# Render

Había una CPU y GPU.

Mandamos cartas por una pipeline y le decimos que tiene que pintar (opengl, vulkan).

Esto funciona pq el DirectX tiene un State, q guarda la configuración. Nos comunicamos con DirectX con mensajes.

Hay Cartas que cambian de estado o que dicen que pinte.

Un dibujo de un pizarrón blanco

Descripción generada automáticamente con confianza media

* **Device: create** textura, create Buffer, etc. estos crean. El device es el propietario de la GPU. **(SET, crea cosas)**
  + **Crea o cambia (créate y set), me parece**
* **Ctx: context** (tubería de mensajes): lo que conecta CPU con GPU. Son comandos que cambian DXState- Esto contiene la configuración del DXState. **(Draw, pinta)**

Buffer para guardar la casa, buffer para guardar el player… (Create) 🡪SET

Y para PINTAR:

* Vertex SHADER
* Pixel shader
* Blend space

Los pixeles tienen bufferes. Yo le digo pinta 3. Draw(3). Coge los bufferes activos, saca 3 y pinta. ¿Qué vertex shader se usa? EL QUE ESTE EN EL ESTADO.

Un dibujo de un pizarrón blanco

Descripción generada automáticamente con confianza media

* Habrá un mensaje para decir que cambie el Pixel Shader 1 por el 2-
* Otro para decir que el VS recibe un input
* Todas las conexiones las establezco yo con cartitas

OpenGL 2,3, hace mucho teníamos un abanico muy grande de cosas que podíamos cambiar. Le decíamos lo que queríamos y el OpenGL tenia mucho trabajo (la GPU), entonces dijeron si queréis que vaya mas rápido teníamos que tener configuraciones. Y se dividieron las etapas

Un dibujo de un pizarrón blanco

Descripción generada automáticamente con confianza media

Entonces los rpogramadores ahora tenían que meter una estructura con la configuración.

* Rasterizacion: tiene configuración 1 por ejemplo
* Blend configuration: aquí le decimos en otra carta, activa la config 3

Y hay otras cartas que se tienen q crear antes (créate blend config 1).

Pipeline es todo el conjunto.

Hay alguna cajita q podemos cambiar parámetros (texturas y no se que). Para pintar una casa o pintar un player hay que cambiar texturas, pero el Pixel shader a lo mejor no.

Todos los programas reciben argumentos

* Pos de la cámara
* Pos del mundo
* Etc.

Estos parámetros del shader tamb habrá que cambiarlos.

DrawInstance: pinta cosas.

Buffer instancias, y yo le digo pinta esto 10 veces. Entonces con una carta le digo que pinte muchas cosas.

Los de la GPU puede hacer mas cosas: **compute pipeline**: no hay VS, ps,etc. Hay compute que es un programa que puede leer lo que quieres.

Modo Compute:

* **Compute shader** (tiene parámetros de textura y bufferes que puede leer). Y yo le digo ejecuta. No vamos a entrar. Esto puede hacer muchas cosas que no son pintado. Test de Culling. Y me haces un buffer de matrices que han pasado el test.

Compute: orientado mas a datos (es para el futuro)

Lo primero es mas para pintar.

Carpeta de render del Motor

Texto

Descripción generada automáticamente

Render.h:

* **Device**: representa la tarjeta grafica (le pedimos cosas para usar muchas veces). **Crea objetos**
* **Devicectx**: el canal para perdirle cosas (le pedimos cosas si creo y destruyo). **Para cambiar estados**
* **View**: en sql para ver los datos de una manera. Aquí igual. Yo creo un buffer (ej una textura). La textura la puedo usar para varias cosas. En un Shader como input . Para leerla. Para eso necesito un shader view. Para poder usar algo en un shader necesito un shader view.
* **RenderTargetView**: me permite usar la textura como un RenderTarget.

Sirve pq la textura tendrá 4 canales RGBA (8 bits) y para usarlo como RenderTarget le tengo que decir que cuando escriba 4 floats, me hace la traducción. Es una relación 1 a 1 normalmente.

En el State de DirectX, hay un currentRenderTargetView[4]. Yo puedo pintar mas de un render target a la vez. (para pintar posición, albedo, y 2 cosas mas). Ahora directX creo q permite hasta 8.

Entonces yo escribo en el currentRenderTargView y estoy pintando en el recurso

Z-buffer para saber la profundidad. DEPTHSTENCILview. Entonces ese, lo voy a poder usar como textura.

Entonces, yo voy a poder interpretar esa textura como distintas cosas.

**Hay unos resources (bufefers y texturas) y con la palabra view es para usar esos recursos en alguna parte del State.** Voy a usar esos resources en el momento del pintado.

**Compute shader:** puede leer (view) o escribir ()

Una vez creo una textura, creo el DepthStencilView

Texto

Descripción generada automáticamente

Stencil (de 0 a 255).

Luego, la textura tamb quiero usarla como Shader

Texto

Descripción generada automáticamente

Hemos creado device, renderTarget ¿Qué otras cosas?

Texto

Descripción generada automáticamente

Para decir el Z configuration.

DEFAULT, es lo que hay

Texto

Descripción generada automáticamente

* Default: lo q esta detrás no se pinta
* DISABLE\_ALL: no hay Z buffer
* Testea y no se escribe
* …
* Este nombre se lo inventa el profe

Texto

Descripción generada automáticamente

Rasterización

* Se pinta lo que se ve
* Lo contrario
* No hay culling (se ve todo). Si tengo un cubo

Un dibujo de un pizarrón blanco

Descripción generada automáticamente con confianza baja hay caras que no se ven. Esto viene determinado por el orden de las caras. Y podemos detectar qué caras me están mirando (esas que se pinten. Aquí se pintan todos)

* Shadows
* Wireframe: solo aristas.

Texto

Descripción generada automáticamente

Ccomo se mezcla la salida.

* DEFAULT
* No hay blend
* Falknflkan

En la Pipeline de direct12 State, están los VSConstansts.

Texto

Descripción generada automáticamente con confianza media

El 1 es pq la API me deja ponerle 3, yo le paso la 1

* Forward: lo que veo
* Deferred

Para pintar una esfera y un shader, necesitamos Modules.

Pizarrón con letras de colores

Descripción generada automáticamente con confianza media

JOHN copia de solid.fx (bin/data/shaders) y hemos cambiado el color a una constante (lila) para identificarlo

Luego vamos al pipelines.json

Texto

Descripción generada automáticamente

Le ponemos el john y le ponemos el john.fx

Texto

Descripción generada automáticamente

Le podemos cambiar el Color en el john.fx y en la misma ejecución cambia.

Imagen que contiene Texto

Descripción generada automáticamente

Podemos cambiar la pipelin desde el IMgui (y le pongo el wireframe por ejemplo).

El verde hacia arriba 0,1,0

Tengo una pipeline pero no esta asociada a una entidad. No puedo rotarla. Yo puedo hacer que mi modulo tenga una transform (sin tener una entidad asociada). Vamos a ello

Captura de pantalla de computadora

Descripción generada automáticamente

Y lo pintamos en el menú del .cpp

Texto

Descripción generada automáticamente

Captura de pantalla de un celular

Descripción generada automáticamente

Ahora puedo mover y girar la tetera

Diagrama

Descripción generada automáticamenteImagen que contiene papalote, exterior, colorido, vuelo

Descripción generada automáticamente

Problema: las normales no cambian, lo suyo es que lo verde vaya hacia arriba SIEMPRE. Hacia arriba de la tetera. Transformamos la normal \* world

Interfaz de usuario gráfica, Texto

Descripción generada automáticamente

En vez de ser un input.Normal, la multiplicamos por el object\_world