

ECE4893A/CS4803MPG:

# MULTICORE AND GPU PROGRAMMING FOR VIDEO GAMES

GPUs – Under the Hood



Prof. Aaron Lanterman



School of Electrical and Computer Engineering

Georgia Institute of Technology



# Bandwidth – Gravity of Modern Computer Systems

- The bandwidth between key components ultimately dictates system performance
  - Especially true for massively parallel systems processing massive amount of data
  - Tricks like buffering, reordering, caching can temporarily defy the rules in some cases
  - Ultimately, the performance goes falls back to what the “speeds and feeds” dictate

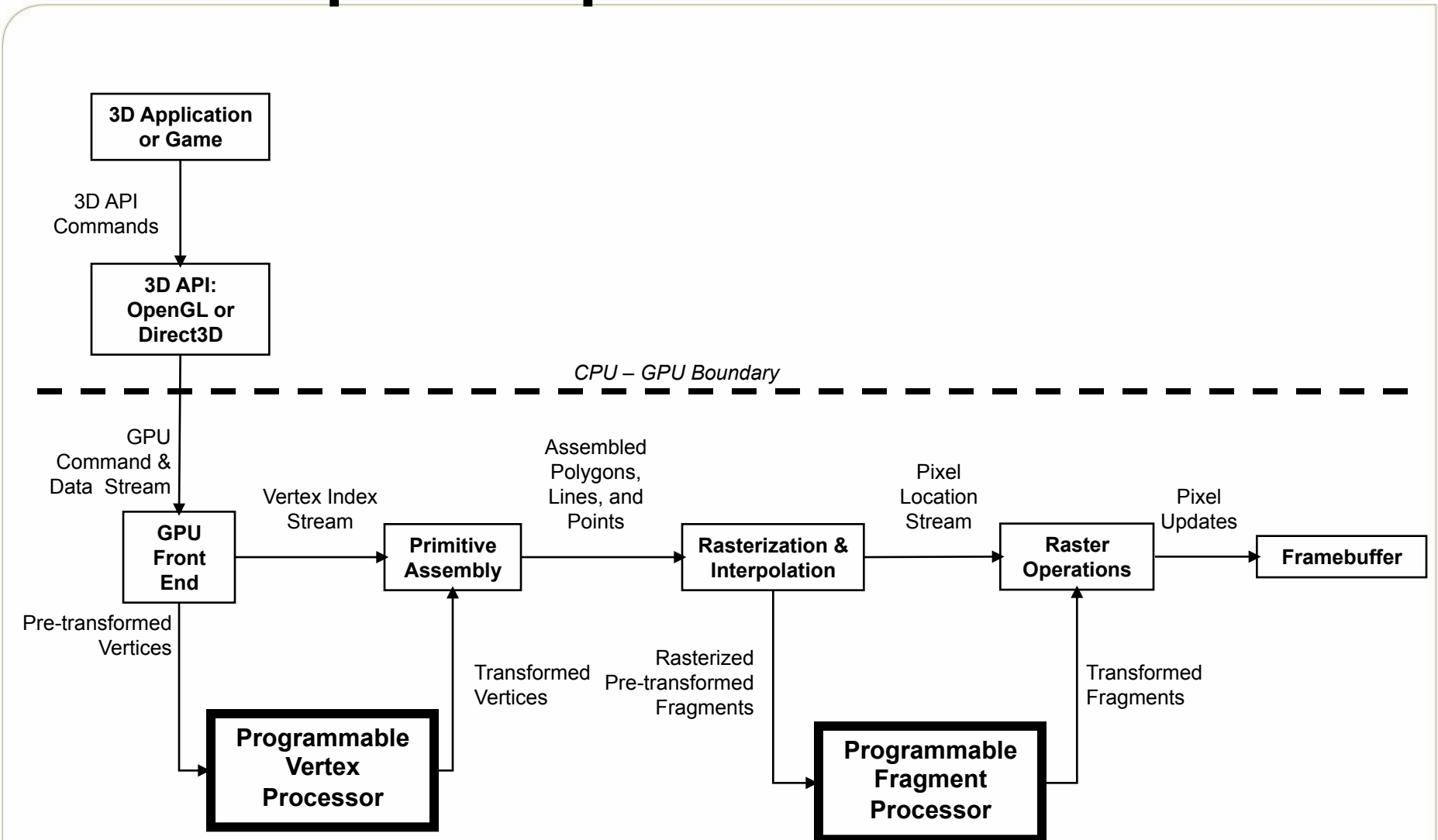
# Interface “feeds and speeds”

- **AGP: Advanced Graphics Port** – an interface between the computer core logic and the graphics processor
  - AGP 1x: 266 MB/sec – twice as fast as PCI
  - AGP 2x: 533 MB/sec
  - AGP 4x: 1 GB/sec → AGP 8x: 2 GB/sec
  - 256 MB/sec readback from graphics to system
- **PCI-E: PCI Express** – a faster interface between the computer core logic and the graphics processor
  - PCI-E 1.0: 4 GB/sec each way → 8 GB/sec total
  - PCI-E 2.0: 8 GB/sec each way → 16 GB/sec total

# 3D Buzzwords

- Fill Rate – how fast the GPU can generate pixels, often a strong predictor for application frame rate
- Performance Metrics
  - Mtris/sec - Triangle Rate
  - Mverts/sec - Vertex Rate
  - Mpixels/sec - Pixel Fill (Write) Rate
  - Mtexels/sec - Texture Fill (Read) Rate
  - Msamples/sec - Antialiasing Fill (Write) Rate

# Adding Programmability to the Graphics Pipeline



Slide by David Kirk/NVIDIA and Wen-mei. W. Hwu, 2007,  
from UIUC ECE498 Lecture 5, Fall 2007; used with permission  
See [courses.ece.uiuc.edu/ece498/a1](http://courses.ece.uiuc.edu/ece498/a1)

# Specialized Instructions (GeForce 6)

- Dot products
- Exponential instructions:
  - EXP, EXPP, LOG, LOGP
  - LIT (Blinn specular lighting model calculation!)
- Reciprocal instructions:
  - RCP (reciprocal)
  - RSQ (reciprocal square root!)
- Trigonometric functions
  - SIN, COS
- Swizzling (swapping xyzw), write masking (only some xyzw get assigned), and negation is “free”

# Easy cross products and normalization

## Vector Cross Product

```
# | i      j      k      | into R2.
# | R0.x   R0.y   R0.z   |
# | R1.x   R1.y   R1.z   |
MUL R2, R0.zxyw, R1.yzxw; // swizzle
MAD R2, R0.yzxw, R1.zxyw, -R2; // negation
```

## Vector Normalize

```
# R1 = (nx,ny,nz)
#
# R0.xyz = normalize(R1)
# R0.w   = 1/sqrt(nx*nx + ny*ny + nz*nz)
DP3 R0.w, R1, R1;
RSQ R0.w, R0.w; // write-mask
MUL R0.xyz, R1, R0.w; // promotion
```

CS448 Lecture 12

Kurt Akeley, Pat Hanrahan, Fall 2001

# Blinn lighting in one instruction

LIT d, s

$$s.x = N \cdot L$$

$$s.y = N \cdot H$$

$$s.z = s$$

(-128 < m < 128)

$$d.x = 1.0$$

$$d.y = \text{CLAMP}(N \cdot L, 0, 1)$$

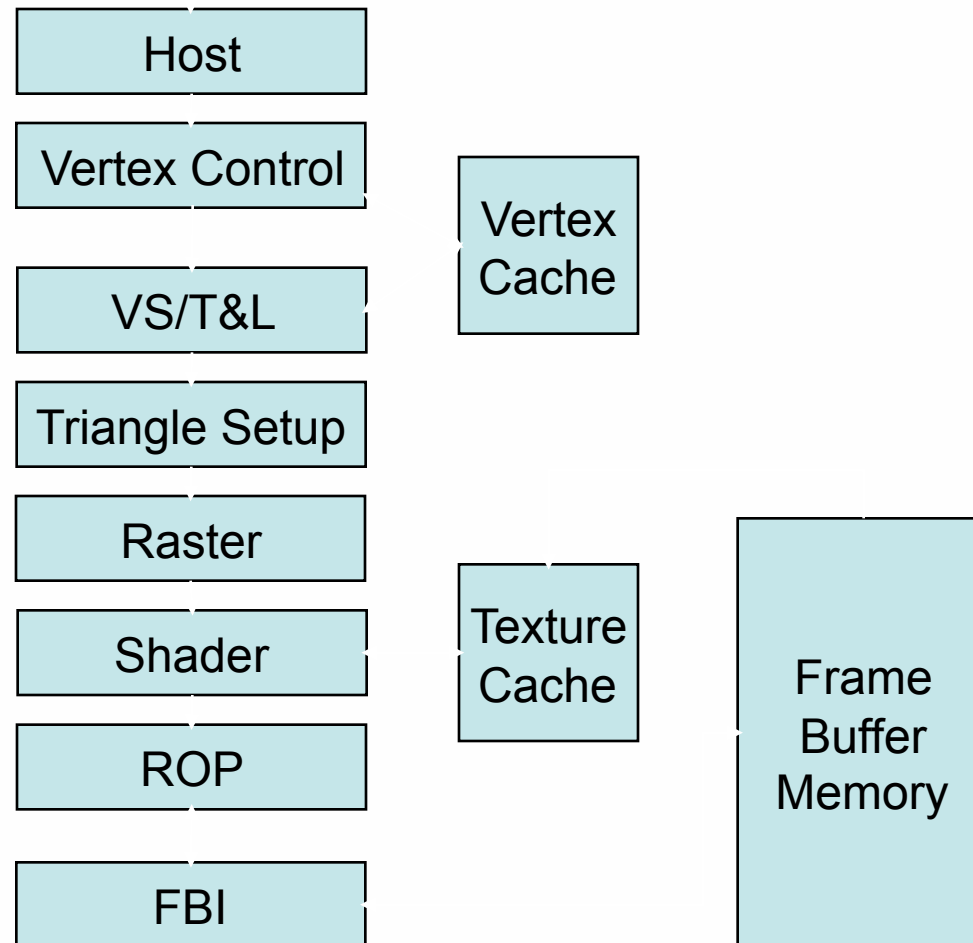
$$d.z = \text{CLAMP}(N \cdot H, 0, 1)^s$$

$$d.w = 1.0$$

# Simple graphics pipeline

```
# c[0-3] = Mat; c[4-7] = Mat^{-T}
# c[32] = L; c[33] = H
# c[35].x = Md * Ld; c[35].y = Ma * La
# c[36] = Ms; c[38].x = s
DP4 o[HPOS].x, c[0], v[OPOS]; # Transform position.
DP4 o[HPOS].y, c[1], v[OPOS];
DP4 o[HPOS].z, c[2], v[OPOS];
DP4 o[HPOS].w, c[3], v[OPOS];
DP3 R0.x, c[4], v[NRML]; # Transform normal.
DP3 R0.y, c[5], v[NRML];
DP3 R0.z, c[6], v[NRML];
DP3 R1.x, c[32], R0; # R1.x = L DOT N
DP3 R1.y, c[33], R0; # R1.y = H DOT N
MOV R1.w, c[38].x; # R1.w = s
LIT R2, R1; # Compute lighting
MAD R3, c[35].x, R2.y, c[35].y; # diffuse + ambient
MAD o[COL0].xyz, c[36], R2.z, R3; # + specular
END
```

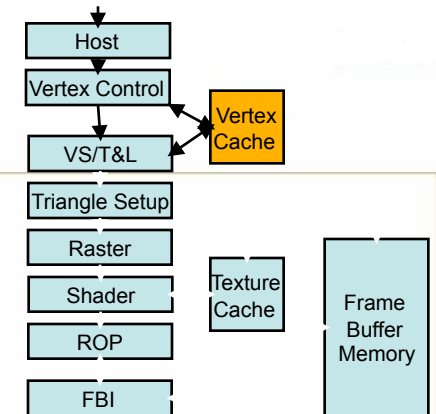
# The GeForce Graphics Pipeline



Slide by David Kirk/NVIDIA and Wen-mei. W. Hwu, 2007,  
from UIUC ECE498 Lecture 5, Fall 2007; used with permission  
See [courses.ece.uiuc.edu/ece498/al1](http://courses.ece.uiuc.edu/ece498/al1)

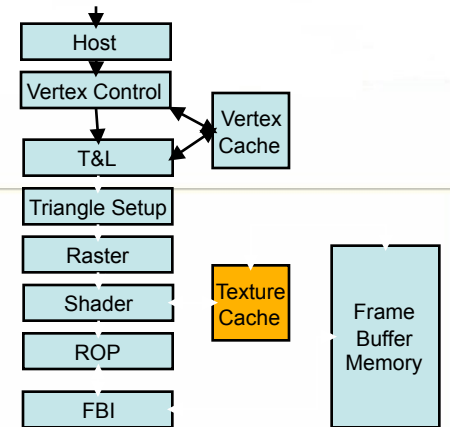
# Vertex Cache

- Temporary store for vertices, used to gain higher efficiency
- Re-using vertices between primitives saves AGP/PCI-E bus bandwidth
- Re-using vertices between primitives saves GPU computational resources
- A vertex cache attempts to exploit “commonality” between triangles to generate vertex reuse
- Unfortunately, many applications do not use efficient triangular ordering



# Texture Cache

- Stores temporally local texel values to reduce bandwidth requirements
- Due to nature of texture filtering high degrees of efficiency are possible
- Efficient texture caches can achieve 75% or better hit rates
- Reduces texture (memory) bandwidth by a factor of four for bilinear filtering

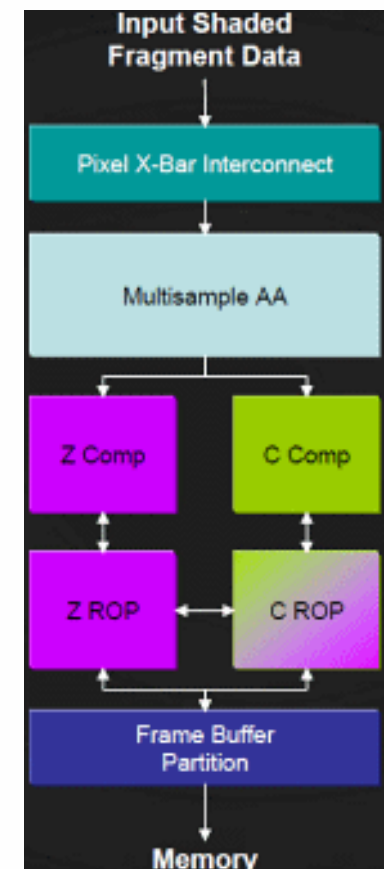
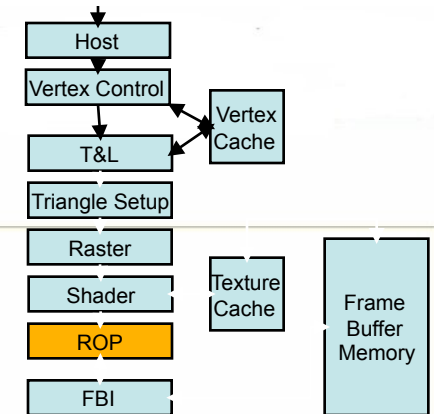


# Built-in Texture Filtering (GeForce 6)

- Pixel texturing
  - Hardware supports 2D, 3D, and cube map
  - Non power-of-2 textures OK
  - Hardware handles addressing and interpolation for you
    - Bilinear, trilinear (3D or mipmap), anisotropic
- Vertex texturing
  - Vertex processors can access texture memory too
  - Only nearest-neighbor filtering supported in G60 hardware

# ROP (from Raster Operations)

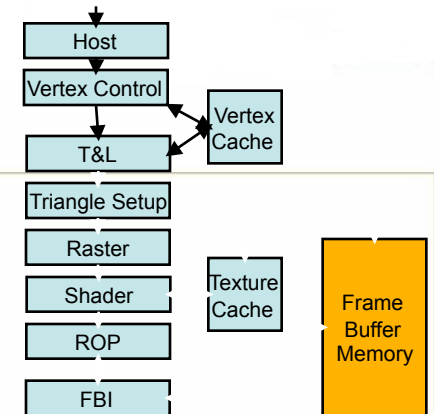
- C-ROP performs frame buffer blending
  - Combinations of colors and transparency
  - Antialiasing
  - Read/Modify/Write the Color Buffer
- Z-ROP performs the Z operations
  - Determine the visible pixels
  - Discard the occluded pixels
  - Read/Modify/Write the Z-Buffer
- ROP on GeForce also performs
  - “Coalescing” of transactions
  - Z-Buffer compression/decompression



Slide by David Kirk/NVIDIA and Wen-mei. W. Hwu, 2007,  
from UIUC ECE498 Lecture 5, Fall 2007; used with permission  
See [courses.ece.uiuc.edu/ece498/al1](http://courses.ece.uiuc.edu/ece498/al1)

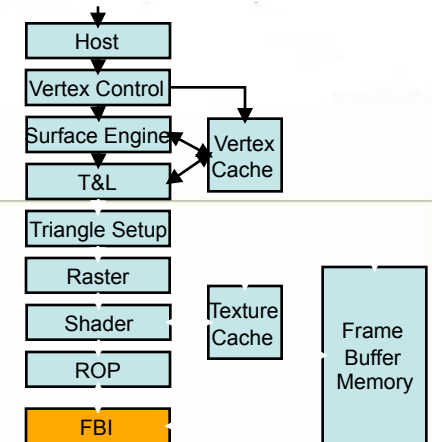
# The Frame Buffer

- The primary determinant of graphics performance other than the GPU
- The most expensive component of a graphics product other than the GPU
- Memory bandwidth is the key
- Frame buffer size also determines
  - Local texture storage
  - Maximum resolutions
  - Antialiasing resolution limits

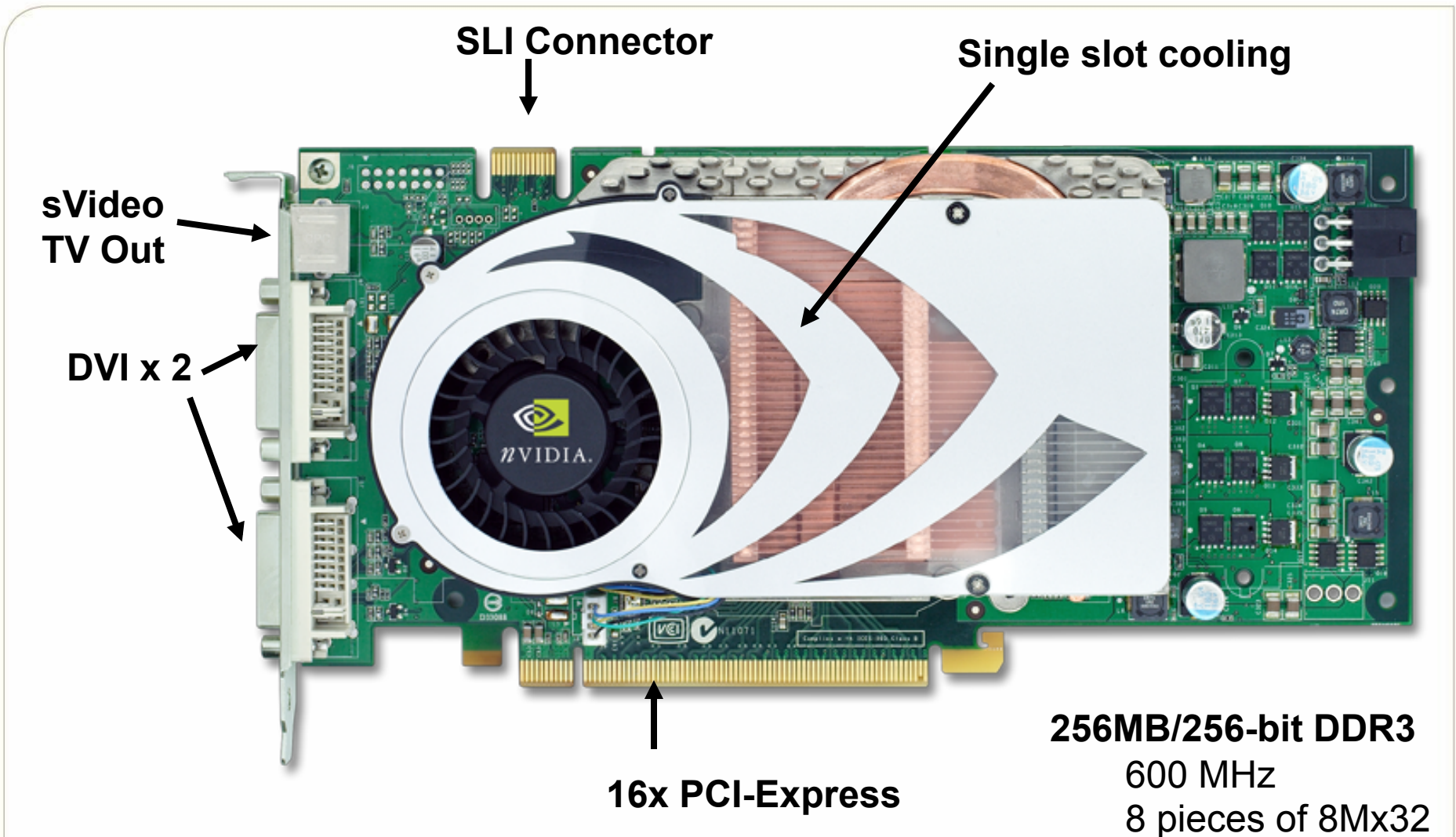


# Frame Buffer Interface (FBI)

- Manages reading from and writing to frame buffer
- Perhaps the most performance-critical component of a GPU
- GeForce's FBI is a crossbar
- Independent memory controllers for 4+ independent memory banks for more efficient access to frame buffer



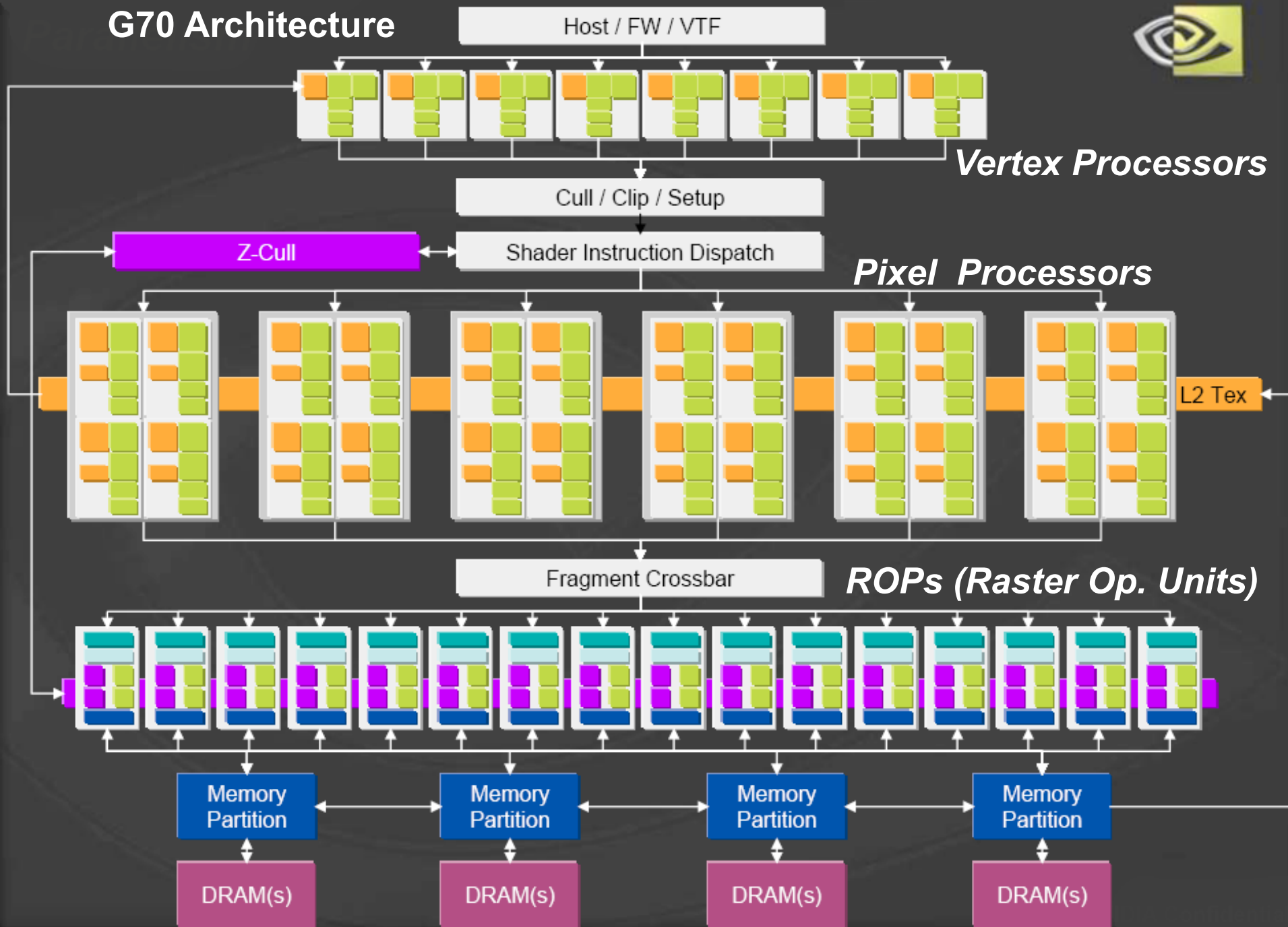
# GeForce 7800 GTX Board Details



Slide by David Kirk/NVIDIA and Wen-mei. W. Hwu, 2007,  
from UIUC ECE498 Lecture 6, Fall 2007; used with permission  
See [courses.ece.uiuc.edu/ece498/al1](http://courses.ece.uiuc.edu/ece498/al1)



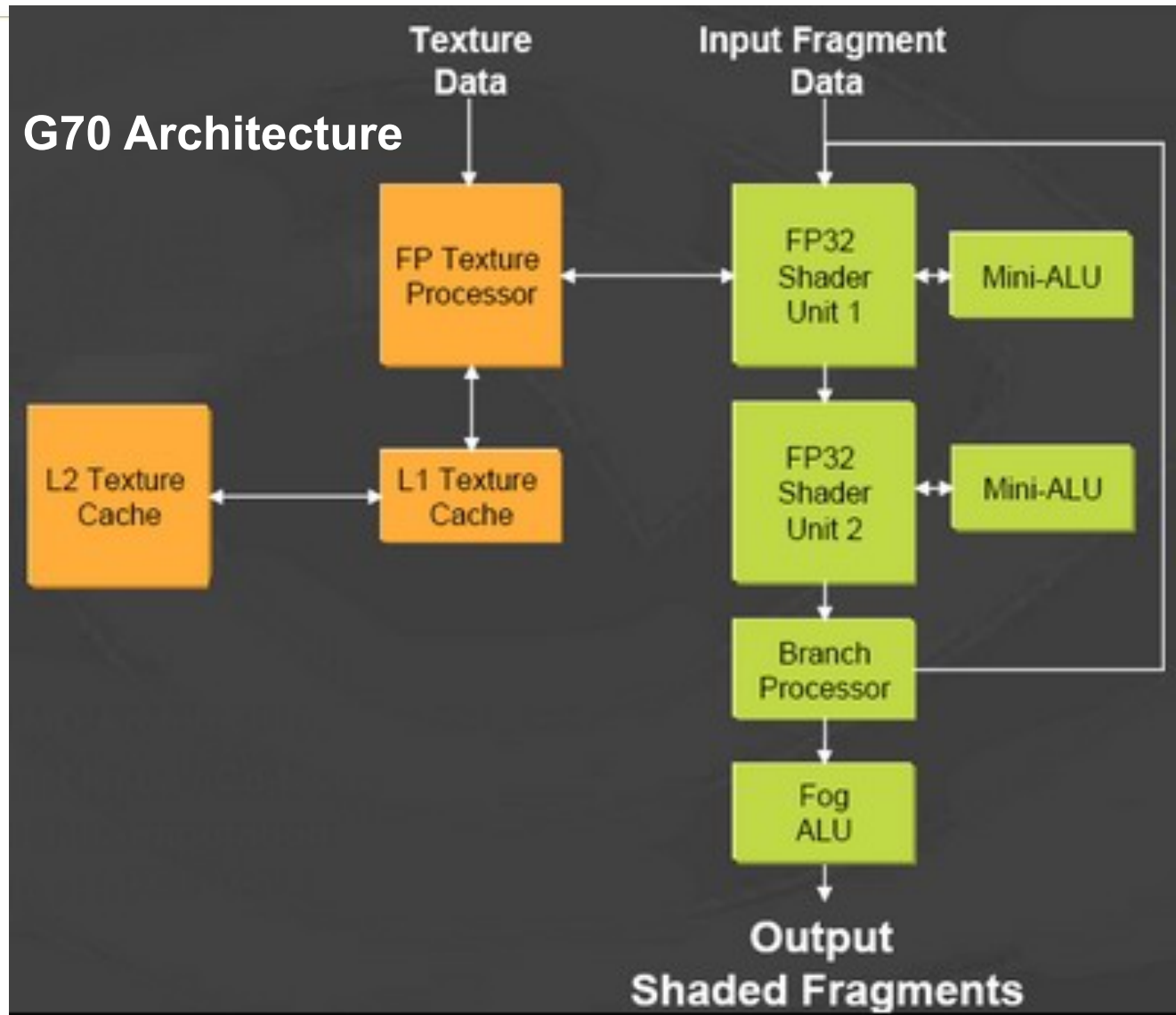
# G70 Architecture



# NVIDIA 7800 GTX - Pixel Processors

8 MADD  
(multiply/add)  
instructions in  
a single cycle

7800 GTX  
has 24 of  
these!



# NVIDIA 7800 GTX - Vertex Processors

7800 GTX has 8 of these!

