



Architectures

EDA075
Mobile Graphics



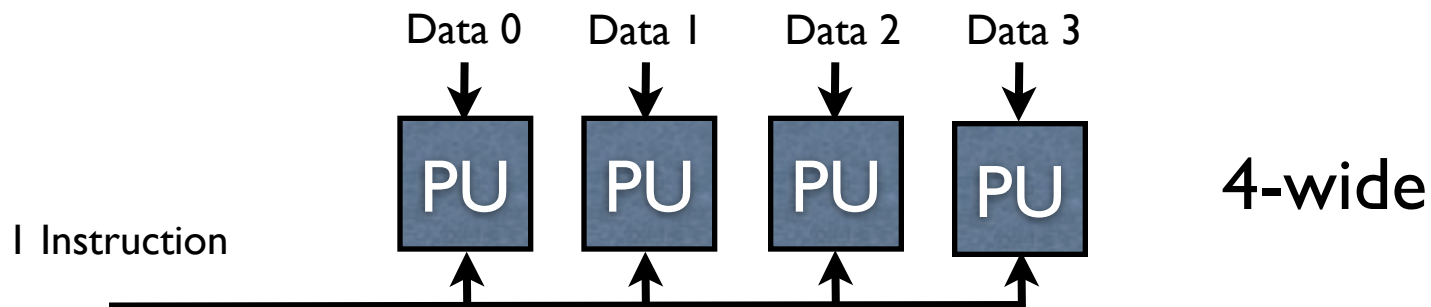
Michael Doggett
Department of Computer Science
Lund University

Overview of today's lecture

- The idea is to cover some of the existing graphics architectures (for mobile platforms)
 - Extremely little is known to the public ☹
- General info about graphics architectures
- XBOX
 - Traditional 'straight' pipeline
 - But slowly disappearing
 - non-unified architecture
- Tiling architectures [different from "tiled rasterization"!]
 - Kyro (PC-architecture)
 - Popular for mobile architectures where bandwidth and memory are limited
- ATI Bitboys' architecture
- Unified shaders
 - XBOX 360
 - High end PC graphics ATI Radeon 2900, 4870, 5870

Graphics Processing Unit GPU

- How to turn millions of triangles into pixels in 1/60 of a second?
- Parallelism !!
 - Pipelining
 - Single Instruction Multiple Data (SIMD)
- Memory Bandwidth reductions (covered previously)

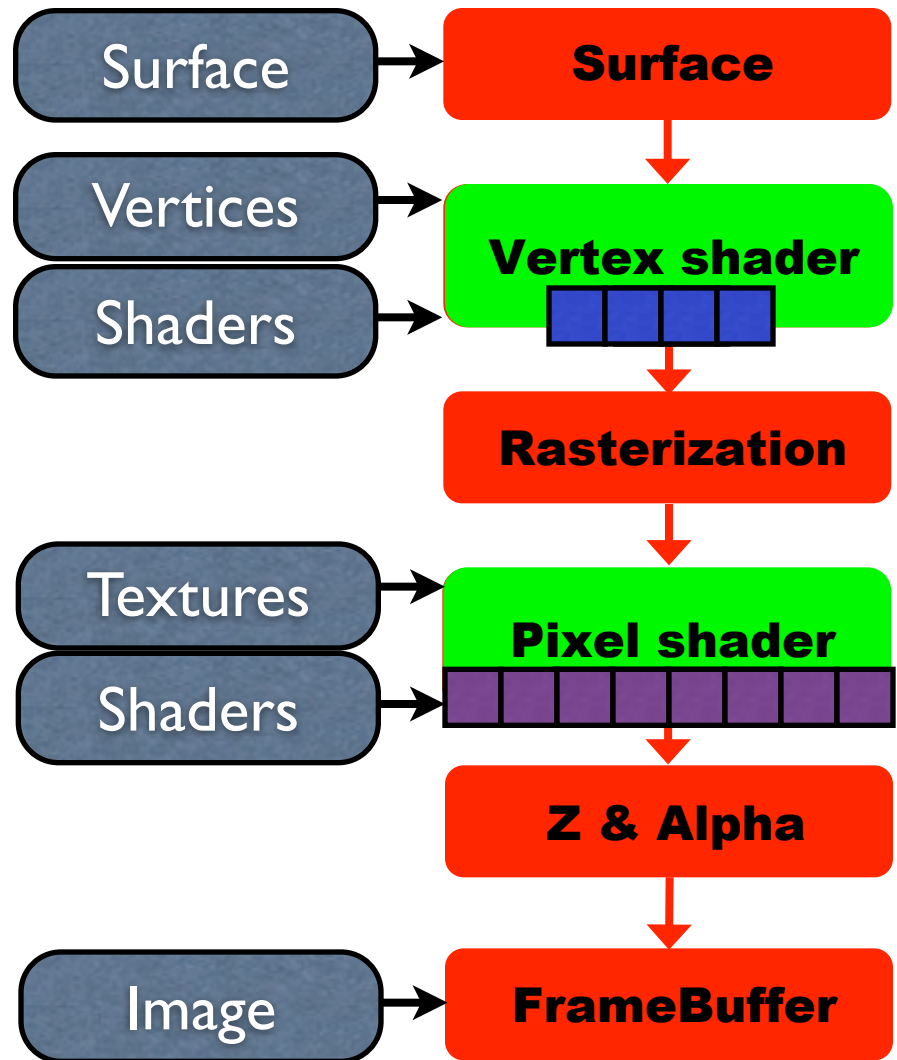


GPU Pipeline

New surface
tessellation stages

Programmable shaders
SIMDs

- 2001 -- Radeon 8500 (R200)
 - 2-wide Vertex shader
 - 4-wide SIMD Pixel shader
- 2002 -- Radeon 9700 (R300)
 - 4-wide Vertex shader
 - 8-wide SIMD Pixel shader

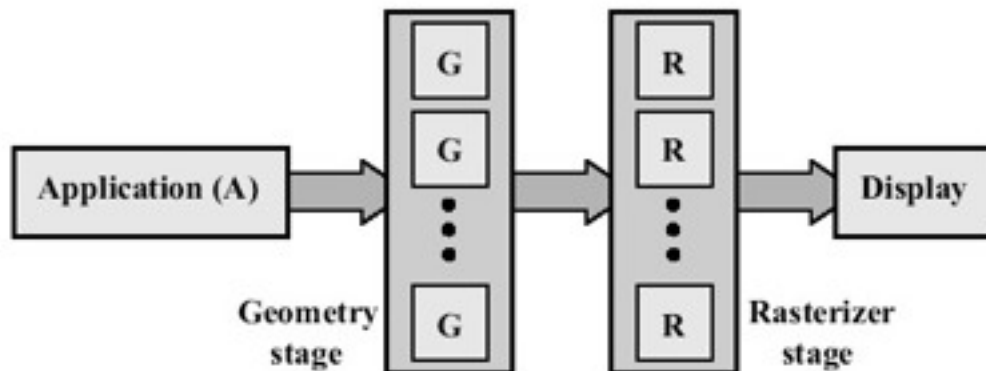


Briefly about pipelining

- In GeForce3: 600-800 pipeline stages!
 - 57 million transistors
 - Pentium IV: 20 stages, 42 million transistors
- Some desktop graphics cards:
 - GeForce FX 5800: 125 M transistors, 500 MHz
 - 2005: GeForce FX 7800: 302 M transistors, 430 MHz
 - 2007: ATI Radeon 2900, 700 M transistors, 740 MHz
 - 2009: ATI Radeon 5870, 2154 M transistors, 850 MHz
- Ideally: n stages \rightarrow n times throughput
 - But latency increases!
 - However, not a problem here
 - Chip runs at, say, 200 MHz (5ns per clock)
 - $5\text{ns} \cdot 700\text{stages} = 3.5 \mu\text{s}$
 - We got about 20 ms per frame (50 frames per second)
- Graphics hardware is simpler to pipeline because:
 - Pixels are independent of each other
 - Few branches and much fixed functionality
 - Don't need that high clock freq: bandwidth to memory is often bottleneck
 - This is changing with increased programmability though
 - Simpler to predict memory access pattern (do prefetching!)

Parallelism

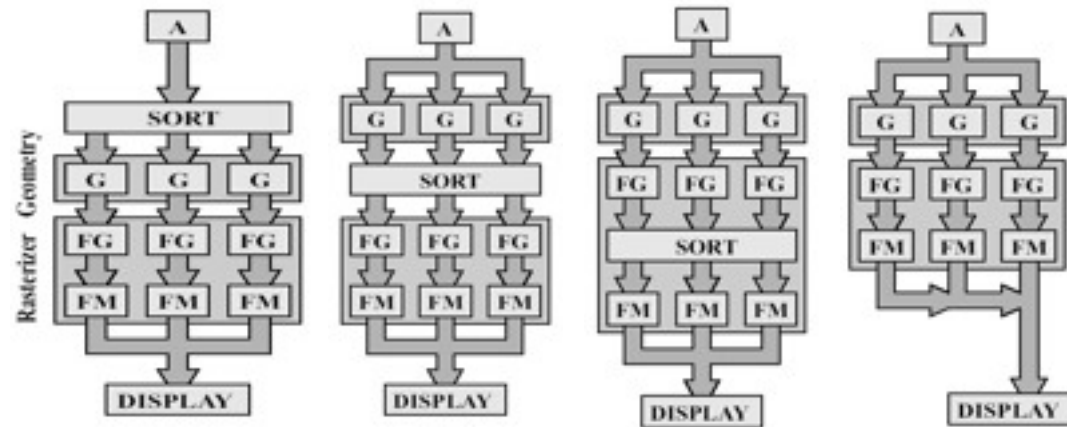
- "Simple" idea: compute n results in parallel, then combine results
- GeForce FX 5800: 8 pixels/clock, 16 textures/clock
 - With a pipeline of several 100 stages, there are many pixels being processed simultaneously
 - This does not apply directly to mobiles, though
- Not always simple!
 - Try to parallelize a sorting algorithm...
 - But pixels are independent of each other, so simpler for graphics hardware
- Can parallelize both geometry (vertex) and rasterizer (pixel):



Taxonomy of hardware

- Need to sort from model space to screen space
- Gives four major architectures:

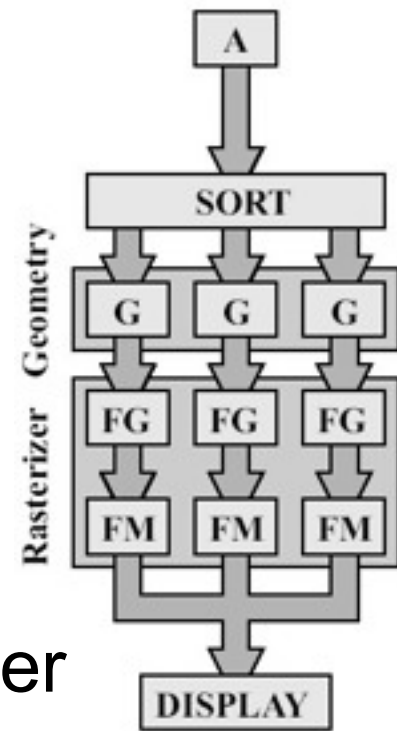
- Sort-first
- Sort-middle
- Sort-Last Fragment
- Sort-Last Image



- Will describe these briefly, and then focus on sort-middle and sort-last fragment (used in commercial hardware)

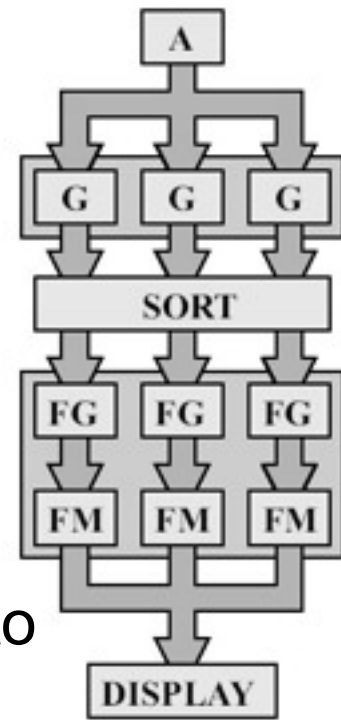
Sort-First

- Sorts primitives before geometry stage
 - Screen is divided into large regions
 - A separate pipeline is responsible for each region (or many)
- G is geometry, FG & FM is part of rasterizer
 - A fragment is all the generated information for a pixel on a triangle
 - FG is Fragment Generation (finds which pixels are inside triangle)
 - FM is Fragment Merge (merges the created fragments with various buffers (Z, color))
- Not explored much at all – kind of weird architecture
 - poor load balancing



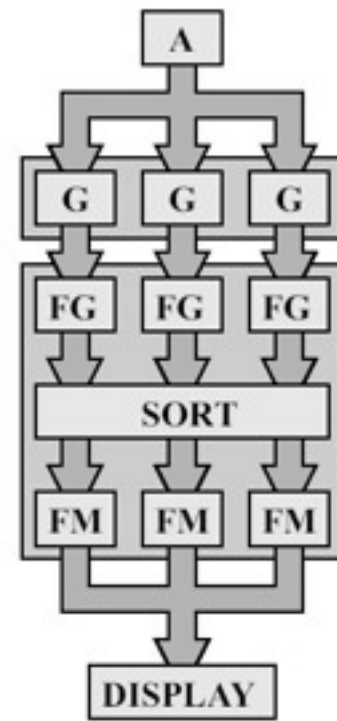
Sort-Middle

- Sorts between G and R
- Pretty natural, since after G, we know the screen-space positions of the triangles
- Spread work arbitrarily among G's
- Then depending on screen-space position, sort to different R's
 - Screen can be split into "tiles". For example:
 - Rectangular blocks (e.g., 4x4 pixels)
 - Every n scanlines
- The R is responsible for rendering inside tile
- A triangle can be sent to many FG's depending on overlap (over tiles)



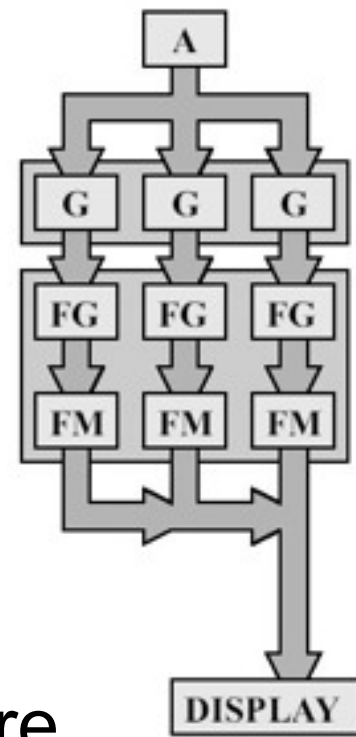
Sort-Last Fragment

- Sorts between FG and FM
- XBOX uses this!
- Again spread work among G's
- The generated work is sent to FG's
- Then sort fragments to FM's
 - An FM is responsible for a tile of pixels
- A triangle is only sent to one FG, so this avoids doing the same work twice
 - Sort-Middle: If a triangle overlaps several tiles, then the triangle is sent to all FG's responsible for these tiles
 - Results in extra work



Sort-Last Image

- Sorts after entire pipeline
- So each FG & FM has a separate frame buffer for entire screen (Z and color)
- After all primitives have been sent to pipeline, the z-buffers and color buffers are merged into one color buffer
- Can be seen as a set of independent pipelines
- Huge memory requirements!
- Used in research, but probably not commercially



Taxonomy – why?

- Good to know if you want to discuss what type of architecture a company want to sell to you
- Good to know if you want to build your own architecture
 - What are the advantages/disadvantages of each...

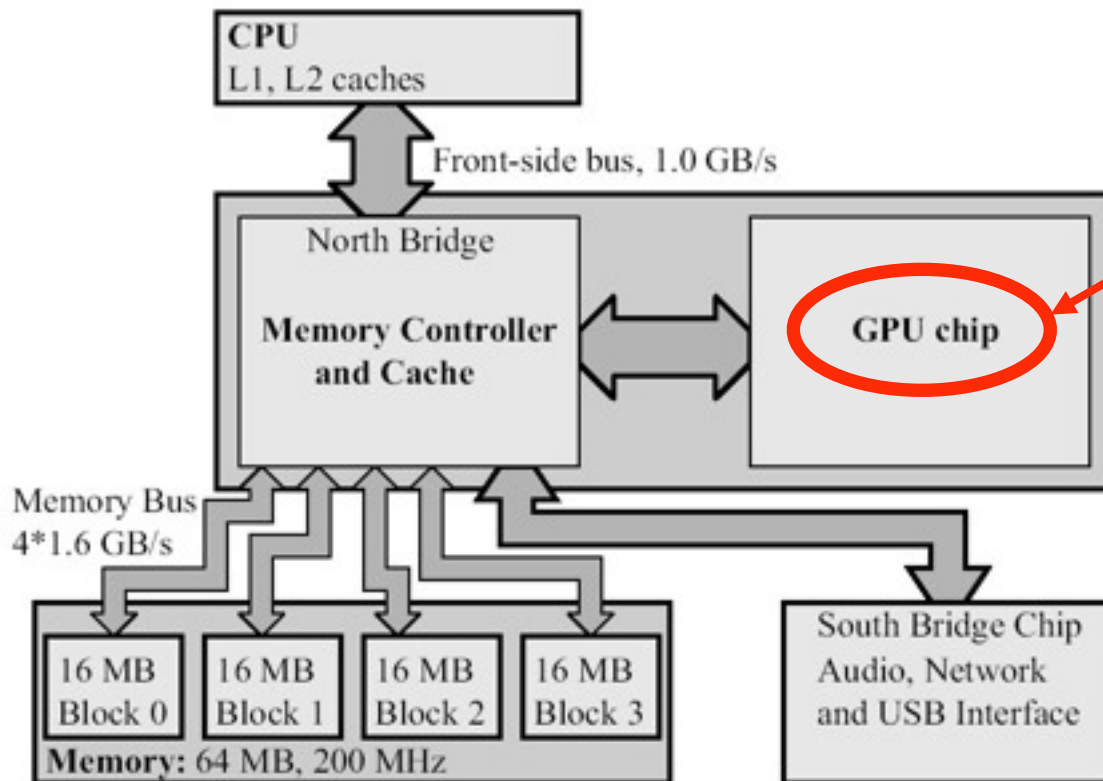
The Xbox game console

- Built by Microsoft and NVIDIA
- Is almost a PC:
 - Pentium III, 733 MHz
 - An extended GeForce3
- Why talk about it here?
 - It resembles some mobile architectures a bit, and there is information available...



Xbox is a UMA machine

- UMA = unified memory architecture
 - Every component in the system accesses the same memory

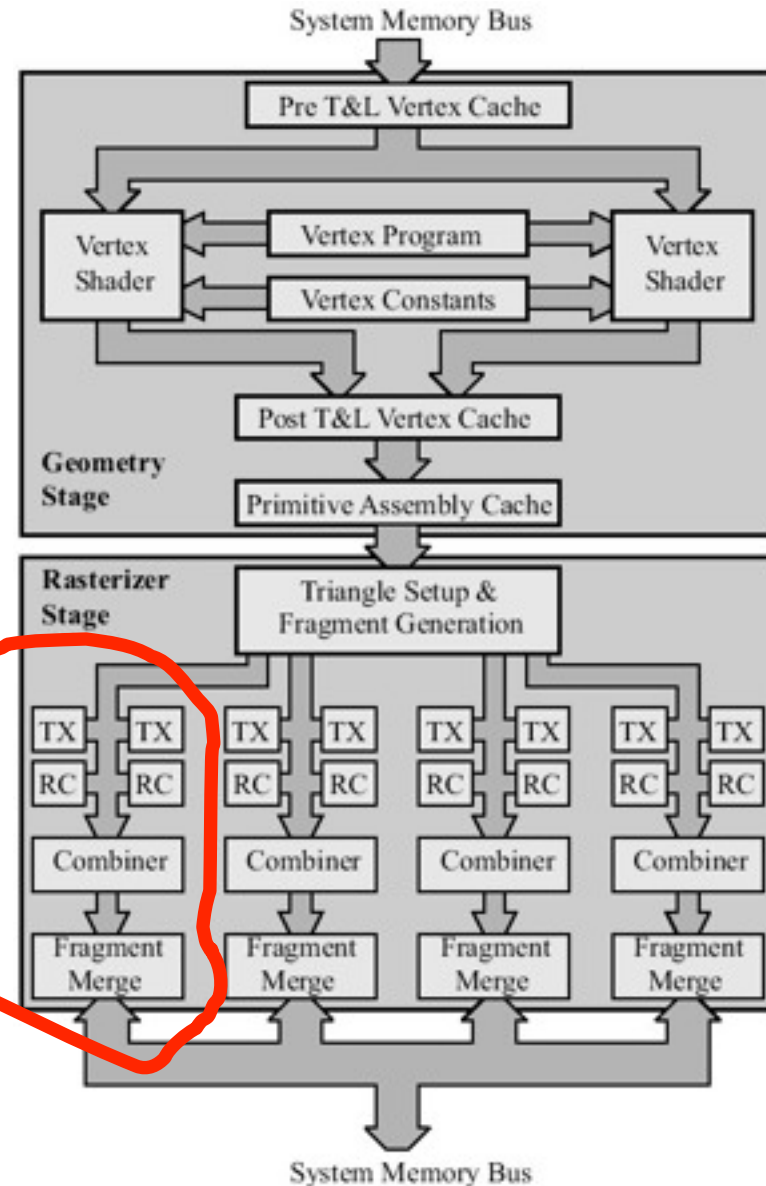


We focus on the GPU

Many mobile graphics architectures are UMA as well

Xbox Graphics Processing Unit

- Supports programmable vertex shaders
 - No fixed-function geometry stage
- Is sort-last fragment architecture
- Rasterizer: handles four pixels per clock
- Runs at 250 MHz

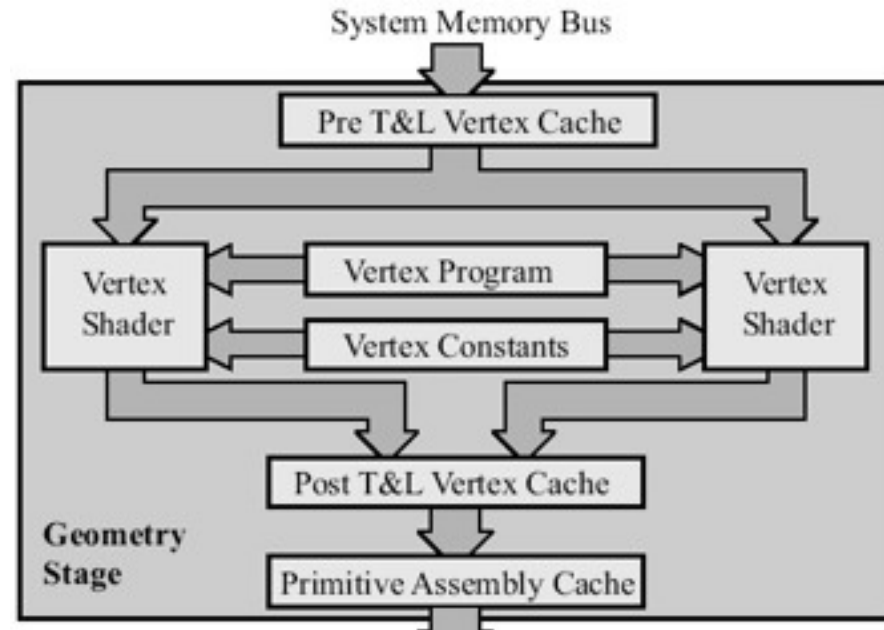


Imagine this is a programmable fragment shader unit, and we have a pretty modern architecture

Xbox

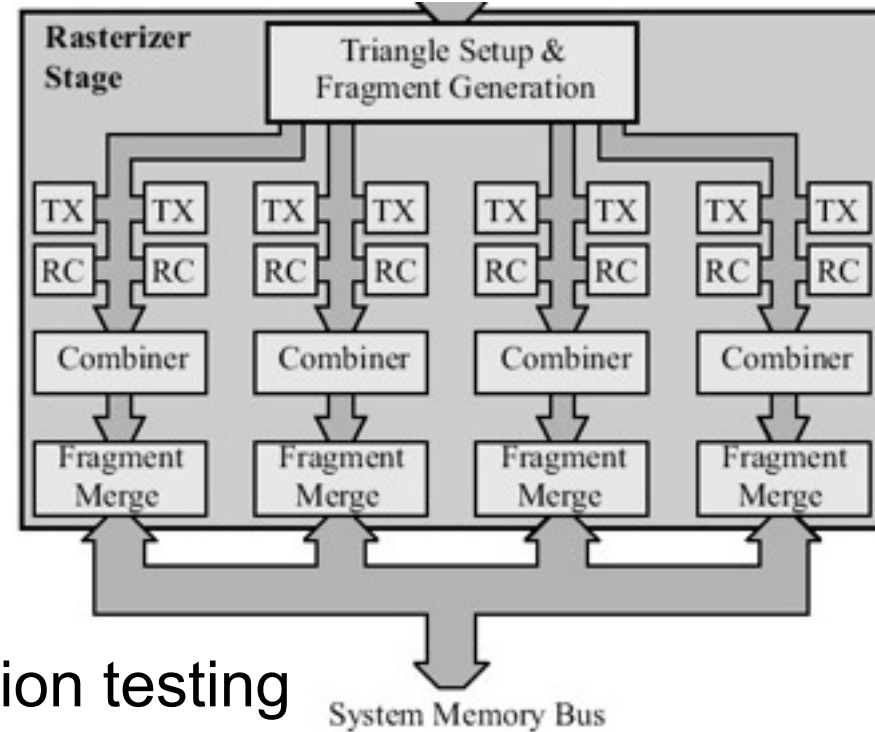
geometry stage

- Dual vertex shaders
 - Same vertex program is executed on two vertices in parallel
- Vertex shader unit is a SIMD machine that operates on 4 components at a time
 - The point is that instead of a fixed function geometry stage, we have now full control over animation of vertices and lighting etc.
- Uses DMA (direct memory access), so that the GPU fetches vertices directly from memory by itself!
- Three different caches – for better performance!



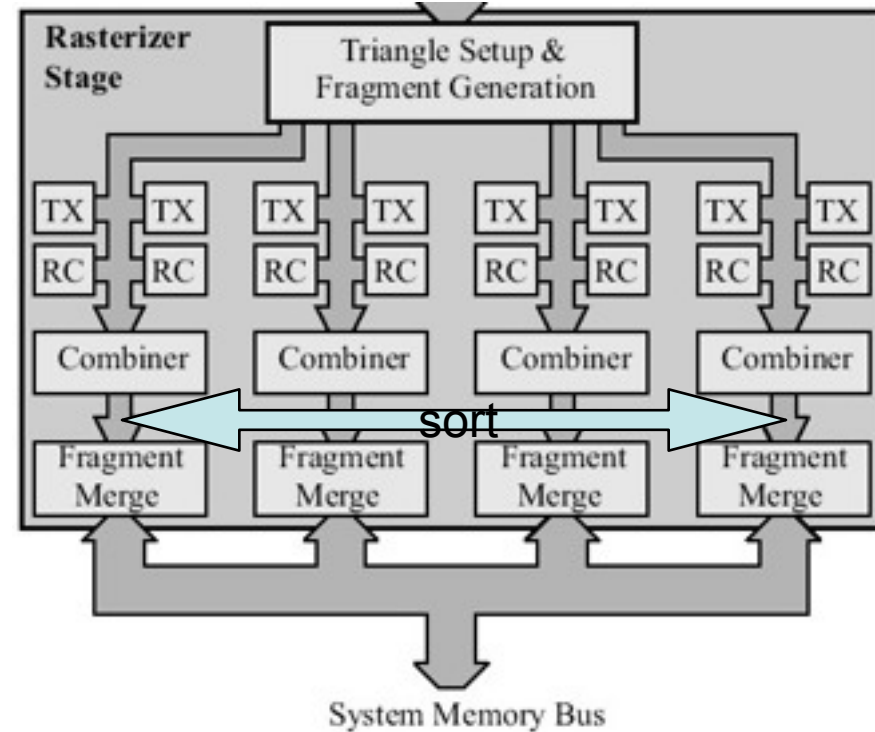
Xbox rasterizer

- First block: triangle setup (TS) and FG
- TS computes various deltas and other startup info
- This block also does Z-occlusion testing
- FG generates fragments inside triangles
 - Tests 2x2 pixels at a time, and forwards these to the four pipelines that follow
 - Note: near edges, not all pixels are inside triangles, and therefore 0-3 pipelines may be idle
 - There are many strategies on how to find which fragments are inside triangle, but exactly how this is done on the XBOX is not known



Xbox rasterizer

- Sorting is done after FG
 - Sort-last fragment arch.
- First: 2 texture units
 - Can be run twice → 4 texture lookups
- Assume TX+RC+Combiner is a "Programmable Fragment Shader Unit"
 - The Xbox only had little flexibility there...
 - Register Combiners (programmable)
- Finally, result from TXs, RCs, shading interpolation, fog interpolation is merged into a final color for that pixel



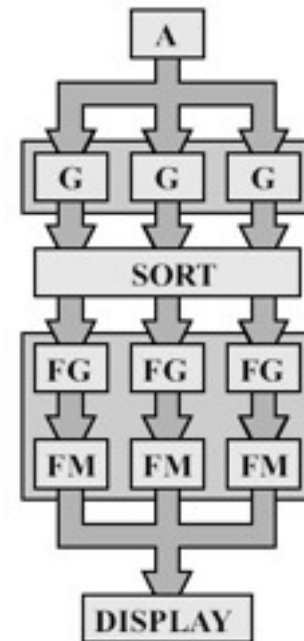
Xbox rasterizer: Fragment merge

- The combiner produced a final color for the pixel on a triangle
- FG merges this with:
 - Color in color buffer (alpha blending)
 - Respect to Z-buffer
 - Stencil testing
 - Alpha testing
- Z-compression and decompression is handled here as well
- Writes final color over the system memory bus

Questions on XBOX?

Tiling architectures

- There is an ongoing battle between traditional architectures (what we've looked at so far) and tiling architectures
 - Which is best?
 - Depends...
- Sort-Middle architecture!

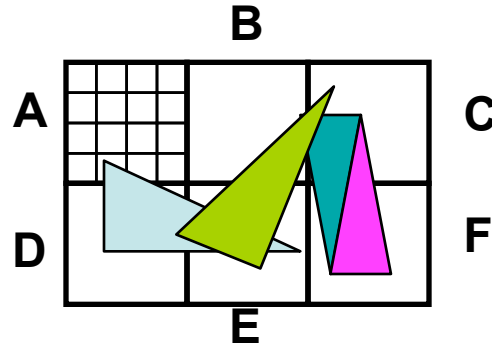


Examples of existing tiling architectures







- Imagination technologies
 - Kyro II (PC graphics card)
 - Sega Dreamcast
 - PowerVR MBX (for mobiles)
 - PowerVR SGX (for mobiles)
- Falanx (Norway) [now owned by ARM]
 - Mali architectures
 - For mobiles as well
- Intel Larrabee
 - Software Tiled based rasterizer
 - Coming soon...

Tiling: basic idea (1)

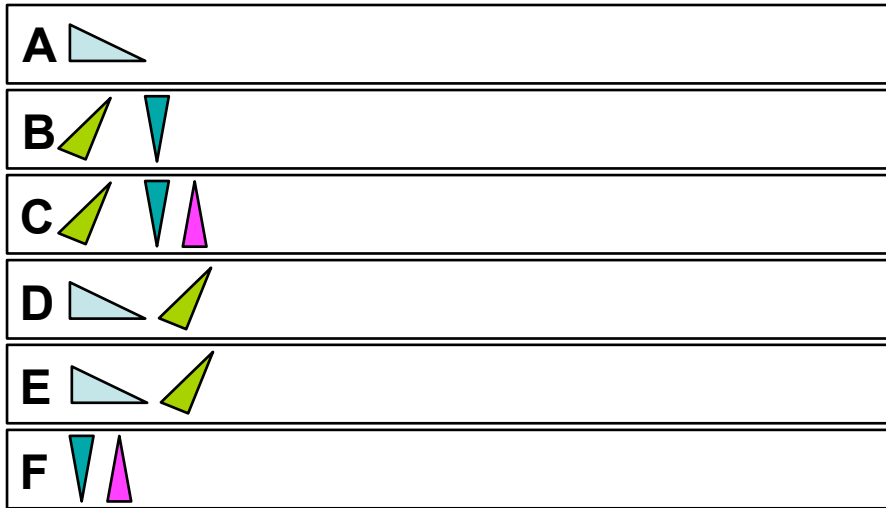
- Apply vertex shader (incl. projection) to vertices



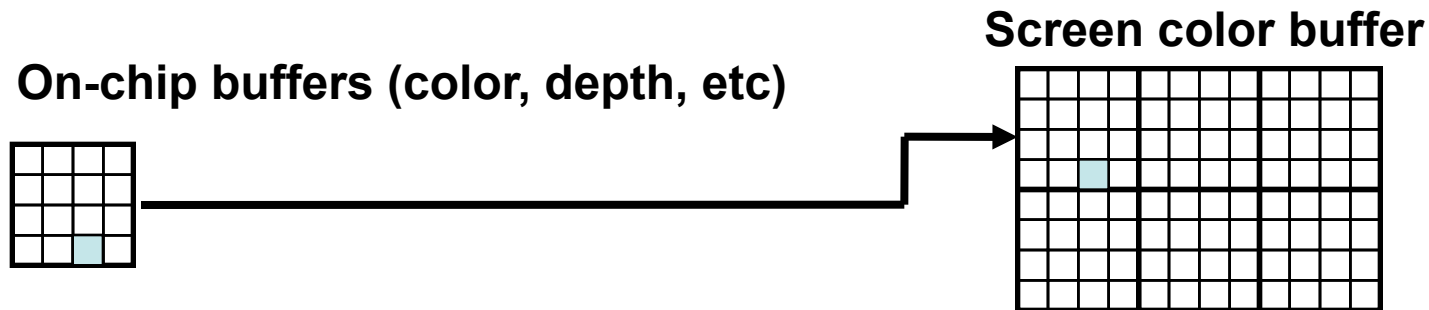
- Create a triangle list for each tile
 - Holds pointers to all triangles overlapping a tile

A	
B	 
C	  
D	 
E	 
F	 

Tiling: basic idea (2)



- Process one tile at a time, and rasterize triangles in tile's triangle list

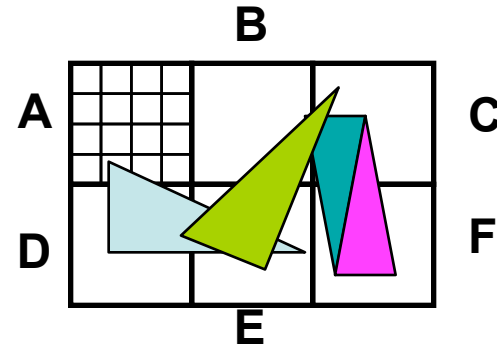


Start with tile A's triangles

When all triangles in list have been processed, copy on-chip color buffer to screen color buffer

Need to write out depth buffer as well! Because, the programmer may want to access it

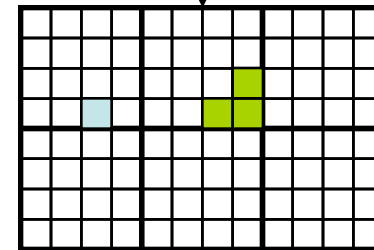
Tiling: basic idea (3)



On-chip buffers (color, depth, etc)



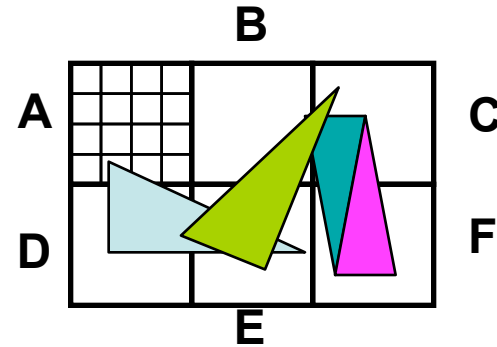
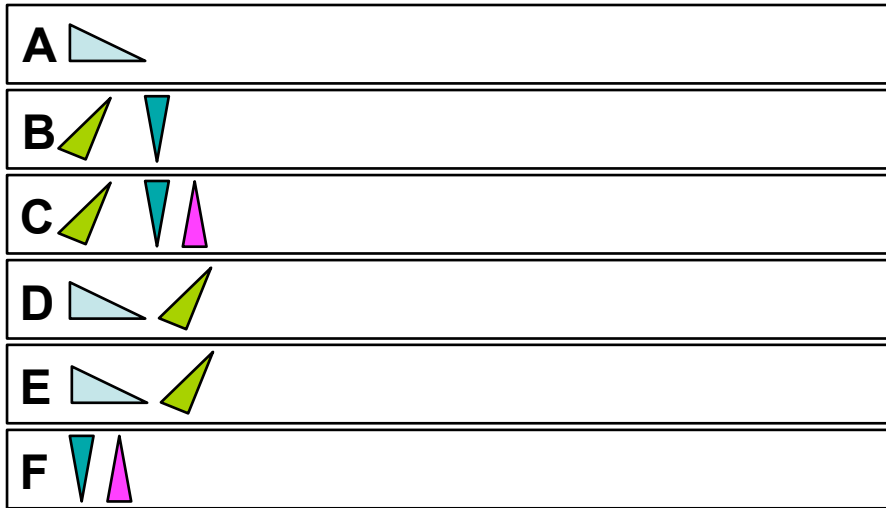
Screen color buffer



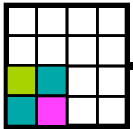
Then tile B's triangles

When all triangles in list have
been processed, copy on-chip color buffer
to screen color buffer

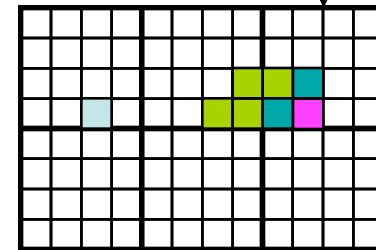
Tiling: basic idea (4)



On-chip buffers (color, depth, etc)



Screen color buffer



Then tile C's triangles

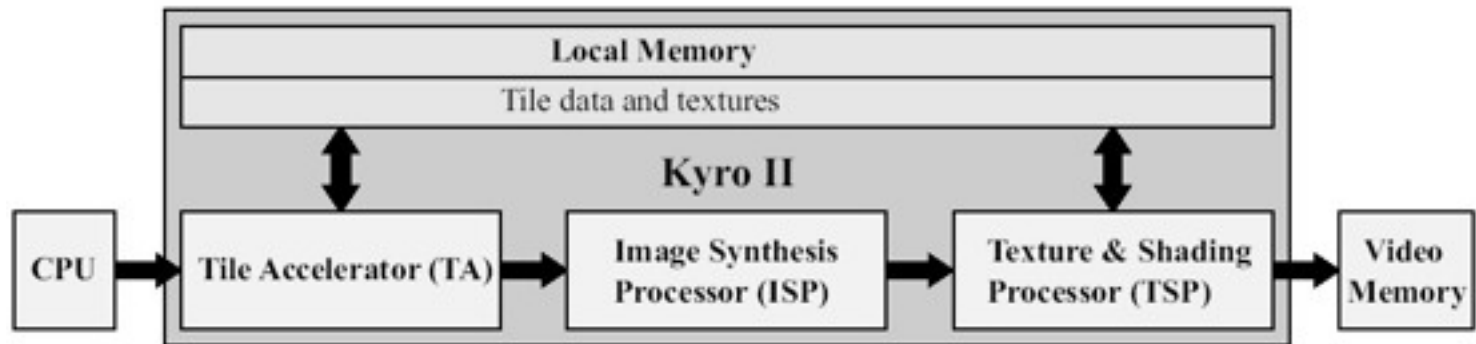
When all triangles in list have
been processed, copy on-chip color buffer
to screen color buffer

**AND
SO
ON...**

KYRO – a PowerVR-based tiling architecture

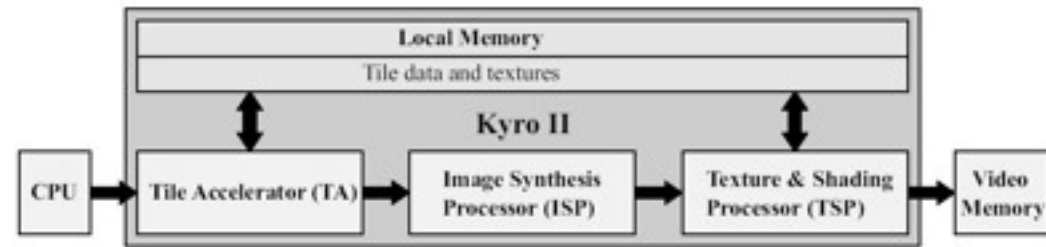
- For KYRO II: tile=32x16 pixels
- Tiling advantage: can implement temporary color, stencil, and Z-buffer in fast on-chip memory for a single tile
- Saves memory and memory bandwidth!
 - Claims to save 2/3 of bandwidth compared to traditional architecture (without Z-occlusion testing, no buffer compression...)

KYRO architecture overview



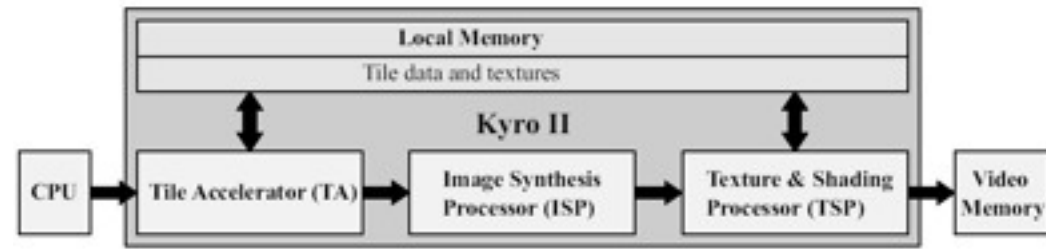
- CPU sends triangle data to KYRO II
- Tile Accelerator (TA)
 - Needs an entire scene before ISP and TSP blocks can start
 - So TA works on the next image, while ISP and TSP works on the current image (i.e., they work in a pipelined fashion)
 - TA sorts triangles, and creates a list of triangle pointers for each tile (for tris inside tile)

KYRO



- Tile accelerator:
 - When all triangles for entire scene are sorted into tiles, the TA can send tile data to next block: ISP
 - And the TA then continues on the next frame's sorting in parallel
- Image synthesis processor (ISP):
 - Implements Z-buffer, color buffer, stencil buffer for tile
 - Depth testing:
 - Test 32 pixels at a time against Z-buffer
 - Records which pixels are visible
 - Groups pixels with same texture and sends to TSP
 - These are guaranteed to be visible, so we only texture each pixel once

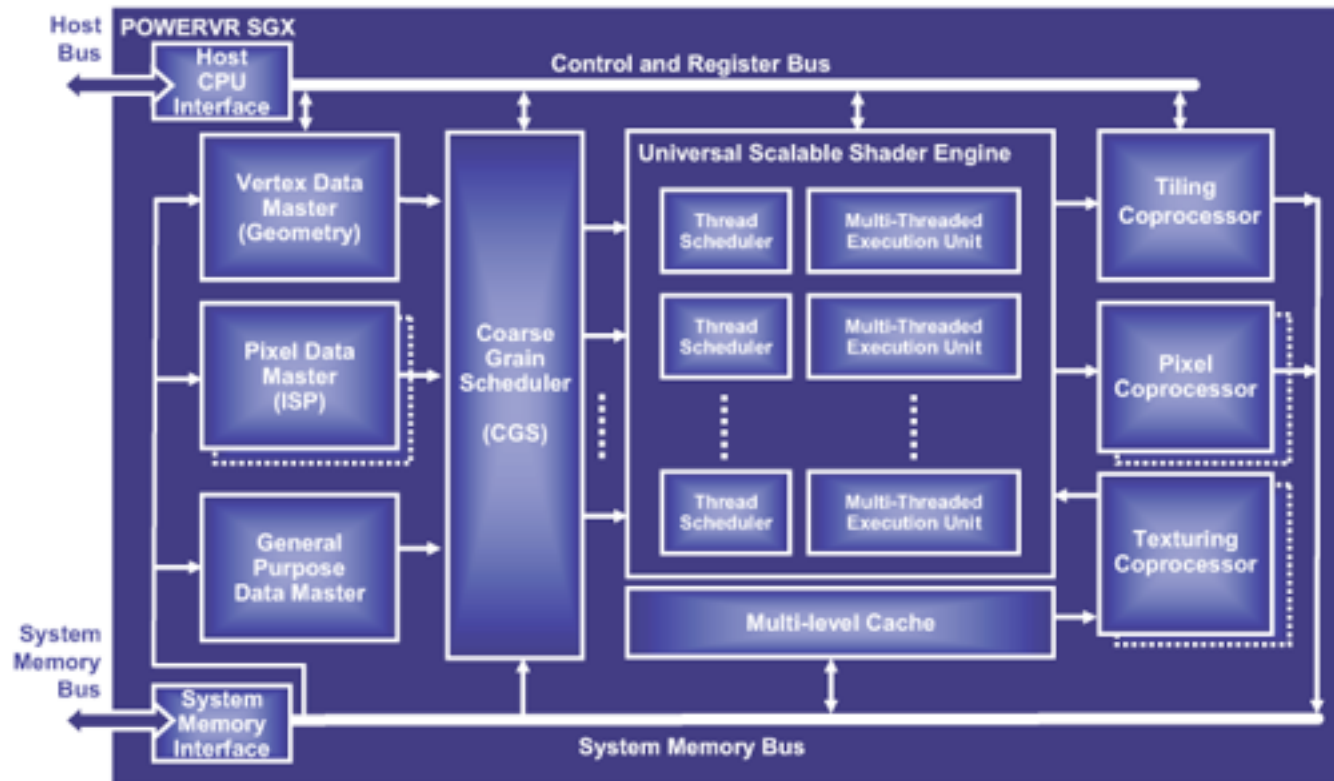
KYRO: TSP



- Texture and Shading Processor (TSP):
 - Handles texturing and shading interpolation
- Has two pipelines that run in parallel
 - 2 pixels per clock
- Can use 8 textures at most
 - Is implemented by "looping" in TSP
- Texture data is fetched from local memory
- Supersampling: 2x1, 1x2, and 2x2
 - Renders a larger image and filters and scales down
 - For 2x2: Need only 4x the size of tile (or rather, render 4x as many tiles, i.e., need not 4x memory)

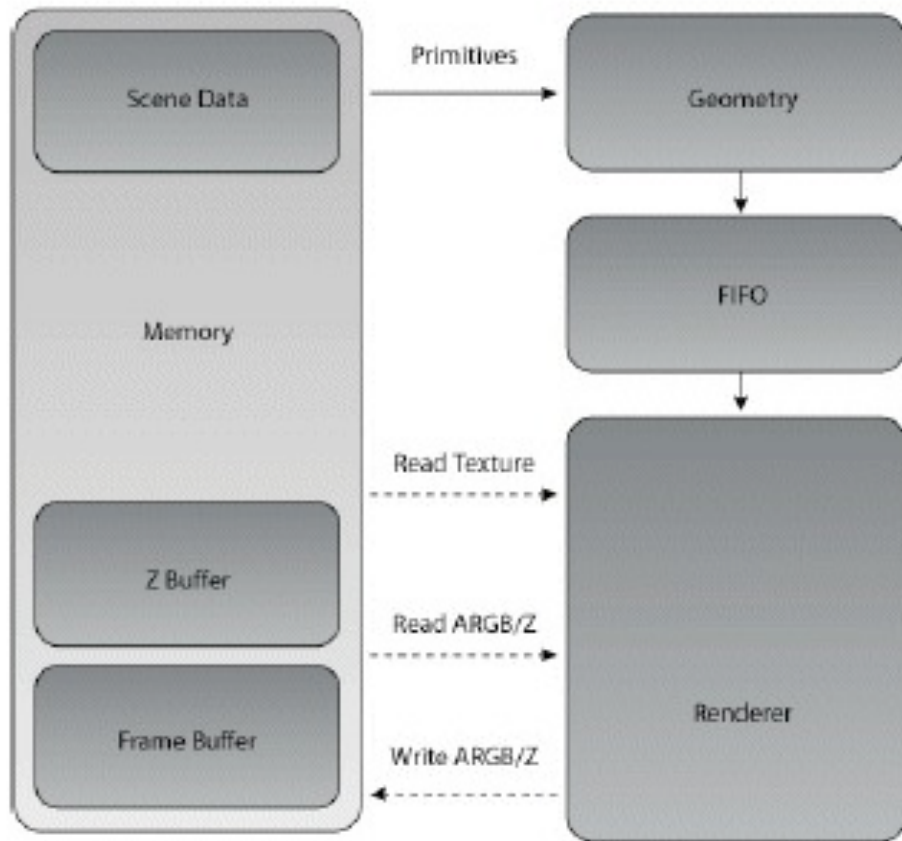
PowerVR MBX and SGX

- Kyro was for the PC market...
- MBX and SGX are designed for the mobile market
- Tiling: creates depth buffer first, then applies pixel shader
 - Does not apply pixel-shader for hidden fragments: called "deferred shading"
- Uses PVR Texture Compression
- SGX (newest): uses unified shaders
 - Programmable vertex and pixel shaders

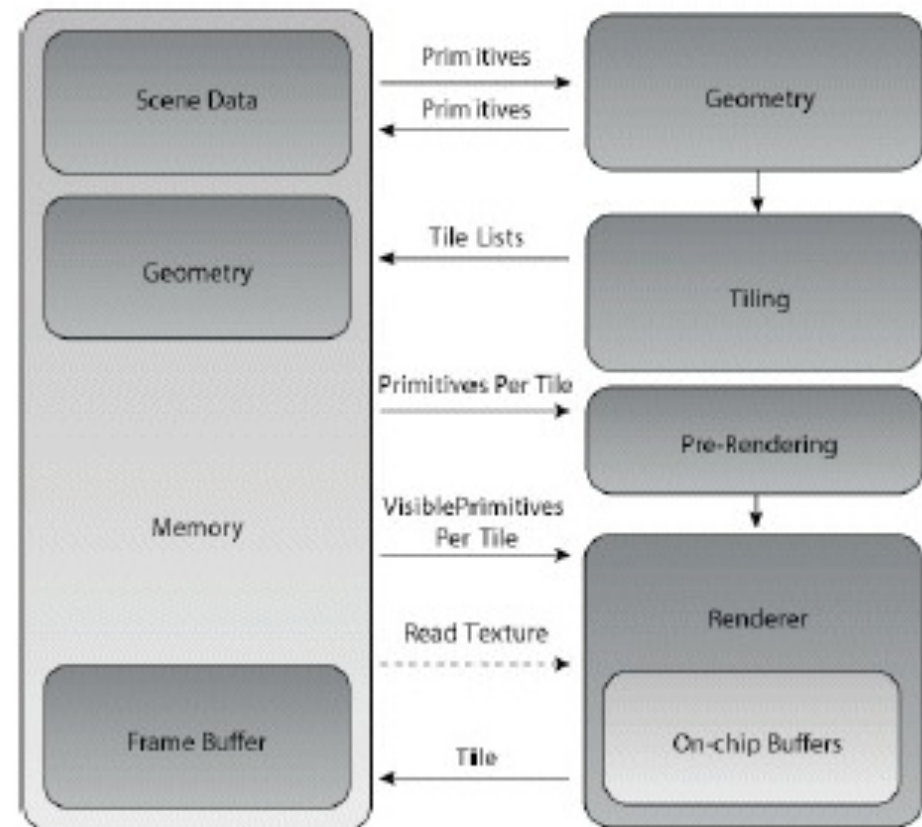


Traditional vs Tiling

Traditional Immediate Mode



Traditional Tile-Based Mode

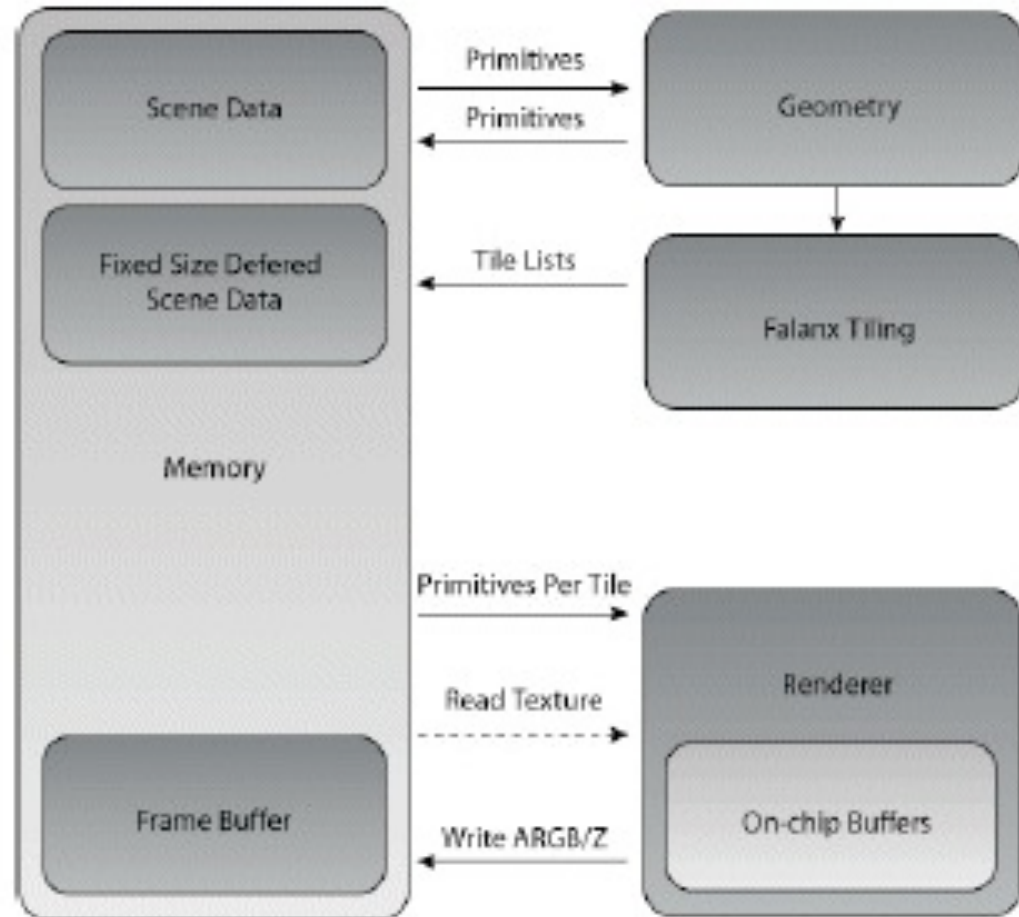


Falanx Mali Architecture

- Tiling architecture (tile size=16x16)

- Does **not** create Z-buffer first, and then apply pixel shader (as do PowerVR)
- Instead two other techniques to reduce BW:
 - Efficient Zmax-culling
Eliminates ~50% of occluded pixels [this varies, of course...]
 - 2 bits-per-pixel texture compression (FLXTC)
- Plus fantastic antialiasing (more about that next lecture)

Falanx Architecture



Questions on tiling architectures?

Bitboys' G40 architecture

- Not a tiling architecture



Bitboys, bought by ATI, ATI bought by AMD, AMD sold handheld graphics to Qualcomm

G40 - Main development guidelines

- Target volume market mobile phones in 2007-2010 timeframe
 - We expect 3D graphics breakthrough in mobile phones in 2006 timeframe – Japan first, then Europe, followed by US
- Industry standard content creation tools and game art will be largely based on the use of shaders
 - Don't want to stray from this path
- Scene complexity and performance target
 - 60 FPS
 - 20-30k polygons/frame
 - QVGA or VGA display resolution
 - Depth complexity 5
 - Relatively complex pixel shaders
 - High sustained pixel fillrate



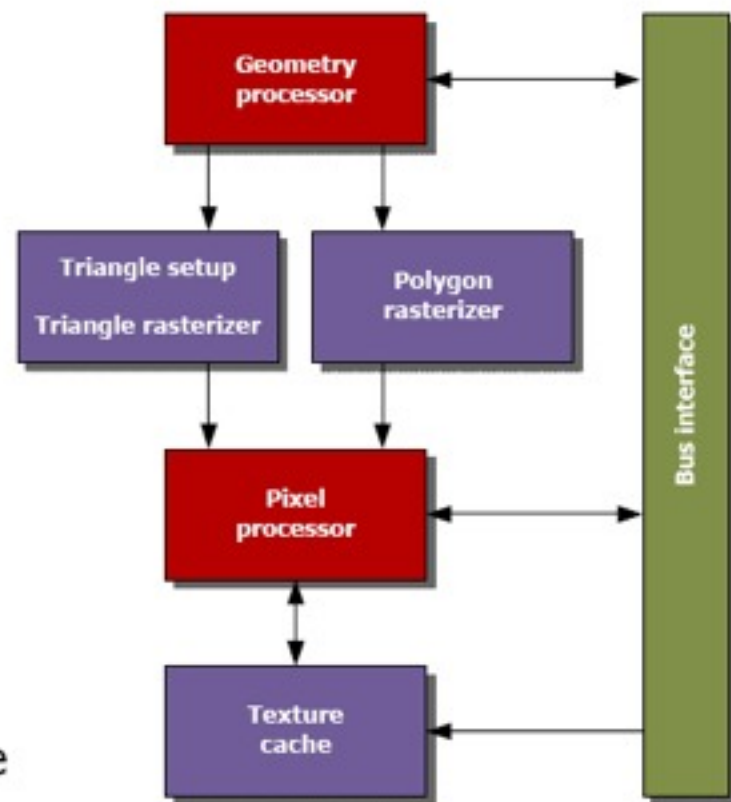
G40 – Rendering features

- 2D graphics rendering
 - BitBlts, fills, ROPs (256)
 - Small separate core for rendering bitmap-based user interfaces
- Vector graphics rendering
 - SVG Basic level feature set, targeting OpenVG
 - Anti-aliased rendering of concave and convex polygons
 - Rasterization integrated into the 3D pipeline
 - Support for linear and radial gradients
 - Arbitrary clip paths
 - 10-50x performance over software rendering
- 3D graphics
 - Transformation and lighting in hardware
 - Floating-point vertex and pixel shaders
 - Multitexturing: Four textures per pixel
 - Fully programmable architecture, no fixed-function pipeline
 - FLIPQUAD full-screen anti-aliasing
 - PACKMAN hardware texture decompression



Architecture

- Rendering pipeline based on OpenGL® 2.0 shader architecture
- Fully floating-point, programmable, well integrated architecture
- Fixed function fully emulated using the programmable pipeline
- Designed from ground up to power mobile phones and other handheld devices



What can we expect?



To tile or not to tile? (1)

- Tiling:
 - Good:
 - Uses a small amount of very fast memory
 - $16 \times 16 \times (4 \text{ (color)} + 4 \text{ (depth)} + 1 \text{ (stencil)}) = 2304$ bytes
 - Bandwidth to z-buffer, color buffer, stencil for free
 - Design is parallel (add more pipes)
 - Bad:
 - More local memory is needed (for tile sorting)
 - Bandwidth increase since geometry needs sorting
 - State changes: what happens when you need to switch from one long pixel-shader program to another? (several times per tile...)
 - Harder to predict performance
 - A triangle covering several tiles, is processed several times
 - E.g., triangle setup...
 - Ugly:
 - Amount of local memory places a limit on how many triangles can be rendered
 - Overflow?
 - 3 MB can handle a little over 30,000 triangles (info from Kryo II)

To tile or not to tile? (2)

- Traditional:
 - Good:
 - More straightforward to implement
 - Easier to predict performance
 - Other bandwidth reduction techniques can be used (buffer compression, zmin/zmax-culling, e.g.)
 - No (big) problems with state changes
 - Bad:
 - Cannot have the entire framebuffer in on-chip memory
 - Buffer accesses costs!
 - Have to execute pixel shader unnecessarily for some occluded fragments [but Zmax/Pre-Z pass can help]
 - Ugly:
 - Too traditional – not that exotic...

To tile or not to tile? (3)

- Who knows?
 - Definitely scene-dependent!

Unified shader architecture

- First on XBOX 360 (2005)
 - DirectX10 uses unified shader model
 - All PC graphics unified from Radeon 2xxx and GeForce 8xxx up
- iPhone's PowerVR SGX
- Made possible since vertex shaders and fragment shader instruction set converged
- Basic idea:
 - Have n "unified shader units"
 - Can be used for either vertices or fragments
- Why?
 - Load balancing!
 - Increasing vertex processing

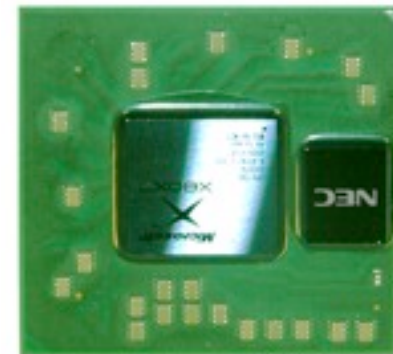
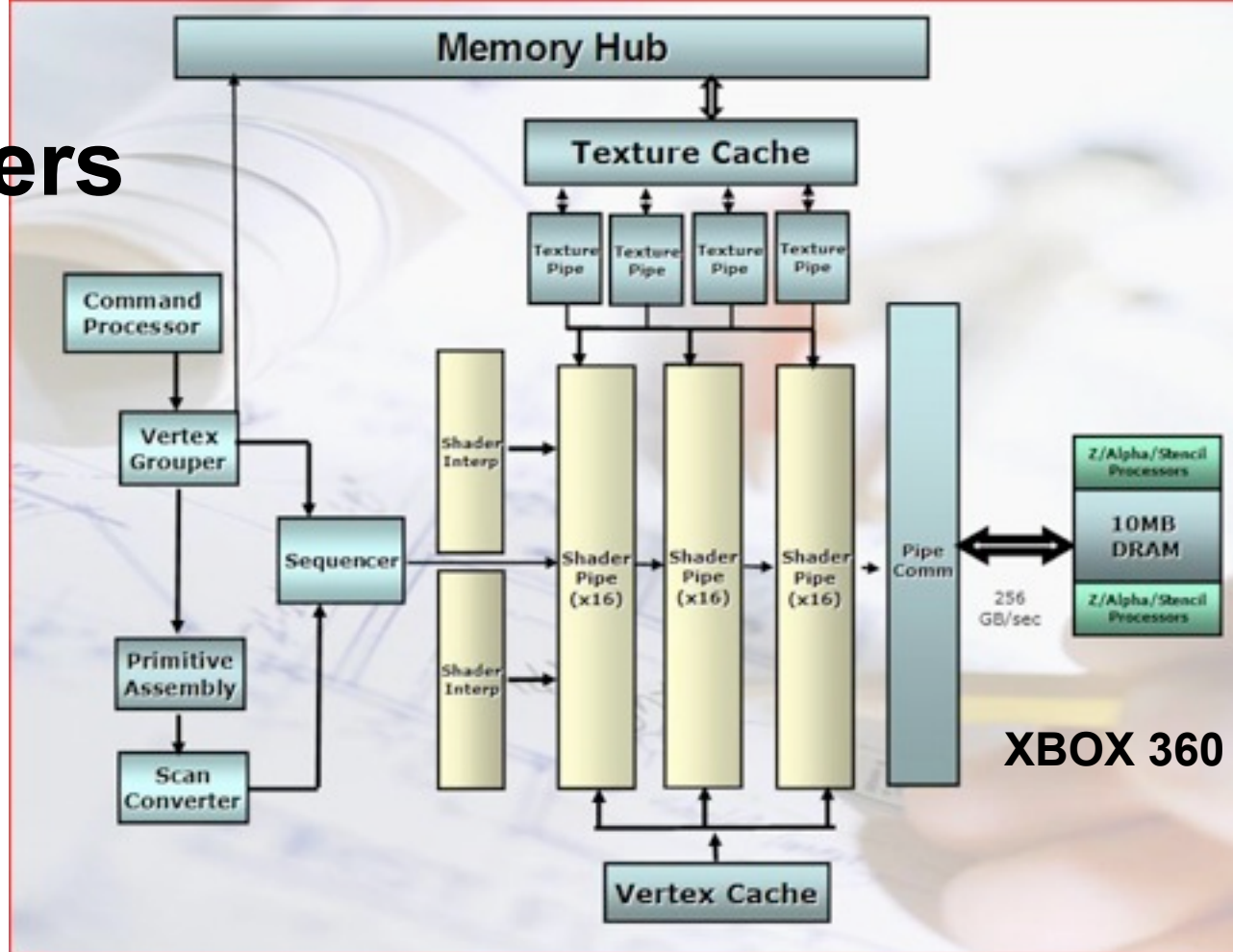
Unified Shader: XBOX 360

- 48 unified shader units
 - e.g 16 used for vertices, and 32 for fragments
 - Scheduling is dynamic
 - Split depends on what is being rendered
- Consequence: **can** run at ~100% efficiency all the time, rather than having some pipeline instructions waiting for others
 - Non-unified shader pipeline based high-end PC chips can run at 50-60% efficiency.



Unified shaders

- Sequencer's goal:
 - Keep all 48 unified shader units busy ~100% of the time in the most efficient way possible.
 - Threads are grouped in sets of 64 and run together
 - If a thread needs to wait for data, another thread can be processed and sent to the unified shader units.



ATI Radeon 2900 (R600)

- Merge the XBOX360 unified shader with X1800 (R520) series
- Support Microsoft's DirectX Direct3D 10
 - Unified shader language
 - New Geometry Shader pipeline stage
 - Supported on mobiles chips
- Launched 2007

R600 Top Level

Red – Compute

Yellow – Cache

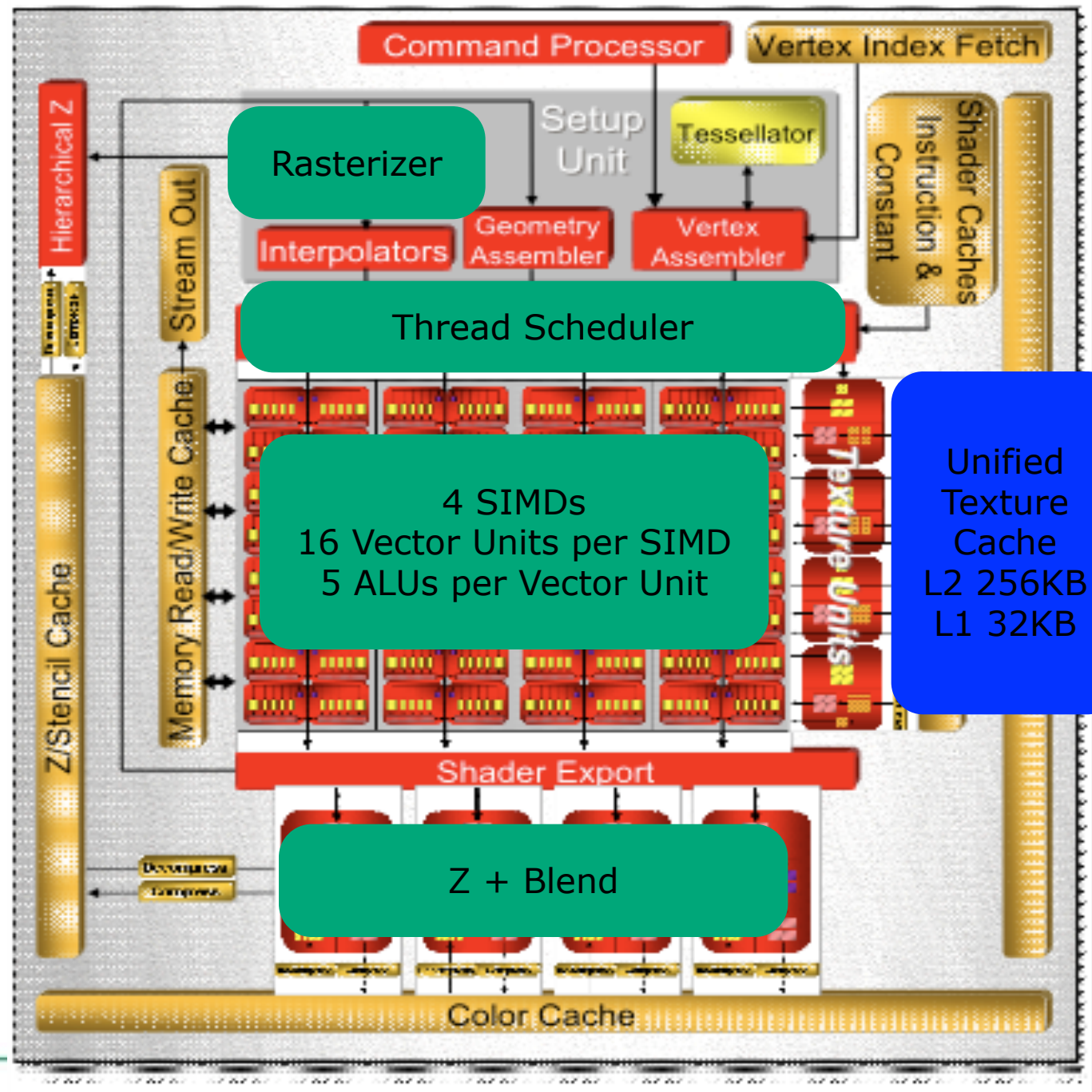
Unified shader

Shader R/W

Instr./Const. cache

Unified texture cache

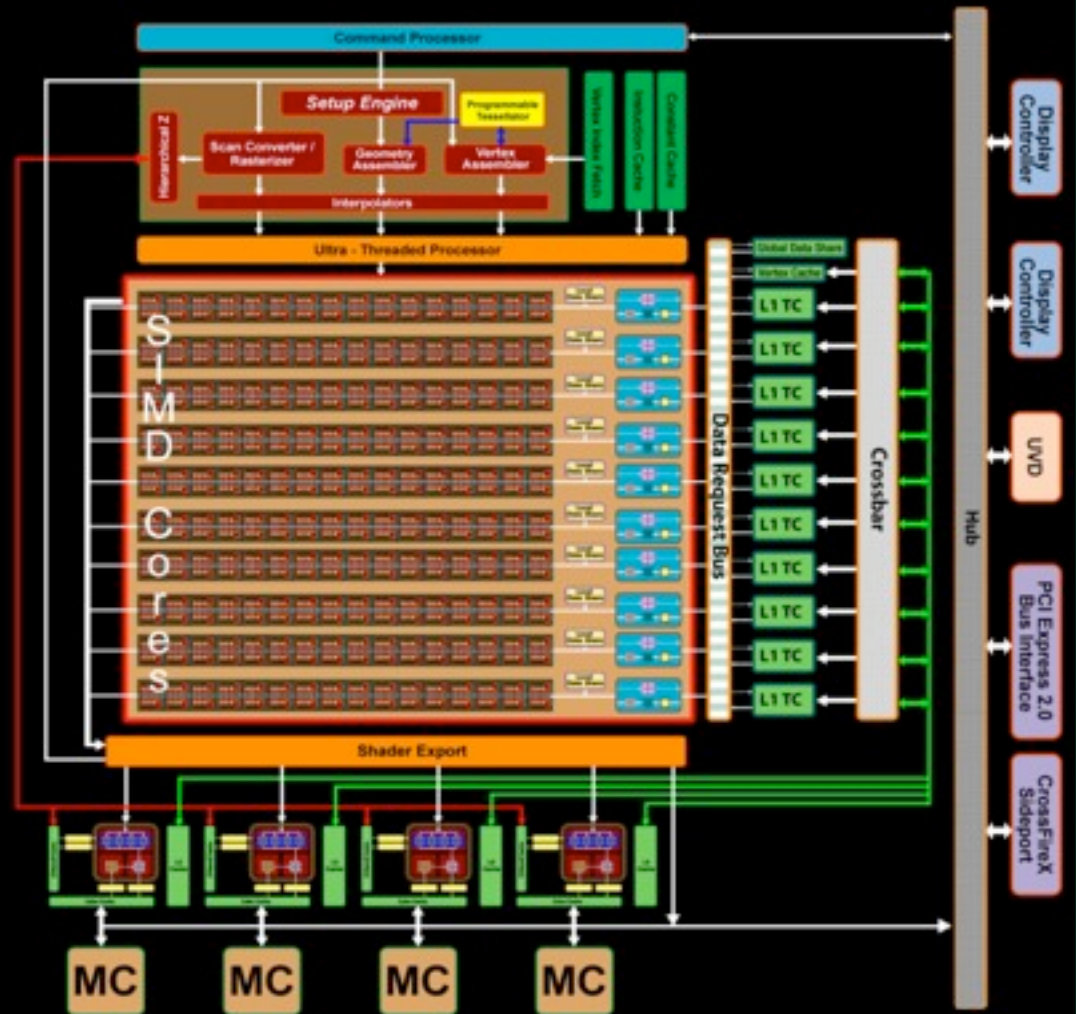
Compression



ATI Radeon™ 4800 Series Architecture

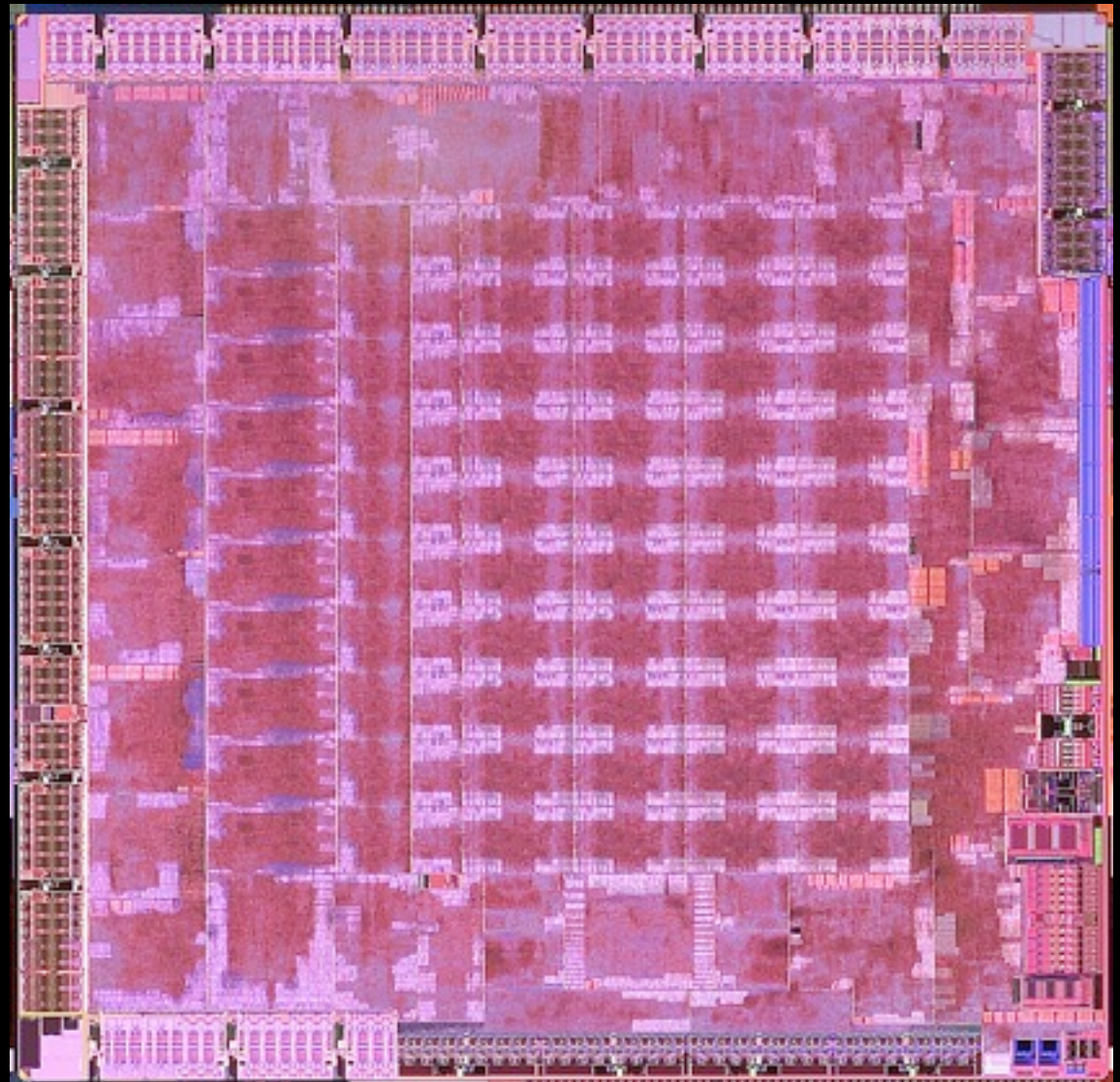
- 10 SIMDs
- 1.2 teraFLOPS

$$= 10 \text{ SIMDs} \times 16 \text{ ALUs} \times 5 \text{ (32bit FP)} \times 2 \text{ ops (muladd)} \times 750 \text{ MHz}$$



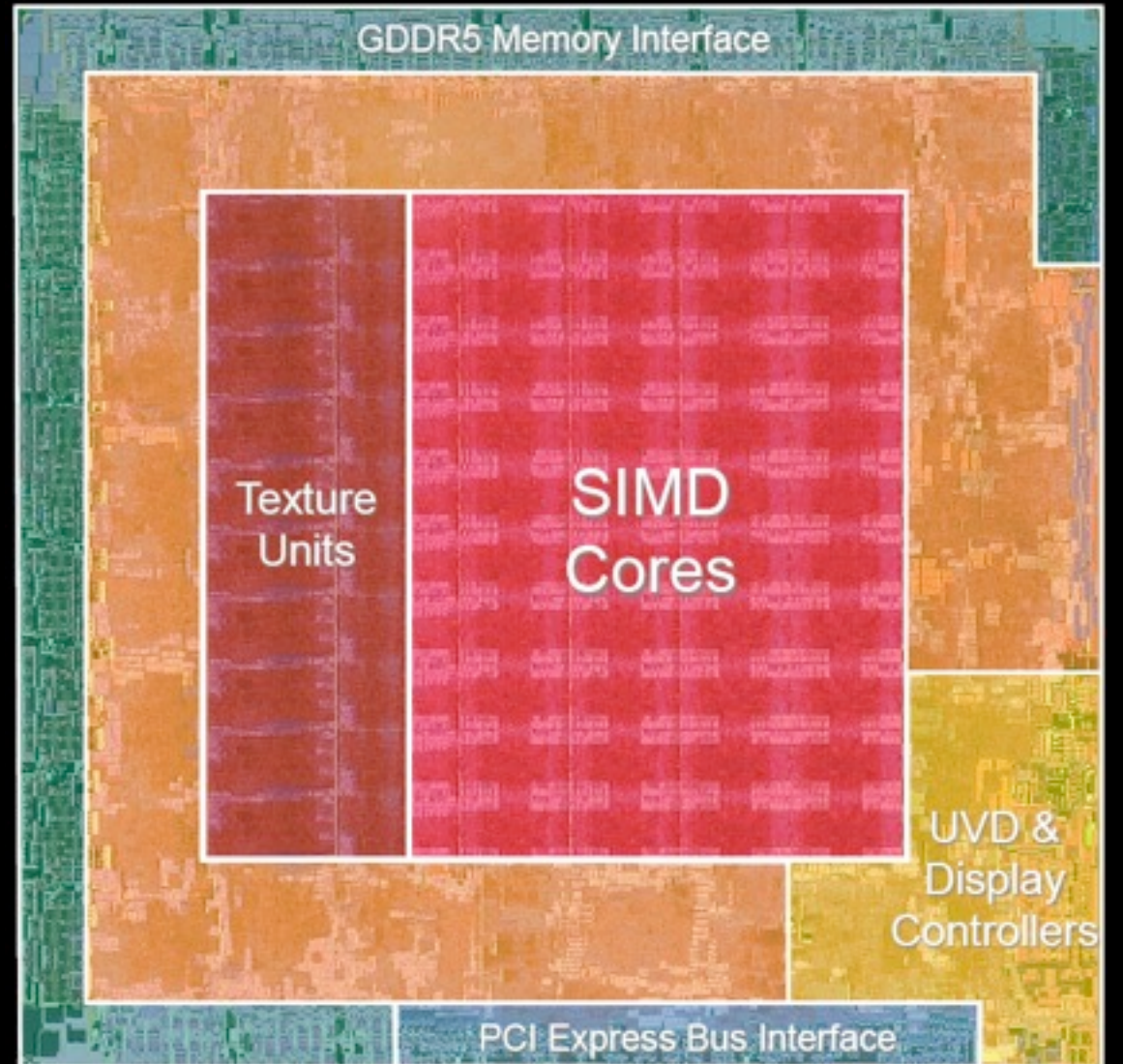
Radeon 4800 Series (R770)

- Launched 2008
- 260mm²
- 956 MTransistors



Radeon 4800 Series

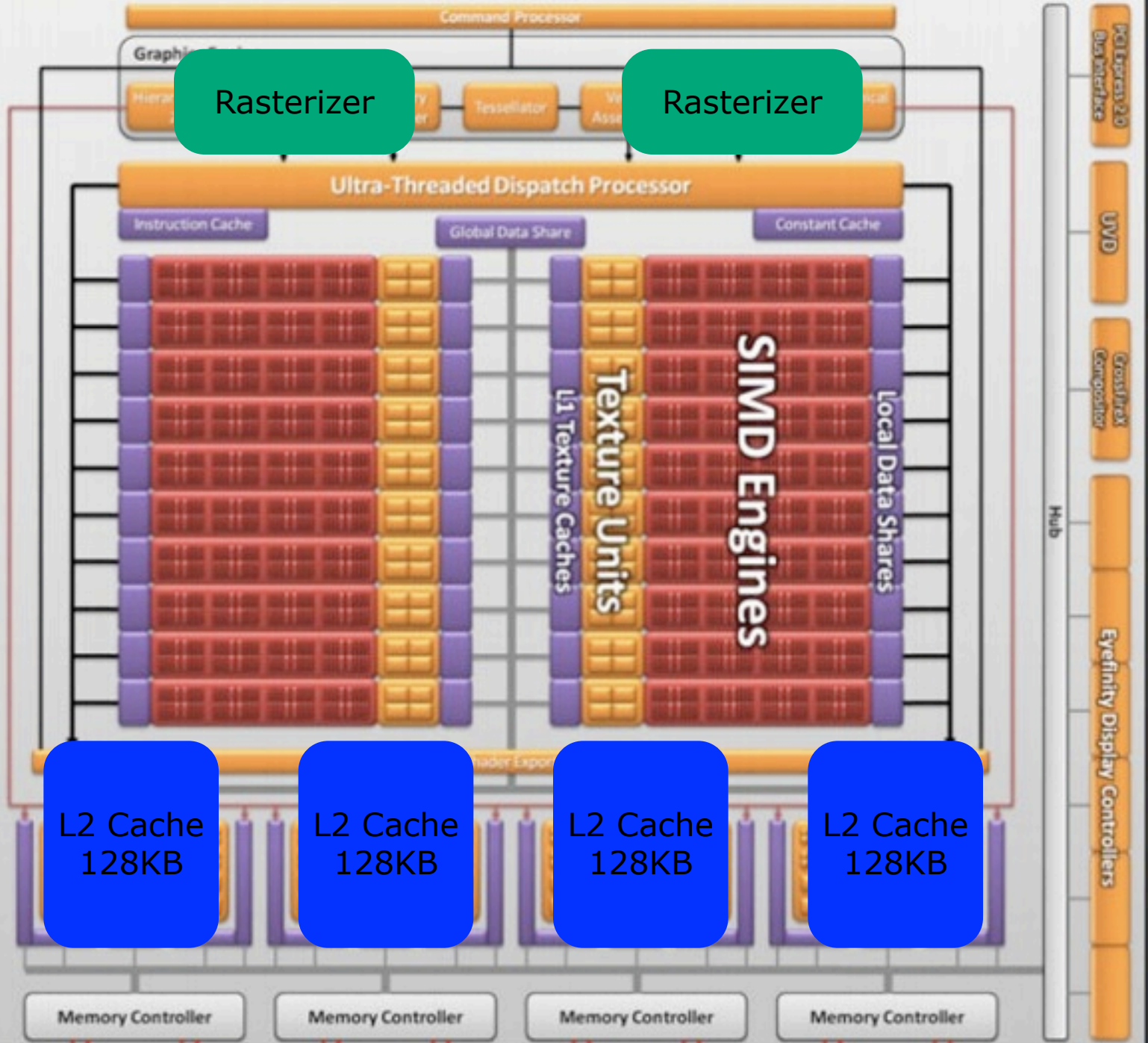
- Launched 2008
- 260mm²
- 956 MTransistors
- 64 z/stencil
- 40 texture
- 10 SIMDs
 - Increasingly larger percentage of chip



ATI Radeon 5870 (R800)



- Released 23 September 2009
- First Direct3D 11 GPU
- 2 x 10 SIMDs



Rasterizer

Rasterizer

L2 Cache
128KB

L2 Cache
128KB

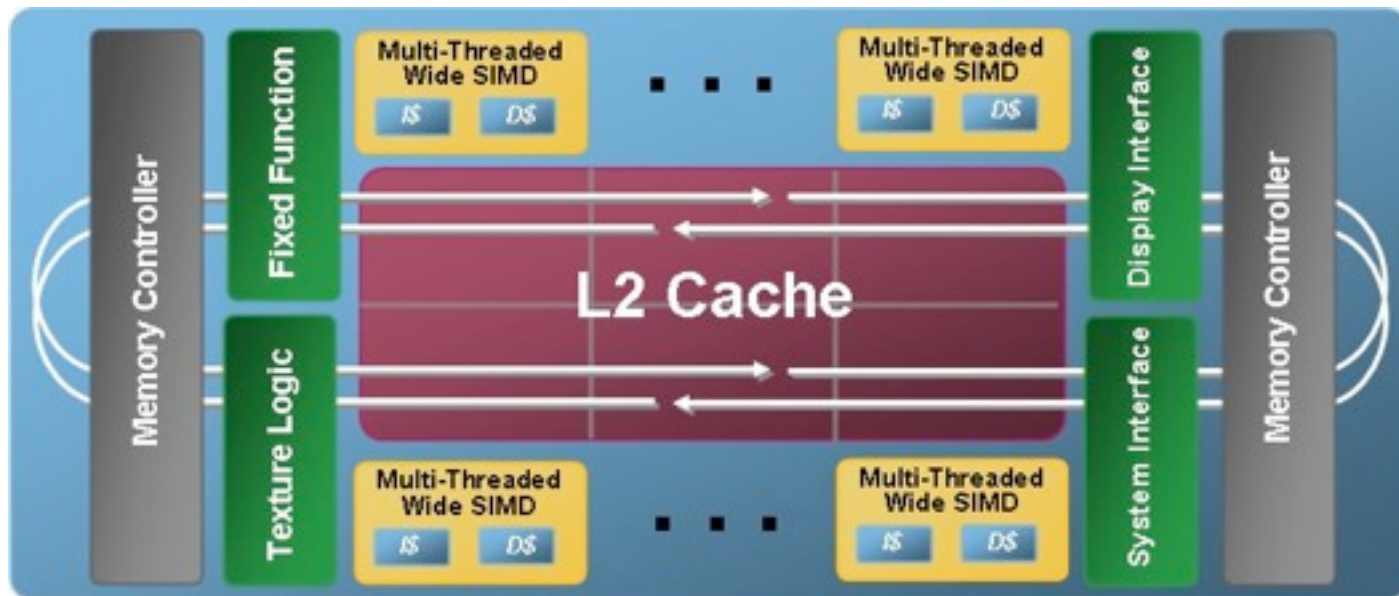
L2 Cache
128KB

L2 Cache
128KB

from <http://www.anandtech.com/video/showdoc.aspx?i=3643&p=5>

Intel's Larrabee

- n (16-32) Pentiums with 16 wide 32bit SIMD
- 1024 bit ring bus



The end