

Streaming Algorithms In Graphics Hardware

Suresh Venkatasubramanian

AT&T Labs–Research

Two Converging Trends In Computation...

– The accelerated development of graphics accelerator cards (GPUs)

- Current graphics accelerators are cheap and ubiquitous.
- They are developing faster than CPUs (roughly 1.7 times faster per year)

– The increasing need for *streaming computations*

- Original motivation from dealing with large data sets
- Also interesting from perspective of multimedia computations, image processing, visualization, and other areas.

Two Converging Trends In Computation...

– The accelerated development of graphics accelerator cards (GPUs)

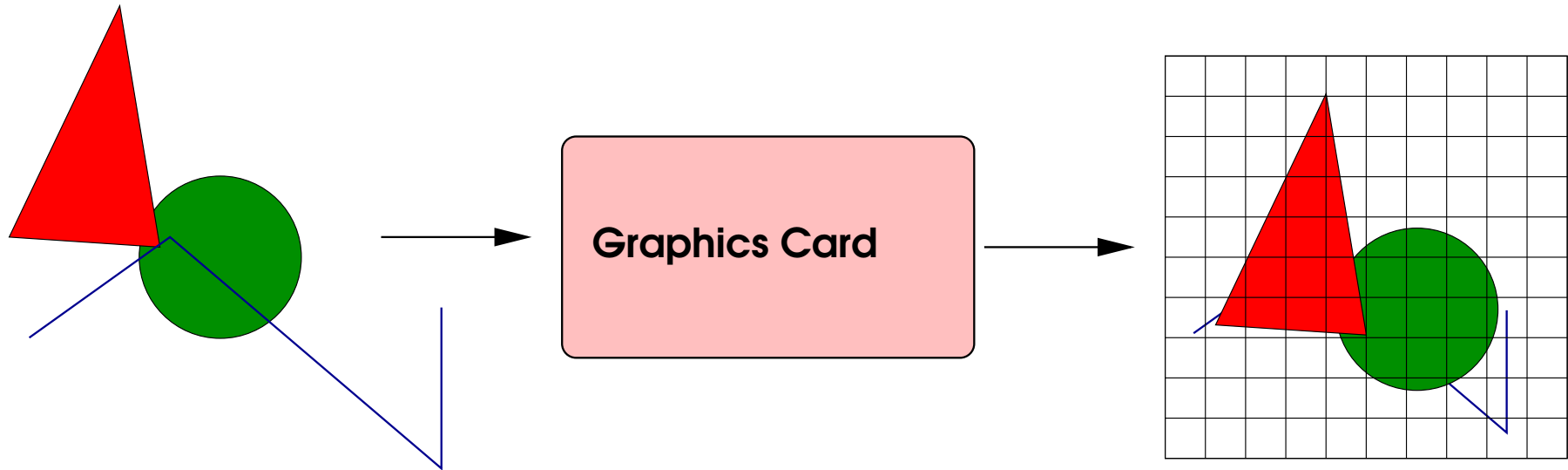
- Current graphics accelerators are cheap and ubiquitous.
- They are developing faster than CPUs (roughly 1.7 times faster per year)

– *The increasing need for streaming computations*

- Original motivation from dealing with large data sets
- Also interesting from perspective of multimedia computations, image processing, visualization, and other areas.

Graphics Cards Can Compute !

A graphics card takes a stream of objects (points, lines, triangles), and renders them on a screen.



Each pixel in the screen can be viewed as a small processing unit.

glBlend	$a = a \oplus b$
z-test	$a = \min(a, b)$

Large Set Of Diverse Applications

- Occlusion Culling in scenes
- Shading on objects
- View dependent Simplification of Shapes
- Geometric Optimization
- Motion Planning and Collision Detection
- Image processing (wavelet analysis)
- Physical Simulations
- Scientific Computations (matrix multiplication)
- Data analysis (especially spatial data)

THE GRAPHICS PIPELINE: A CLOSER LOOK

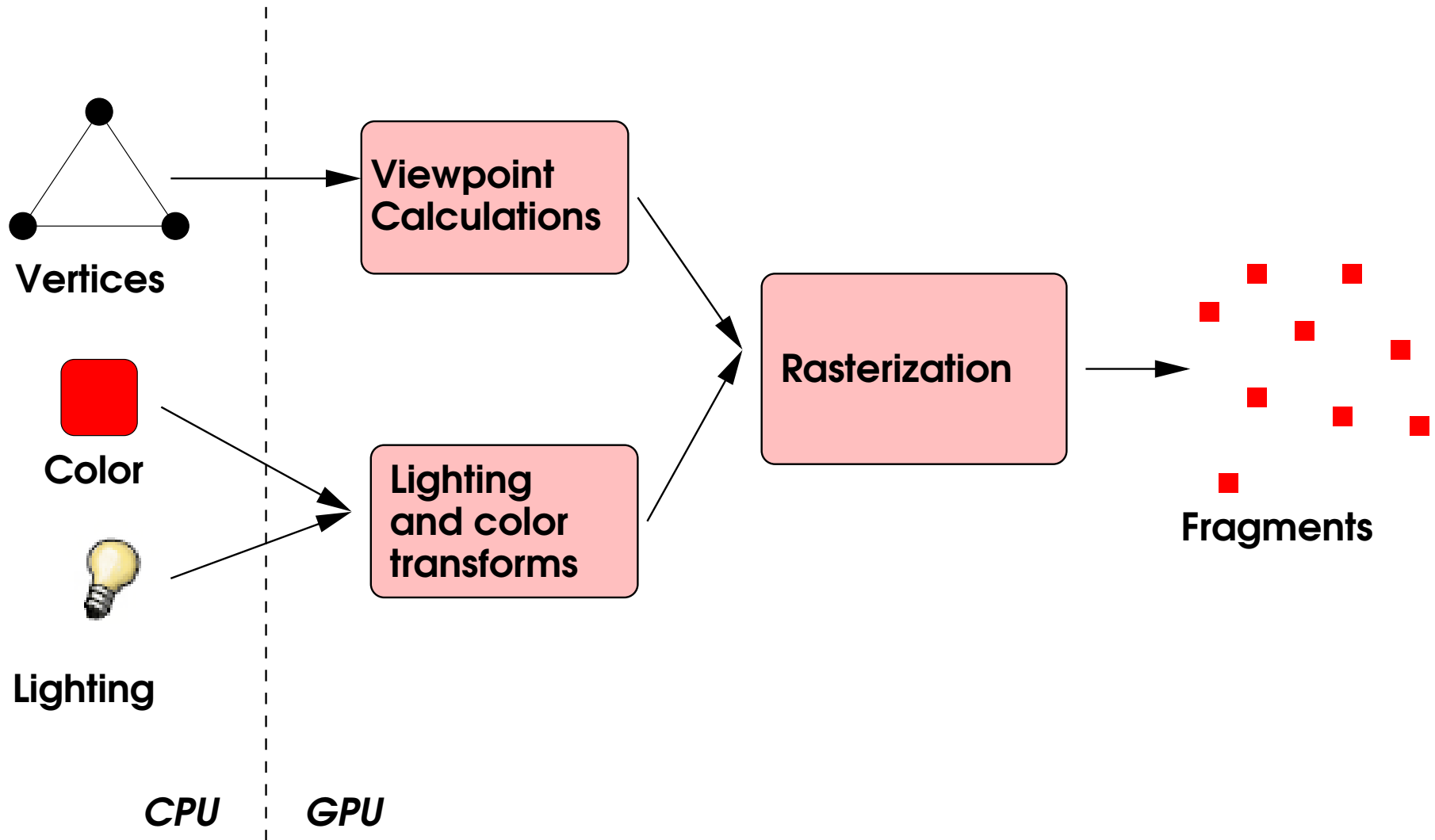
Suresh Writes A Program

```
#include <gl.h>
...
glLight(..) // Set lighting
glOrtho(..) // Set viewpoint

// Now draw objects
glColor(1,0,0);
glBegin(GL_TRIANGLES)
glVertex(x1,y1,z1)
...
glEnd()

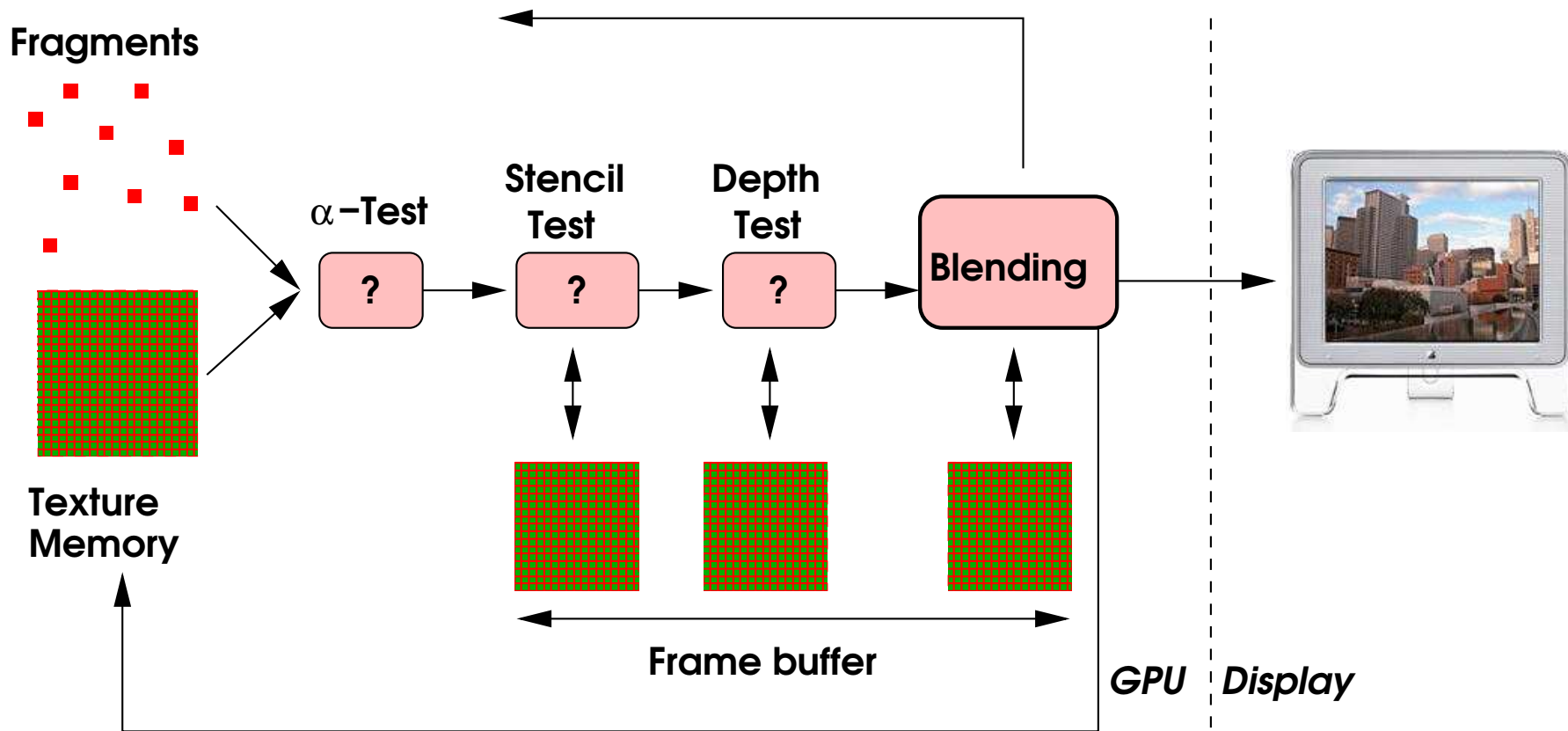
gcc triangle.cc -lGL
```

Processing Objects in the GPU: Step 1



The Fixed-Function Pipeline

Processing fragments in the GPU: Step 2

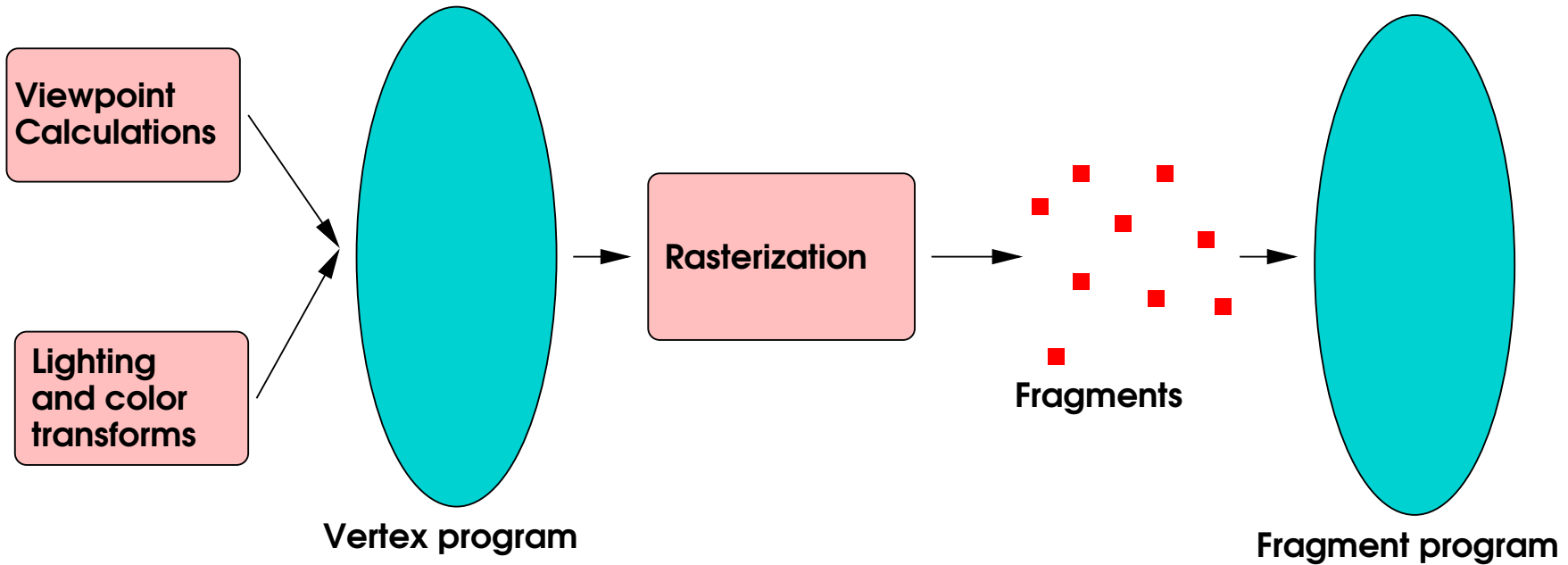


The Fixed-Function Pipeline

So where's the computation ?

- Stencil test
if (buffer.stencil = K) continue
else drop fragment.
 - Depth test
if (frag.depth < buffer.depth) continue
else drop fragment.
 - Blending operations
buffer.color = buffer.color *op* fragment.color
- General arithmetic and boolean function for blending.
 - General comparison functions.
 - Convolution and histogramming operators.

Programmable Pipelines



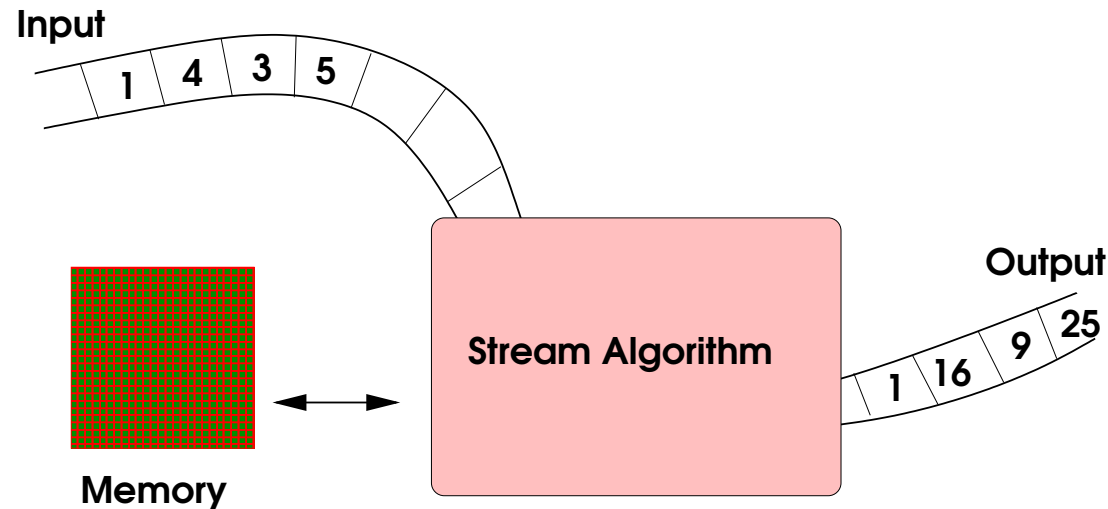
- Vertex program executes on each vertex.
- Fragment program executes on each fragment.

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.
- No looping in fragment programs; some looping permitted in vertex programs.

Haven't We Seen This Before?

Standard streaming model of computation

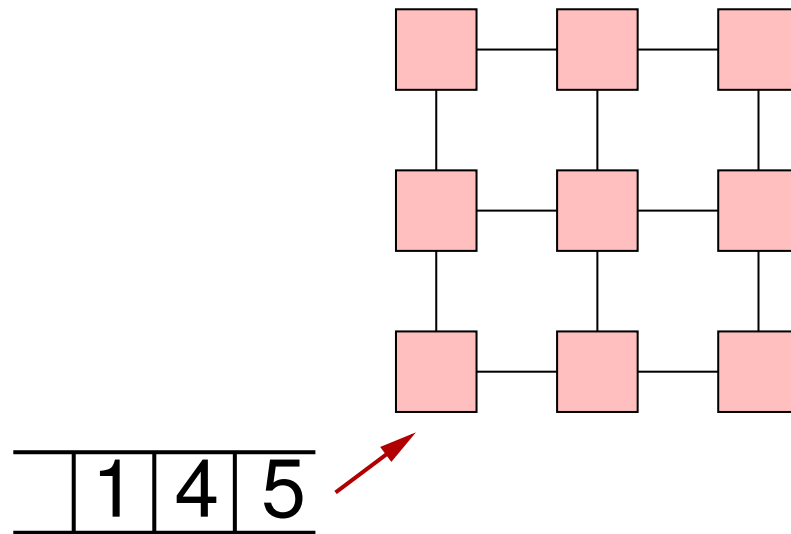


What's different ?

- Limited memory (really a constant vs $\text{polylog } n$).
- Pipelining restriction: all items have to be treated the same way.
- Multi-pass potential: standard streaming models assume exactly one pass (with a few exceptions).

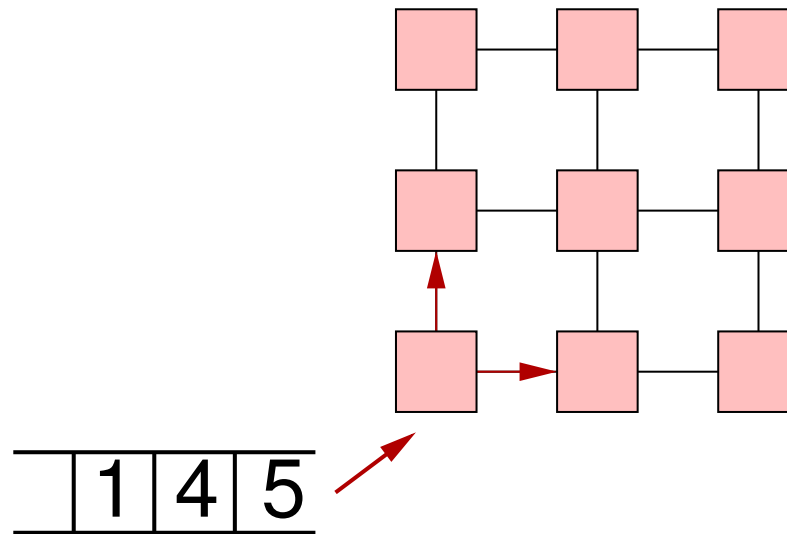
Maybe We Have Seen This Before

Systolic Arrays [Kung+Leiserson 1978]



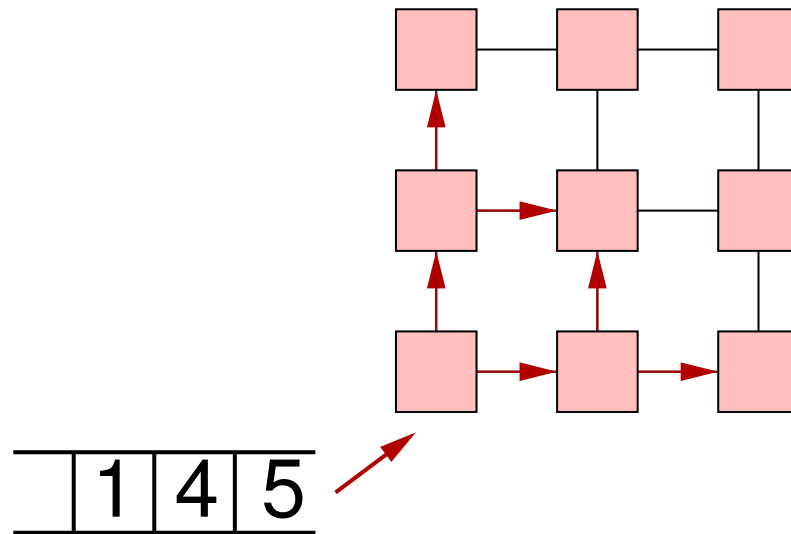
Maybe We Have Seen This Before

Systolic Arrays [Kung+Leiserson 1978]



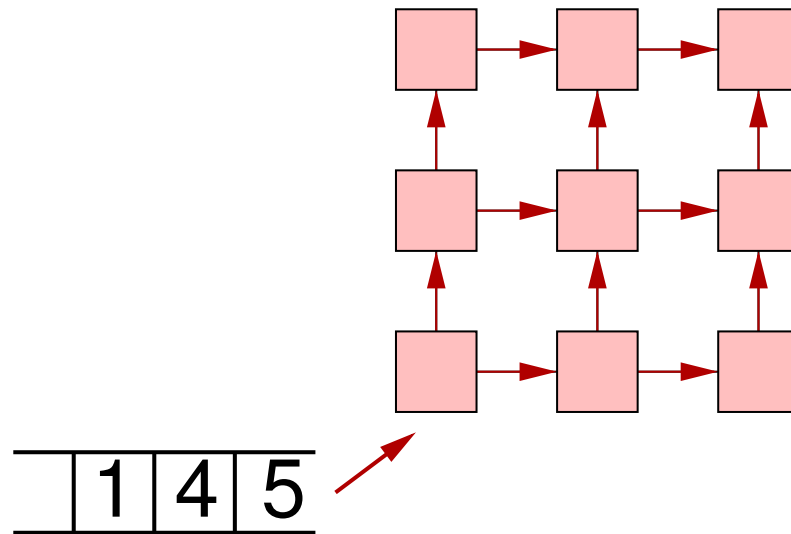
Maybe We Have Seen This Before

Systolic Arrays [Kung+Leiserson 1978]



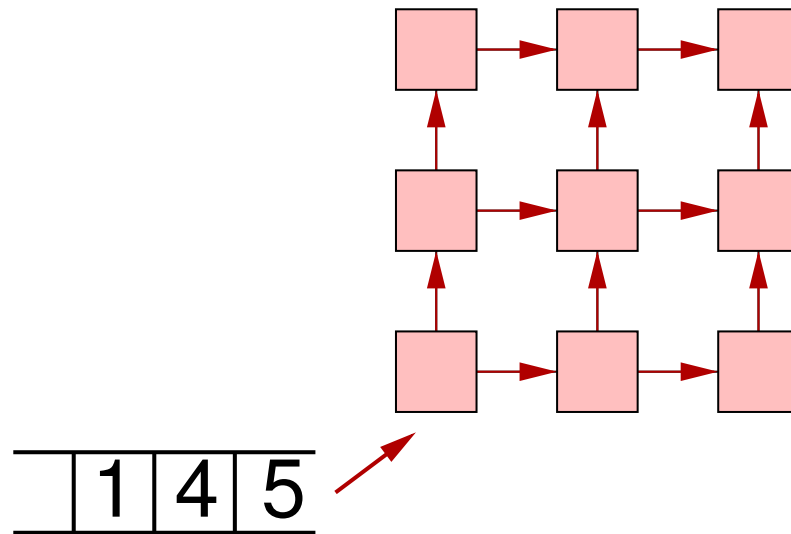
Maybe We Have Seen This Before

Systolic Arrays [Kung+Leiserson 1978]



Maybe We Have Seen This Before

Systolic Arrays [Kung+Leiserson 1978]



- Special case (1-D) of systolic arrays
- Have more memory access
- Early graphics card design was in the framework of systolic computation !

Graphics Card: Streaming Pipelined Architecture

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

Graphics Card: Streaming Pipelined Architecture

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

Graphics Card: Streaming Pipelined Architecture

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

Graphics Card: Streaming Pipelined Architecture

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

Graphics Card: Streaming Pipelined Architecture

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

Graphics Card: Streaming Pipelined Architecture

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

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

EXAMPLES

An Example: Voronoi Diagrams [HCKLM99]

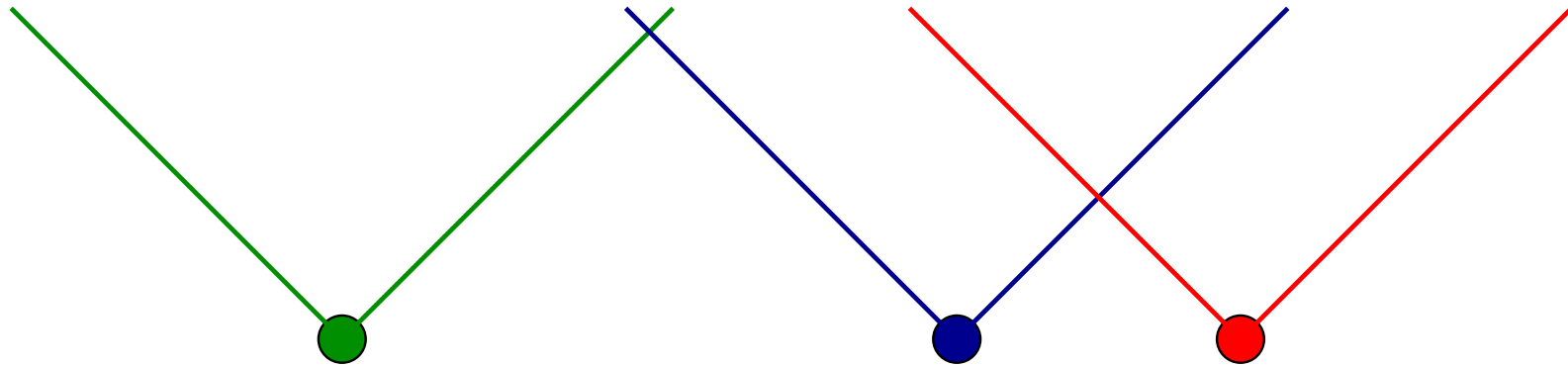


An Example: Voronoi Diagrams [HCKLM99]



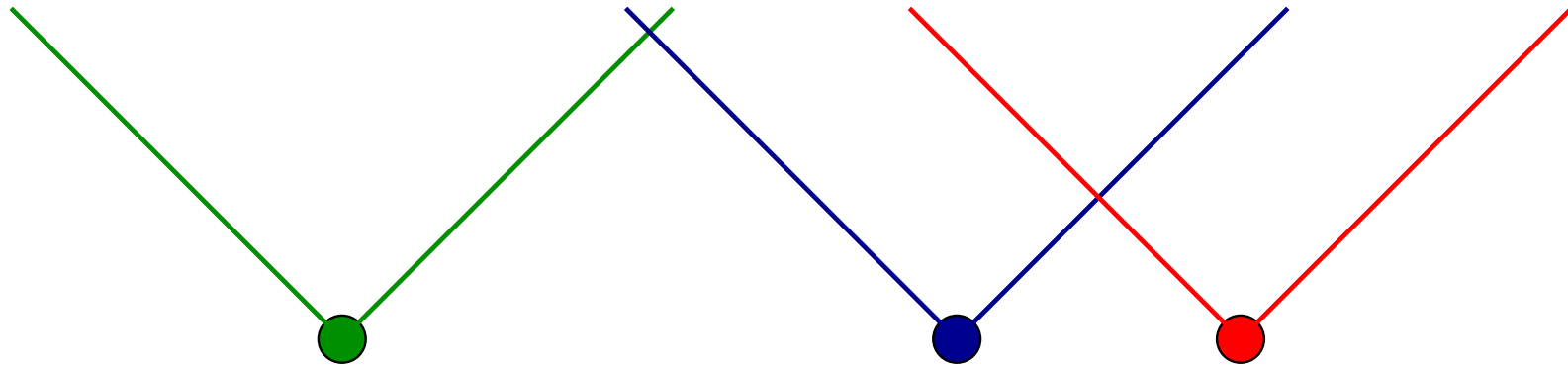
- Render right-angled cones for each point.

An Example: Voronoi Diagrams [HCKLM99]



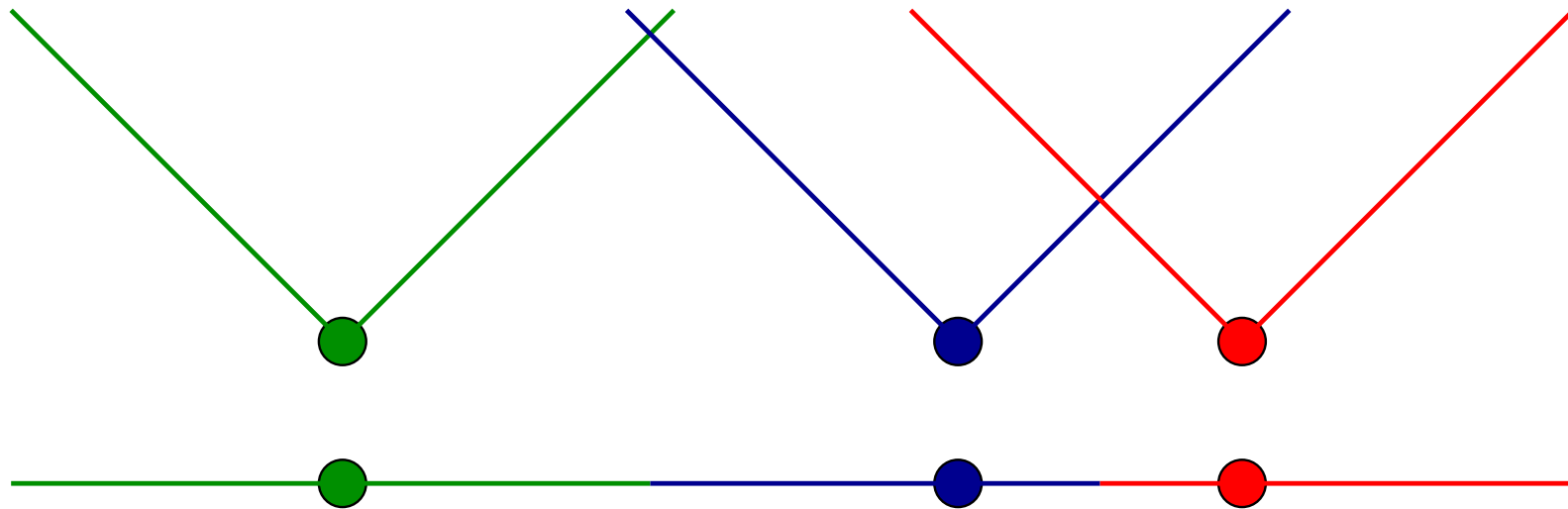
- Render right-angled cones for each point.

An Example: Voronoi Diagrams [HCKLM99]



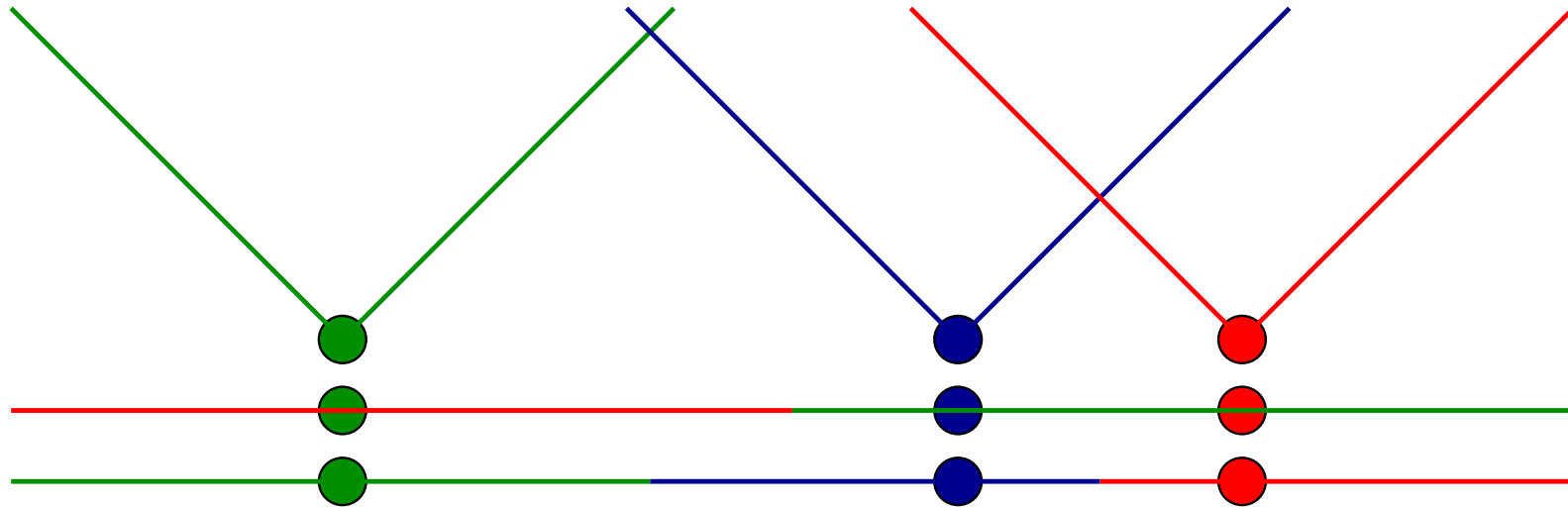
- Render right-angled cones for each point.
- Set *depth-test* to `LESS`, so only the closest points to the viewpoint are rendered.

An Example: Voronoi Diagrams [HCKLM99]



- Render right-angled cones for each point.
- Set *depth-test* to `LESS`, so only the closest points to the viewpoint are rendered.

An Example: Voronoi Diagrams [HCKLM99]



- Render right-angled cones for each point.
- Set *depth-test* to `LESS`, so only the closest points to the viewpoint are rendered.
- Also get diameter for free - using `GREATER`

Bounding Box [AKMV03]

- Each point in the primal is dualized to a plane.
- Framebuffer viewed as dual plane: Each pixel represents a direction
- Upper and lower envelopes in dual give extreme points (*a la* convex hulls).
- Superimposing different duals (using Gauss map), a simple fragment program computes the bounding box

Quantile Computation

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.

Easy in stream model:

- **[MP80]**: Computing in p passes requires $O(N^{1/p})$ memory.
- **[MRL98]**: $(1 + \epsilon)$ -approximation to rank in **ONE** pass with $O(1/\epsilon \log^2 \epsilon N)$ memory.
- **[GK01]**: $O(1/\epsilon \log \epsilon N)$ memory.

None of these algorithms are pipelined.

One- and two-sided tests [GKMV03]

With hardware, we have $O(1)$ memory $\Rightarrow \Omega(\log n)$ passes for *general streaming algorithm*.

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 $a < \text{fragment.depth} < b$?”

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

Where do we find a two-sided test ?

Shadow test in pipeline (only in nVidia chips) **[C02]**.

- Used to render shadows on objects.
- Functionally, provides (texture) buffer for second side of test.
- α -test is used to simulate second side.
- This can also be done using fragment programs.

Other areas where two-sided test is useful **[GKMV03]**:

- Sweeping an arrangement of shapes
- Used to compute boolean combinations of objects.

How Do We Write Programs

- Cg (from nVidia): C-like system calls are compiled into vertex and fragment programs.
 - Can compile for different targets (OpenGL/DirectX)
 - Can incorporate limits on programs on different cards
 - HLSL: Microsoft High Level Shader Language
 - GL 2.0: OpenGL Standard for higher level programming constructs.
- General Purpose Stream Programming
- High level stream programming constructs built over shader languages (BROOK)

Pipelined Streaming: Conclusions

- These architectures are ever more prevalent.
- Graphics chips a good platform for general purpose computing.
- Numerous applications; demonstrable performance gain.

Pipelined Streaming: Conclusions

- These architectures are ever more prevalent.
- Graphics chips a good platform for general purpose computing.
- Numerous applications; demonstrable performance gain.

What computational model do these architectures fit into ?

Pipelined Streaming: Conclusions

- These architectures are ever more prevalent.
- Graphics chips a good platform for general purpose computing.
- Numerous applications; demonstrable performance gain.

What computational model do these architectures fit into ?

- Strictly weaker for general streaming; probably stronger than circuits
- Results from systolic computation useful ?
- New ideas needed for proving upper/lower bounds because of multipass nature of computations.