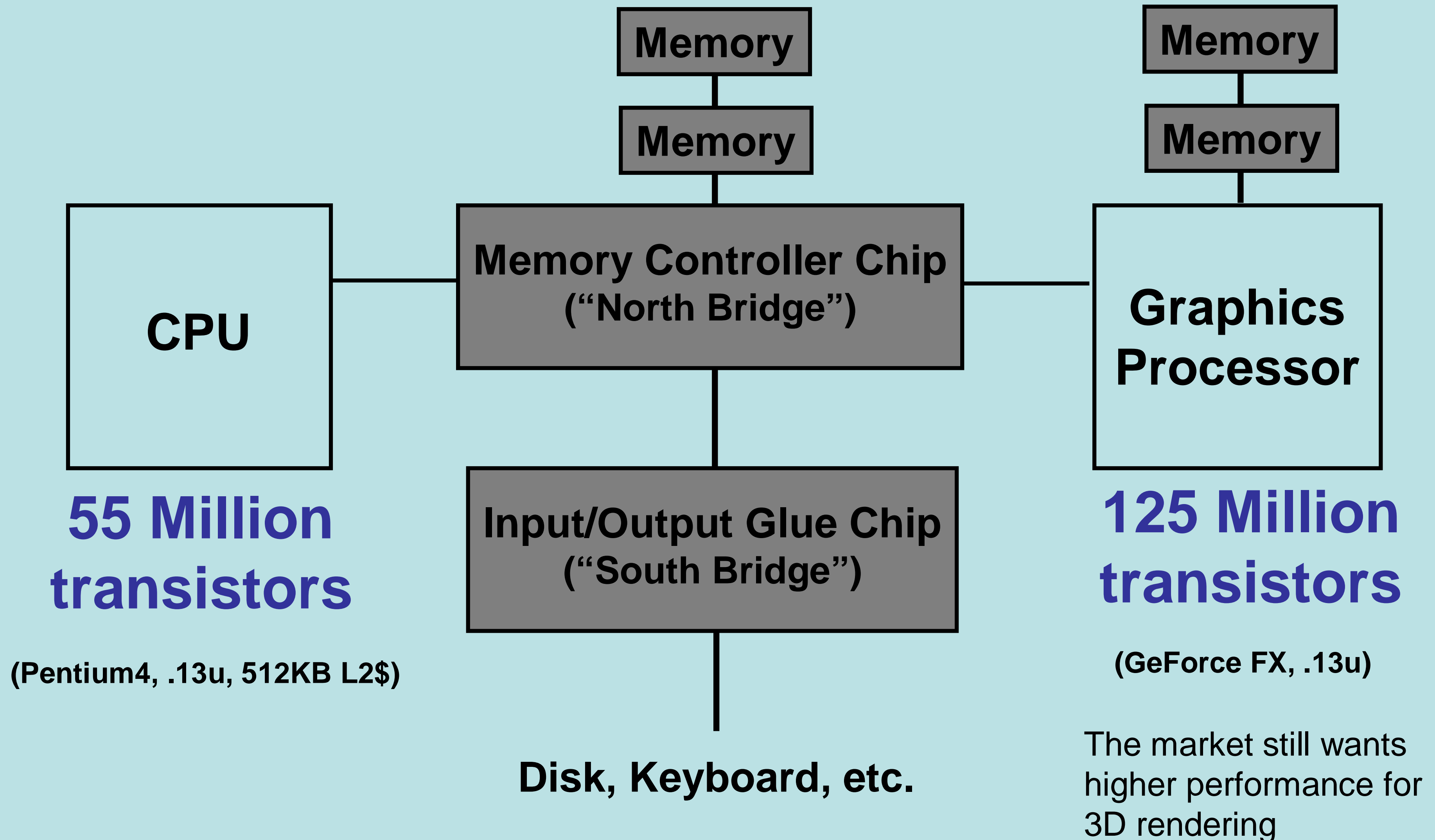
3D Graphics (Rendering) Overview

William R. Mark
University of Texas at Austin

Workshop on Streaming Systems
August 23, 2003

Two processors inside today's PCs



What is rendering?

Given a description of a 3D scene, generate a 2D image of the scene for a particular viewpoint and view direction.

Toy Story
Image Courtesy of Disney

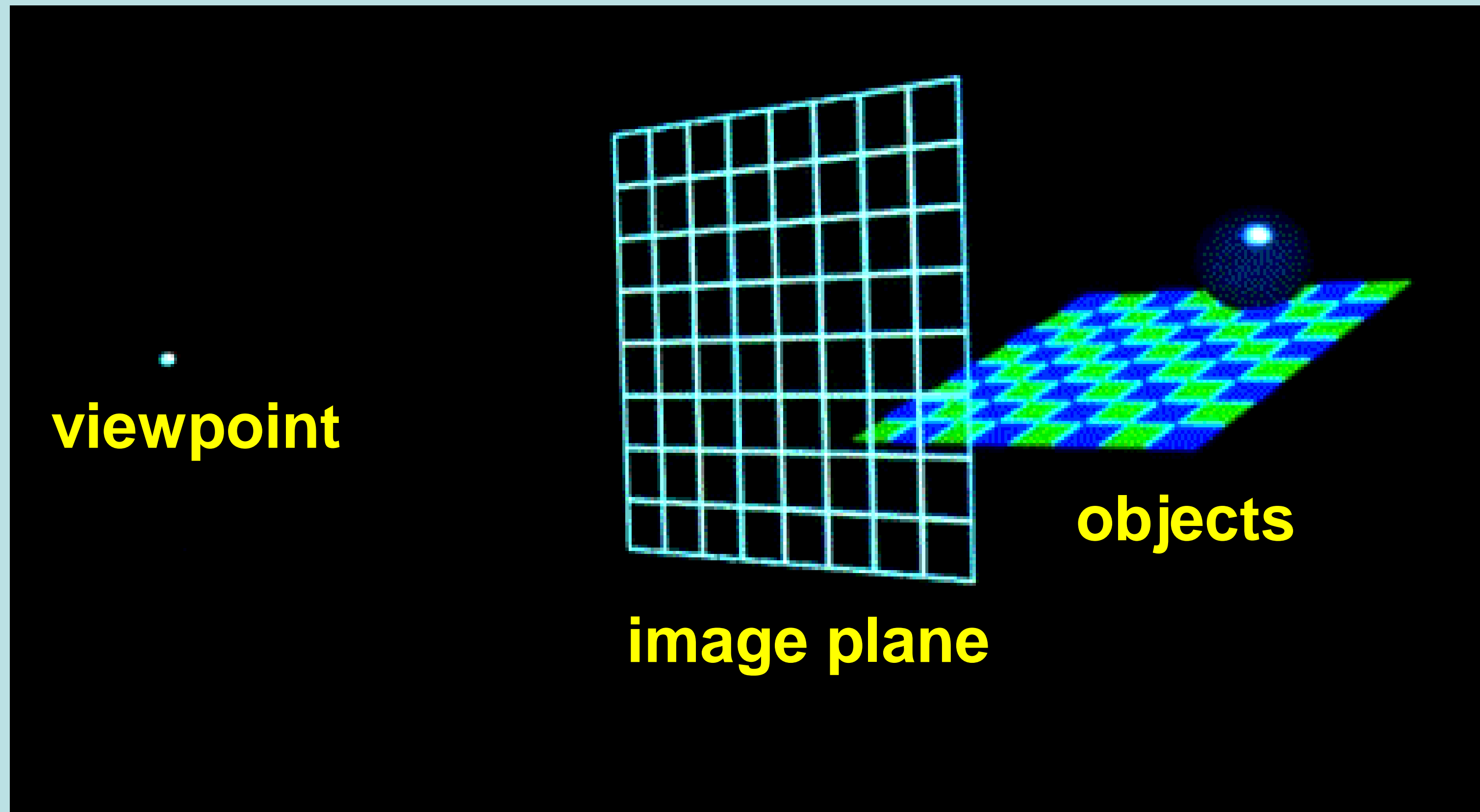
Computational tasks may include…

- Transforming between different coordinate systems
- Visibility determination
  (i.e. selecting nearest surface along a ray)
- Simulation of surface/light interaction
  - “local illumination”, or “shading”
- Simulation of global light transport
  - “global illumination”
- General data management
- And perhaps other kinds of simulation

A large variety of algorithms can be used for rendering

- Z-buffer with local illumination (today's graphics HW)
- Ray tracing
- More general Monte Carlo techniques
- Radiosity (solve sparse system of linear equations)
- Database lookup (image based rendering)
- Hybrids of the above; and more
- Computations are highly parallelizable

Z-buffer rendering – object order

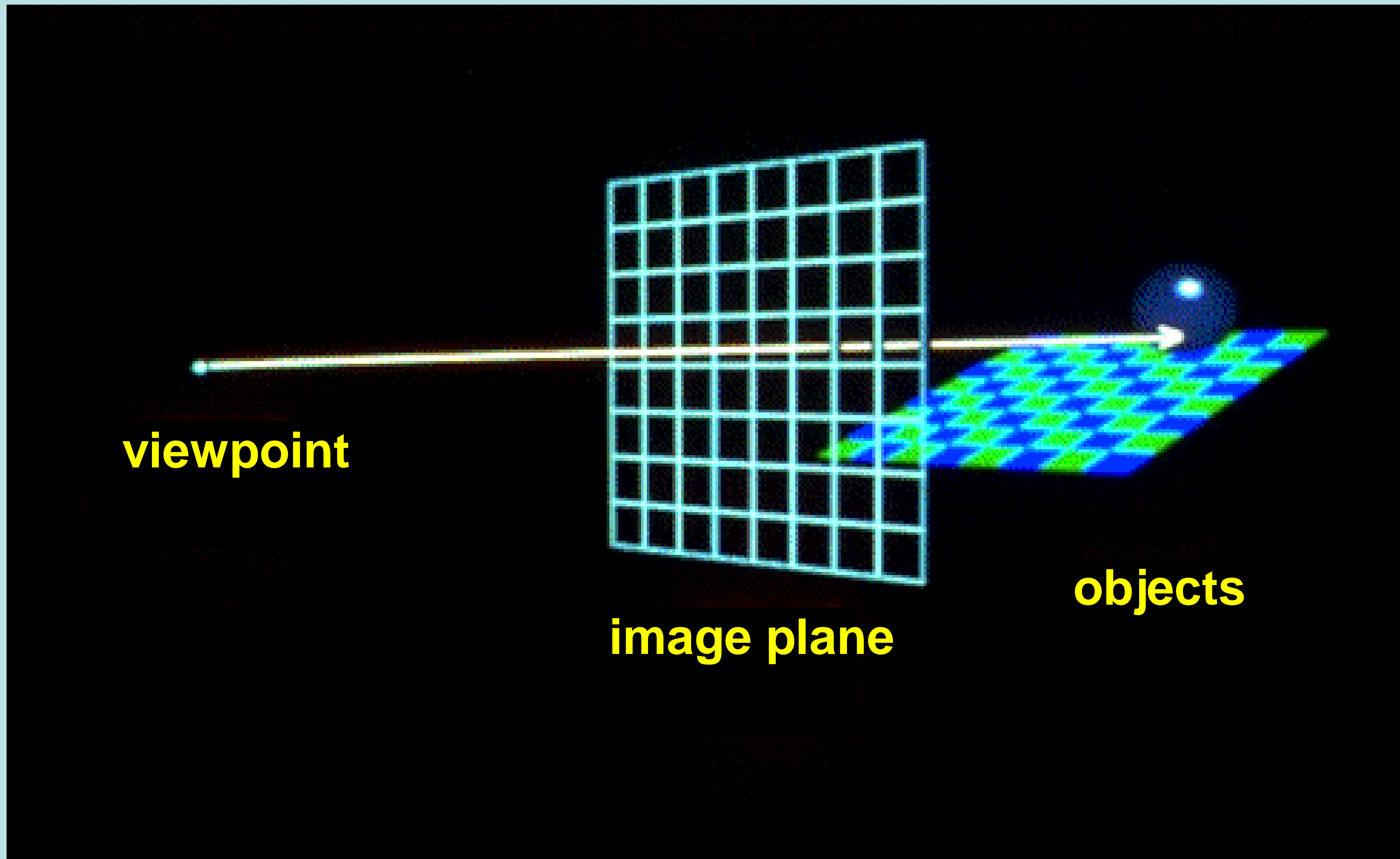


1) Project 3D object onto 2D image plane

2) Compute object color and distance at each pixel

3) Store color in image, if new distance is less than old

Ray tracing – image order



[source: SIGGRAPH 91 Education Slides]

Why is there so many different approaches to rendering?

- We are computing an **approximate solution**.
- The quality of the result is measured after **human perception**.
  - This space is very complex and task dependent
  - Many approximations are OK
- Some applications can be constrained to avoid weaknesses of algorithms!
  - Especially games
- Goal becomes: best image for $XXX.XX
- Yes, this is a vague goal
  - This makes rendering different from most other computations
  - Challenging to apply classical quantitative analysis
  - Contrast with goal of running SPEC and TPC faster
  - Often, researchers solve yesterday's graphics problems

In the longer term, graphics hardware goes through discontinuities

Slope ~2.4x/year
(Moore's Law ~ 1.7x/year)

Peak Performance (Δ's/sec)

10^8
10^7
10^6
10^5
10^4

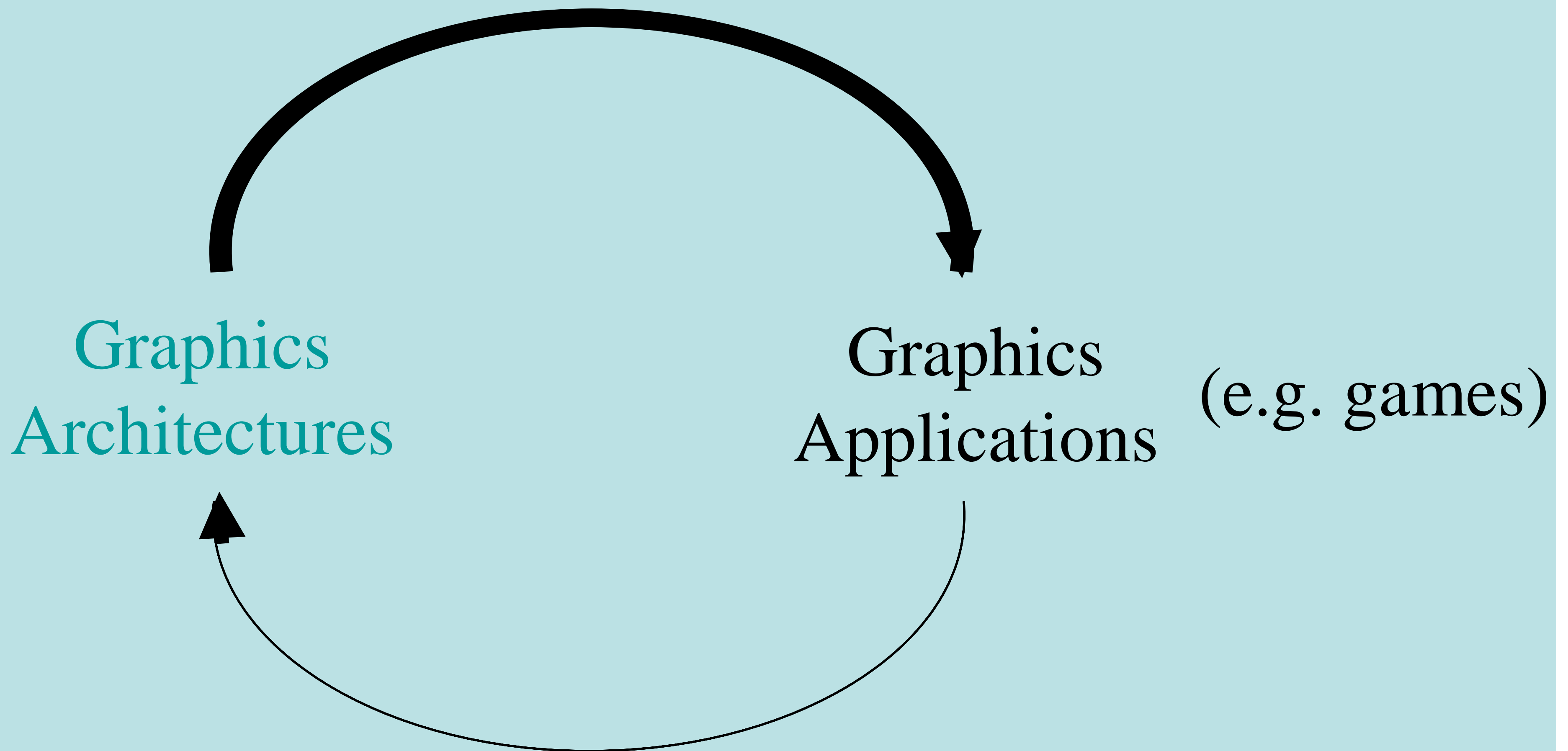
UNC/HP PixelFlow
SGI R-Monster
SGI IR
E&S Harmony
NVidia
3DLabs
SGI Cobalt
Division Pxpl6
Glint
Accel/VSIS
Voodoo
Megatek
PC Graphics
E&S Freedom
Division VPX
UNC Pxpl5
SGI SkyWriter
SGI VGX
HP TVRX
SGI RE1
SGI RE2
E&S F300
Textures
Antialiasing
HP VRX
Flat shading
Stellar GS1000
Gouraud shading
SGI GT
UNC Pxpl4
HP CRX
SGI Iris

86 88 90 92 94 96 98 00

Year

Source: John Poulton

In the shorter term, hardware strongly influences applications



Graphics HW is now programmable

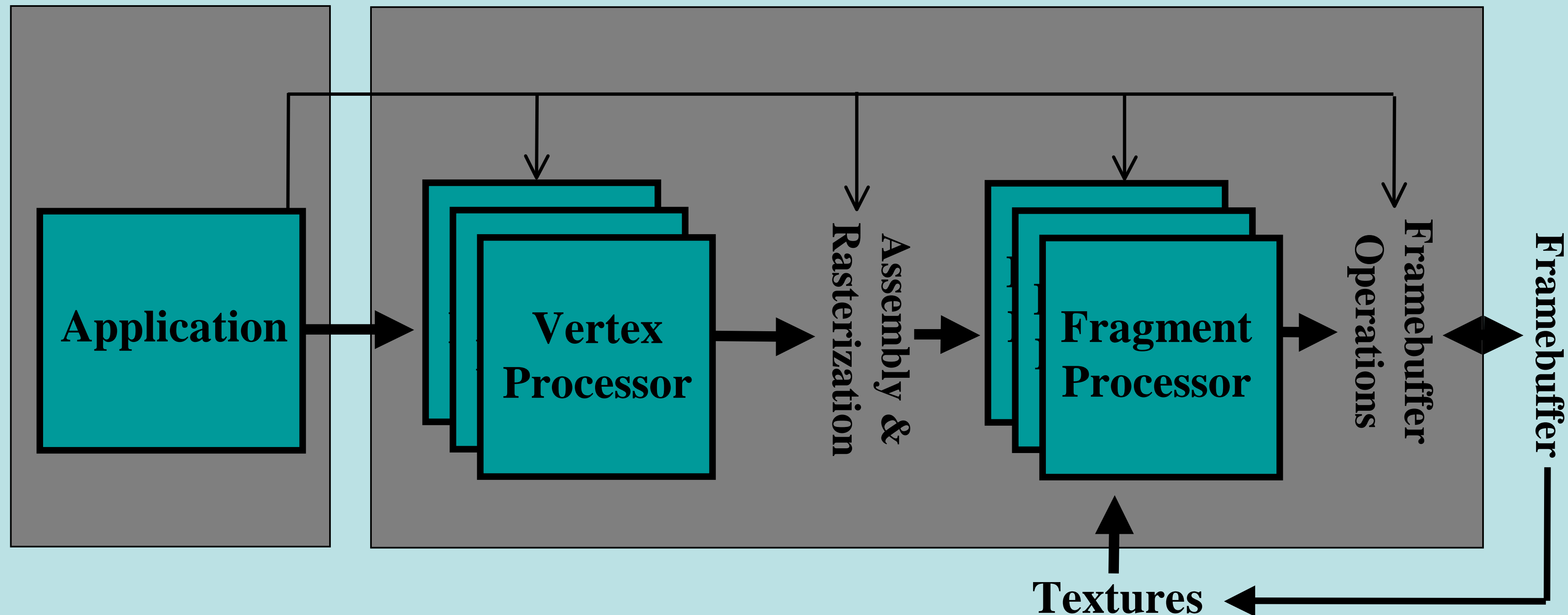
Industry goal:
“Toy Story in real time”

High-level programming languages:
e.g. Cg – C for graphics processors

```
...
L2weight = timeval - floor(timeval);
L1weight = 1.0 - L2weight;
ocoord1  = floor(timeval)/64.0 + 1.0/128.0;
ocoord2  = ocoord1 + 1.0/64.0;
L1offset = tex2D(tex2, float2(ocoord1, 1.0/128.0));
L2offset = tex2D(tex2, float2(ocoord2, 1.0/128.0));
...
```

But, programmability is embedded within hardwired Z-buffer pipeline

Embedding limits the usefulness of the programmability:
limited communication, synchronization, memory accesses



Some current trends

- Next frontier in image quality is global illumination:
  - Shadows
  - Precomputed approximations
  - Ray tracing – especially for dynamic scenes
- Z-buffer algorithm is ill-suited to these challenges
  - Difficulty even with shadows
- Benefits of specialization in graphics HW are decreasing
  - Focus for performance growth is on programmable parts of HW

Where we are today

CPU

Very flexible

Poor performance on highly parallelizable code

Graphics Processor

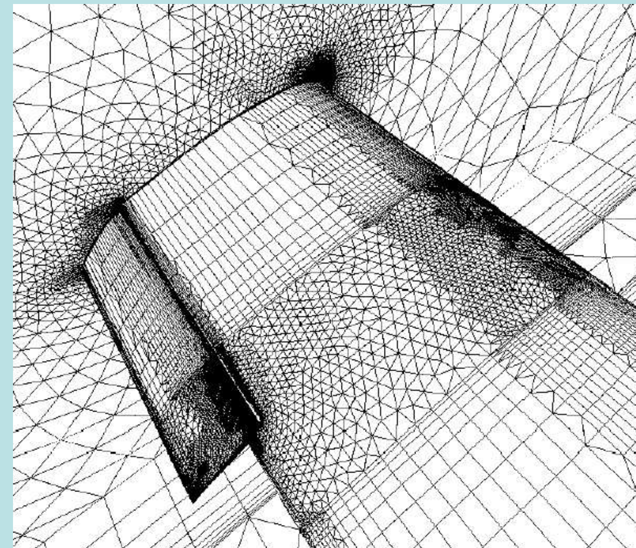
Highly specialized

High performance parallel computer

Inspiration from other areas: Smarter algorithms win

- These algorithms will not run well on today's specialized graphics processors.
- They need:
  - read/write caches
  - MIMD control flow
  - Inter-processor communication

Other classes of simulation have switched to adaptive algorithms



Adaptive meshing for finite element analysis
Source: D. Mavriplis

Today’s graphics architectures

Modern GPUs

Vertex Processor

Triangle Assembly & Rasterization

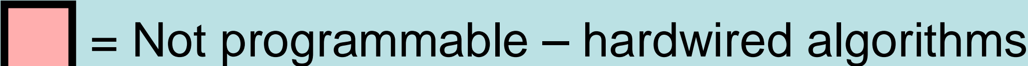
Fragment Processor

Framebuffer Operations

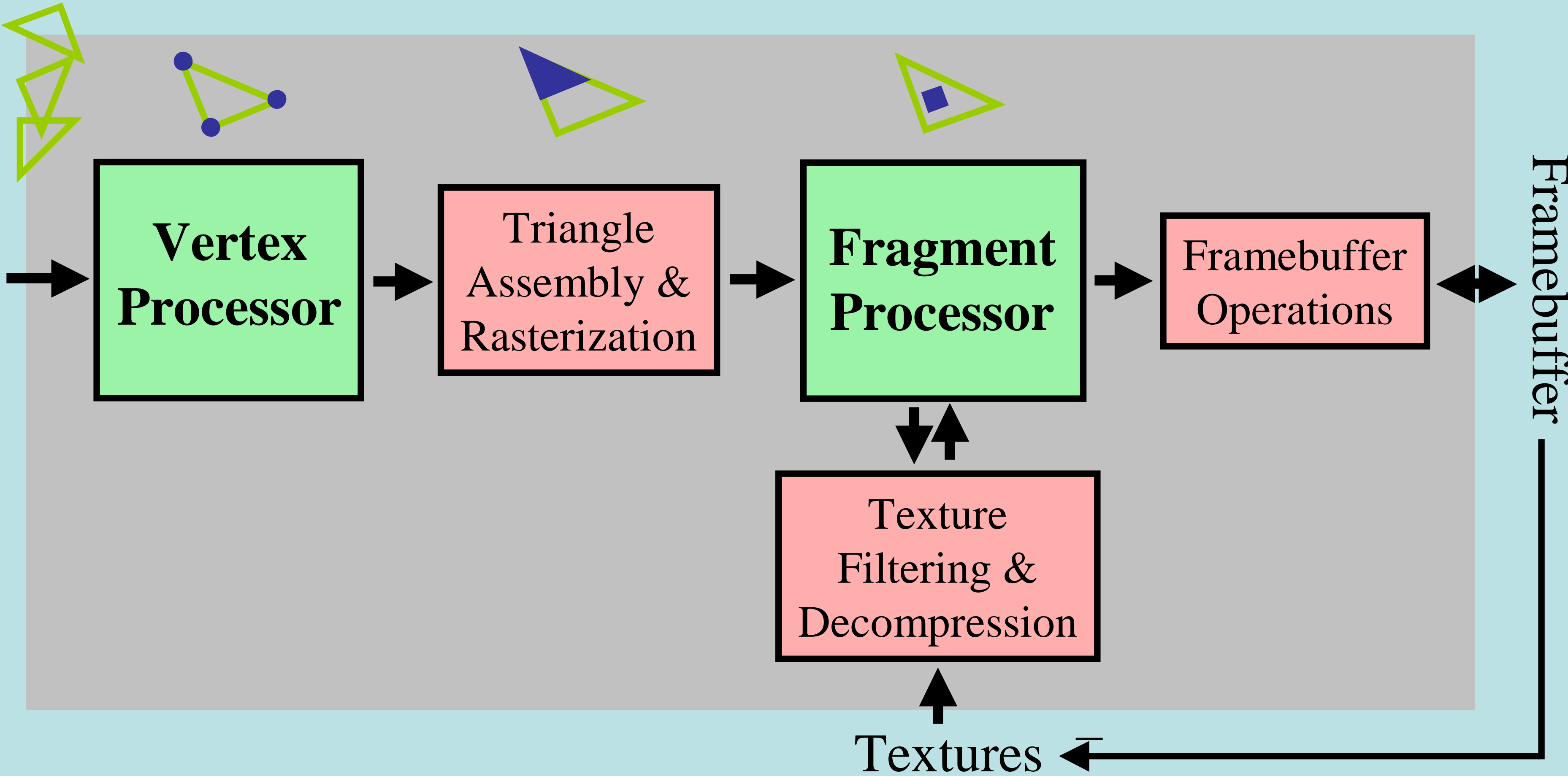
Framebuffer

Texture Filtering & Decompression

Textures



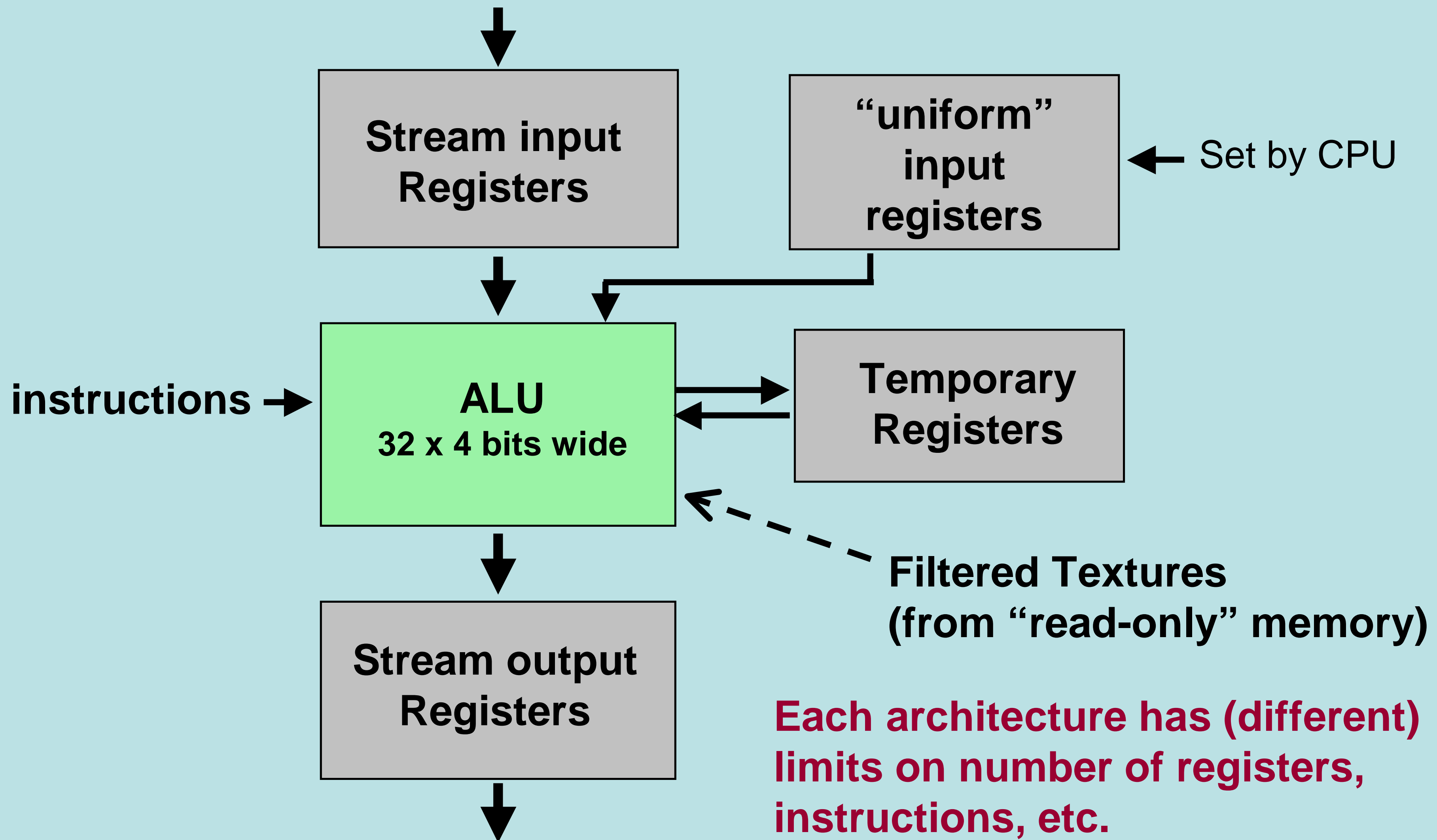
Modern GPUs



= Programmable

= Not programmable – hardwired algorithms

Inside a GPU's stream processor



Example instructions

- Short-vector arithmetic
  ADD R0, R1.wzxy, R2.xyzw
- Scalar arithmetic
  RSQ R3.x, R4.x
- Specialized arithmetic
  LIT R0, R1
- Texture lookup:
  TEX R5, R4, TEX5, 2D
- Control flow
  BRA target, (EQ.x)

Some more details…

SPMD

SIMD

Vertex Processor

Triangle Assembly & Rasterization

Fragment Processor

Framebuffer Operations

Framebuffer

CACHE

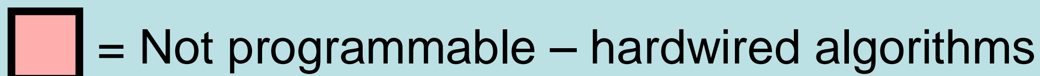
Texture Filtering & Decompression

“looping”

CACHE

CACHE

Textures



Don’t take this figure too literally

Coming soon – vertex textures

SPMD

SIMD

Vertex Processor

Triangle Assembly & Rasterization

Fragment Processor

Framebuffer Operations

Framebuffer

CACHE

Texture Filtering & Decompression

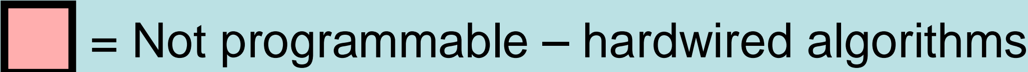
Texture Filtering & Decompression

"looping"

CACHE

CACHE

Textures



Current HW summary

- Limits on most resources
  - e.g. number of temporary registers, instruction count
- Supported functionality:
  - Most arithmetic operations (except bitwise/int)
  - Control flow, in vertex processor
  - Read from memory
- Missing functionality:
  - Inter-processor communication, synchronization
  - Write to memory using computed address (scatter)
  - Efficient reduction operations
  - Conditional streams

Why is graphics hardware fast?

Why is graphics HW fast?

- Specialization
  - Importance is decreasing, particularly for arithmetic
- Parallelization
  - Importance is increasing

Traditional Specializations (1 & 2)

- Low-precision, fixed-point arithmetic
  - For texture filtering, framebuffer blend
  - But, programmable units going to FP32
- Fixed algorithm – pure datapath HW
  - Never completely true for vertex ops
  - Now gone for fragment ops too
  - Still important for rasterization

Traditional Specializations (3 & 4)

- Customized memory-access management
  - Caching, latency hiding,
    batching of memory requests,
    texture decompression
  - Specialization still important, but
    programmability forces more generality
- Synchronization and communication
  - To maintain in-order rendering semantics
  - For atomic R/M/W blend, Z-test operations
  - To compute mipmap level
  - Specialization still important here

Parallelization

- Z-buffer algorithm is parallelizable
  - Many triangles, vertices, fragments
  - Programming models insure parallelism
- Why is this good?
  - Higher arithmetic performance (duh!)
  - On cache miss, can do other work
  - Allows memory system to be optimized for throughput instead of latency
- Very different from CPUs

NVIDIA Historicals

Season	Product	Mtri/sec	Yr rate	Mfrag/sec	Yr rate
2H97	Riva 128	5	-	100	-
1H98	Riva ZX	5	1.0	100	1.0
2H98	Riva TNT	5	1.0	180	3.2
1H99	Riva TNT2	8	1.0	333	3.4
2H99	GeForce	15	3.5	480	2.1
1H00	GeForce2 GTS	25	2.8	666	1.9
2H00	GeForce2 Ultra	31	1.5	1000	2.3
1H01	GeForce3	40	1.7	3200	10.2
1H02	GeForce4	65	1.6	4800	1.5

Source: Kurt Akeley, NVIDIA

1.8 2.4

Yearly growth well above CPU rate of ~1.5

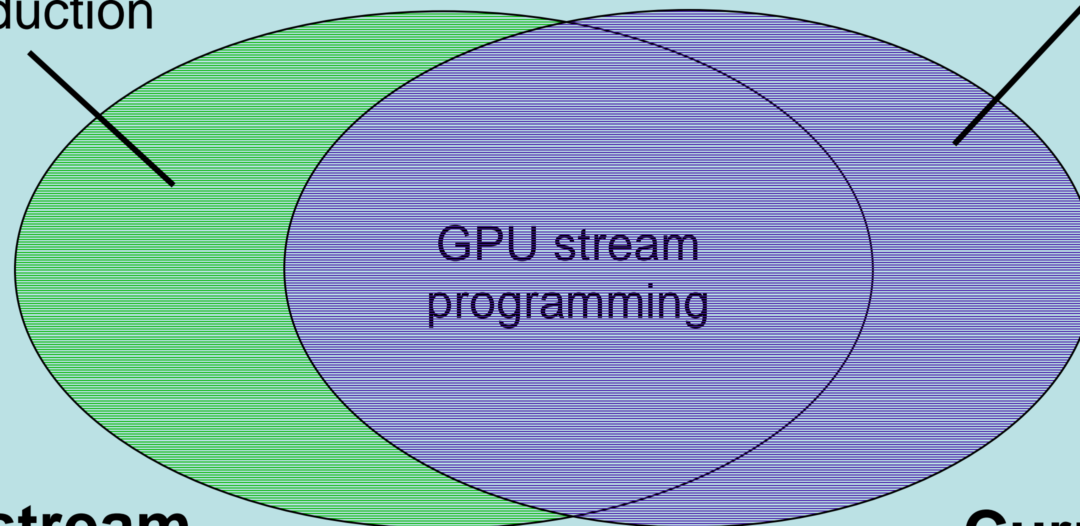
Two parallel computation models

- Stream processor
- Shared-memory multiprocessor

Full stream proc $\not\equiv$ GPU

- Scatter to memory
- Conditional kernel outputs
- Efficient reduction
- etc.

- Tagged caches
- R/M/W blend,Z
- etc.



Full stream programming
(e.g. Imagine processor)

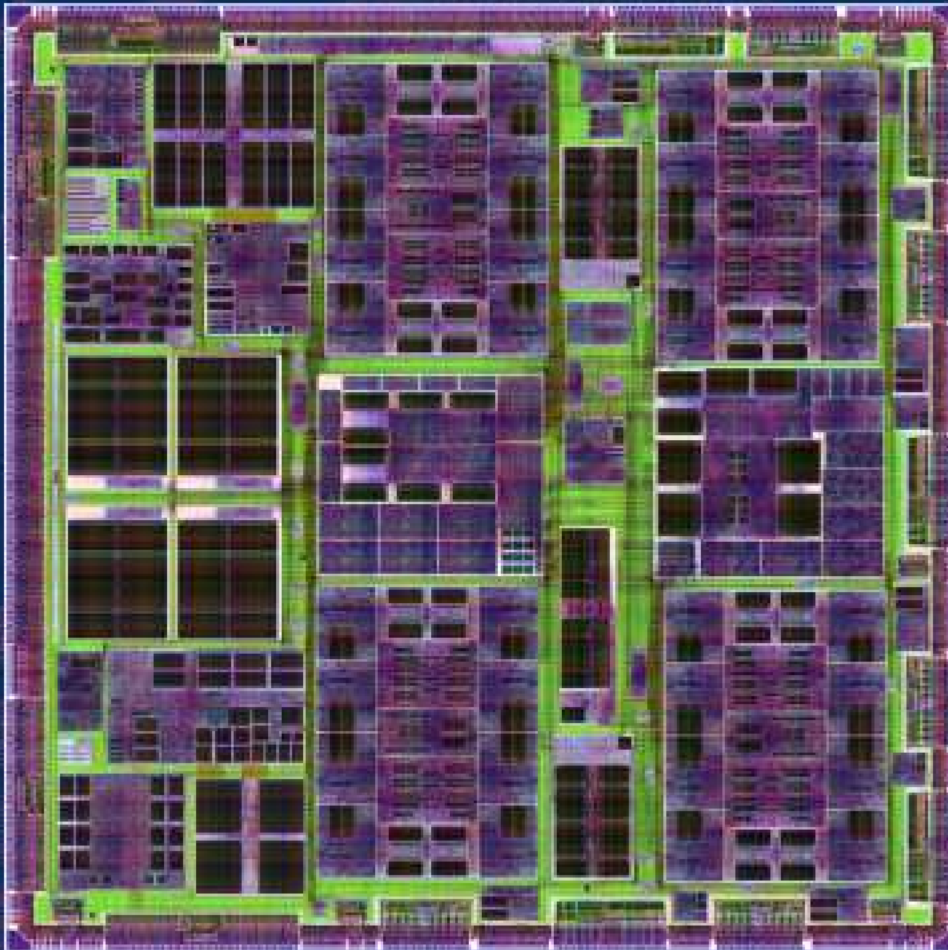
Current GPU's
(efficient Z-buffer rendering, with programmable shading)

Fortunately, many of the more general stream programming features that today's GPU's lack could be added with minimal impact on cost and rendering performance

Another parallel model: shared-memory MIMD

- More flexible than stream model
  - But more expensive – communication, I-mem
  - How much of this flexibility is worth the cost?
- Multi-chip examples:
  - SGI Origin, Sun Enterprise, etc.
- Programming model
  - Lots of processors, each can run different code
  - Processors share an address space
  - Built-in support for synchronization, etc.
  - Use a library like ‘pthreads’

A current example – network processor



- 32 processors
  - **256 Threads**
- 128.1 Gbps Aggregate I/O Bandwidth
  - **64 Gbps HCC**
  - **51.2 Gbps SPI 4.2**
  - **10.8 Gbps QDR**
  - **2.1 Gbps PCI**
- Speed Grades: 266, 300, & 333 MHz
- Power: 18.5W, typical
- 175M transistors
- 1036 pin HPBGA
- 0.13u "G" TSMC

O'Connor, Mike. "Inside the iFlow 20Gbps Packet Processor," 2002 Embedded Processor Forum.

Summary

- The programmable engines in current GPU's can be thought of as limited stream processors
  - Other parts of GPU are less stream-like
  - GPU's could become good stream processors
  - Stream processors don't make good GPU's
- Future GPU's could incorporate additional stream-processing functionality
  - Incremental change from today's designs
- But that may not be the end of the story
  - More general parallel computation models are attractive to support different graphics algorithms
    - Particularly the creation of dynamic data structures
  - Is the cost worth it?
  - We would still want support for stream programming

The End