Pipeline

<a.soroa@ehu.eus>

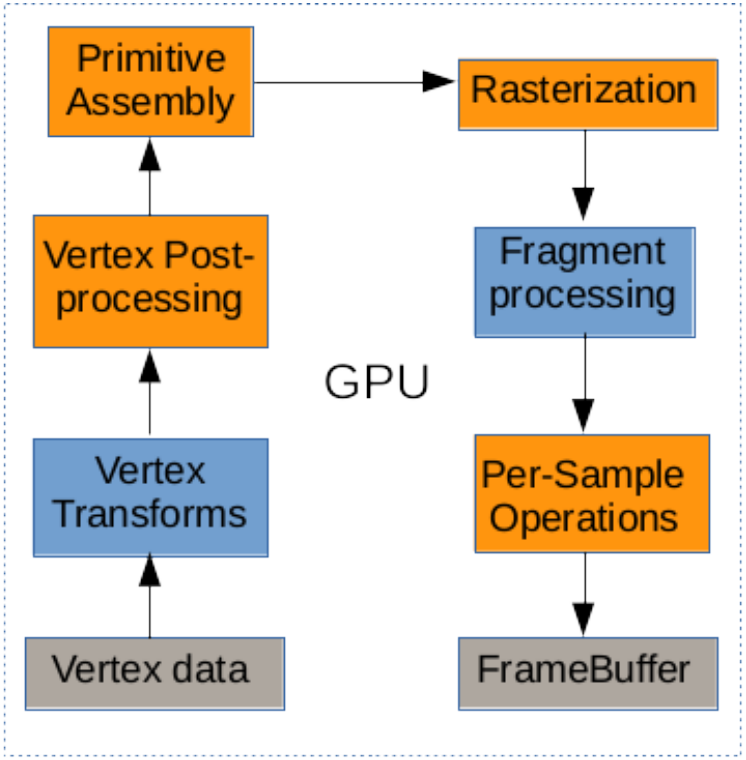
EHU

pipeline

- Llamamos *pipeline* a las etapas que se ejecutan dentro de la GPU.
- Tradicionalmente, el *pipeline* ha estado prefijado (*fixed*)
- Con la introducción de los *shaders*, el *pipeline* es ahora programable.
- Vamos a ver los pasos del *pipeline* fijo.

Fixed-function pipeline

- El *pipeline* estándar hasta hace relativamente poco (OpenGL 2.0)
- Funciones prefijadas para las transformaciones y cálculos de color.
  - El flujo de datos es siempre el mismo.
  - Implementado en el *hardware* (GPU).



Etapa *Vertex transformations*

- Esta etapa necesita los atributos de vértices: posición, colores, normales, coordenadas de texturas, ...
- En esta etapa ocurren las transformaciones de vértices:
  - Transformación de las posiciones.
  - Sombreado de los vértices.
  - Creación y/o transformación de coordenadas de texturas
- La salida de esta etapa consiste en vértices transformados:
- La posición del vértice en el llamado *clipping space*
  - Después de la proyección, antes de dividir por la coordenada w'.

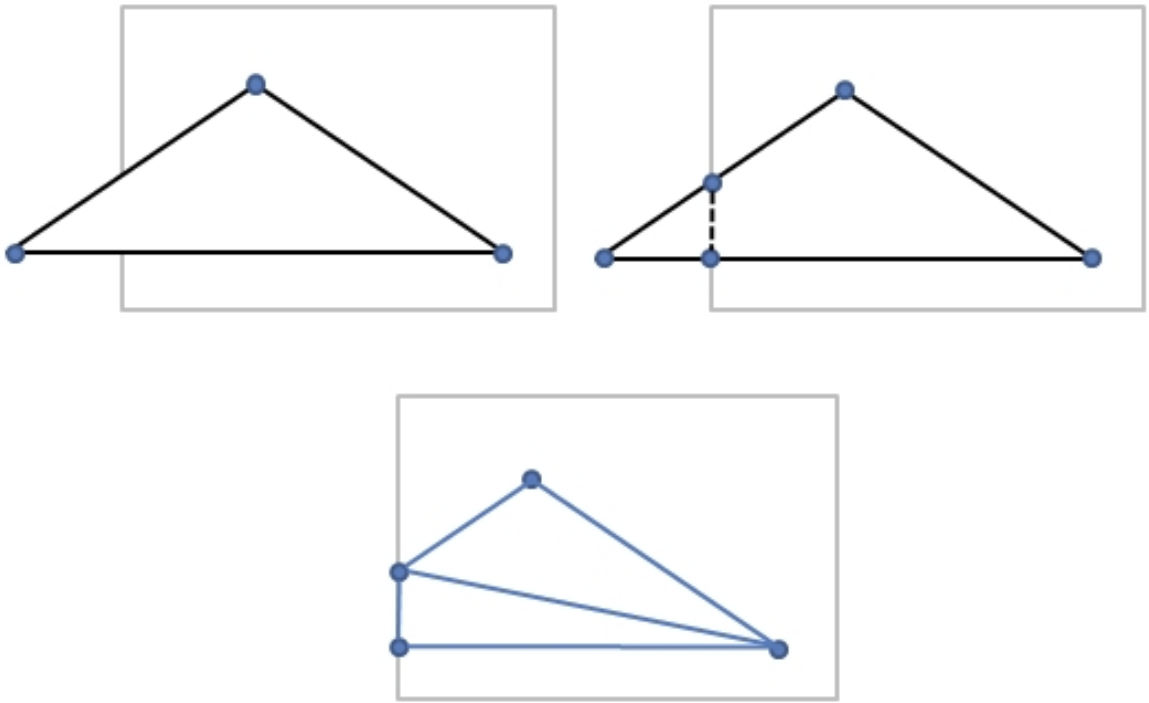
$$\begin{pmatrix} x' \\ y' \\ z' \\ w' \end{pmatrix} = \mathbf{M}_{\text{projection}} \mathbf{M}_{\text{modelview}} \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix}$$

Etapa *Vertex Post-processing*

- Post-proceso de vértices
- Entrada:
  - vértices definidos en el *clipping space*
- Procesos:
  - *Clipping*: decidir qué vértices están fuera de la pantalla.
  - *Perspective divide*: se divide los vértices por la coordenada w
  - *Viewport transformation*: se obtienen las coordenadas de la pantalla.

Vertex Post-processing

- *Clipping*:



Vertex Post-processing

- *Perspective divide*: se divide la coordenada por la componente w (*Normalized device coordinates*)

$$\begin{pmatrix} x_{\text{ndc}} \\ y_{\text{ndc}} \\ z_{\text{ndc}} \end{pmatrix} = \begin{pmatrix} \frac{x}{w} \\ \frac{y}{w} \\ \frac{z}{w} \end{pmatrix}$$

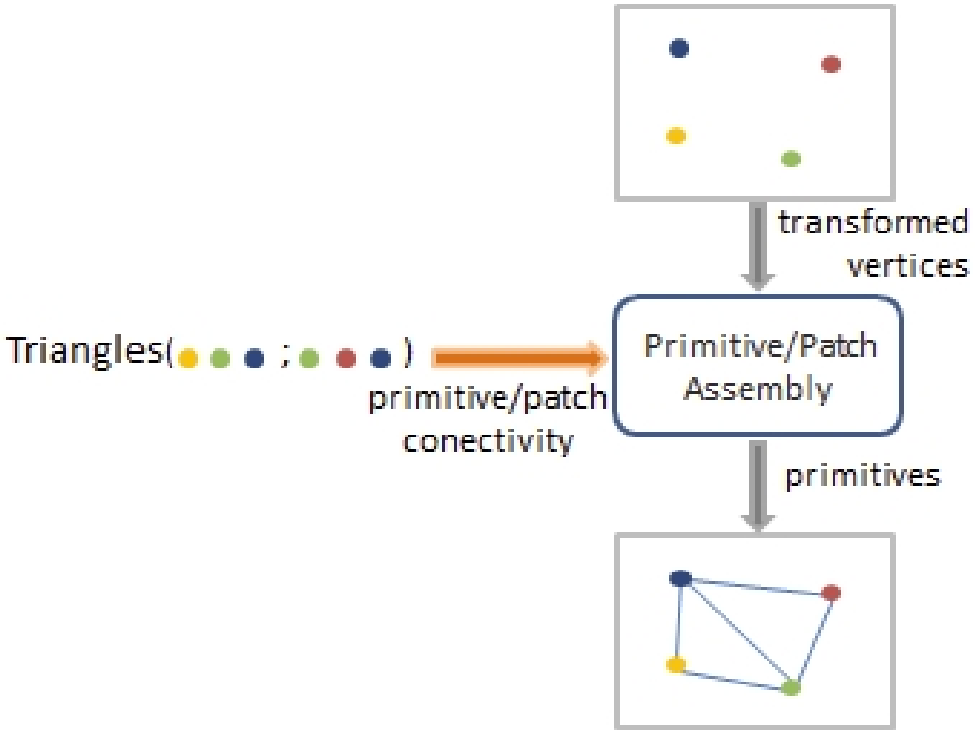
- *Viewport transformation*: se obtienen las coordenadas de la pantalla
  - teniendo en cuenta la anchura/altura de la ventana.
- Como resultado se obtienen:
  - las coordenadas del vértice en la pantalla (*Screen coordinates*).
  - la profundidad del vértice (z)

Etapa *Primitive Assembly*

- La entrada de esta etapa es doble:
  - Vértice transformado (*Screen coordinates*).
  - Conectividad de los vértices: Qué vértices componen un triángulo.
- Operaciones:
  - Crear las aristas de los triángulos
  - Borrar las caras traseras (*backface culling*).

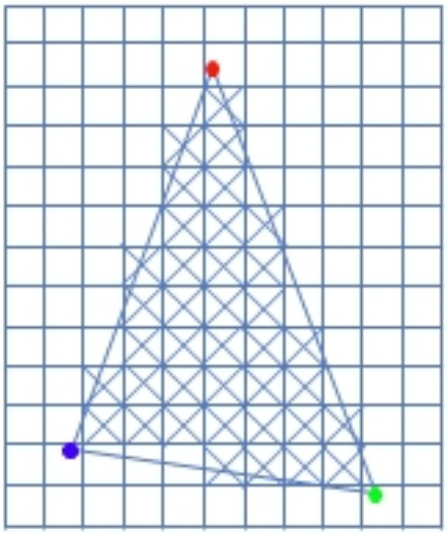
Primitive Assembly

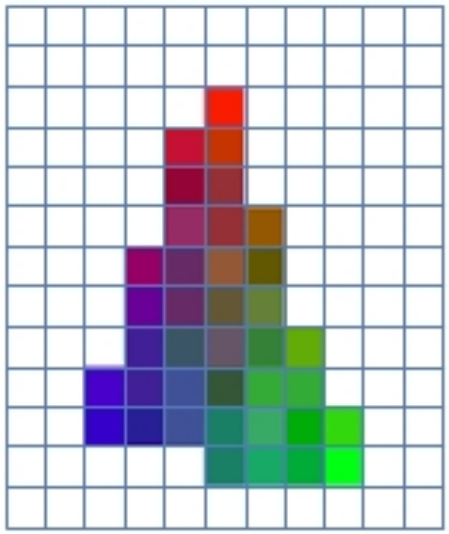
- Crear las aristas de los triángulos



Etapa *Rasterization*

- Discretización de triángulos
  - algoritmo *scan conversion*
- Entrada:
  - primitivas (triángulos) en coordenadas de pantalla
- Salida: Conjunto de fragmentos (*fragment*)
  - posición (incluyendo profundidad)
  - atributos interpolados





Etapa *Fragment Processing*

- Procesa cada fragmento generado por el *Rasterizer*
- Entrada: conjunto de fragmentos
  - incluyendo información asociada (interpolada)
- Salida:
  - Color del fragmento
  - Profundidad del fragmento

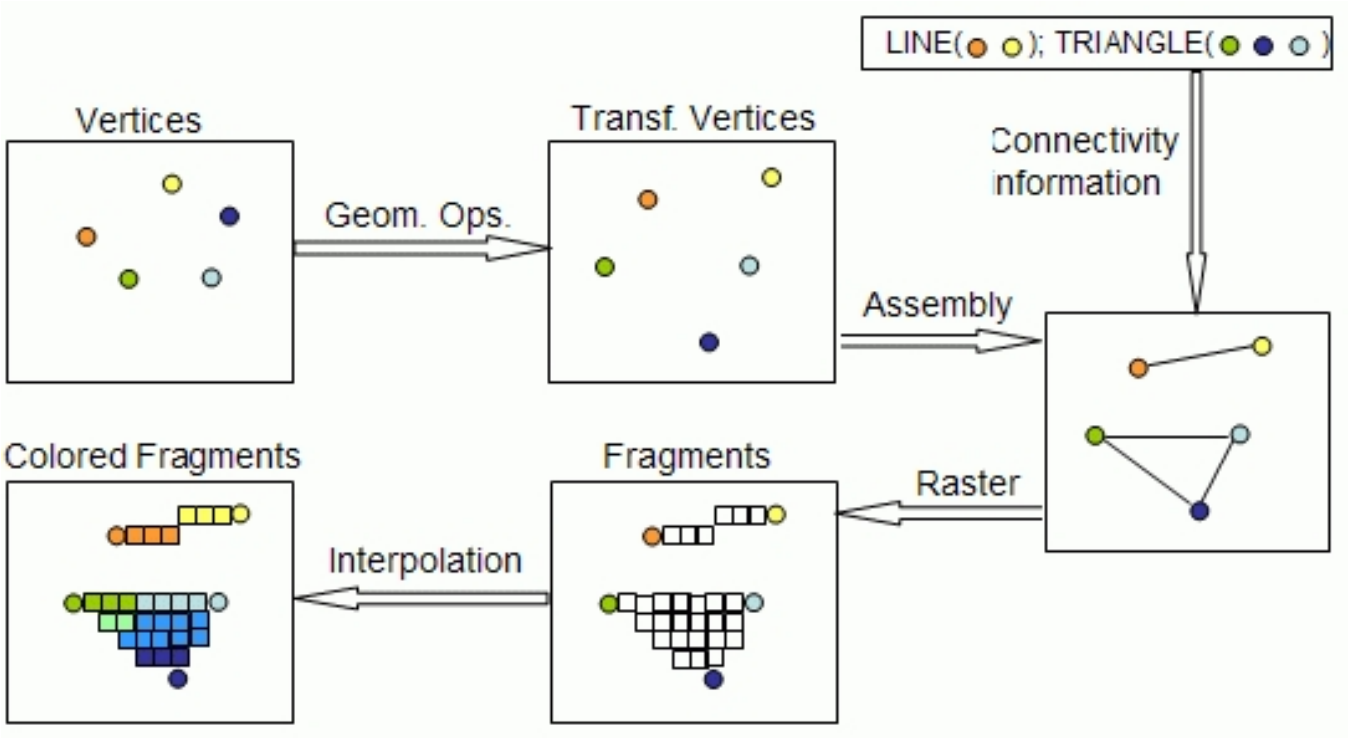
Etapa *Fragment Texturing & Coloring*

- Entrada: información interpolada de los fragmentos
- Operaciones:
  - Combinar color interpolado con un *texel* (elemento de textura)
  - Combinar color con efectos globales (niebla y otros)
- Salida: color final del fragmento

Etapa *Per-Sample Operations*

- Entrada:
  - Conjunto de fragmentos
    - posición en pantalla (incluyendo profundidad)
    - color
  - Se realizan varios tests para descartar el fragmento
    - *Depth test*, *Scissor test*, *Alpha test*, *Stencil test*,
  - Si los tests se pasan, se dibuja el *pixel*
    - se puede mezclar con el color de la pantalla para ese *pixel* (*blending*)

Resumiendo


LINE( ); TRIANGLE( )
Vertices
Transf. Vertices
Connectivity
nformation
Geom. Ops.
Assembly
Colored Fragments
Fragments
Raster
Interpolation