# Repaso de matrices

## Objeto

Tenemos un obj en coordenadas locales.

Una bounding box alrededor del 0.

## Matriz Local to World (object space to world space)

Hay una matriz **World Object** (pasa de local a World coordinates)

Transformación fácil porq es una matriz de 4x4

Un dibujo de un pizarrón blanco

Descripción generada automáticamente con confianza baja

x,y,z,1 🡪 1x4 \* 4x4

Ejemplo: nos pasa el modelo de los artitas del 0 a la montaña

## Camera View

Pasa de World a View Coords. El origen ahora está en la cámara. Todo está en relación a la cámara.

Una caricatura de un pizarrón blanco

Descripción generada automáticamente con confianza mediaDibujo de un pizarrón blanco

Descripción generada automáticamente con confianza baja

Ojo: el Z aquí va del revés.

Está pasando de World a Local(local de la cámara), por lo que habrá que usar una inversa.

Para construir la matriz se suele usar una función LookAt.

Igual que pasa de World a su espacio local, si saco la inversa de esta me servirá para pasar al revés.

## Camera Projection

Las coordenadas están en camera projective space.

Prepara los datos porq ahora va a venir el HW y tenemos que dividir por w.

Esto se hace por tema de proporciones. Screen y = y / z

Imagen que contiene persona, objeto, interior, sostener

Descripción generada automáticamente

Ahora viene el HW y hace 1/w y pasamos a espacio homogéneo.

La matriz proyectiva era así:

Un dibujo de una persona

Descripción generada automáticamente con confianza media

El valor x en projection space = x view space \* cotangente(fov/2)

Yproyective = xview space \* cotang (fov/2)

Zproyective = Zviewspace \* Alpha + Beta (va entre 0 y 1, para el Zbuffer)

Wproyective = -Zviewspace (el menos es por lo del cambio de z), pero lo importante es la Z (distancia a la camara).

## Space Homogeneous

El 0 esta en el centro de la pantalla. Y nosotros queremos números entre -1 … 1

## Viewport Transform

Aquí ya salen coords para la pantalla (0…1920) y (0…1080)

Hasta aquí es la cámara, que salga en pantalla lo que ve la cámara. Ahora vamos con las luces:

Cuando pinto lo de la cámara también tengo una luz. Para que se vea el sombreado necesito un render completo desde la luz.

Un dibujo de un pizarrón blanco

Descripción generada automáticamente con confianza media

La luz actúa también como una cámara con un fov, es como un proyector (hay omnis tamb pero de momento vemos esta).

Cuando queremos pintar un punto de la sombra, si estoy en el mundo, en el Pixel shader que se que soy la World Pos querré saber en que parte de la pantalla estoy respecto a la luz. Paso por el view proyection de la luz.

Con la WorldPos \* ViewProyectionLuz

Deferred

Para todos los objetos de mi escena pinta en un Gbuffer (son varias texturas):

* Albedo / metalness
* Normal (w glosiness)

Pizarrón verde con letras blancas

Descripción generada automáticamente con confianza media

La normal se hace un dot producto entre los vectores y me da una distancia (127 en este caso, lo que significa que el punto tendrá que avanzar 127 unidades)

* Z lineal
* Z proyectivo (del ZBuffer), entre 0 y 1. La distribución no era lineal.

Para cada Luz, pinta luz: buscamos la zona de la pantalla donde afecta esa luz, reconstruir esos pixeles. Y tenemos una función del GBuffer para tocar nosotros lo que queríamos.

Texto

Descripción generada automáticamente

Para hacer las sombras necesitamos un render desde la luz.

getShadowFactor nos dice si está o no en sombra a partir de una posición del mundo