



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Departament d'Arquitectura de Computadors

Tarjetas Gráficas y Aceleradores

Attila

Attila team (attila.ac.upc.edu)

Departament d'Arquitectura de Computadors

Facultat d'Informàtica de Barcelona

Universitat Politècnica de Catalunya



Attila

□ Proyecto Iniciado en 2003

□ Investigación en GPUs

- Centrado en la microarquitectura

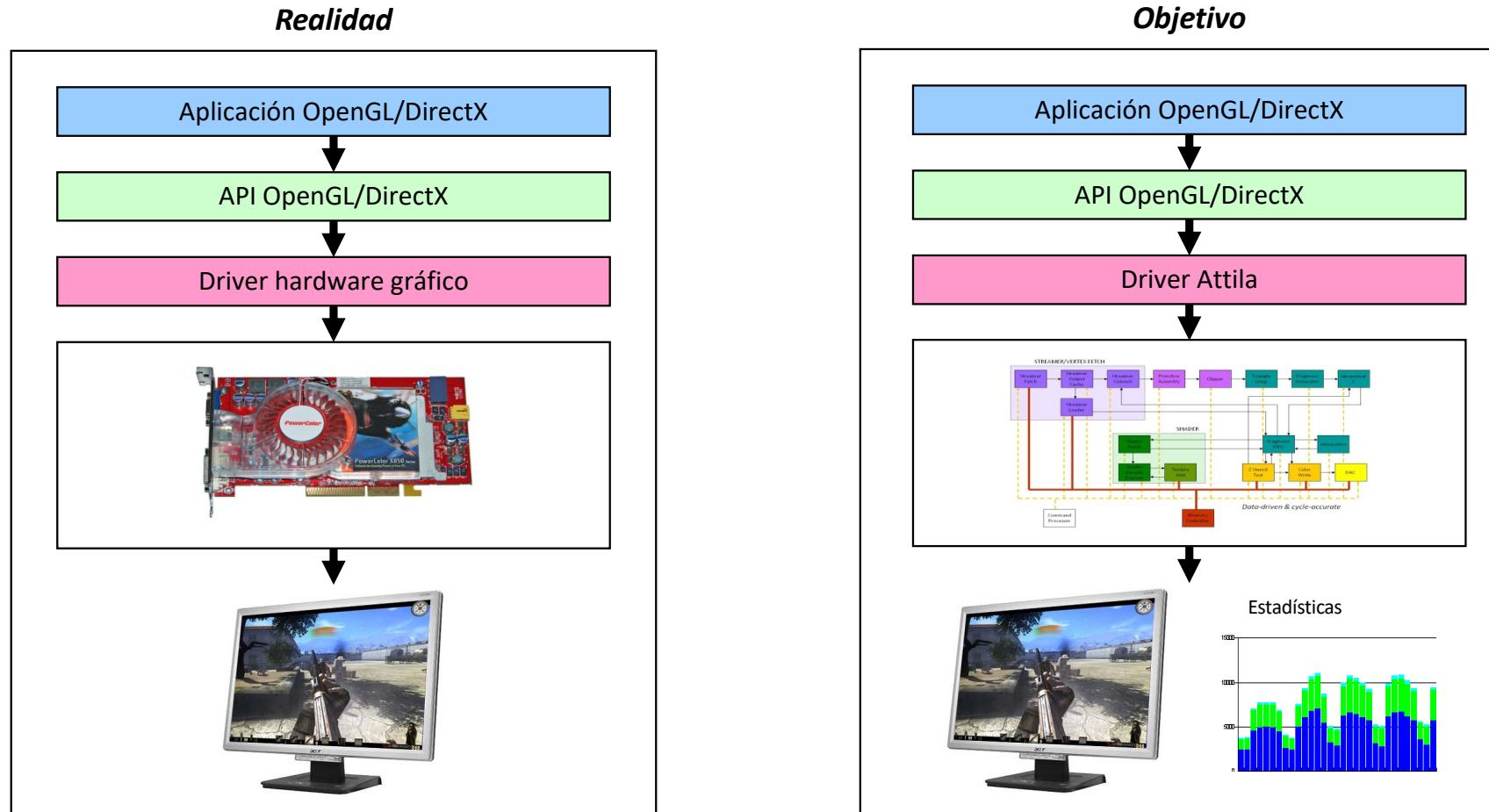
- Uso de juegos reales como benchmarks

- Análisis de la relación entre ancho de banda/latencia/paralelismo

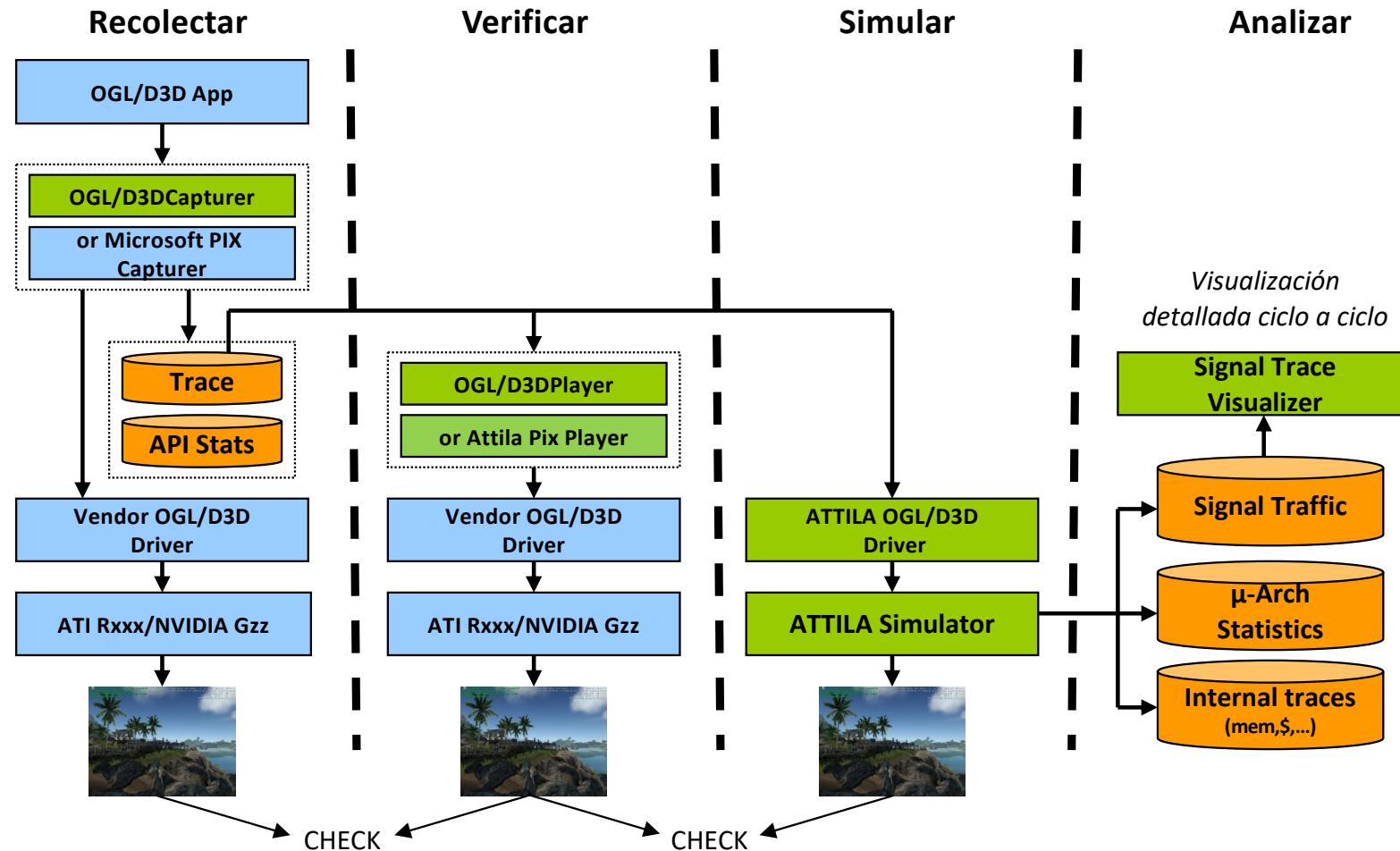
□ ¿Qué tarjetas había en 2003?

	Code Name	Tecn. Fab.	Bus	Mem (MB)	Clock (MHz)		Shaders				Memory			API compliance	
					Core	Mem	VS	FS	TU	ROP	GB/s	Bus type	Bus width	DirectX	OpenGL
GeForce FX 5950 Ultra	NV38	130 nm	AGP x8	256	475	950	3	4	8	4	30,4	GDDR3	256 bits	9.0a	1.5
Radeon 9800XT	R360	150 nm	AGP x8	256	412	730	4	8	8	8	23,36	DDR	256 bits	9.0	2.0

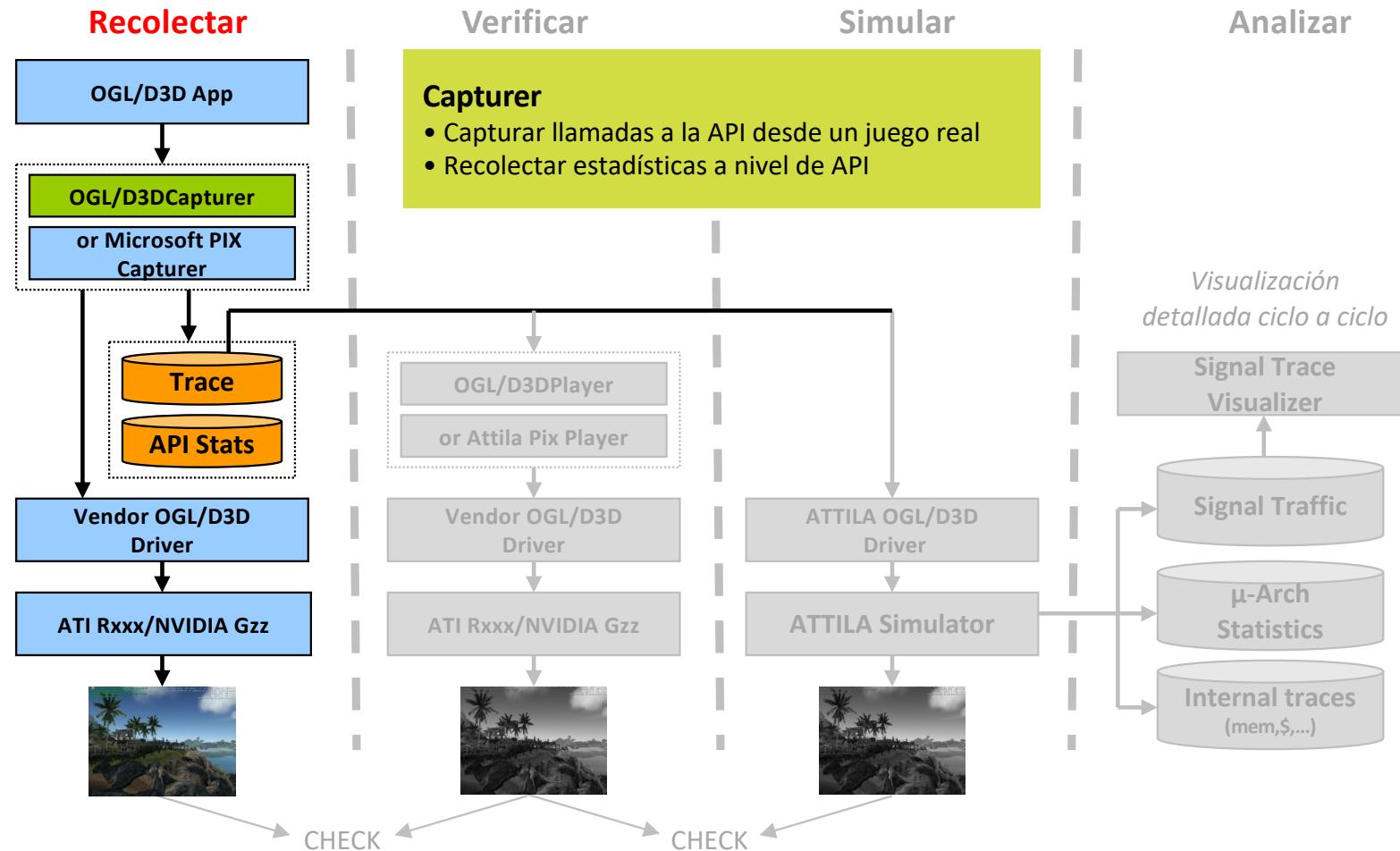
Attila: ¿Qué queríamos hacer?



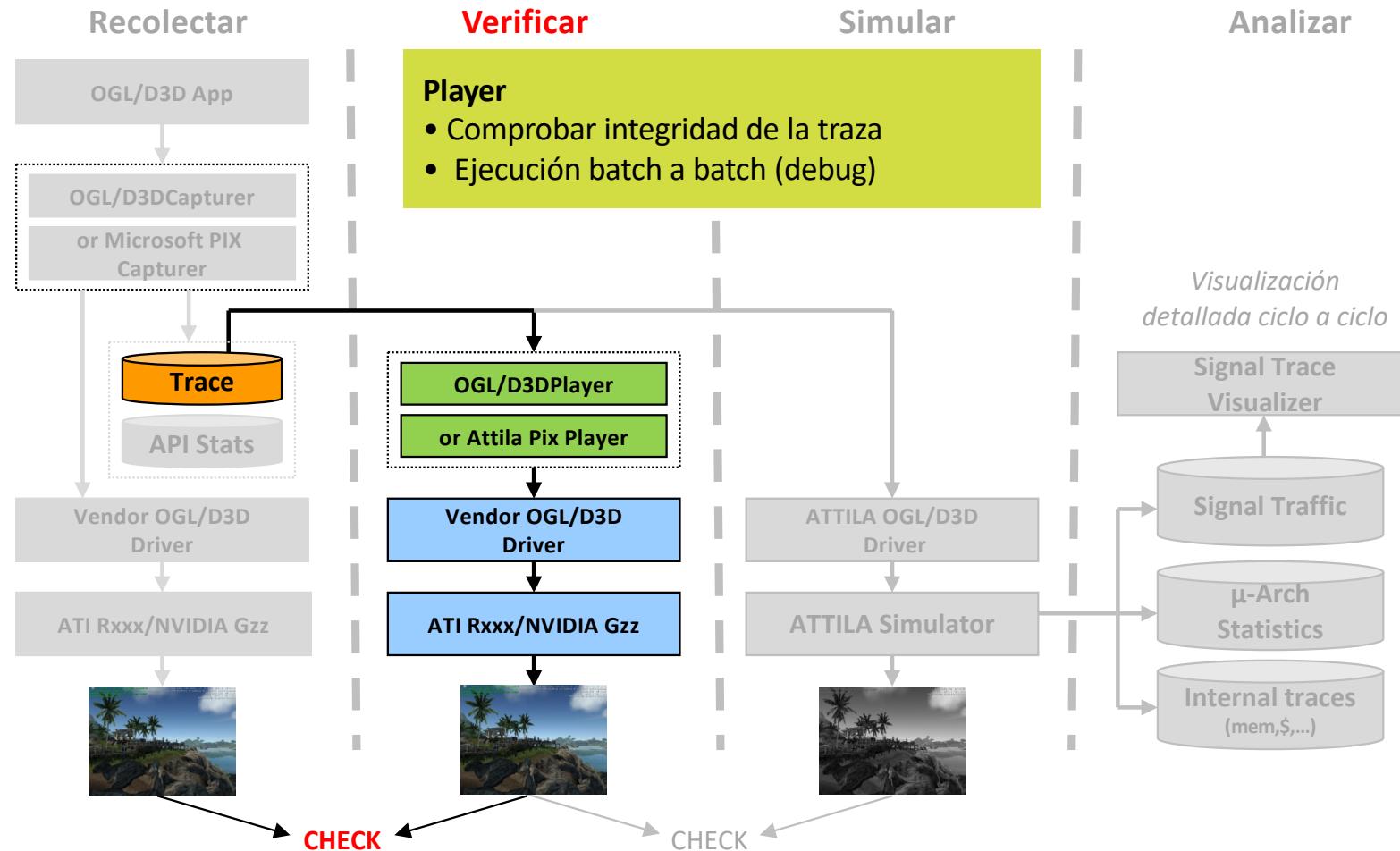
Attila: Entorno de trabajo



Attila: Entorno de trabajo

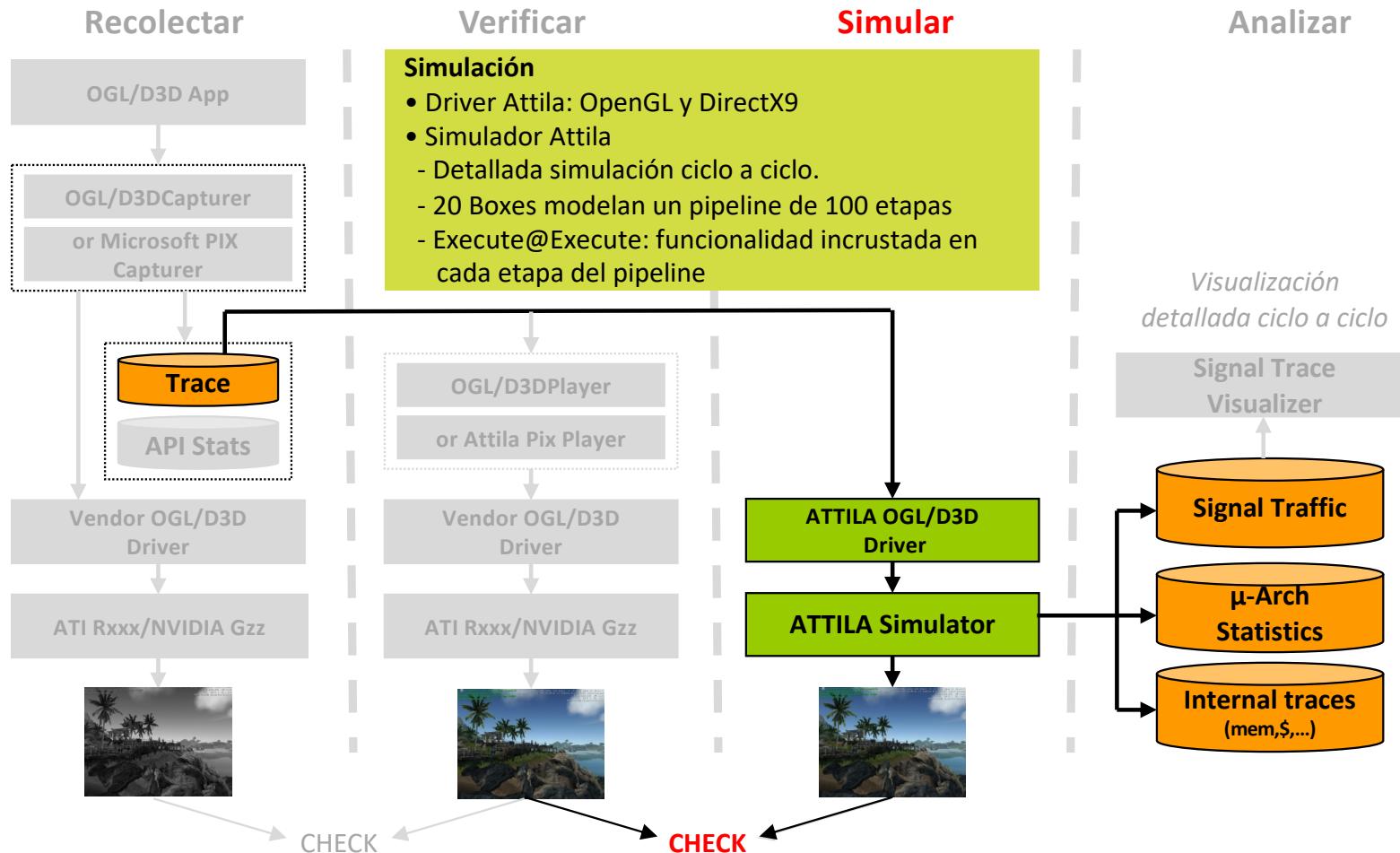


Attila: Entorno de trabajo

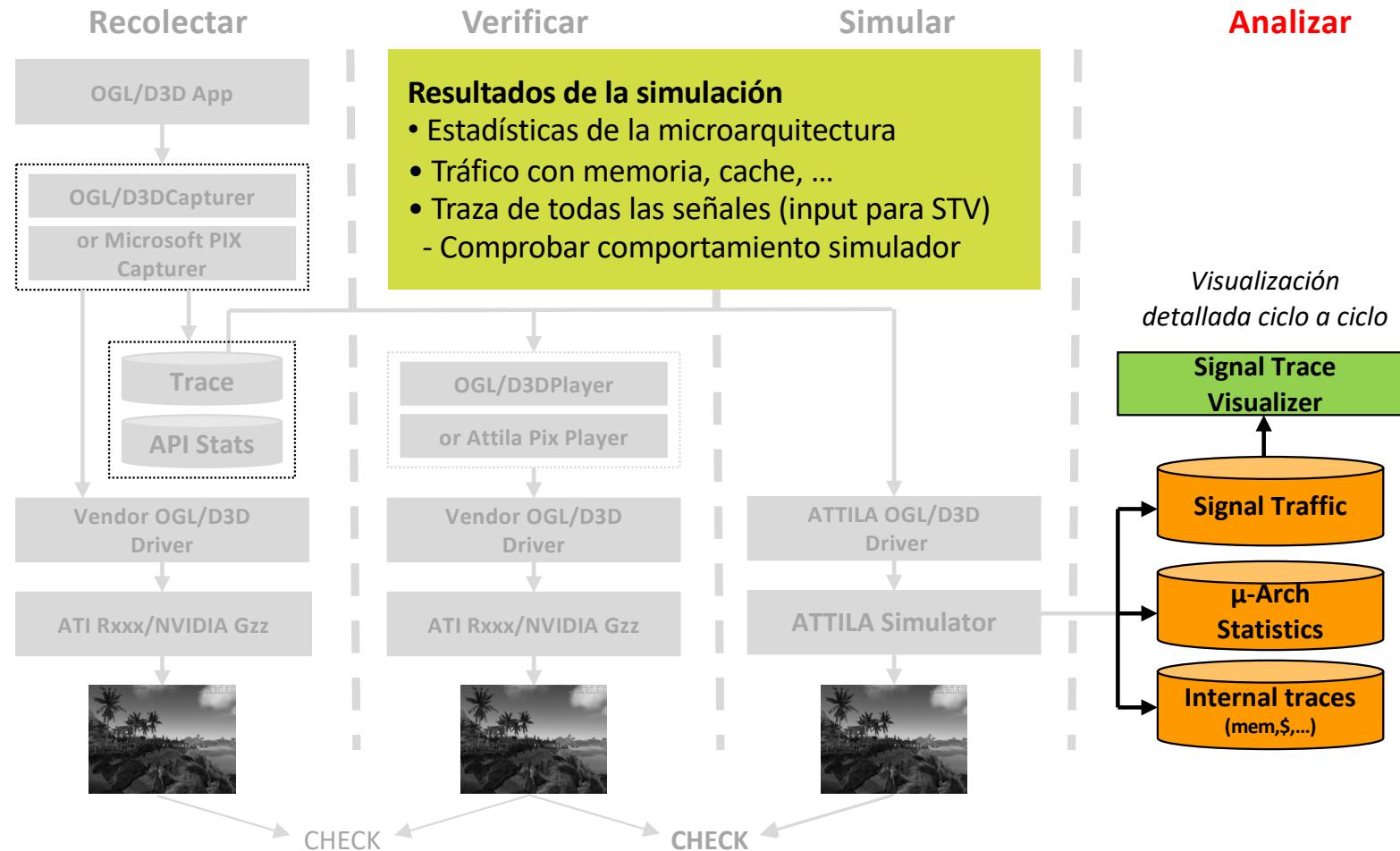


¿Cómo trabaja una GPU?

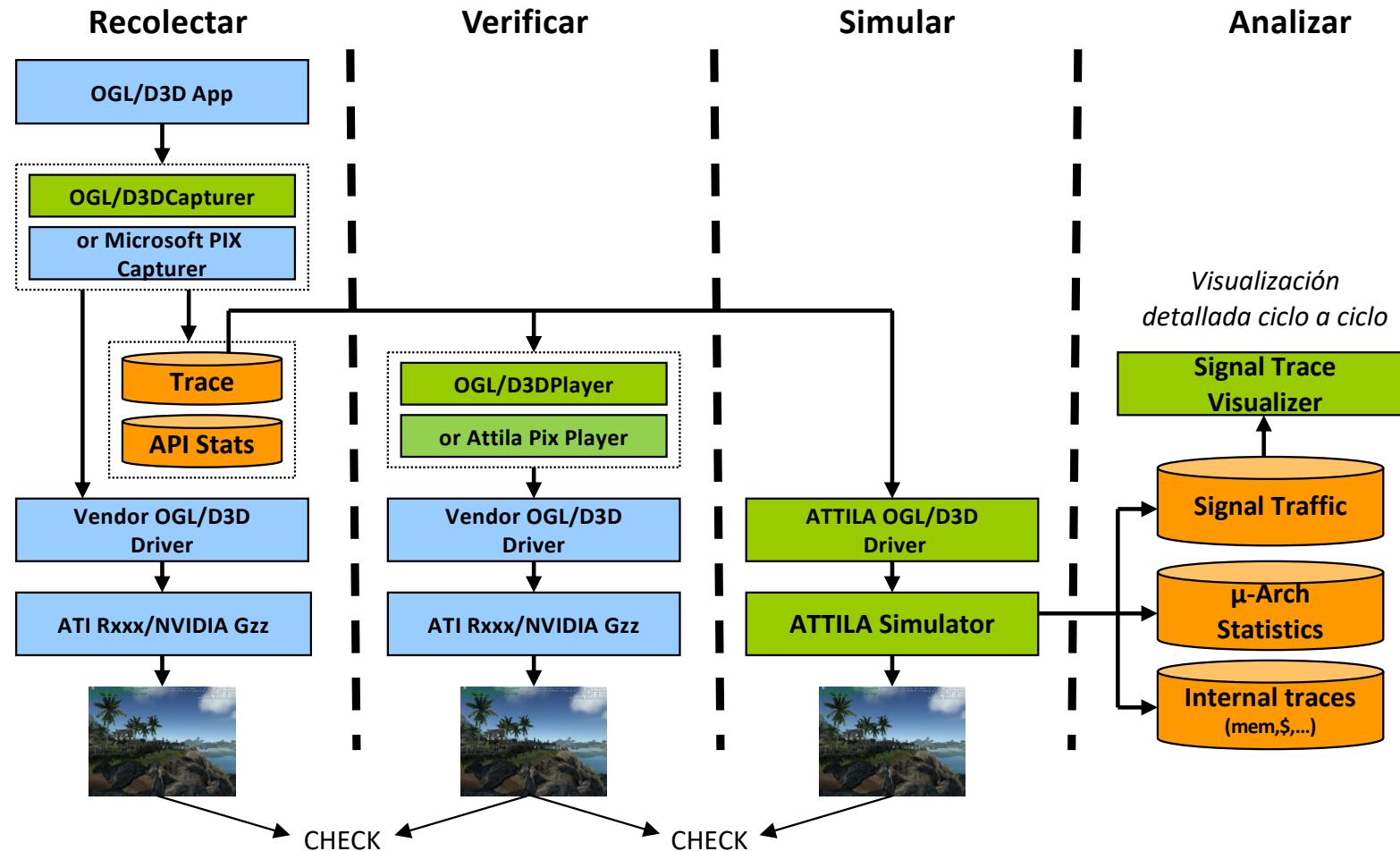
Attila: Entorno de trabajo



Attila: Entorno de trabajo

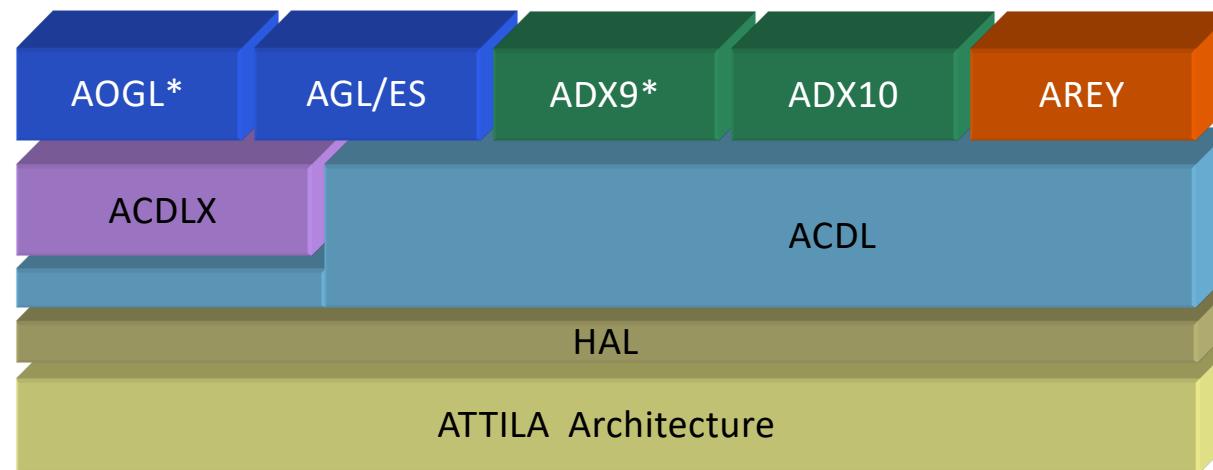


Attila: Entorno de trabajo



Attila: Arquitectura del Driver

- **ACDL:** Parte común del driver. Gestiona zonas de memoria, texturas, buffers, etc.
- **ACDLX:** Función fija
- **AOGL:** API OpenGL
- **ADX9:** API DirectX9
- **HAL:** Hardware Abstract Level



Attila: Juegos Soportados



Doom 3



UT 2004



Quake 4



Riddick



Prey



Half Life 2



Crysis Warhead



Left 4 Dead



Need for Speed



Burnout paradise



Unreal
Tournament 3



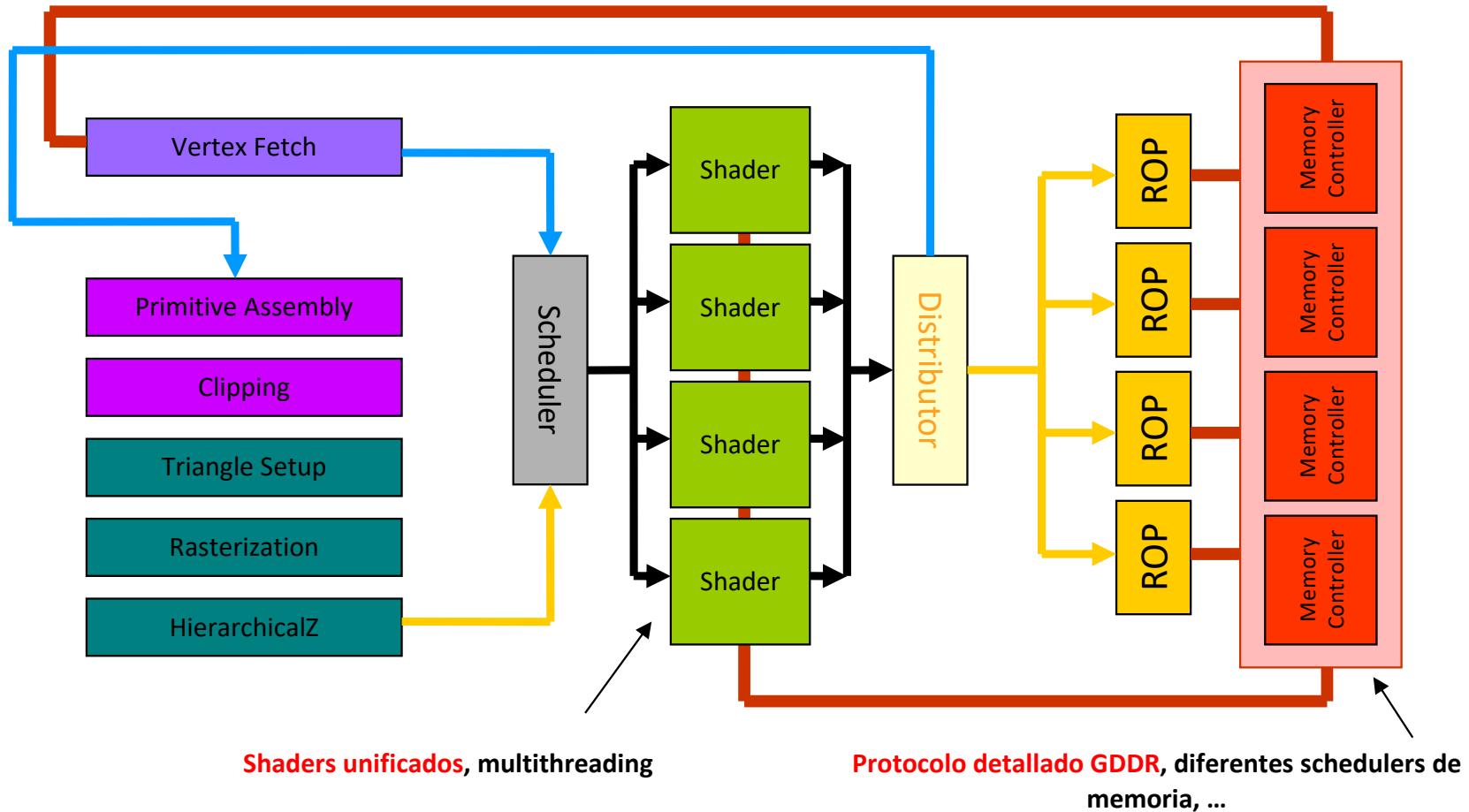
GRID



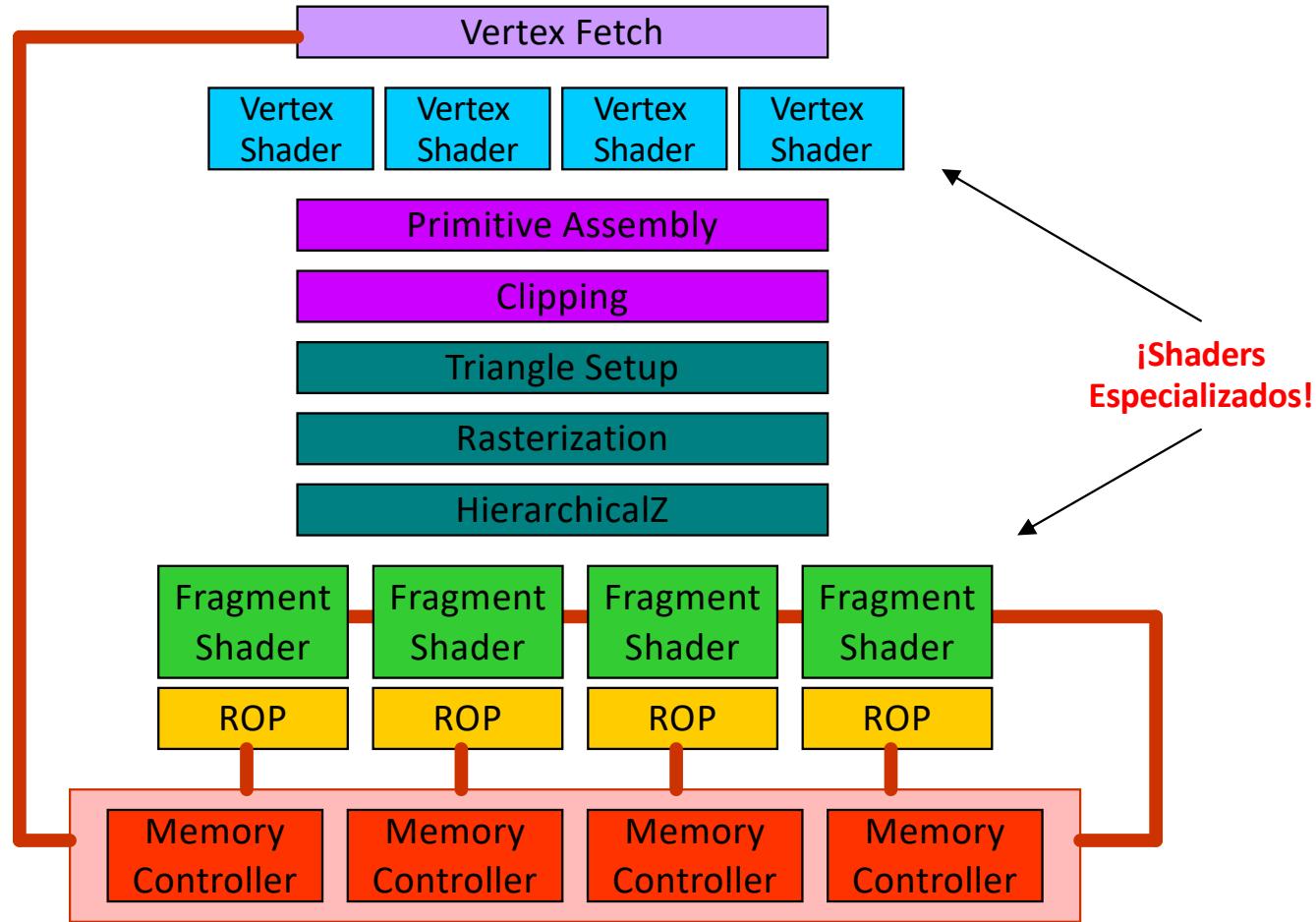
Call of Duty 4

...

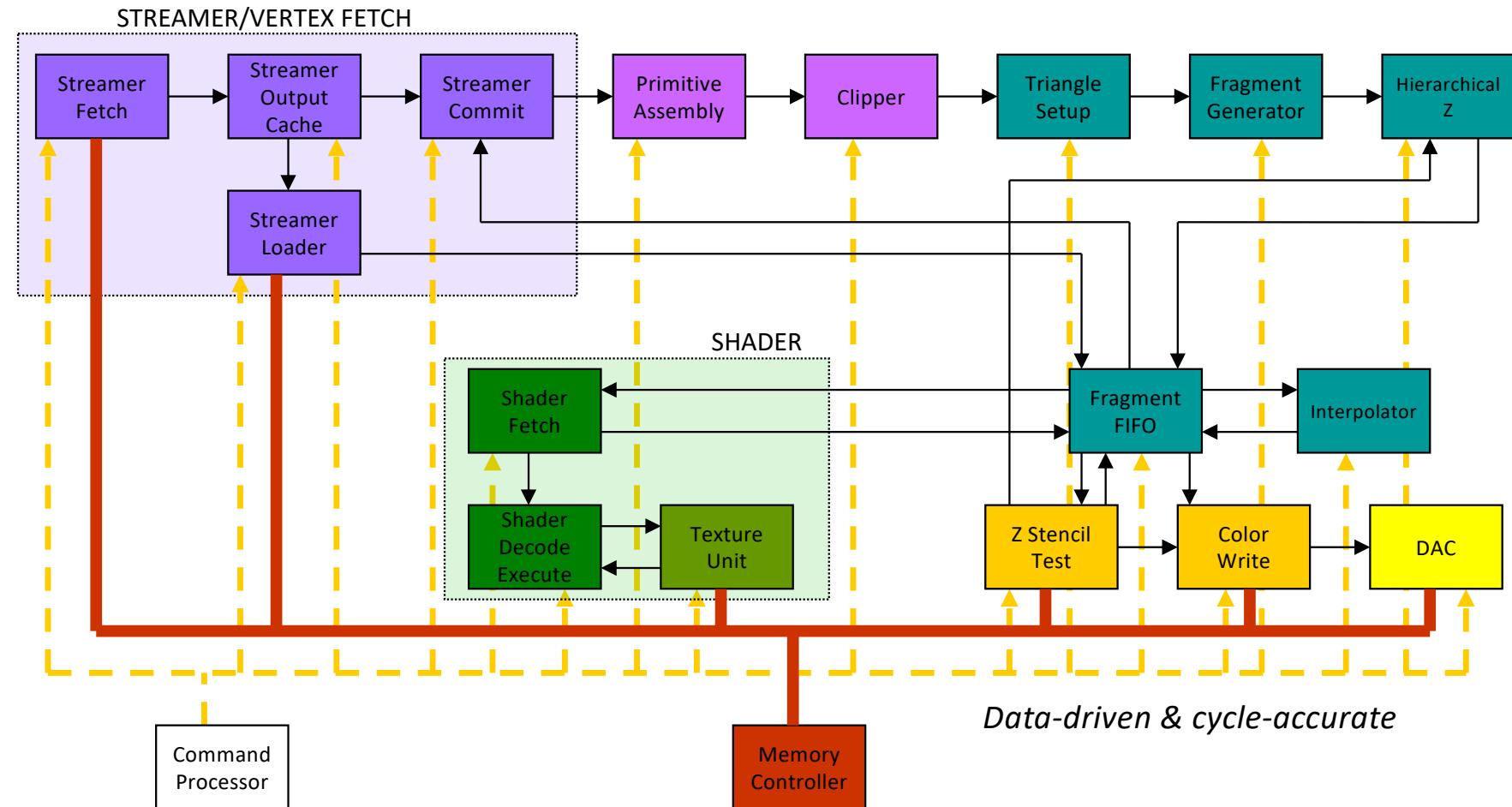
Attila: Arquitectura



GPU Clásica



Attila: Implementación Simulador



Attila: Algunos Números

- Velocidad simulación: 1 frame/hora con frames de 1280×1024

- Líneas de código

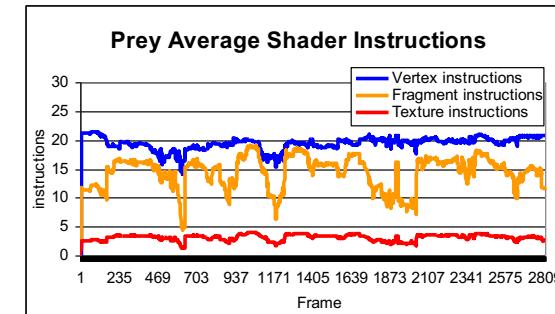
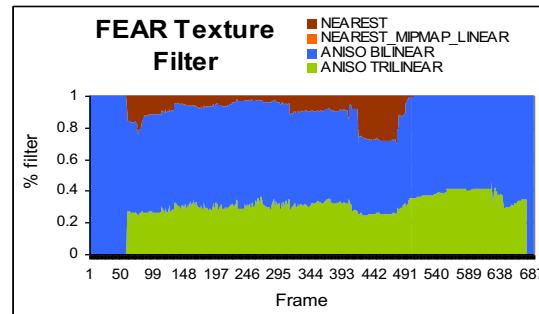
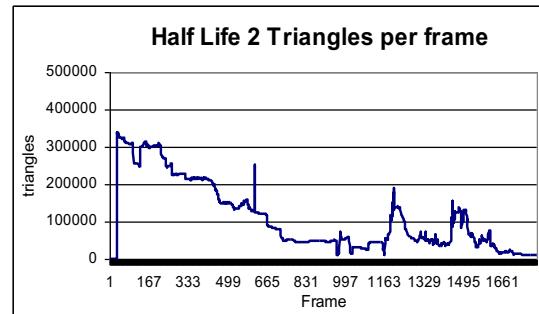
- Simulador: alrededor de 150.000 líneas
- Librería, driver y herramientas : 200.000 líneas (p.e. ACDL, 50.000 líneas)

- Parámetros del simulador

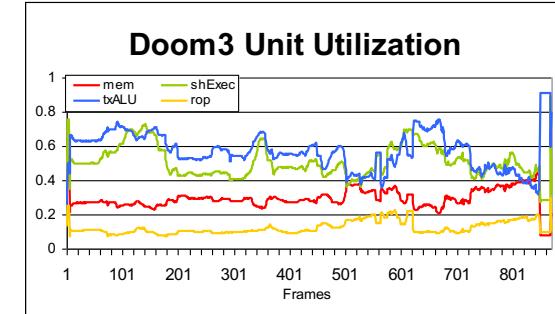
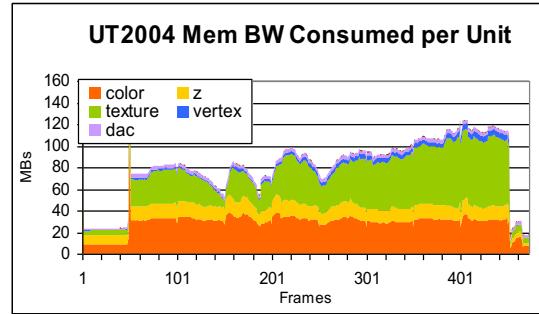
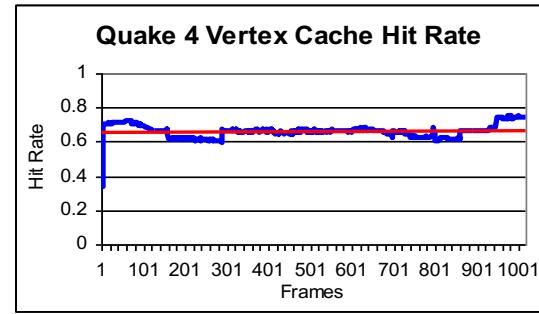
GPU Unit	#param	Ejemplos
COMMAND PROCESSOR	1	Batch pipelining
MEMORY CONTROLLER	42	Size, channels and banks (number and interleaving).
STREAMER	13	Fetched indices and attributes per cycle
PRIMITIVE ASSEMBLY	4	Assembled triangles per cycle
CLIPPER	5	Clipping latency
SETUP + RASTERIZER	43	MSAA samples/cycle, Enabled HZ
UNIFIED SHADER UNIT	39	Fetch Instrs/cycle, temp regs, scalar ALU
TEXTURE CACHE	19	Line size, ways, port width
ROP (Z + COLOR)	47	Compression, cache size.
DAC	9	Refresh rate
TOTAL	222	

Estadísticas: Alto Nivel

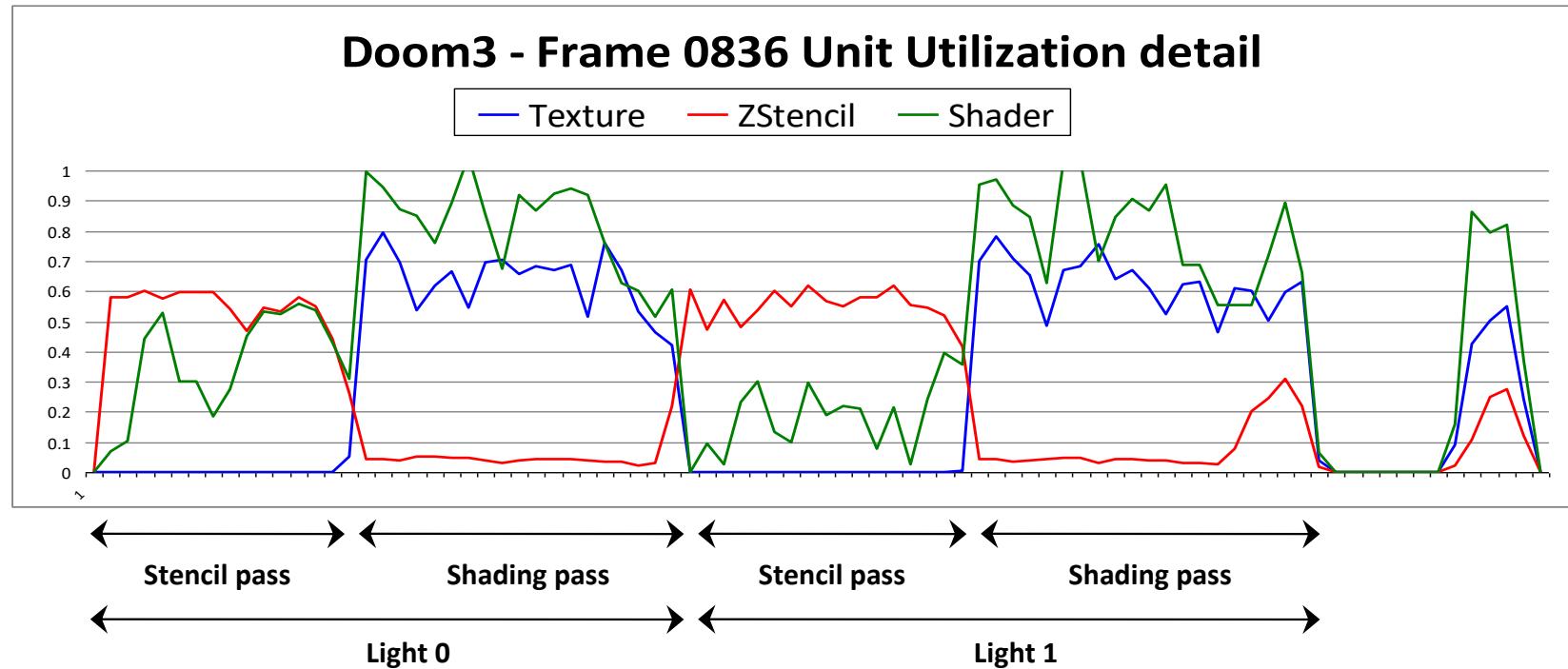
□ API level



□ μ-arch level

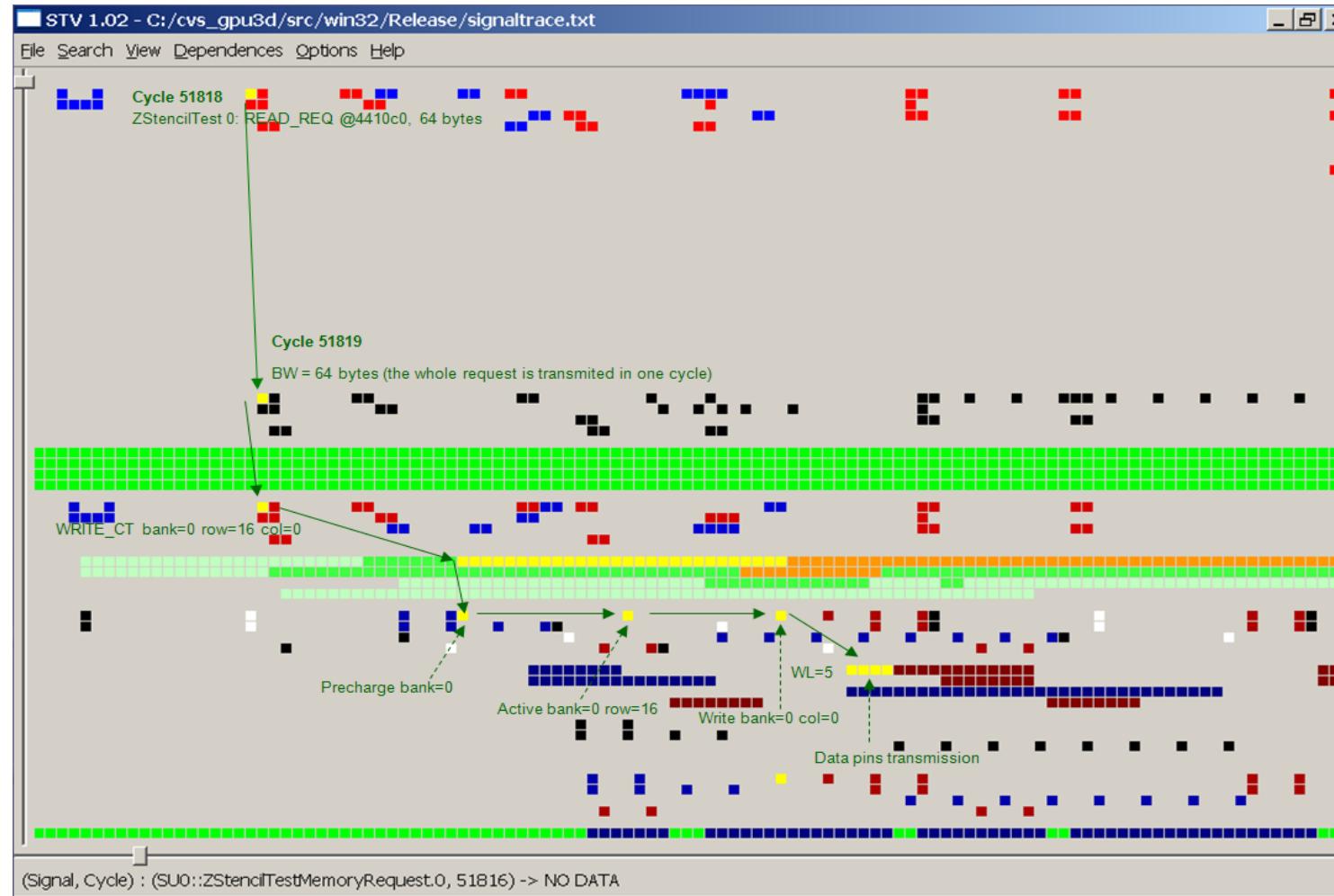


Estadísticas – Zooming In

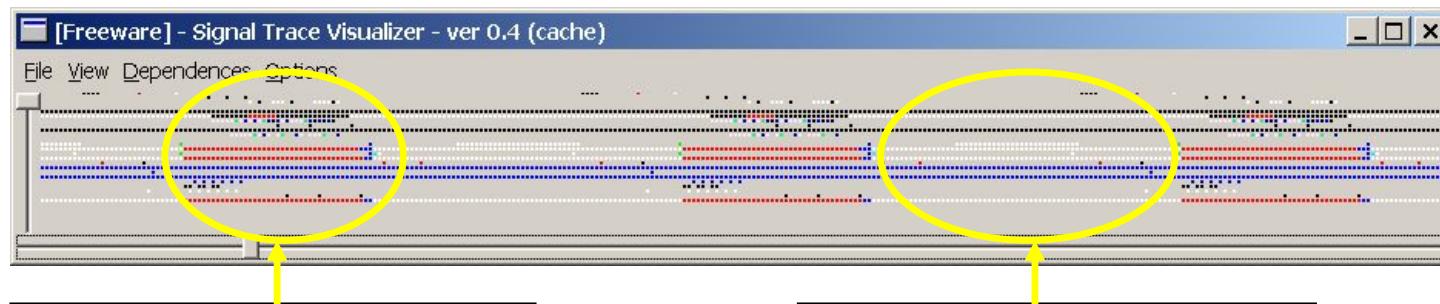
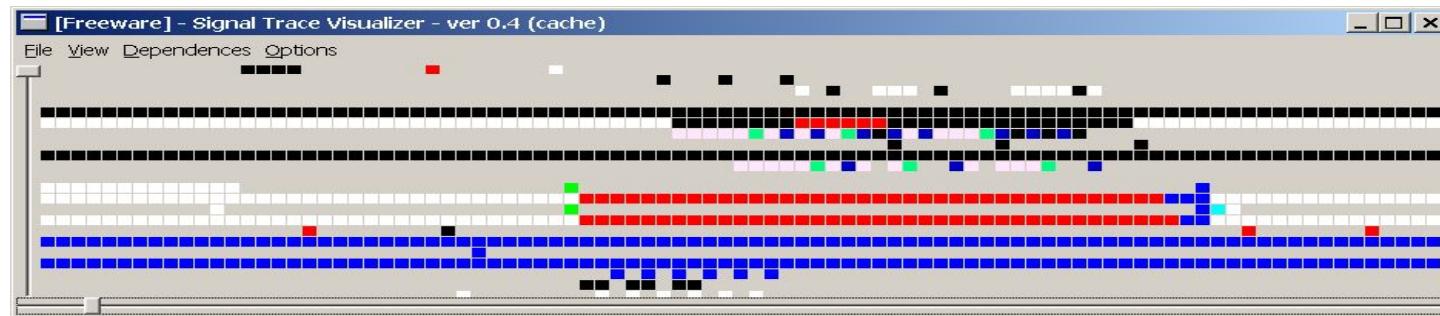


- Estadísticas de grano fino. Configurable por ejemplo a 100, 1.000 o 10.000 ciclos de ejecución.

Estadísticas a nivel de ciclo



Estadísticas a nivel de ciclo



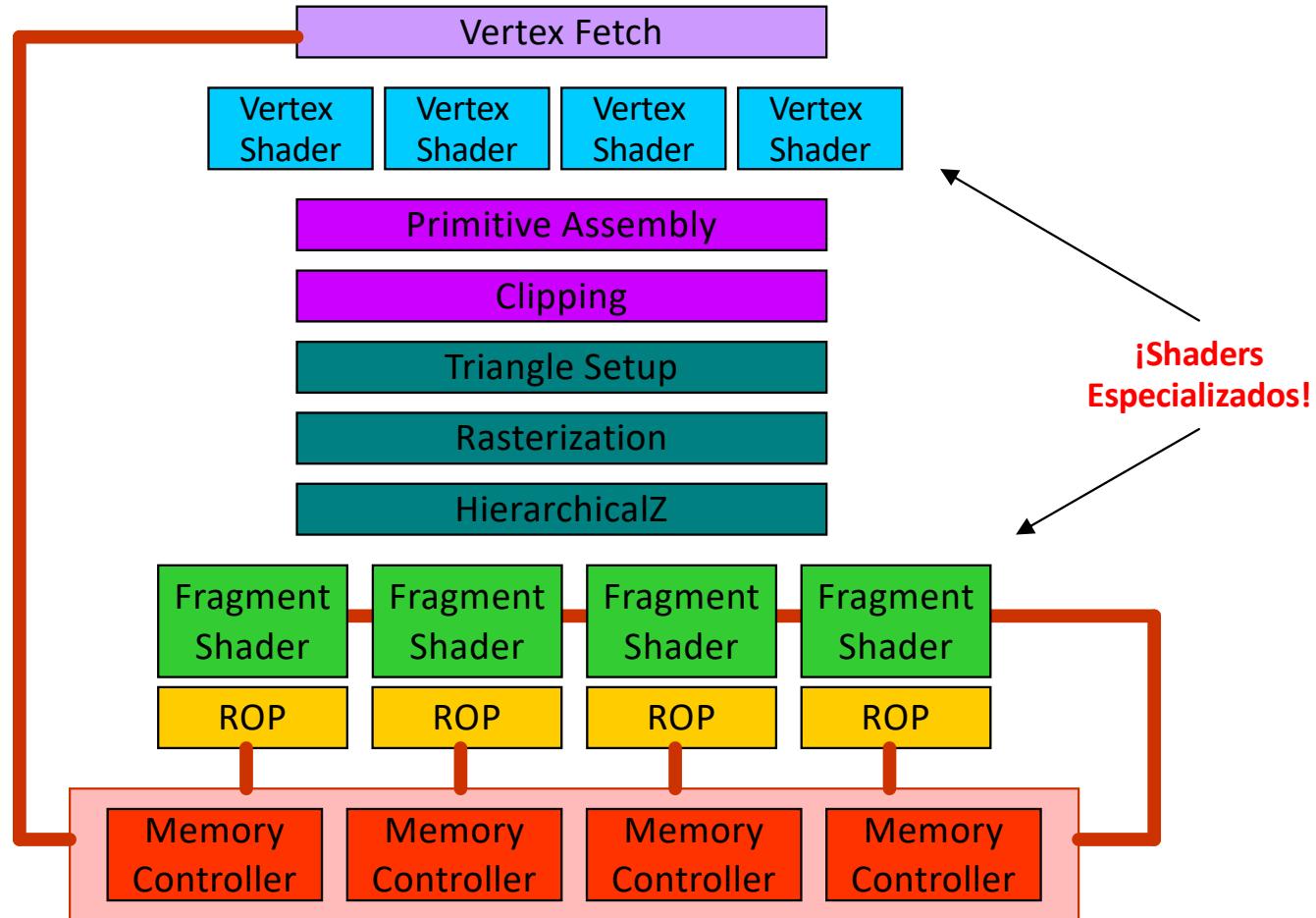
Cada uno de estos grupos indica el procesado de un vértice por el vertex program activo
Por tanto, en la imagen se está mostrando como se procesan 3 vértices diferentes

Los huecos entre el procesado de cada vértice muestran el tiempo de carga de los atributos de un vértice desde la memoria local del simulador a los registros INPUT del vertex shader

Attila

- Proyecto Iniciado en 2003
- Investigación en GPUs
 - Centrado en la microarquitectura
 - Uso de juegos reales como benchmarks
 - Análisis de la relación entre ancho de banda/latencia/paralelismo
- Mucho tiempo desarrollando herramientas
- 3 proyectos de tesis
 - Shaders Unificados
 - Controlador Memoria
 - MicroPolígonos

GPU Clásica



Shaders Especializados

□ Desbalanceo de carga

- Hasta el 30% de la potencia de cálculo de los fragment shaders no se utiliza.
- Hasta el 70% de la potencia de cálculo de los vertex shaders no se utiliza.
- Puntualmente el uso de los shaders llega al 100%.
- Experimentos realizados con 8 vertex + 4 fragment shaders (quads).

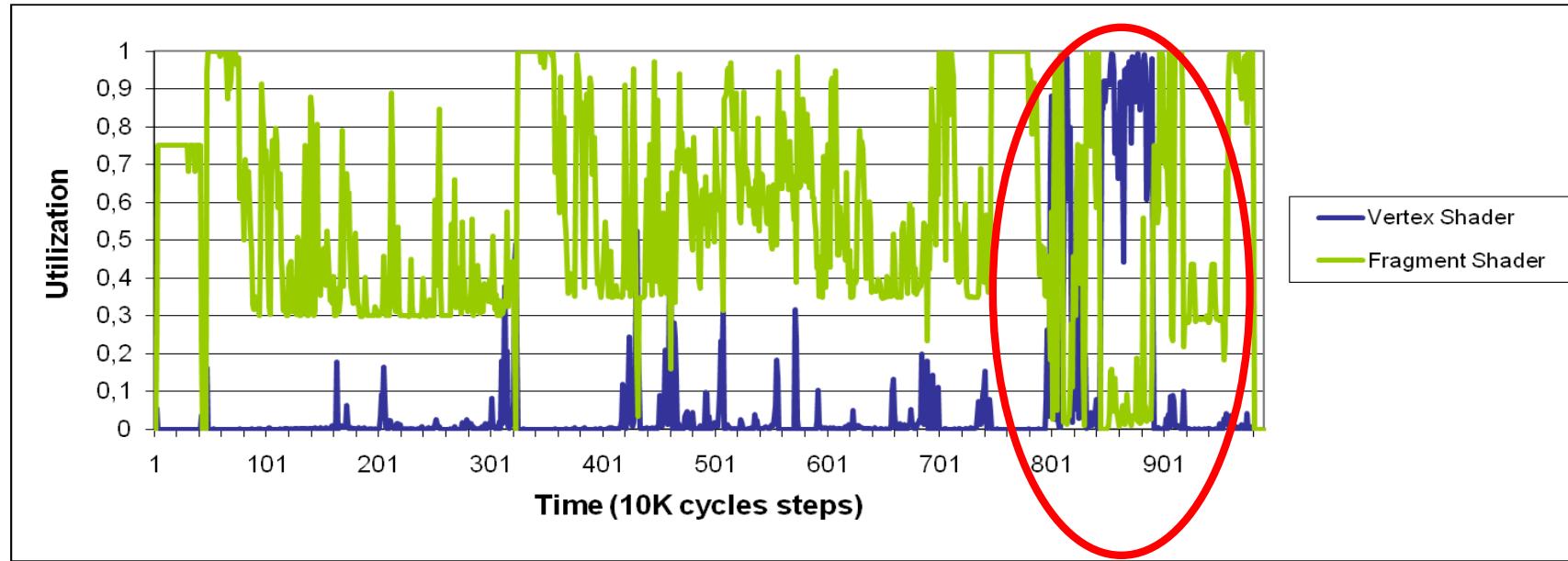
□ Área dedicada

- 4 vertex shaders tienen la misma potencia de cálculo que 1 fragment shader
- 4 vertex shader requieren el 66% del área de un fragment shader.

□ Diseños diferentes

- Aumenta la complejidad de la microarquitectura.
- Incrementa el tiempo de desarrollo y verificación.

Utilización de los shaders



- El uso de los shaders no es uniforme
- Si los shaders estuvieran unificados se podría balancear la carga

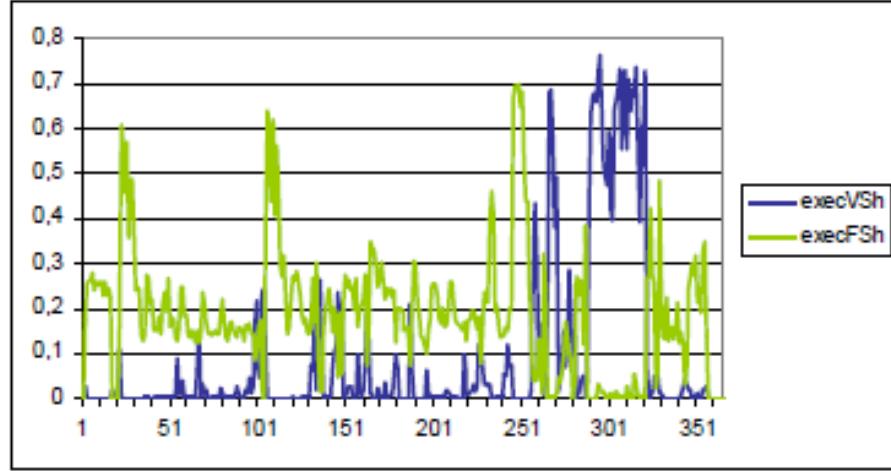
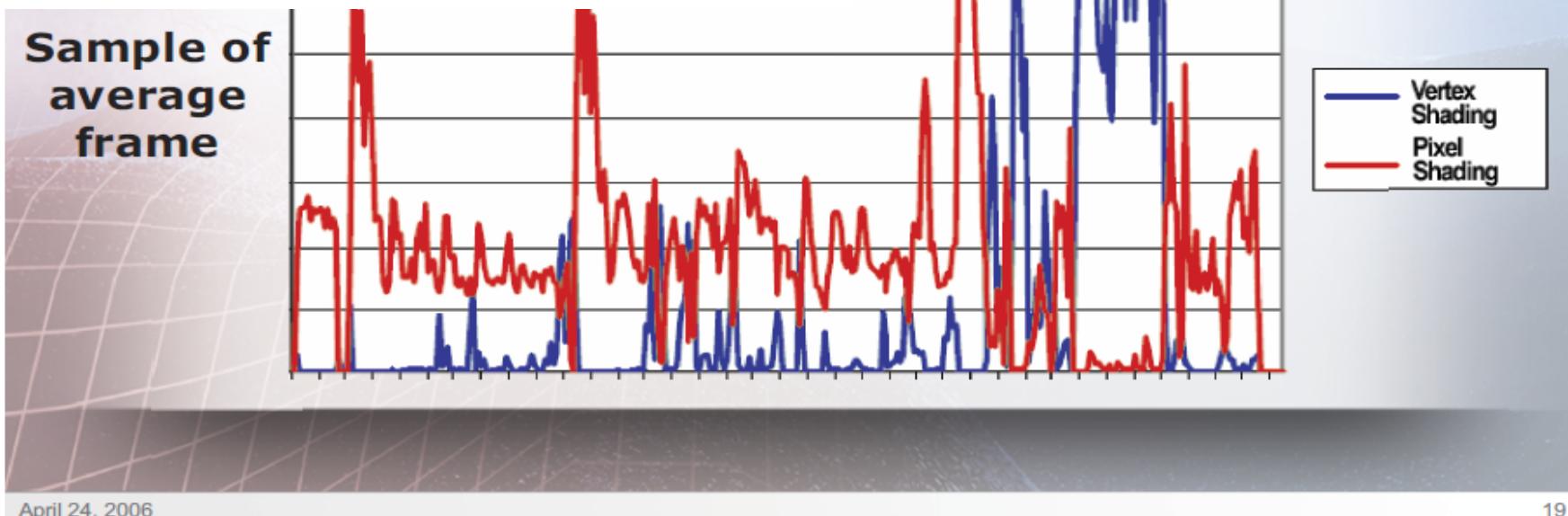


Figure 7. Workload balance in frame 330.



Victor Moya, Carlos González, Jordi Roca, Agustín Fernández and Roger Espasa. *Shader Performance Analysis on a Modern GPU Architecture*. In The 38th Annual IEEE/ACM International Symposium on Microarchitecture, 2005 (MICRO-38), pp. 355-364, Barcelona (Spain), Nov 2005.

not at the same time!

Figure 7. Workload balance in three tasks.

6.3. Detailed performance analysis

In this section we will analyse in detail what happened during the experiments. We will first analyse the results obtained by the baseline system and then we will compare them with the results obtained by our proposed system. Finally, we will analyse the results obtained by the other applied systems and compare them with our results.

Figure 8 shows the names of users and figures 9–12 show the corresponding performance measures. In figure 8, the names of the users whose performance measures are evaluated in table 2 are listed. Figure 9 shows the mean performance measure of each user. Figure 10 shows the mean performance measure of each user for each task. Figure 11 shows the mean performance measure of each user for each task in the case of the randomised schedule. Figure 12 shows the mean performance measure of each user for each task in the case of the non-randomised schedule.

Figure 8 shows the names of users and figures 9–12 show the corresponding performance measures. In figure 8, the names of the users whose performance measures are evaluated in table 2 are listed. Figure 9 shows the mean performance measure of each user. Figure 10 shows the mean performance measure of each user for each task. Figure 11 shows the mean performance measure of each user for each task in the case of the randomised schedule. Figure 12 shows the mean performance measure of each user for each task in the case of the non-randomised schedule.

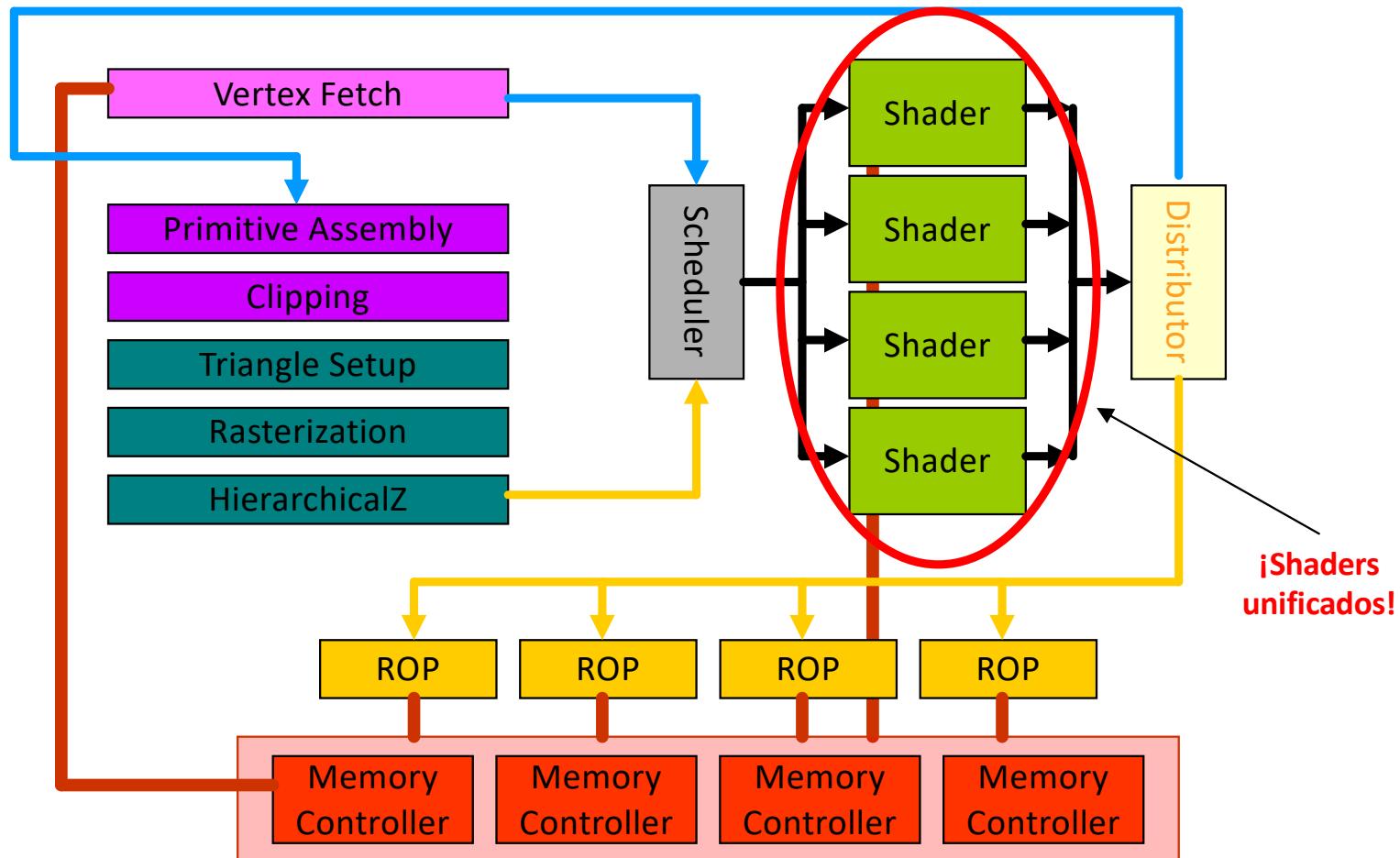
Figure 9 shows the mean performance measure of each user. The x-axis represents the user's name and the y-axis represents the mean performance measure. The mean performance measure of each user is calculated as the sum of the performance measures of all the tasks divided by the number of tasks. The mean performance measure of each user is calculated as the sum of the performance measures of all the tasks divided by the number of tasks.

Figure 10 shows the mean performance measure of each user for each task. The x-axis represents the user's name and the y-axis represents the mean performance measure. The mean performance measure of each user for each task is calculated as the sum of the performance measures of all the tasks divided by the number of tasks.

Figure 11 shows the mean performance measure of each user for each task in the case of the randomised schedule. The x-axis represents the user's name and the y-axis represents the mean performance measure. The mean performance measure of each user for each task in the case of the randomised schedule is calculated as the sum of the performance measures of all the tasks divided by the number of tasks.

Figure 12 shows the mean performance measure of each user for each task in the case of the non-randomised schedule. The x-axis represents the user's name and the y-axis represents the mean performance measure. The mean performance measure of each user for each task in the case of the non-randomised schedule is calculated as the sum of the performance measures of all the tasks divided by the number of tasks.

Shaders Unificados



Shaders Unificados

□ Beneficios

- Modelo de programación unificado
- No sólo hay vertex y fragment programs
 - ✓ DX10 : geometry shader
 - ✓ DX11: hull shader, domain shader
 - ✓ GPGPU
- Balanceo de carga
 - ✓ Los shaders se utilizan según se necesiten en cualquier instante.

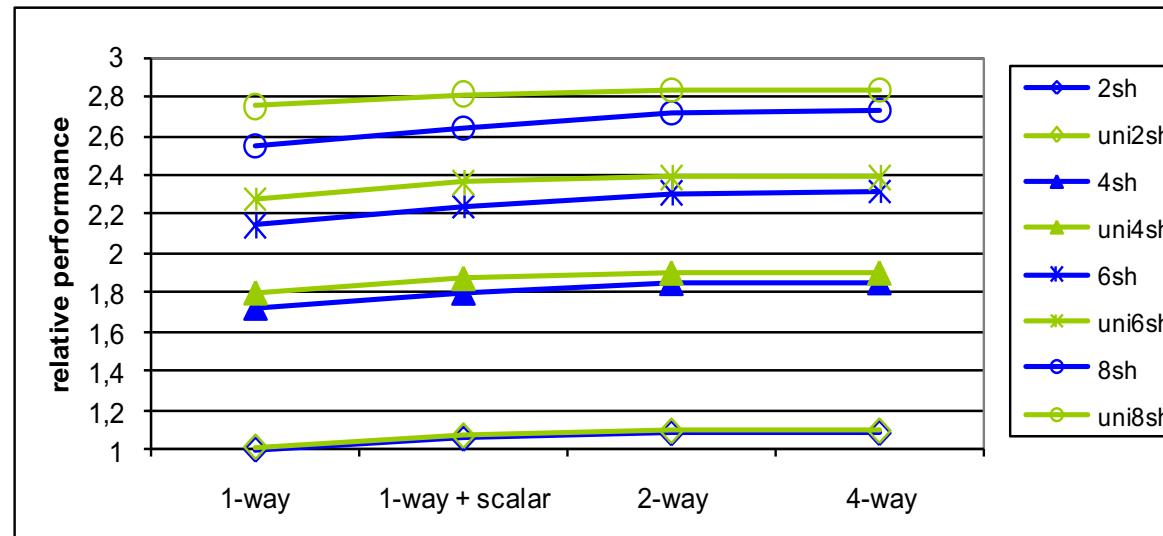
□ Costes

- Scheduler
 - ✓ Selección de cual es el tipo de carga que se procesa a continuación
 - ✓ Parcialmente implementado con el multithreading en el fragment shader para ocultar la latencia en el acceso a texturas.
- Memoria de instrucciones y banco de registros más grande
- Nuevo routing en la GPU

Experimentos

- Trazas UT2004, OpenGL, resolución 1024×768
- Simulación de 160 frames
- 4 Vertex Shaders: versión clásica
- Configuración shaders unificados:
 - 32 threads
 - ✓ 4 fragments/vertices por thread
 - ✓ 16 128-bit FP registros disponibles por thread
 - n SIMD ALUs
 - 1 scalar ALU (opcional)
 - 1 Unidad de Texturas por Shader
 - ✓ 16 KB cache
 - ✓ 1 ciclo para filtro bilinear y 2 para el trilinear
 - ✓ Filtro anisotrópico hasta 16x
- Geometría y rasterización limitadas a 1 vértice y 1 triángulo por ciclo
- 4 canales memoria DDR: 8B/ciclo por canal

Rendimiento Shaders Unificados



- La mejora de los shaders unificados llega hasta el 8%:
 - Limitado por el fetch de vértices, la rasterización y sobre todo por el ancho de banda con memoria.
- Los vertex shaders se pueden suprimir (ahorro área de chip) o se pueden utilizar para añadir nuevos shaders unificados (mejor rendimiento).

La Memoria en las GPUs Actuales

□ Tremendo Ancho de banda con memoria

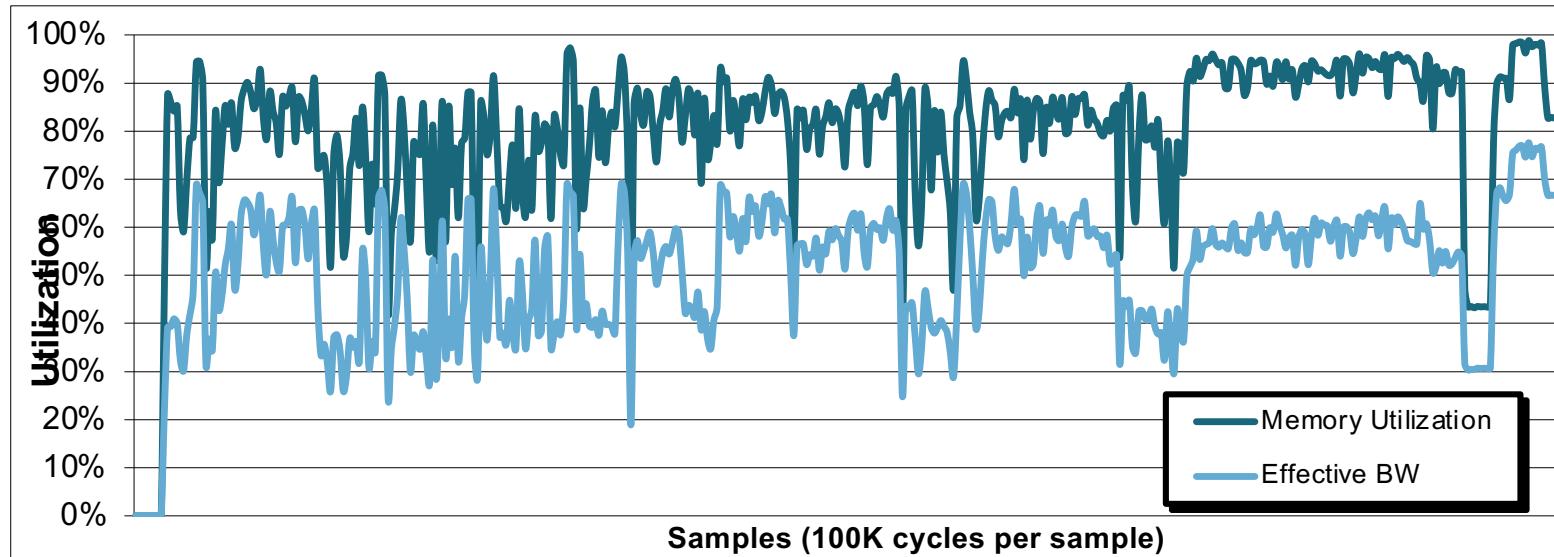
- Core 2: 12 GB/s vs NVIDIA G80 > 100 GB/s

□ Pero ...

- Docenas de clientes accediendo simultáneamente a memoria
- Diferentes patrones de acceso
- Un scheduling de las transacciones con memoria puede provocar una pérdida de rendimiento.
 - ✓ Workload No balanceado
 - El Ancho de Banda disminuye
 - ✓ Scheduling ineficiente
 - La latencia con memoria aumenta (overhead protocolo DDR)

□ Degradación del rendimiento general ☹

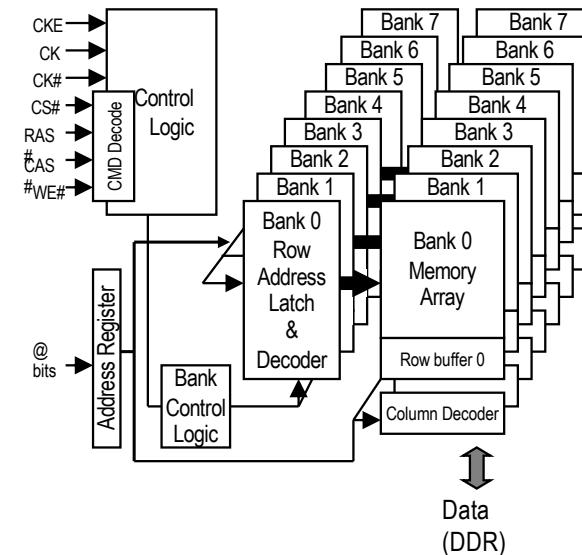
Utilización de la Memoria



- GPU configurada como una ATI RV740 a 600 MHz
- Memoria configurada como 8 GDDR3 a 900 MHz
- La Memoria está ocupada un **80%** (en media), mientras que el ancho de banda efectivo sólo llega al **50%**.

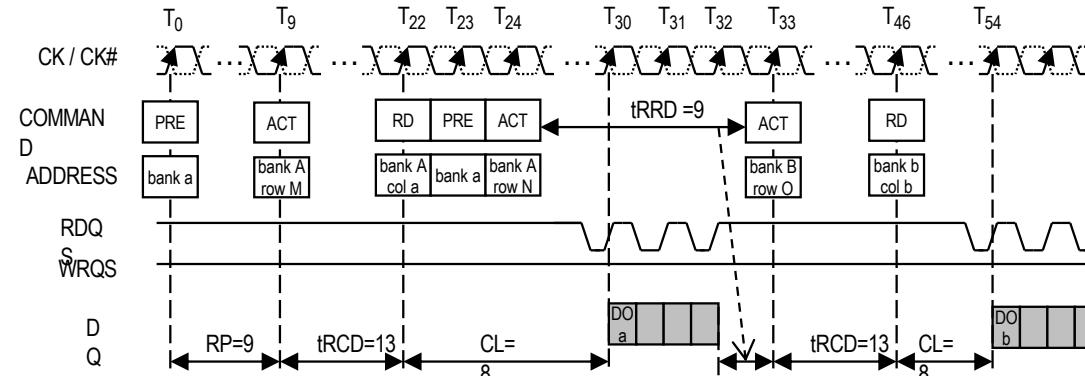
Arquitectura de una GDDR3/4

- Especificaciones de la Hynix HY5RS123235BFP (GDDR3 a 900 MHz)
- Comandos soportados: ACTIVE, PRECHARGE, READ and WRITE.
- Ráfagas, en secuencia, de 4 y 8 (\times 32bits)
- Restricciones de tiempo definidas en el protocolo GDDR3:
 - $tRDD$: Retardo entre ACTIVE(b1) y ACTIVE(b2): 9 ciclos
 - $tRCD$: Retardo entre ACTIVE y READ: 13 ciclos
 - $tWTR$: Retardo entre WRITE y READ: 8 ciclos
 - $tRTW$: Retardo entre READ y WRITE: 2 ciclos
 - tWR : Recuperación después de un WRITE que incluye PRECHARGE: 12 ciclos
 - tRP : PRECHARGE: 12 ciclos
 - CAS: Latencia CAS: 10 ciclos
- Nuevas versiones de GDDR son similares, sólo cambia el timing.

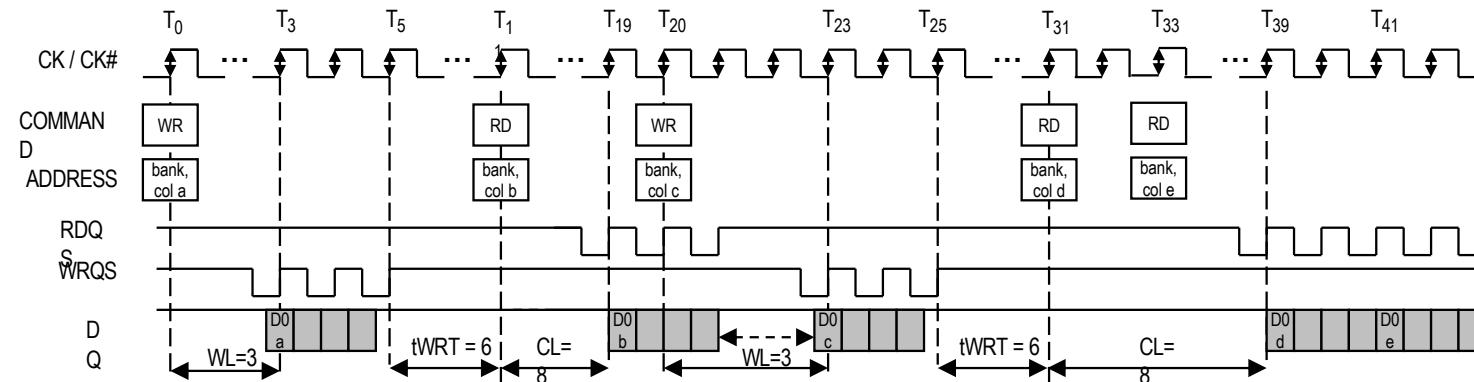


Conflictos en el protocolo GDDR

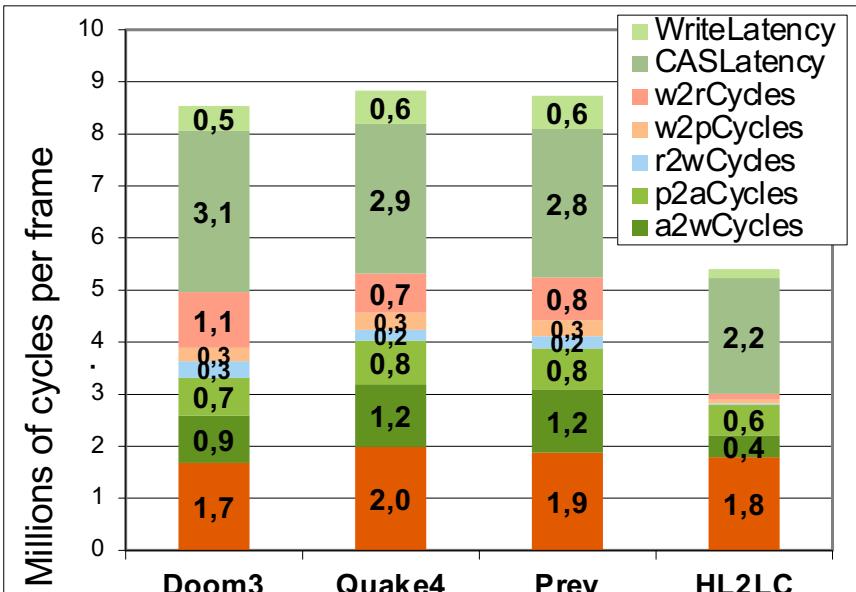
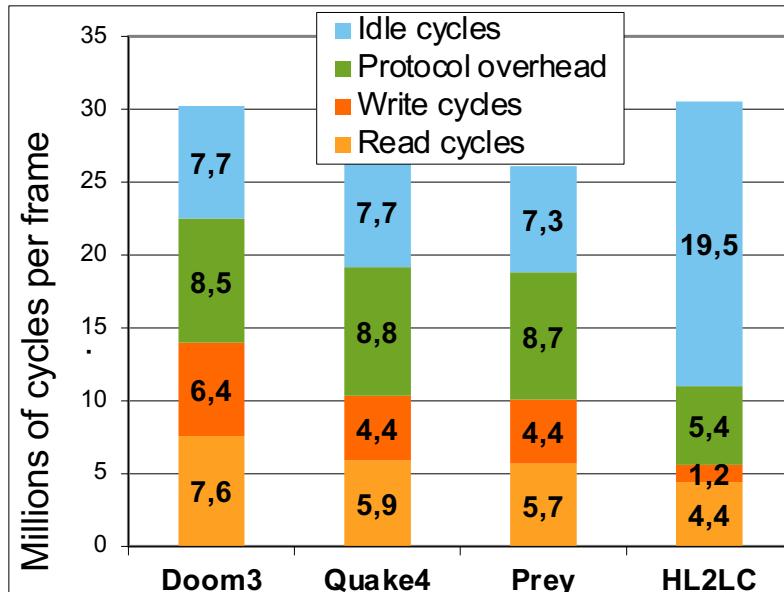
- Overhead provocado por los conflictos al abrir y cerrar páginas.



- Overhead provocado por las transiciones entre READ/WRITE y viceversa

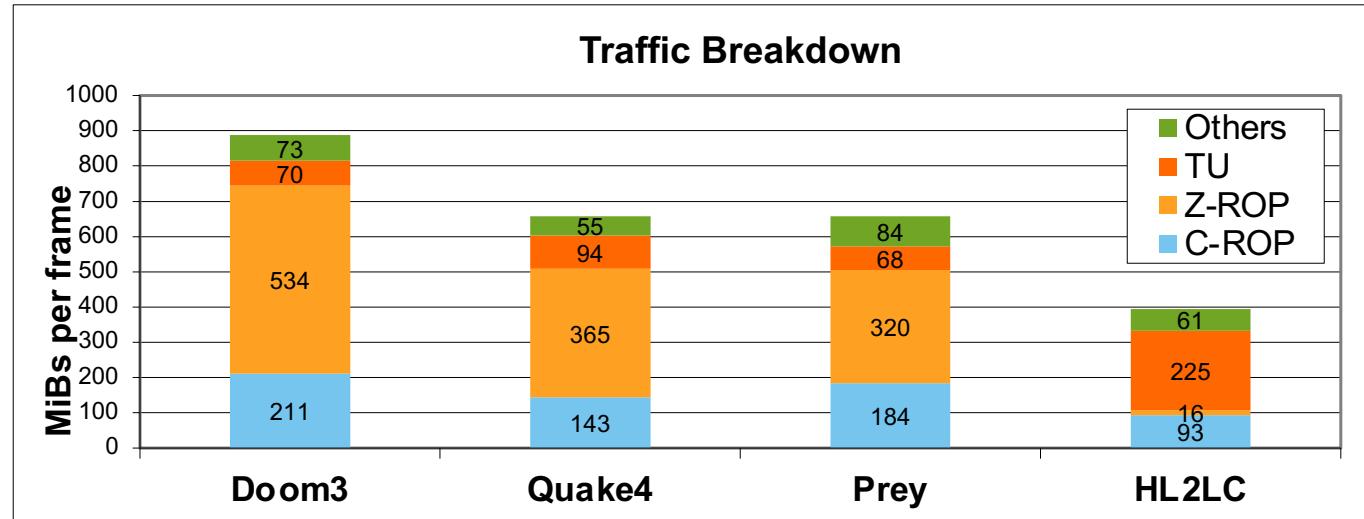


Análisis del Overhead del protocolo



- Análisis realizado con un controlador de memoria tipo FIFO
- Overhead supone entre un 18 y 33% del total.
- Los ciclos inactivos son más altos de lo esperado.
- Overhead debido a los fallos de página.
- Overhead debido a las transiciones entre lecturas y escrituras.

Tráfico con Memoria



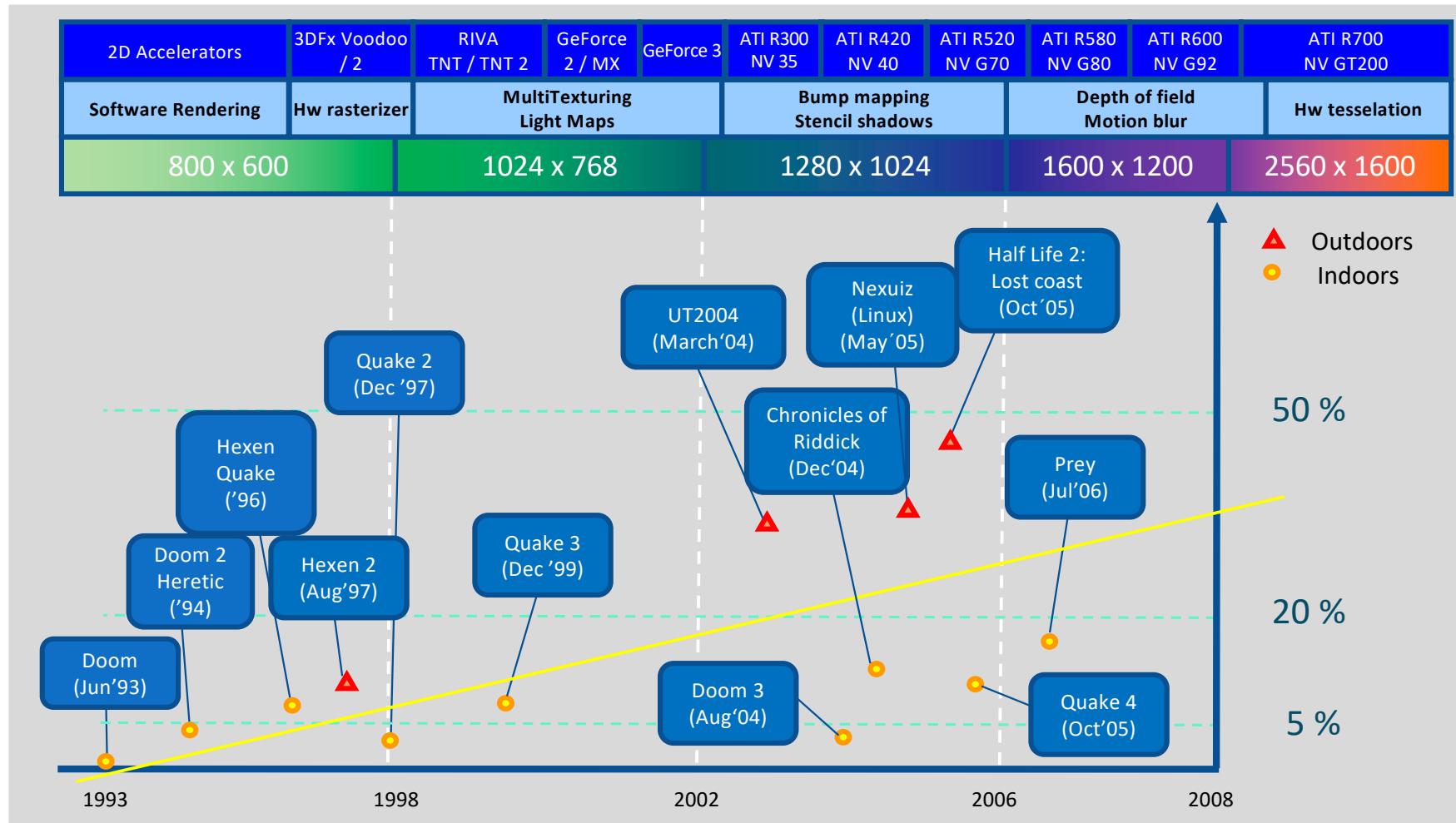
- Los elementos que más tráfico generan son TU, C-ROP y Z-ROP
- La influencia de las ROP es debida al MSAA
- HL2LC tiene el MSAA desactivado
- La distribución no es uniforme
- La cantidad de información que se lee/escribe en memoria es muy elevada.

Comentarios Finales

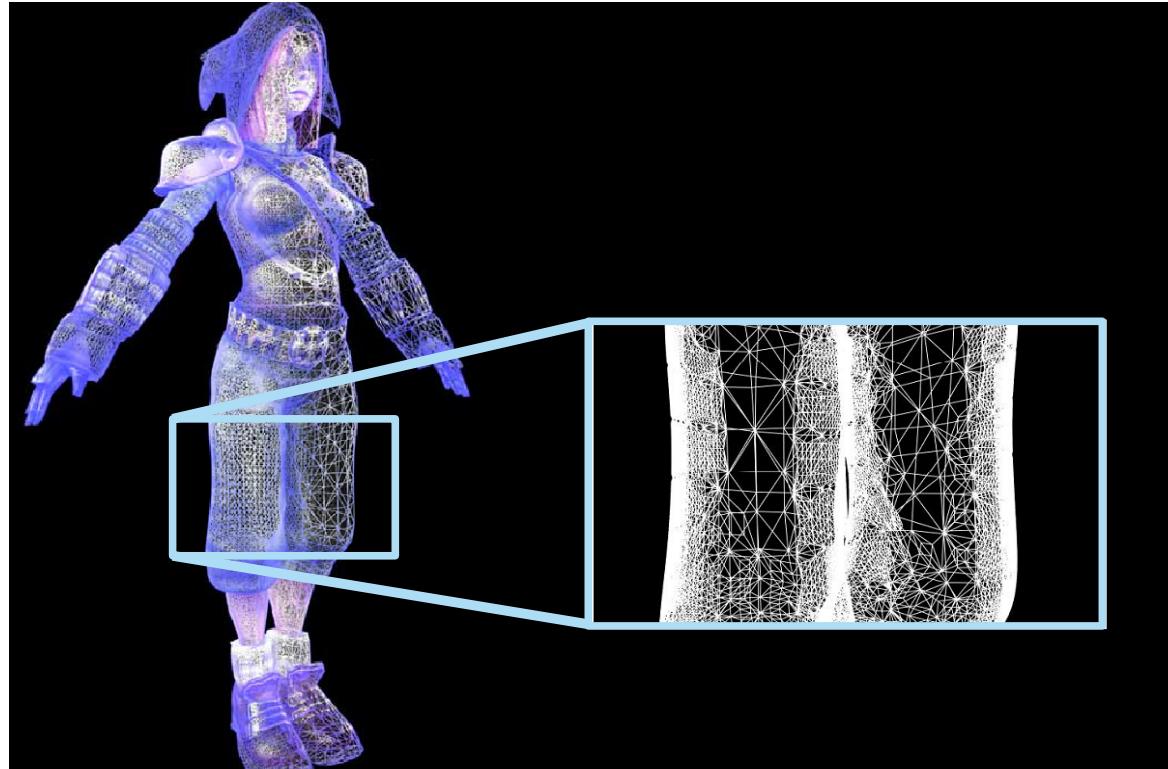
- Hemos realizado un estudio exhaustivo de cómo utilizan la memoria las aplicaciones de gráficos 3D.
- Hemos diseñado y evaluado un conjunto de schedulers de memoria y comprobado que es un tema a tener en cuenta al diseñar una GPU.
- Este tipo de estudio ya ha sido realizado previamente para una CPU convencional, pero no para una GPU.

- Hemos realizado estudios adicionales respecto a cómo distribuir la información entre los diferentes chips de memoria. Básicamente, dada una dirección de la memoria de la GPU, decidir que bits utilizamos para seleccionar el canal, el chip, el banco, ...

%primitivas ≤ “1 pixel”



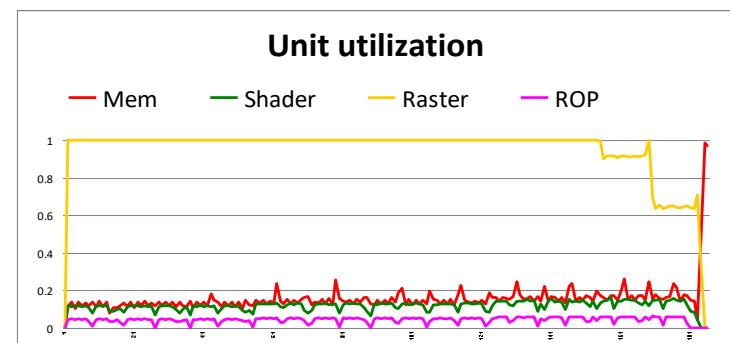
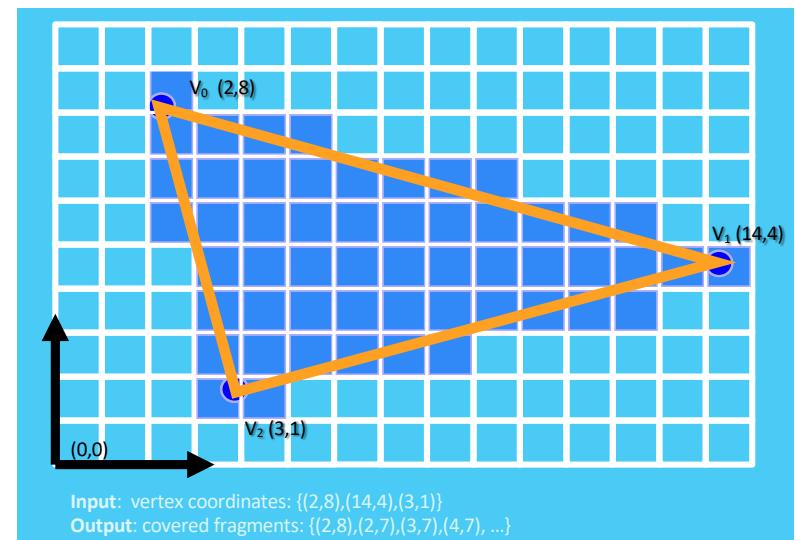
Teselación hardware



- Andrei Tatarinov, *Instanced Tessellation in DirectX10* Game Developers Conference, NVIDIA Developer Technology, February 2008

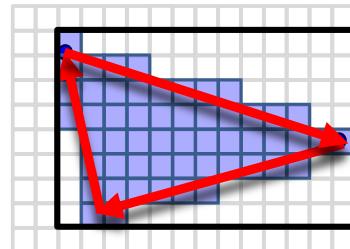
Rasterización: cuello de botella

- Muchas GPUs sólo pueden rasterizar 1 polígono por ciclo.
- Si los triángulos son grandes no es problema.
- Cuando los triángulos son muy pequeños (≈ 1 pixel). La rasterización se convierte en el cuello de botella del sistema.
- Ejemplo de utilización con un benchmark sintético.
- Dos métodos para rasterizar:
 - Cross product
 - Edge equations (usado en GPUs desde 1998)

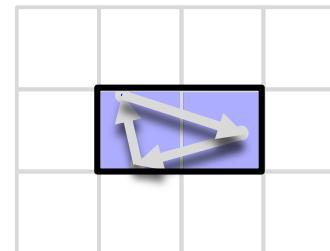


Rasterización: cuello de botella

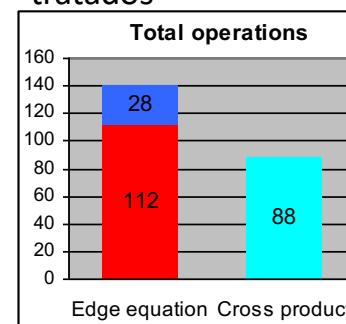
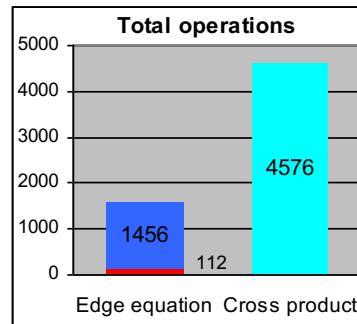
Técnica de rasterización		Operaciones (coste)			
		MULs(3)	ADDs(1)	RCPs(5)	
Método 1	Cross products (por pixel)	9	17	0	44 ops
Método 2	Edge equations (por pixel)	3	5	0	14 ops
	Setup of Edge equations (por triángulo)	30	17	1	112 ops



BB (13x8) = 104 pixels tratados

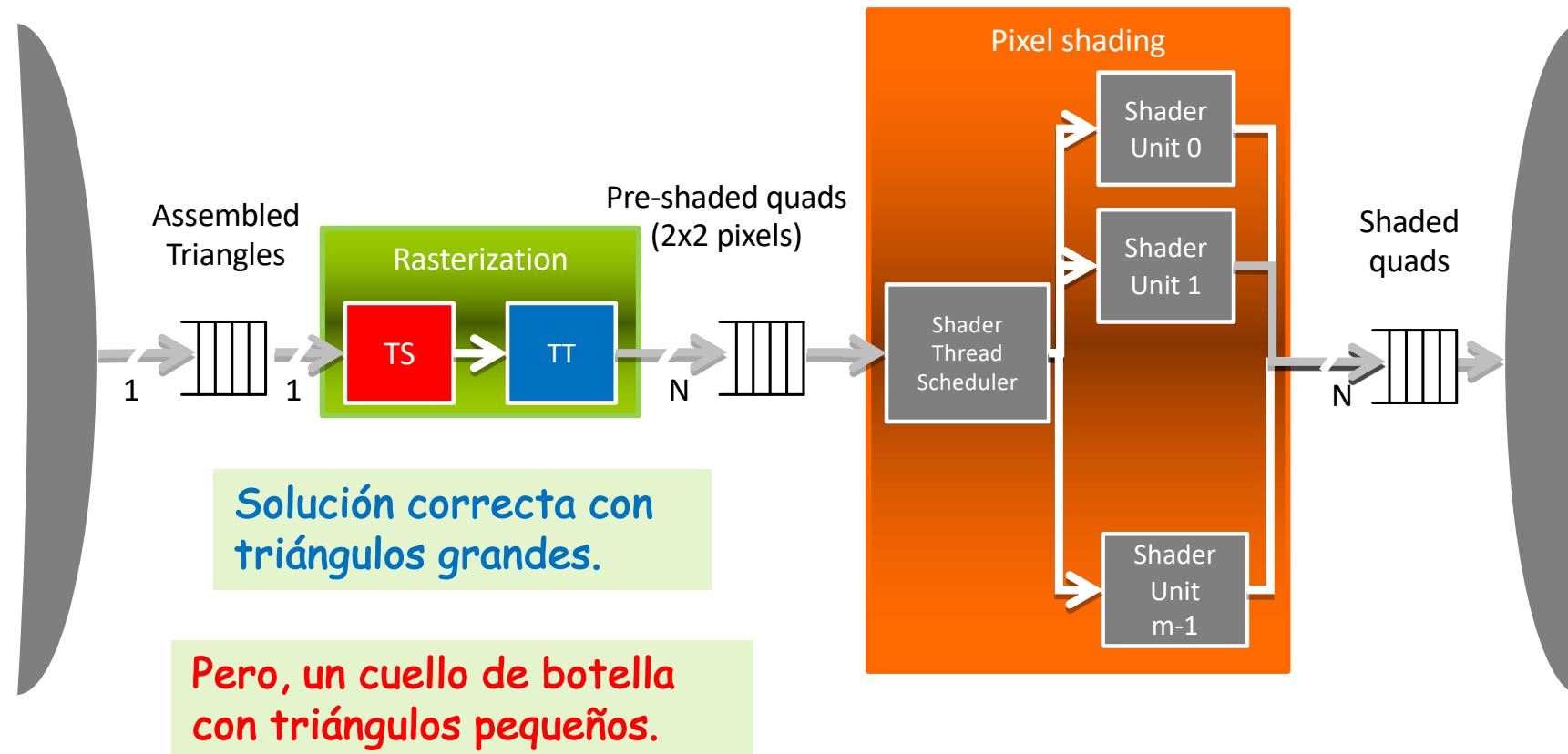


BB (2x1) = 2 pixels tratados

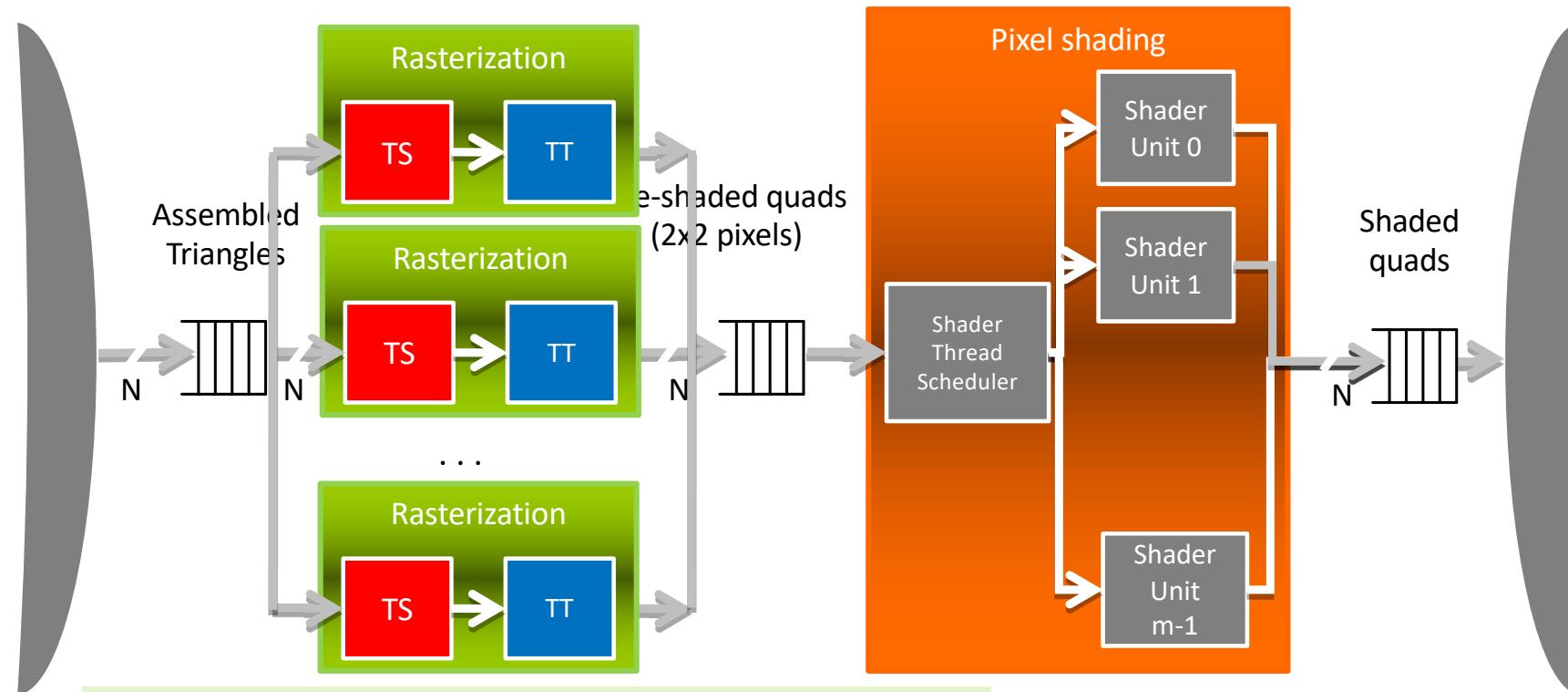


¡Para triángulos menores de 4 pixeles es mejor la alternativa del cross product!

Rasterización en las GPUs actuales



¿Replicar el rasterizador?



Con triángulos grandes sólo se utilizaría 1.

Con triángulos pequeños sería más costoso de lo necesario.

Posibles soluciones

- A) Definir un camino alternativo para los MicroTriángulos
- B) Cortocircuitar el rasterizador para los MicroTriángulos
 - El fragment shader deberá incluir un prólogo para rasterizar el Micro Triángulo
- C) Rasterizar todos los triángulos en el fragment shader
 - Eliminar el rasterizador.

Programa para rasterizar MicroTriángulos

a

```
# Compute e0, e1 and e2 directed edges in input vertex order:  
a) ADD t1.yzw, i0.zzw, -i0.wwy;  
b) ADD t3.yzw, i1.wwy, -i1.zzw;  
  
# Compute p0, p1 and p2 vectors and -e1.  
c) ADD t2.xyzw, i0.yxxx, -i0.zyzw;  
d) ADD t0.xyzw, i1.yxxx, -i1.zyzw;  
  
# Compute cross products (areal coordinates) and  
# face orientation using fixed-point operations.  
e) FXMUL fx0.xyzw, t1.yyzw, t0.xzwy;  
f) FXMAD t0.xyzw, t3.yyzw, t2.xzwy, fx0.xyzw;  
  
# Test fragment-inside-triangle and cull CW faces.  
g) KIL t0.xyzw;  
  
# Test the tie break rule for edges intersecting the pixel center.  
h) CMP_KIL t2.yzw, -t0.yyzw, c255.xxxx, t1.yyzw;  
i) CMP_KIL t2.yzw, -t2.yyzw, c255.xxxx, t3.yyzw;  
  
# Perspective interpolation of z value at the center  
j) DP3 t2.x, t0.yzww, i2.yzww;  
k) RCP t2.x, t2.x;  
l) MUL t2.x, t2.x, t0.x;  
  
# Compute 1/2A for attribute interpolation  
m) RCP t0.x, t0.x;  
  
# Output fragment's interpolated Z value.  
n) MOV o0.z, t2.x;
```

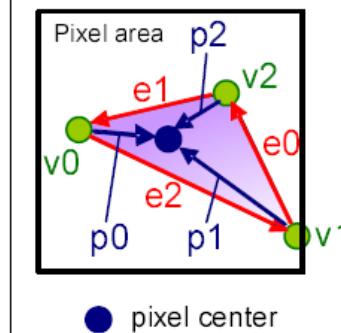
b

Shader inputs				
	x	y	z	w
i0	fr.y + 0.5	v2.y	v0.y	v1.y
i1	fr.x + 0.5	v2.x	v0.x	v1.x
i2	-	1/v2.z	1/v0.z	1/v1.z
i3				
i4				
i5				
...				

Microtriangle vertex attributes

Shader constants				
	x	y	z	w
c255	1.0	-	-	-

c



¡Aumenta la latencia de procesar los fragmentos!

Publicaciones (PFCs)

- Iván Pizarro. Implementación RTL/Verilog de un procesador de shader para una GPU. June 2012.
- Enrique Álvarez. Análisis de aplicaciones 3D en el sistema operativo Google Android. January 2012.
- Daniel Ramírez. Implementació d'un Tesselador de DirectX11 per una tarjeta gràfica. March 2011.
- Albert Murciego. *Anàlisi de jocs Direct3D9 mitjançant estadístiques d'API i comptadors hardware*. July 2010.
- Vicente Escandell. *ACD: A common OpenGL and Direct3D driver for the Attila GPU*. February 2010.
- Christian Perez. *Caracterización e implementación de algoritmos de compresión en la GPU ATILA*. Jan 2008.
- Chema Solis. *Extensión a Direct3D del driver de un simulador de GPU* July 2007.
- David Abella. *Librería Direct3D*. July 2007.
- Jordi Roca. *Shader generation and compilation for a programmable GPU*. July 2005
- Carlos González. *Support tools for a 3D graphics processor simulation framework*. June 2004.

Publicaciones

- Jordi Roca, Victor Moya, Carlos González, Vicente Escandell, Albert Murciego, Agustín Fernández and Roger Espasa. A SIMD-Efficient 14 instruction Shader Program for High-Throughput Microtriangle Rasterization. In Computer Graphics International 2010 (CGI 2010), Singapur, (Singapur), Jun 2010.
- Jordi Roca and Mark Grossman. *Scaling of 3D Game Engine Workloads on Modern Multi-GPU Systems*. In The 1st ACM Conference on High-Performance Graphics (HPG 2009), pp. 37-46, New Orleans, LA (United States), Aug 2009.
- Jordi Roca, Victor Moya, Carlos González, Chema Solis, Agustín Fernández and Roger Espasa. *Workload Characterization of 3D Games*. In 2006 IEEE International Symposium on Workload Characterization (IISWC-2006), pp. 17-26, San José, CA (United States), Oct 2006.
- Victor Moya, Carlos González, Jordi Roca, Agustín Fernández and Roger Espasa. *ATTILA: A Cycle-Level Execution-Driven Simulator for Modern GPU Architectures*. In IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS-2006), pp. 231-241, Austin, TX (United States), Mar 2006.
- Victor Moya, Carlos González, Jordi Roca, Agustín Fernández and Roger Espasa. *A Single (Unified) Shader GPU Microarchitecture for Embedded Systems*. In First International Conference on High Performance Embedded Architectures and Compilers (HiPEAC 2005), pp. 286-301, Barcelona (Spain), Nov 2005.
- Victor Moya, Carlos González, Jordi Roca, Agustín Fernández and Roger Espasa. *Shader Performance Analysis on a Modern GPU Architecture*. In The 38th Annual IEEE/ACM International Symposium on Microarchitecture, 2005 (MICRO-38), pp. 355-364, Barcelona (Spain), Nov 2005.

Conclusiones

Con Attila cubrimos todos los aspectos relacionados con la investigación en GPUs

- Captura trazas de las llamadas a la API gráfica de aplicaciones reales:
 - OpenGL, DirectX9, DirectX10, OpenGL ES en Android
- Convierte las llamadas a la API en llamadas de bajo nivel a la GPU
- Simula ciclo a ciclo la microarquitectura de una GPU.
- La simulación está gobernada por los datos de entrada
- Múltiples configuraciones posibles
- Múltiples estadísticas a varios niveles
 - A nivel de API
 - A nivel de frame
 - A nivel de batch
 - A nivel de ciclo
- Las estadísticas y la traza del tráfico de señales permiten evaluar y depurar las microarquitecturas propuestas.
- También es posible trabajar a nivel de driver.

□ ¡ PROYECTO CERRADO !

Lectura Complementaria

- Jordi Roca, Victor Moya, Carlos González, Chema Solis, Agustín Fernández and Roger Espasa.
Workload Characterization of 3D Games.
In 2006 IEEE International Symposium on Workload Characterization (IISWC-2006), pp. 17-26, San José, CA (United States), Oct 2006.

- Toda el software /documentación de Attila está disponible en:

<https://github.com/attila-gpu/attila-sim> (última act. 2015)

<http://attila.ac.upc.edu>



UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH

Departament d'Arquitectura de Computadors

Tarjetas Gráficas y Aceleradores

Attila

Attila team (www.attila.ac.upc.edu)

Departament d'Arquitectura de Computadors

Facultat d'Informàtica de Barcelona

Universitat Politècnica de Catalunya

