

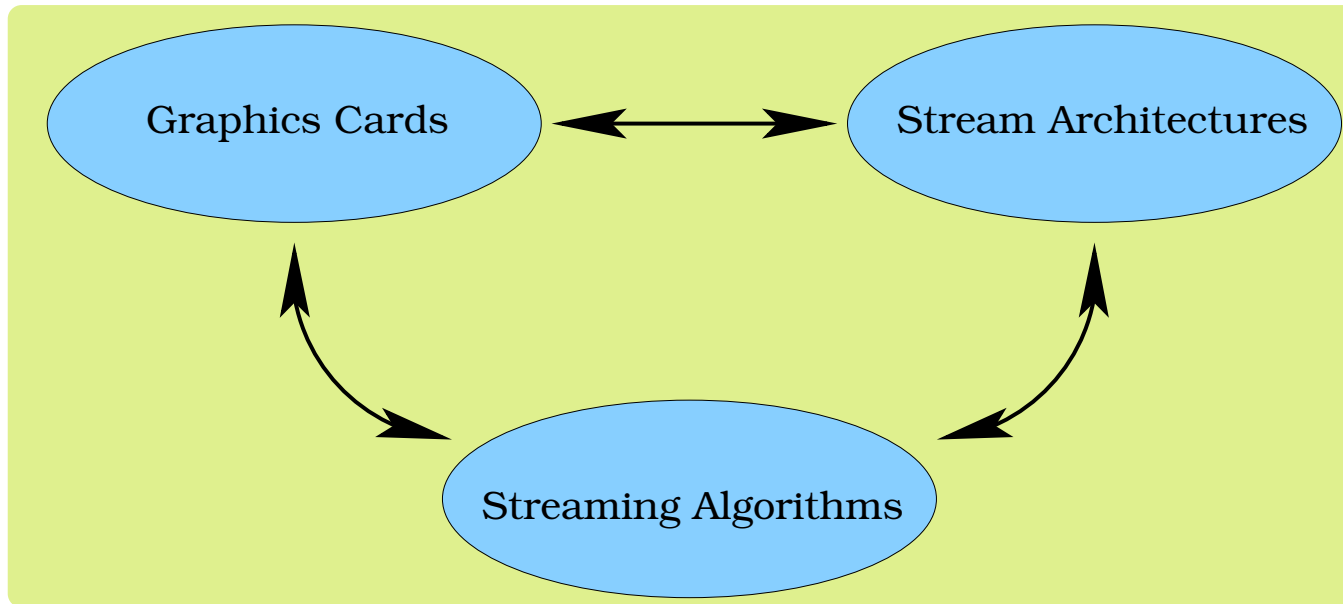
The Graphics Card As A Streaming Computer

Suresh Venkatasubramanian

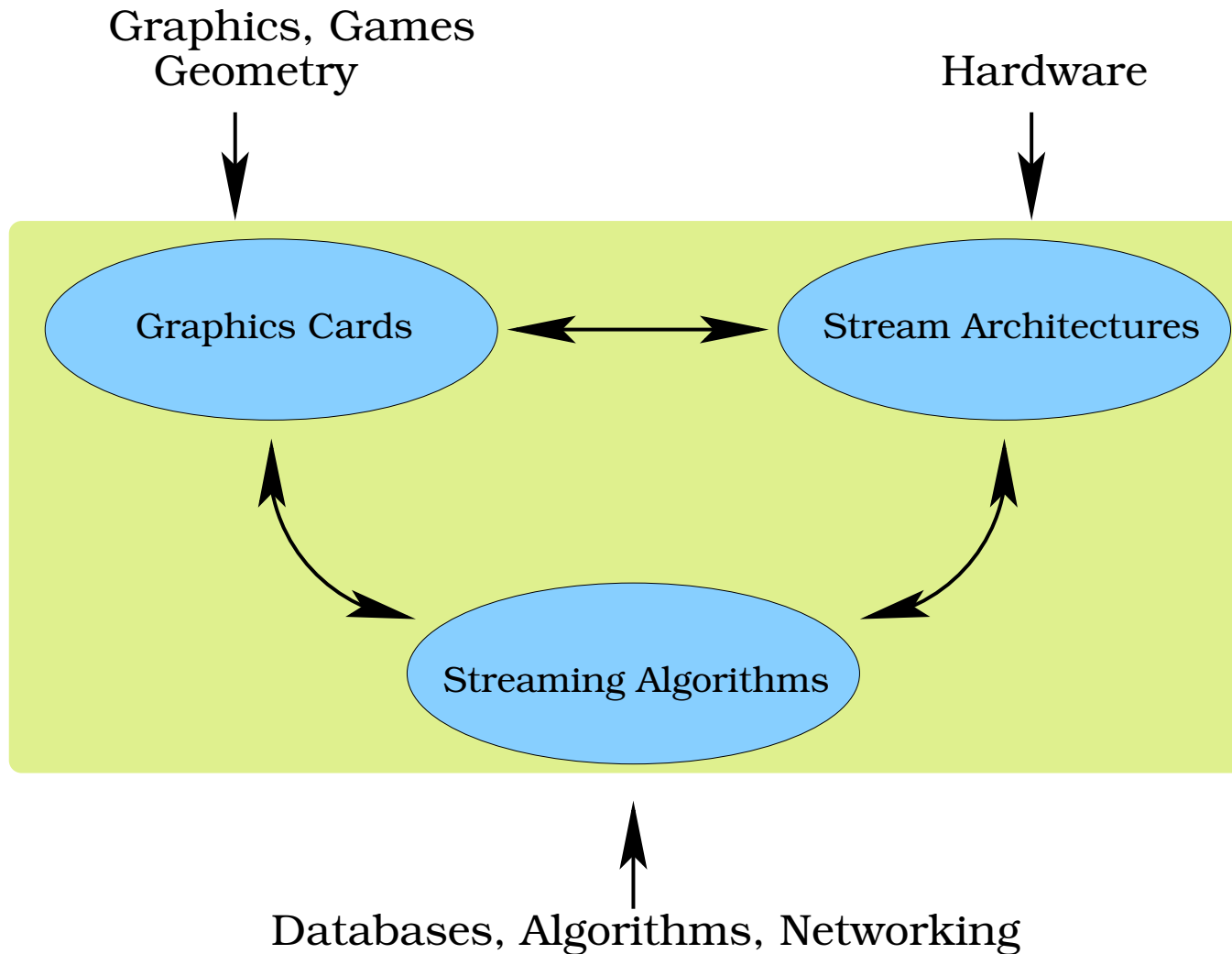
AT&T Labs–Research



An Inter-Disciplinary Endeavour



An Inter-Disciplinary Endeavour



Graphics And The GPU

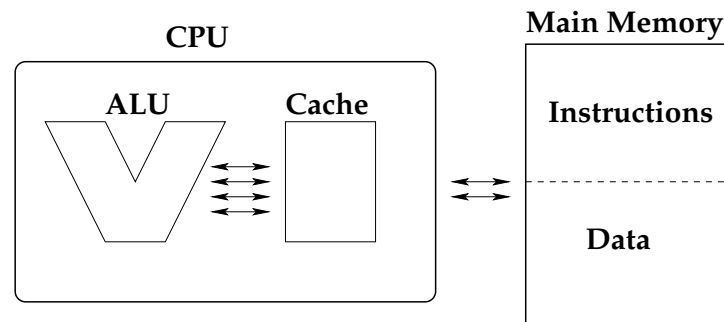
Through the 80s, advances in VLSI design fueled the development of specialized hardware for rendering graphics.

Initially, graphics cards were good for raw rendering; most of the specialized surface/texture calculations were done in software. As graphics rendering grew more sophisticated, more of this functionality got pushed down to the hardware.

The need to write complex shaders led to the idea of *programmable graphics cards*. Originally these were implemented via multi-pass rendering (“streaming”). The next generation of graphics cards (circa 2002) allowed an extensive list of programmable features.

Streaming Architectures

The CPU can process data far faster than it can be brought in via the memory bus. This discrepancy is only getting worse.



Caches and prefetching strategies have greatly improved CPU performance. However, a large portion of modern CPU area is devoted to cache circuitry (over 60% by some estimates).

Systemic arrays, SIMD, vector/superscalar architectures are some of the architectures designed to address the von Neumann bottleneck.

Massive Data And Streaming Algorithms

A primary motivation for studying streaming is *transient* data. Examples include data coming over a network, router data, sensor data, visual data *etc.*

Another motivation, is *large, disk-resident* data, which is expensive to access. Here, performance considerations as well as data considerations motivate streaming.

Streaming algorithms are only allowed **one** pass over the data set. With limited memory, such algorithms are limited, and much research has gone into examining the power and limits of such streaming computations.

A New Paradigm: Streaming Pipelined Architecture

- Objects stream in one-by-one.
- Once processed, an object is passed to the next phase and does not return.
- There is limited local memory: each objects essentially *carries* its own state with it.
- Pipelining: *Each object is processed in the same way.*
- **Spatial Parallelism: Each pixel processes a different stream.**

Significant advantages accrue from exploiting *data parallelism* and the *pipeline* model.

The Graphics Card

The graphics card in a desktop or video game console is the most striking example of the power of a stream architecture.

Its primary purpose is the rendering of complex 3D scenes. As rendering becomes more and more complex, more computation is performed in the graphics card itself.

Graphics cards are much faster than CPUs for certain computations. Furthermore, the growth rate for GPU performance is superior to that of CPUs.

Graphics cards provide restricted streaming functionality and a virtual SIMD architecture as well. For all practical purposes, *each pixel on the screen* can be viewed as an SIMD unit receiving a stream of data.

A Roadmap For The Talk

1. Inside the GPU



2. Some Examples



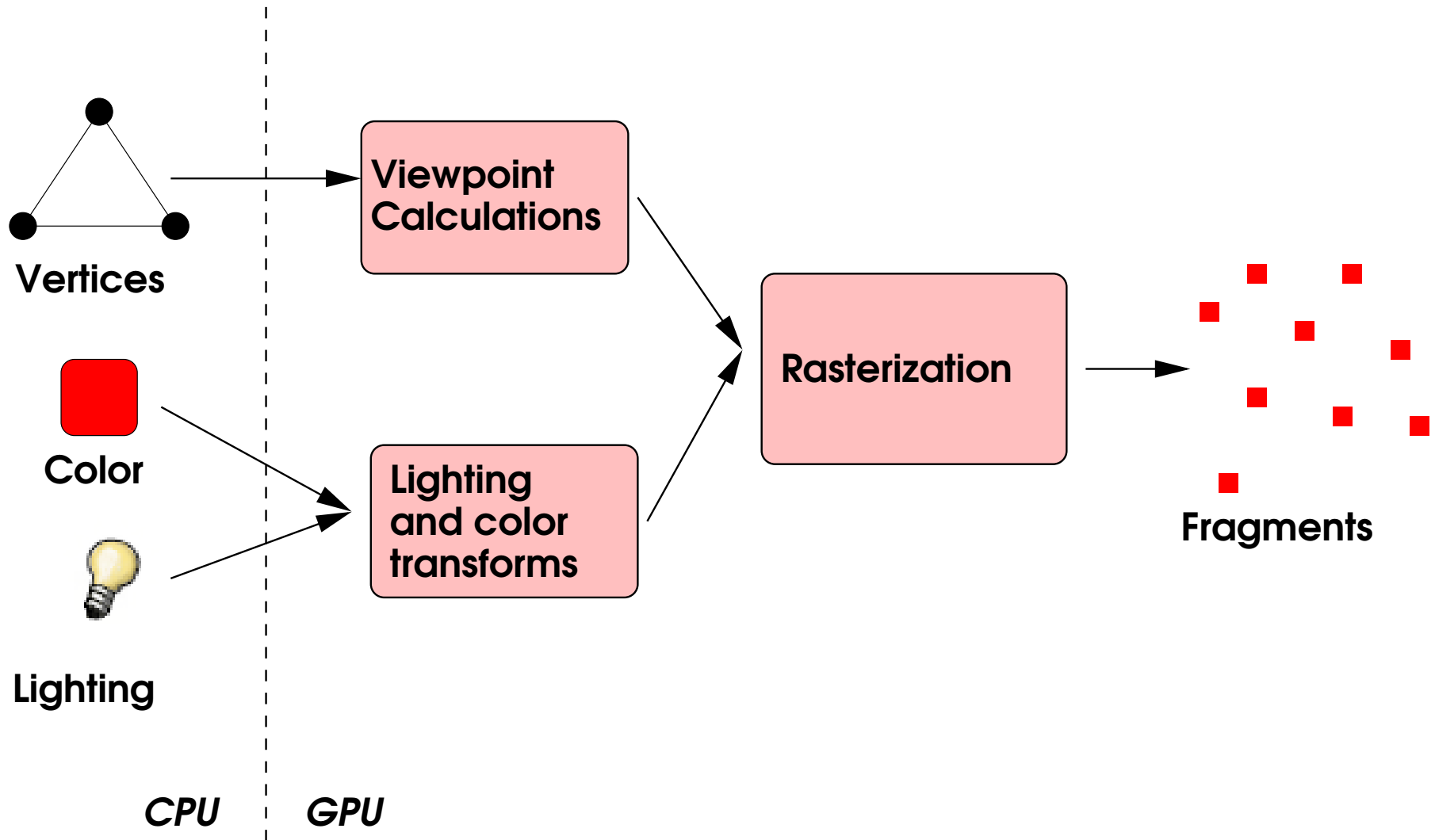
3. How To Write Streaming Programs



4. Where Are We Headed ?

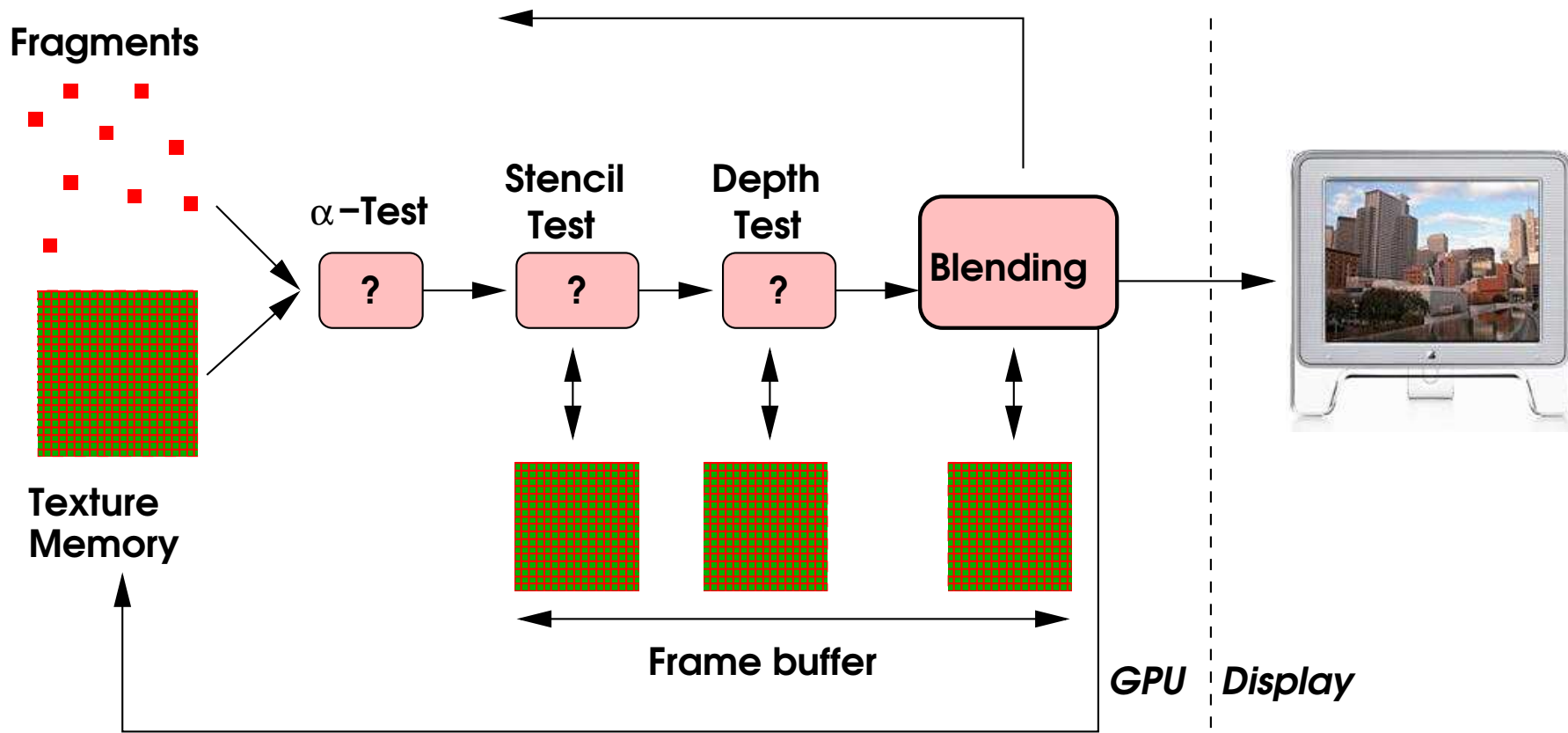
THE GRAPHICS PIPELINE: A CLOSER LOOK

Processing Objects in the GPU: Step 1



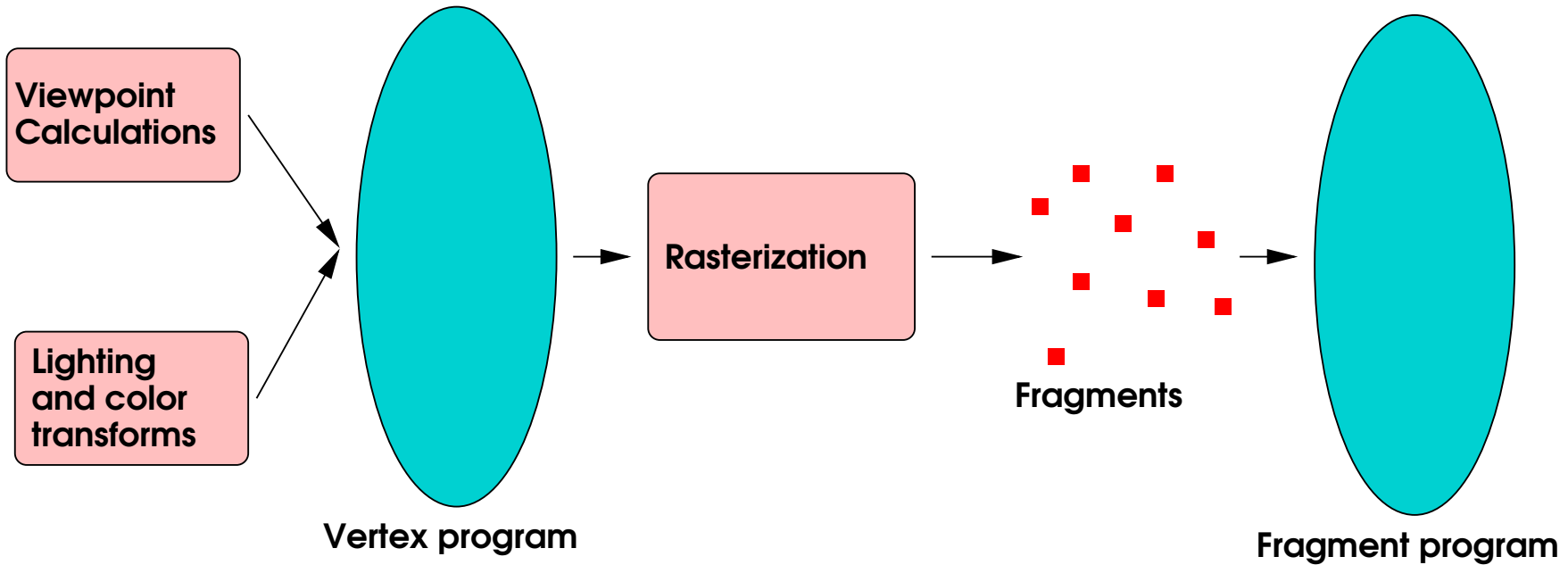
The Fixed-Function Pipeline

Processing fragments in the GPU: Step 2



The Fixed-Function Pipeline

Programable Pipelines



- Vertex program executes on each vertex.
- Fragment program executes on each fragment.

Capabilities

- Large instruction set: general purpose arithmetic and scientific calculations on scalars and vectors
- Programs can be large: *hundreds* of instructions can be executed in a single pass.
- Texture buffers allow more general purpose memory access.
- Some limited pointer indirection for array lookups.
- Pipeline and stream discipline maintained: program runs on each element of the stream.
- No looping in fragment programs; some looping permitted in vertex programs.

Aspects Of The Graphics Card

Storage: Graphics cards have 128-256 MB memory: this is increasing.

Speed: A card can process over 4 billion pixels, or 50 million triangles, per second.

Precision: Current cards perform operations on 4-dimensional vectors, each component being 32 bits.

Cost Model: Reading data *back* from the card is an expensive operation. Thus, the *pass* is the basic unit of cost.

Minimizing the number of passes in a multi-pass computation is the key algorithmic question.

EXAMPLES

1. Data Analysis [KMV02]

Visualization \Leftrightarrow Analysis

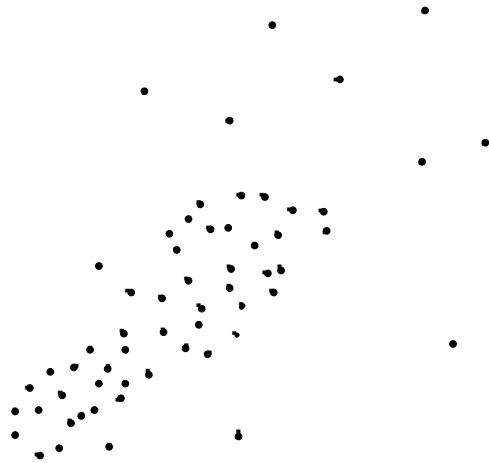
In exploratory data analysis, the right visualization aids immensely in the analysis of large data. For such applications, graphics cards provide a way of performing *interactive* data analysis.

1. Data Analysis [KMV02]

Visualization \Leftrightarrow Analysis

In exploratory data analysis, the right visualization aids immensely in the analysis of large data. For such applications, graphics cards provide a way of performing *interactive* data analysis.

Tukey Depth Contours



1. Data Analysis [KMV02]

Visualization \Leftrightarrow Analysis

In exploratory data analysis, the right visualization aids immensely in the analysis of large data. For such applications, graphics cards provide a way of performing *interactive* data analysis.

Tukey Depth Contours



Depth Contour Demonstration

2. *Geometry And Robotics [AKMV03]*

Many optimization problems in shape modelling and analysis can be viewed as envelope calculations over a family of functions.

- Diameter, bounding box, width of a point set.
- Computing enclosing balls/annuli of objects.
- Penetration depth of two polytopes (used for collision detection).
- Medial axis and Voronoi region calculations (for path planning)

Graphics cards can be used to compute (provably) approximate robust solutions for these problems.

Bounding Box Demonstration

3. Stream Selection [GKMV03]

We want to compute the k^{th} -highest element of a sequence.

- Depth ordering in scenes.
- Natural streaming primitive (selection and sorting).
- Relates to various geometric optimization problems.

Depth test provides the **one-sided test** “Is `fragment.depth < d`?”

Lemma. *Computing k^{th} highest element of a sequence requires k passes with a one-sided test.*

Suppose we had a **two-sided test** “Is `a < fragment.depth < b`?”

Lemma. *With a two-sided depth test, k^{th} highest element can be computed in $\log n$ passes (randomized)*

This is a simple demonstration of how to prove limits on what streaming hardware can do.

Other Application Areas

– **Rendering;**

Many of the complex rendering calculations originally performed on the CPU are now being performed on the GPU.

– **Scientific Computing;**

Graphics cards have been used to aid in simulation of PDEs, physical modelling, conjugate gradient methods, ...

– **Computer Vision:**

One of the first uses of stream architectures was in recovering stereo scenes from two dimensional images. Graphics cards have also been used to compute FFTs on images.

www.gpgpu.org has many other examples

PROGRAMMING THE GPU

Traditional Method: Setting Flags In OpenGL

```
glEnable( GL_BLEND ) ;  
glBlendEquationEXT ( GL_MIN_EXT ) ;  
glBlendFunc( GL_ONE, GL_ONE ) ;
```

<draw stuff>

```
glClearColor(1.0, 1.0, 1.0, 0.0) ;  
glClearDepth( 1.0 ) ;  
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT |  
        GL_STENCIL_BUFFER_BIT ) ;
```

```
glEnable(GL_DEPTH_TEST) ;  
glDisable(GL_STENCIL_TEST) ;  
glDepthFunc( GL_LESS ) ;
```

<draw more stuff>

Fragment Programs: Back To Assembly

- Fragment/vertex program specified as a sequence of assembly instructions

```
# This is a comment
MUL          tmp, fragment.texcoord[0], size.x;
FLR          intg, tmp;
FRC          frac, tmp;
SUB          frac_1, frac, 1.0;
MAD          tmp, intg, size.y, -size.z;
TEX          g0, tmp, texture[0], 2D;
```

- Loaded into OpenGL program via gl call:

```
glProgramString(.... , fragmentProgramString );
```

Caveat: Code is not typically portable because of resource constraints.

Cg: C-like Programming

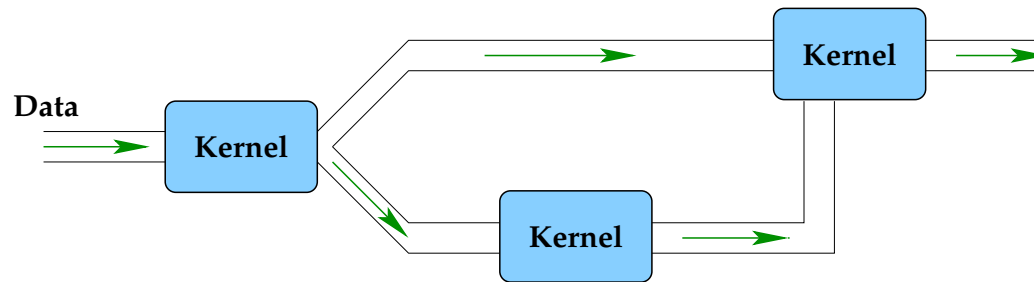
Cg is a language developed by NVIDIA for programming the GPU.

```
float4 main(in float2 texcoords : TEXCOORD0,  
            in float2 wpos : WPOS,  
            uniform samplerRECT pBuffer,  
            uniform sampler2D nvlogo) : COLOR  
{  
    float4 currentColor = texRECT(pBuffer, wpos);  
    float4 logo = tex2D(nvlogo, texcoords);  
    return currentColor + (logo * 0.0003);  
}
```

Cg programs are compiled using the Cg compiler. This allows a degree of platform independence.

The Future: Stream Programming Languages

New programming constructs have emerged that use a *stream* as the basic computational unit. Each stream is processed by a specialized computational unit called a kernel.



- BROOK: C API with streams.

Relates to more general approaches to programming with streams (Hancock, *etc.*).

CONCLUSIONS

The Buzz

From the New York Times, May 26th, 2003:

As perhaps the clearest evidence yet of the computing power of sophisticated but inexpensive video-game consoles, the National Center for Supercomputing Applications at the University of Illinois at Urbana-Champaign has assembled a supercomputer from an army of Sony PlayStations.

From IEEE Computer, Oct 2003:

The graphical processing unit is visually and visibly changing the course of general purpose computing

Michael Macedonia, "The GPU enters computing's mainstream"

Where Are We Headed ?

Stream architectures are increasingly an alternative model of computation for high-performance computing and the graphics card provides a readily accessible, easy-to-use platform for working with streams, and has demonstrated its value in a variety of applications.

- Can we extend standard theoretical models of streaming to this new, somewhat restricted notion ? What are the implications for multi-pass stream algorithms ?
- What are the limitations of stream architectures ? What other problems can be addressed using this framework ?
- A host of questions in programming language design, compilers, operating systems and hardware design...

Acknowledgements

Many thanks to Pankaj Agarwal, K. Aingaran, Nat Duca, Kathleen Fisher, Sudipto Guha, Shankar Krishnan, Nabil Mustafa and S. Muthukrishnan