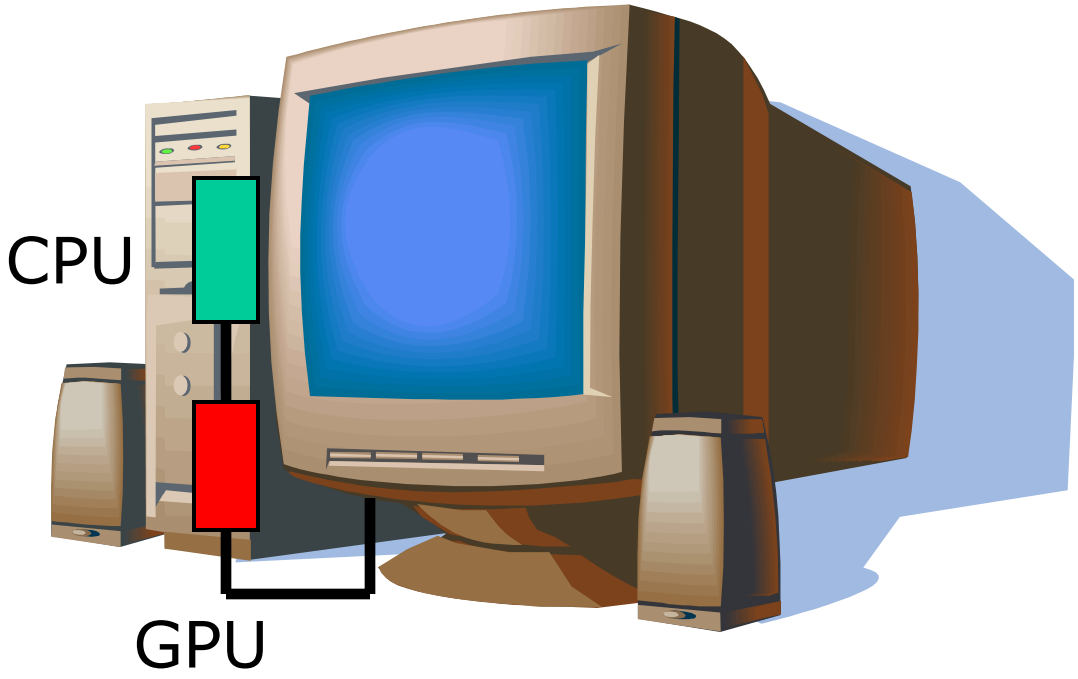
Arquitecturas Orientadas a la Realidad Virtual

Realidad Virtual y Animación

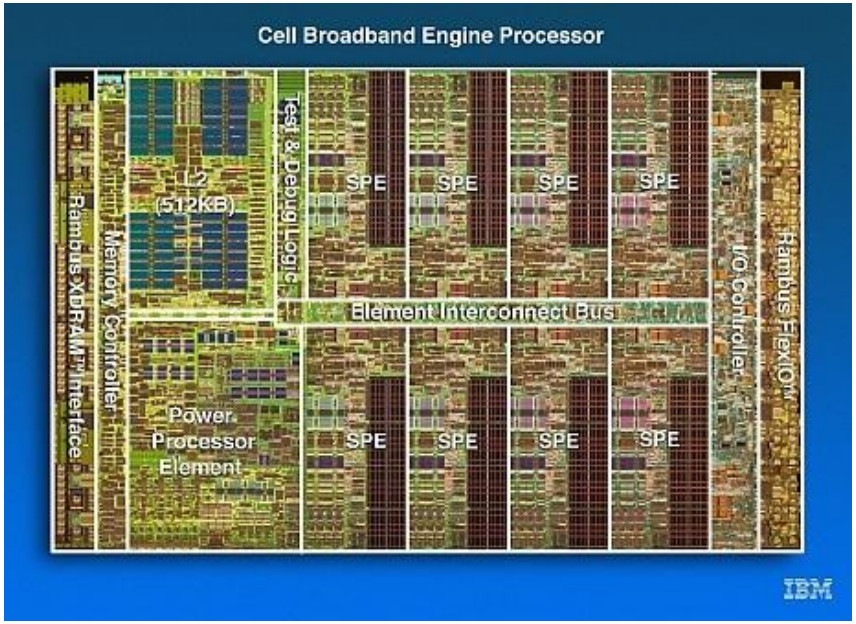
Miguel Ángel Otaduy
Marcos García Lorenzo

Curso 2011/2012

Una arquitectura simple



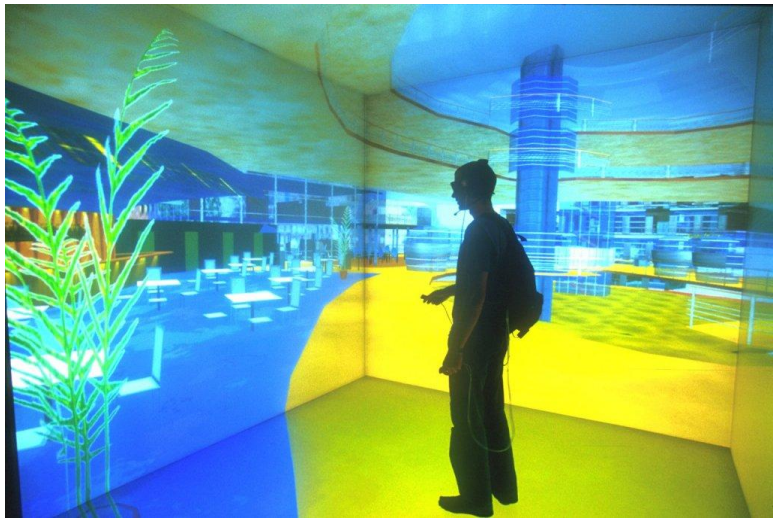
Una arquitectura ¿simple?



CPU multi-core
(IBM cell, Intel Larrabee…)

GPU programable
(NVIDIA Tesla S870)

Múltiples Periféricos



Cueva (CAVE) para realidad virtual

Mosaico de pantallas

Sistema gráfico + háptico

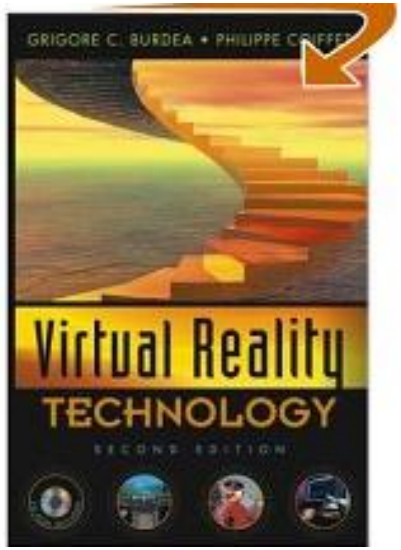
Globalización

Juegos en red

Second Life

Google StreetView

Bibliografía



"Virtual Reality Technology" Ed. Wiley-Interscience (Second Edition). Grigore C. Burdea & Philippe Coiffet.

(sólo arquitecturas para realidad virtual, obsoleto en cuanto a arquitecturas gráficas)

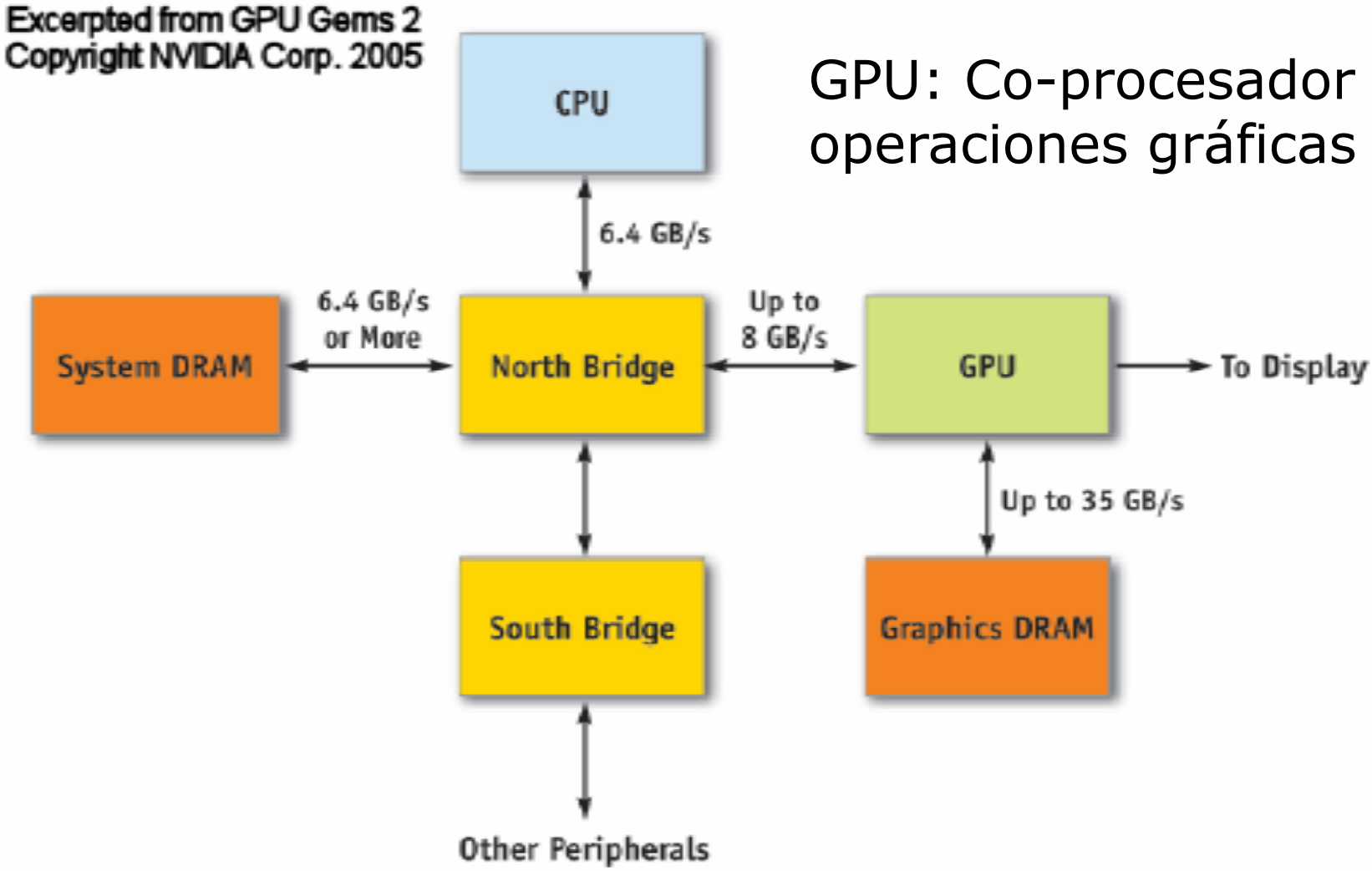
Índice

- Arquitectura de la GPU
- Gráficos Multi-Proceso / Multi-Pantalla
- Gráficos Distribuidos

Índice

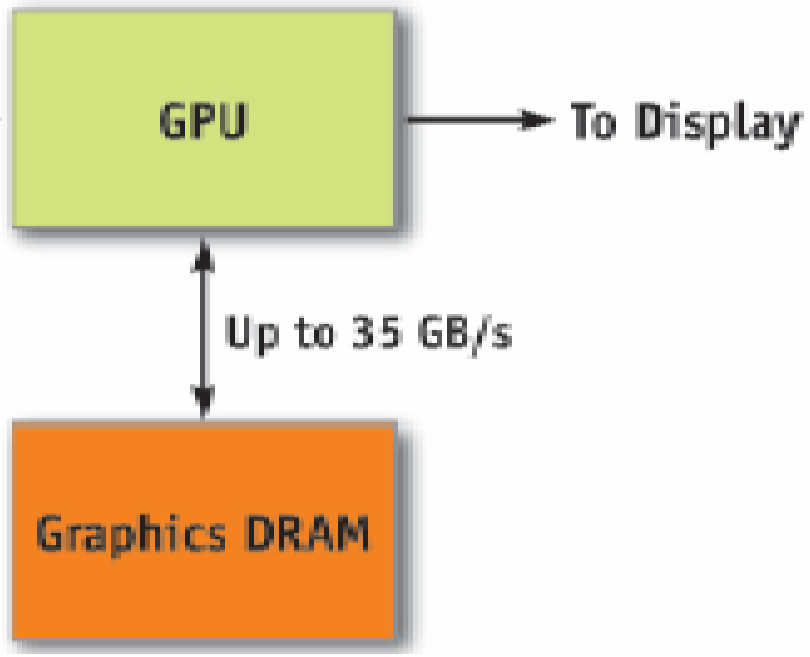
- Arquitectura de la GPU
- Gráficos Multi-Proceso / Multi-Pantalla
- Gráficos Distribuidos

CPU – GPU



GPU: Co-procesador para operaciones gráficas

CPU – GPU

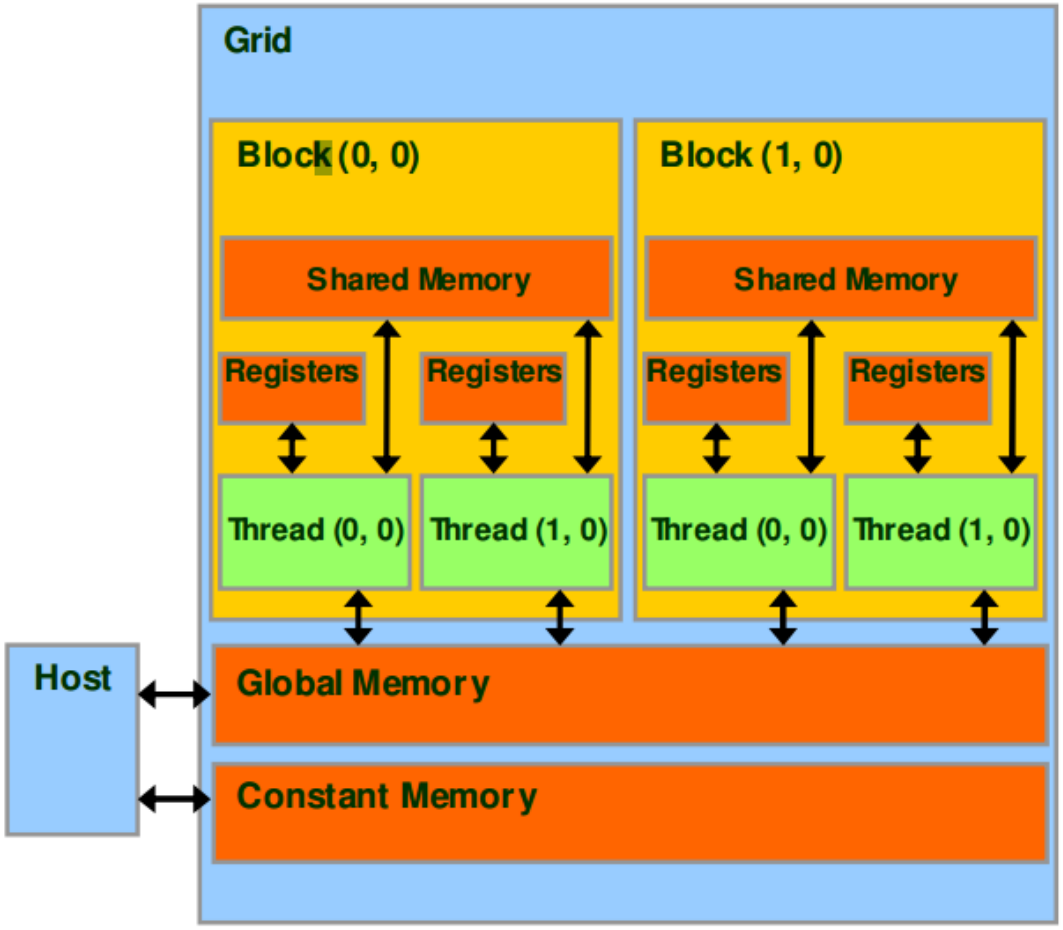


Pieza clave: memoria dinámica de alta velocidad de acceso*

Almacena geometría y texturas

* NVIDIA 9600GT: 512MB a 57.6GB/s

CPU – GPU

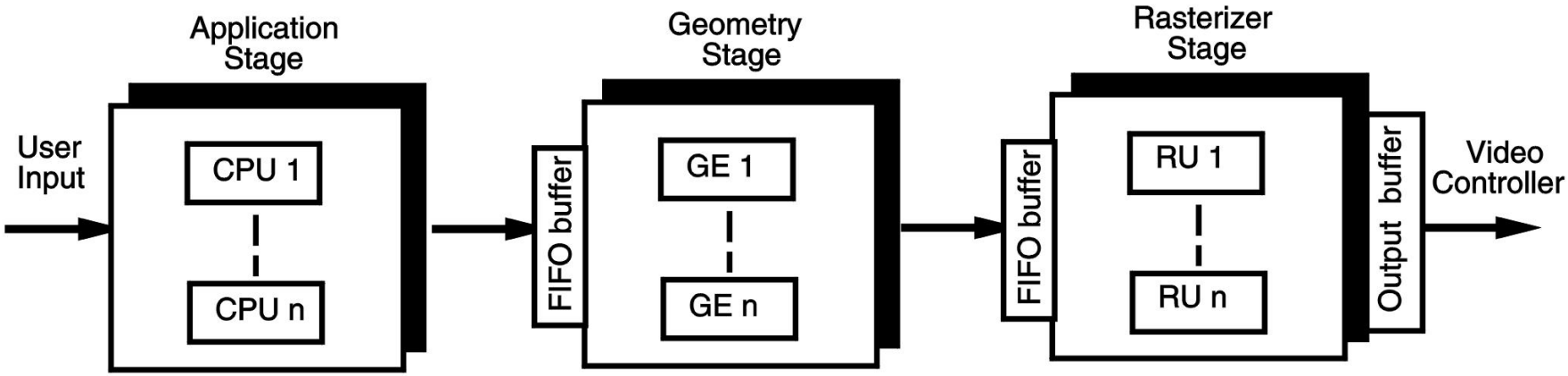

Grid
Block (0, 0)
Block (1, 0)
Shared Memory
Shared Memory
Registers
Registers
Registers
Registers
Thread (0, 0)
Thread (1, 0)
Thread (0, 0)
Thread (1, 0)
Host
Global Memory
Constant Memory

Pipeline

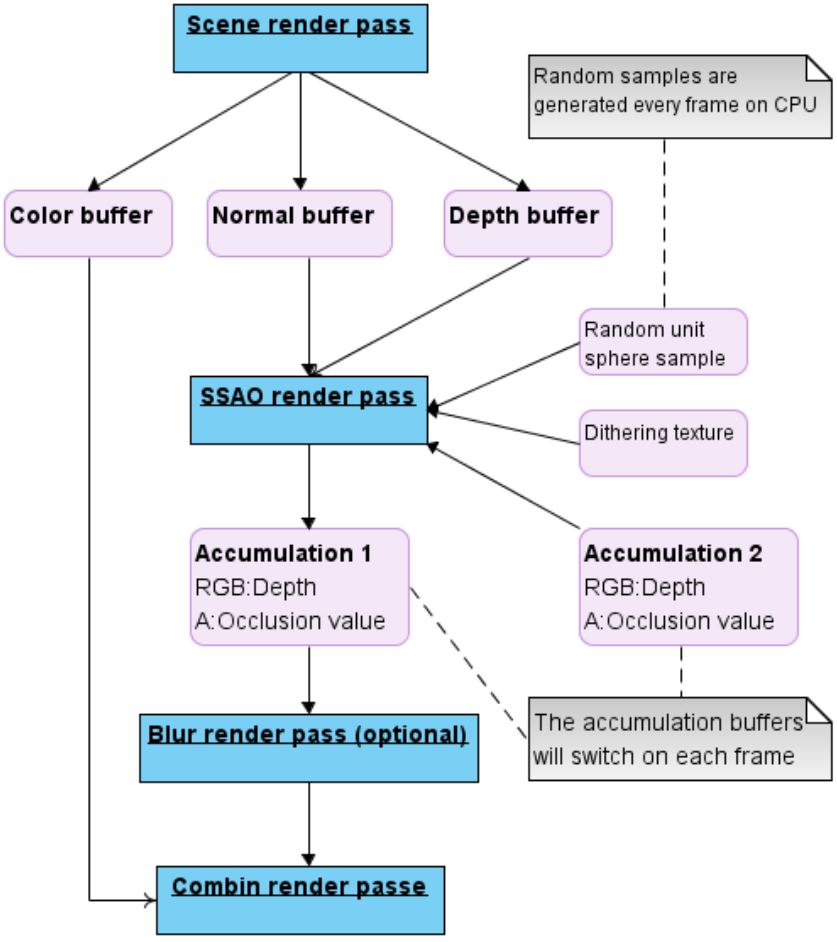
- Definición: división de un proceso en etapas, asignando a cada una de ellas distintos recursos.
- Pipelines
  - CPU
  - Gráfico
  - Háptico

Pipeline gráfico

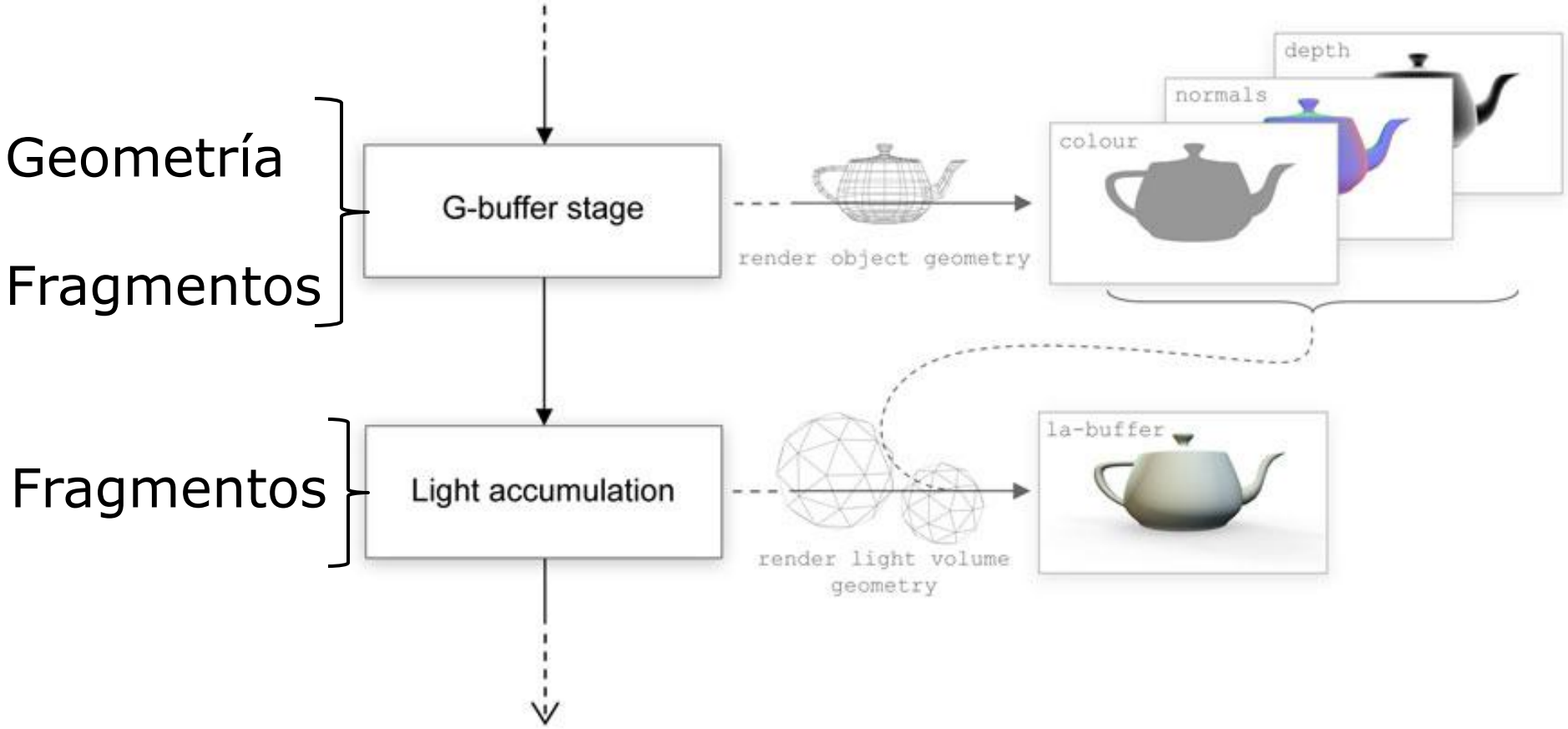
- Paralelización de las etapas de pipeline



Deferred Render

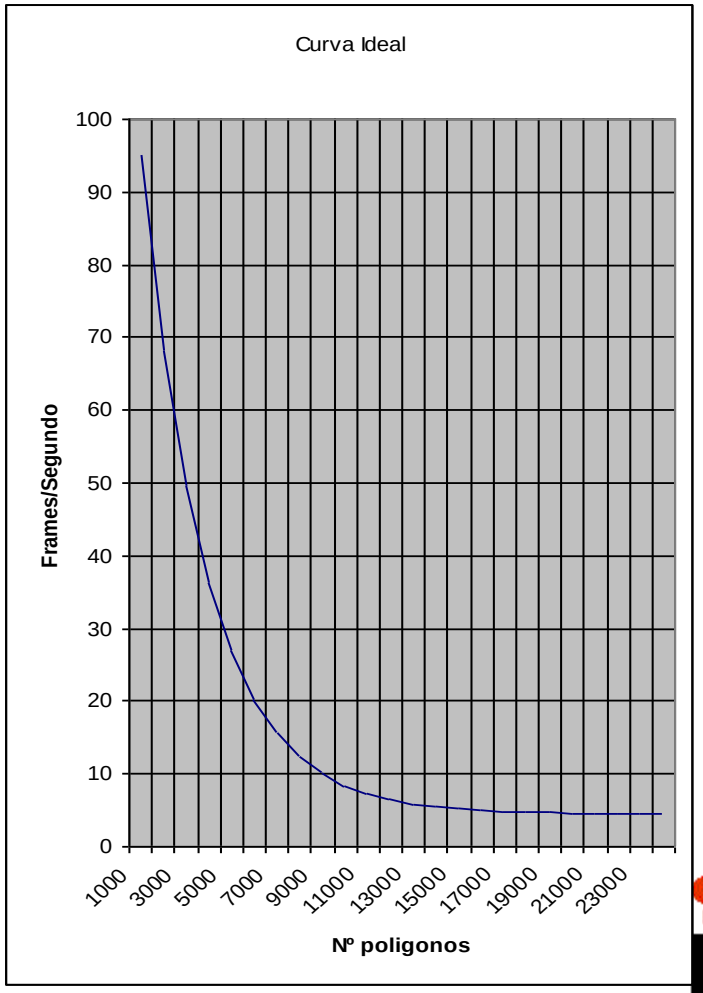


Deferred Render



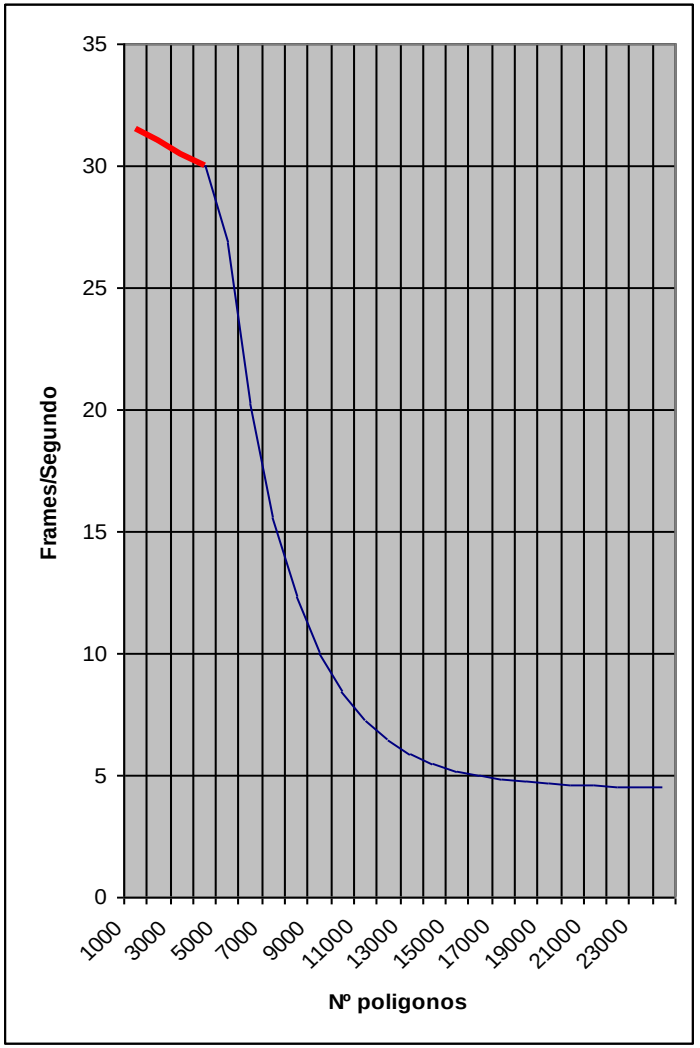
Cuellos de botella

- Óptimo
  - Todas las etapas trabajando al 100%
- Siempre hay etapas que trabajan más
- Curva ideal de funcionamiento
  - Exponencial



Cuellos de botella

- Cuello de botella en la etapa de aplicación al 100%
- CPU-limited



Cuellos de botella

- Cuello de botella en la etapa gráfica
  - Transform-limited
  - Si aumenta los frames por segundo al disminuir el número de luces
- Cuello de botella en la etapa de rasterizado
  - Fill-limeted
  - Si el número de frames disminuye al aumentar la resolución

Soluciones

- CPU-Limited
  - Remplazar la CPU por una más rápida
  - Añadir otra CPU
  - Modificar el código
    - Optimizar el código para una determinada CPU
    - Uso de mejores compiladores
    - Minimizar las multiplicaciones y divisiones
    - Utilizar simple precisión frente a doble precisión
  - Reducir la complejidad de la escena
    - Uso de modelos con menor nivel de detalle

Soluciones

- Transform-limited
  - Reducir el número de luces
  - Modelo de sombreado
    - Flat
    - Gouraud
    - Phong
- Fill-limited
  - Reducción del tamaño de la ventana
  - Reducción de la resolución

Soluciones

- Otras soluciones (si la frecuencia de refresco es buena)
  - Fill-limited: Aumentar la complejidad de la escena, introducir nuevas luces …
  - Transform-limited: Aumentar la resolución

Evolución de las GPUs

- Pipeline asociada a estándares OpenGl

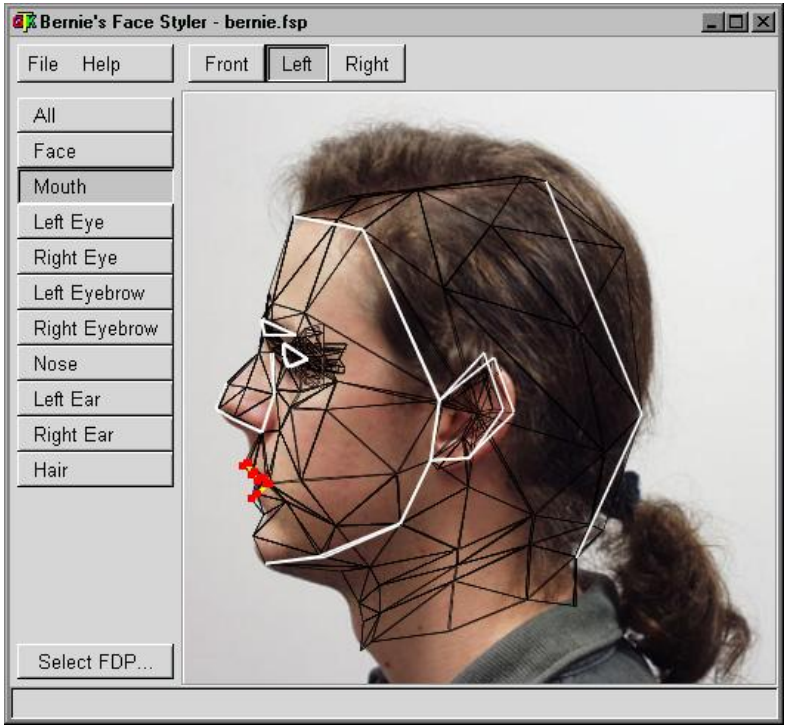
- OpenGl 1.5: Pipeline fija para máxima eficiencia
- OpenGl 2.1: Pipeline programable para mayor versatilidad (Dispositivos movil3es)

- CUDA (NVIDIA): Arquitectura unificada, procesador paralelo general.

- OpenGl 3.3: Desaparecen las etapas fijas
- OpenGl 4.2: Tesselación!!!

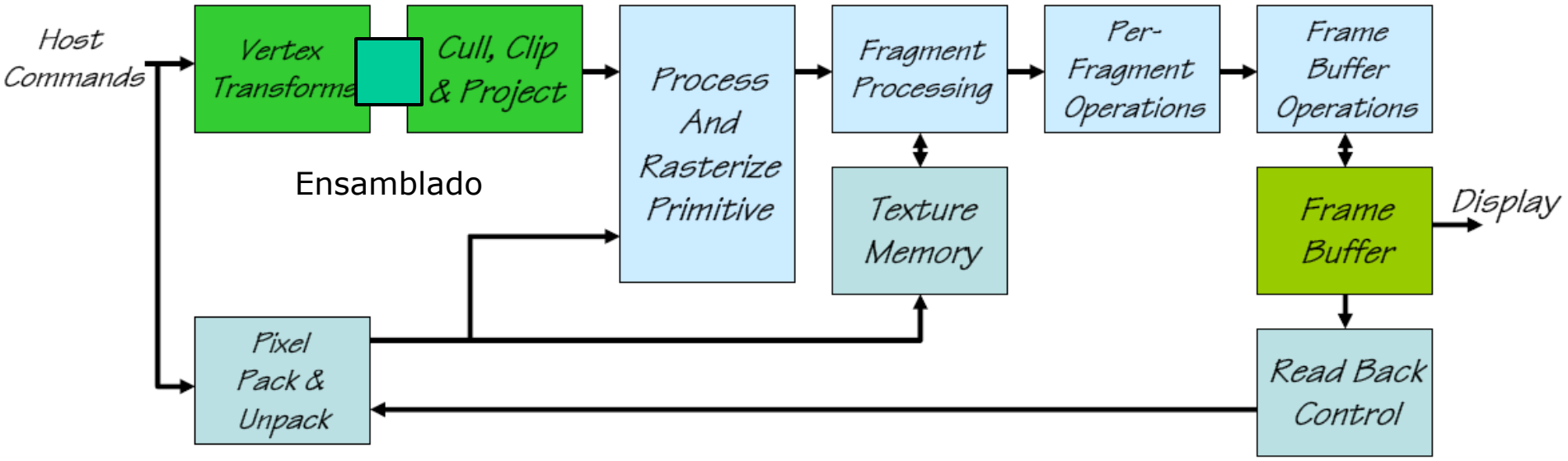
Pipeline Fija

Pipeline Fija



- Trabajo con texturas: impostores, environment mapping, sombras, reflexiones, transparencia…

Pipeline Fija

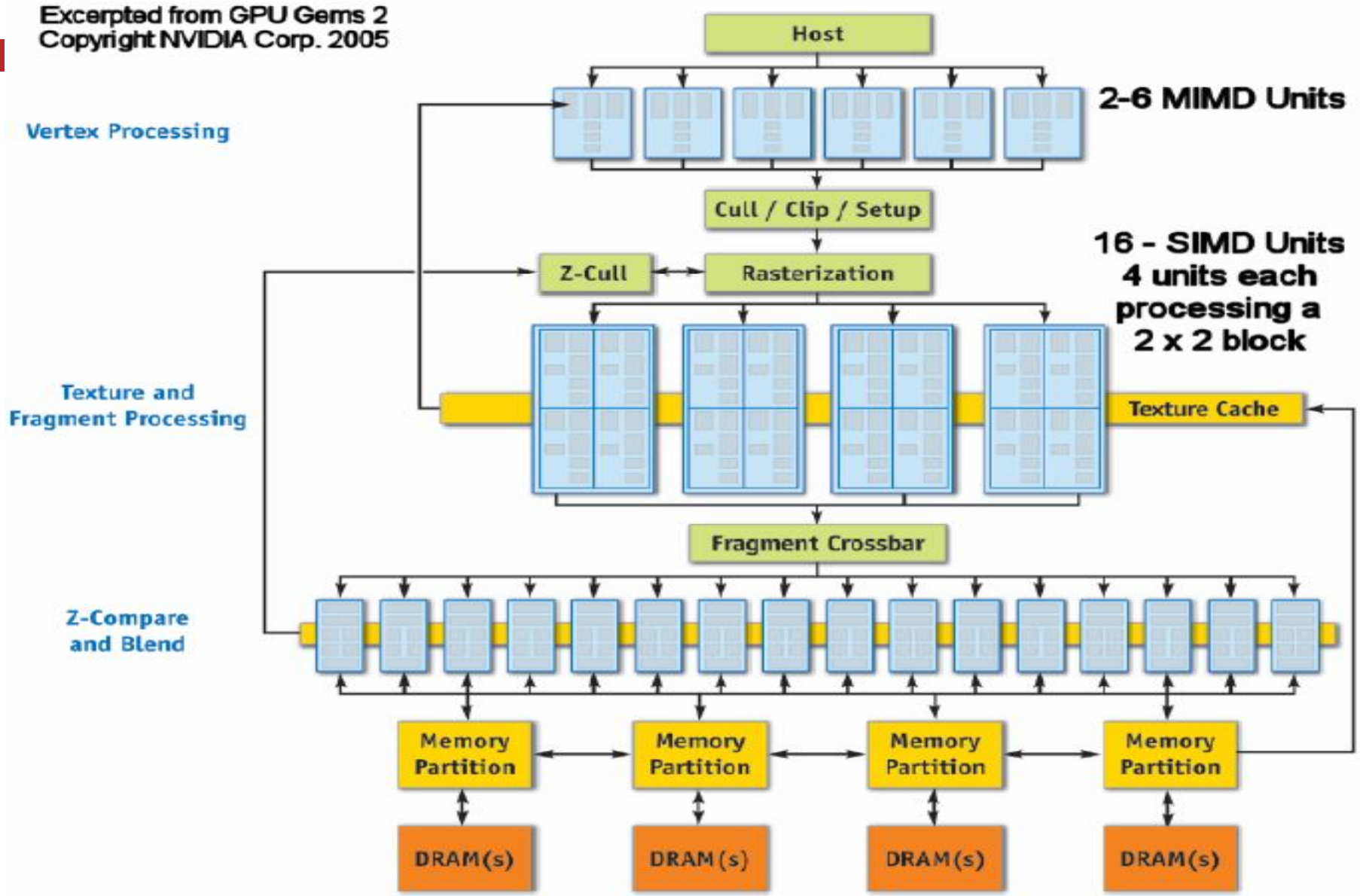


Entradas: primitivas (vértices, polígonos), texturas, comandos

Procesar sólo vértices o fragmentos (pixels). Operaciones fijas.

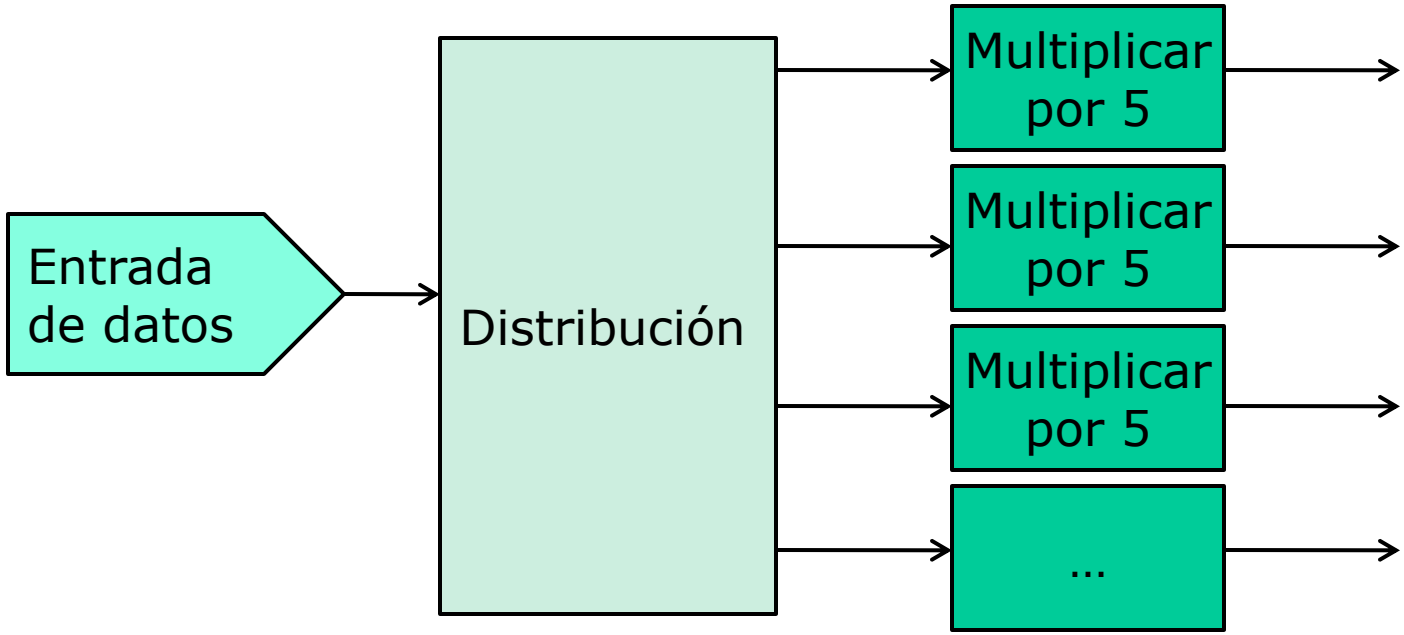
Las operaciones a realizar se determinan mediante el *estado* de OpenGL. El estado se selecciona mediante comandos.

Bloques de la GPU



Stream Processing

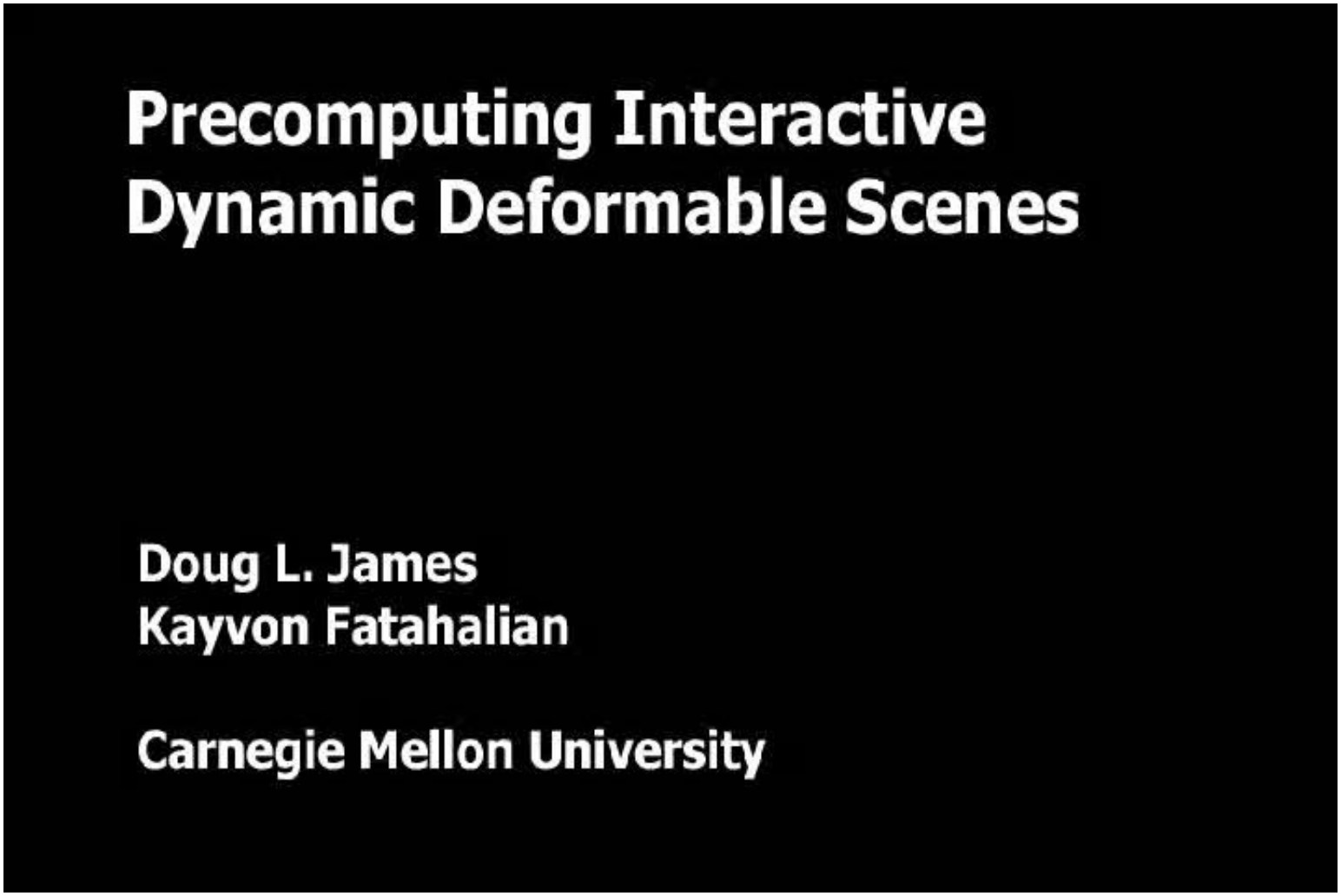
- SIMD: Single instruction, multiple data stream.
- MIMD: Multiple instruction…



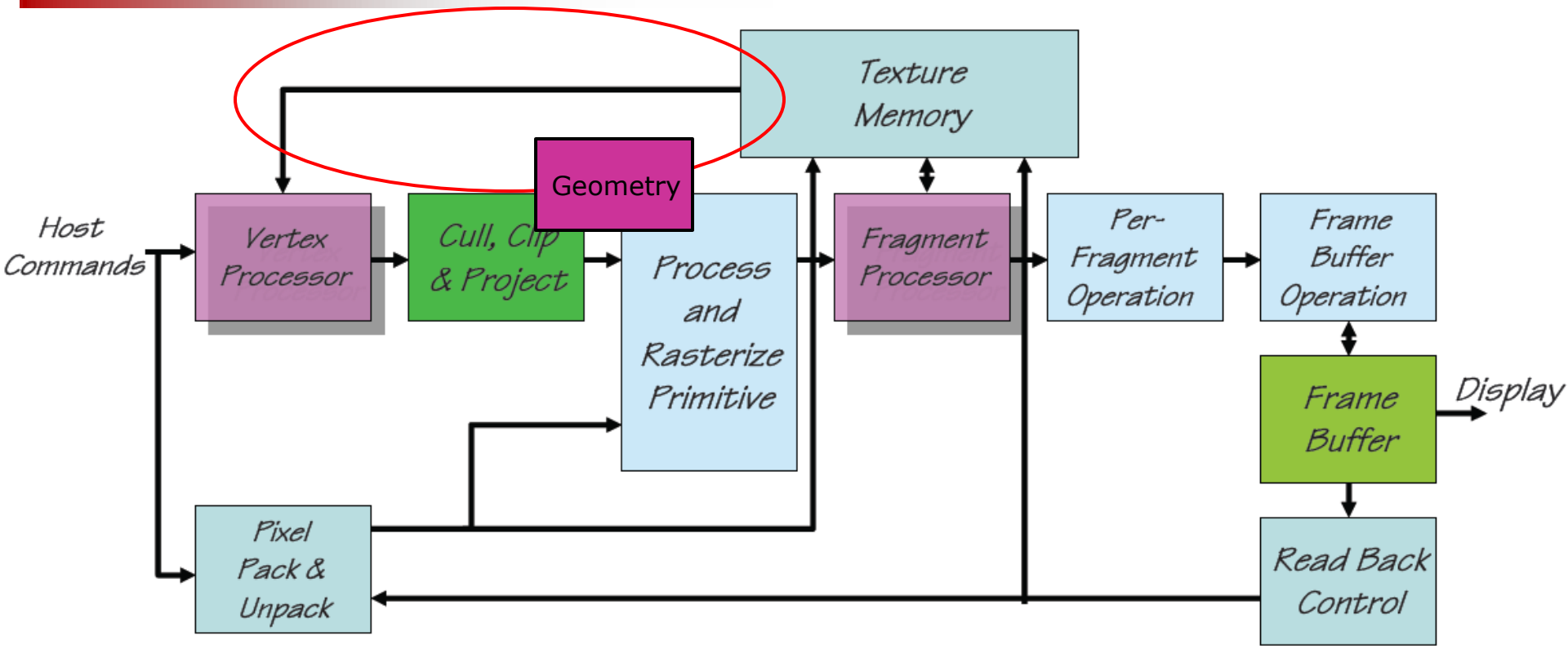
Pipeline Programable

Pipeline Programable

Pipeline Programable



Pipeline Programable



Permite al usuario programa las etapas de vértices, geométrica y fragmentos.

Memoria de textura general: accesible también en el procesado de vértices, y se puede escribir a ella desde el frame buffer!

Pipeline Programable

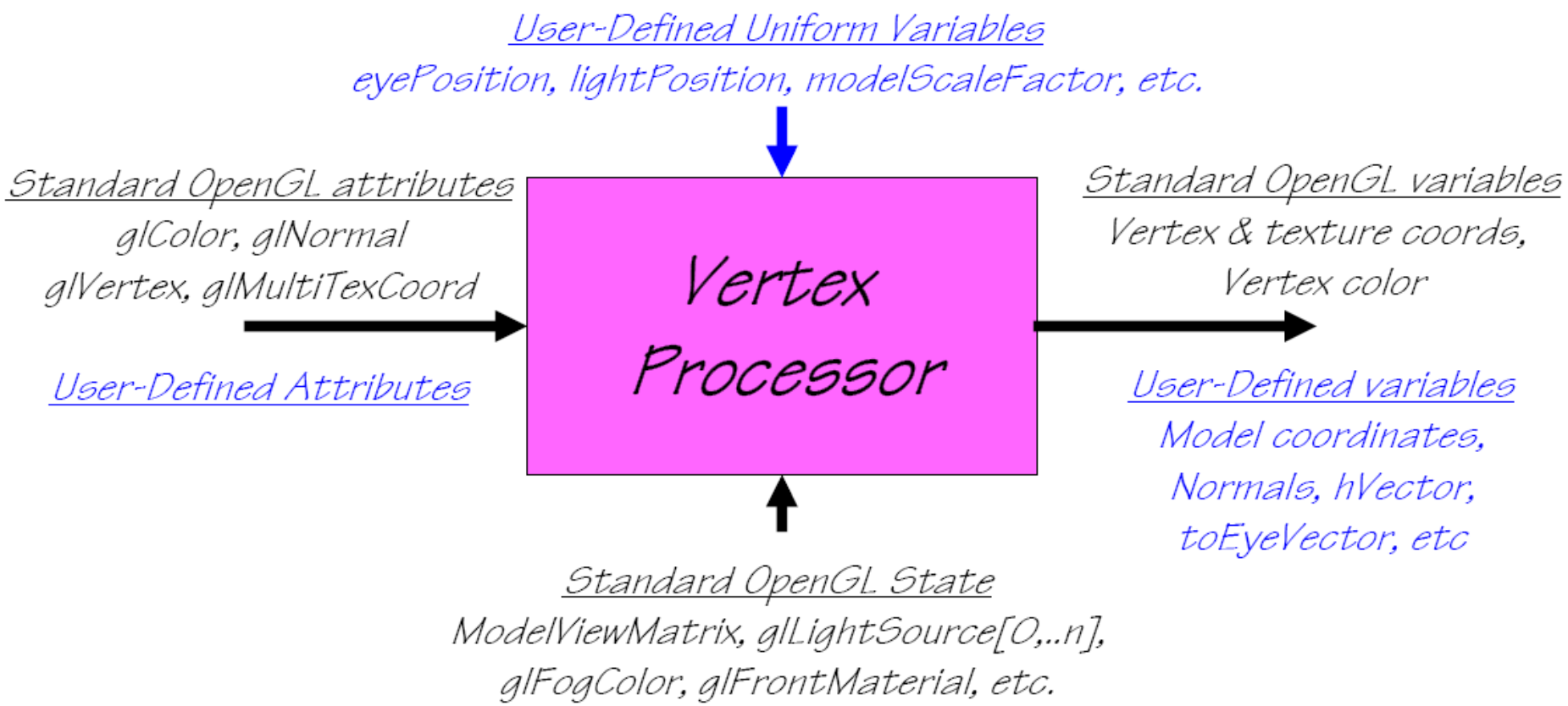
- El pipeline no cambia básicamente, pero se exponen al usuario el procesado de vértices y fragmentos

- Se pueden modificar las funciones, o se pueden diseñar funciones completamente distintas (incluso para aplicaciones no-gráficas!)

Vertex Processing

- Funciones que se pueden sustituir/modificar:
  - Tranformación de coordenadas y normales
  - Normalización, escalado
  - Cálculo de iluminación
  - Aplicación de color
  - Generación y transformación de coordenadas de texturas

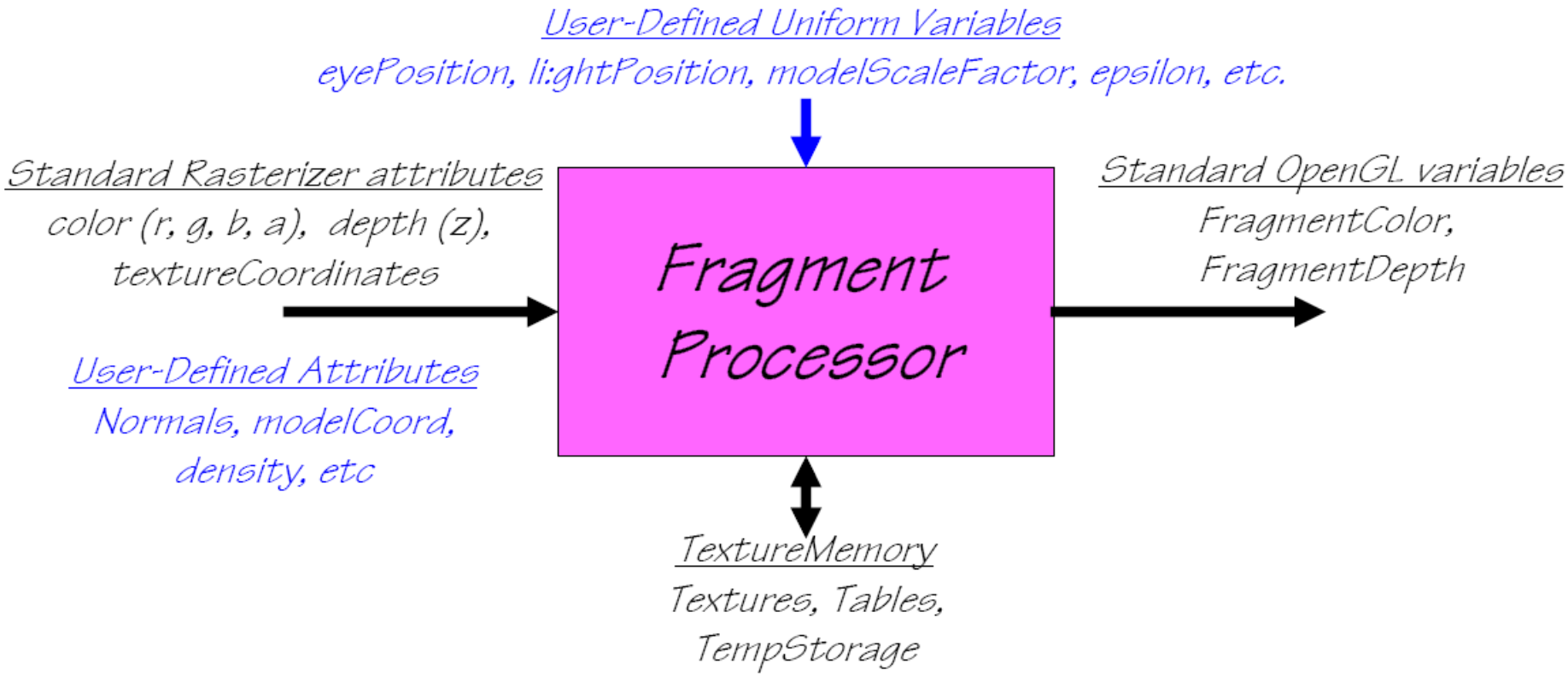
Vertex Processing



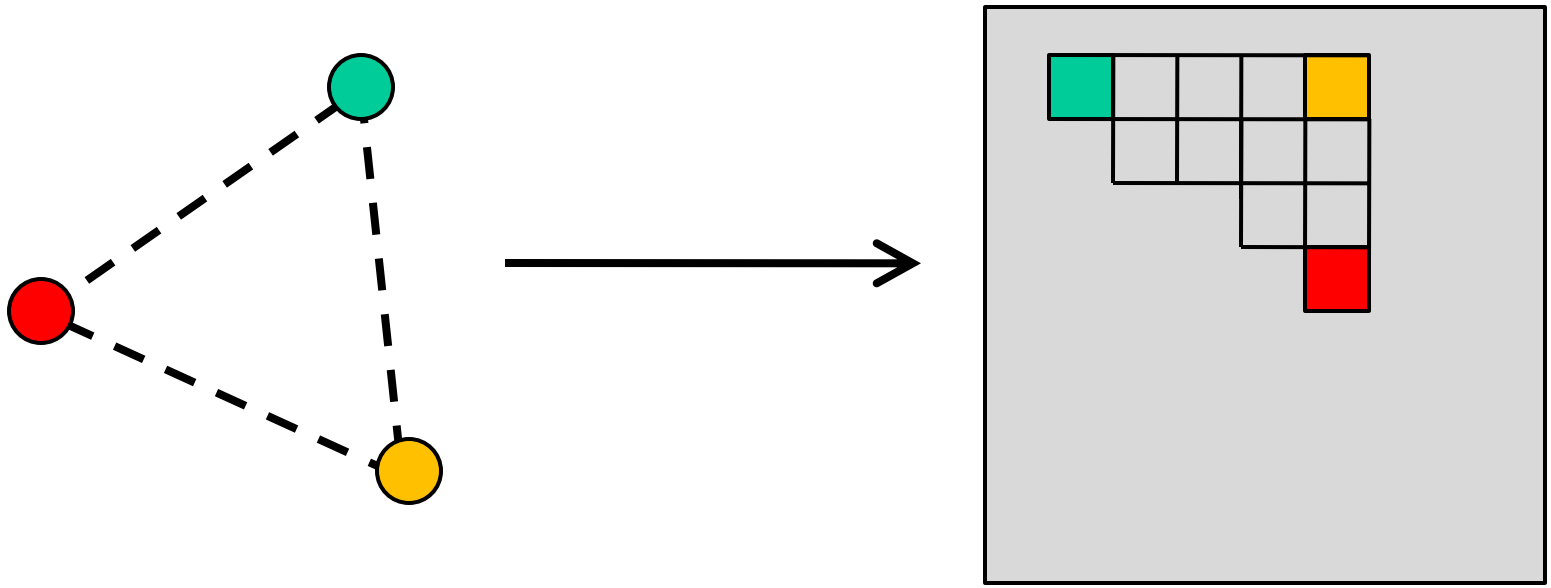
Fragment Processing

- Funciones que se pueden sustituir/modificar:
  - Acceso y aplicación de texturas
  - Suma y mezcla de colores (blending)
  - Profundidad
- Funciones que NO se sustituyen:
  - Z-test (profundidad)
  - Posición del fragmento (scan-conversion)
  - …

Fragment Processing



Limitaciones



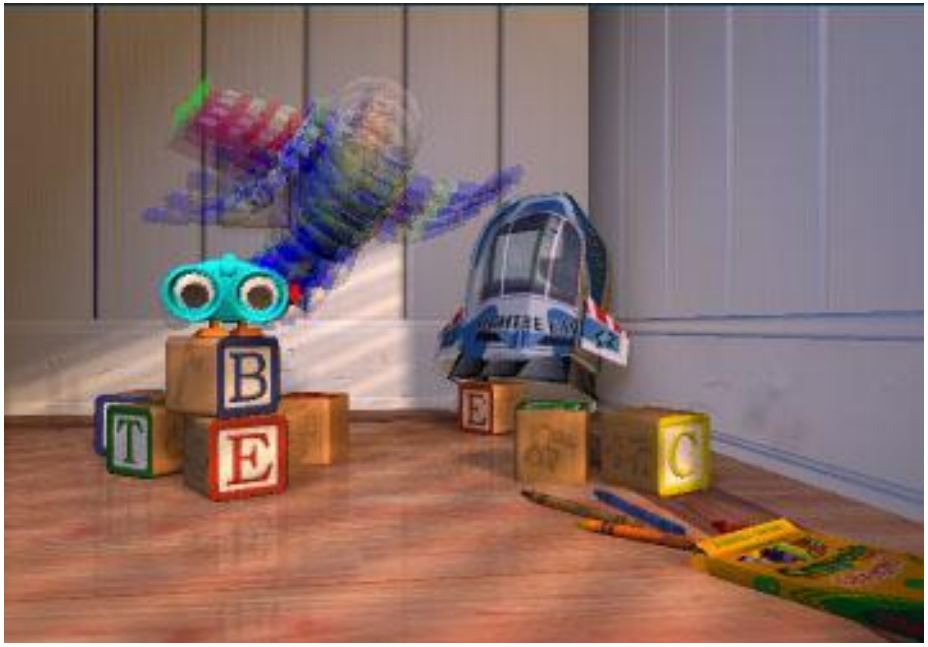
No se puede crear geometría (nuevos vértices)

No se puede cambiar la posición de los fragmentos

Multi-Pass Rendering

- Para crear efectos complejos, realizar varios pases del pipeline, y en cada pase una operación distinta
- El resultado de un pase se puede escribir directamente a la memoria de textura (render to texture)
- En un nuevo pase, se leen los datos escritos a las texturas
- Programas: Frame Buffer Object (FBO)

Ejemplo 1: Motion Blur (Efectos de Movimiento)



4 pases

16 pases

Ejemplo 2: Depth of Field (Enfoque de la Cámara)

Sin enfoque

Con enfoque

Ejemplo 3: Soft Lighting (Luces con Área)

Luz puntual

Luz con área

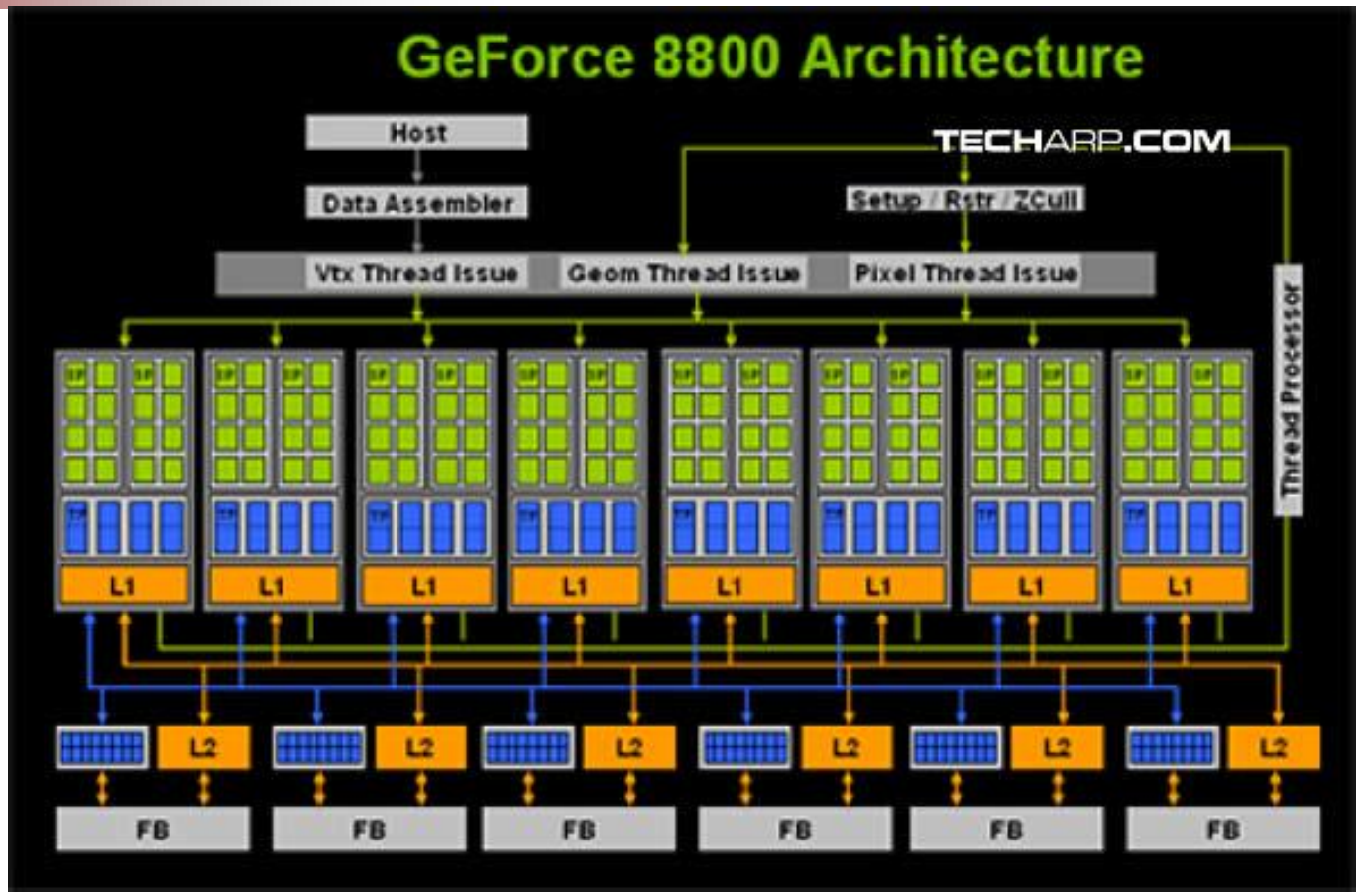
Lenguajes de Shading

- Cg (NVIDIA): se programa y compila por separado, y se carga desde la aplicación
- GLSL (OpenGL): se programa directamente en la aplicación, y se compila *on-the-fly*; se puede modificar!
- HLSL (DirectX): similar pero de Microsoft

Ejemplo Cg

```
// input vertex
struct VertIn {
    float4 pos : POSITION;
    float4 color : COLOR0;
};
// output vertex
struct VertOut {
    float4 pos : POSITION;
    float4 color : COLOR0;
};
// vertex shader main entry
VertOut main(VertIn IN, uniform float4x4 modelViewProj) {
    VertOut OUT;
    OUT.pos = mul(modelViewProj, IN.pos); // calculate output coords
    OUT.color = IN.color; // copy input color to output
    OUT.color.z = 1.0f; // blue component of color = 1.0f
    return OUT;
}
```

Pipeline Unificada

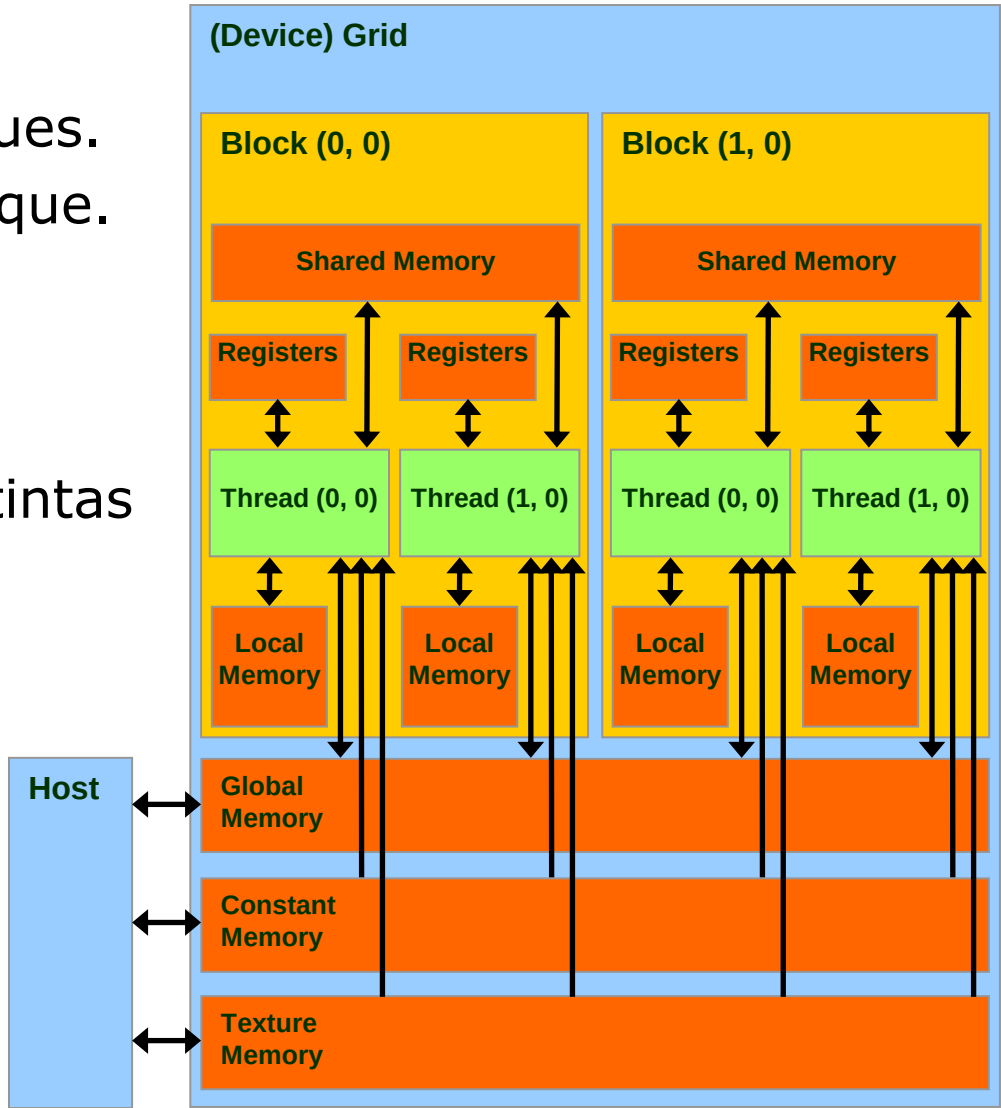


Massive parallel processing: multitud de procesadores ejecutando lo mismo. Desde la CPU, lanzar hebras e indicar cuántas se ejecutan, dónde, y qué memoria utilizan.

Bloques de Hebras

- Hebras distribuidas por bloques.
- Comparten memoria por bloque.
- Identificador de bloque y de hebra.

- 3 tipos de memoria, con distintas propiedades y aplicabilidad



Comunicación CPU-GPU

Creación/destrucción de memoria y transferencia de datos (parecido a C):

- cudaMalloc((void**)&Md.elements, size);
- cudaFree(Md.elements);
- cudaMemcpy(Md.elements, M.elements, size, cudaMemcpyHostToDevice);
- cudaMemcpy(M.elements, Md.elements, size, cudaMemcpyDeviceToHost);

Código CUDA

- En general, se programa como en C
- Funciones (indicando quién las llama y dónde se ejecutan):

	Ejecutada en	Llamada por
`__device__ float DeviceFunc()`	GPU	GPU
`__global__ void KernelFunc()`	GPU	CPU
`__host__ float HostFunc()`	CPU	CPU

Ejemplo CUDA

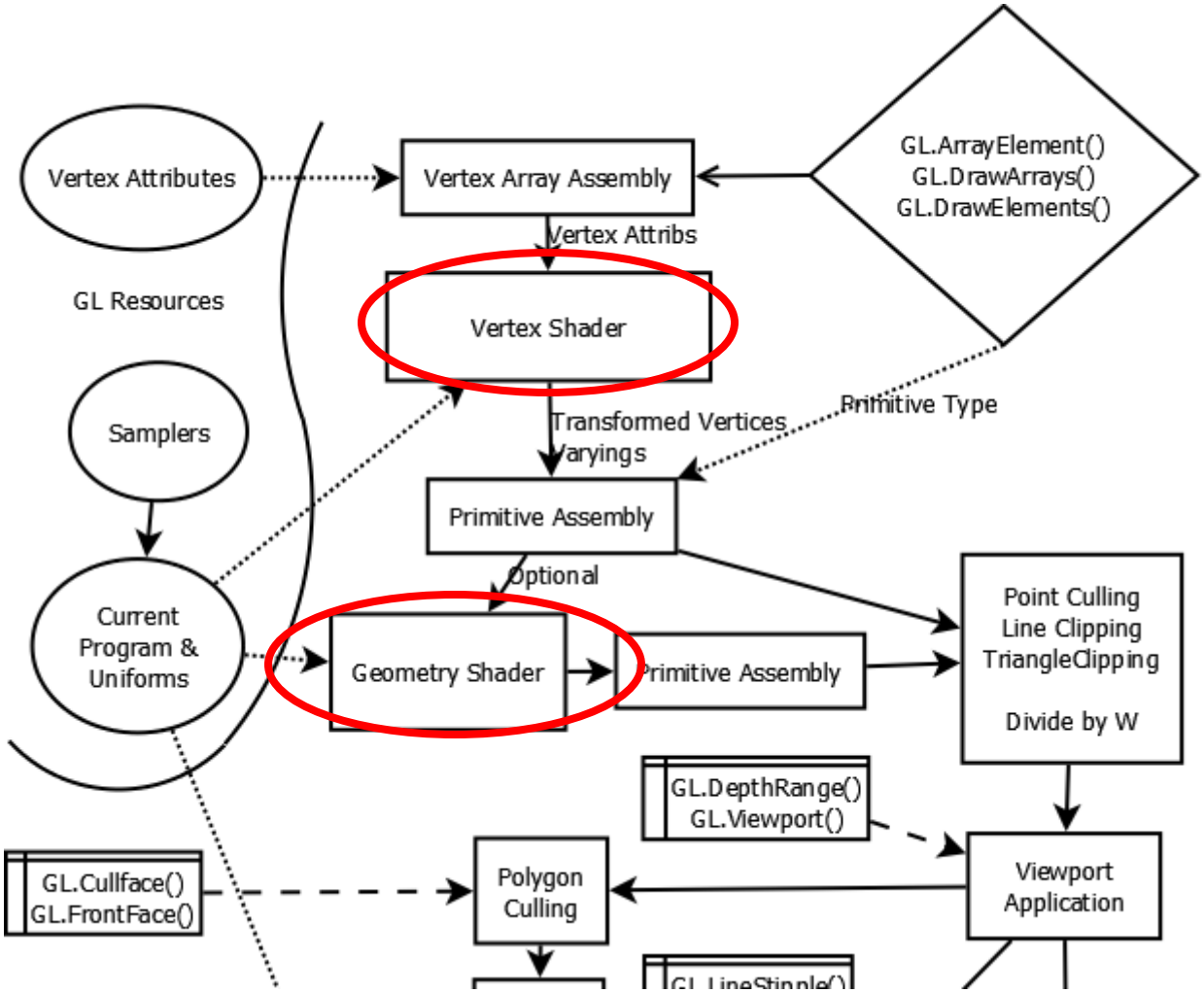
http://www.youtube.com/watch?v=VpEpAFGplnI

http://www.youtube.com/watch?v=HScYuRhgEJw

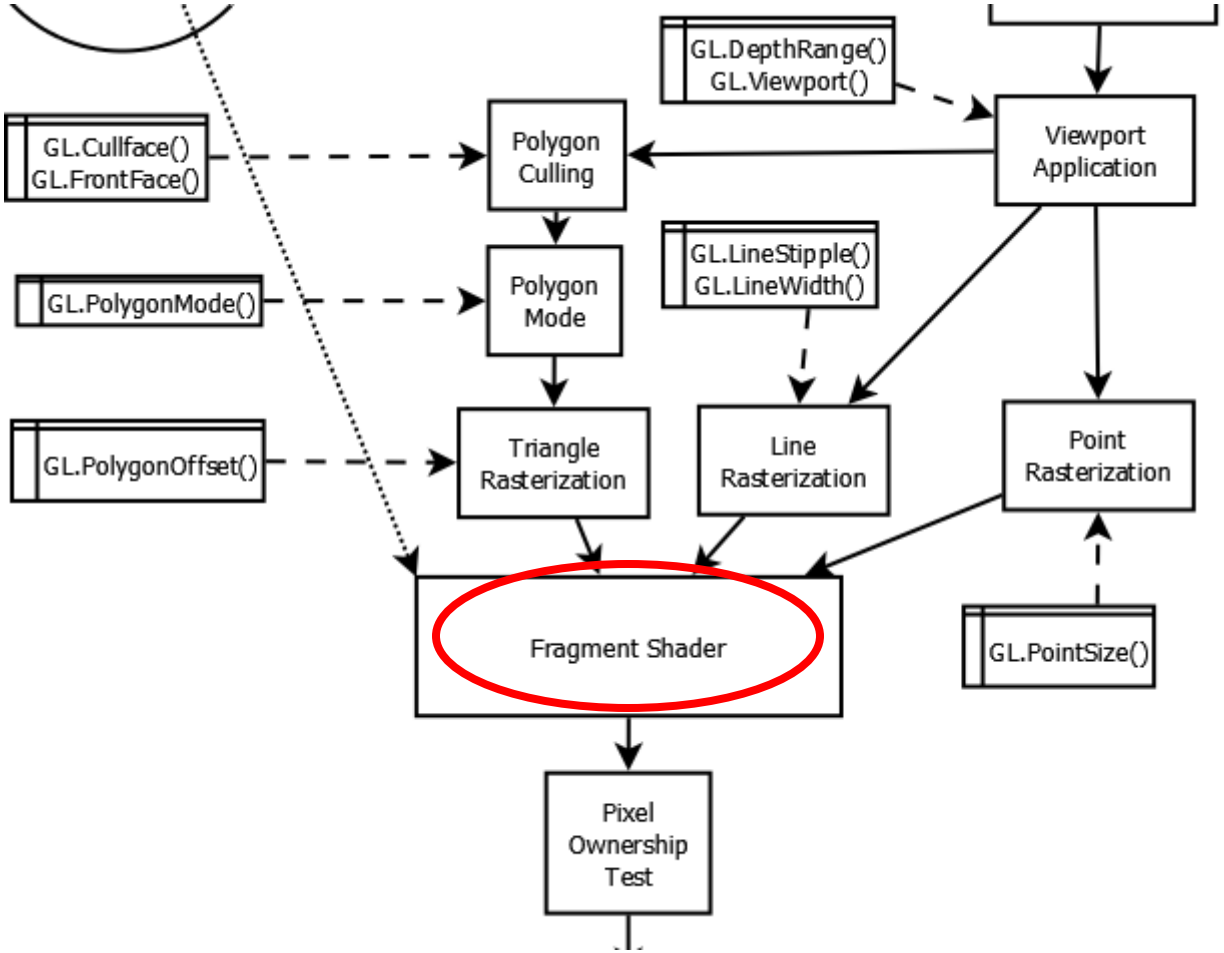
http://research.nvidia.com/publication/efficient-sparse-voxel-octrees

Ejemplo CUDA

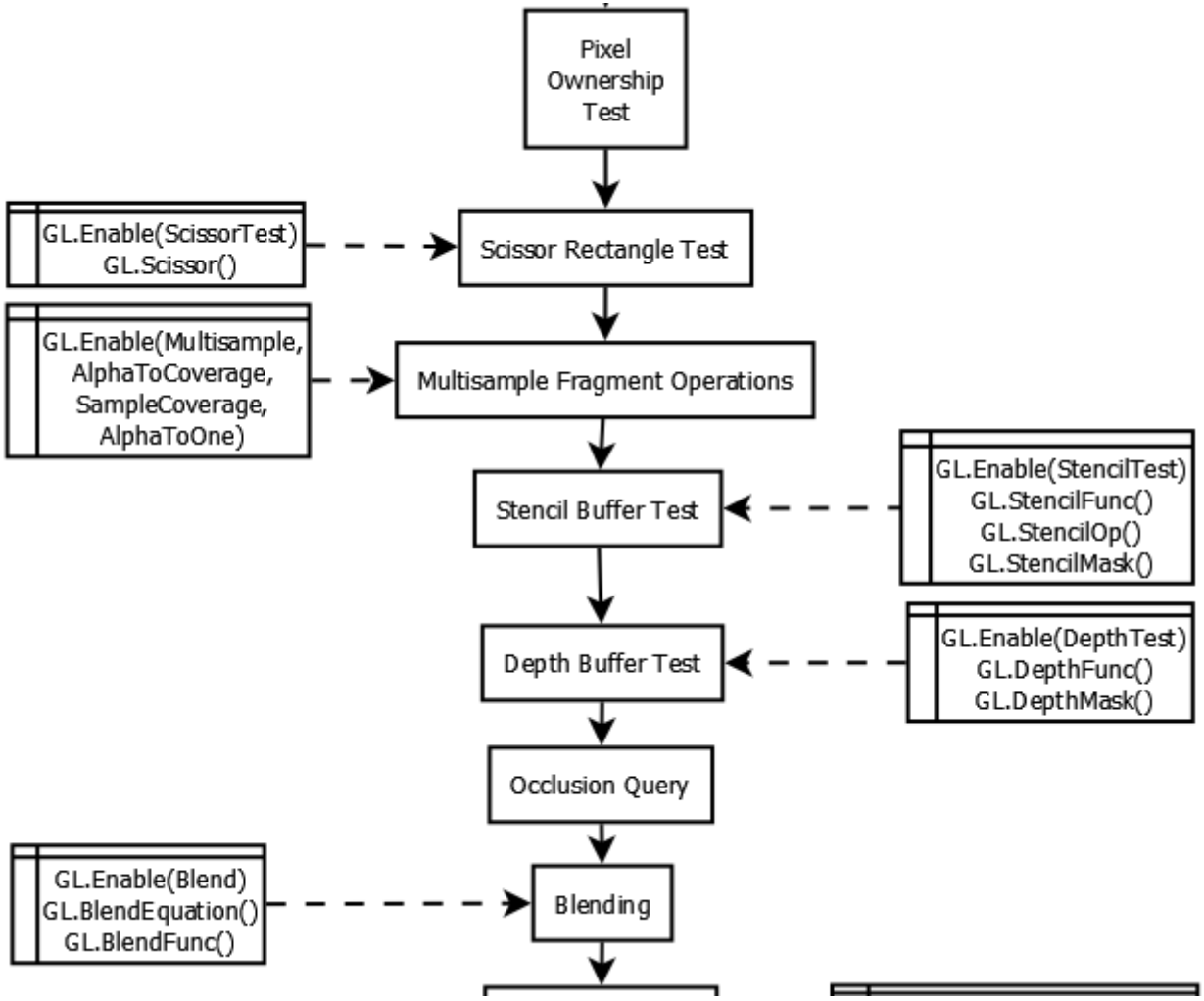
OpenGL 3.3 (Compatibility)



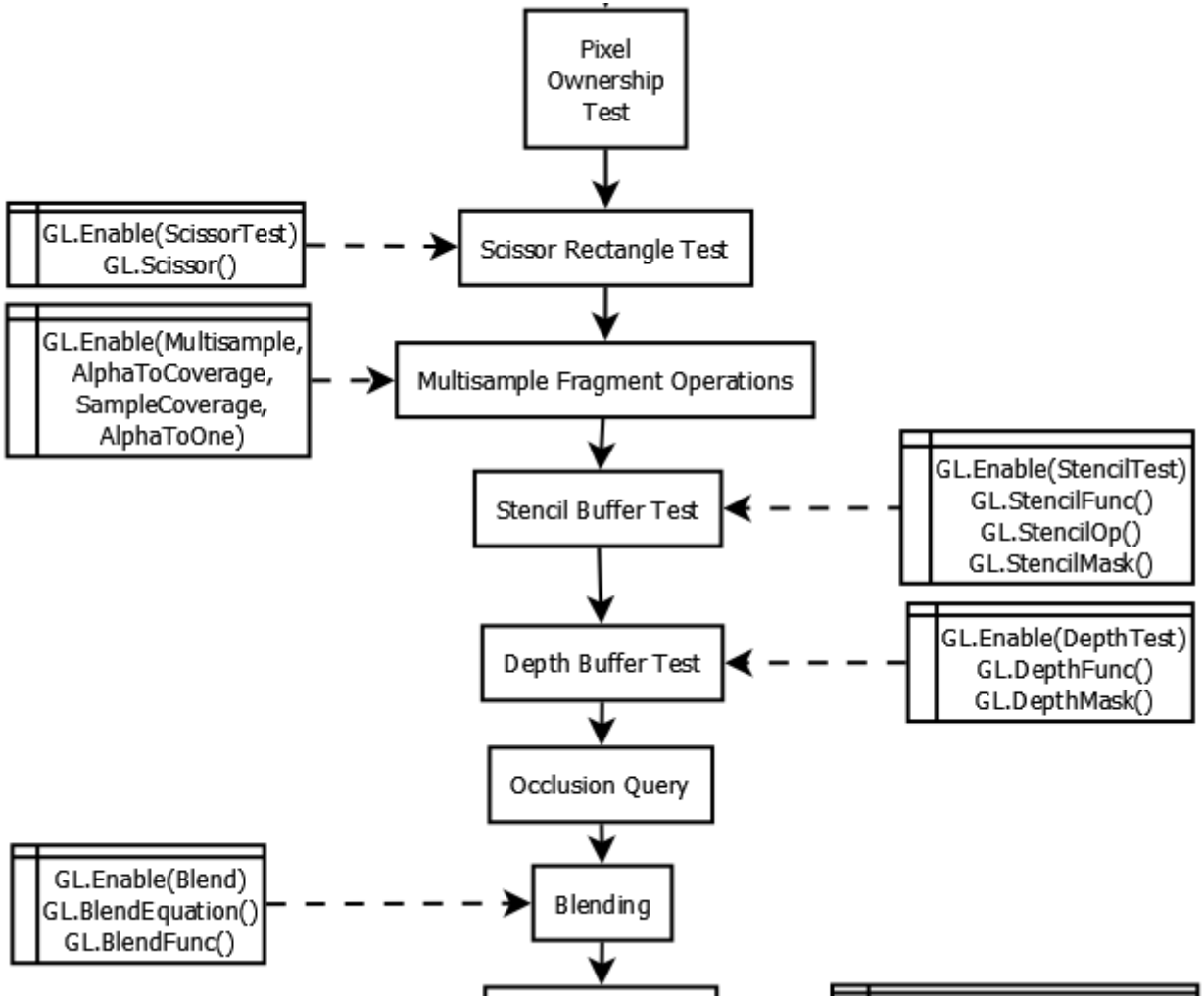
OpenGL 3.3 (Compatibility)



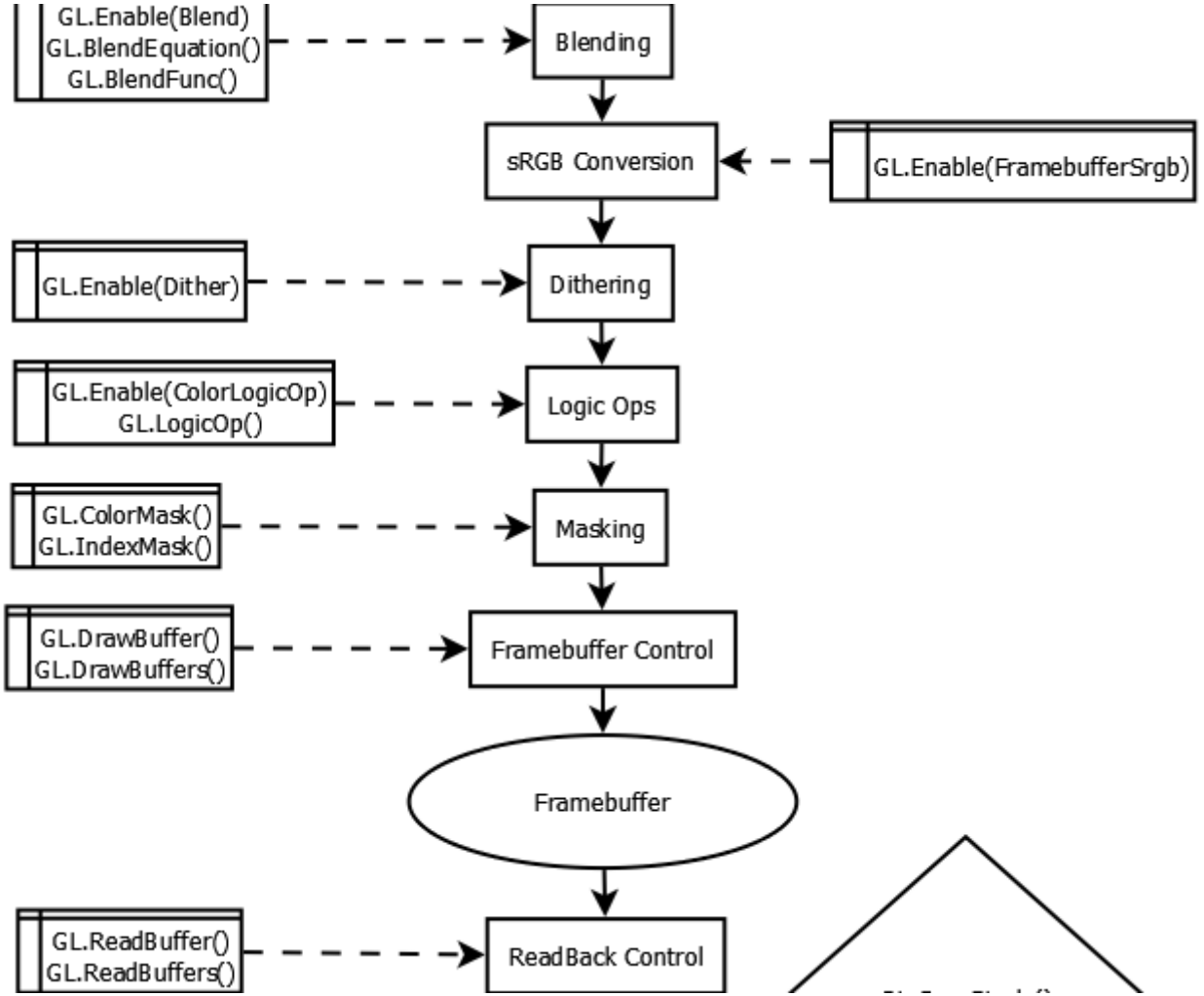
OpenGL 3.3 (Compatibility)



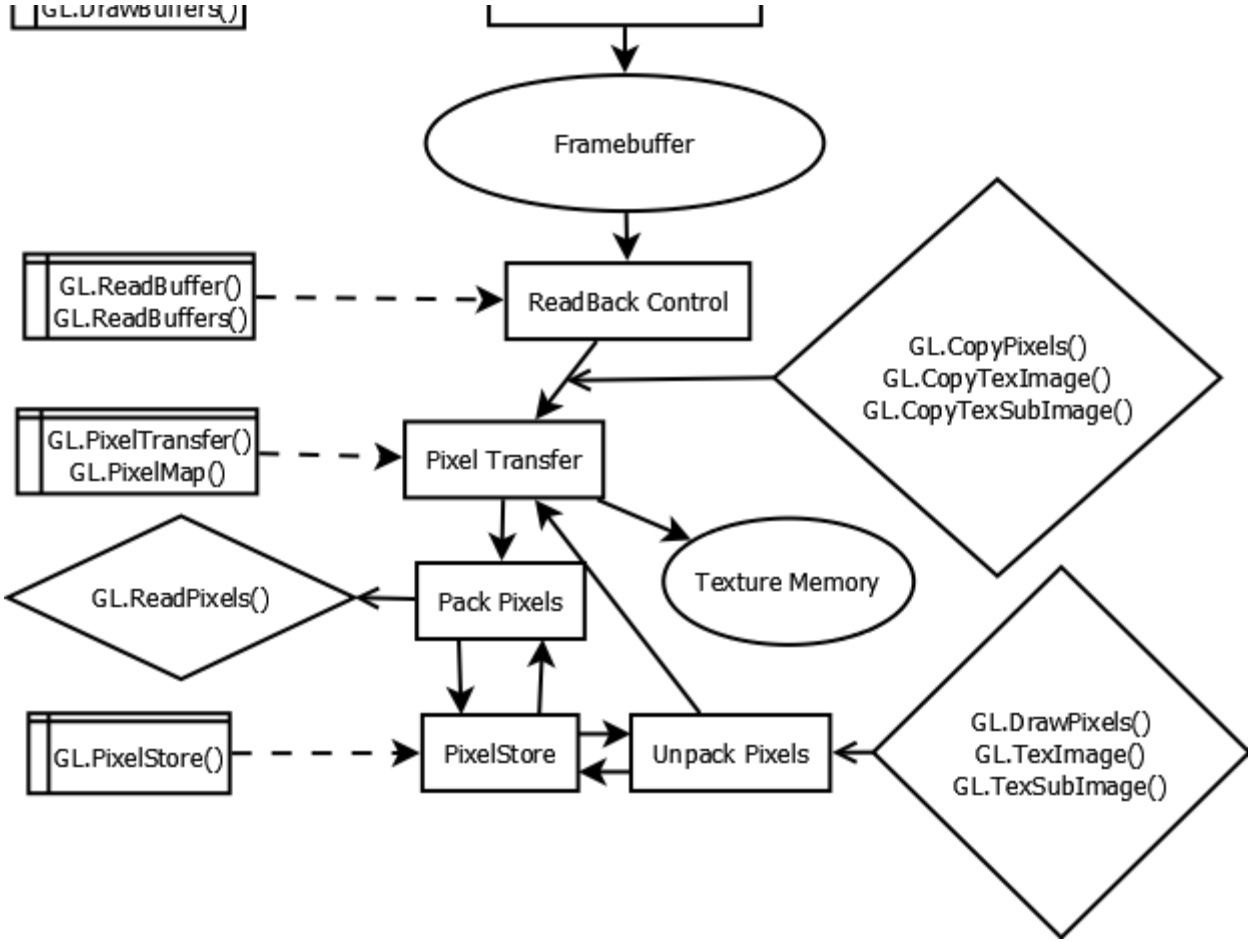
OpenGL 3.3 (Compatibility)



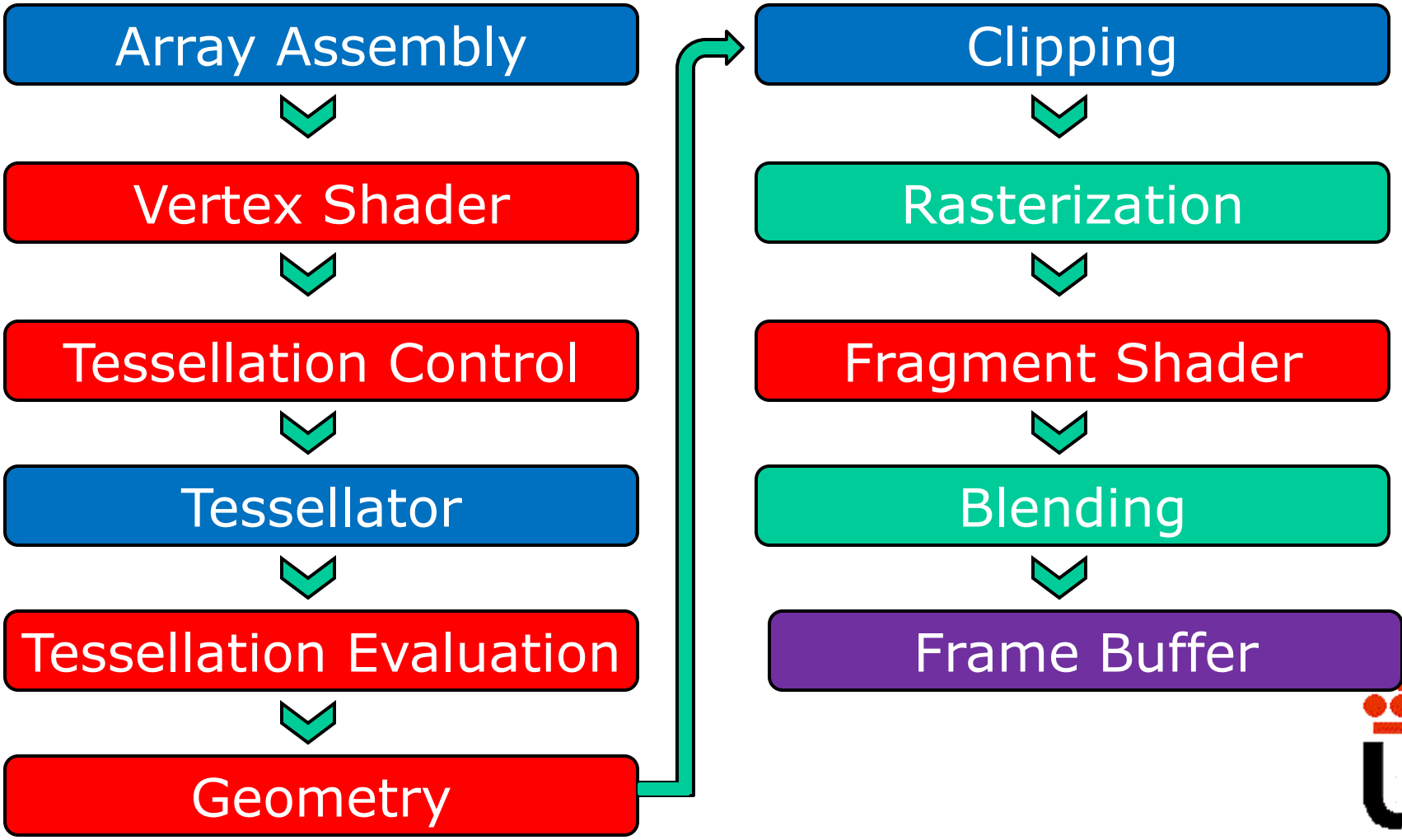
OpenGL 3.3 (Compatibility)



OpenGL 3.3 (Compatibility)



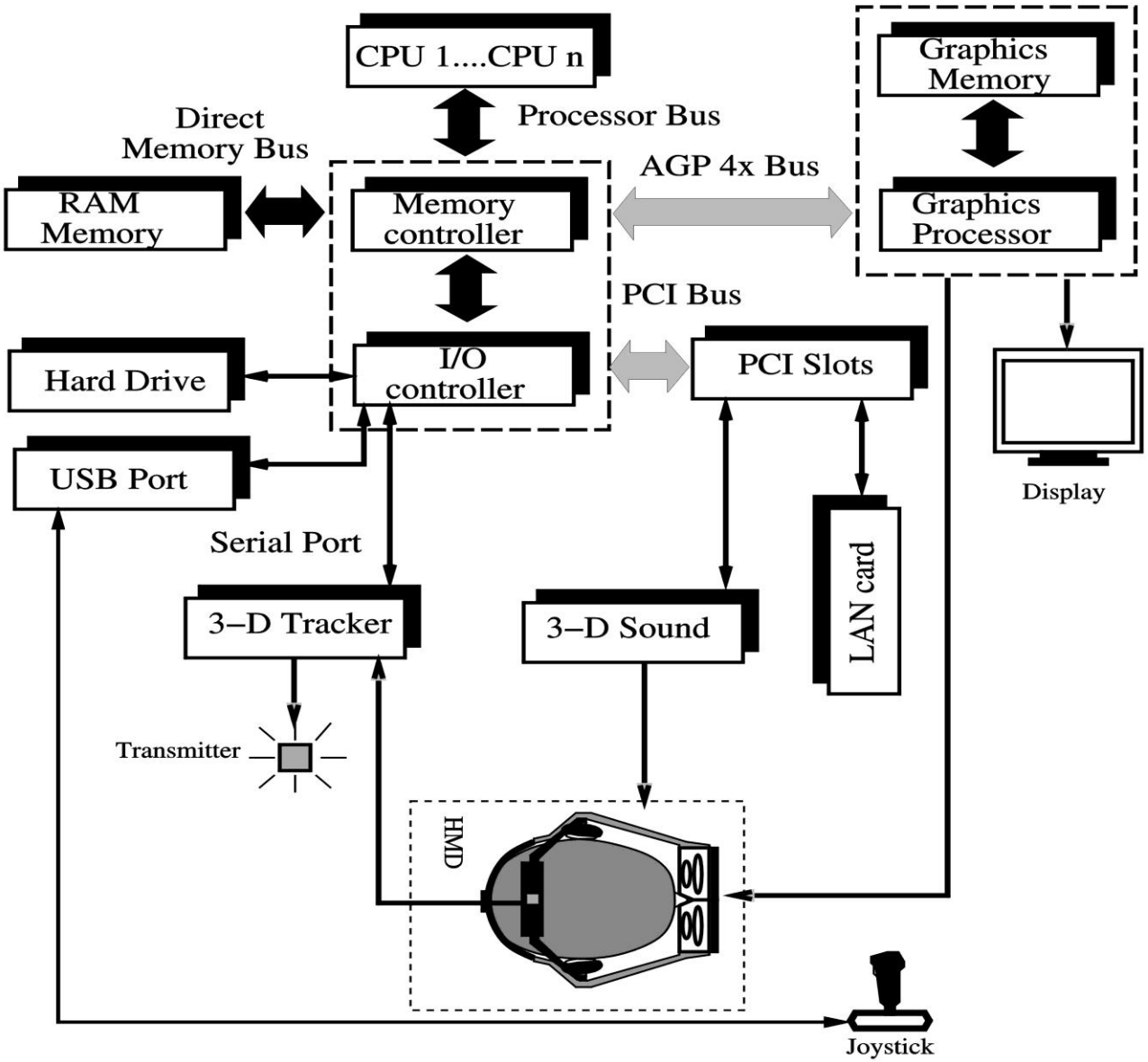
OpenGL 4.2


Array Assembly
Vertex Shader
Tessellation Control
Tessellator
Tessellation Evaluation
Geometry
Clipping
Rasterization
Fragment Shader
Blending
Frame Buffer

Índice

- Arquitectura de la GPU
- Gráficos Multi-Proceso / Multi-Pantalla
- Gráficos Distribuidos

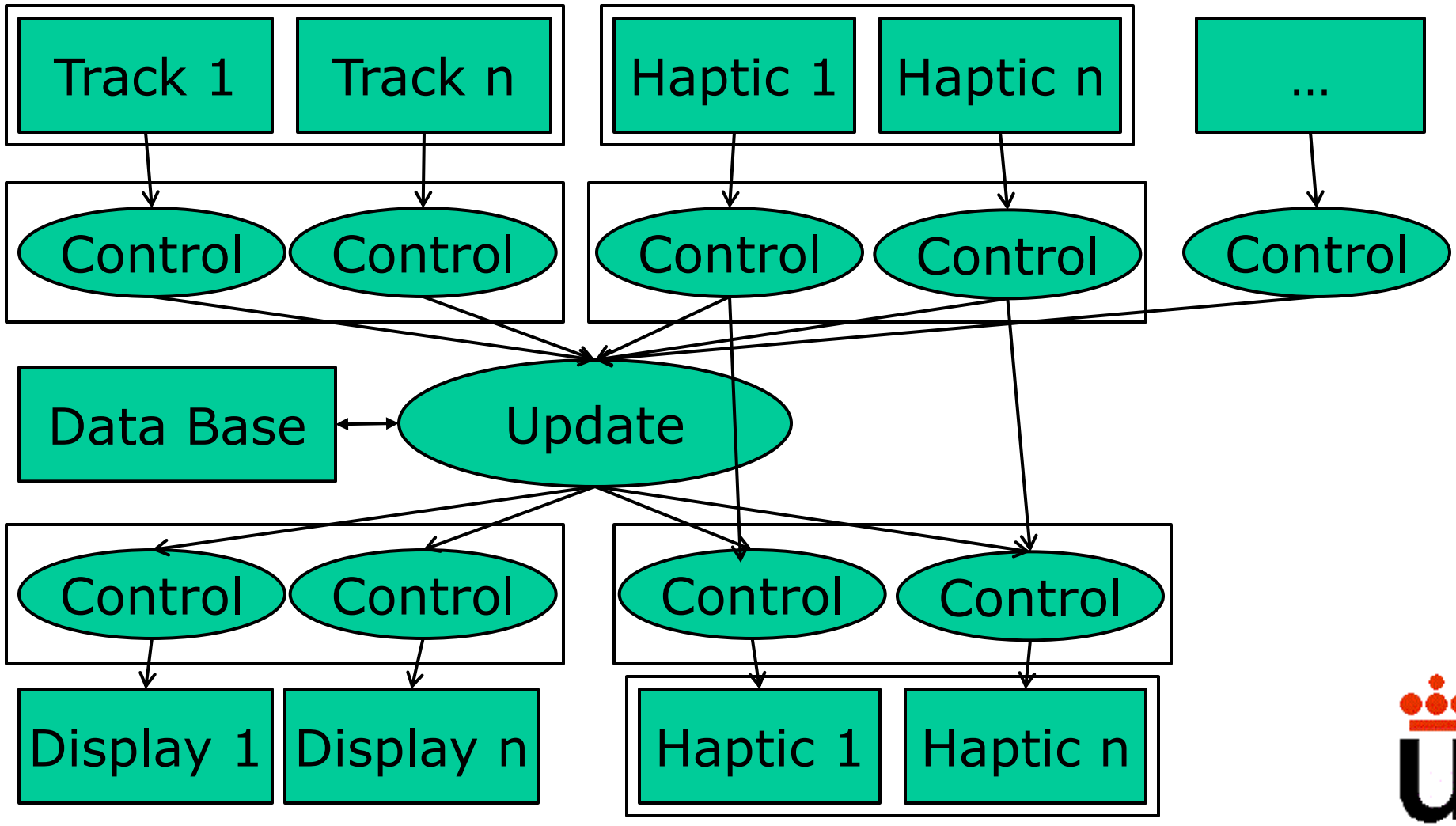
Ejemplo


CPU 1....CPU n
Processor Bus
Direct
Memory Bus
RAM
Memory
Memory
controller
AGP 4x Bus
Graphics
Memory
Graphics
Processor
PCI Bus
Hard Drive
I/O
controller
PCI Slots
Display
USB Port
Serial Port
3–D Tracker
3–D Sound
LAN card
Transmitter
HMD
Joystick

Motor de RV

- Componente clave
- Tareas:
  - Lectura de los dispositivos de entrada
  - Lectura de la base de datos
  - Actualización del estado (base de datos)
  - Escritura en los dispositivos de salida

Motor de RV


Track 1
Track n
Haptic 1
Haptic n
...
Control
Control
Control
Control
Control
Data Base
Update
Control
Control
Control
Control
Display 1
Display n
Haptic 1
Haptic n

Motor de RV: Tiempo Real

- Carga computacional elevadas:
  - Gran número de sistemas complejos
  - Imposible predecir las acciones de los usuarios
  - Requisitos impuestos por los factores humanos
    - Refresco visual 25-30Hz
    - Refresco táctil 500-1000Hz

Motor de Gráficos: Tiempo Real

- Carga computacional elevadas:
  - Latencia: tiempo transcurrido desde que el usuario efectúa una acción hasta que recibe una respuesta

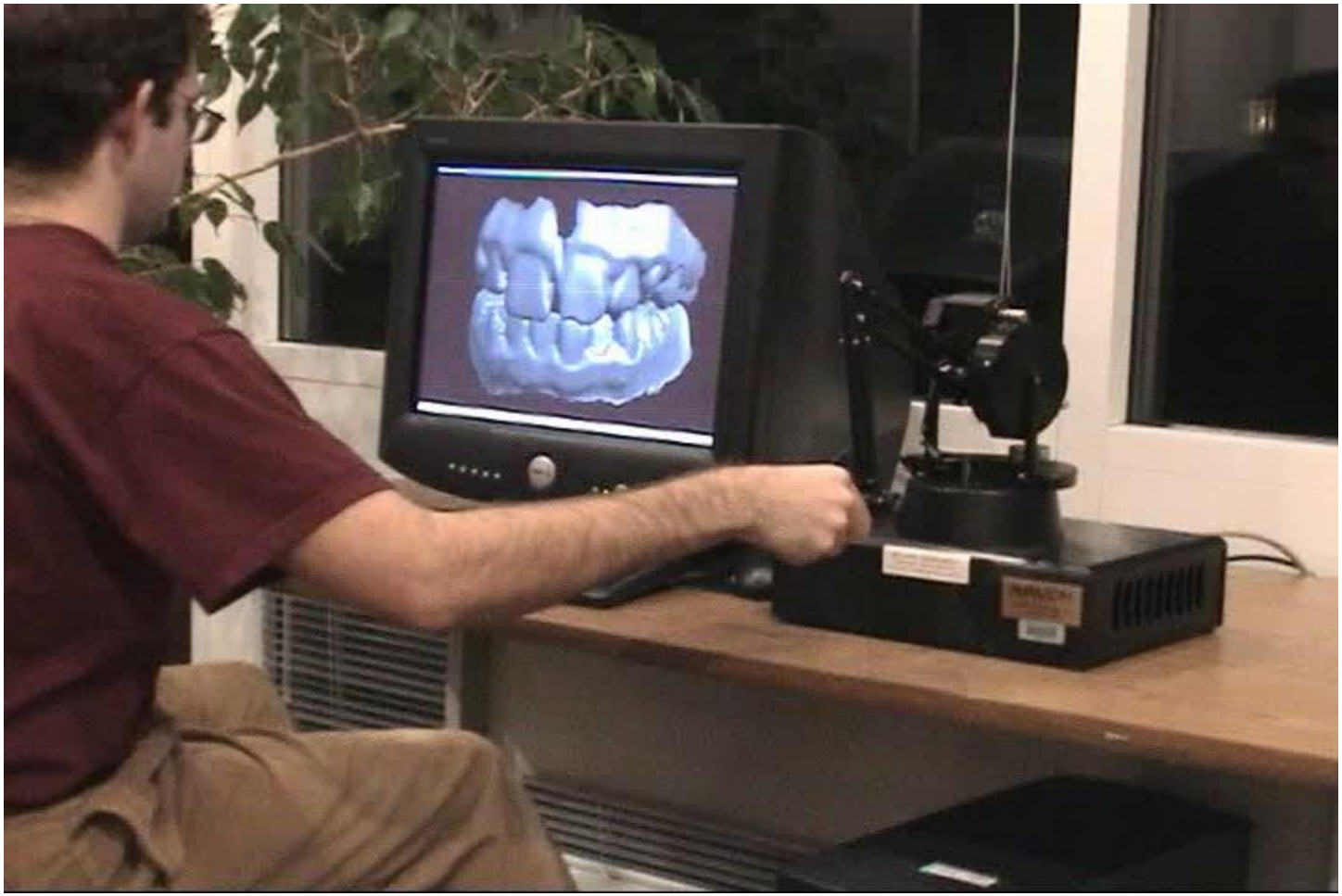
    Latencia total = latencia del sensor
    + retraso del bus
    + cálculo del nuevo estado
    + retraso del bus
    + latencia del dispositivo de salida
  - La latencia visual no debe exceder 100ms

Pipeline Háptico

1. Detección de colisiones
2. Cálculo de fuerzas
   - Depende de las propiedades de cada objeto
   - Suele utilizarse la ley de *Hooke* (computacionalmente ligero)
3. Suavizado de fuerza
   - Como *Phong shading*, para cálculo de fuerzas
4. Mapeo de la fuerza

- Texturas hápticas
  - Añaden: fricción, suavidad, temperatura
  - Utilización de bumps
- Control háptico: Bucle pesado Vs. Bucle ligero

Sistema Gráfico + Háptico



Arquitecturas

- Arquitecturas basadas en PC’s
- Arquitecturas basadas en estaciones de trabajo
- Arquitecturas basadas en sistemas distribuidos

Arquitecturas Basadas en PC's

- Es la opción más utilizada hoy
  - Espectacular incremento de prestaciones
    - A nivel de micros +3GHz
    - A nivel de buses
    - Nuevas tarjetas gráficas
  - Actualmente son sistemas multiusuario
  - Pueden controlar varios dispositivos salida
  - Precio

Bus

PCI (Intel, 1993)

AGP (Intel, 1997): 2.1 GB/s en el AGPx8

PCI Express 3.0:
16 GB/s con 32 canales
(SLI)

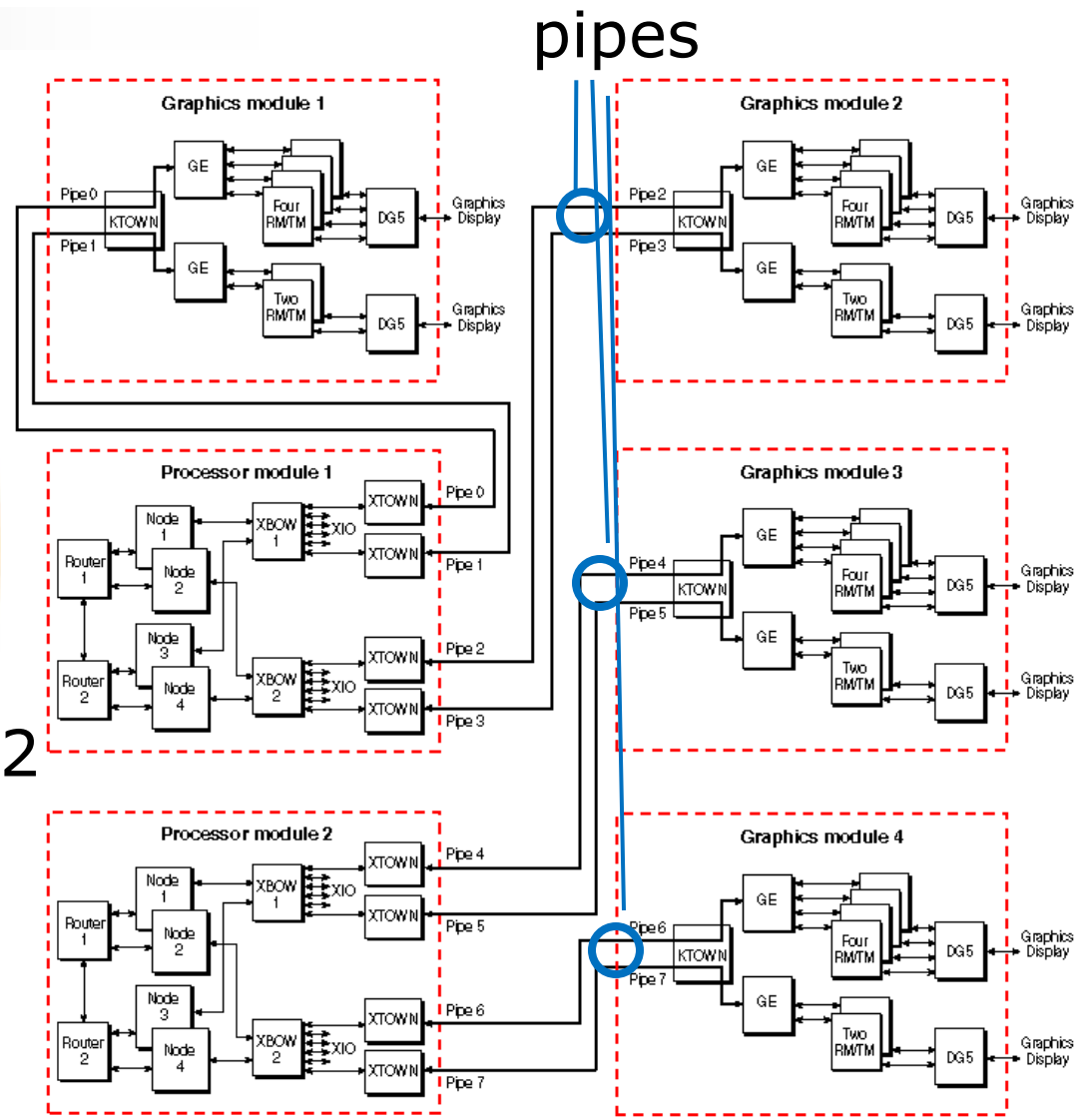
AGP: bus dedicado para gráficos, eliminó el cuello de botella.

Arquitecturas Basadas en Workstations

- Después de los PC's es la opción más utilizada
- Mayor capacidad de cómputo
  - Gracias a
    - Arquitecturas superescalables
    - Arquitecturas multiprocesador
    - Mayor velocidad en los buses
  - Tiempo real
- Orientados a la multitarea
  - Requerida por la mayoría de las aplicaciones VR
  - Por lo general usan sistemas operativos UNIX
- Mayor facilidad para soportar varios displays
  - Pueden soportar varias tarjetas gráficas

Un poco de historia

Onyx InfiniteReality2



Arquitecturas Distribuidas

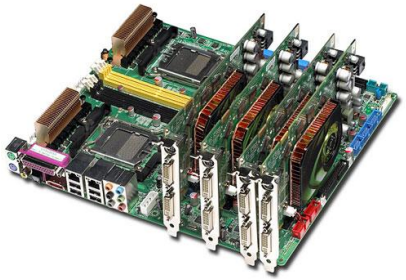
- Sistemas monousuario
  - Control de varios dispositivos
    - Sistemas con varios pipelines
      - Hápticos
      - Render
- Sistemas multiusuario
  - Entornos de cooperación

Sistemas con Varios Pipelines Gráficos

- Sistemas que necesitan varios displays
  - Mosaicos de pantallas (tiled displays)
  - Cuevas
  - Varios proyectores, utilizando una misma pantalla
    - Ejemplo: sistemas estéreo pasivo
  - ...

Sistemas con Varios Pipelines Gráficos

a) Tarjetas multipipe
- Barato
- No son escalables
- Comparten el bus, memoria y CPU



b) RACs con varias tarjetas gráficas
- Necesitas equipos con más de un bus
- Son sistemas caros (workstations)
- Uso de PC's y tarjetas sobre slots PCI.

c) Un mismo pipe para varios dispositivos
- Pérdida de resolución

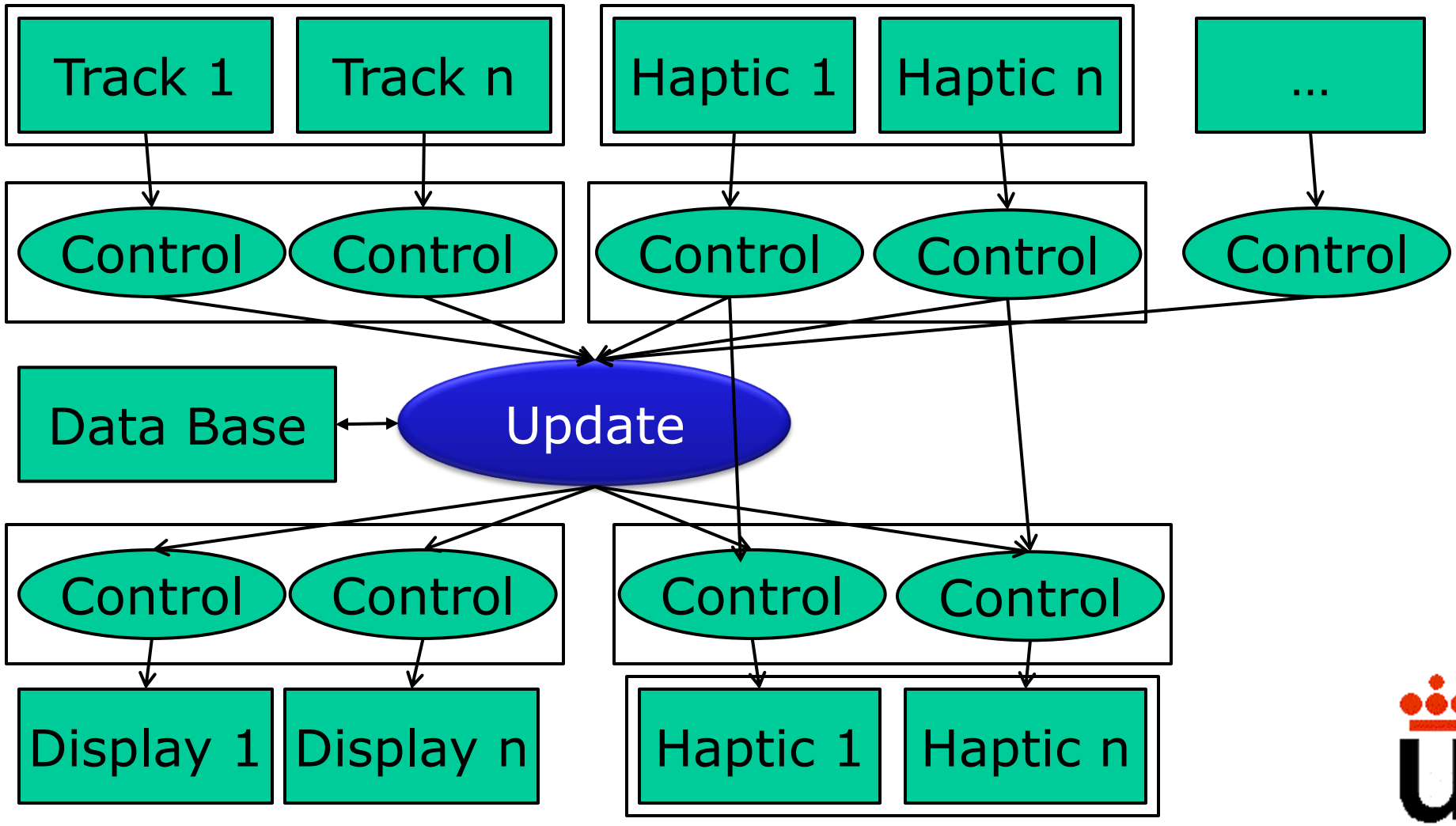
d) Cluster de PC's, con su propia tarjeta gráfica
- Conectados por redes de alta velocidad
  - Es el cuello de botella
  - Topologías y protocolos de red optimizados (WireGl)

WireGl - Chromium

WireGl - Chromium

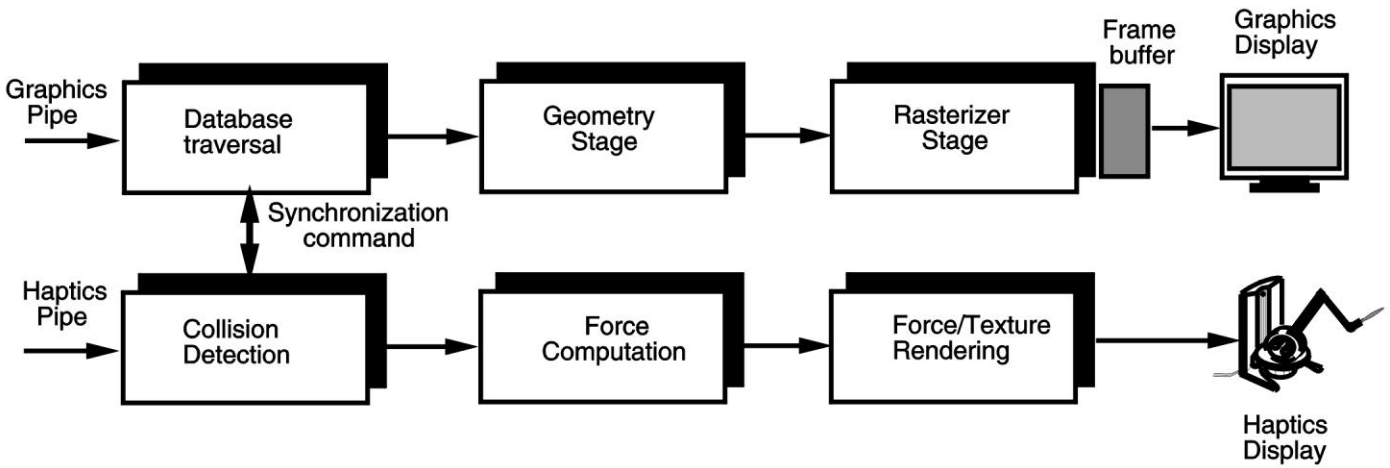
- Protocolo basado en TCP/IP o en protocolos Myrinet
- Se basa en distintos servidores de control cada uno de los cuales supervisa varios servidores de render
- El servidor de control captura los comandos OpenGl y los distribuye entre los distintos servidores de render
- Codifica las instrucciones OpenGl compactando y empaquetando datos y operaciones
- El servidor de control lleva un control del estado de cada nodo de render
  - Solo envía la información que ha cambiado

Sincronización de dispositivos heterogeneos



Sincronización de Dispositivos Heterogéneos

- La sincronización se realiza a nivel software
- Necesidad de desacoplado
  - Distintas frecuencias de trabajo



Sincronización de Pipelines Hápticos

- La sincronización se realiza a nivel de aplicación
  a) Después de la detección de colisiones
     - Más carga en el dispositivo dedicado
     - Más carga en la red
  b) Después del calculo de fuerzas
     - Sólo se trasmite el vector de fuerzas
     - Menos carga en el dispositivo dedicado

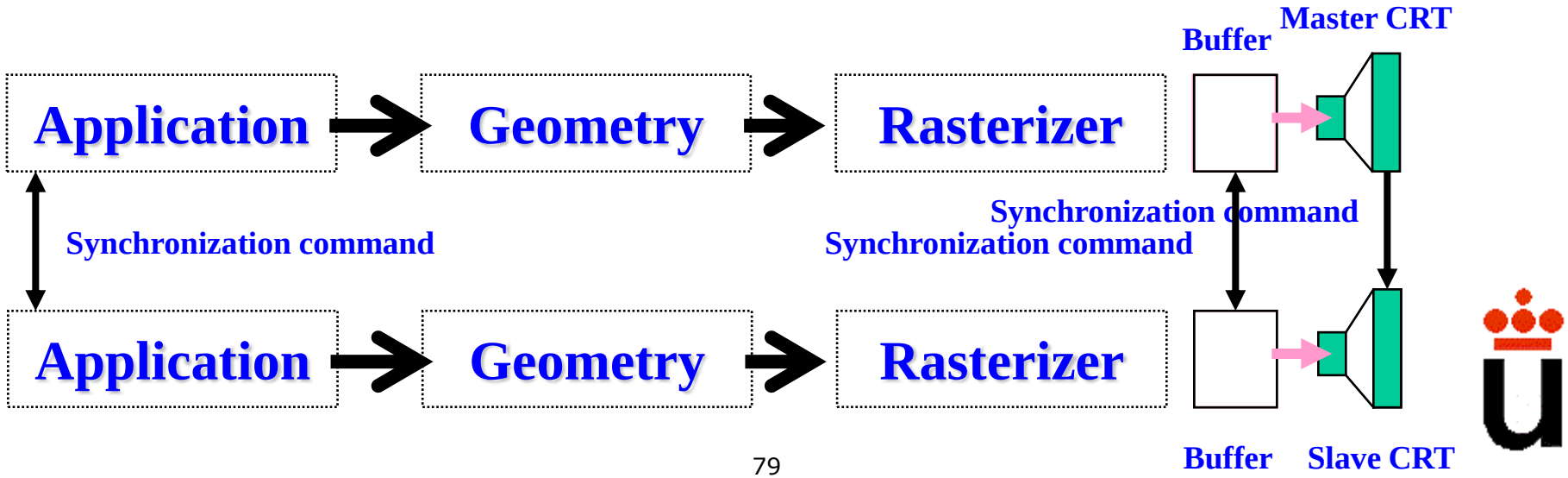
- Tareas del dispositivo dedicado
  - Suavizado
  - Mapeado
  - Texturado

Sincronización de Pipelines Gráficos

- Crítica en mosaicos de displays
  - Especialmente cuando se usan varios monitores CRT
    - Requieren sincronización píxel a píxel
- En el caso de utilizar imágenes estéreo la sincronización es todavía más crítica
  - Se requiere que todos los dispositivos muestren a la vez el mismo campo de la imagen (izquierdo o derecho)
- La falta de sincronización produce parpadeo y malestar en el usuario

Sincronización de Pipelines Gráficos

- La sincronización puede darse a 3 niveles
  - Nivel de aplicación
  - Nivel de cambio de buffers
  - Nivel de vídeo



Sincronización a Nivel de Aplicación

- Se comienza a procesar un nuevo frame a la vez en todos los pipeline

- Puede que haya pipelines que terminen antes que otros
  - Cuando se llena un framebuffer, se muestra; no se espera

Sincronización a Nivel de Cambio de Buffers

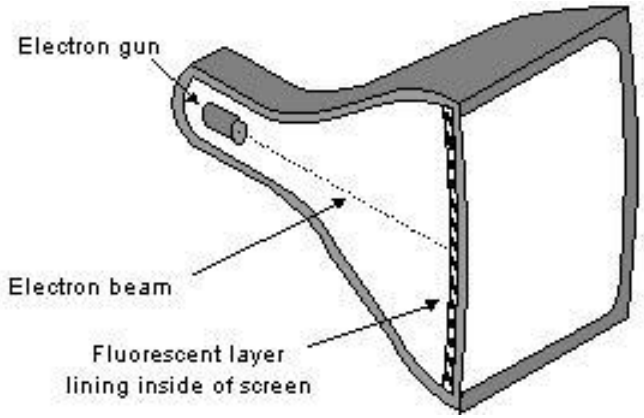
- Cuando termina un pipeline, espera a que termine el resto

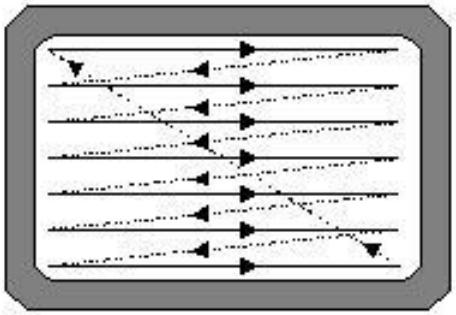
- Cuando todos han terminado se intercambian los buffers
  (escritura ←→ visualizado)
  - glutSwapBuffer() (GLUT)

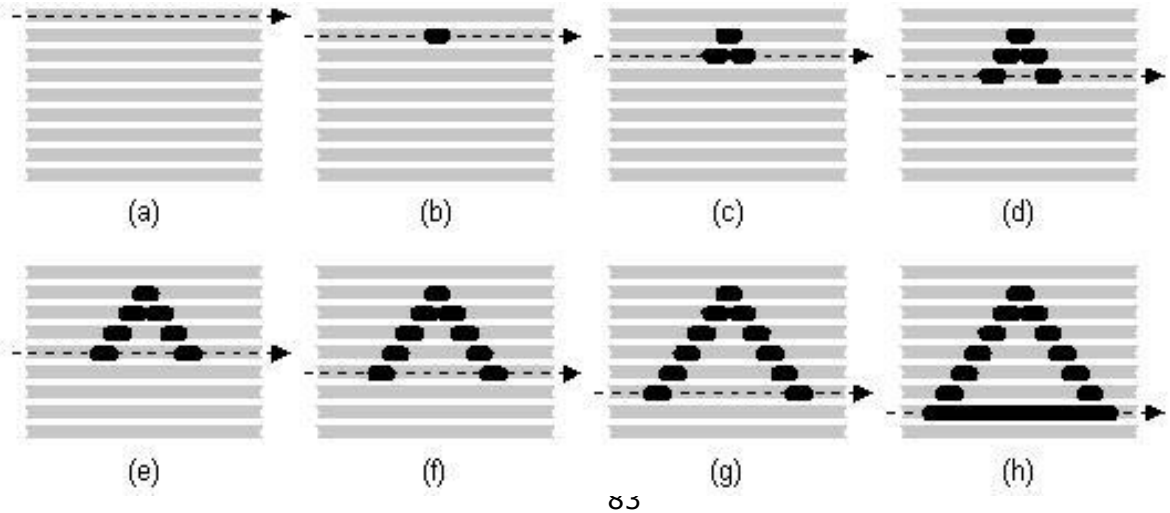
Sincronización a Nivel de Cambio de Buffers

- Problema: cada display comienza a pintar en momentos distintos, dependiendo de su ciclo y de su frecuencia de refresco
  - Los monitores CRT empiezan a pintar cuando su disparador retorna de la esquina inferior derecha a la esquina superior izquierda

Monitores CRT







Sincronización a Nivel de Vídeo

- Sincronización a nivel Hardware
  - Interconexión entre tarjetas de vídeo
- Arquitectura maestro-esclavo
- Asegura que los dispositivos utilicen la misma frecuencia y ciclo de refresco

Genlock

- Señal externa generada por hardware para sincronizar distintos pipelines
- Conexión de dipositivos en cadena (Daisy-Chain)
- Realiza
  - La sincronización del intercambio de buffers
    - Cuando el maestro termina pregunta al esclavo si éste ha terminado también. Si el esclavo también ha terminado manda la señal de *done*, si no, no manda nada hasta que termina
  - La sincronización a nivel de vídeo
  - La sincronización de campos en imágenes estéreo
  - La sincronización con otros dispositivos (como trackers)
    - Reducción de interferencias

Índice

- Gráficos Multi-Proceso / Multi-Pantalla
- Arquitectura de la GPU
- Gráficos Distribuidos

Sistemas VR Distribuidos Multiusuario

- Objetivo: generar entornos virtuales distribuidos
  - Varios usuarios separados físicamente comparten un mismo entorno virtual
- Todos los usuarios pueden modificar su entorno
  - Entornos de colaboración
    - Sólo un usuario puede manipular un objeto al mismo tiempo
  - Entornos cooperativos
    - Varios usuarios pueden manipular el mismo objeto a la vez
- A través de red LAN ó WAN

Topología de la Red

- Factores
  - Tipo de aplicación
    - Entornos de colaboración
    - Entornos cooperativos
  - Número máximo de usuarios
  - Otros
    - Latencia máxima admitida
    - Escalabilidad
    - Tolerancia a fallos
    - Tipo de interacción (visual, háptica, audio…)

Topología de la Red

- La clave: reducción del tráfico
  - Todos los equipos mantienen una copia de la escena
  - Sólo transmiten las modificaciones que hacen
  - Utilizan un buffer de transmisión para mantener la coherencia en caso de fallo de la red
- Tipos de redes
  - Conexión punto a punto vs. broadcast
  - Topologías con un único servidor
  - Topologías con varios servidores
  - Protocolos peer-to-peer (P2P)

Conexión Punto a Punto

- Sólo permiten interconectar dos equipos
- Direccionamiento unicast
- Funcionan bien sobre TCP/IP
  - Protocolo orientado a conexión

Topologías con un Único Servidor

- Direccionamiento unicast, sobre TCP/IP
- Los usuarios notifican al servidor cuando modifican la escena
- El servidor trasmite los cambios en la escena sólo a los nodos interesados

Topologías con un Único Servidor

- El número máximo de usuarios lo determinan la capacidad del servidor y la capacidad de la red
- Si la red es rápida, el servidor es un cuello de botella

Topologías con Varios Servidores

- Direccionamiento unicast, sobre TCP/IP
- Cada nodo lleva asociado un servidor
- Los usuarios notifican a su servidor cuando modifican la escena
- El servidor trasmite los cambios en la escena al resto de servidores y a los nodos asociados interesados

Topologías con Varios Servidores

- Estructura jerárquica de la red
- Mayor tráfico y mayor latencia, pero el servidor no es un cuello de botella
- El número máximo de usuarios lo determina la capacidad de la red

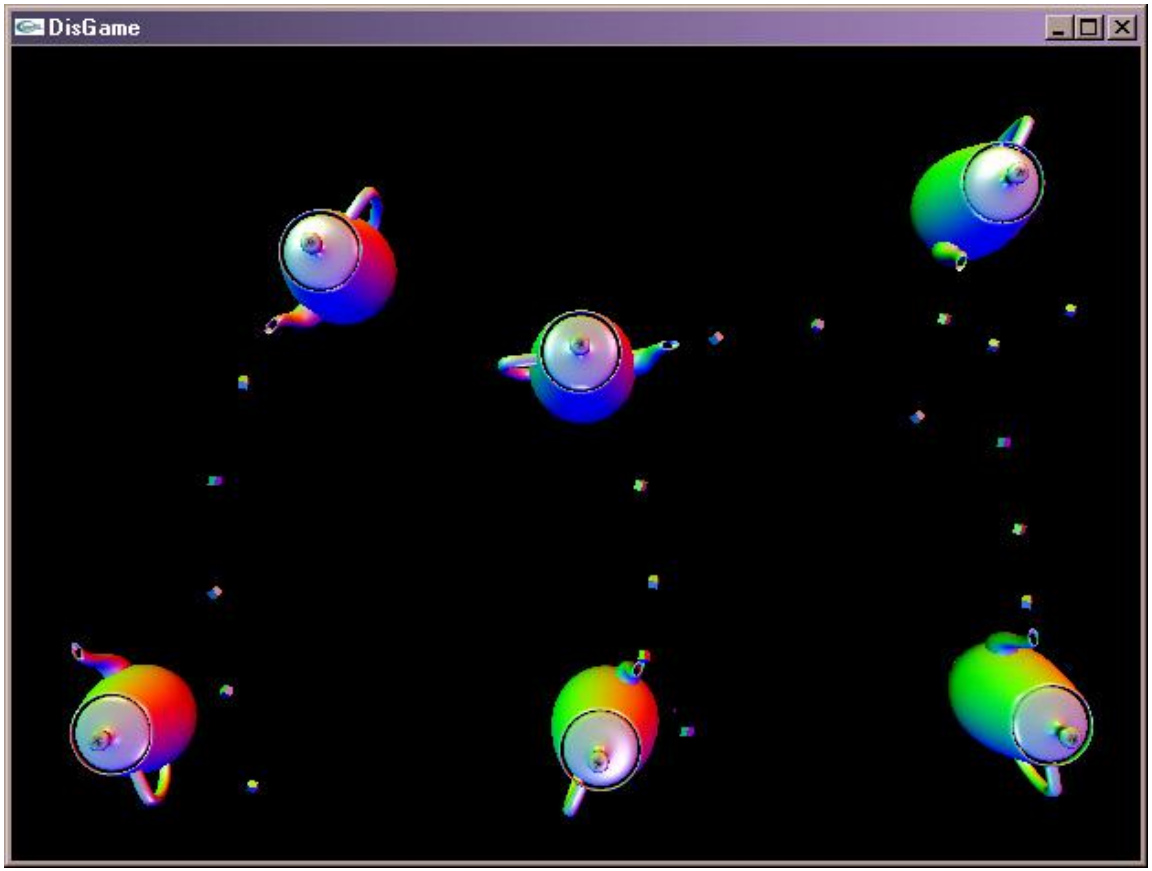
Protocolos Peer-to-Peer

- Direccionamiento multicast, sobre UDP/IP
- Los nodos notifican al resto cuando modifican la escena mediante mensajes multicast
- Un nodo procesa los mensajes sólo cuando le interesa

Protocolos Peer-to-Peer

- Si el sistema se queda sin usuarios se pierde el entorno
- Un nodo al conectarse descarga el entorno de otro nodo mediante TCP-IP

Ejemplo: Juego Distribuido



Dueling Teapots

http://www.scheib.net/school/243/index.html