



**CS 563 Advanced Topics in
Computer Graphics**
Chapter 15: Graphics Hardware

by Dan Adams

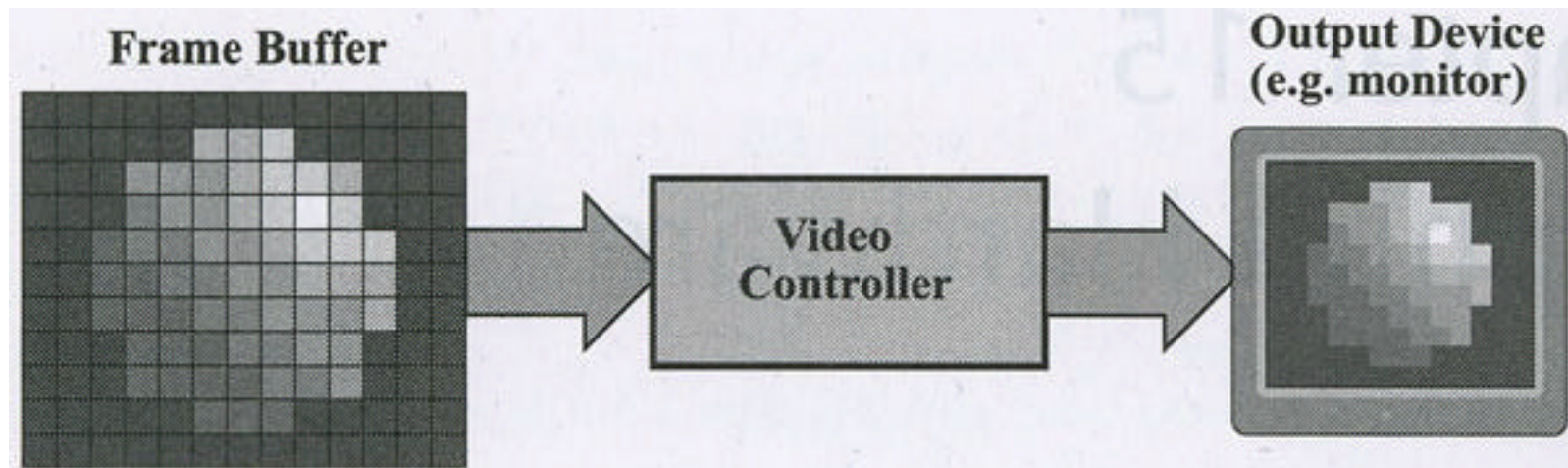


Topics

- Buffers
 - Color, Z, W, Stereo, Stencil, and Accumulation Buffers
- Hardware Architecture
 - Pipelining and Parallelization
 - Implementing the Stages in Hardware
 - Memory and Bandwidth
- Case Studies
 - Xbox
 - InfiniteReality
 - KYRO

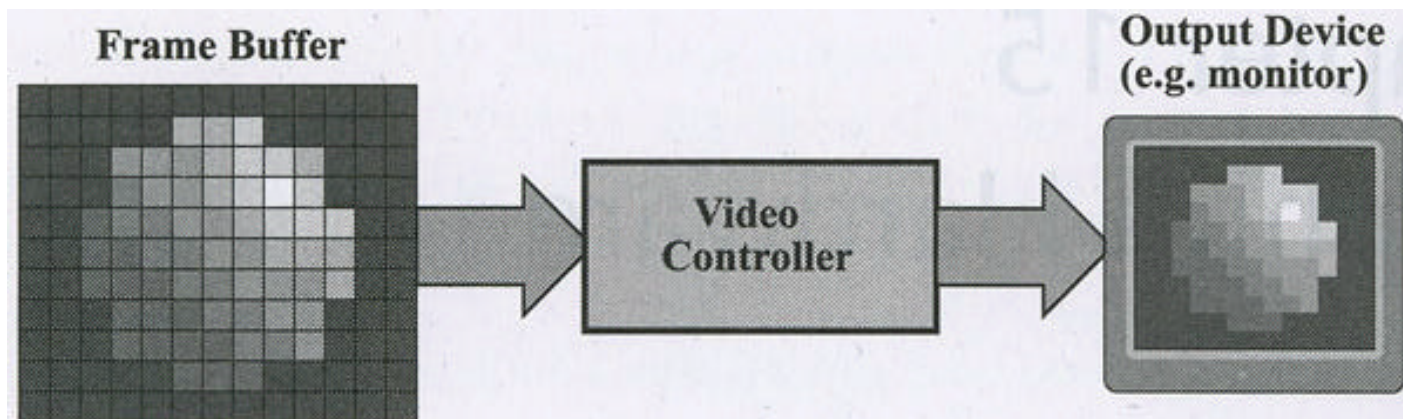
Display Basics

- Frame Buffer
 - Can be in host memory, dedicated memory, or memory shared by buffers and textures
 - Connected to a video Controller



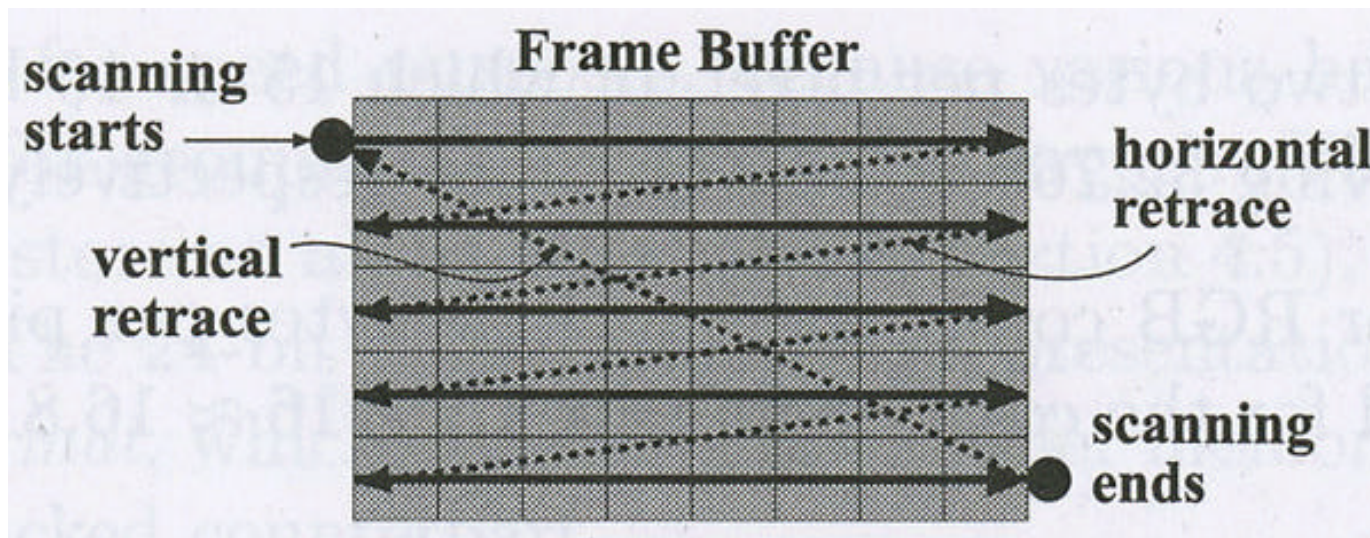
Display Basics

- Video Controller
 - AKA Digital-to-Analog Converter (DAC)
 - Converts digital pixel values to analog signals for the monitor
 - Monitor has a fixed refresh rate (60 to 120 Hz)
 - Sends color data to monitor in sync with the monitor beam



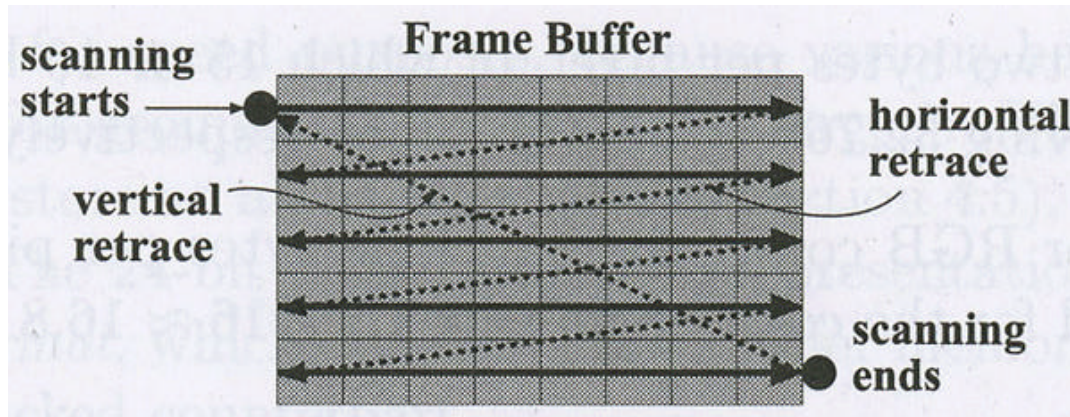
Display Basics

- Monitor beams moves left to right, top to bottom
- Horizontal retrace
 - Beam moves from end of one line to the beginning of the next
 - Does not actually set any colors



Display Basics

- Horizontal refresh rate (aka line rate)
 - Rate at which it can draw a line
- Vertical retrace
 - Returns to top left after the entire screen is drawn
- Vertical refresh rate
 - How many times per second it can refresh entire screen
 - Noticeable < 72 Hz by most people



Display Example

- Example:
 - 1280x1024 screen with 75 Hz refresh rate
 - Updates every 13.3 ms ($1 / 75 = 0.0133\text{s}$)
 - Screen has a specified “frame size” which is 1688 x 1066 for this resolution
 - Pixel clock – rate at which pixels are refreshed
 - $1688 * 1066 * 75 \text{ Hz} = 134,955,600 \text{ Hz} = 134 \text{ Mhz}$
 - $\text{rate} = 1 / 134 \text{ Mhz} = 7.410\text{e}9 = 7.4 \text{ nanoseconds}$
 - Line rate
 - $1066 * 75 \text{ Hz} = 79,950 \text{ lines/sec}$
 - Vertical retrace rate
 - $1066 - 1024 = 42$
 - $42 * 1688 * 7.4 \text{ ns} = 525 \text{ ns}$



Monitor Types

- Noninterlaced (aka progressive scan)
 - Most common for computer monitors
- Interlaced
 - Found in TVs
 - Horizontal lines are interlaced (evens first, then odds)
- Converting is non-trivial

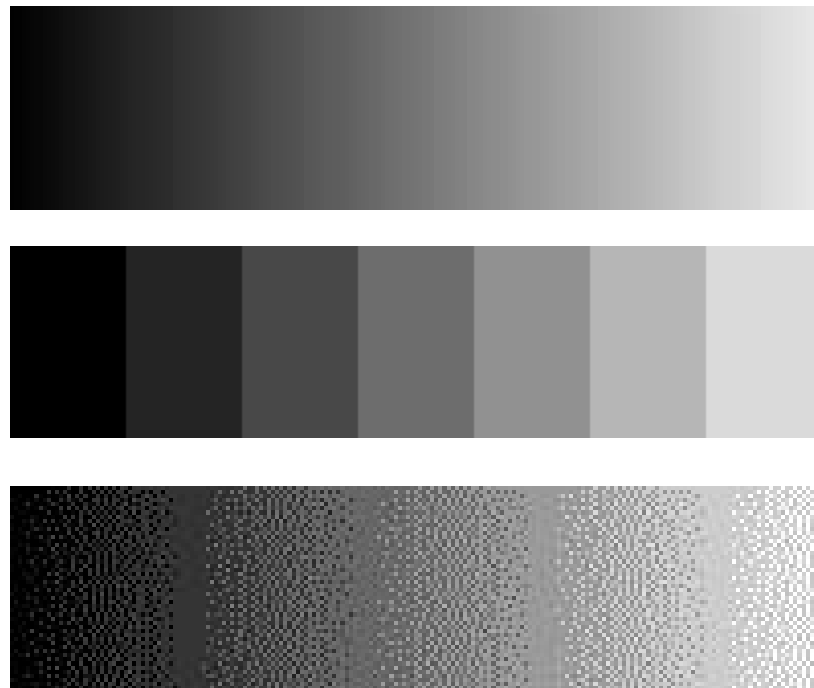


Color Modes

- High Color
 - 2 bytes per pixel (15 or 16 bits for color)
 - either 32,768 or 65,536 colors
 - Uneven division between colors ($16 / 3 = 5.3$ pixels per color?)
 - Green given an extra bit because it has more effect on the eye
 - $32 * 64 * 32 = 65,536$
 - Extra bit used not used or used for an alpha channel
 - $32 * 32 * 32 = 32,768$

Color Modes

- Can cause a Mach banding effect
 - Differences in color are noticeable by the eye
 - Similar problem with Gouraud shading
 - Can use dithering to lessen effect





Color Modes

- True Color
 - 3 or 4 bytes per pixel (24 bits for color)
 - 16.8 million colors
 - 8 bits per color
 - $255 * 255 * 255 = 16,581,385$ colors
 - 24 bit format is called the “packed pixel” format
 - Saves frame buffer space
 - 32 bit format
 - Some hardware optimized for groups of 4 bytes
 - Extra 8 bits can be used for an alpha
 - Using 24 bits corrects some problems found with high color
 - Quantization effects due to low precision (eg when using multipass)



Z-buffer

- Normally stores 24 bits / pixel
- Orthographic viewing
 - Depth resolution is uniform
 - Ex:
 - Near and far planes are 100 meters apart
 - Z-buffer stores 16 bits per pixel
 - $100 \text{ meters} / 2^{16} = \text{about } 1.5 \text{ mm}$
- Perspective viewing
 - Depth resolution is non-uniform
 - Farther away you go, more precision you need to be accurate
 - Can create artifacts or popping effects



Z-buffer

- After applying perspective transformation to a point we get a vector v :
 - $v = (x, y, z, w)$
- v then divided by w so that $v = (x/w, y/w, z/w, 1)$
- z/w is mapped to the range $[0, 2^b - 1]$ and stored in Z-buffer
- The farther away the point, the smaller z/w is after being mapped, and the less precision it has.

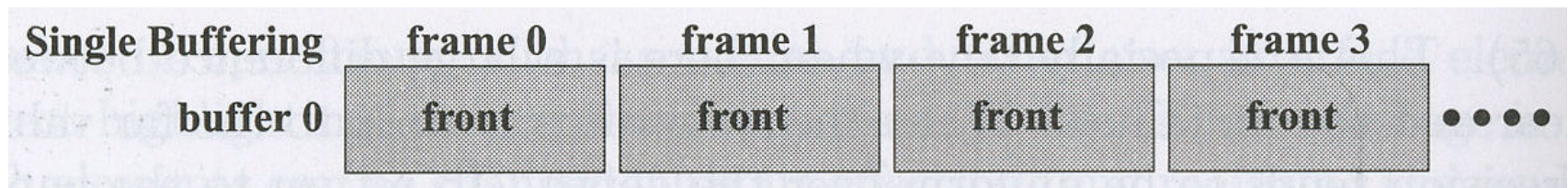


W-buffer

- Alternate to storing a Z-buffer
- Stores the w value
- Results in uniform precision
- Don't have to fool around with the near and far planes
- Becoming deprecated... (?)

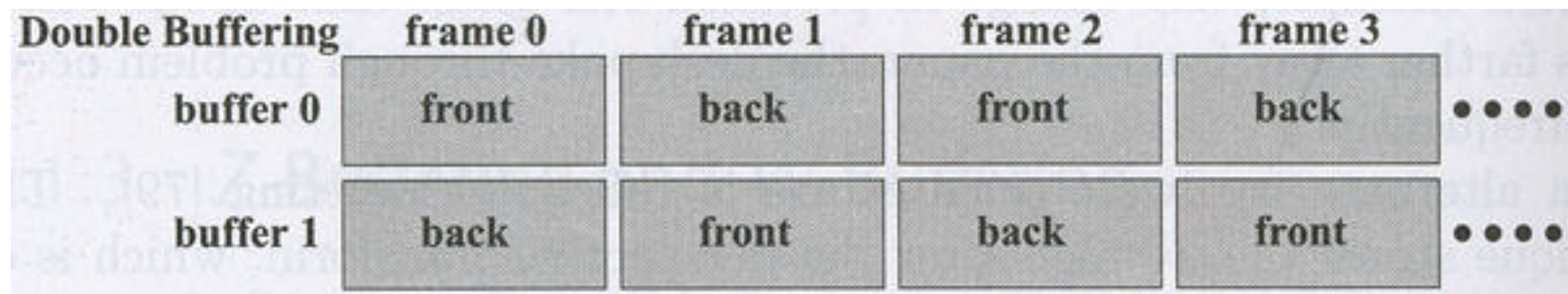
Single Buffering

- Only uses one buffer which is used for drawing
- You can see primitives being drawn on the screen
- Can result in “tearing”
 - User can see part of a primitive as its being drawn
- Not very useful for real-time graphics
- Can be used if you don't update very often
 - Windows in a workspace



Double Buffering

- Overcomes problems with single buffering
- Front buffer is display while the back buffer is drawn to
- Buffers are swapped during vertical retrace
 - Avoids tearing but is not required



Double Buffering

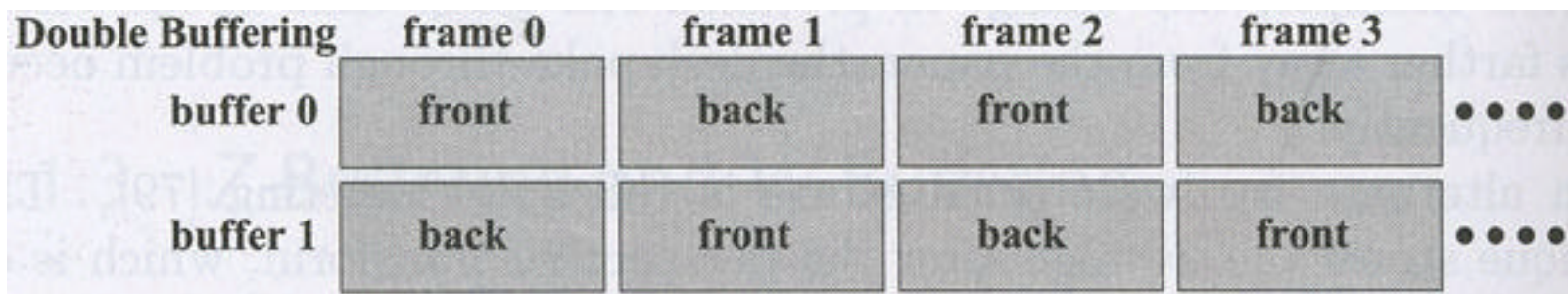
- Swapping methods

- Page flipping

- Address of front buffer is stored in a register. Points to (0,0)
 - Swap by writing the address of the back buffer to the register
 - Easy method for doing screen panning

- Blitting (or BLT swapping)

- Back buffer is simply copied over the front buffer



Triple Buffering

- Adds a second back buffer, the pending buffer
- Do not have to wait for vertical retrace to start drawing the next frame
- Do not have to wait for the back buffer to be cleared

Triple Buffering	frame 0	frame 1	frame 2	frame 3	...
buffer 0	pending	back	front	pending	••••
buffer 1	front	pending	back	front	••••
buffer 2	back	front	pending	back	••••

Triple Buffering

- Can increase frame rate:
 - Monitor is 60 Hz
 - If image generation $< 1/60^{\text{th}}$ sec then double and triple buffering will get 60 fps
 - If it takes $> 1/60^{\text{th}}$ sec, double buffering gets 30 fps while triple gets (almost) 60 fps

Triple Buffering	frame 0	frame 1	frame 2	frame 3	...
buffer 0	pending	back	front	pending	••••
buffer 1	front	pending	back	front	••••
buffer 2	back	front	pending	back	••••

Triple Buffering

- Increases latency
 - Response time for user is basically 2 frames behind
- Pending buffer requires another color buffer
- Works well if the hardware supports it

Triple Buffering	frame 0	frame 1	frame 2	frame 3	...
buffer 0	pending	back	front	pending	••••
buffer 1	front	pending	back	front	••••
buffer 2	back	front	pending	back	••••

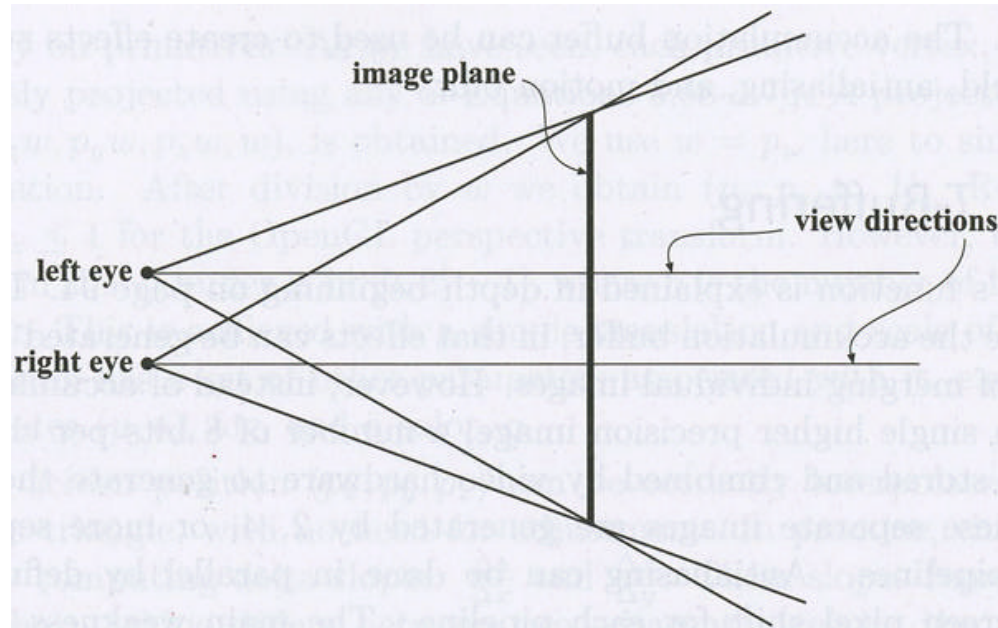
Triple Buffering

- DirectX supports it, OpenGL does not
- Can (in theory) use as many buffers as you want...
 - Good when image generation time varies a lot
 - Increases latency

Triple Buffering	frame 0	frame 1	frame 2	frame 3	...
buffer 0	pending	back	front	pending	••••
buffer 1	front	pending	back	front	••••
buffer 2	back	front	pending	back	••••

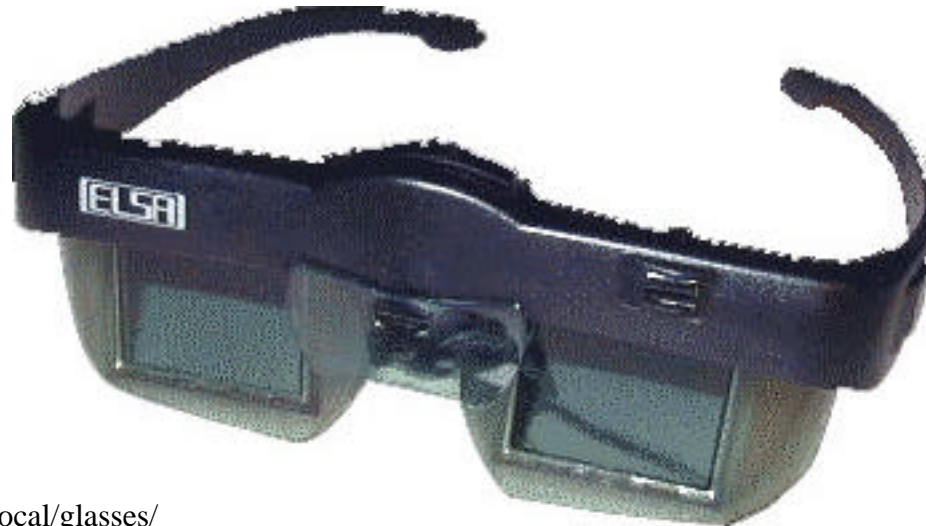
Stereo Buffers

- Called stereopsis or stereo vision
- Two images (one for each eye) are rendered to fool your eyes into giving objects real depth
 - Can be really convincing!
 - Not everyone is able to do it (magic 3d image?)



Stereo Buffers

- Hardware
 - Old-school paper 3d glasses
 - HMD (head mounted display)
 - Shutter glasses (cheap and work well)
 - Synchronizes shutter speed with the monitor refresh rate
 - Now supported in the display itself
- Doubles the amount of buffer memory needed





Stencil &

Accumulation

- Normally the same size as the color buffer
- Stencil buffer
 - Used to mask off regions of the color buffer
 - 1 bit – simple masking
 - 8 bits – complex effects like shadow volumes
- Accumulation buffer
 - Used to add and subtract images from the color buffer
 - Needs much higher precision than the color buffer
 - 48 bits for a 24 bit color buffer would allow 256 composed images
 - Useful for effects like depth of field, antialiasing, motion blur



T-buffer

- Useful for supporting fast antialiasing in hardware
- Contains a set of 2, 4, or more image and Z-buffers
- Send down a triangle once, it is sent to multiple buffers in parallel with a different offset in each
- Images are recombined to do AA
- Does not require multiple passes to do AA
- Raises hardware cost; much of the pipeline has to be duplicated in each parallel unit
- 3dfx was the only one to ever implement it

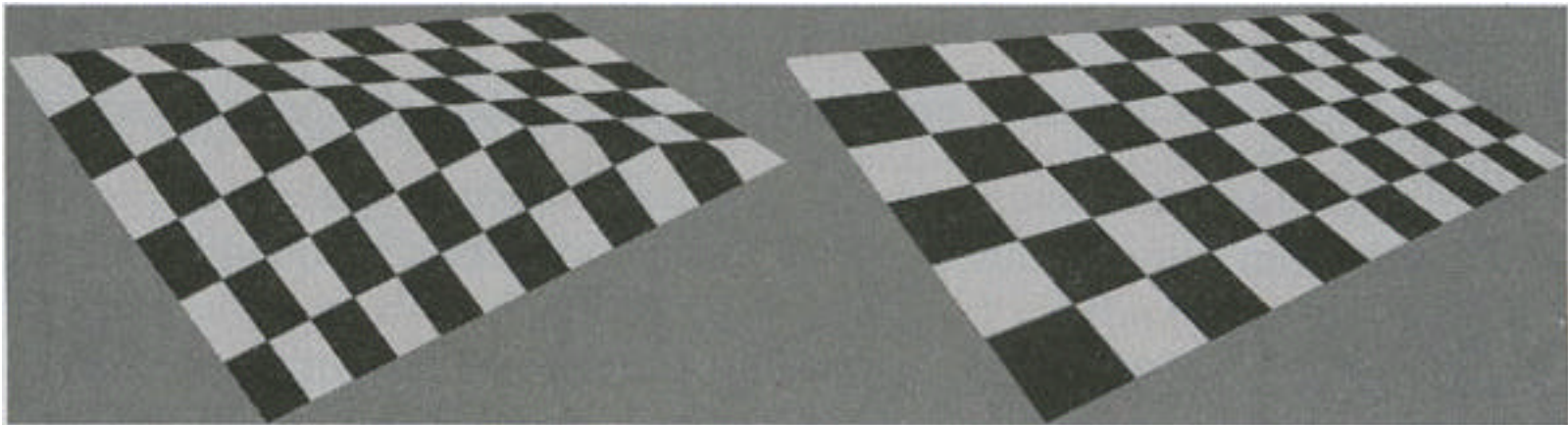


Buffer Memory

- How much memory do we need for all these buffers?
- Ex:
 - Color buffer is $1280 * 1024 * 3$ (bytes) = 3.75 MB
 - Double buffering doubles this to 7.5 MB
 - Z-buffer has 24 bpp = 3.75 MB
 - Stencil buffer of 8 bpp and accumulation buffer of 48 bpp = 8.75 MB
 - $7.5 + 3.75 + 8.75 = 20$ MB
 - If you are using stereo, this doubles the color buffer and adds another 1.25 MB
 - Only one Z-buffer is ever needed for the current color buffer

Perspective-Correct Interpolation


- Implemented by the rasterizer in hardware
- Vertex position can be lerped (linear interpolation)
- Cannot do this for colors and texture coordinates





Perspective-Correct Interpolation

- To correct this:
 - Linearly interpolate both the textured coordinate and $1/w$
 - Thus $(u,v) = (u/w, v/w) / (1/w)$
 - Called hyperbolic interpolation or rational linear interpolation



Graphics Architecture

- A little history...
 - Early processors just interpolated/textured spans
 - 1996 – 3Dfx Voodoo 1 introduced triangle setup
 - 1999 – NVIDIA GeForce256 introduced geometry stage acceleration in hardware (fixed function pipeline)
 - Today – Accelerated geometry and rasterization plus programmable shaders
- Trying to put as much on the card as possible
- Still some support on the CPU
 - Pentium 4 has SSE 2 which are SIMD extensions for parallel processing of vectors

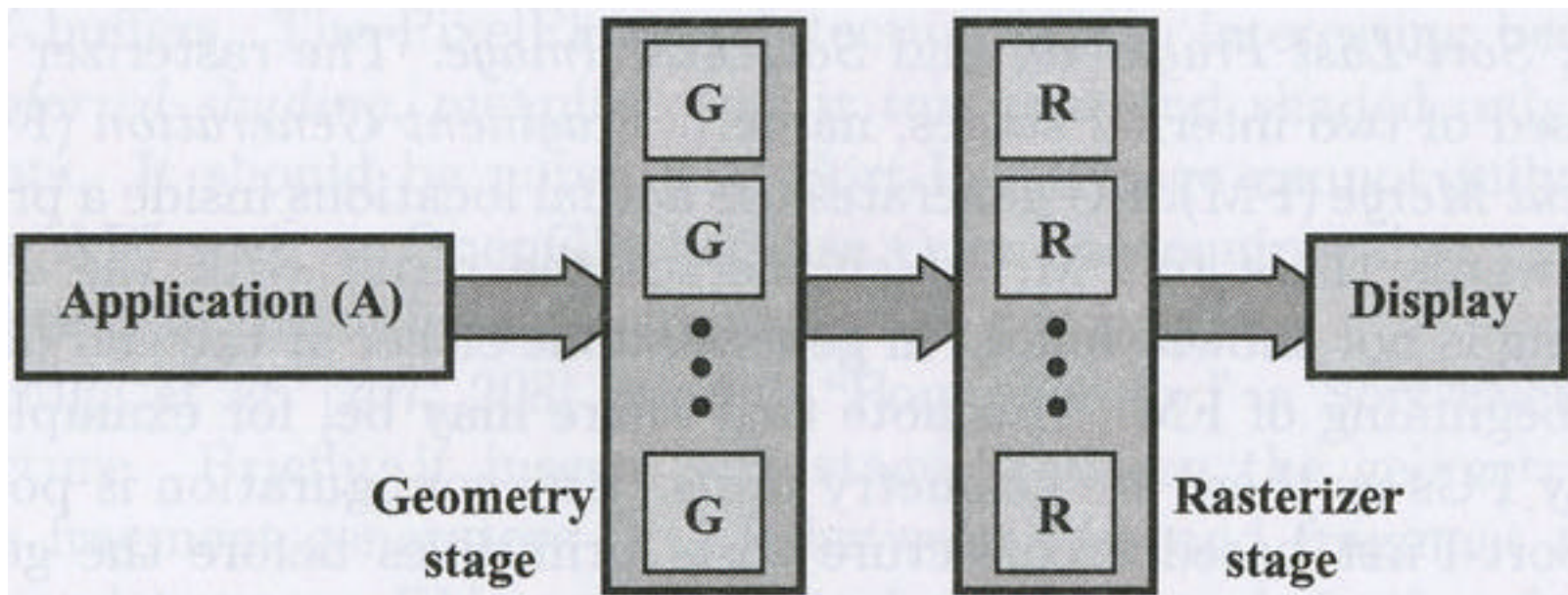


It's Wicked Fast

- Two approaches to getting super-high performance
 - Pipelining and Parallelizing
- Pipelining
 - N stages implemented in the hardware that are pipelined together
 - Gives a speedup of N
 - A GPU is 100 times faster than its equivalent CPU because of using pipelining
 - CPU actually has a higher clock speed
 - Hardware is customized for one area (ie graphics rendering)
 - More operations implemented in hardware (20 pipeline stages in the Pentium 4 and 600-800 in the GeForce3)

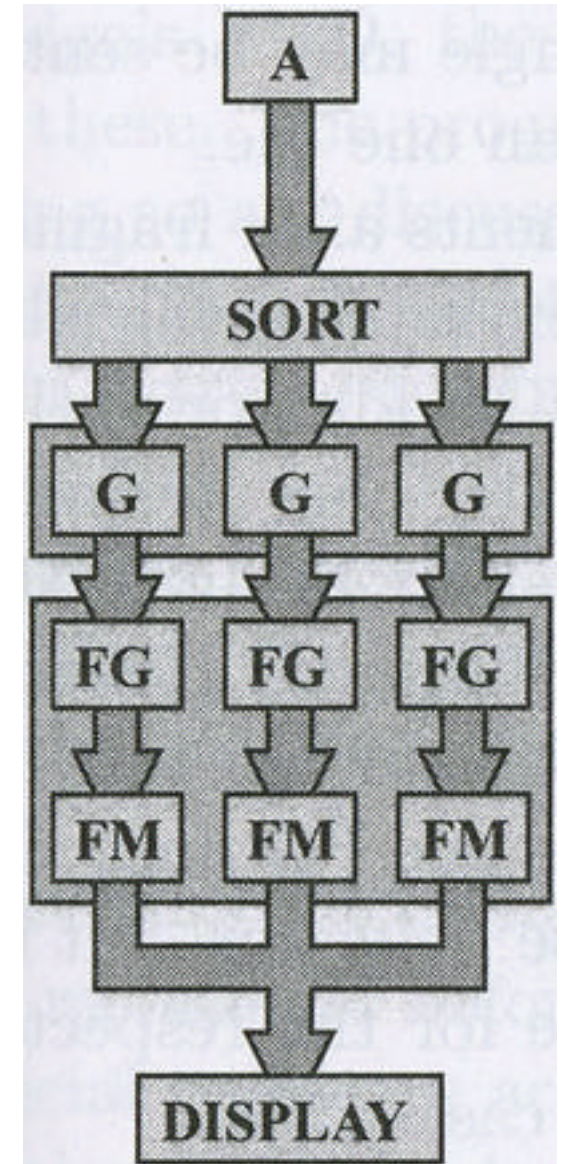
Parallelization

- Divide processing into N parallel processing units and merge results later
- Normally done for the geometry and rasterizer stages
- Results must be sorted at some point



Sort-First

- Primitives are sorted before geometry stage
- Screen is divided into a number of regions, or tiles
- Each processor responsible for a region
- Not used much among normal implementations





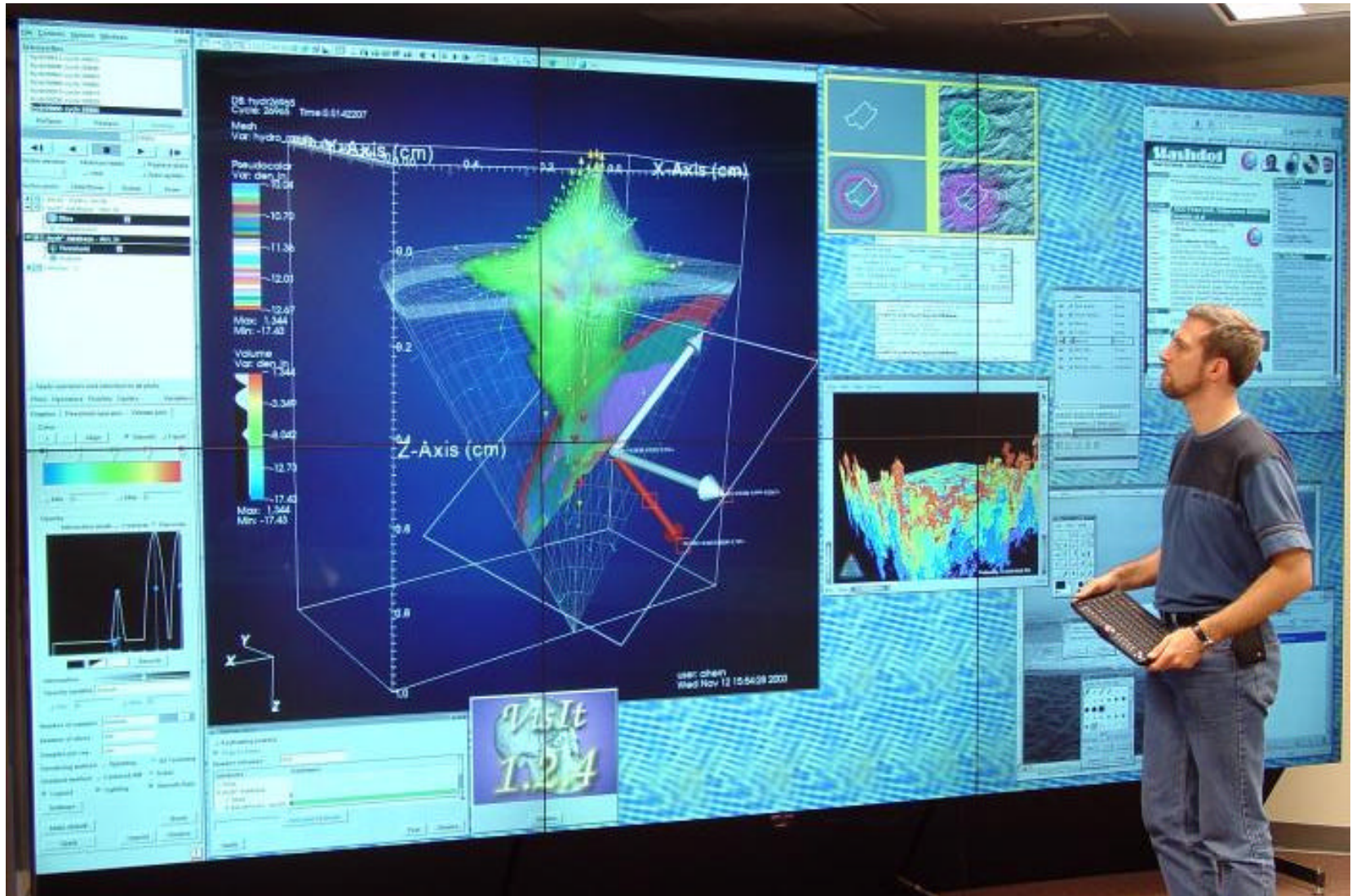
Sort-First

- Stanford WireGL project
 - Is now in the Chromium project (on sourceforge)
 - Rendering is done with a cluster machines and displayed using a projector for each tile
 - Created as an OpenGL implementation (Quake3 anyone?)
 - Pretty awesome if you need a HUGE display with REDICULOUS resolution
 - System has also added motion tracking (!)

<http://chromium.sf.net>

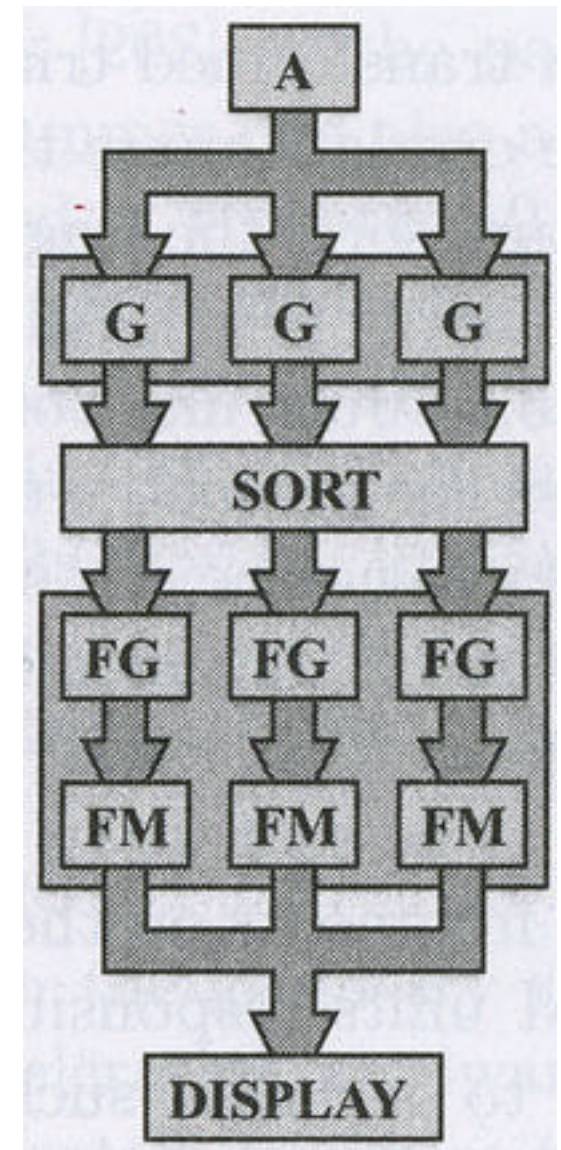
I Need One of These

- Maybe my wife can get me one for Christmas...



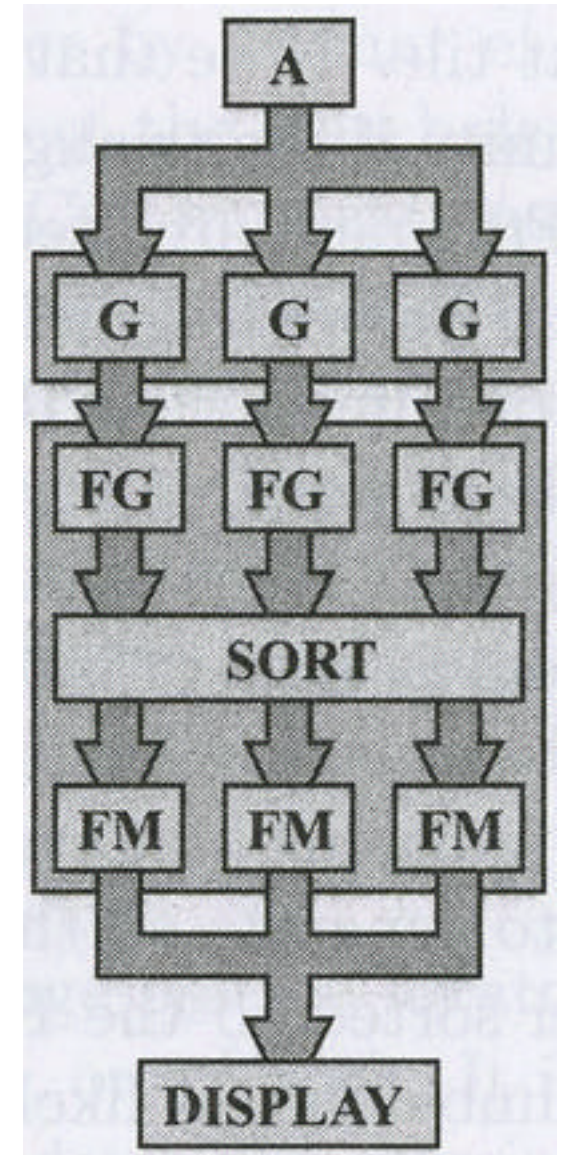
Sort-Middle

- Distribute over the geometry units and then sort results
- Used in the InfiniteReality and KYRO systems
- After geometry stage, the primitive location is known so it can be resorted to the right FG
- Each FG is responsible for a region
- If a triangle overlaps more than one region it can be sent to multiple FGs (which is bad)



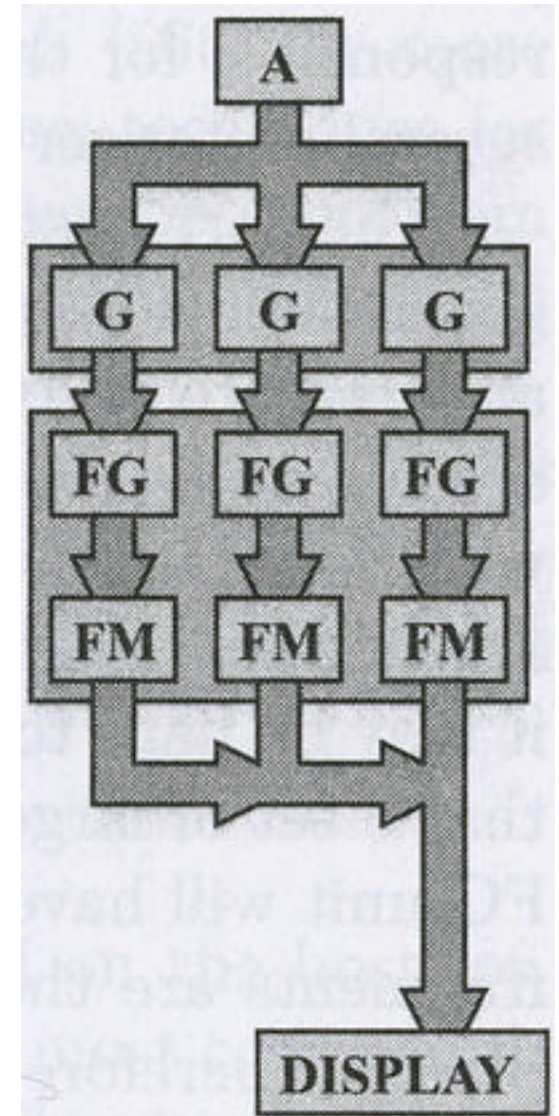
Sort-Last Fragment

- Sorts fragments after fragment generation and before fragment merge
- Used by the Xbox
- No overlap of as there is in Sort-Middle
- Imbalance can occur if a set of large triangles are sent to one FG



Sort-Last Image

- Sorts after all rasterization is done
- Each pipeline renders with depth and then results are composed based on z values
- Implemented in the PixelFlow architecture
 - Used deferred shading: only textured and shaded visible fragments
- Cannot be implemented in OpenGL because it does not render primitives in the order sent in



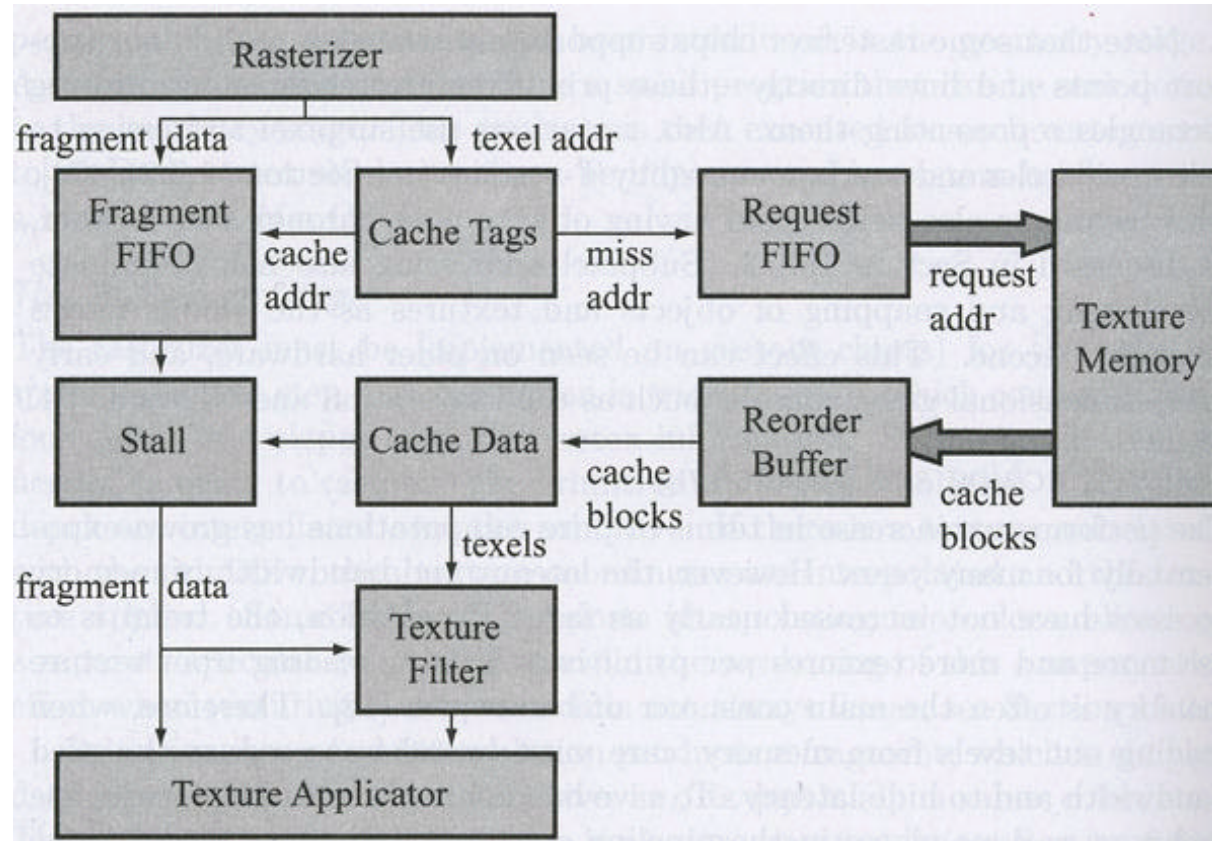


It's All About the Textures

- GPU computation speed is growing exponentially
- Memory speed and bandwidth is not
- Yet textures are the way to go for fast real-time graphics
- Caching and prefetching used to speed up texture access

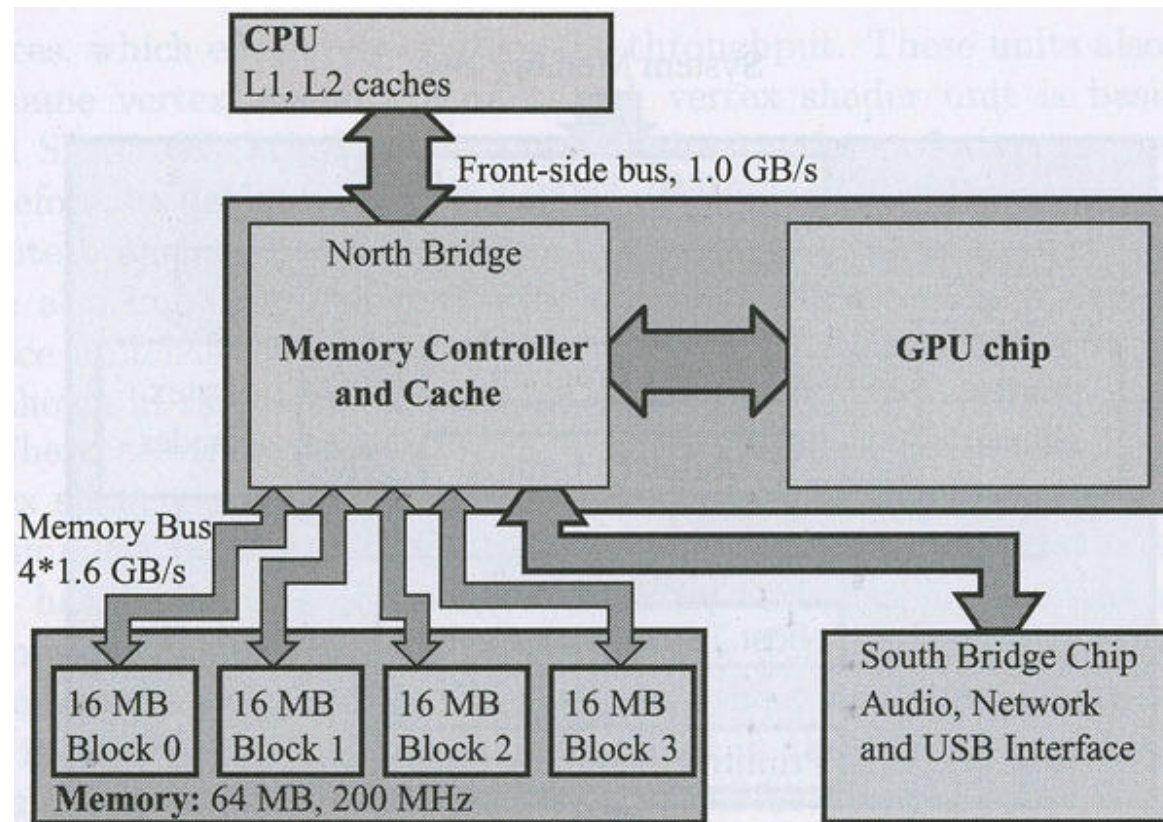
Texture Catching


- Rasterizer produces fragments
- Request queue gets textures from memory
- Reorder buffer sorts blocks as they were requested
- Performs at 97% of the ideal (no latency)



Memory Architectures

- Unified Memory Architecture (UMA)
 - GPU can use any of the host memory
 - CPU and GPU share bus
 - Used by Xbox and the SGI O2





Memory Architectures

- Non-unified memory
 - GPU has dedicated memory not accessible by the CPU
 - Does not have to share bus bandwidth
 - Used by the KYRO and InfiniteReality



Buses and Bandwidth

- Two methods for sending the GPU data, Pull and Push
- Pull
 - Data is written to system memory
 - GPU then pulls the data from memory
 - aka Direct Memory Access (DMA)
 - GPU can lock a region of memory that CPU cannot use
 - GPU works faster and CPU time is saved
- AGP uses pull
 - A dedicated port (ie bus) that only the CPU and GPU use
 - AGP 4x is 1067 Mbytes/sec. AGP 8x (3.0) is 2.1Gbytes/sec



Push Method

- Data is written to the GPU once per frame
 - More bandwidth left over for the application
 - Graphics card needs to have a large FIFO buffer
- Pull method is better for memory data that is static since it can just stay in memory



Ex: How Much Bandwidth?

- For each pixel we need to read the z-buffer, write back a z-value and the color buffer, and one or more texture reads. = 60 bytes per pixel
- Assume 60 fps with 1280x1024 resolution and a depth complexity of 4:
 $4 * 60 * 1280 * 1024 * 60 \text{ bytes/s} = \text{about } 18 \text{ Gbytes/s}$



Ex: How Much Bandwidth?

- Assume bus speed is 300 MHz with DDRAM (256 bits per clock):
 $300\text{MHz} * 256/8 = \text{about } 9.6 \text{ Gbytes/s} < 18 \text{ Gbytes/s}$
- Memory bandwidth becomes a big bottleneck
- This can be reduced with texture caching, prefetching, compression, etc

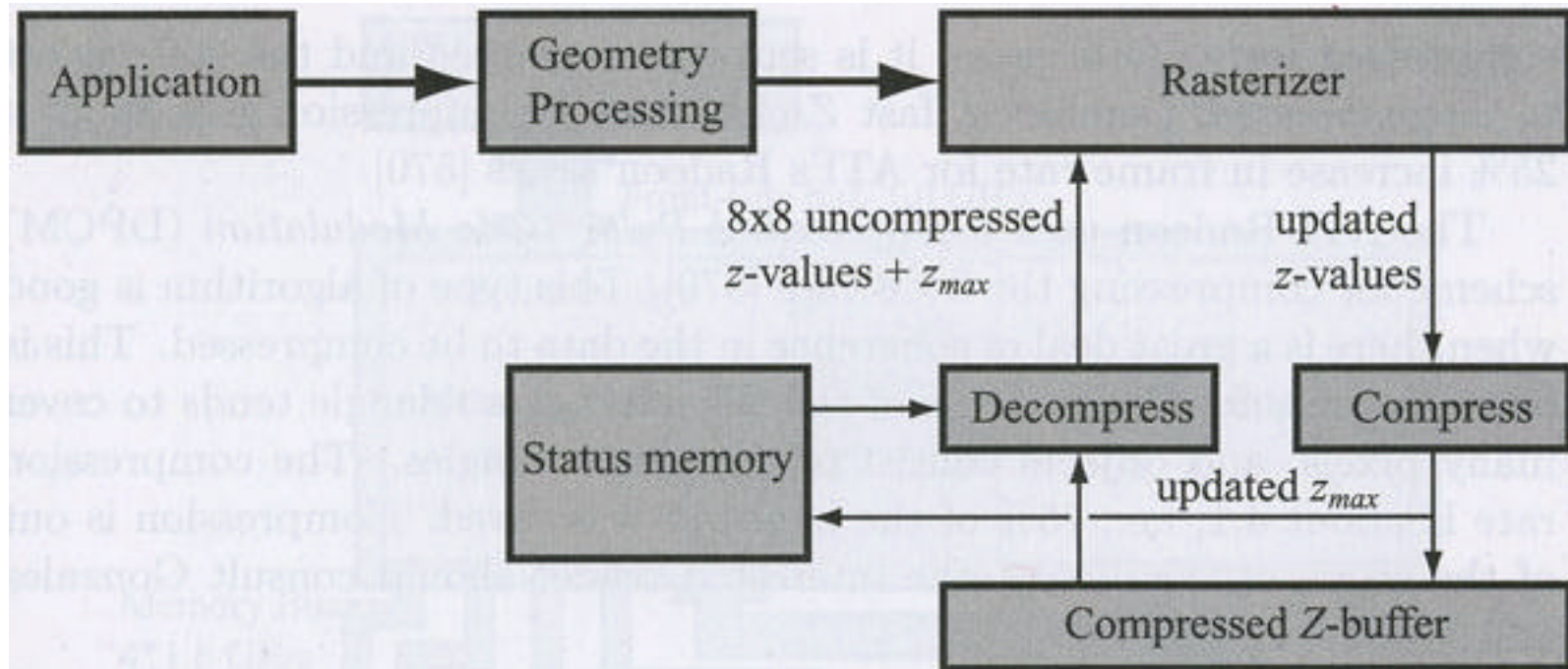


Reading from Buffers

- Reading from buffers (Z-buffer, etc) can be slow
- Often writing is done over AGP while reading is done over PCI
- Reading from the GPU should probably be avoided

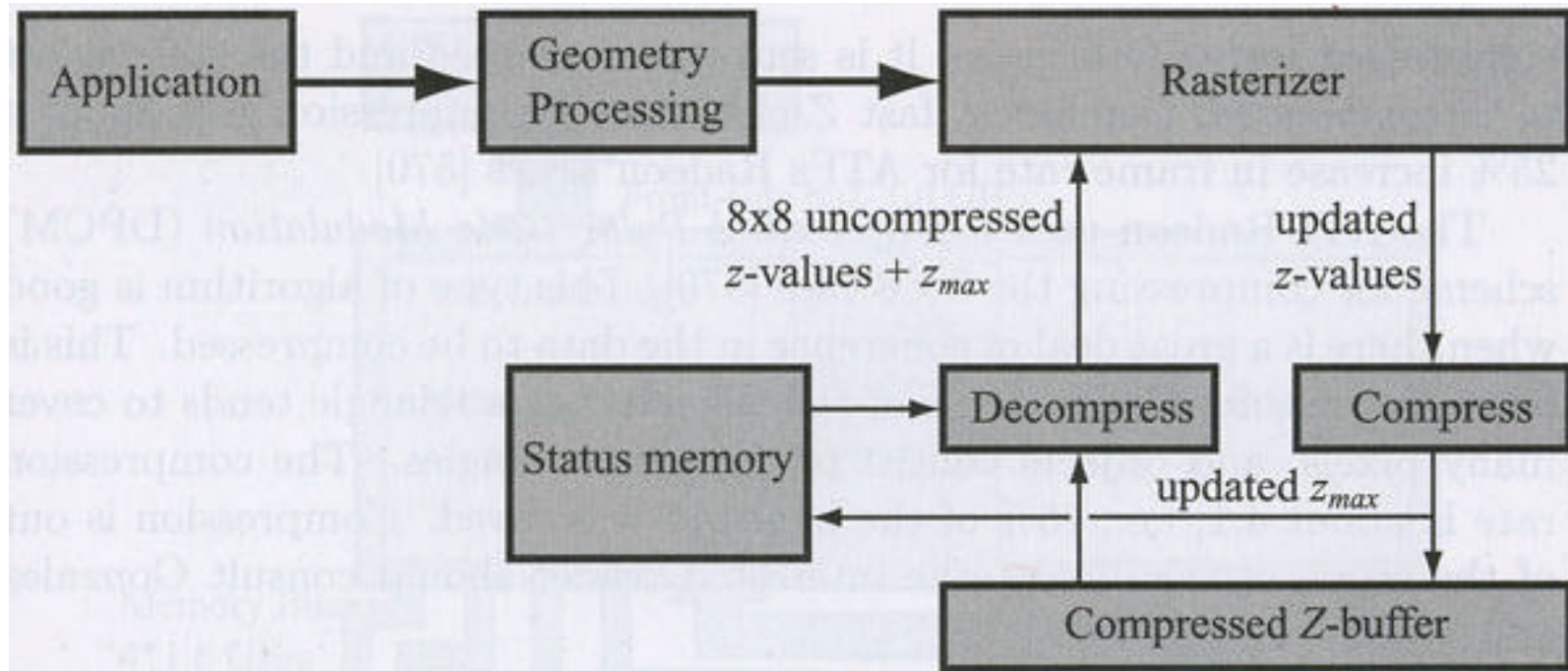
Hardware Z-buffer

- Status memory stores the state of a 8x8 pixel tile in the frame buffer and a z_{max} for each tile
 - State can be compressed, uncompressed, or cleared



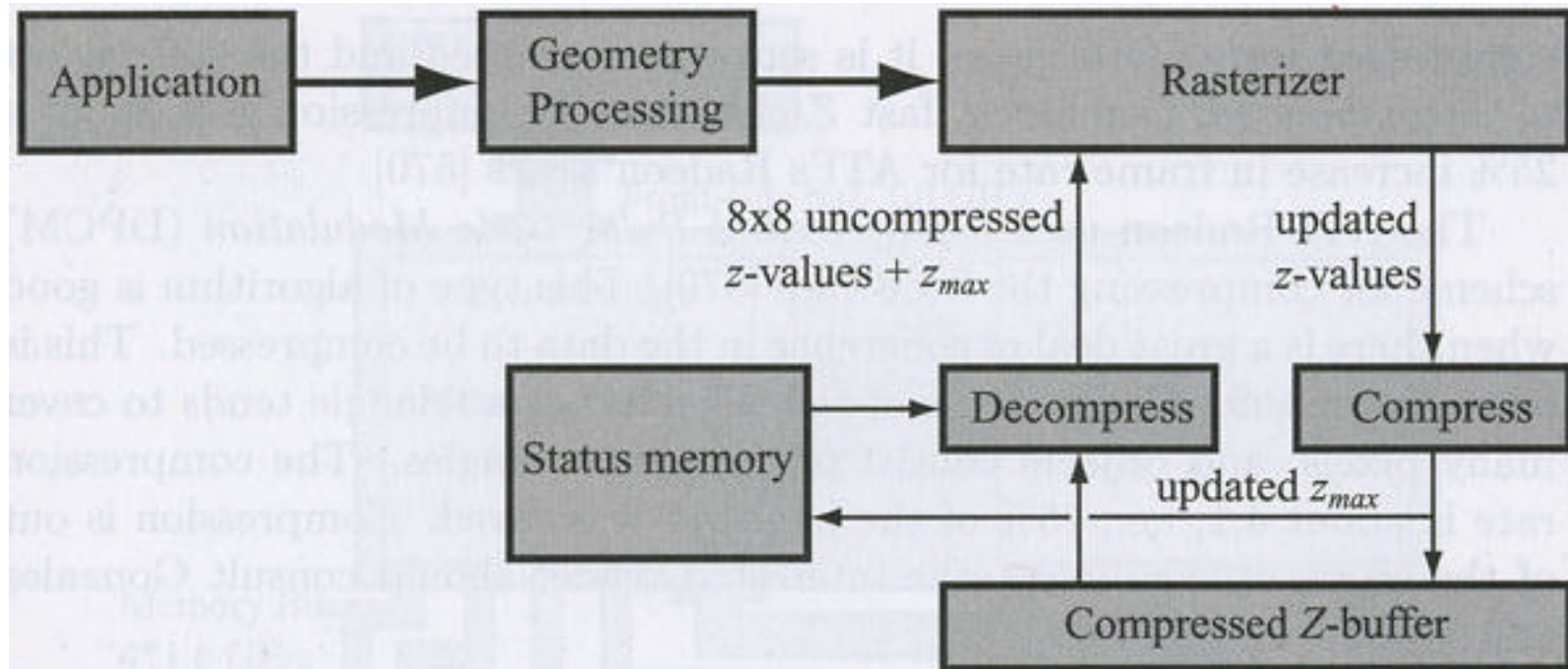
Hardware Z-buffer

- To do a fast clear, just set all states to “cleared”



Hardware Z-buffer

- ATI Radeon uses fast Z-clear and Z-compression for a 25% frame rate increase
 - A Differential Pulse Code Modulation (DPCM) is used
 - Good when high coherence in the data
 - Reduces memory bandwidth by 75%





Hardware Occlusion Culling

- Z_{max} is used to check whether a tile is occluded and then pipeline is exited early
- Saves on bandwidth
- Different methods
 - Check the minimum z -value of the triangle vertices against z_{max} . Not very accurate but very fast.
 - Test the corners of the entire tile against z_{max} . If larger than the tile it can be skipped.
 - Test each pixel against z_{max} .
- Implemented in both GeForce3 and Radeons

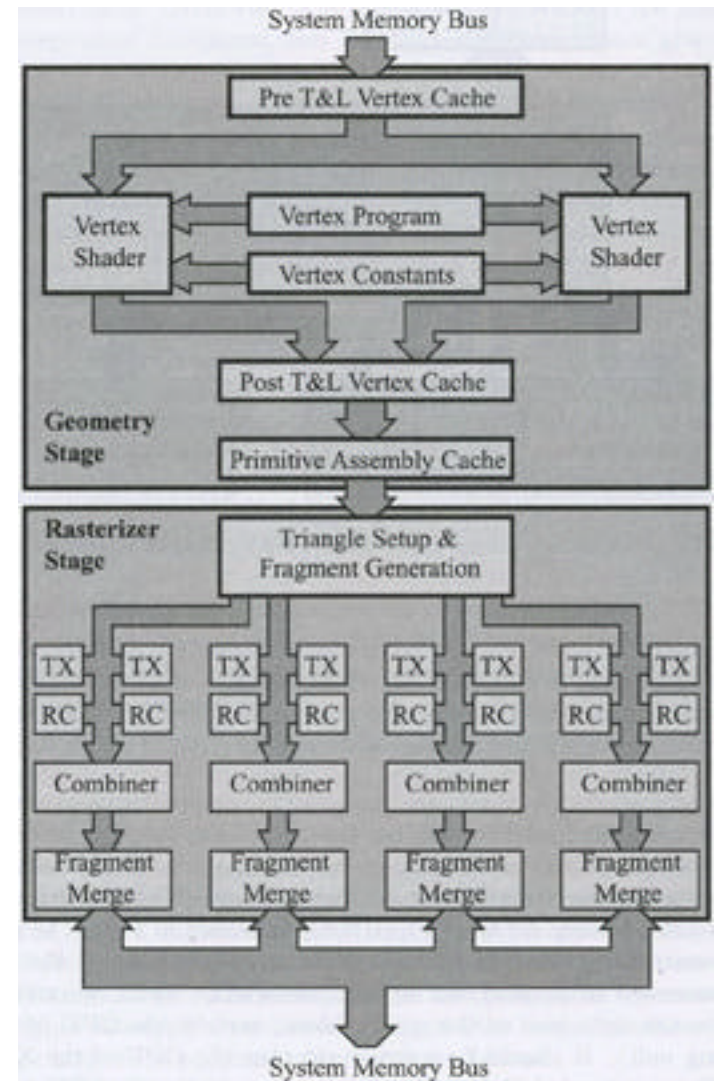
Case Study: Xbox

- Built by NVIDIA and Microsoft
- Uses the UMA
 - Memory is divided into blocks and can be accessed in parallel
- CPU is an Intel Pentium III 733 Mhz
- GPU is an extended GeForce3
- Supports programmable vertex and fragment shaders



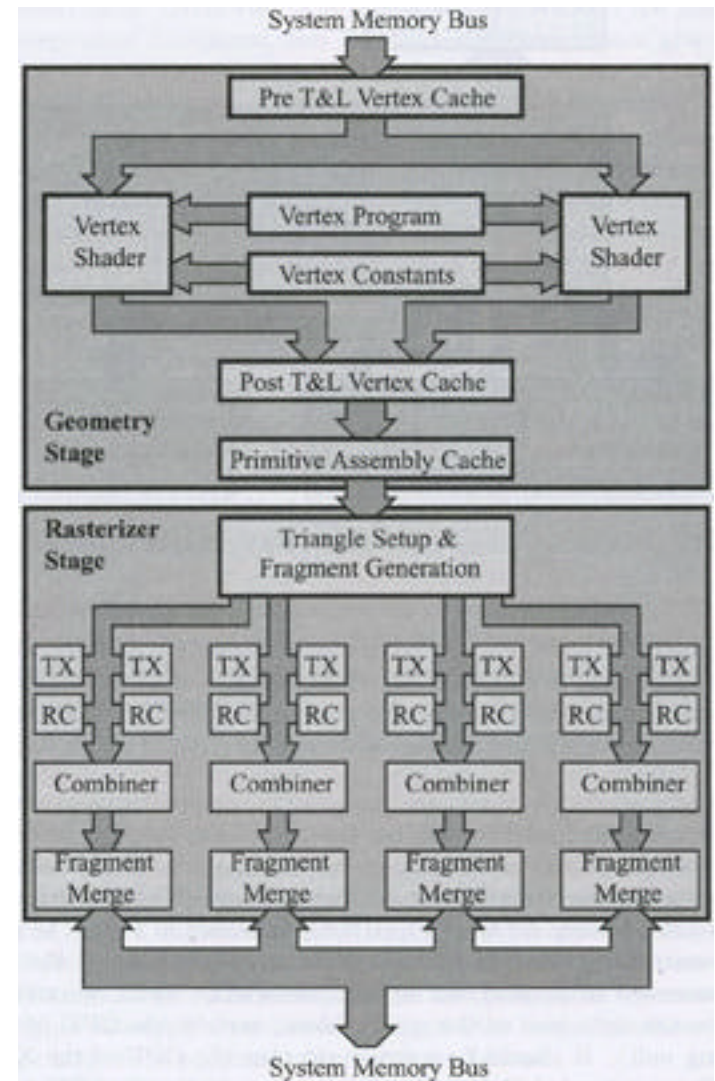
Case Study: Xbox

- Has dual vertex shaders which doubles throughput
- Pre T&L cache (4 bytes) avoids redundant memory fetches
 - Caches vertices (used by 6 triangles on average)



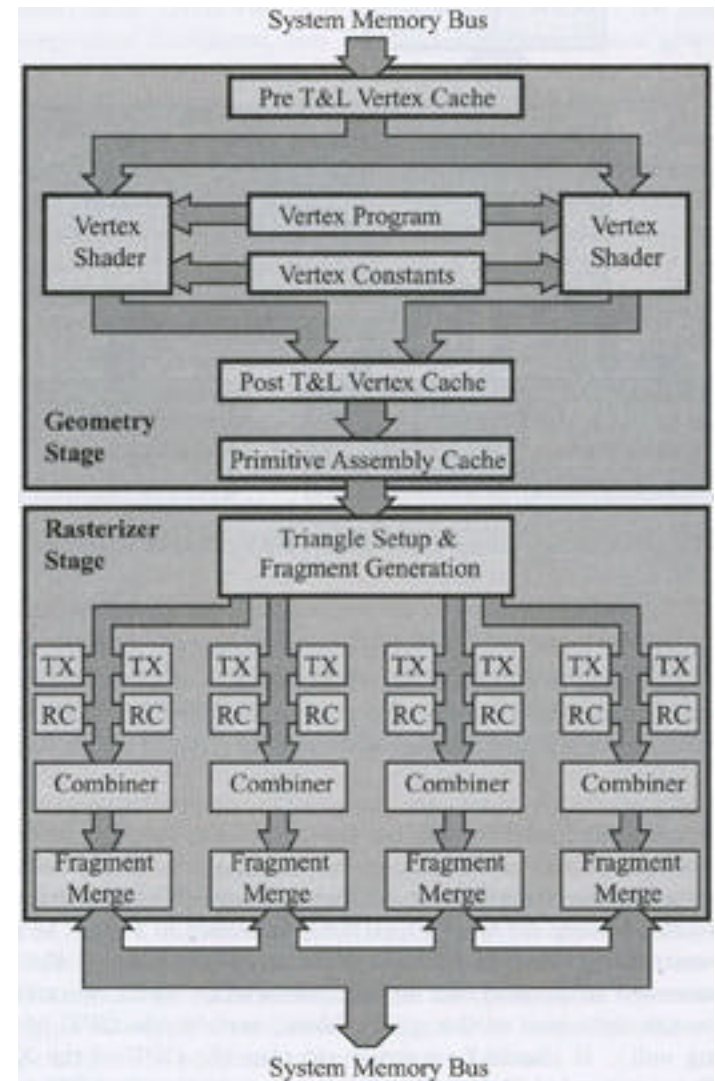
Case Study: Xbox

- Post T&L cache (16 vertices) avoids processing the same vertex more than once with the shader
- Primitive Assembly Cache (3 fully shaded vertices) avoids redundant fetches to the Post T&L cache
 - ex: a vertex may be needed multiple times in a triangle strip



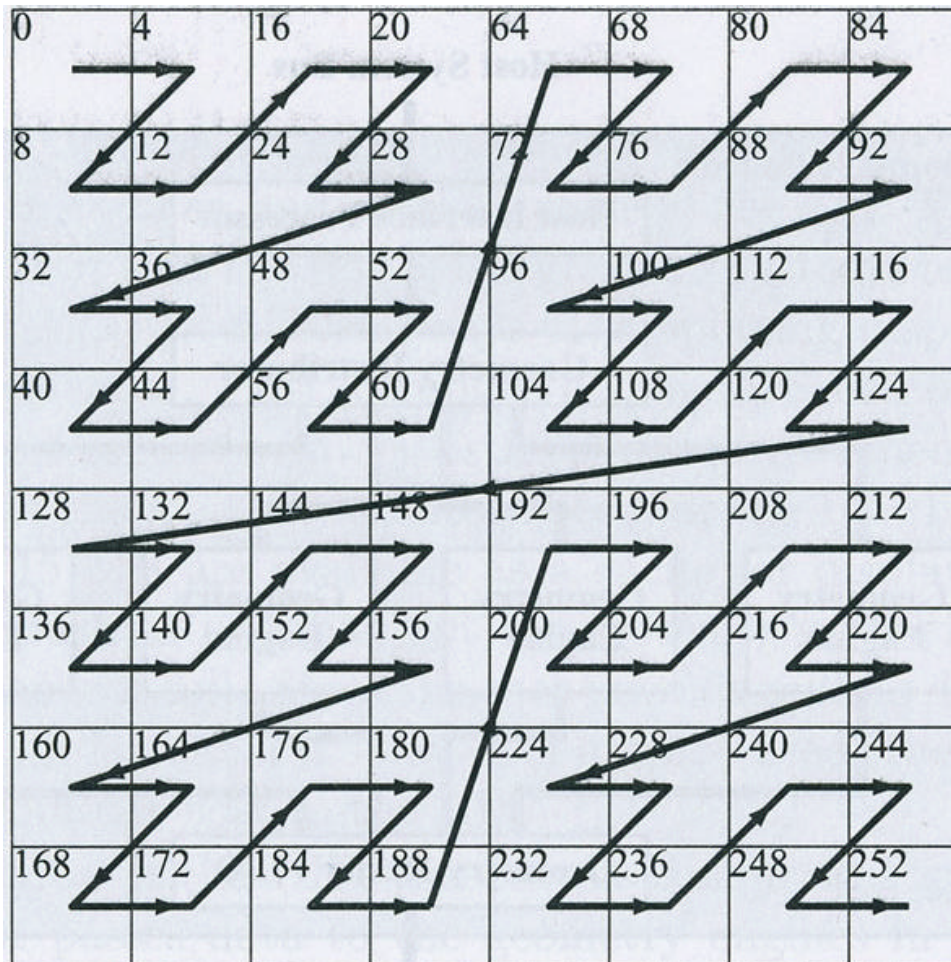
Case Study: Xbox

- Rasterizer has 4 parallel, programmable pixel shaders
 - TX – texture unit. Processes fragments as they come in
 - RC – register combiner.
 - Combiner – computes the final fragment color (after fog, etc)
 - Fragment merge – computes final pixel color (z-buffer, etc)



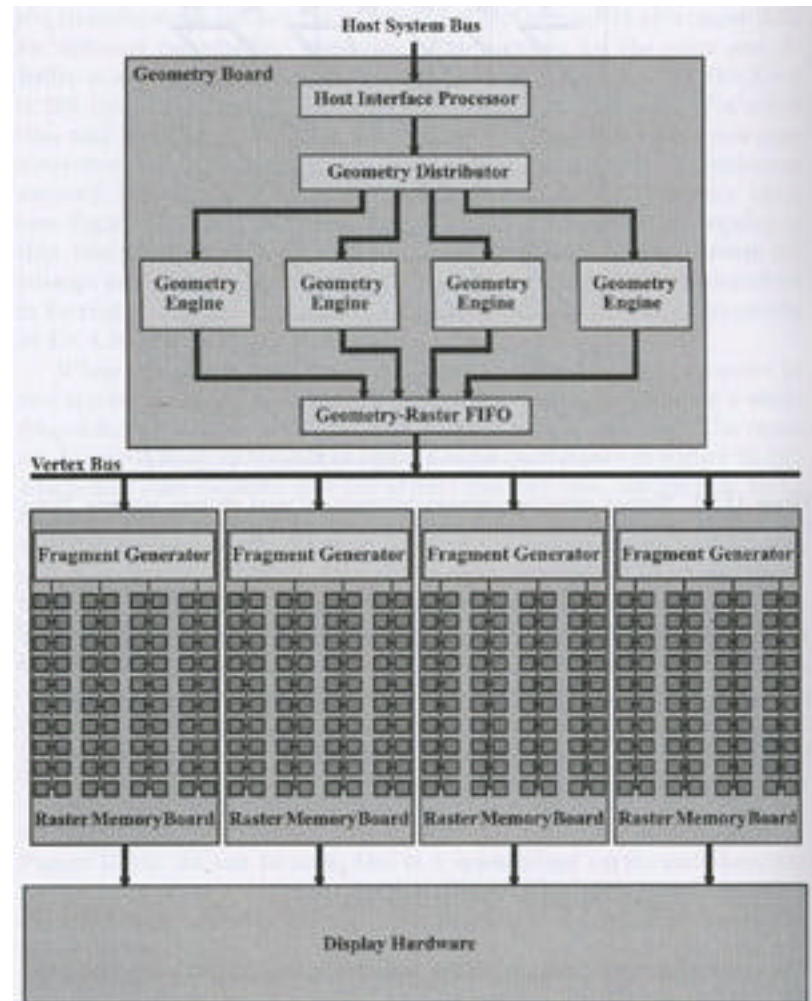
Case Study: Xbox

- Uses texture swizzling to increase cache performance and page locality



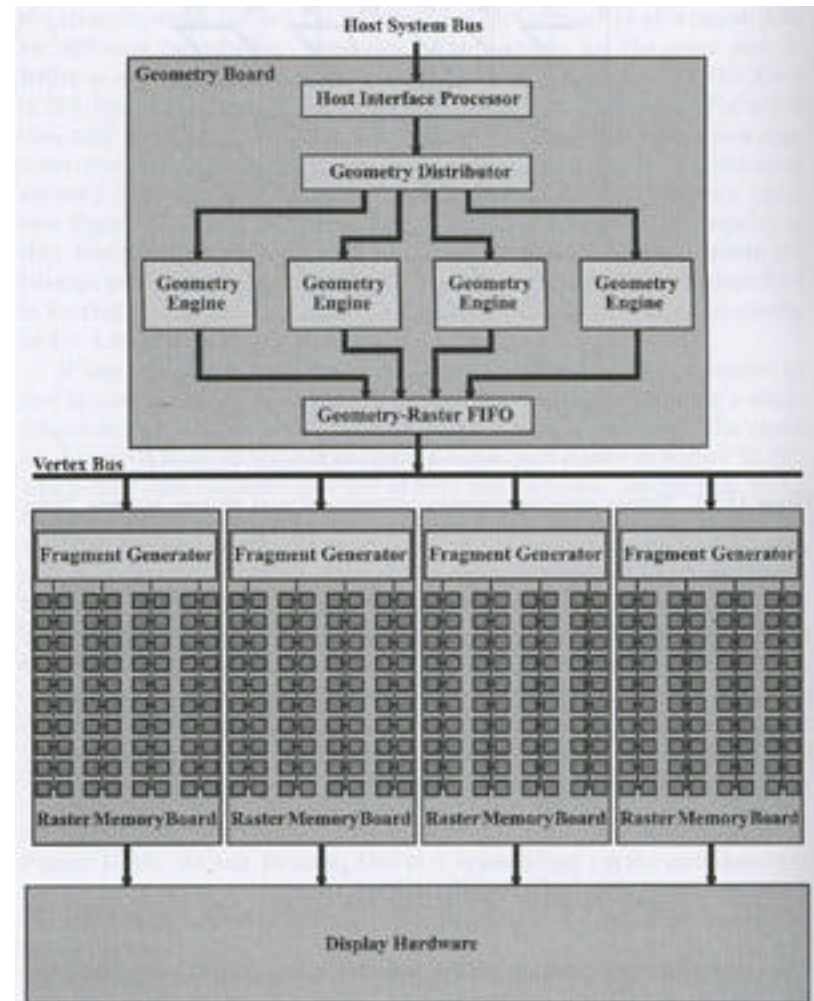
Case Study: InfinteReality

- A Sort-Middle architecture produced by SGI
- Host Interface Processor
 - Responsible for bring in work to the system
 - Can get display lists from memory with DMA
 - Also has a 16MB cache for display lists
 - Per vertex info can be sent directly from the host



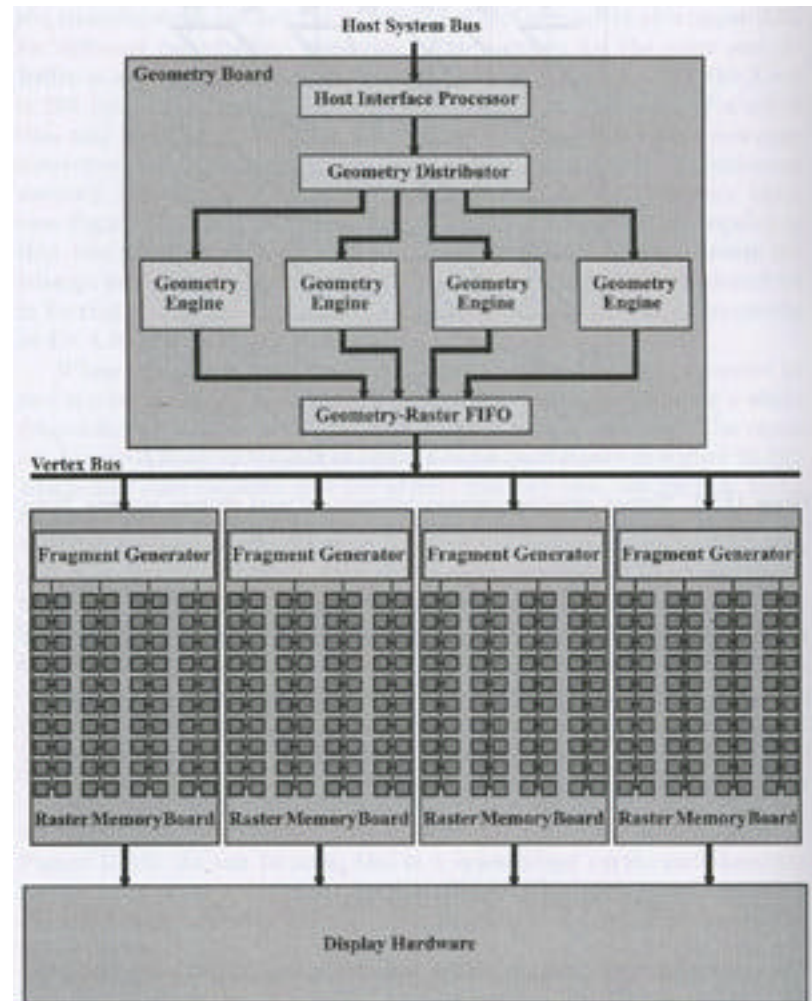
Case Study: InfinteReality

- Geometry Distributor
 - Passes work to the least busy of the geometry engines
 - Each work item has a number so they can be sorted later into the order they came in (compatible with OpenGL)



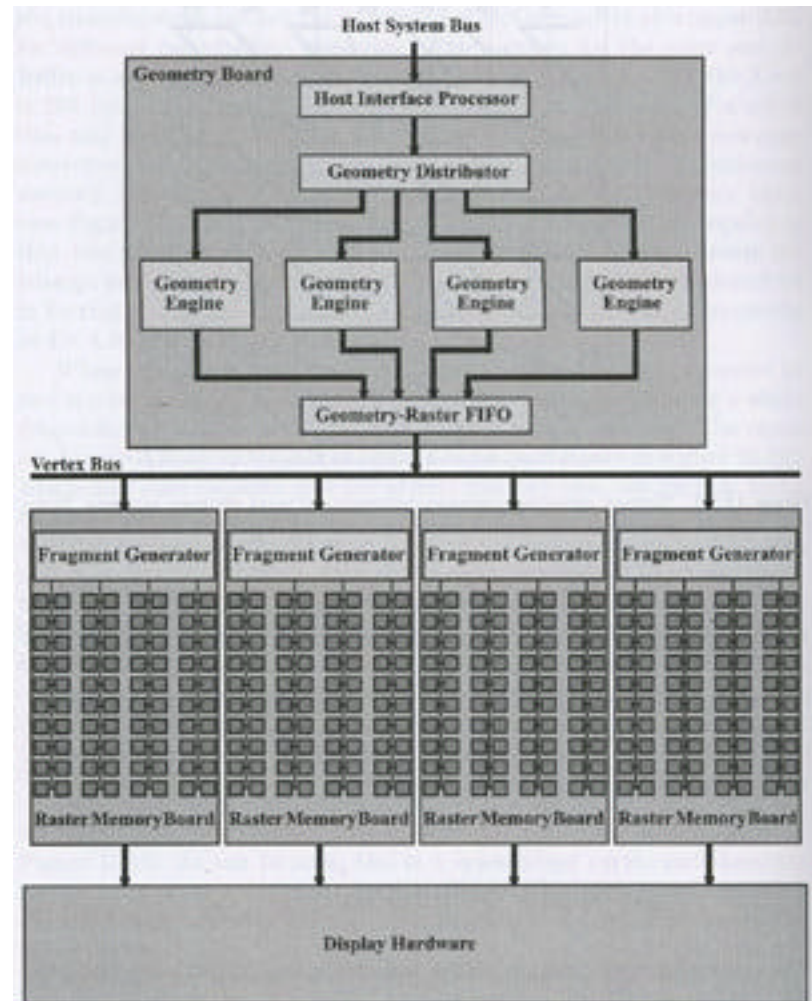
Case Study: InfinteReality

- Geometry Engine
 - Contains 3 Floating-point Cores (FPC) so the elements of a vertex can be processed in parallel (SIMD)
 - Each FPC is like a mini-cpu for processing vertices
 - Four-stage pipeline
 - In-chip memory (2560 words)



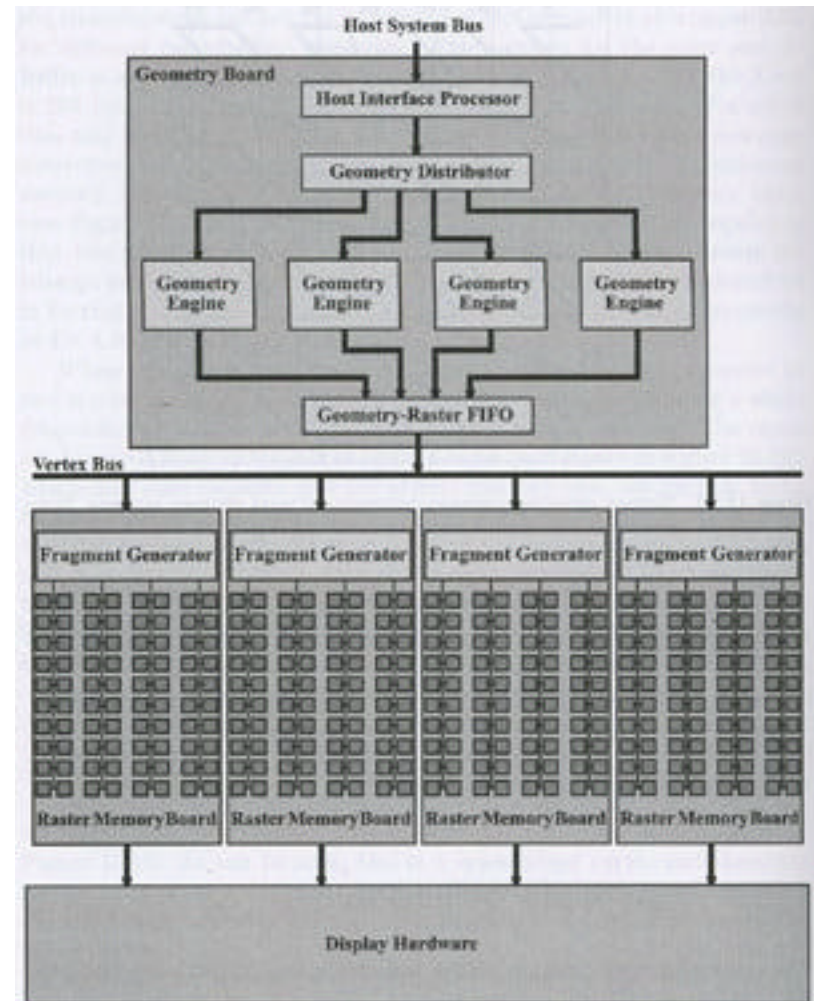
Case Study: InfinteReality

- Geometry-Raster FIFO
 - Reorders vertices using the numbers given to them
 - Feeds the correct stream of vertices onto the Vertex Bus
 - Can hold 65,536 vertices



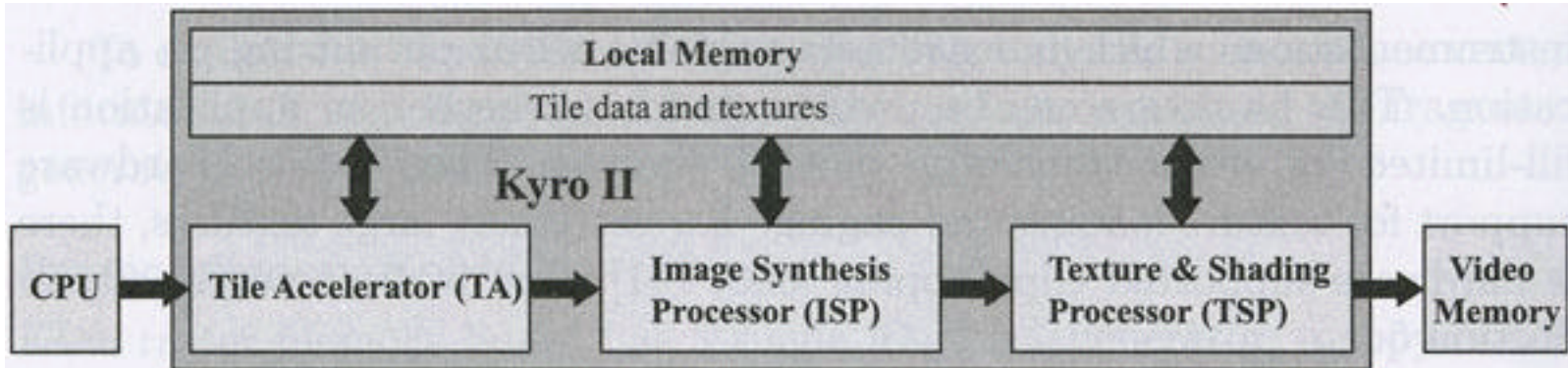
Case Study: InfinteReality

- Raster Memory Board
 - A fragment generator
 - Contains a copy of entire texture memory
 - Scan conversion by evaluating the plane equations rather than doing interpolation
 - Better for small triangles
 - Contains 80 image engines
 - Distributes work to the image engines



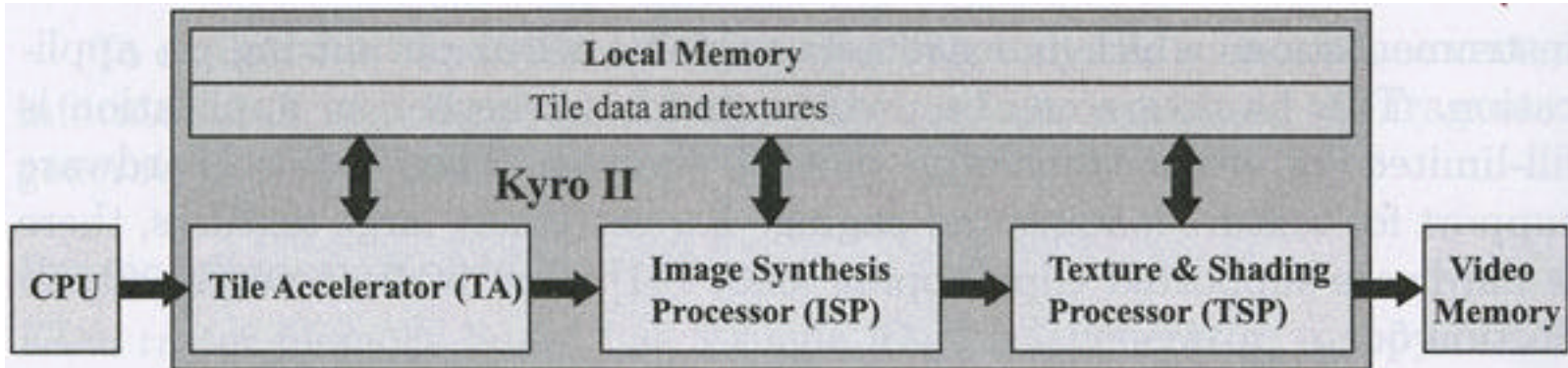
Case Study: KYRO

- Implements a tile-based algorithm in hardware
- Screen divided into equal size, rectangle regions
- Back buffer, Z-buffer, and stencil buffer only need to be the size of a tile
 - Stored on chip
 - About 1/3 the bandwidth as the normal approach
- Geometry currently done the CPU but could be added to the chip



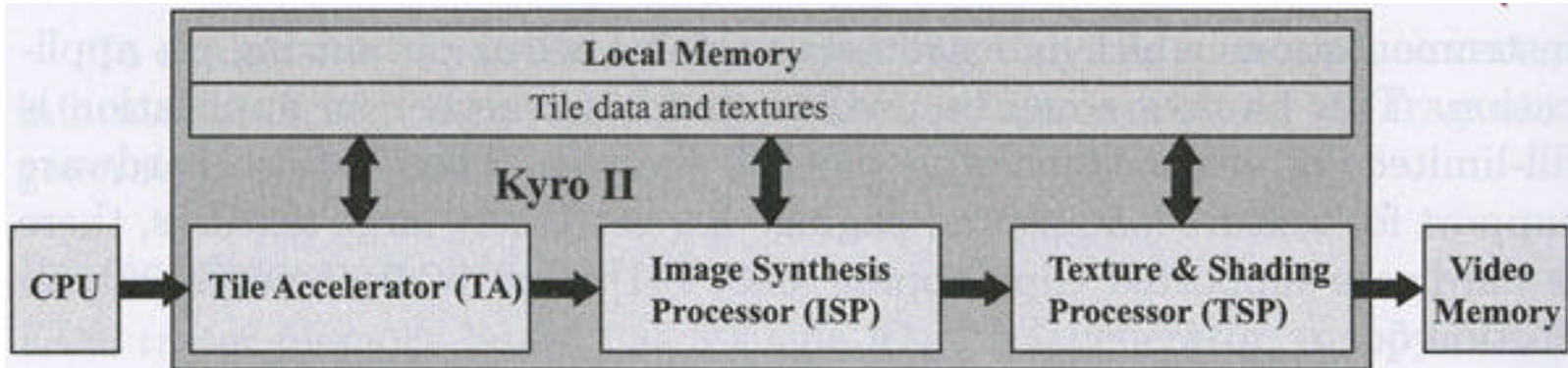
Case Study: KYRO

- Queues all incoming vertices until and arranges them by tiles
- Creates triangle strips on the fly for each tile
- The ISP processes complete tiles while the TA processing incoming vertices in parallel
- ISP handles Z-buffer, stencil buffer, and occlusion culling
 - Eliminates occluded triangles early to save bandwidth



Case Study: KYRO

- TSP does deferred shading; done after the ISP has already done depth testing
- Spans of pixels are grouped by what texture they use before being sent to the TSP
 - Swapping textures is expensive
- Can texture 2 pixels simultaneously
- KYRO designed to be pipelined/parallelized
 - Existing pipeline could be duplicated and parallelized





The End
(questions?)