

---

## CMSC 498M: Chapter 5 Graphics Architectures

### Sources:

- Lecture notes by Kurt Akeley and Pat Hanrahan of Stanford.

### Overview:

- Graphics architecture history
- Development of graphics hardware
- Streaming computation for graphics
- Parallelism in GPUs

Chapter 5, Slide 1  
Copyright © David Mount and Amitabh Varshney

---

## Graphics Architecture History

### Development of Interactive Graphics Systems:

#### Earliest systems:

- Flight simulators
- CAD systems

#### Platforms:

- Mainframe terminals
- Workstations
- PC graphics cards

#### Architectures characterized by:

- Parallelism
- Pipelining
- Lots of memory
- High bandwidth



Silicon Graphics Iris: circa 1988

Chapter 5, Slide 2  
Copyright © David Mount and Amitabh Varshney

## Graphics Architectures History

Improvements along three main axes:

### Performance:

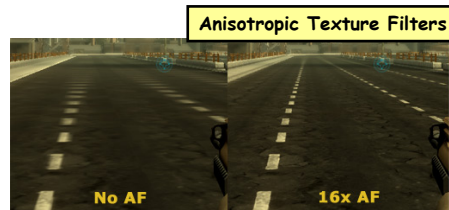
- Triangles / second
- Pixel fragments / second

### Features:

- Hidden surface elimination
- Image (Texture) mapping
- Antialiasing

### Quality:

- Bits of numeric resolution
- Image filters



Chapter 5, Slide 3  
Copyright © David Mount and Amitabh Varshney

## Parallelism in Graphics

### Functional Parallelism:

- **Pipelining:** A small number of distinct stages:  
Transformation → culling → lighting → scan-conversion (rasterization) →  
texturing → compositing.
- **Speed:** Limited by the slowest stage in the pipeline.

### Data Parallelism:

- **Object parallelism:** Geometry parallelism.
- **Image parallelism:** Rasterizer/fragment/pixel parallelism.

### Temporal Parallelism:

- Parallelism over **successive frames** (long animation sequences).

Chapter 5, Slide 4  
Copyright © David Mount and Amitabh Varshney

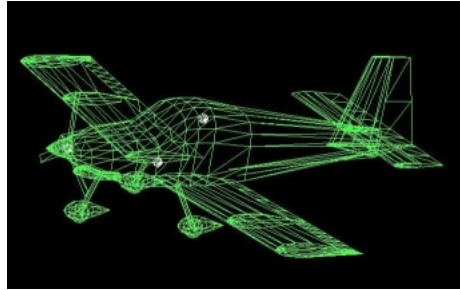
## First Generation: Wireframe Graphics

**Vertex:** Transform, clip, and project.

**Rasterization:** Color interpolation (points, lines).

**Fragment:** Overwrite.

**Dates:** Prior to 1987.



Chapter 5, Slide 5  
Copyright © David Mount and Amitabh Varshney

## Second Generation: Shaded Solids

**Vertex:** Lighting Calculation.

**Rasterization:** Depth interpolation (triangles).

**Fragment:** Depth buffer, color blending.

**Dates:** 1987 - 1992.



Chapter 5, Slide 6  
Copyright © David Mount and Amitabh Varshney

## Third Generation: Texture Mapping

**Vertex:** Texture coordinate transformation.

**Rasterization:** Texture coordinate interpolation.

**Fragment:** Texture evaluation, antialiasing.

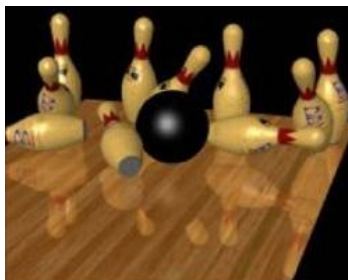
**Dates:** 1992 - 2000.



Chapter 5, Slide 7  
Copyright © David Mount and Amitabh Varshney

## Fourth Generation: Programmability

- Programmable shading.
- Image-based rendering.
- Convergence of graphics and media processing.
- Curved surfaces.



Chapter 5, Slide 8  
Copyright © David Mount and Amitabh Varshney

## Fifth Generation: Global Evaluation

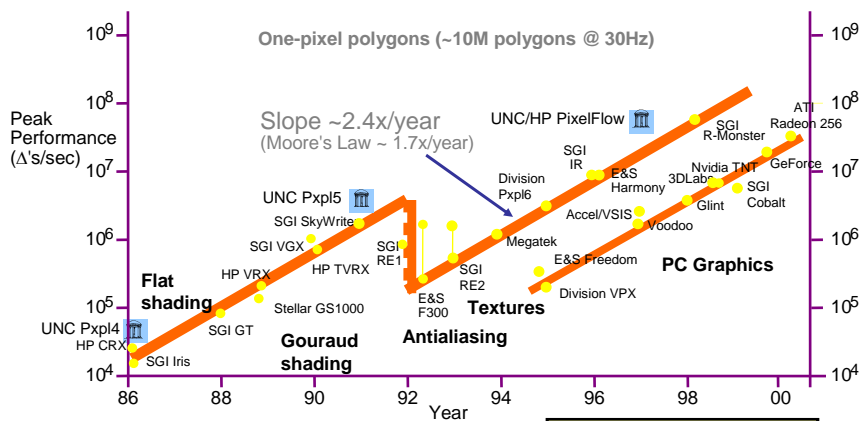
**Ray Tracing:** Visibility and integration.

**True Shadows:** Path tracing, photon mapping.



Chapter 5, Slide 9  
Copyright © David Mount and Amitabh Varshney

## Current Trends



Chapter 5, Slide 10  
Copyright © David Mount and Amitabh Varshney

## Motivations for Streaming

### When input and/or output data is real time:

- Video, audio, databases, network data, 3D graphics.

### When data is expensive to obtain or send:

- The **memory wall** problem. (Access time dominates compute time.)

### When the problem is compute intensive:

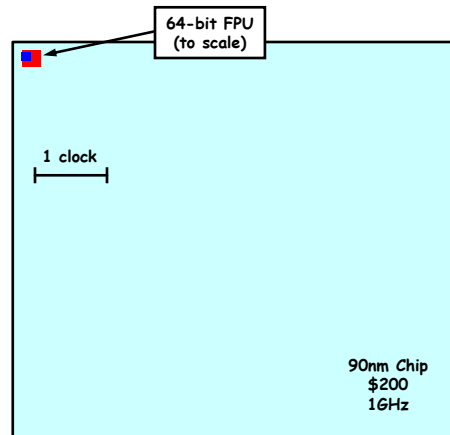
- Traditional architectures favor **memory-intensive apps**.
- One or two CPUs, a few GB of RAM, 100's GB of disk.
- A large fraction of on-chip CPU area is devoted to **caches** and their management.
- **Examples:** graphics, scientific computing, computational biology, multimedia processing, real-time computer vision, machine learning, real-time planning, ...

Chapter 5, Slide 11  
Copyright © David Mount and Amitabh Varshney

## Architect's Dilemma

### Chip Characteristics:

- Packing several **FPU** (floating point units) is now possible.
- But how to **communicate** among them and with RAM?
- Communication cost grows with **distance**.



Source: Bill Dally, Stanford

Chapter 5, Slide 12  
Copyright © David Mount and Amitabh Varshney

## Lessons from ASICs

### ASIC: Application-Specific Integrated Circuit

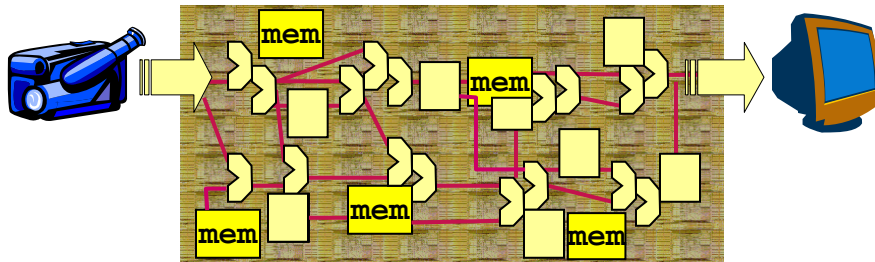
- Used in a wide variety of applications:
  - NPU - network processor units (wired and wireless devices).
  - GPU - graphics processor units.
  - SPU - storage processor units.
  - Encryption.
  - Audio/Video Processing.
  - Automotive industry, ...
- Works at a **lower clock speed** than a CPU, and yet provides a **higher throughput**.
- Typically much **cheaper** than a CPU.

Chapter 5, Slide 13  
Copyright © David Mount and Amitabh Varshney

## How does the ASIC do it?

### ASIC characteristics:

- ALUs, registers, **small local memories** distributed throughout.
- Carefully **hand-crafted connections** to minimize wire length.
- **Proximity** reduces power consumption.
- Stream data model for **high throughput**.



Chapter 5, Slide 14  
Copyright © David Mount and Amitabh Varshney

## How is Stream Computing Different?

---

### Von-Neumann Computers:

- **Classical CPU** model.
- Designed to do **many operations** on **each datum**.
- **Data stays (mostly) still**, while instructions flow past.
- Substantial set of different operations, but each has **fixed primitive function**.

### Data-Streaming Computers:

- Designed to do **same operation** on **many data**.
- **Operation stays still** (set up), and the data flows past.
- Want very powerful **vector operations**, so as to flow the data few times.
- A whole different way of programming.

Chapter 5, Slide 15  
Copyright © David Mount and Amitabh Varshney

## How GPUs are like Data-Stream Computers

---

### Based on fixed, primitive operations:

- Vertex transformations
- Lighting (non-programmed, that is)
- Rasterization
- ...

### Data (vertex/fragment) flows through processors:

- Permits a high degree of **parallelism**.

### Instructions use streaming table lookup:

- Vertex and fragment textures.

### Streaming to-memory operations:

- Copy to texture, render to texture.
- Z-buffering.

Chapter 5, Slide 16  
Copyright © David Mount and Amitabh Varshney

## CPU/GPU Computational Power

### GPUs are fast...

- 3.0 GHz dual-core Pentium4: 24.6 GFLOPs.
- NVIDIA GeForceFX 7800: 165 GFLOPs.
- 1066 MHz FSB Pentium Extreme Edition : 8.5 GB/s.
- ATI Radeon X850 XT Platinum Edition: 37.8 GB/s.

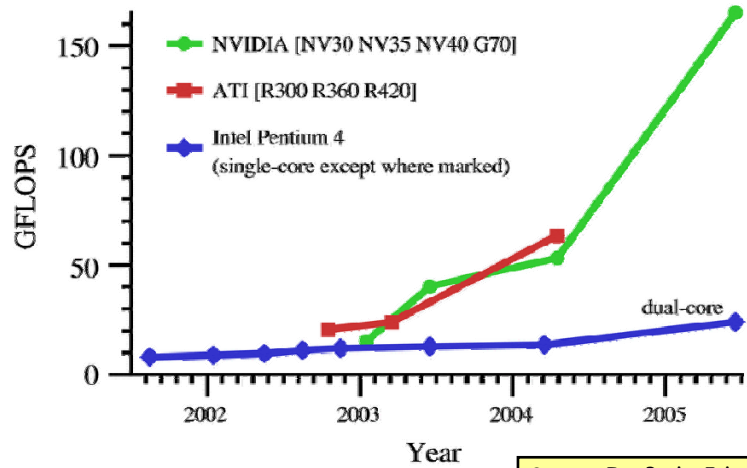
### GPUs are getting faster, faster:

- CPUs: 1.4× annual growth.
- GPUs: 1.7×(pixels) to 2.3× (vertices) annual growth.

Source: Kurt Akeley, Ian Buck & Tim Purcell,  
GPU Gems

Chapter 5, Slide 17  
Copyright © David Mount and Amitabh Varshney

## Current Trends

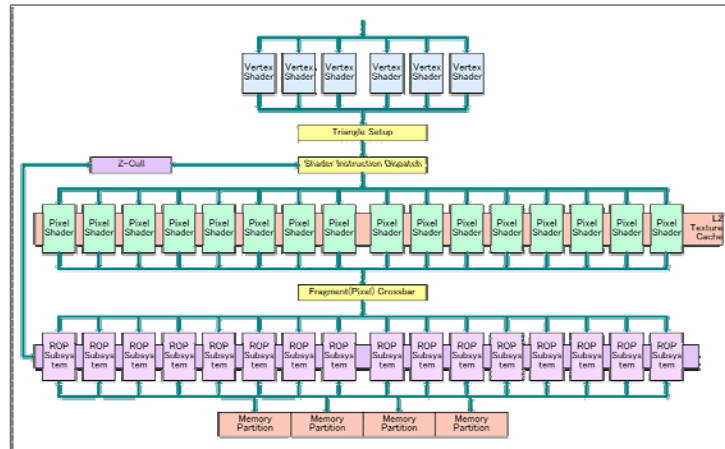


Source: Ian Buck, John Owens

Chapter 5, Slide 18  
Copyright © David Mount and Amitabh Varshney

## Example: NVIDIA NV40 Architecture

NV40 3D Pipeline



Copyright © 2004 NVIDIA. All rights reserved.

Chapter 5, Slide 19

Copyright © David Mount and Amitabh Varshney

## Types of Parallelism

**SIMD:** (Single Instruction, Multiple Data)

- **Same instruction executed synchronously** by all execution processing units (PUs) on different data.
- **Cheapest** to implement but most restrictive.
- Suitable for **fine-grained parallelism**.

**MIMD:** (Multiple Instruction, Multiple Data)

- Each processor executes **its own program**.
- Processors need fuller capabilities/memory.
- Better suited to **coarse-grained parallelism**.
- **Synchronization/communication** are important issues.

**SPMD:** (Single Program, Multiple Data)

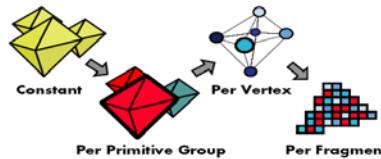
- A restriction of MIMD, where all processors execute the **same program**, but unlike SIMD, they do so **asynchronously**.

Chapter 5, Slide 20

Copyright © David Mount and Amitabh Varshney

## Relationship to Hardware

- Shading pixels and vertices is a **highly parallel task**.
- Constant and per-primitive (i.e, per triangle) group operations **do not require parallelism**.
- Current shading languages map the per-vertex and per-fragment operations to the GPUs as:
  - Vertex programs, and
  - Fragment programs, respectively.
- Vertex and fragment programs are executed in a **SIMD or MIMD parallel fashion**.



Source: Akeley and Hanrahan

Chapter 5, Slide 21  
Copyright © David Mount and Amitabh Varshney

## Summary of Current Shading Languages

Currently all shading languages follow a SPMD model:

- Implementation of SPMD can be on a **single processor, SIMD**, or **MIMD** machine.
- **SIMD**: handles **conditionals** by **disabling** a set of processors.
- **MIMD**: allows execution of **completely different programs**.

Every primitive and every sample is processed independently:

- No **scatter/gather** or **nearest-neighbor communications**. (Although the latest GPUs do support this.)
- Order independence implies no need for **inter-processor synch** and **no wires between processors**.
- Processors can be **packed more densely**.
- Execution can proceed **on groups of samples, pipelines**, or **one at a time** - whichever is most efficient.



Chapter 5, Slide 22  
Copyright © David Mount and Amitabh Varshney

## Summary of Current Shading Languages

---

### Favor computation over communication:

- **Communication costs are high** (VLSI, compiling, execution considerations).
- **Arithmetic intensity** (ratio of computation to bandwidth).

### Inter-processor communication is possible if absolutely needed (usually for general-purpose computation):

- **First pass:** Compute and write to texture memory.
- **Second pass:** Read from texture memory and compute.
- Any processor and **read any data from texture.**

Chapter 5, Slide 23  
Copyright © David Mount and Amitabh Varshney

## Summary

---

### Summary:

- Graphics architecture history
- Development of graphics hardware
- Streaming computation for graphics
- Parallelism in GPUs

### What's Next?

- Geometry and Modeling

Chapter 5, Slide 24  
Copyright © David Mount and Amitabh Varshney