

Introducción a los gráficos por computador

Informática Gráfica I

Material de: **Ana Gil Luezas**
Adaptado por: **Elena Gómez y Rubén Rubio**
{mariaelena.gomez,rubenrub}@ucm.es



Contenido

- 1 Motivación
- 2 Primitivas geométricas
- 3 Otros elementos
 - Colores
 - Imágenes rasterizadas
- 4 Informática Gráfica
 - Proceso de visualización
 - Escena
 - Cámara
 - Cauce gráfico
 - GPU
 - APIs

Motivación

Definición

- Gráficos por computador:
 - Generar imágenes mediante un computador.
 - Bibliotecas gráficas:
 - **OpenGL** (*Open Graphics Library*).
 - **Vulkan**.
 - **Direct3D**.
- Procesamiento de imágenes:
 - Mejorar, alterar o analizar imágenes previamente creadas, por una cámara de fotos, por vídeo, o por computador, entre otros.
 - Bibliotecas y programas:
 - **OpenCV** (*Open Computer Vision*, biblioteca libre de visión artificial).

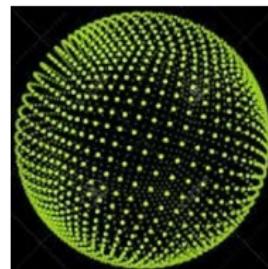
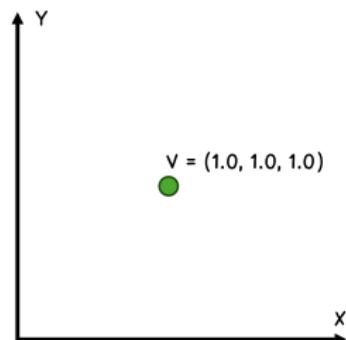
Motivación

Primitivas geométricas

- Un gráfico se compone de una serie de elementos básicos, denominados **primitivas geométricas**:
 - Puntos.
 - Poli-líneas.
 - Regiones rellenas (triángulos).
- Cada primitiva tiene asociada atributos, tales como **color** y **grosor**, entre otros.
- También pueden incorporar:
 - Imágenes rasterizadas (texturas).
 - Texto.

Dibujo por puntos

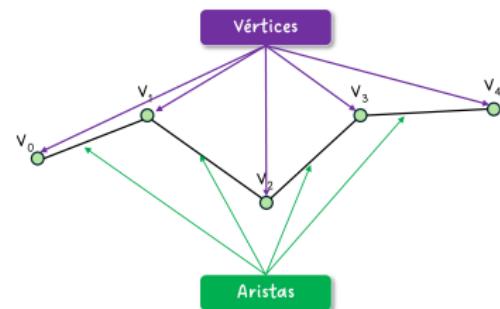
- Un **punto** se determina por las **coordenadas (x, y, z)** de un vértice en el espacio 3D.
- Un gráfico construido a base de puntos se denomina **dibujo por puntos**.
- Atributos asociados: **grosor** y **color**.



Poli-líneas

Una **polilínea** es una sucesión conectada de **segmentos** (aristas) determinados por **vértices** (coordenadas).

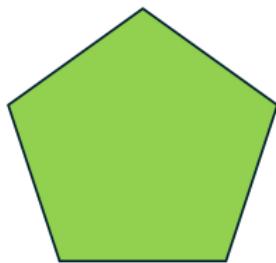
- Atributos asociados: **grosor**, **color** y **patrón de trazado**.
- La polilínea más simple es un segmento.
- Un gráfico construido a base de polilíneas se denomina **dibujo de líneas**.



Poli-líneas

El **polígono** es una polilínea en la que el primer y el último vértice están conectados mediante una arista.

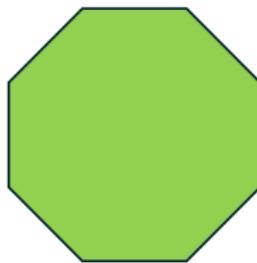
- Es **simple** si no existen dos aristas que se crucen.
- Es **convexo** si el segmento que une cualquier par de sus vértices es interior, en cualquier otro caso es **cóncavo**.



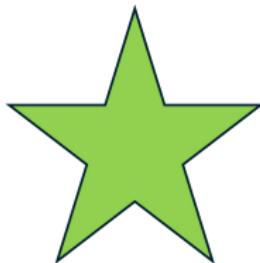
Polígono simple



Polígono no simple



Polígono convexo

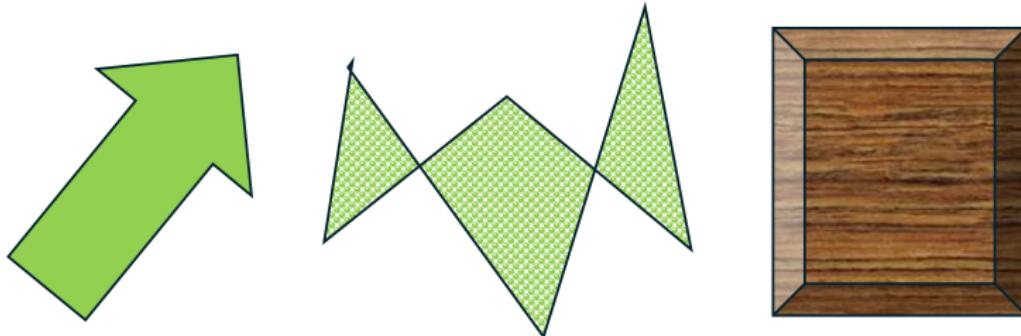


Polígono no convexo

Regiones rellenas

Las **regiones rellenas** son formas cerradas, llenas con algún color, patrón o textura.

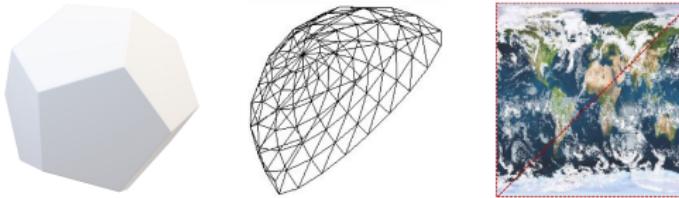
- En general están formadas por triángulos (polígono simple, convexo y plano).
- Atributos asociados: **color, patrón de relleno y textura.**



Triángulos

Los **triángulos** son fundamentales en informática gráfica:

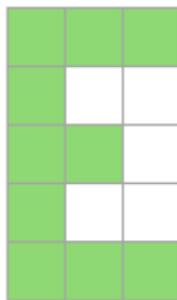
- Son fáciles de definir, son simples, convexos y planos, por lo que muchos de los algoritmos de *rendering* (proceso de visualización por el cual se genera una imagen a partir de un modelo matemático) funcionan de forma óptima para triángulos.
- Se utilizan para aproximar superficies y componer otros polígonos.



Texto

Los **caracteres** pueden definirse mediante una polilínea o un mapa de bits.

- *True Type Font (TTF)* tipos de letra escalables.
- Atributos asociados: **color, tamaño, espaciado y fuente**.



Bitmapped

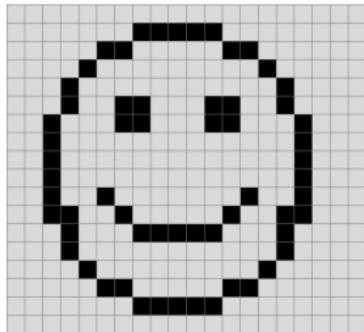


stroke

Imágenes rasterizadas

Las **imágenes rasterizadas** son imágenes previamente generadas que aparecen en el gráfico.

- Por ejemplo, podemos incorporar una imagen almacenada en un fichero .bmp dentro de nuestro gráfico.
- Una imagen rasterizada se almacena en el ordenador como una matriz de valores numéricos (bitmap).



2	2	2	2	2	2	2	2	2
2	2	2	2	2	2	2	2	7
2	2	2	2	2	2	7	7	1
2	2	2	2	2	2	7	1	1
2	2	2	2	2	2	7	1	1
2	2	2	2	2	2	7	1	1
2	2	2	2	2	2	7	1	1
2	2	2	2	2	2	7	1	1
2	2	2	2	2	2	7	1	1

👉 Cada valor numérico indica el color de un *téxel* (píxel).

Modelos de color

- **Modelo de color RGB (Red, Green, Blue):**

- Utilizado en monitores.
- Los colores se obtienen mediante la mezcla aditiva (sobre negro) de la intensidad de los colores rojo, verde y azul (colores primarios)

- **Modelo aditivo:**

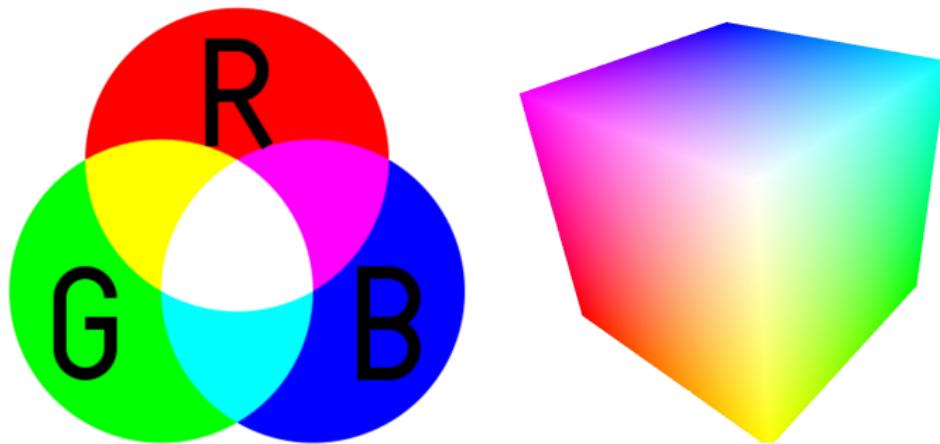
- Los colores se obtienen sumando las tres componentes.
- Colores secundarios: **cian**, **magenta**, **amarillo**.
 - El **cian** se obtiene sumando el **verde** y el **azul**.
 - El **magenta** se obtiene sumando el **rojo** y el **azul**.
 - El **amarillo** se obtiene sumando el **rojo** y el **verde**.
 - El **blanco** es la suma de los tres primarios.

Modelos de color

- ¿Cuáles son los colores primarios?
- ¿Cuál es la relación entre el modelo matemático y un espacio de color (gama de colores absolutos)?

Modelos de color

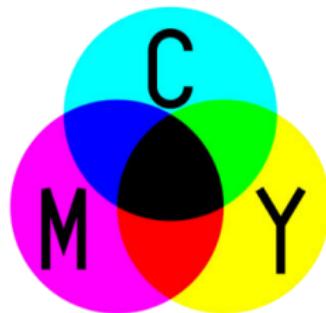
Modelo aditivo Rojo, verde, azul



Modelos de color

Modelo de color CMY (Cyan, Magenta, Yellow)

- Utilizado en impresoras.
- Los colores se obtienen mediante la mezcla sustractiva (sobre blanco) de los colores cian, magenta, amarillo (colores primarios).
 - Cyan es el opuesto al rojo (filtra o absorbe el rojo).
 - Magenta el opuesto al verde.
 - Yellow el opuesto al azul.
- Colores secundarios: rojo, verde y azul.
- CMYK añade el color negro ($K \rightarrow$ negro).



Representación del color

Modelo de color RGB (Red, Green, Blue):

- Un color es un tripleta ordenada (r , g , b), que representa las intensidades de rojo, verde y azul, respectivamente.
- El intervalo de valores para indicar la intensidad de cada componente suele ser:
 - un entero entre 0 y 255 (1 byte).
 - un real entre 0 y 1 (float/double).

Representación del color

Modelo de color RGB (Red, Green, Blue):

- La **profundidad de color** se define como la suma de los bits asociados a las componentes r, g, b.
 - Si utilizamos 8 bits para cada color, tenemos una profundidad de color de 24 bits por píxel, y una gama de colores de $2^{24} = 16,777,216$ posibles colores.
- La profundidad de color de 24 bits por píxel, se denomina **color verdadero**.
- La profundidad de 32 bits agrega un **canal alfa** (RGBA) que representa la transparencia (0, transparente; 1, opaco).

Representación del color

- Para colores de 3 bytes se usa la base hexadecimal, siendo cada componente de 1 byte son dos dígitos hexadecimales:

	0xHex	Decimal	3 int (bytes)	3 float/double
Rojo	0xFF0000	16711680	(255, 0, 0)	(1.0, 0.0, 0.0)
Verde	0x00FF00	65280	(0, 255, 0)	(0.0, 1.0, 0.0)
Azul	0x0000FF	255	(0, 0, 255)	(0.0, 0.0, 1.0)
Cian	0x00FFFF	65535	(0, 255, 255)	(0.0, 1.0, 1.0)
Magenta	0xFF00FF	16711935	(255, 0, 255)	(1.0, 0.0, 1.0)
Amarillo	0xFFFF00	16776960	(255, 255, 0)	(1.0, 1.0, 0.0)
Blanco	0xFFFFFFFF	16777215	(255, 255, 255)	(1.0, 1.0, 1.0)
Gris	0x808080	8421504	(128, 128, 128)	(0.5, 0.5, 0.5)
Negro	0x00000000	0	(0, 0, 0)	(0.0, 0.0, 0.0)

Representación del color

- **Operaciones:** suma, resta, producto y combinación lineal.

$$C_1 = (r_1, g_1, b_1) \text{ y } C_2 = (r_2, g_2, b_2) : 0 \leq r_i, g_i, b_i \leq 1$$

$$C_1 \cdot C_2 = (r_1 \cdot r_2, g_1 \cdot g_2, b_1 \cdot b_2)$$

$$C_1 + C_2 = (r_1 + r_2, g_1 + g_2, b_1 + b_2)$$

$$C_1 - C_2 = (r_1 - r_2, g_1 - g_2, b_1 - b_2)$$

$$C_1 \cdot s = (r_1 \cdot s, g_1 \cdot s, b_1 \cdot s) \quad 0 \leq s \leq 1$$

Representación del color

- **Interpolación lineal** (mix):

$$C_1 = (r_1, g_1, b_1), C_2 = (r_2, g_2, b_2) : 0 \leq r_i, g_i, b_i \leq 1$$

$$\text{mix}(C_1, C_2, a) = C_1 \times (1 - a) + C_2 \times a : \text{si } 0 \leq a \leq 1$$

- La **luminancia** (escala de grises) de un color RGB sería $\text{lum} = \frac{R+G+B}{3}$, y el color asociado sería el gris ($\text{lum}, \text{lum}, \text{lum}$). Pero debido a que el ser humano no percibe igual los tres colores, se dan distintos pesos:

$$\text{lum} = R \times 0,299 + G \times 0,587 + B \times 0,114$$

$$\text{lum} = R \times 0,2125 + G \times 0,7154 + B \times 0,0721 (\text{sRGB})$$

Representación del color

- Se utiliza una **tabla de colores** que ofrece una asociación configurable entre el valor (índice) de cada *téxel* y el color final que representa:

Tabla de colores

i	R	G	B
0	0	0	0
1	255	255	255
2	100	100	100
3	255	0	255
4	0	255	0
5	0	0	255

1 byte:
 2^8 (256)
 posibles
 colores

3 bytes: 2^{24} posibles
 colores (color verdadero)

Imagen									
0	0	0	0	0	0	0	0	0	0
0	2	3	4	5	5	4	3	2	0
0	1	1	2	2	2	2	1	1	0
0	1	1	2	2	2	2	1	1	0
0	2	3	4	5	5	4	3	2	0
0	0	0	0	0	0	0	0	0	0

Una imagen que utiliza pocos colores diferentes ocupa menos espacio con índices (hay que guardar la tabla)

Imágenes rasterizadas

Formato PNG (*Portable Network Graphics*).

- Utiliza un **algoritmo de compresión sin perdida para bitmaps**: En blanco y negro, en color RGB(A) y con paleta de colores.
- Permite transparencias (canal A o componente Alpha).

Rango total de opciones de color soportados

Profundidad de bits por canal	1	2	4	8	16
Imagen indexadas (1 canal)	1	2	4	8	
Escala de grises (1 canal)	1	2	4	8	16
Escala de grises con alfa (2 canales)				16	32
Color verdadero (RGB) (3 canales)				24	48
Color verdadero con alfa (4 canales)				32	64

Generación de imágenes

- Proceso de visualización (renderizado)
 - ↳ Cauce gráfico (OpenGL pipeline)
 - ↳ GPU (Graphics Processing Unit)
- Escena, cámara y puerto de vista

Proceso de visualización o renderizado

- Dada una escena:
 - Hacer y revelar (mostrar) una foto.
 - Determinar el color de cada píxel.
- Técnicas para determinar el color de cada píxel:
 - **Interactivas:** procesamiento de primitivas (triángulos) dadas por vértices y posterior relleno (interpolación).
 - 👉 Cauces gráficos: OpenGL, Direct3D, Vulkan
 - **Realistas:** por píxel
 - 👉 Trazado de rayos (*Ray Tracing*), Radiosidad (*Radiosity*), ...

Elementos de la escena

- Objetos:
 - Descripción local: Geometría y materiales.
 - Disposición en la escena: Posición y orientación.
- Luces:
 - Descripción lumínica.
 - Disposición en la escena.
- Cámara:
 - Descripción óptica: Volumen de vista y proyección.
 - Disposición en la escena.
- Puerto de vista (ventana en la que se muestra):
 - Posición y dimensiones en píxeles

Escena

- Diseñamos el objeto (**escena**):

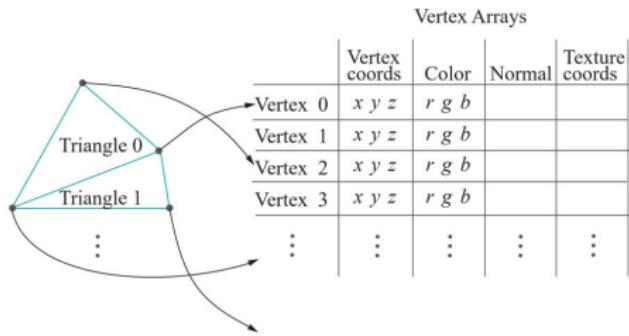


Malla

- Diseño del objeto (**escena**):

👉 **Sistema de coordenadas local.**

👉 **Malla (mesh)**: Coordenadas de los vértices, componentes del color, coordenadas de vectores normales, coordenadas de textura.

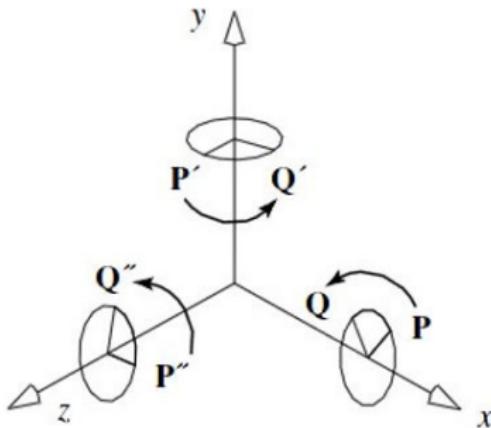


- **Matriz de modelado:**

👉 Transformaciones afines: **traslación, rotación, escala.**

Transformaciones afines

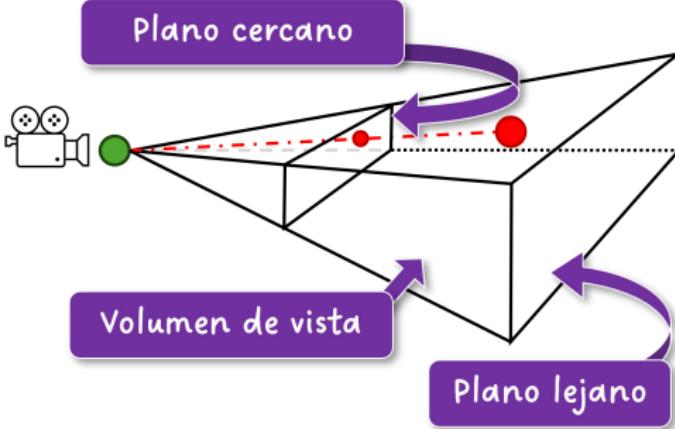
- **Traslaciones:** vector (t_x, t_y, t_z)
- **Escalas:** factor $S = (s_x, s_y, s_z)$
- **Rotaciones** sobre los ejes:



Cámara

- Colocamos la **cámara** y elegimos el tipo de óptica (**proyección**) que deseamos (**ortogonal o perspectiva**).
- Establecemos el **volumen de vista**. Los objetos dentro del volumen serán visibles. El resto de la escena no será visible.

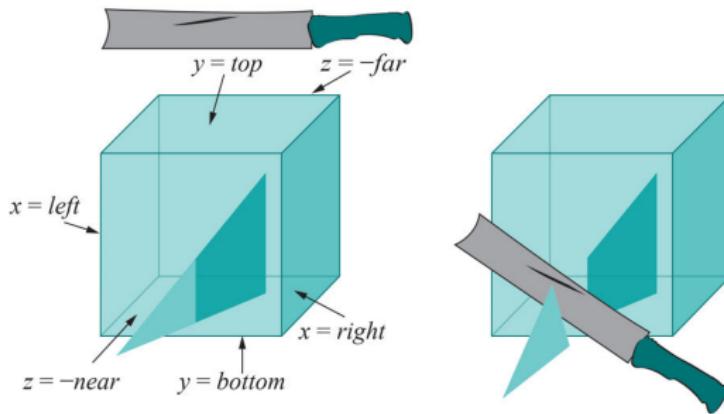
Proyección Perspectiva



Recorte

Volumen de vista

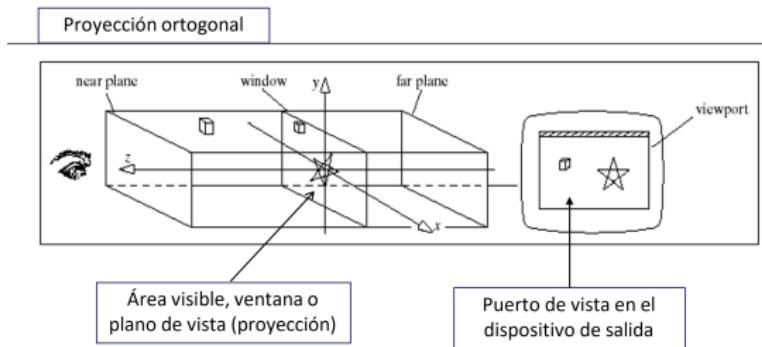
- Los objetos dentro del volumen serán visibles.
- El resto de la escena no será visible.
- Algunos objetos pueden ser recortados (*clipping*).



Puerto de vista

Puerto de vista

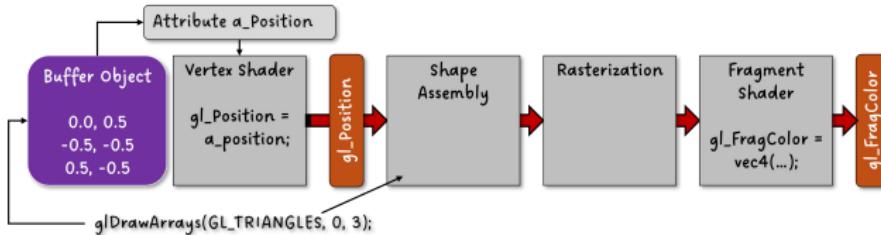
- La proyección obtenida en el plano de vista (proyección) se traslada al puerto de vista establecido en la ventana de visualización.



Cauce gráfico

Etapas del cauce (*pipeline*) gráfico

- Una vez que la malla (coordenadas y atributos de los vértices) se encuentra en la GPU se ejecutan varios procesos encadenados:
 - ① **Vertex Shader**: transforma las coordenadas de cada vértice
 - ② **Rasterization**: genera los fragmentos del interior de la primitiva
 - ③ **Fragment Shader**: determina el color de cada fragmento



Cauce gráfico

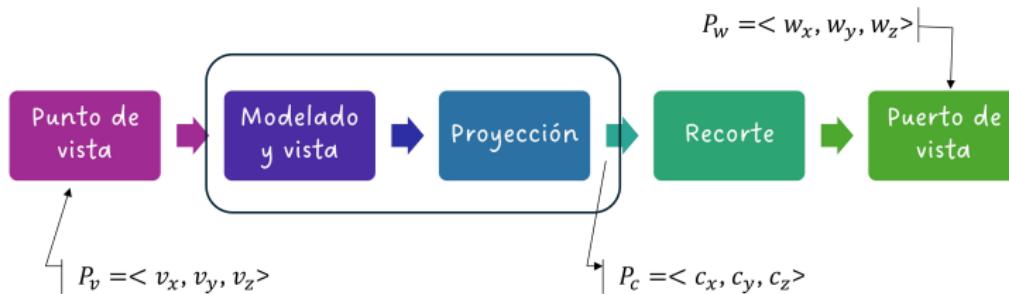
Vertex Shader

- Cada vértice sufre una serie de transformaciones. Cada transformación se realiza multiplicando las coordenadas actuales del vértice por una matriz:
 - **Matriz de modelado y vista (V.M)**
 - **Matriz de modelado (M)**: recoge la traslación, escala y rotaciones para establecer el objeto en la escena
 - **Matriz de vista (V)**: recoge la inversa de la transformación que establece la cámara en la escena
 - **Matriz de proyección (Pr)**: Proyecta la escena 3D sobre el plano de vista, guardando la distancia a la cámara (profundidad)
- **Matriz del puerto de vista (Vp)**: Ajusta la imagen proyectada a la parte de la ventana especificada por el puerto de vista.

Cauce gráfico

Vertex Shader

- Transforma las coordenadas originales de los vértices.

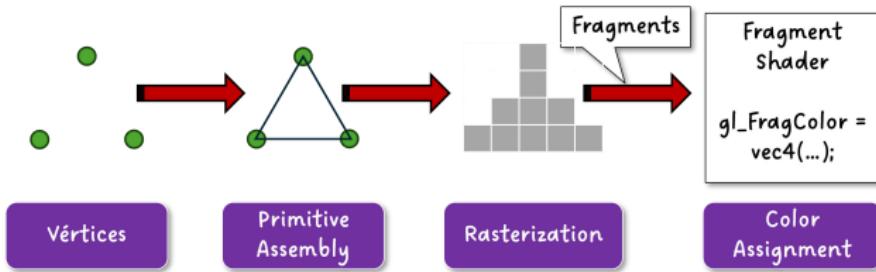


$$Pc = (Proy \times V \times M) Pv$$

Cauce gráfico

Rasterización

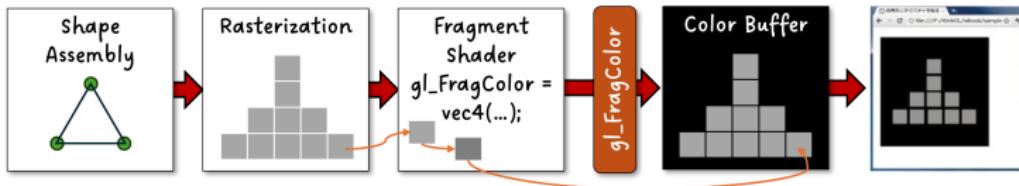
- Genera los fragmentos del interior de la primitiva realizando un proceso de interpolación sobre los atributos de los vértices de la primitiva.



Cauce gráfico

Fragment Shader

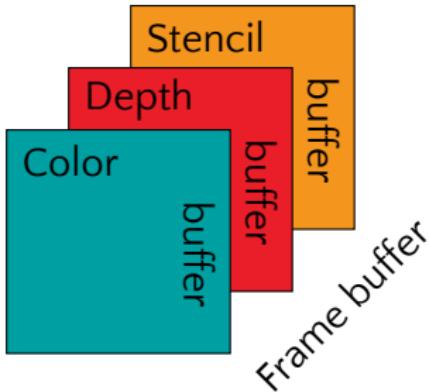
- Determina el color de cada fragmento escribiendo el resultado en el **Color Buffer (BACK / FRONT)**
- También se utiliza el **Depth Buffer** para determinar la visibilidad de cada fragmento.



- Los colores de los fragmentos se pueden combinar para obtener el color final que se visualizará.

Frame Buffer

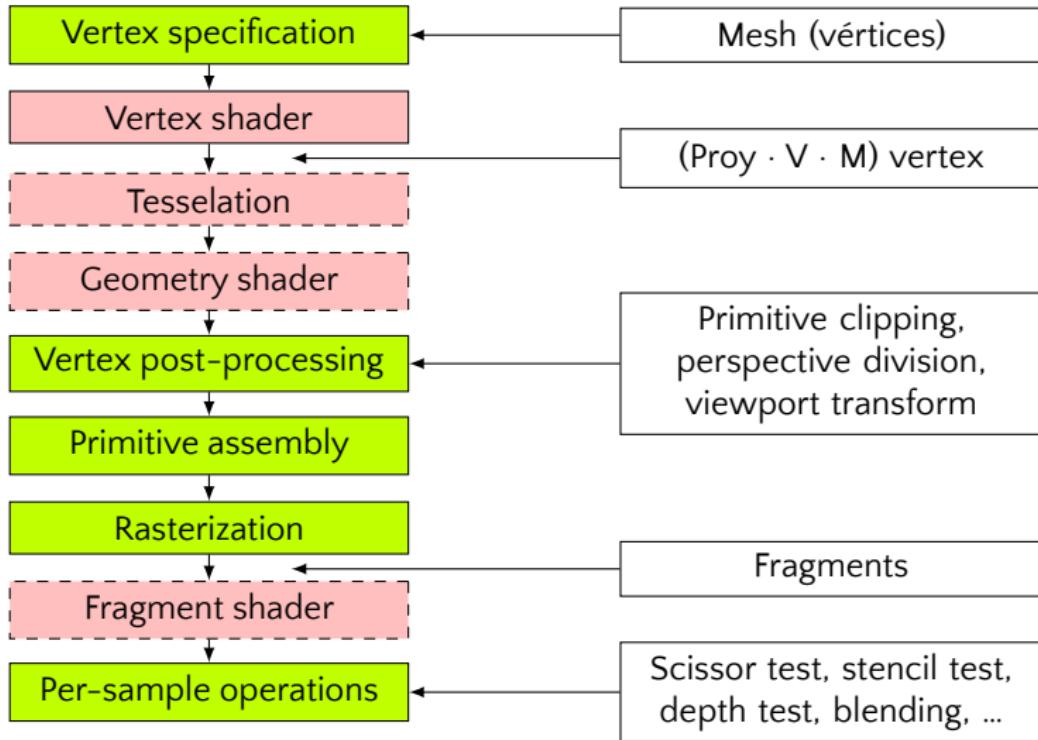
- Guarda información sobre los píxeles en la GPU
 - Color Buffer** (FRONT y BACK): componentes RGBA del fragmento.
 - Depth Buffer** (Z-buffer): distancia del fragmento al plano de vista
 - Stencil Buffer**: marcas para restringir los fragmentos a procesar



Doble Buffer

- (FRONT / BACK): Mientras se realiza el proceso, se muestra un buffer y se escribe en el otro.
- Al finalizar el proceso, se intercambian.

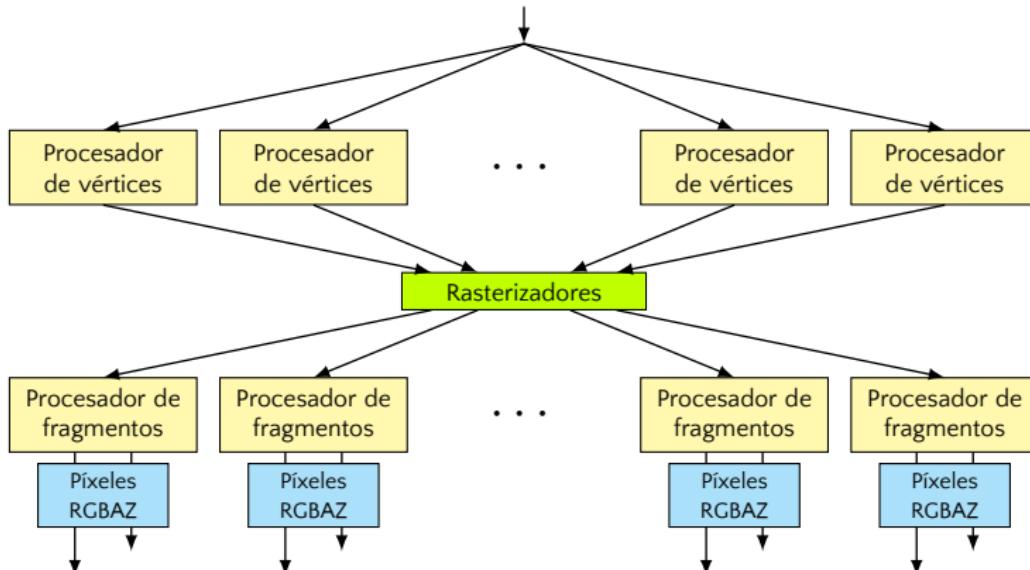
Cauce gráfico



Fuente: https://www.khronos.org/opengl/wiki/Rendering_Pipeline_Overview

GPU

- La interacción con la GPU la hacemos a través de **OpenGL**.
- Muchos núcleos (*cores*) especializados que trabajan en paralelo: El mismo proceso sobre diferentes datos independientes.



APIs gráficas – OpenGL

Numerosas versiones de OpenGL hasta la fecha: 1.0, 1.1, 1.2, 1.2.1, 1.3, 1.4, 1.5, 2.0, 2.1, 3.0, 3.1, 3.2, 3.3, 4.0, 4.1, 4.2, 4.3, 4.4, 4.5, 4.6

● 1992-2003: OpenGL 1.X

- Primera especificación que formaliza una API gráfica independiente del hardware e implementada por los fabricantes.
- El cauce gráfico es configurable, pero no se puede programar.

● 2004-2006: OpenGL 2.X

- Incorpora la posibilidad de programar algunas etapas de la tubería gráfica (*vertex and fragment shaders*) utilizando el lenguaje GLSL (OpenGL Shading Language), un lenguaje similar al C.

APIs gráficas – OpenGL

● 2008-2010: OpenGL 3.X

- Declara como obsoleta una parte importante de las funciones de las primeras versiones del lenguaje.
- OpenGL 3.2 introduce los perfiles *core* y *compatibility* con la funcionalidad recomendada y obsoleta respectivamente.

● 2010-2017: OpenGL 4.X

- Facilidades de depuración, mejoras en *shaders* y de eficiencia, etc.

APIs gráficas – Alternativas

- **Direct3D** de Microsoft ha sido el competidor tradicional de OpenGL desde 1996.
- **OpenGL ES** se introdujo en 2003 como una API simplificada para dispositivos móviles.
- Apple presenta **Metal** en 2014 como un reemplazo de más bajo nivel de OpenGL, inicialmente para iOS.
- El consorcio Khronos se hace cargo de la API **Vulkan** en 2017 y abandona el desarrollo de nuevas versiones de OpenGL.
- Metal, Vulkan y Direct3D 12 son APIs de mucho más bajo nivel que OpenGL, que permiten aprovechar mejor el hardware a costa de una menor abstracción y mayor dificultad de aprendizaje.
- OpenGL sigue siendo ampliamente usada y tiene una comunidad activa de desarrolladores.