

Render Pipeline

Gráficas por Computadora (CC2018) - 2026

Render Pipeline

Semestre 02, 2026

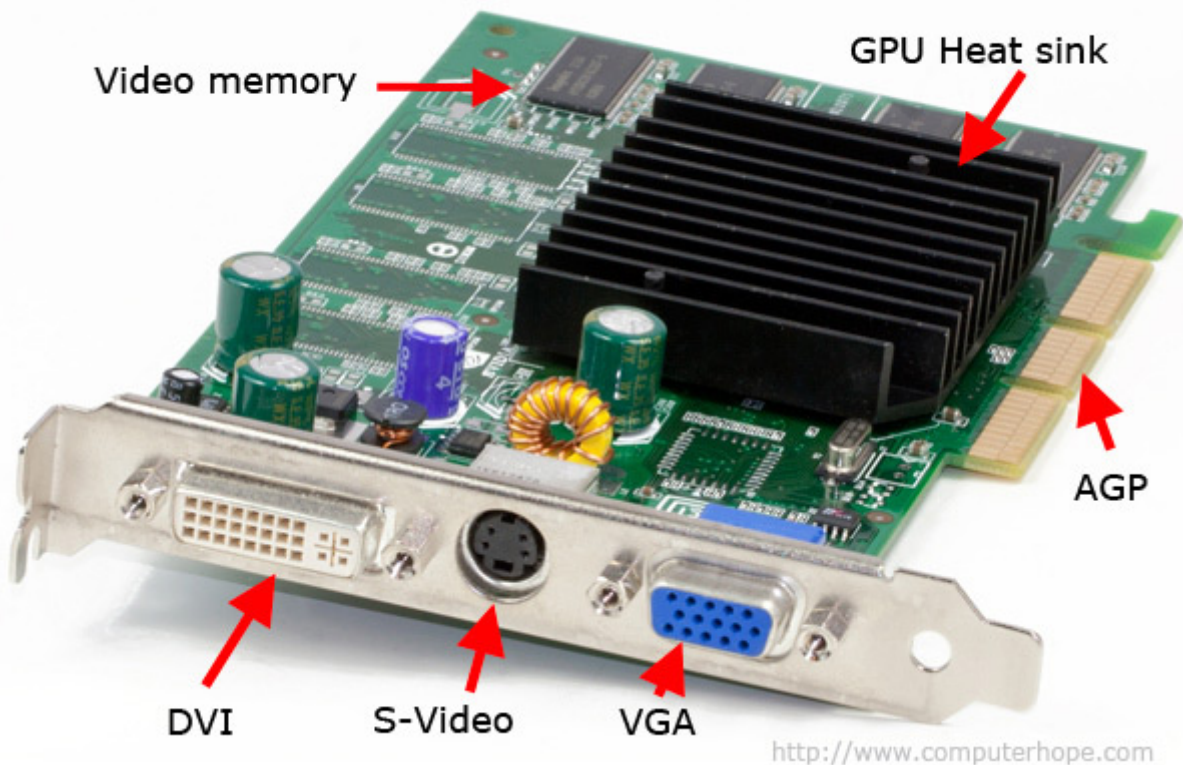
Objetivo

- Describir el render pipeline como la secuencia de etapas que convierte una escena 3D en una imagen 2D.
- Distinguir las etapas programables de las etapas fijas del hardware.
- Entender el vertex shader como la etapa que lleva cada vértice de su espacio local al clip space.
- Definir los uniforms y diferenciarlos de los atributos por vértice y de los varyings interpolados.
- Explicar qué es un fragmento y qué información transporta hacia el fragment shader.
- Seguir el recorrido de un dato, desde el archivo del modelo hasta el píxel encendido.

Render Pipeline

El render pipeline es el conjunto ordenado de etapas que transforma una escena 3D en la imagen 2D que aparece en la pantalla.

Internal computer video expansion card



Funciona como una línea de ensamblaje, donde la salida de una etapa alimenta a la siguiente. La primera etapa corre en la CPU y el resto dentro de la GPU.

El pipeline frente al ray tracer

Son dos maneras distintas de resolver el mismo problema.

- Render pipeline: atajo optimizado que proyecta la geometría sobre la pantalla y rellena los píxeles que cubre. Es rápido y por eso lo usa el hardware gráfico.
- Ray tracer: simulación física del recorrido de la luz. Es más fiel a la realidad y mucho más lento, porque cada rayo se resuelve contra toda la escena.

Conviene aclarar que el pipeline es un concepto, no una pieza de hardware. En una API como OpenGL o Vulkan las etapas corren dentro de la GPU, mientras que

en un software rasterizer escrito a mano las mismas etapas corren en la CPU, en el mismo orden y con las mismas responsabilidades.

Usos del render pipeline

- Convertir geometría, materiales y cámara en una imagen visible.
- Aplicar traslaciones, rotaciones, escalas y proyecciones.
- Simular la interacción de la luz con los materiales.
- Integrar sombras, reflejos y transparencias.
- Sostener tiempo real en videojuegos y simuladores.
- Alcanzar máxima calidad en render offline para cine.

Las etapas del pipeline

El recorrido completo pasa por seis etapas antes de llegar al framebuffer.

1. Aplicación, en la CPU, que prepara la escena y la envía.
2. Vertex shader, que transforma cada vértice.
3. Primitive assembly y clipping, que arman y recortan las figuras.
4. Rasterización, que convierte las figuras en fragmentos.
5. Fragment shader, que calcula el color de cada fragmento.
6. Operaciones por fragmento, que deciden qué se escribe en pantalla.

El orden es estricto, porque cada etapa consume lo que produjo la anterior.

Etapas programables y etapas fijas

- Etapa programable: su comportamiento lo define un programa escrito por el desarrollador. Ese programa se llama shader y se ejecuta dentro de la GPU.

- Etapa fija: está implementada en el hardware. No se reescribe, solo se configura con parámetros como el modo de dibujo o la función de profundidad.

Las etapas fijas son fijas porque así corren mucho más rápido, a costa de flexibilidad.

Etapas	Dónde corre	Naturaleza
Aplicación	CPU	Código propio
Vertex shader	GPU	Programable
Primitive assembly y clipping	GPU	Fija
Rasterización	GPU	Fija
Fragment shader	GPU	Programable
Operaciones por fragmento	GPU	Fija configurable

Aplicación

La aplicación es la primera etapa del pipeline. Se ejecuta en el procesador principal y prepara todos los datos que la GPU necesitará para dibujar el cuadro.

- Los vértices de cada modelo cargado desde disco.
- La transformación que ubica a cada objeto en la escena.
- La posición y la orientación de la cámara.
- Los parámetros globales del cuadro, que son los uniforms.

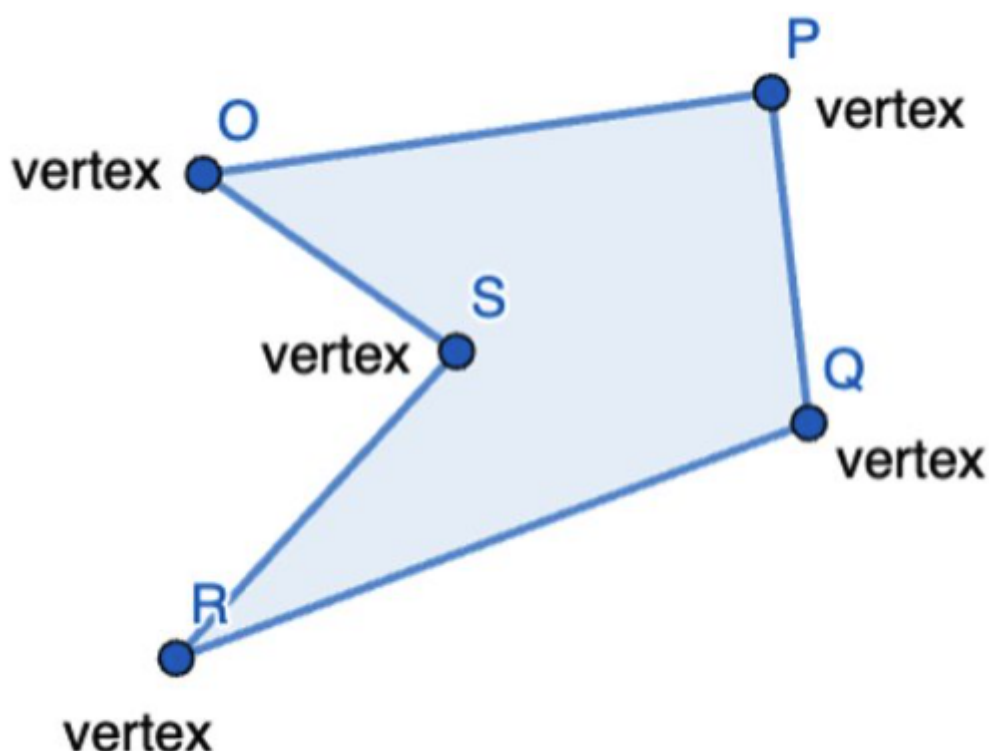
El draw call

El draw call es la orden que la aplicación envía a la GPU para que dibuje un conjunto de vértices con un programa de shaders y una configuración determinados.

Se emite a través de una API gráfica como OpenGL, DirectX, Vulkan o Metal. Cada draw call tiene un costo fijo, así que los motores agrupan objetos para emitir la menor cantidad posible.

Vertex Shader

El vertex shader es la primera etapa programable de la GPU. Recibe un vértice con sus atributos y devuelve ese mismo vértice transformado al clip space.



Se ejecuta una vez por cada vértice del modelo, y los vértices se procesan en paralelo, sin conocerse entre sí.

El vértice y sus atributos

- Vértice: punto del espacio que define una esquina de la geometría del modelo.
- Atributo: dato adjunto a un vértice, distinto para cada uno, que la aplicación envía junto con la posición.

Los atributos más comunes son cuatro.

- Posición: dónde está el vértice en el espacio del modelo.
- Normal: hacia dónde mira la superficie en ese vértice.
- Coordenada UV: qué punto de la textura le corresponde.
- Color: tinte propio del vértice, cuando el modelo lo trae.

Los espacios de coordenadas

Un vértice atraviesa una cadena de espacios antes de volverse un píxel.

Espacio	Significado
Model space	Coordenadas tal como las guardó el archivo del modelo, con su propio origen.
World space	El objeto ya colocado en la escena, junto a los demás.
View space	La escena vista desde la cámara, que queda en el origen.
Clip space	El resultado de aplicar la proyección, listo para recortarse.

Espacio	Significado
NDC	Coordenadas normalizadas en el rango de menos uno a uno.
Screen space	Píxeles reales dentro de la ventana.

El vertex shader produce los tres espacios intermedios y termina en clip space. Los dos últimos saltos los hace el hardware, sin código del programador.

Matriz de modelo

La matriz de modelo es una matriz de 4x4 que aplica traslación, rotación y escala a los vértices para colocar el objeto en su lugar dentro de la escena.

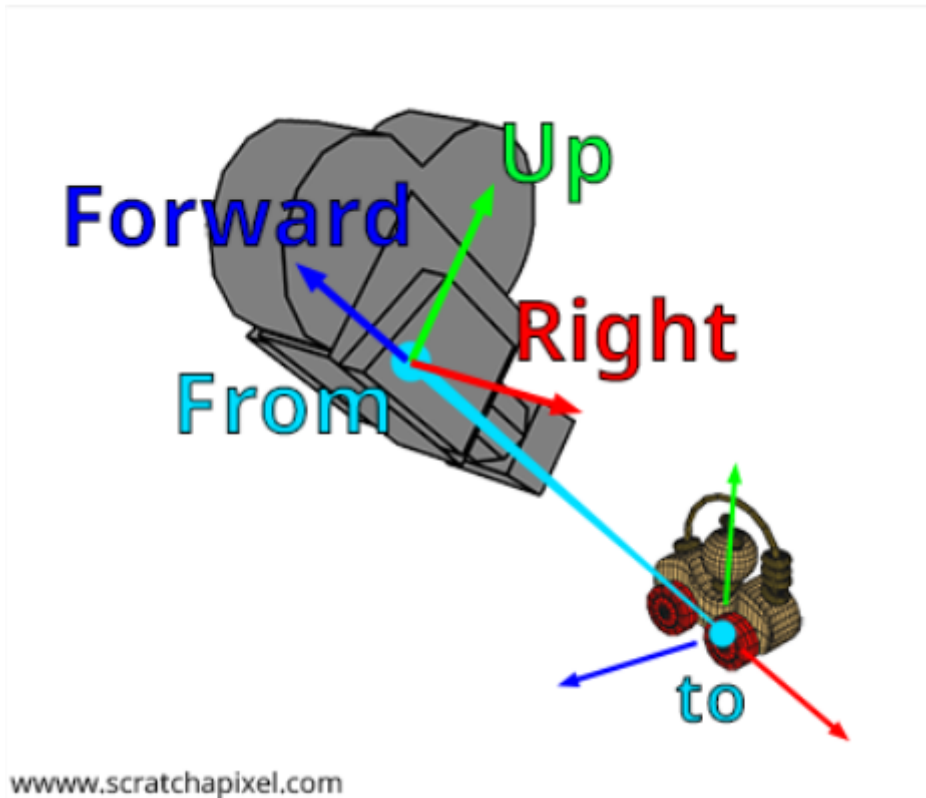
Una traslación de cinco unidades sobre el eje x se escribe así.

$$M = \begin{bmatrix} 1 & 0 & 0 & 5 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad M \cdot (x, y, z, 1) = (x+5, y, z, 1)$$

La matriz es de 4x4 y no de 3x3 porque la traslación no es una transformación lineal. Una matriz de 3x3 siempre deja el origen fijo, así que rota, escala y sesga, pero nunca mueve. La cuarta columna guarda el vector de traslación y la cuarta componente del punto es la que la activa.

Matriz de vista

La matriz de vista mueve toda la escena para que la cámara quede en el origen mirando hacia el eje z negativo.



Se arma con la rotación y la traslación inversas de la cámara.

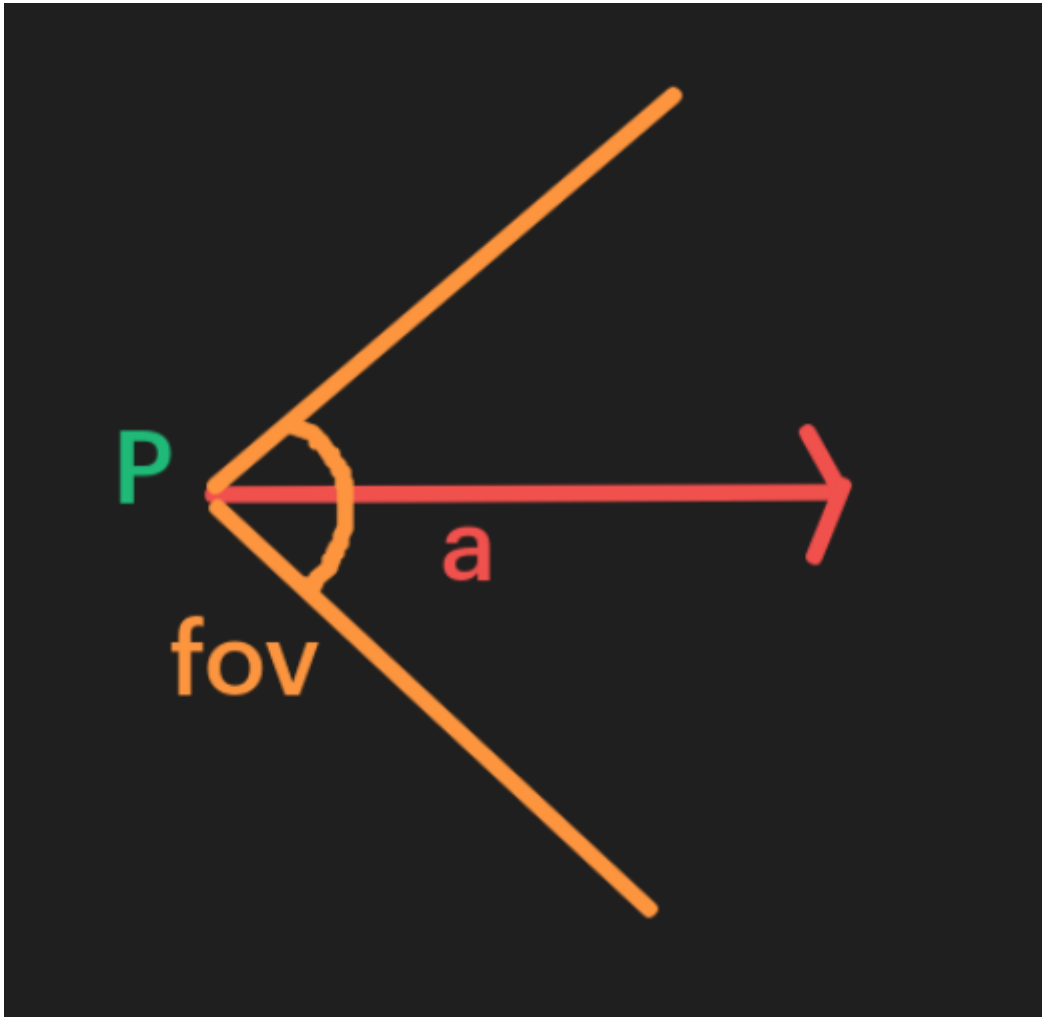
$$V = T^{-1}\{cam\} R^{-1}\{cam\}$$

En la práctica se construye con una función lookAt, que recibe el ojo, el objetivo y el vector hacia arriba. Conviene insistir en que esta matriz no mueve la cámara, mueve el mundo entero en sentido contrario.

Matriz de proyección

La matriz de proyección define el frustum, que es el volumen de espacio visible, y aplasta ese volumen dentro del cubo del clip space.

$$P = \begin{bmatrix} \frac{1}{\tan(\theta/2)} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan(\theta/2)} & 0 & 0 \\ 0 & 0 & \frac{f+n}{n-f} & \frac{2fn}{n-f} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$



- θ es el campo de visión, el ángulo vertical que abarca la cámara.
- a es el aspect ratio, el ancho del viewport dividido entre su alto.
- n es el near plane, el límite más cercano que se dibuja.
- f es el far plane, el límite más lejano que se dibuja.

La transformación completa

$$P_{clip} = P \cdot V \cdot M \cdot P_{objeto}$$

La matriz de modelo ubica el objeto dentro de la escena, la matriz de vista coloca esa escena frente a la cámara y la matriz de proyección aplica la perspectiva. El producto se lee de derecha a izquierda, porque el vértice entra por la derecha.

Coordenadas homogéneas

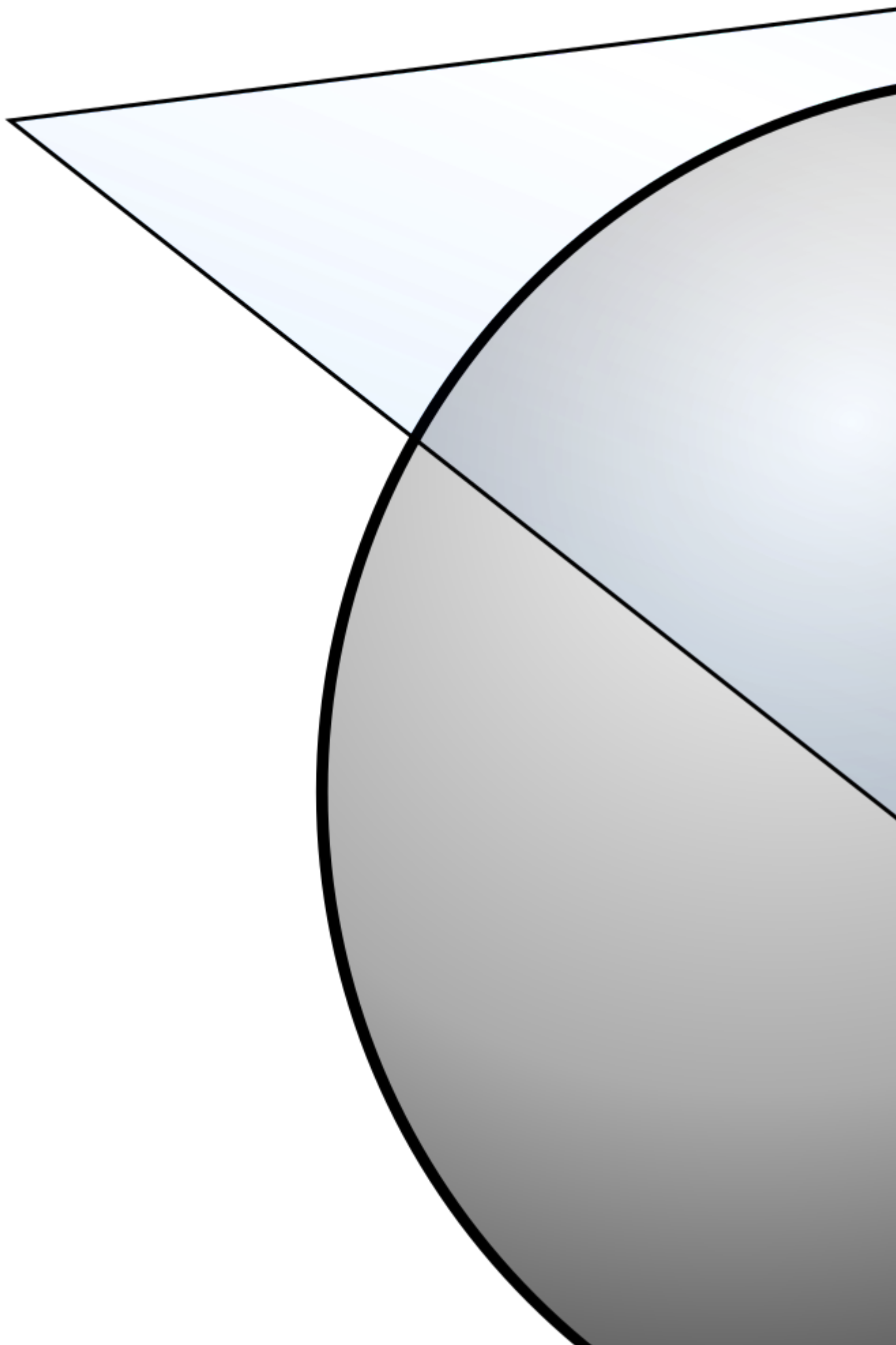
Las coordenadas homogéneas representan un punto de tres dimensiones con una cuarta componente w , que permite expresar la traslación como una multiplicación de matrices.

- Un punto se escribe $(x, y, z, 1)$ y sí se traslada.
- Una dirección se escribe $(x, y, z, 0)$ y no se traslada.

La traslación entra multiplicada por w , así que a las direcciones y a las normales no las toca.

Transformación de normales

Las normales no se transforman con la matriz de modelo, sino con su inversa transpuesta.



$$n' = (M^{-1})^T n$$

Con una escala no uniforme, usar la matriz de modelo dejaría la normal torcida y la iluminación saldría mal.

La salida del vertex shader

- Salida obligatoria: la posición del vértice en clip space, que en OpenGL se escribe en la variable `gl_Position`.
- Salida opcional: los atributos que seguirán viajando hacia el fragment shader, como la normal, la coordenada UV y el color.

El vertex shader no puede crear ni eliminar vértices, solo transformar el que recibe.

Uniforms

Un uniform es una variable global de un programa de shaders cuyo valor fija la aplicación desde la CPU y que permanece constante durante todo el draw call.

- Es visible desde cualquier etapa del programa de shaders, tanto el vertex como el fragment.
- Dentro del shader es de solo lectura, porque ninguna invocación puede modificarlo.
- Conserva su valor entre cuadros hasta que la aplicación lo actualiza.

Atributos, uniforms y varyings

Dato	Quién lo produce	Cada cuánto cambia
Atributo	La aplicación, en el buffer de vértices	En cada vértice

Dato	Quién lo produce	Cada cuánto cambia
Uniform	La aplicación, antes del draw call	En cada draw call
Varying	El vertex shader	En cada fragmento

Un varying es el valor que el vertex shader entrega y que la rasterización interpola a lo largo de la primitiva antes de pasarlo al fragment shader.

Uniforms típicos

- Las matrices de modelo, vista y proyección.
- La posición, el color y la intensidad de la luz.
- La posición de la cámara en el mundo.
- El tiempo transcurrido desde que arrancó el programa.
- La textura que se va a muestrear, enviada como sampler.
- El ancho y el alto del viewport en píxeles.

Ejemplo de uniform

Se requiere que una esfera lata, creciendo y encogiéndose sin parar, sin volver a cargar el modelo ni tocar sus vértices en la CPU.

La aplicación envía el tiempo como uniform antes de cada draw call, y el vertex shader escala la posición con un factor que depende de ese tiempo. Los vértices originales nunca cambian, porque lo que cambia es un único número por cuadro.

Primitive Assembly

El primitive assembly es la etapa que agrupa los vértices ya transformados para formar primitivas, que son las figuras básicas que el hardware sabe dibujar.

El vertex shader trabaja vértice por vértice, de forma aislada. La pantalla, en cambio, se llena con figuras completas y no con puntos sueltos.

Tipos de primitivas

- Punto, formado por un vértice.
- Línea, formada por dos vértices.
- Triángulo, formado por tres vértices.

El triángulo domina el pipeline entero. Tres puntos siempre definen un plano y la figura nunca queda cóncava, así que la interpolación de atributos sobre su superficie está bien definida.

Modo	Cómo agrupa los vértices
GL_POINTS	Cada vértice se dibuja por separado.
GL_LINES	Cada par de vértices forma una línea independiente.
GL_LINE_STRIP	Cada vértice se encadena con el anterior.
GL_LINE_LOOP	Igual que la cadena, pero el último se une con el primero.
GL_TRIANGLES	Cada grupo de tres vértices forma un triángulo.
GL_TRIANGLE_FAN	Todos los triángulos comparten el primer vértice.

Face culling

El face culling es el descarte de los triángulos que le dan la espalda a la cámara, decidido por el orden en que están escritos sus tres vértices.

$$s = (x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1)$$

El signo de s revela si el orden de los vértices quedó horario o antihorario, y con eso se sabe qué cara se ve.

Clipping

El clipping es el recorte de las primitivas contra el frustum, de modo que solo sobrevive la parte que cae dentro del volumen visible.

- Una primitiva completamente fuera se descarta sin más trámite.
- Una primitiva que cruza el borde se parte en triángulos nuevos que sí caben.

El ahorro es grande, porque rasterizar lo invisible costaría igual que rasterizar lo visible.

Perspective divide

El perspective divide es la división de las tres primeras componentes del vértice entre su componente w , que convierte el clip space en coordenadas normalizadas.

$$(x_{ndc}, y_{ndc}, z_{ndc}) = \left(\frac{x}{w}, \frac{y}{w}, \frac{z}{w} \right)$$

Aquí nace la perspectiva, porque la proyección deja w proporcional a la distancia y lo lejano termina encogido.

Viewport transform

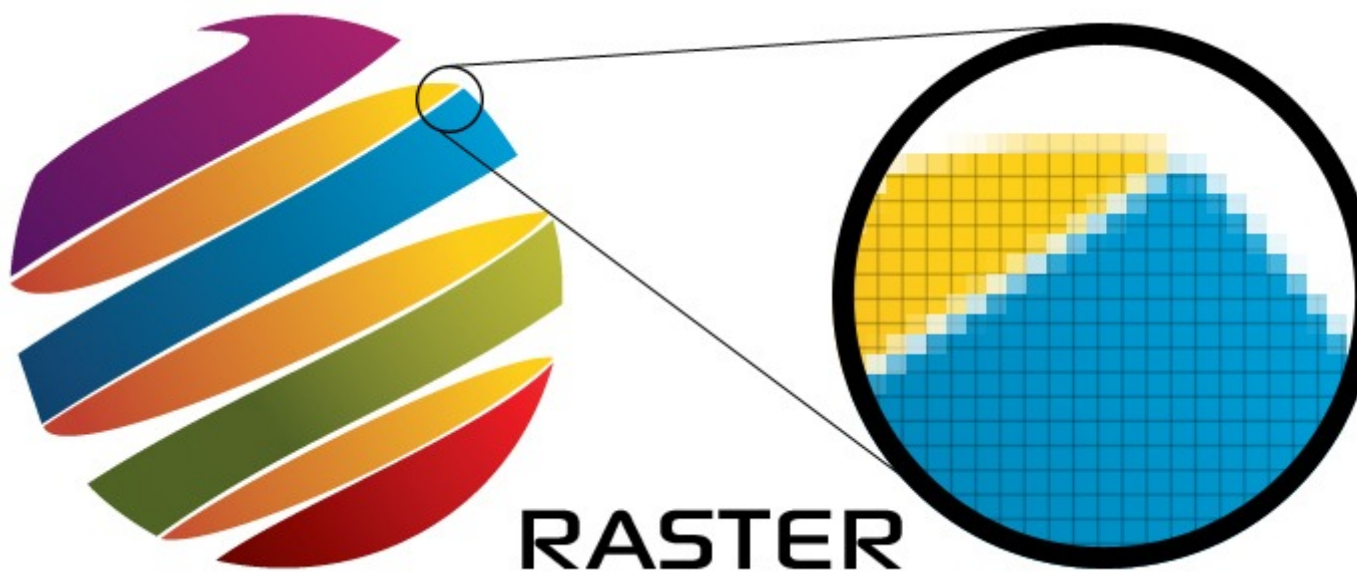
El viewport transform es el mapeo final que lleva las coordenadas normalizadas al rango de píxeles de la ventana donde se está dibujando.

$$x_{\text{pantalla}} = \frac{x_{\text{ndc}} + 1}{2} \cdot \text{ancho} \quad y_{\text{pantalla}} = \frac{1 - y_{\text{ndc}}}{2} \cdot \text{alto}$$

El eje vertical se invierte, porque en NDC crece hacia arriba y en pantalla crece hacia abajo.

Rasterización

La rasterización es la conversión de cada primitiva en el conjunto de fragmentos que cubre sobre la cuadrícula de la pantalla.



Es el salto de la geometría continua a la cuadrícula discreta de píxeles, y en el camino interpola los atributos que venían en los vértices.

El bounding box

El bounding box es el rectángulo mínimo alineado a los ejes que contiene por completo al triángulo que se va a rasterizar.

Se obtiene con el mínimo y el máximo de las coordenadas de los tres vértices. Solo se examinan los píxeles de ese rectángulo, en vez de recorrer la pantalla entera.

Coordenadas baricéntricas

Las coordenadas baricéntricas son tres pesos que expresan un punto del triángulo como combinación de sus vértices, calculados como razones de áreas.

$$\lambda_1 = \frac{A_{PBC}}{A_{ABC}}, \quad \lambda_2 = \frac{A_{PCA}}{A_{ABC}}, \quad \lambda_3 = \frac{A_{PAB}}{A_{ABC}}$$

Los tres pesos suman uno, y el punto está dentro del triángulo solo si ninguno sale negativo.

Interpolación de atributos

Cualquier atributo del punto interior se arma con los mismos tres pesos.

$$A_P = \lambda_1 A_A + \lambda_2 A_B + \lambda_3 A_C$$

- Coordenadas UV, para saber qué texel tocar.
- Normales, para que la iluminación varíe suave.
- Colores, para los degradados entre vértices.
- Profundidad, que decide después qué fragmento gana.

Como ejemplo, un triángulo con coordenadas UV $A=(0.5,0)$, $B=(0,1)$ y $C=(1,1)$, con el punto P justo en el centro y por lo tanto con pesos iguales, produce lo siguiente.

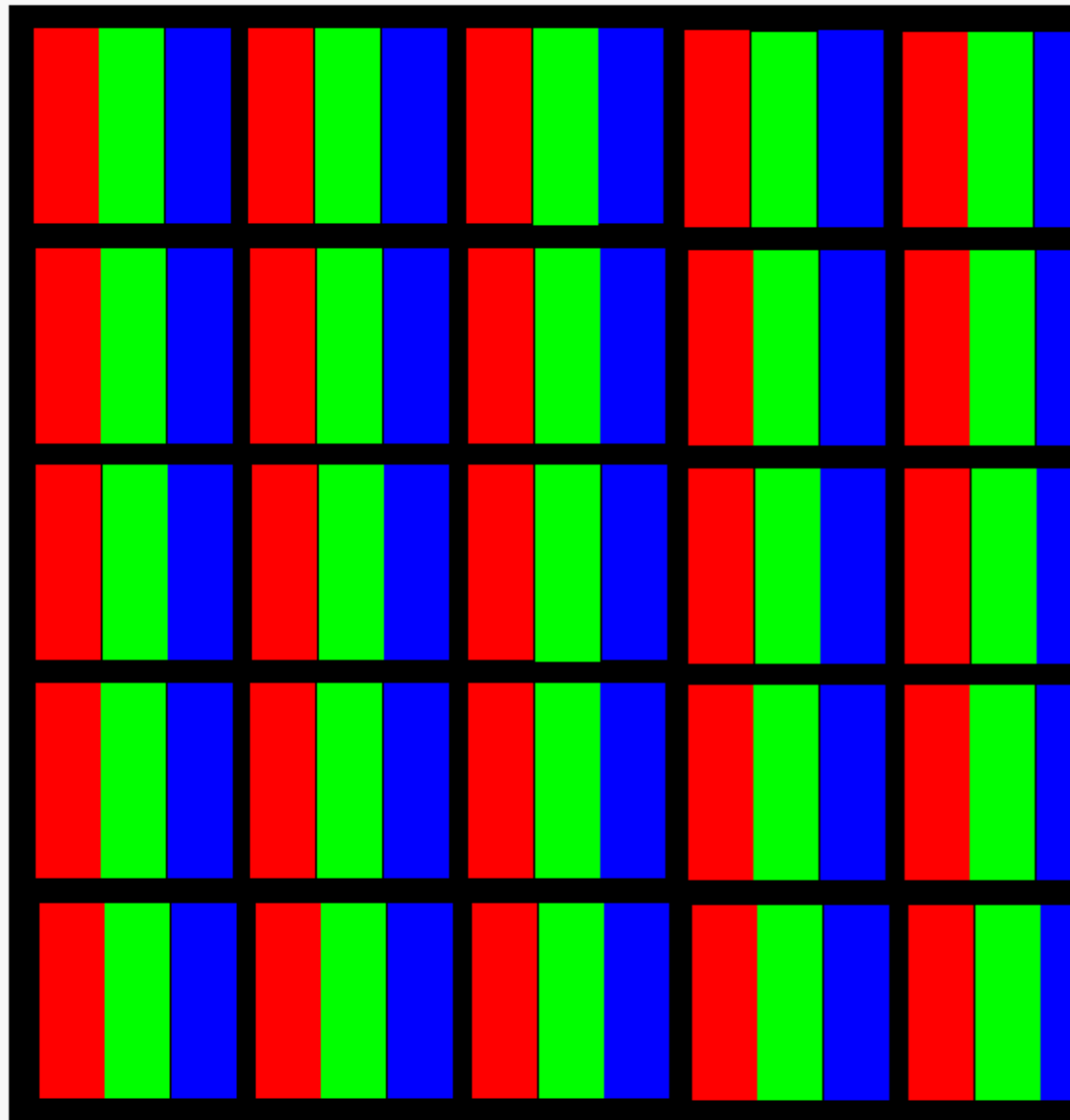
$$u_P = 0.5 \left(\frac{1}{3} \right) + 0 \left(\frac{1}{3} \right) + 1 \left(\frac{1}{3} \right) = 0.5 \quad \lambda_2 \quad v_P = 0 \left(\frac{1}{3} \right) + 1 \left(\frac{1}{3} \right) + 1 \left(\frac{1}{3} \right) \approx 0.67$$

El fragmento del centro recibe la UV $(0.5, 0.67)$, y con ella el fragment shader irá a buscar su color en la textura.

El fragmento

Un fragmento es el paquete de datos que la rasterización genera por cada posición de pantalla cubierta por una primitiva. Es un candidato a píxel.

Highly enlarged schematic re



5 x 5 pixel with RGB subpixels

IONOS

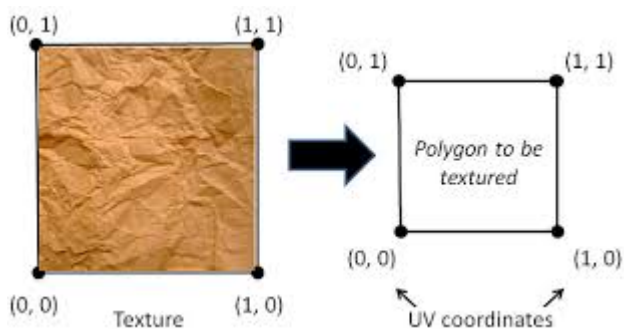
Un fragmento no es un píxel, porque todavía puede descartarse, y un mismo píxel puede recibir varios fragmentos de objetos distintos.

- Posición: en qué celda de la pantalla cae.
- Profundidad: qué tan lejos está de la cámara.
- Normal: la orientación interpolada de la superficie.
- Coordenada UV: el punto de la textura que le toca.
- Color: el tinte heredado de los vértices.
- Posición original: dónde estaba el punto en el mundo.

Fragment Shader

- Entrada: un fragmento con todos sus atributos ya interpolados, más los uniforms del draw call.
- Salida: un color de cuatro componentes, que son rojo, verde, azul y alfa.

Es la segunda etapa programable, y se ejecuta millones de veces por cuadro, así que su costo domina el rendimiento.



- Muestrear la textura en la coordenada UV del fragmento.
- Calcular la iluminación con el modelo de Phong.
- Aplicar sombras, transparencias y reflejos.
- Descartar el fragmento, cuando el material lo deja invisible.

Como ejemplo, un fragmento llega con la coordenada $(u, v) = (0.5, 0.25)$ y el material tiene asignada una textura de 512 por 512 píxeles.

$$x = 0.5 \cdot 512 = 256 \quad y = 0.25 \cdot 512 = 128$$

El color del fragmento sale del texel que está en la fila 128 y la columna 256 de la imagen.

Operaciones por fragmento

Las operaciones por fragmento son la serie de pruebas fijas que decide si el color calculado por el fragment shader llega al framebuffer y cómo se combina con lo que ya estaba ahí.

Depth test

El depth test es la comparación entre la profundidad del fragmento y la que ya estaba guardada en esa posición. El fragmento solo pasa si está más cerca de la cámara.

La profundidad vive en el z-buffer, que es una matriz del tamaño de la pantalla con un valor por píxel. Gracias a él los objetos pueden enviarse en cualquier orden y la oclusión sale correcta.

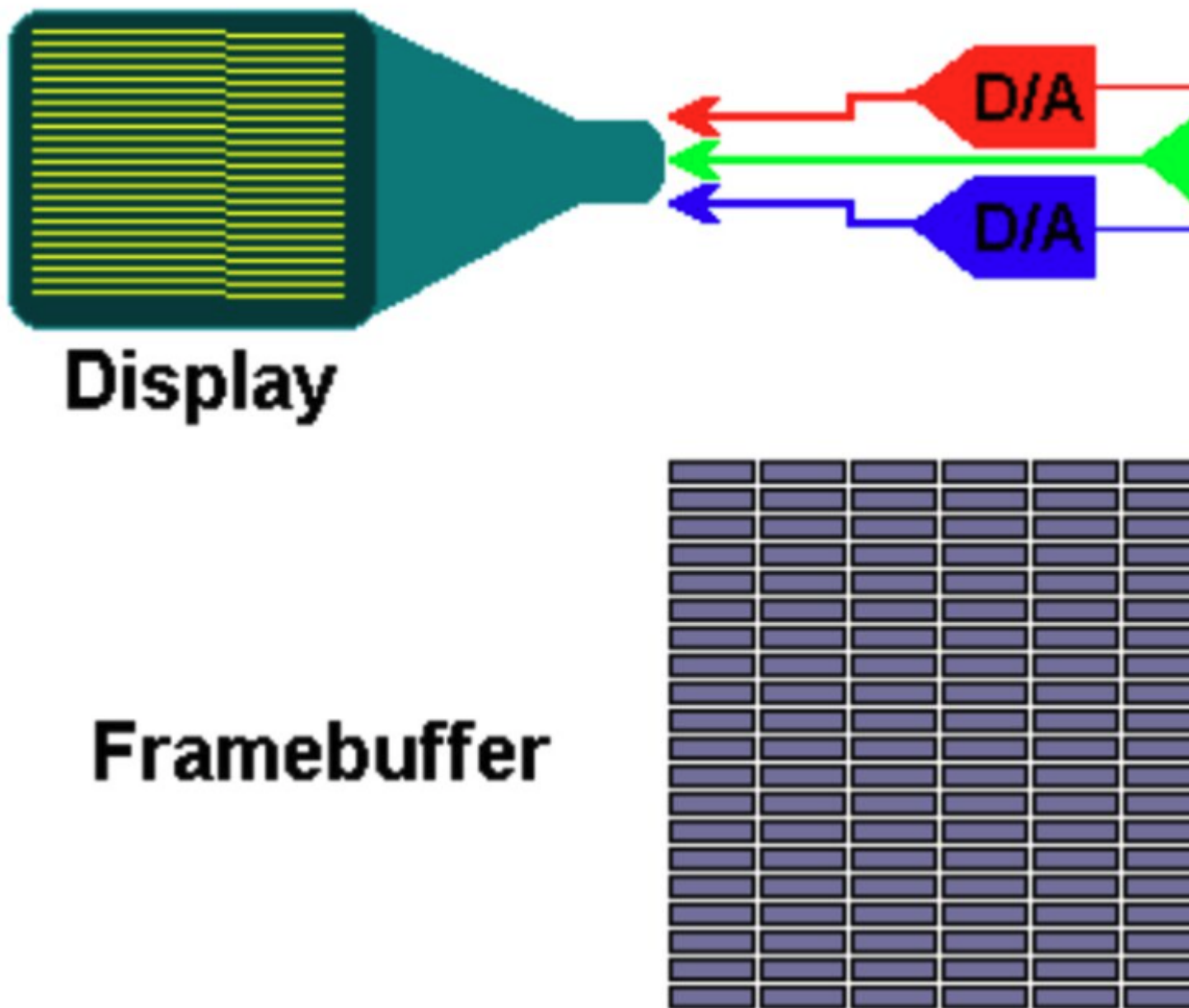
Stencil test y blending

- Stencil test: prueba contra una máscara guardada por píxel, que permite dibujar solo dentro de una región marcada. Se usa en espejos, recortes y siluetas.
- Blending: mezcla del color del fragmento con el que ya ocupaba el píxel, ponderada por el canal alfa. Es lo que produce el vidrio, el humo y las

interfaces translúcidas.

Framebuffer

El framebuffer es la región de memoria que guarda el color de cada píxel de la imagen final, junto con los buffers auxiliares de profundidad y stencil.



Escribir en él es el último paso del pipeline. Cuando el cuadro termina, su contenido se muestra en la pantalla.

Recorrido de un dato

Etapas	Entrada	Salida
Aplicación	Archivo del modelo	Vértices y uniforms
Vertex shader	Un vértice	Vértice en clip space
Primitive assembly	Vértices sueltos	Triángulos
Clipping	Triángulos	Triángulos visibles
Rasterización	Triángulos	Fragmentos interpolados
Fragment shader	Un fragmento	Un color RGBA

Glosario

Término	Definición
Shader	Programa que corre en la GPU dentro de una etapa programable.
Atributo	Dato que acompaña a un vértice y cambia en cada uno.
Uniform	Valor global que la CPU fija y permanece igual durante el draw call.
Varying	Salida del vertex shader que se interpola hacia el fragment shader.
Clip space	Espacio donde queda el vértice tras aplicar la proyección.

Término	Definición
Frustum	Volumen de espacio que la cámara alcanza a ver.
Primitiva	Figura básica que el hardware dibuja, punto, línea o triángulo.
Rasterización	Conversión de una primitiva en los fragmentos que cubre.
Fragmento	Candidato a píxel, con todos sus atributos interpolados.
Texel	Celda individual de una textura.
Z-buffer	Matriz que guarda la profundidad ya escrita en cada píxel.
Framebuffer	Memoria donde se arma la imagen final del cuadro.