Shaders

<a.soroa@ehu.eus>

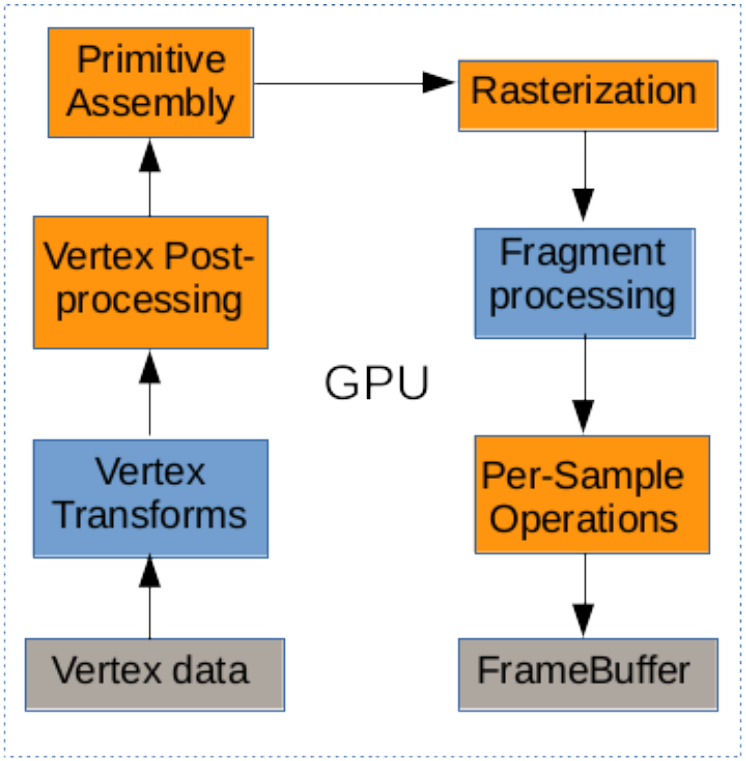
EHU

Shader

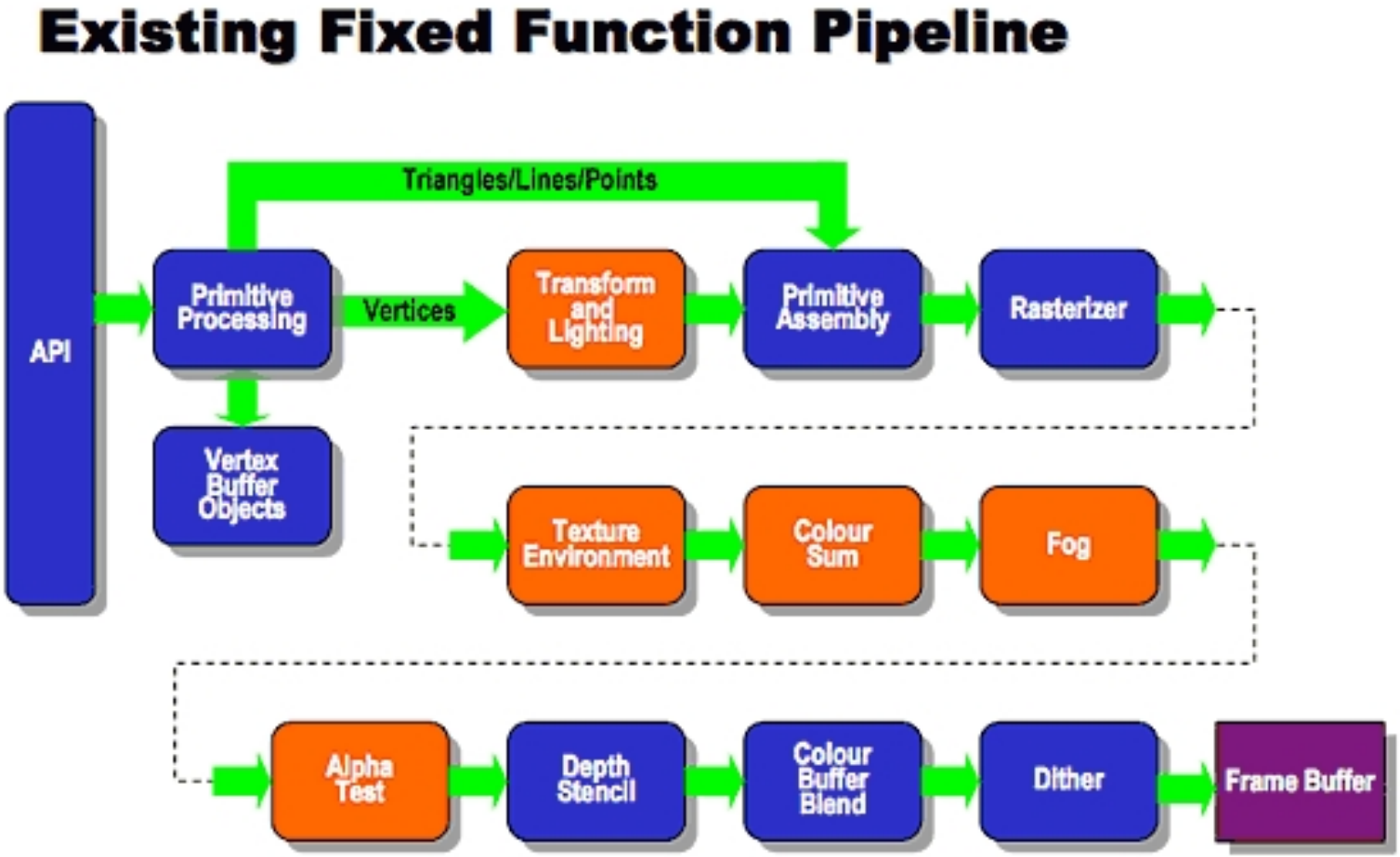
- Pequeños programas que se ejecutan directamente en la GPU.
- Situados en etapas precisas del *pipeline*.
- El procesamiento pesado a cargo de la GPU.
- Amplio abanico de posibilidades
  - una aplicación puede tener muchos *shader*s.

Del *pipeline* fijo al programable

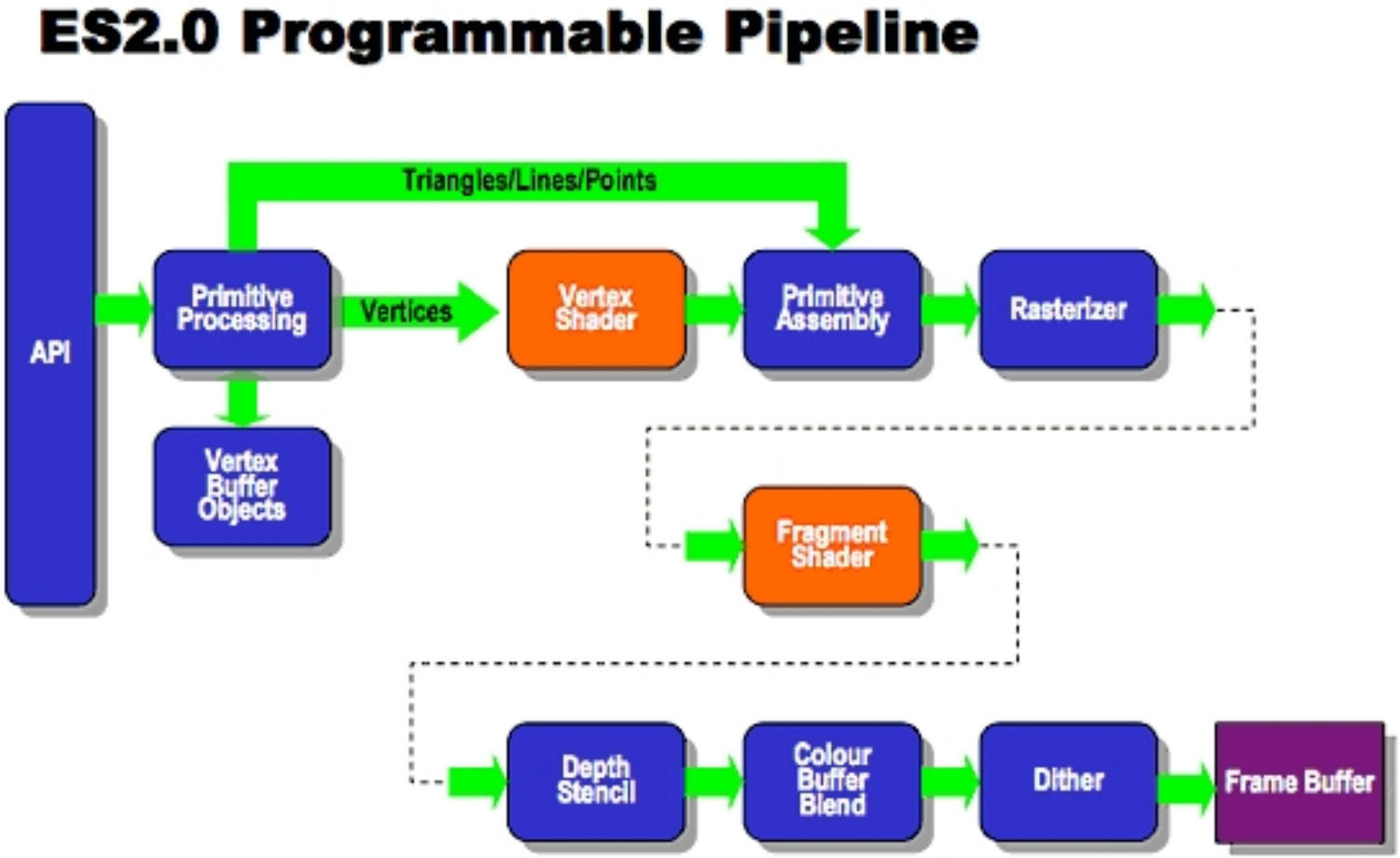
- Cambiar el *pipeline* fijo e insertar módulos programables.
  - *Vertex shader*: reemplaza la etapa *Vertex Transforms*
  - *Fragment shader*: reemplaza la etapa *Fragment processing*



Del *pipeline* fijo al programable



Del *pipeline* fijo al programable



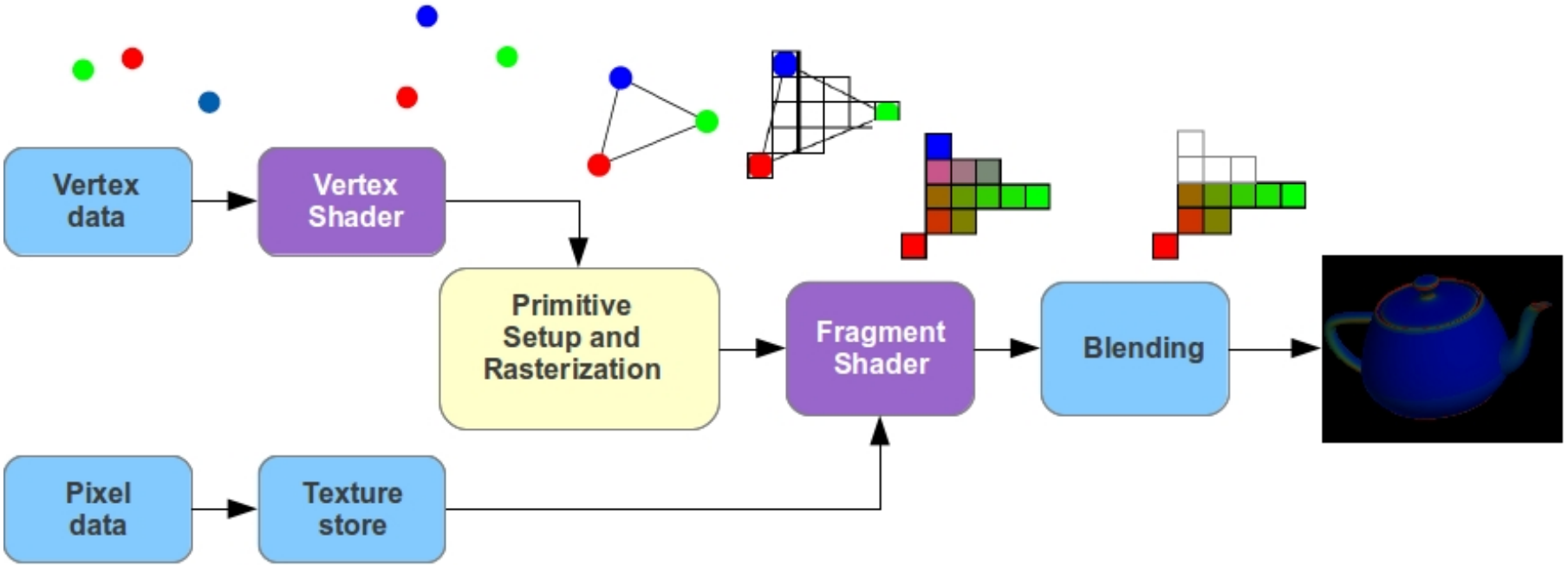
Shader

- Escritos en lenguaje GLSL (OpenGL).
  - Cg (tarjetas Nvidia, Playstation)
  - HLSL (DirectX, Xbox)
- Paralelización
  - *Vertex shader*: una ejecución vez por vértice.
  - *Fragment shader*: una ejecución por píxel.

Paralelización

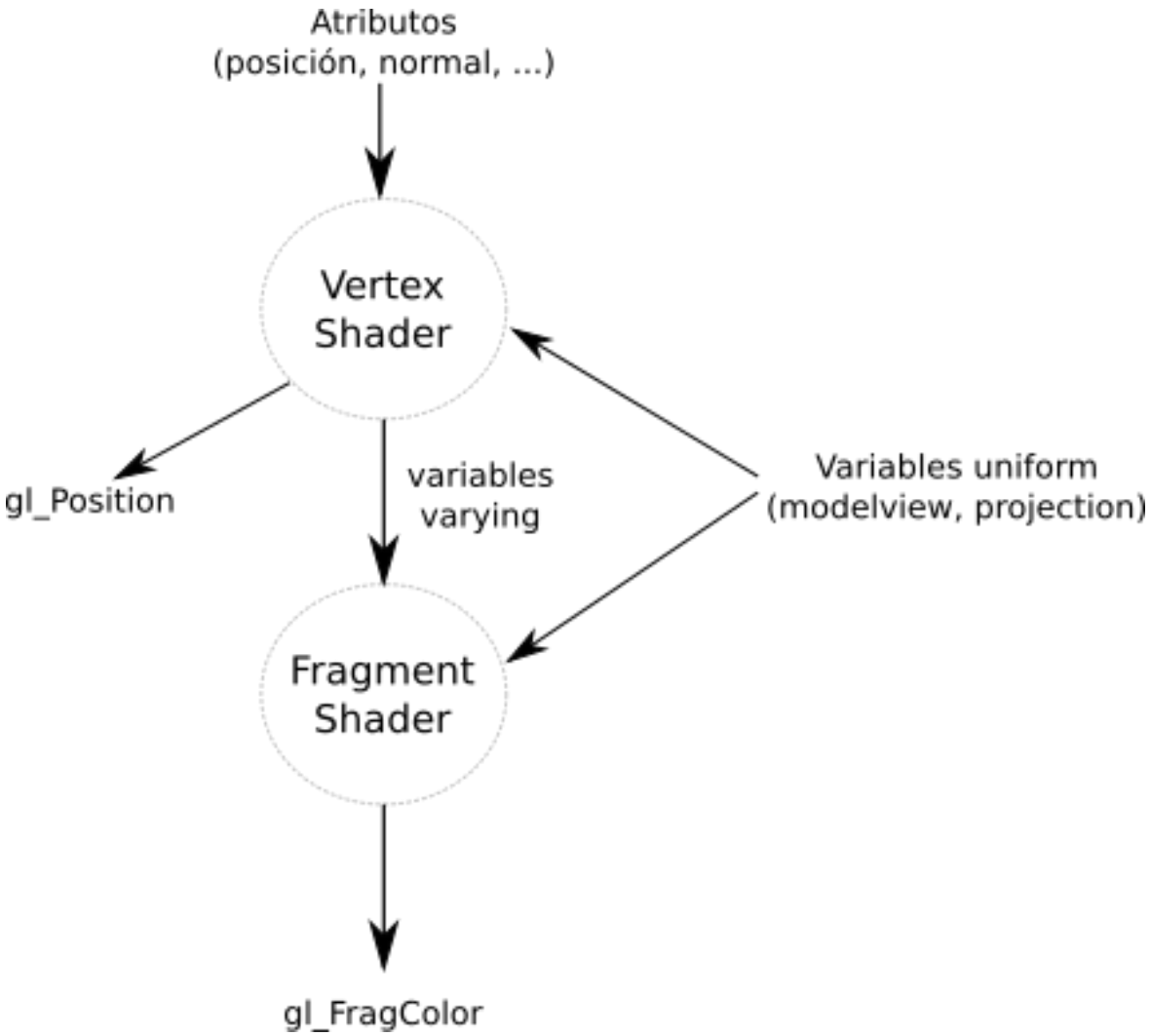
- Un *shader*, en cualquier etapa, opera completamente independiente de cualquier otro shader de esta etapa
  - Un *shader* toma un conjunto de entradas, las procesa, y produce un conjunto de salidas
  - No existen correlaciones entre las ejecuciones separadas de un *shader*

Esquema del *pipeline* programable



Aplicación OpenGL 3.1

Flujo de datos



Parámetros

- Los *shaders* necesitan información de la aplicación:
  - atributos de los vértices (`attribute`)
  - variables de tipo general (`uniform`)

Atributos

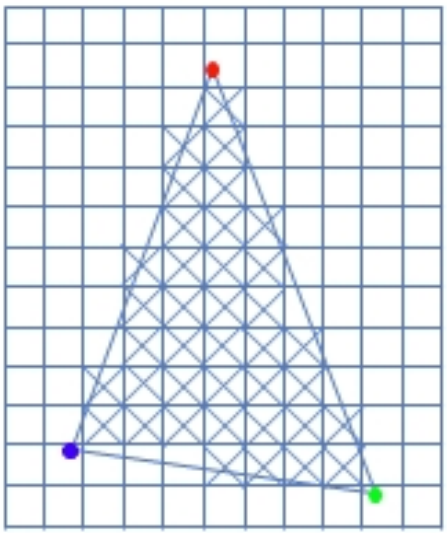
- Variables sólo lectura asociadas a los vértices.
- Sólo accesibles por el *vertex shader*.
- Definidos en la geometría del objeto:
  - posición
  - normal
  - coordenada de textura
  - ...

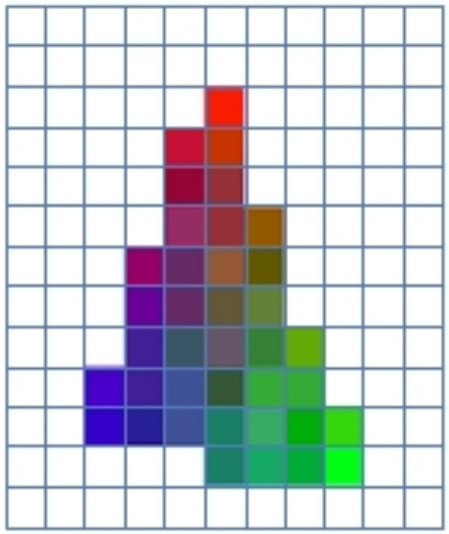
Uniform

- Variables sólo lectura enviados por la aplicación.
- Accesibles por el *vertex shader* y el *fragment shader*.
- Ejemplos típicos:
  - matriz *modelview*
  - matriz de proyección
  - información del material (color, ...)
  - información de luces (posición, color, ...)
  - . . .

Varying

- Variables que el *vertex shader* envía al *fragment shader*
  - la salido sel *vertex shader* es por cada vértice
  - la entrada del *fragment shader* es por cada fragmento
- Se produce una interpolación (*varying*)





Vertex shader. Operaciones.

- Recibe información de los vértices.
- Operaciones comunes:
  - Transformar la posición del vértice.
  - Transformar la normal del vértice.
  - Calcular la iluminación del vértice.
  - Crear y transformar las coordenadas de texturas.
  - . . .

Operaciones

- Las siguientes operaciones **no** pueden ser ejecutadas por el *vertex shader*
  - *clipping*
  - División de perspectiva.
  - Borrado de caras anteriores (*back face culling*)

Salida

- Debe escribir la variable `gl_Position`
  - posición del vértice en *clipping coordinates*
- Además, usualmente escribe varias variables `varying`

Fragment shader: operaciones

- Operaciones
  - Cálculo del color:
    - Para ello, utiliza información de las variables interpoladas (*varying*)
    - quizá también de las variables *uniform*.
  - Aplicación de texturas.
  - Efectos:
    - Procesamiento de imágenes: *blurring*, *glow*, etc.
    - Niebla.
    - ...
  - Descartar fragmentos.

Operaciones

- Las siguientes operaciones **no** pueden ser ejecutadas por el *fragment shader*:
  - Mezclado (*blending*).
  - *Stencil test*.
  - *Z-buffer test*.
  - *Stippling*.

Salida

- Debe escribir la variable `gl_FragColor`
  - color RGBA del fragmento.
- Además, puede escribir la variable `gl_FragDepth`
  - profundidad del fragmento (`float`).
  - si no lo hace, se asigna la profundidad por defecto (etapa *Rasterizer*).