**PARTE CUDA**

Taxonomía de Flynn

• **SISD:** Single Instruction, Single Data stream

Arquitectura x86 clásica,…

• **SIMD:** Single Instruction, MultipleData streams

Extensiones MMX, SSEX, 3DNow!, arquitecturas vectoriales,…

• **MISD:** MultipleInstruction, Single Data stream

Control de vuelo de los transbordadores (en general,

sistemas con alta tolerancia a fallos)

• **MIMD:** MultipleInstruction, MultipleData streams

Arquitecturas distribuidas, Multi-core,…

Configurar Kernel

\_\_global\_\_ **void** KernelFunc(...);

dim3 DimGrid(100, 50); // 5000 bloques de hilos

dim3 DimBlock(4, 8, 8); // 256 hilos por bloque

size\_t SharedMemBytes = 64; // 64 bytes de memoria comp.

KernelFunc<<< DimGrid, DimBlock, SharedMemBytes >>>(...);

Clipping y Culling

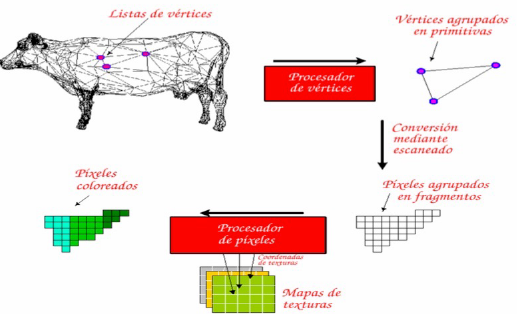
clipping: se define la parte que vamos a visualizar

culling: descarte de las partes que no se visualizan

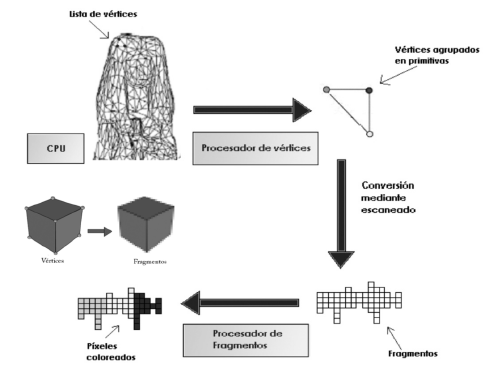
Renderizado

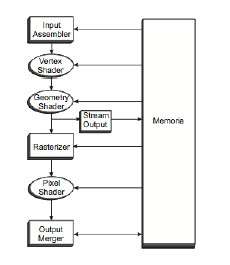
Proceso mediante el cual se procesa una imagen en 3D para conseguir un resultado en pixeles para que se pueda ver perfectamente en un ordenador.

Proceso de renderización:

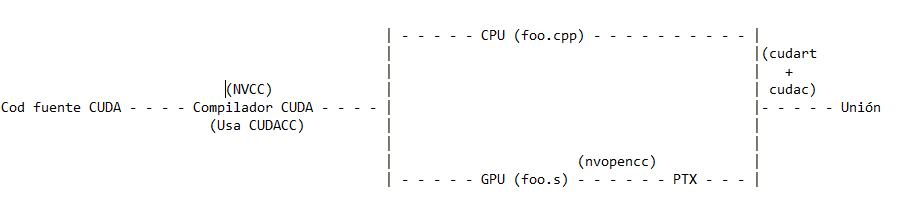


Cauce gráfico



Compilador CUDA



**PARTE SCALA**

Case class

Las clases Case son clases regulares las cuales exportan sus parámetros constructores y a su vez proveen una descomposición recursiva de sí mismas a través de [reconocimiento de patrones](http://docs.scala-lang.org/es/tutorials/tour/pattern-matching.html).

Patter maching

Hacer el matching de una variable con un patrón en un case. Se utiliza una estructura que permite no tener que repetir la variable constantemente

Estas dos construcciones son particularmente útiles para los datos recursivos tipo árbol

Traits

• Scala proporciona un ***trait*** llamado

**scala.Application**. Este ***trait*** tiene declarado el

método **main**, por lo que si lo extiende un objeto

**singleton**, se puede usar como una aplicación de

Scala directamente

• Un ***trait*** encapsula definiciones de métodos y campos,

que se pueden reutilizar mezclándolos dentro de

clases.

• A diferencia de la herencia, donde una clase sólo

puede heredar de una superclase, una clase puede

mezclar cualquier número de traits.

• Se podría considerar que un trait es como una interfaz

en Java con métodos concretos. Pero además los ***traits***

pueden declarar campos y mantener estado. Un ***trait***

define un nuevo tipo. Se consideran como **interfaces**

**de Java enriquecidas**.

Mixins

cuando un trait se mezcla con una clase, se le puede llamar ***mixin***

El orden de los ***mixins*** es significante.

**Van de derecha a izquierda**. Cuando se llama a un

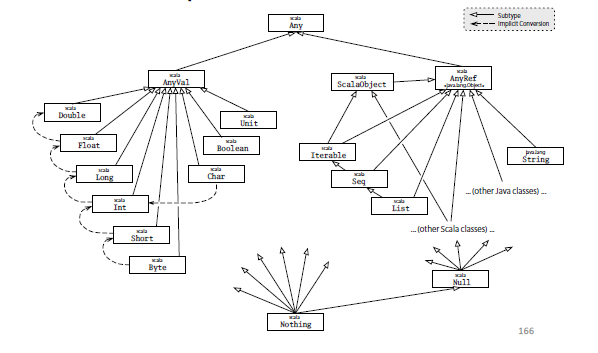
método de una clase con ***mixins***, se llama primero al

método del ***trait*** más a la derecha. Si este método a

su vez se llama **super**, invocará al método del

siguiente ***trait*** a su izquierda.

Jerarquía



Objeto Singleton

• Por definición corresponden con instancias

únicas de una clase:

– Si **a** es una instancia de la clase **A** y

– **b** es una instancia de la clase A

– Se cumple que **a** y **b** son el mismo objeto (ocupan

la misma región de memoria)

• Se utilizan en Scala para reemplazar los

miembros de clase estáticos que existen en

Java.

Clausura

funciones definidas en un ámbito mantienen el

acceso a ese ámbito y pueden usar los valores allí

definidos.

Función anónima

(Función sin nombre y que puede llamarse desde cualquier lado\*)

**PARTE APPENGINE**

Características Google App Engine

Gratuito durante periodo de prueba

Existen escasas aplicaciones comerciales en app-engine

No se pueden descargar o ejecutar scripts

Apps en Java o Python

Para ejecutar el código se necesita una petición HTTP

Tipos de nubes de una empresa

Pago: seguridad y privacidad de datos, lock-in del vendedor, altos requerimientos computacionales, reducción de costes al compartir con otro proyecto de la empresa

Gratuita: Menos seguridad y requerimiento computacionales, pero sin coste

Mixta: Mezcla de las dos anteriores