GFLOPS (observed)
Memory Bandwidth	55.2GB/s (observed)
Price	$549 (Apr 16th)

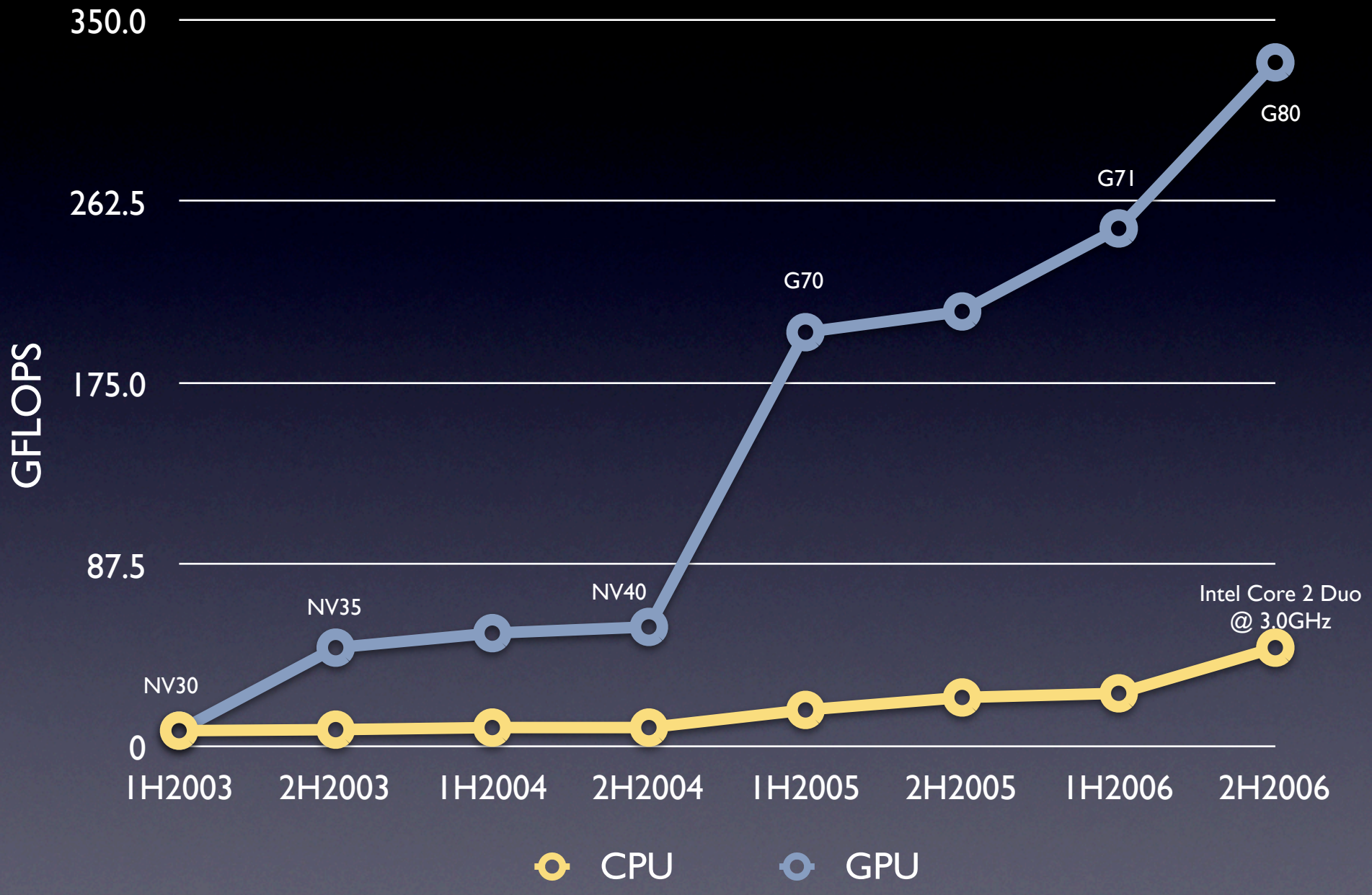
Source: SC06 (GPGPU.org)

"Folding@Home"



Source: Folding@Home, April 2007

Raw Multiply Performance vs. CPU



Source: nVidia Corporation

Suitability

Convolution (neighbourhoods), Arithmetical Intensity, Data Independence

Cellular Autonoma

Neural Networks

Genetic Algorithms

Genetic Programming

(Harding & Banzhaf)

Chemical Simulation

Cryptography and Cryptanalysis

Digital Signal Processing (DSP)

Fast Fourier Transform

Speech Processing

Computer Vision

Linear Algebra

Video Processing (Motion

Compensation, De-interlacing etc.)

Image Effects and Processing

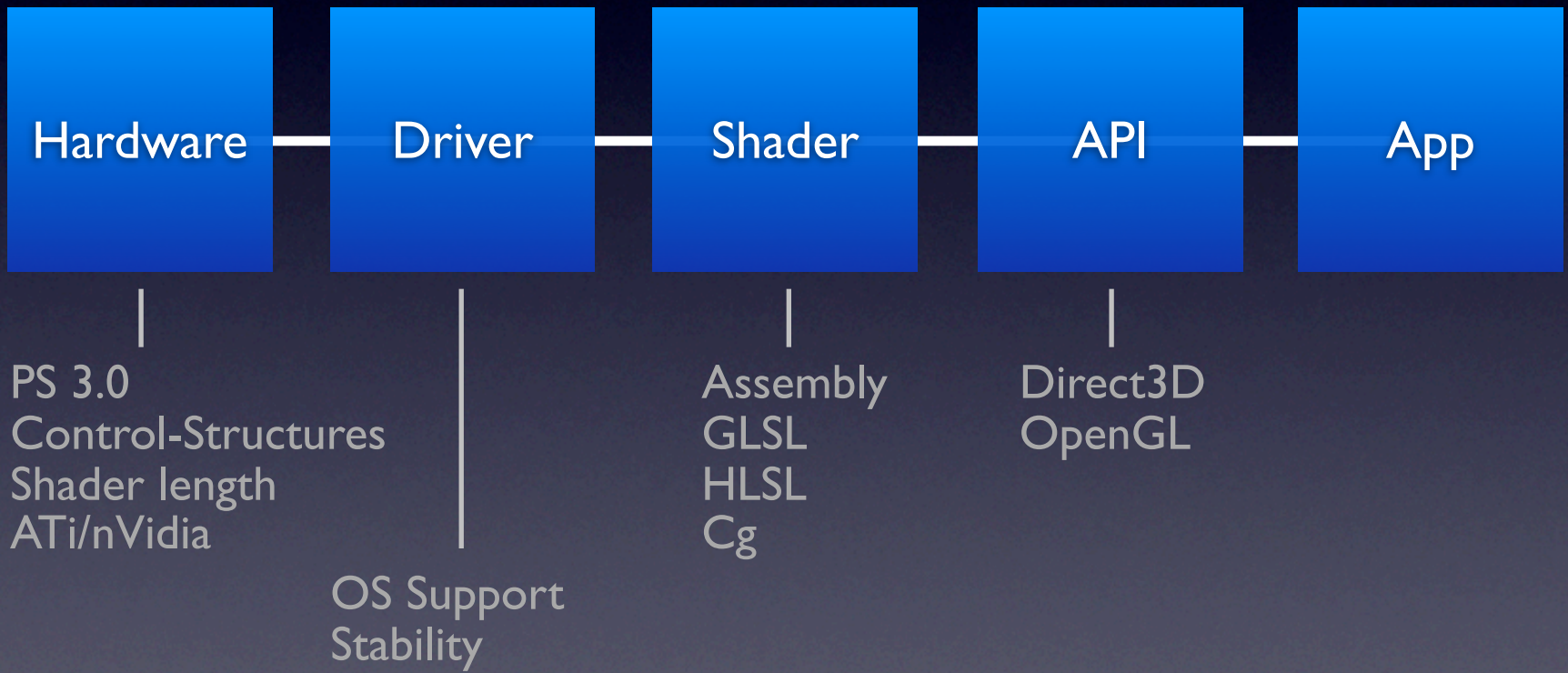
(of course)

How

How can I get it to do that?

Pre-requisites

Setting Up An Environment for GPGPU



Programming Paradigms

These Can be Quite Different from the CPU

Data — CPU Arrays > GPU Textures

Textures are the fundamental data-structure fragment shaders are used on. Therefore, arrays on the CPU become 2D-textures on the GPU.

Code Kernels — CPU Inner Loops > GPU Fragment Programs

On a CPU we would use a loop to iterate over the many elements of an array sequentially. On a GPU, the inner loop of the code becomes the instructions to be applied by the fragment program to the stream of texture data.

Output — Feedback from the GPU

On a CPU, the unified memory model means we can read and write wherever we choose, making getting output easy. On the GPU results must be rendered to a texture, which can then be read off of the GPU, or used in future processing.

Computation — CPU Execution > GPU Drawing

Invocation of computation is straight-forward on the CPU. To run a GPU program, the GPU must 'draw' the result.

More Concepts

Ways in Which a Kernel May Operate on a Stream of Data

Mapping — One-to-One Function

Mapping, also known as Transformation, describes a one-to-one input/output application of a function to the texture array data. E.g., multiplication by two.

Reduction — Many-to-Fewer

Data may be reduced to a smaller stream, for example, averaging. For this example, the GPU architecture is best suited to cascading reductions in several stages, using the output as new input, before arriving at a single element.

Filtering — Non-Uniform Reduction

Removing some items based on certain criteria.

Scatter — Vertex Operation to "Spread" Data

Spreads the data in a certain manner by enlarging the geometry of the underlying plane.

Gather — Random Access Read

Refers to the capability of the Fragment Shader to read from a texture in a random fashion

Search — Parallel Search

The GPU does not speed up the search for an element, but allows a parallel search technique

Source: GPGPU.org, Wikipedia.org/GPGPU

Fragment Shader Code

What Does it Look Like?

```
float4 saxpy (
    float2 coords : TEXCOORD0,
    uniform samplerRECT textureY,
    uniform samplerRECT textureX,
    uniform float alpha ) : COLOR
{
    float4 result;
    float4 y = texRECT(textureY,coords);
    float4 x = texRECT(textureX,coords);
    result = y + alpha*x;
    return result;
}
```

What's the Catch?

Well...

Problems

Why We Aren't Using it Already

Problem must be in the right format for processing on the GPU

Limited memory access (maximum ~2GB)

Slow transfer across PCIe

Texture sizes limited to 4096x4096

Vendor specific adaptations (think HTML3.0) limits adoption

Lack of standardisation of features

Drivers can be flakey, unstable, imprecise

Driver availability for Linux

Only Single-Precision Floating-Point Accuracy

Limited control structures, no recursion

Very poor Multi-GPU support (OS, Drivers, API)

Immature High-Level Shader Languages

Forced to use Graphics-oriented API and terminology

"Loose and Fast" attitude toward driver development

No integers

Software tools are immature, debugging can be a pain (e.g. running shader on hw)

Unusual programming model

Serial-code with lots of complex data interdependencies not suitable

Limits on the length of shaders

Future Prospects

They're Actually Pretty Bright...

The Future of GPGPU

Why We Might be Using it Soon

New intermediary APIs Brook and Sh

nVidia's CUDA: A step further than Brooke and Sh (write in C, special drivers, plus guarantees of future compatibility)

ATi's CTM: Similar to CUDA (better access to memory etc.)

Maturity will inevitably bring stability and homogeneity to drivers/features

AMD + ATi Stream Processing and CPU Integration (alleviate bus latency and memory constraint)

Direct3D, OpenGL, and the cards themselves continue to support more features, double-precision floating point is coming.

Places to Go

Learning More About GPGPU

www.gpgpu.org

www.nvidia.com

www.ati.com

Stream Processing (USA)