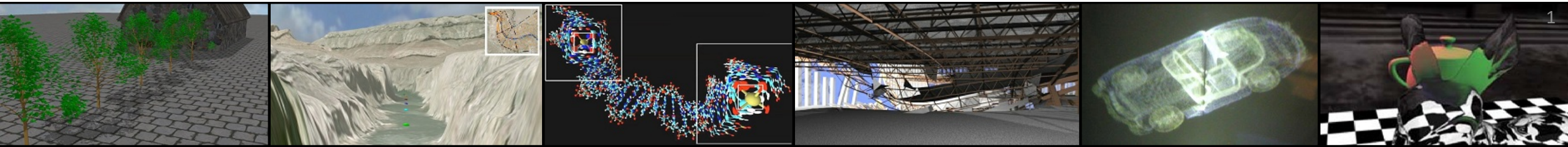
CIS 4930-001: Introduction to Augmented and Virtual Reality

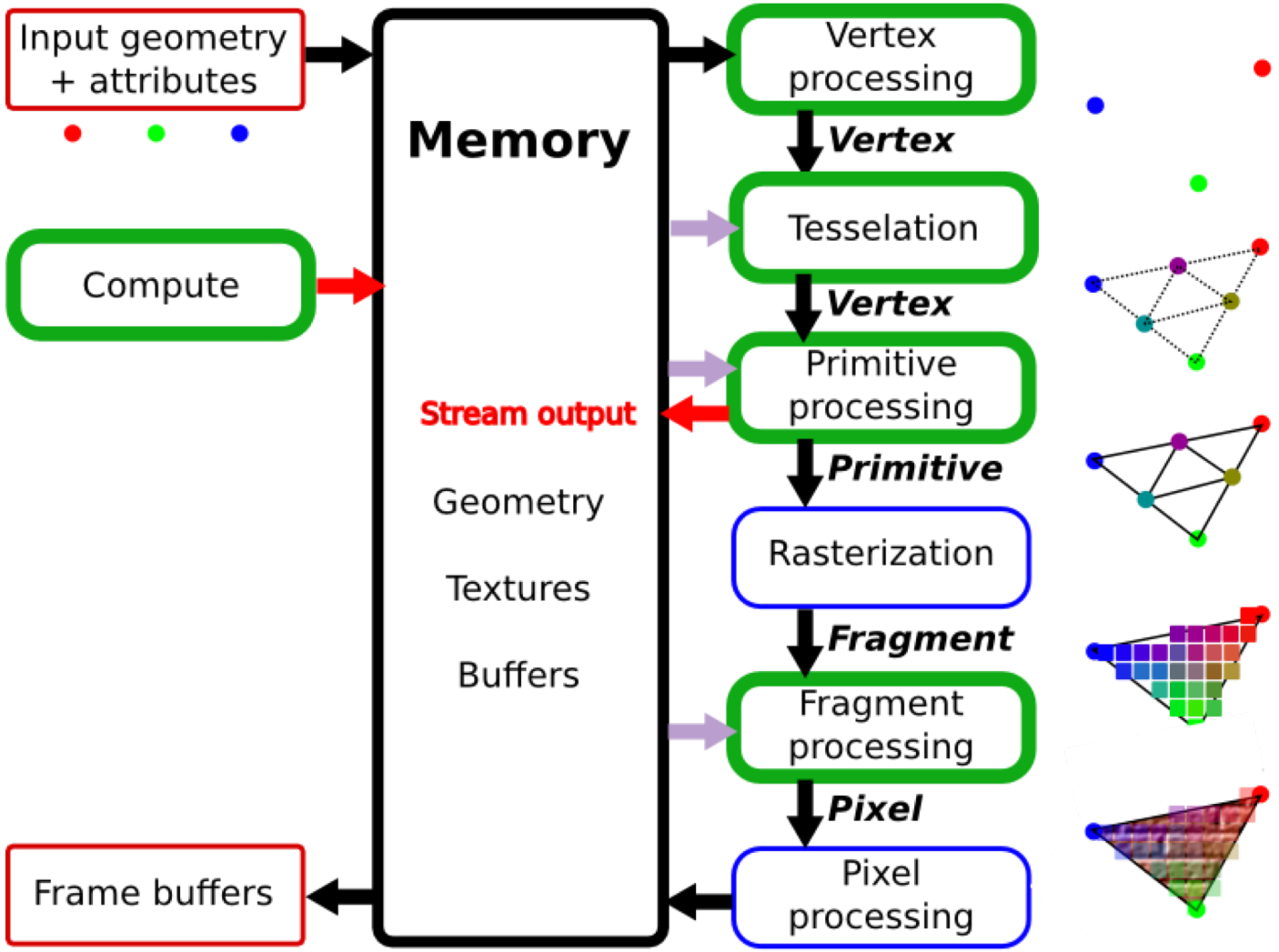
Graphics Pipeline

Paul Rosen
Assistant Professor
University of South Florida

Some slides from: Anders Backman, Mark Billinghurst, Doug Bowman, David Johnson, Gun Lee, Ivan Poupyrev, Bruce Thomas, Geb Thomas, Anna Yershova, Stefanie Zollman

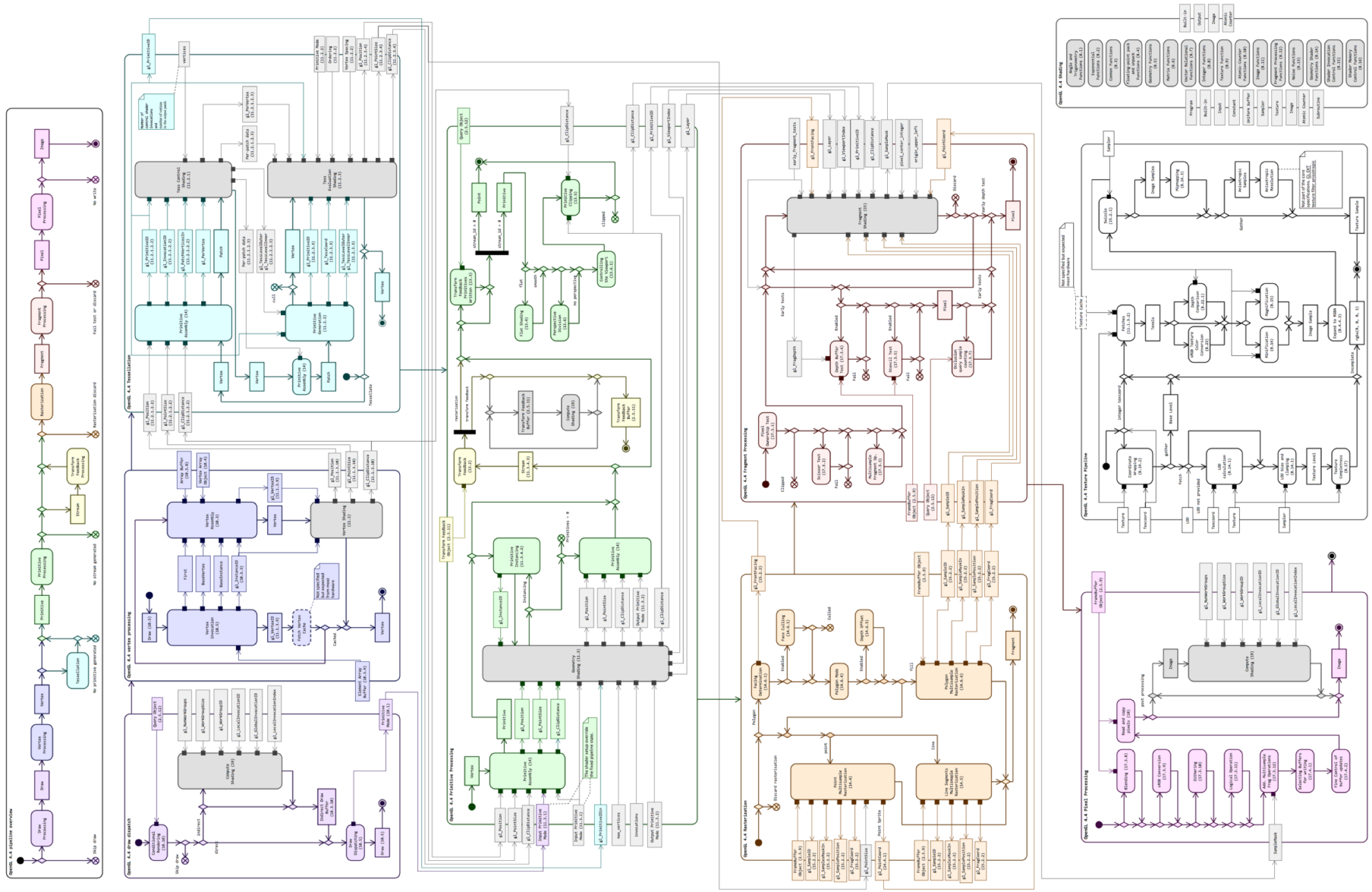


A Simplified View of the Graphics Pipeline



http://romain.vergne.free.fr/teaching/IS/SI03-pipeline.html

FULL OPENGL 4.4 PIPELINE



HTTP://OPENGLINSIGHTS.COM/PIPELINE.HTML

THE "REAL" VIRTUAL REALITY

What does it take to create a virtual world indistinguishable from the real, in the visual sense?

- ***80 million polygons/frame is threshold of "reality", a sufficiently rich approximation that audiences cease to be concerned about its authenticity.***
  - Prediction from Alvy Ray Smith, (Microsoft Graphics Guru) Co-founder of Pixar
- 80M ppf * 60 fps = 4.8 Gpolygons/second

Nvidia GeForce Titan RTX

- Conventional: 170 Gpixels/second
- Ray-tracing: 11 Brays/second
- 4K Resolution: 3840x2160=8.3Million pixels

So, we're approaching/surpassing the reality threshold

Real time "A Bugs Life"

1998 Pixar/Walt Disney released A Bugs Life.

Generating an average frame took 3-4 h (even up to 20h) on a 330Mhz UltraSPARC

For a minimum frame rate of 12 fps would take a speed-up of 150,000 times

Given Moore's law, (acceleration rate of 2 times every 18 months)

- would get us to year 2024.

Graphic chips have developed faster than that

Graphic chipsets doubling every 6 months

- Would lead to 2007

Real time bugs life – Unreal Engine

https://youtu.be/qlyzo9ll9Vw

Real Time "A Bugs Life"

Very unlikely trend can continue

- Electron leakage and Quantum effects will appear when the electronics is shrink far enough. (Colin P. Williams et. al., Explorations in Quantum Computing)

Methods for optimizing the graphics, both in software and later in hardware helped with performance and realism leaps

Real Time Graphics

James Helman in "Architecture and Performance of Entertainment systems" SIGGRAPH-94 noticed that

- If frame rate is lower than display refresh rate (e.g., 60Hz), motions are experienced unnatural
- Ideally we would like to have frame rate == display refresh rate (generally 60Hz)

FRAME RATE - SO VERY IMPORTANT FOR VR

Low frame rate:

- Latency for the user
- No presence

Uneven frame rate

- Due to different object detail in the scene
- Jerky graphics
- Hard to interact with objects when speed in unpredictable

Constant high frame rate – What we wish for

- 6 fps, a sense of interactivity
- 15 fps is real time
- 30-60 minimum target for games
- > 72 fps changes in display rate is undetectable

Why is high frame rate hard: Example

Display

- 1280 x 1024; 24 bit (rgb); 32 bit (depth)
- 30 Hz update rate

Scene

- 100,000 triangles
- Ambient and diffuse illumination

Per-Frame Geometry Calculations

Geometry Traversal

- 300,000 vertices, normals (6 FP load/store)
- 100,000 start/end tokens, (2 integer load/store)
- 2M loads/stores per frame

Modeling transformation

- 300,000 vertices, normals (25 mults + 18 adds)
- 7.5M mults, 5.4M adds per frame

Trivial accept/reject (gross clipping)

- Dot each vertex against six planes
- 24 mults + 18 adds per vertex: 7.4M mults, 5.4M adds per frame

Per-Frame Geometry Calculations

Lighting

- 12 mults + 5 adds per vertex: 3.6M mults, 1.5M adds per frame

Viewing transformation

- 8 mults + 6 adds per vertex: 2.4M mults and 1.8M adds

Clipping

- Highly variable
- Division by w (assuming all primitives within view frustum)
  - 3 divs per vertex, 0.9M divides

Transform to NDC

- 2 mults + 2 adds per vertex: 0.6M mults, 0.6M adds

Per-Frame Geometry Calculations

Total for geometry:

- 30+ million loads & stores (depending on dataflow)
- 22M mults/divides + 15M adds per frame

At 30Hz frame rate, more than **TWO GigaFlops**!
(Not including application, system overhead, rasterization, etc.)

MEASURING PERFORMANCE

Common Measurements

- GFLOPS
- Pixels per second (GP/s)
- Texels per second (GT/s)
- Memory bandwidth (GB/s)

H/W manufacturers present peak rates

- Close to impossible to reach

Real world tests

- Most tests are based on games: Quake, Unreal, etc…
- Measures the throughput of the whole system

WHY HARDWARE?

Why special purpose hardware for graphics?

- Even transmitting the geometry over the bus takes time
- Example:
  - Triangle requires 36 bytes (normals, texture coords, coords)
  - 1 Million polygons
  - 36 Mb/s only for geometry
  - 60 fps -> 2080 Mb/s throughput between processor and graphics hardware.
  - Textures can take up Giga bytes of data
  - Animation data etc...

We need special hardware for an efficient handling of this.

- Upload geometry/textures once.

BEFORE GPUS

Specialized solutions

Very expensive

Small market

Intergraph Wildcat 500 80Mb AGP

SGI Octane

1st Generation GPUs

Fixed function only (No shaders)

Performance (TNT2)

- 150 MHz

230 Million pixels/second

2.4 GB/s memory bandwidth

ATI Rage2

NVIDIA TNT2

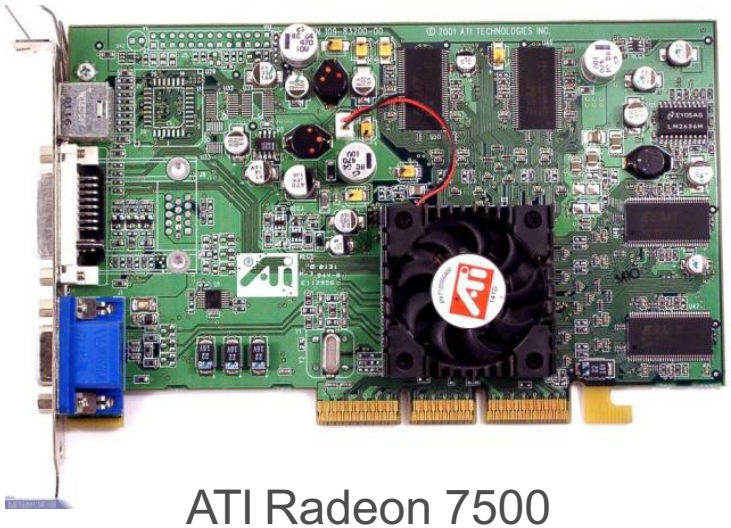
2ND GENERATION

First 'GPU'* (Geforce256)

Still fixed function (no shaders)

Added H/W T&L (Transform and Lighting), cube mapping, and bump mapping

Nvidia Geforce 256



ATI Radeon 7500

* "A SINGLE-CHIP PROCESSOR WITH INTEGRATED TRANSFORM, LIGHTING, TRIANGLE SETUP/CLIPPING, AND RENDERING ENGINES THAT IS CAPABLE OF PROCESSING A MINIMUM OF 10 MILLION POLYGONS PER SECOND." PER NVIDIA

3RD GENERATION

Programmable Vertex shaders

No programmability for Pixel shaders

Nvidia Geforce 3

XBOX

ATI Radeon 8500

4th Generation Cards

Pixel Shaders

Loops and Branches in Vertex shaders (not in Pixel shaders)

Nvidia GeForce FX 5650

Nvidia GeForce FX 5650

5th Generation Cards

Loops and Branches also in Pixel Shaders

Multiple Render Targets (MRT)

Nvidia GeForce 6800

ATI Radeon X800

6TH/7TH GENERATION

Geometry Shaders

Tessellation Shaders

Nvidia GeForce 8800

ATI Radeon X1600

8TH GENERATION STATE OF THE ART

Real-time Ray Tracing

NVIDIA GeForce RTX 2080 Ti

AMD/ATI Radeon 5700 XT

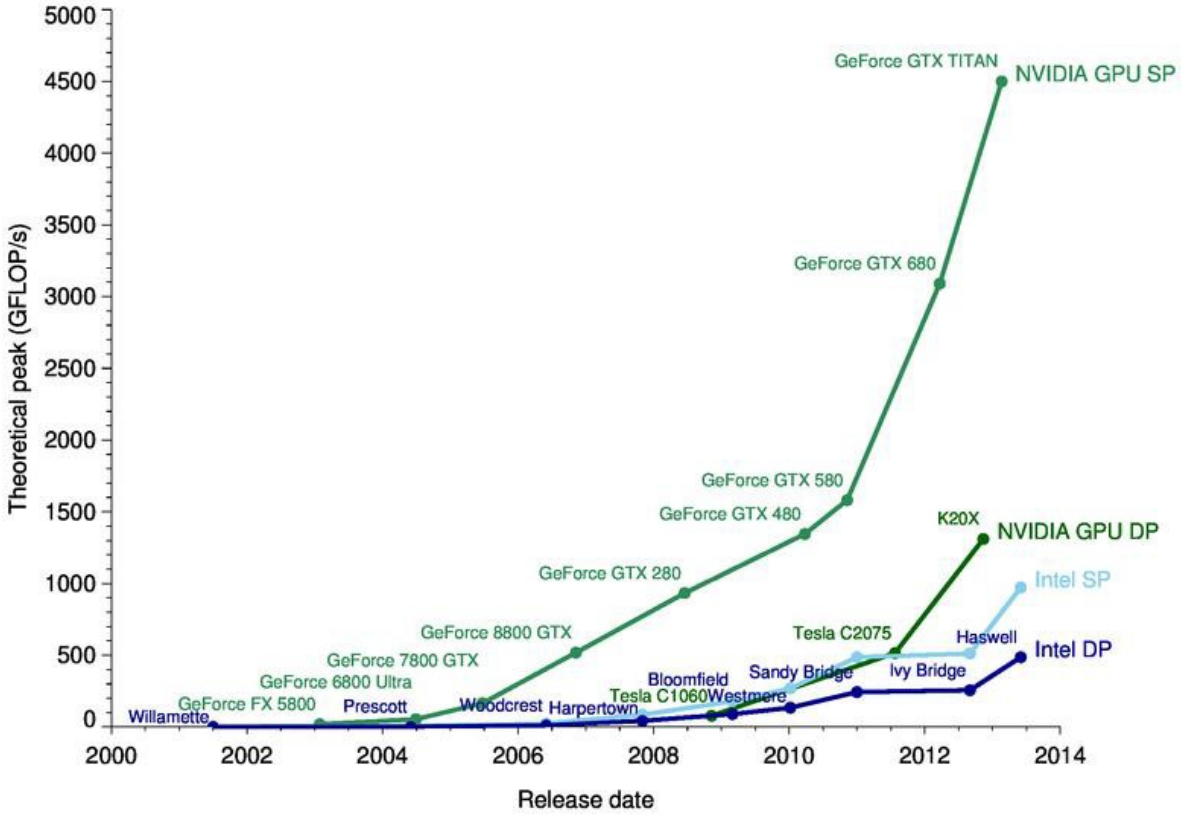
Specs – Old vs New

OLD (~2005)

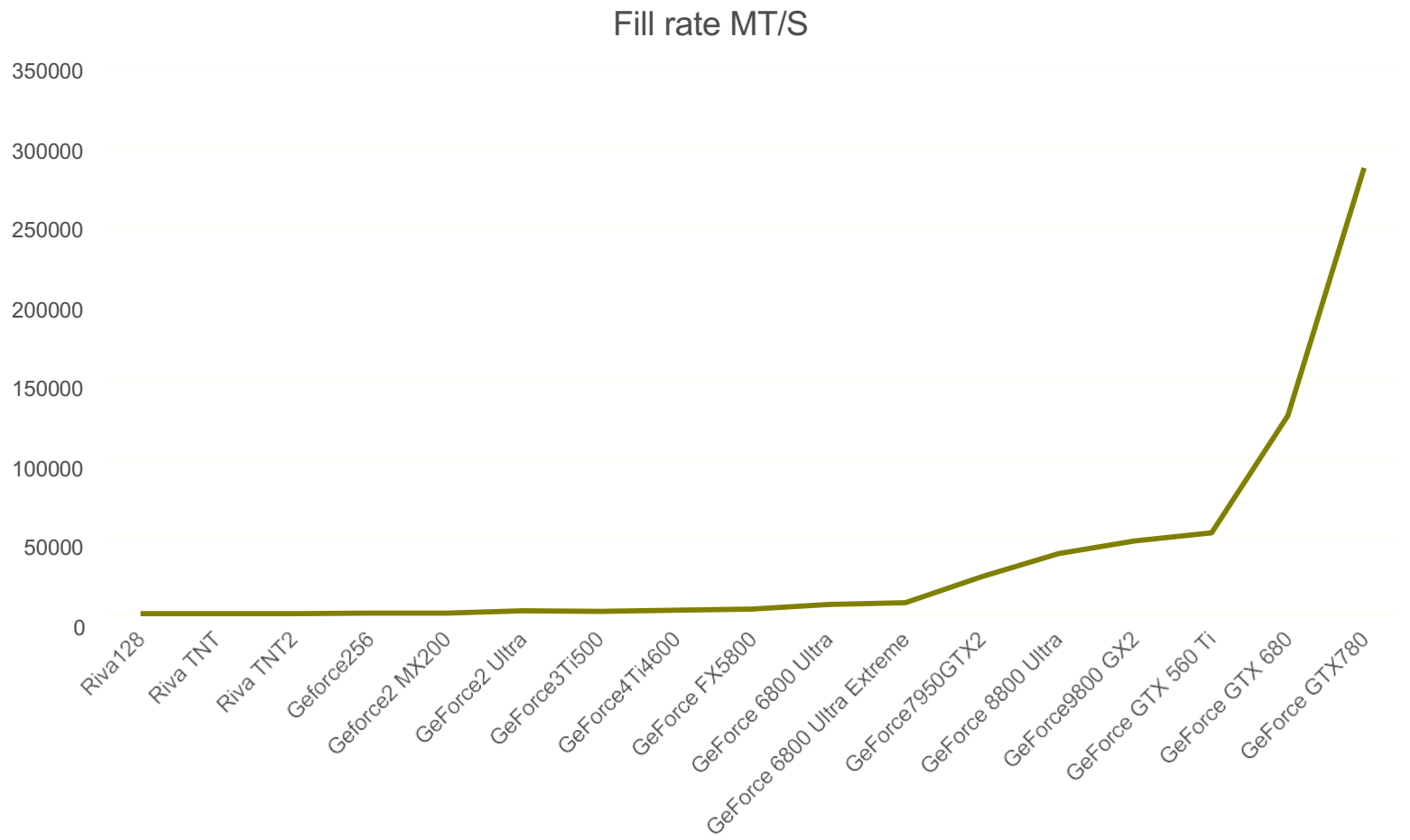
- Intel 3 Ghz
  - 12 GFLOPS peak (via SSE2)
- NVIDIA GeForce 6800
  - 45 GFLOPS peak performance

More recent

- Intel Core i9-7900X
  - 10 cores @ 3.3GHz (4.3GHz max)
  - 1.15 TFLOPS
- NVIDIA GeForce RTX 2080 Ti
  - 2944 Cores @ 1.55GHz
  - 13.45 TFLOPS



FILLRATE DEVELOPMENT

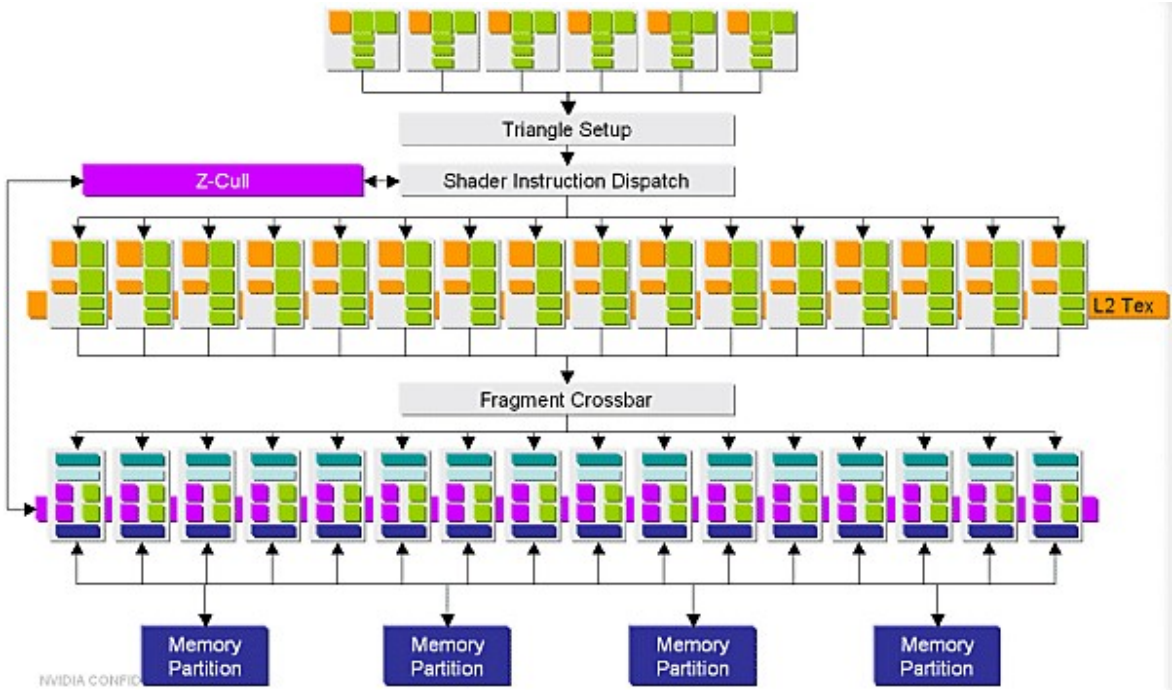


DATA FROM WIKIPEDIA

How GPUs are FAST!

What's really important these days is parallelism, massive parallelism

- Superscalar processing
- NVIDIA GeForce RTX 2080 Ti – 2944 Cores

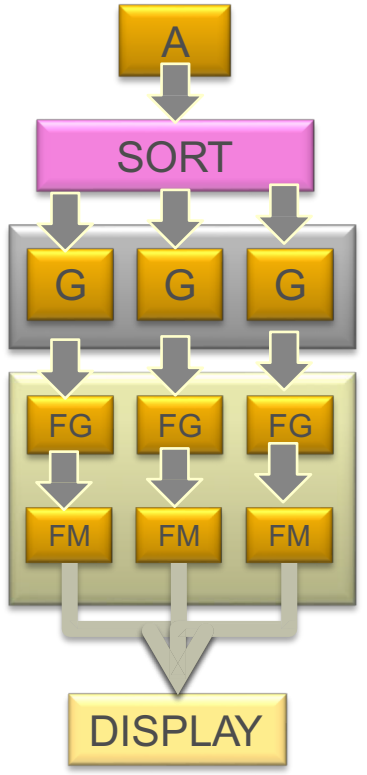


PARALLELIZATION OF RENDERING

Divide work into sub-tasks

Sorting schemas:

- Sort-first
- Sort-middle
- Sort-last fragment
- Sort-last image



A – Application
G – Geometry stage
FG – Fragment generator
FM – Fragment merging

SORT-FIRST

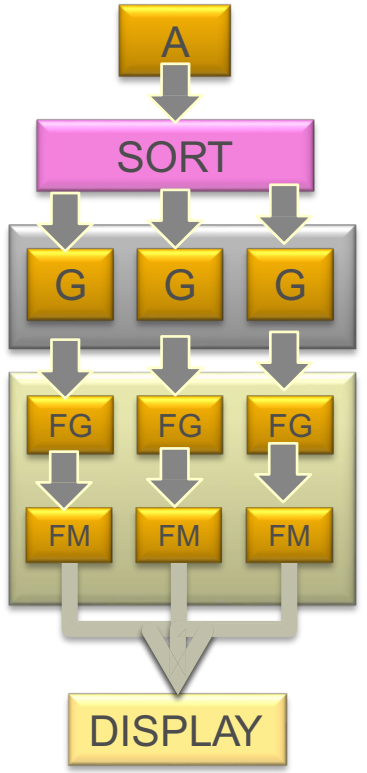
Sort primitives before geometry processing

Divide screen into tiles

Each process work on a tile in parallel, merge result

- Good use for multiscreen-tiled displays
- Used in SLI configurations

Disadvantage: pre-transform are required



A – Application
G – Geometry stage
FG – Fragment generator
FM – Fragment merging

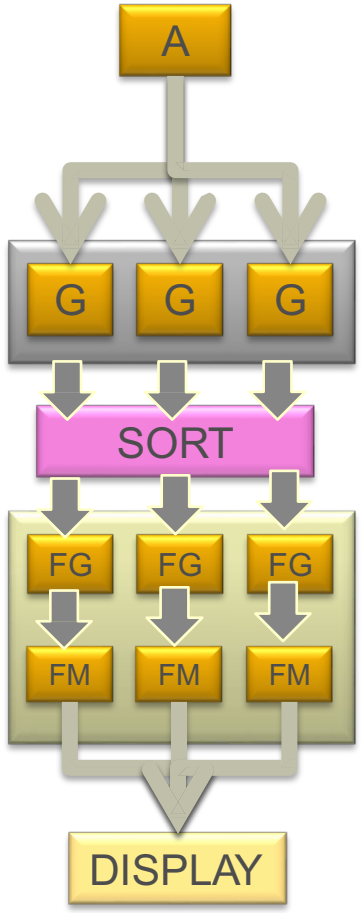
SORT-MIDDLE

Geometry processors are given work arbitrary—load balancing is the goal

Each rasterizer (FG+FM) is given a tile (part of display, scanline, group of pixels etc.)

Transformed primitives is then sorted into corresponding rasterizer

- Multiple rasterizers can get same primitive



A – Application
G – Geometry stage
FG – Fragment generator
FM – Fragment merging

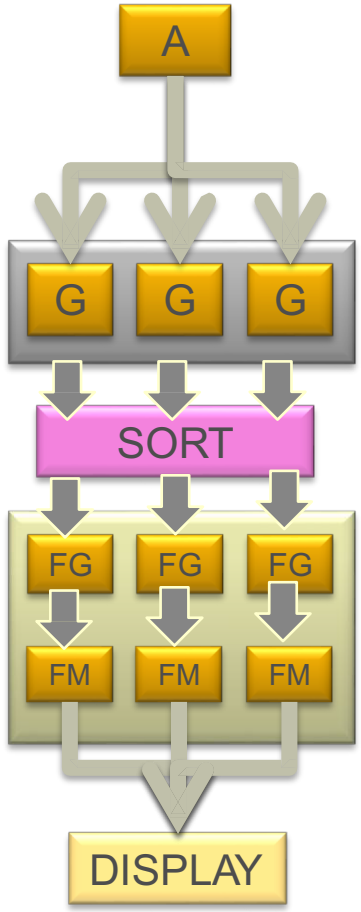
SORT-MIDDLE (CON'T)

Used in most mobile GPUs

- Minimize bandwidth requirement.
  - Need system memory support.
  - High memory bandwidth usage will hammer the whole system.
- Save more penguins (power).
  - Reducing memory access means less power consumption.

Disadvantage

- One triangle may go into multiple tiles.
- Need a sorting buffer after triangle sorting
  - More complex scene, more memory access.
- Completely divide graphic pipeline into two partition.
  - This may harm the performance.



A – Application
G – Geometry stage
FG – Fragment generator
FM – Fragment merging

SORT-LAST FRAGMENT

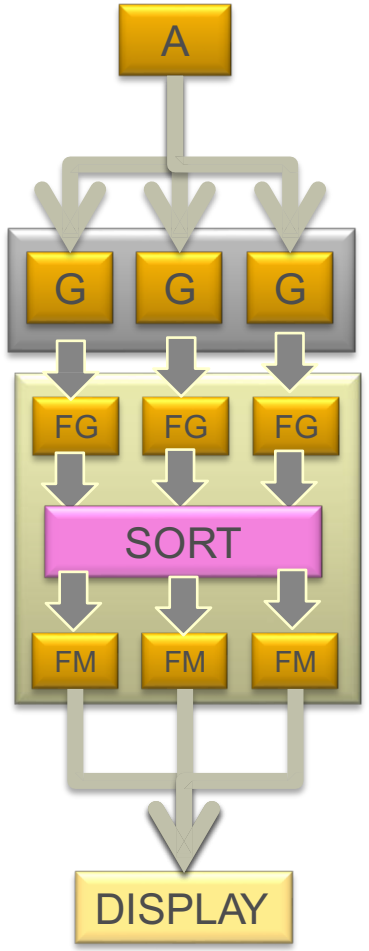
Sort fragments after FG before FM

Primitives distributed evenly over geometry units (G)

Fragment are generated, and then sorted into Fragment-merge (FM)

Imbalance can occur if a set of large triangles are sent to one FG

Used in original XBox



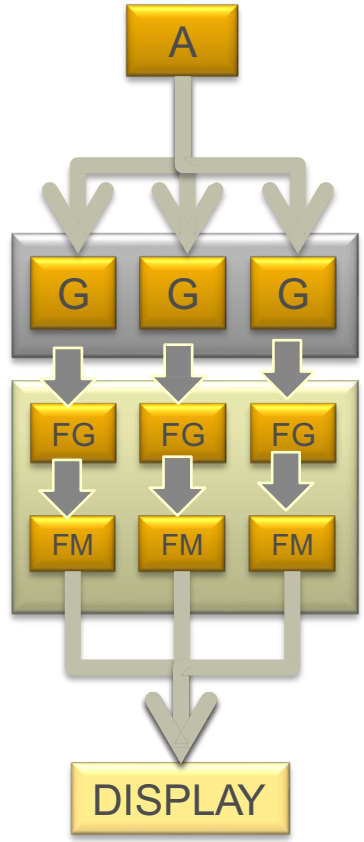
A – Application
G – Geometry stage
FG – Fragment generator
FM – Fragment merging

SORT-LAST IMAGE

Primitives split over separate pipelines, balancing

Complete rendering to z-depth

Final composite into image using depth.



A – Application
G – Geometry stage
FG – Fragment generator
FM – Fragment merging

SORT-LAST IMAGE (CON'T)

Used in most desktop GPUs

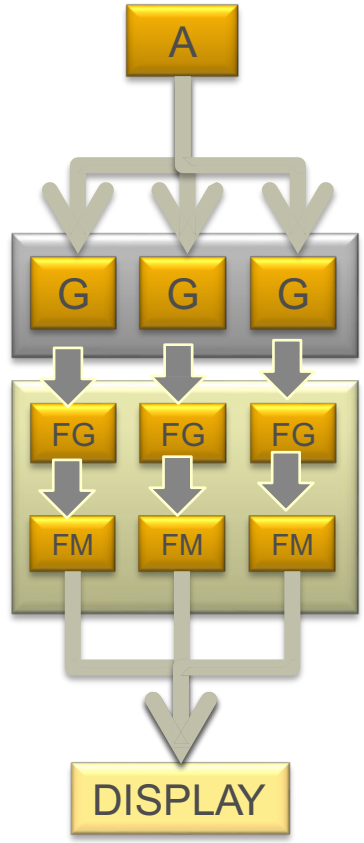
- Minimize performance issue
  - Sorting cost is low
- Desktop GPU has its own dedicated memory.
  - Graphics DRAM usually has high bandwidth with high latency.

Advantage

- Full rendering pipeline without interrupt until per-fragment operation (compare to sort-middle)

Disadvantage

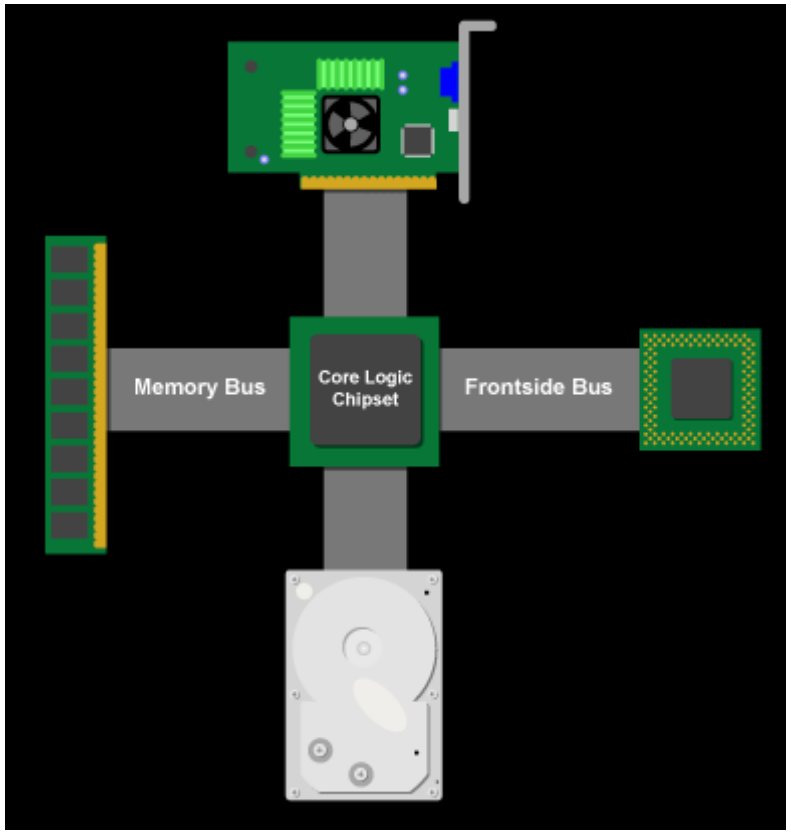
- Large amount of data to be sent
- Huge bandwidth requirement on frame buffer access, particularly in high resolution and anti-aliasing enable.



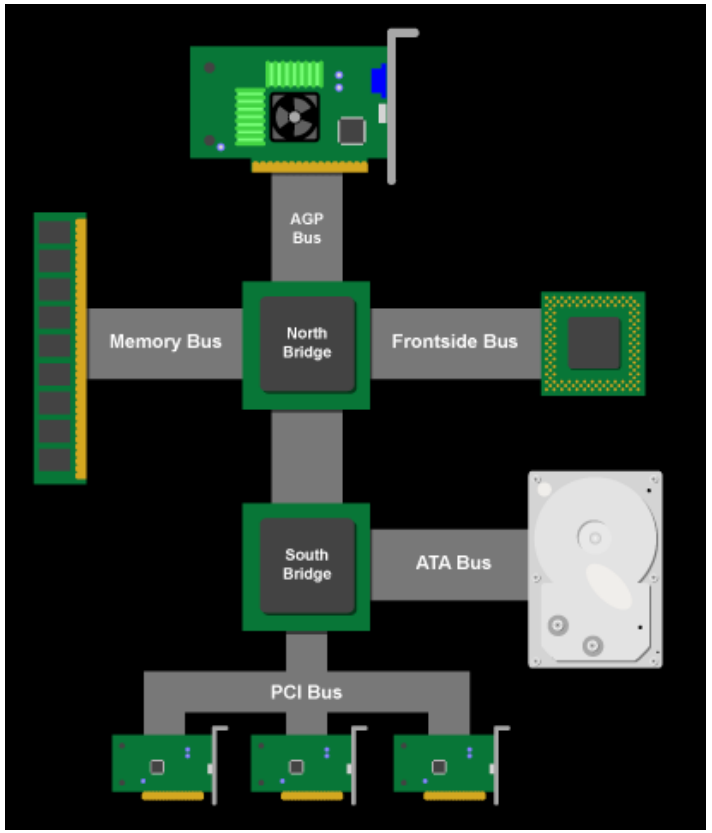
A – Application
G – Geometry stage
FG – Fragment generator
FM – Fragment merging

BUSSES: FEED THE BEAST

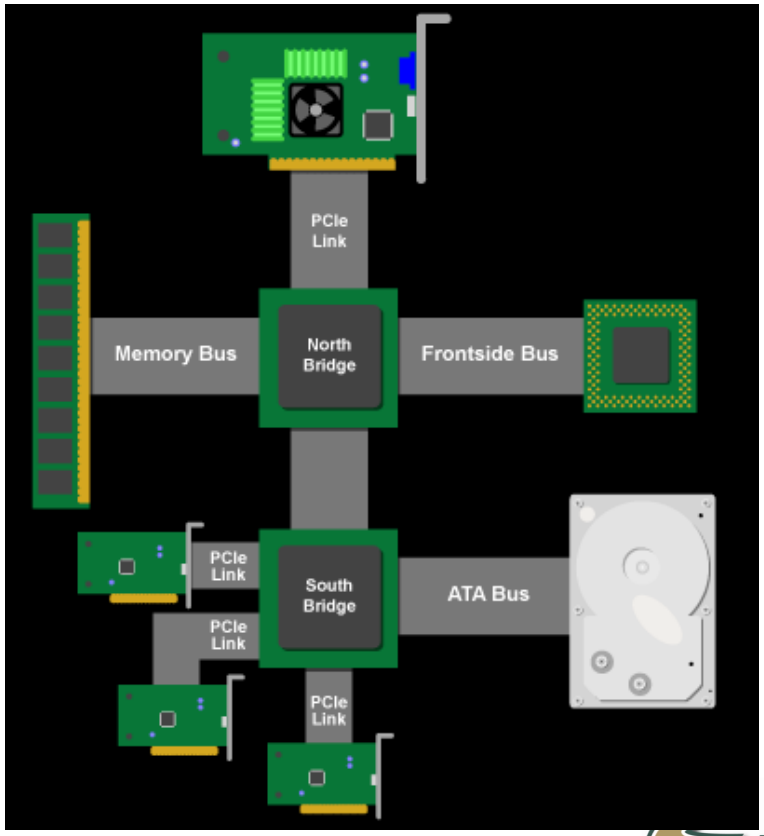
PCI based—one bus shared by all traffic including GPU



AGP Based—Dedicated bus for the graphics cards



PCI express—latest generation of dedicated bus



BUSSES

PCI, 1993
- 33MHz/32-bit PCI bus
- The whole bus can deliver: 133MB/second
- Usually peaks at: 100-110MB/sec

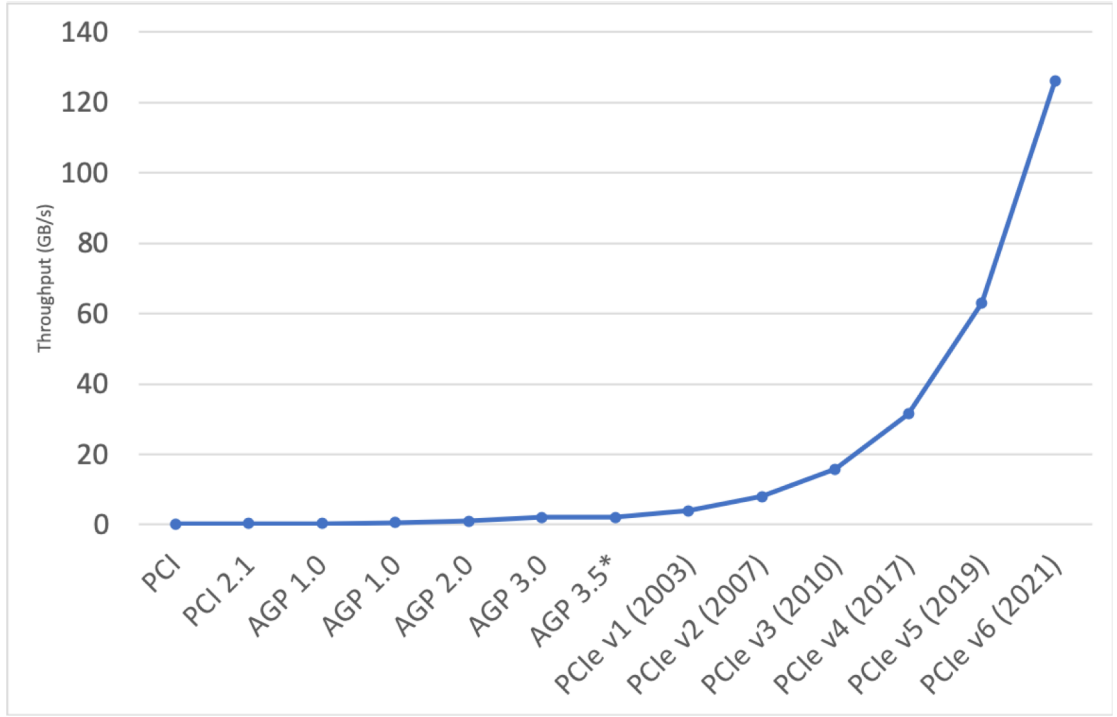
AGP 1.0, 1996
- 1x, 2x – 250MB/second; 32Bit
- Runs in full memory bus speed

AGP 2.0, 1998
- 4x – 266.67MHz, (4 x 66 MHz), 1066MB/second

AGP 3.0, 2002
- 8x – 533MHz, 2.1GB/Sec

PCI-Express, 2003
- 16x – 4GB/sec in each direction
- Doesn't share BW with other PCI-Express slots

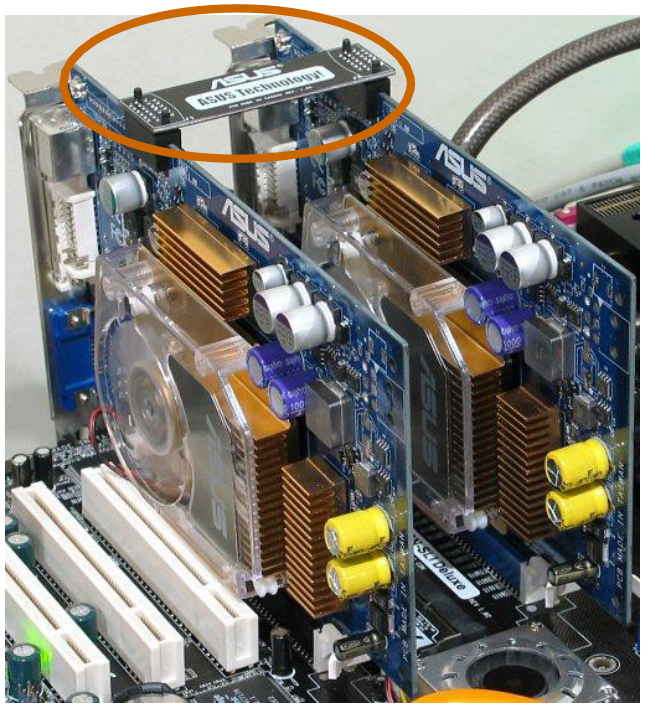
PCI-Express 6.0, 2021
- 16x – 126GB/sec

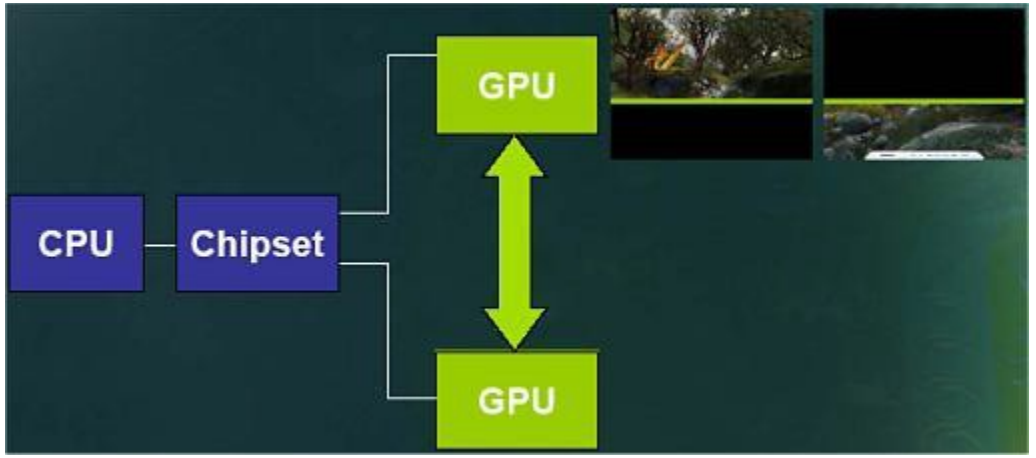


ADDITIONAL BUSSES

NVIDIA SLI

- Hooking up two or more cards so that they can work in parallel.
- Drivers distribute workload between the cards
- For example
  - GPU 0 renders frame 0, 2, 4, 6…
  - GPU 1 renders frame 1, 3, 5, 7…





KNOW YOUR ARCHITECTURE

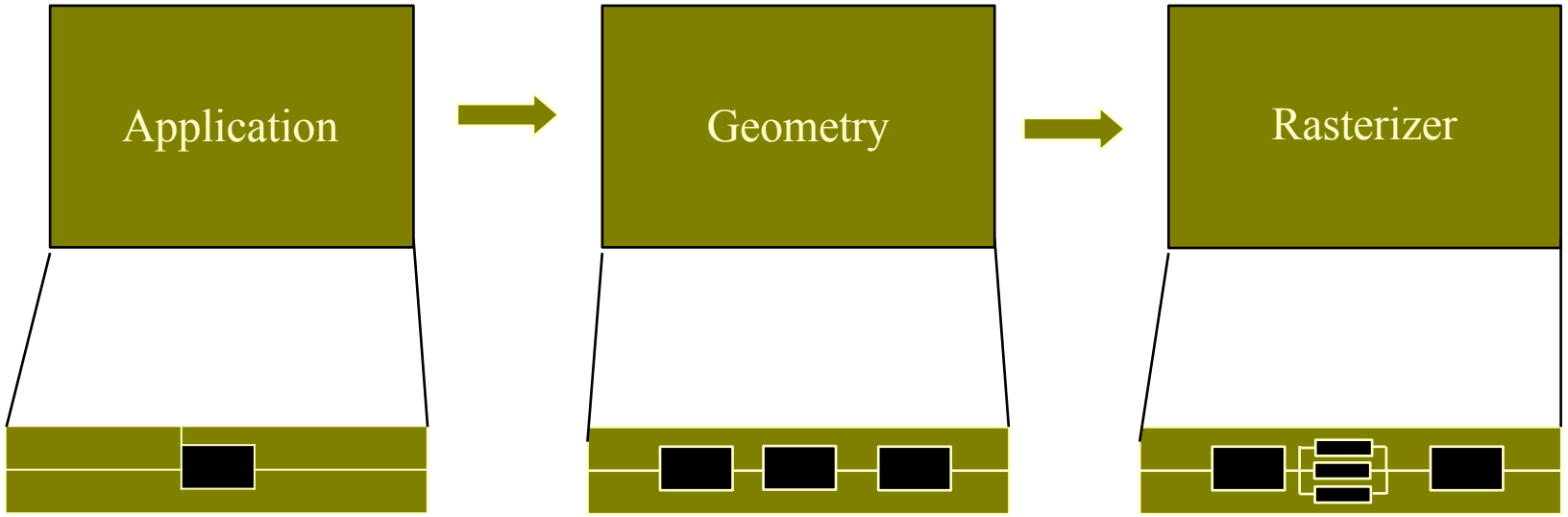
- Optimal performance is usually tied to one platform
- Knowing the target platform is very important

The Graphics Pipeline

We could look at the 3D rendering application as a pipeline

- Compare to a factory assembly
- The stages operate in parallel but are stalled until all stages are done
- We cannot execute faster than the slowest stage which is called the bottleneck
- The rendering process is conceptually built like a pipeline

The Graphics Pipeline



The Graphics Pipeline

Data will be pushed through the stages of the pipeline

The slowest stage will determine the rendering speed, expressed in frames per second.

Each stage can be further parallelized

- Compare to description of NVIDIA hardware

APPLICATION STAGE

User have full control over this stage as it is executed in software

- Input – User interaction, Simulations
- Output – Object transform, View transform, rendering primitives

Reads input on

- From devices
- From the user
- From the Network

The simulation that drives the application

- Physics, AI, collision detection, animation

Performance affected by

- Complexity of simulation, simulation size (Number of objects)
- Reading from slow devices, network etc...

GEOMETRY STAGE

Performance affected by

- Number of lights, geometry complexity, shader complexity, effect complexity, etc.

Find bottleneck by using a profiler and/or disabling various components

RASTERIZER

Output

- Pixels in frame buffer

Performance affected by

- Size of window
- Number of/size of textures

Find the bottleneck by

- Increase the window size, if performance drops -> fill limited application

Optimization

Extreme Programming

- Simple design, implement fast, test often, if need redesign, but test first!

Optimizing code is a hard task, but don't over-optimize

Locate the major bottlenecks, remove them, move on

OPTIMIZING

The bottleneck can move between the different pipeline stages.

Example, assume:

- Application stage: 30ms
- Geometry stage: 20ms
- Rasterizer: 40ms

The maximum framerate is then 1/40ms=25Hz

Optimizing the application stage won't improve framerate!

- Although, we could use the extra time at the geometry stage (20ms idle time) for doing more complex lighting equations

Optimizing

Some stages are hard to test individually, as they are implemented in specialized hardware.

What we have to do is to setup a number of tests, where each test only affects one stage at a time.

TESTING APPLICATION STAGE

Usually executed on CPU

If CPU ~100% we could be CPU limited

Locate the problem

- Use profiling tools
  - CodeAnalyst (AMD)
  - VTune Analyzer (INTEL)
- Send down data that causes the other stages to do little or no work, if CPU still 100%, the problem is in the Application stage

TESTING GEOMETRY

Add/remove lights

Try different kinds of lights

- Directional less computational intensive
- Point lights
- Spotlights - most demanding

If performance changes, we are transform-limited

If we are using programmable shaders, we can remove the lighting equation, to see if performance increases.

Testing Rasterizer

Size of rendered surface

Turn off blending and depth buffering

OVERALL OPTIMIZATION

Reduce the number of primitives

- Culling (later on)
- Reduce the complexity of the models
- Level of detail—render object only as accurate as necessary

Choose as low a precision as is possible

- Vertices, normals, texture coordinates (watch out for anomalies!)

Turn of features not used

- Fog, blending, lighting, transparency etc..

Preprocess the models

- Most hw prefers triangles and quads, not polygons (>4 corners)
- Tri-strips reduces data sent over the bus