Universidad de Huelva

Tema 7

Dibujando en el espacio

Índice

- El proceso de renderizado se lanza al ejecutar los comandos *vkCmdDraw…()*. A partir de aquí se buscan los vértices necesarios para generar las primitivas y se ejecuta el VertexShader sobre los atributos de cada uno de estos vértices.
- El comando más sencillo es *vkCmdDraw()*. En este comando se especifica el número de vértices a utilizar y el número de instancias (repeticiones del proceso). Los atributos de los vértices se leen de forma consecutiva desde sus buffers.

```
void vkCmdDraw (
        VkCommandBuffer         commandBuffer,
        uint32_t                vertexCount,
        uint32_t                instanceCount,
        uint32_t                firstVertex,
        uint32_t                firstInstance);
```

- Para formar primitivas leyendo vértices de forma consecutiva hay que repetir muchos vértices. Para evitar esto es mejor utilizar una lista de índices y realizar un dibujo indexado.
- Para utilizar índices se usa la función *vkCmdDrawIndexed()*. En este caso se especifica el número de índices a utilizar y el número de instancias (repeticiones del proceso). Se puede incluir un desplazamiento a sumar siempre a los índices.

```
void vkCmdDrawIndexed (
    VkCommandBuffer     commandBuffer,
    uint32_t            indexCount,
    uint32_t            instanceCount,
    uint32_t            firstIndex,
    int32_t             vertexOffset,
    uint32_t            firstInstance);
```

- Vulkan permite también leer los datos del dibujo desde un buffer. Este buffer debe contener una estructura de tipo *VkDrawIndirectCommand*.
- La función *vkCmdDrawIndirect()* lanza un proceso de dibujo en el que se especifica un buffer y un desplazamiento para poder buscar la estructura *VkDrawIndirectCommand* y leer de ella los datos de configuración.

```
typedef struct VkDrawIndirectCommand {
    uint32_t    vertexCount;
    uint32_t    instanceCount;
    uint32_t    firstVertex;
    uint32_t    firstInstance;
} VkDrawIndirectCommand;
```

- De la misma forma se puede lanzar un dibujo indexado tomando la configuración de un buffer. En este caso se utiliza la función *vkCmdDrawIndexedIndirect()* que indique el buffer de lectura y la información a leer del buffer debe ser una estructura de tipo *VkDrawIndexedIndirectCommand*.

```
typedef struct VkDrawIndexedIndirectCommand {
    uint32_t   indexCount;
    uint32_t   instanceCount;
    uint32_t   firstIndex;
    int32_t    vertexOffset;
    uint32_t   firstInstance;
} VkDrawIndexedIndirectCommand;
```

- Las posiciones de los vértices se expresan en coordenadas (x,y,z,w).
- La última coordenada suele tomar el valor 1.0 y permite realizar transformaciones de desplazamiento (usando la cuarta columna de las matrices de transformación) y transformaciones de tamaño (en las proyecciones en perspectiva).
- Para realizar transformaciones en las coordenadas se utilizan matrices de transformación.

$$\begin{bmatrix} x_e \\ y_e \\ z_e \\ w_e \end{bmatrix} = \begin{bmatrix} a_{00} & a_{01} & a_{02} & a_{03} \\ a_{10} & a_{11} & a_{12} & a_{13} \\ a_{20} & a_{21} & a_{22} & a_{23} \\ a_{30} & a_{31} & a_{32} & a_{33} \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ w_0 \end{bmatrix}$$

- A la entrada de la etapa de rasterización, la posición de los vértices se transforma a coordenadas homogéneas dividiendo por la coordenada w:

$$\begin{bmatrix} x_n \\ y_n \\ z_n \end{bmatrix} = \begin{bmatrix} x_c / w_c \\ y_c / w_c \\ z_c / w_c \end{bmatrix}$$

- Las coordenadas homogéneas permiten definir el trozo del espacio que se utilizará para generar el dibujo. Este recorte espacial se conoce como *clipping volume*, por lo que las coordenadas homogeneas se conocen también como coordenadas Clip.

- El recorte consiste en dibujar solo aquello que se ecuentra en coordenadas homogéneas entre -1 y 1.
- En realidad, el recorte es siempre entre -1 y 1 en las coordenadas X e Y, pero el tipo de recorte de la coordenada Z es configurable.
- Antes de aplicar el recorte se realiza también un proceso de selección de primitivas (*Culling*). Este proceso permite desechar primitivas y también es configurable.
- Cuando no se utilizan los shaders de teselado ni de geometría la entrada del proceso de rasterización corresponde a la salida del VertexShader. Por tanto, en estos casos la posición de los vértices a la salida del VertexShader debe estar descrita en coordenadas homogeneas.

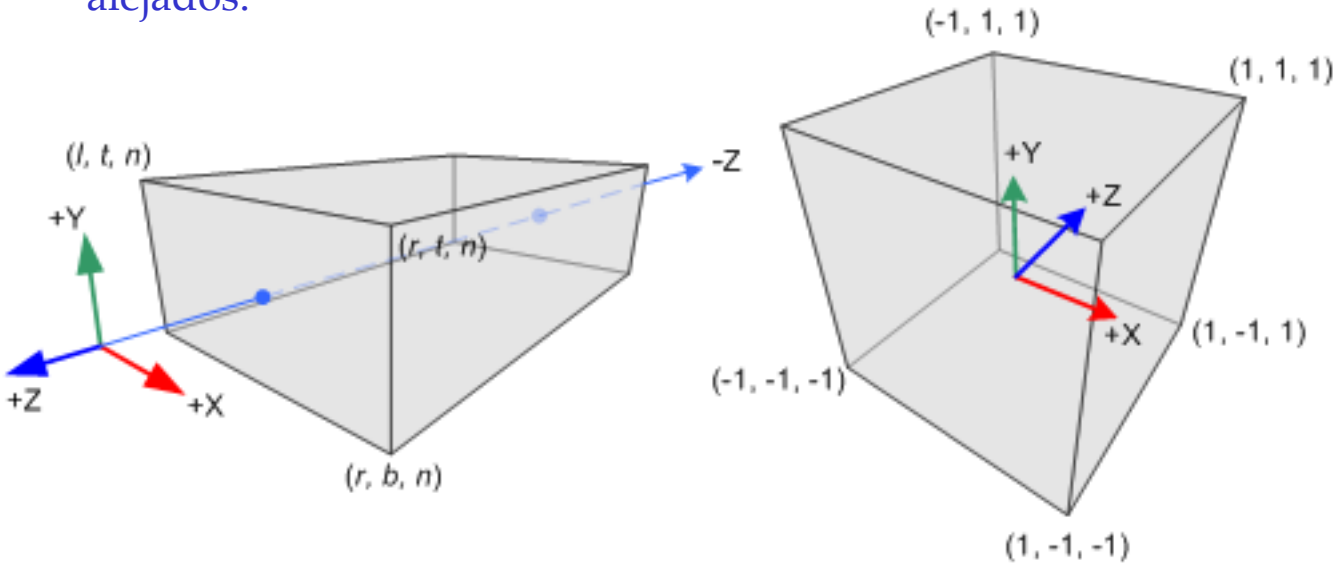
- Entradas predefinidas del VertexShader:
  - ***gl_VertexIndex***, contiene el índice del vértice que se está tratando en la ejecución del VertexShader.
  - ***gl_InstanceIndex***, contiene el número de instancia de la primitiva a la que pertenece el vértice.

    La variable de entrada *gl_InstanceIndex* contiene el índice de la instancia que se está ejecutando en el VertexShader. De esta forma se pueden realizar transformaciones diferentes a los atributos de los vértices en función del número de instancia. Por ejemplo, se podrían ejecutar dos instancias y en la segunda transformar las primitivas para generar una imagen especular.

- Salidas predefinidas del VertexShader:
  - ***gl_Position***, debe almacenar la posición del vértice en coordenadas homogéneas.
  - ***gl_PointSize***, contiene el tamaño en pixels del punto a dibujar (si la topología de las primitivas fuera POINT_LIST o la propiedad polygonMode del estado de rasterización tuviera el valor VK_POLIGON_MODE_POINT). Su uso es opcional.
  - ***gl_ClipDistance***, es un array que debe almacenar la distancia a los planos clip. Esta salida se utiliza en caso de querer personalizar el proceso de clipping. Su uso es opcional.
  - ***gl_CullDistance***, es un array que debe almacenar la distancia a planos cull. Se trata de una característica incluida en la versión 4.5 que permite personalizar la etapa de culling. Su uso es opcional.

- Aunque la salida del VertexShader debe estar descrita en coordenadas homogéneas, lo normal es que la posición de los vértices (introducida como uno de los atributos de entrada en el VertexShader) se describa en el sistema de coordenadas del modelo. Por tanto, el programa del VertexShader debe incorporar una transformación entre un sistema de coordenadas y otro.
- Esta transformación debe convertir el volumen visible del modelo (es decir, la zona del modelo que aparecerá en la imagen) en el volumen clip (es decir, un volumen normalizado entre -1 y 1).
- Esta transformación se conoce como ***Proyección***. Se suelen utilizar dos formas: proyección ortográfica o proyección en perspectiva.

- Proyección ortográfica:
  - Consiste en transformar un volumen de forma rectangular en el volumen Clip. Esto supone un cambio de escala en las diferentes coordenadas (X,Y,Z) para adaptarse al sistema de coordenadas homogéneo. La transformación es muy sencilla, pero provoca que los objetos se vean del mismo tamaño tanto si están cerca como si están alejados.



- Proyección ortográfica
  - La matriz de proyección en este caso es la siguiente:

$$P = \begin{bmatrix} \frac{2}{r-l} & 0 & 0 & -\frac{r+l}{r-l} \\ 0 & \frac{2}{t-b} & 0 & -\frac{t+b}{t-b} \\ 0 & 0 & \frac{-2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

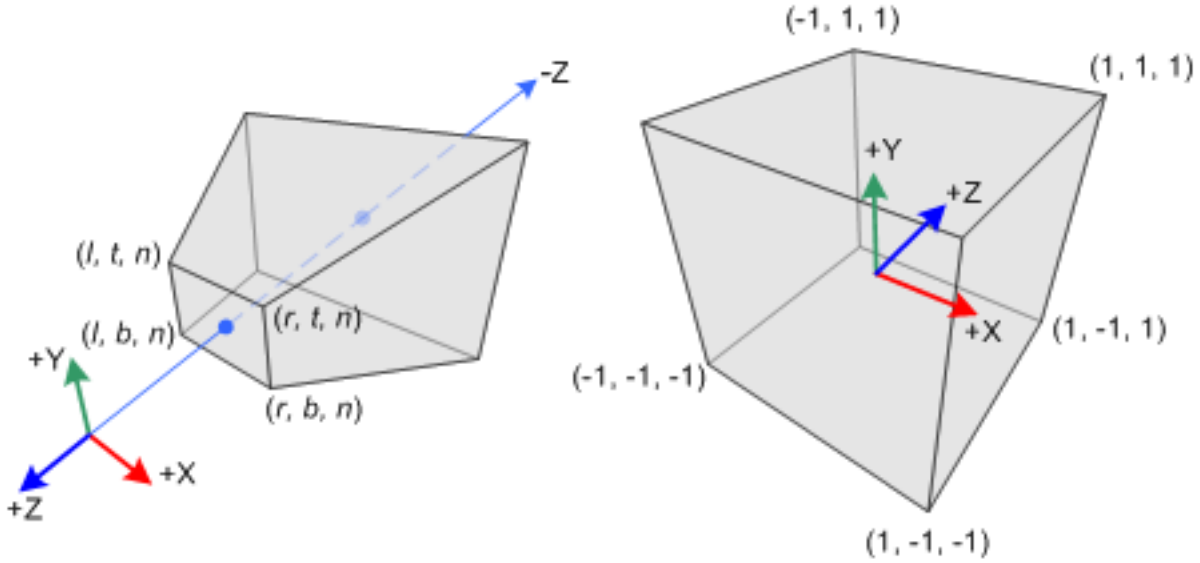
- Proyección ortográfica
  - Si el clipping volume es simétrico (r = -l y t = -b) la matriz queda

$$P = \begin{bmatrix} \frac{1}{r} & 0 & 0 & 0 \\ 0 & \frac{1}{t} & 0 & 0 \\ 0 & 0 & \frac{-2}{f-n} & -\frac{f+n}{f-n} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

  - Se puede generar la matriz de proyección utilizando la función de la librería GLM:

glm::ortho(left, right, bottom, top, near, far)

- Proyección en perspectiva:
  - Consiste en transformar el espacio que se ve desde el observador con un cierto ángulo de visión (*fov-field of vision*) en el volumen Clip. Esta transformación provoca que los objetos más cercanos se vean de mayor tamaño que los objetos más lejanos. La forma del volumen transformado es un tronco de pirámide (en inglés, *frustum*).



- Proyección en perspectiva
  - La matriz de proyección que genera la transformación en perspectiva es la siguiente:

$$P = \begin{bmatrix} \frac{2 \cdot n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2 \cdot n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & \frac{-2 \cdot f \cdot n}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

- Proyección en perspectiva
  - Si el volumen Clip es simétrico (r = -l y t = -b) la matriz queda

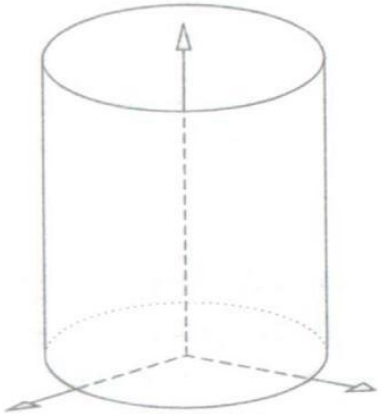
$$P = \begin{bmatrix} \frac{n}{r} & 0 & 0 & 0 \\ 0 & \frac{n}{t} & 0 & 0 \\ 0 & 0 & -\frac{f+n}{f-n} & \frac{-2 \cdot f \cdot n}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix}$$

  - Esta matriz se puede generar con funciones de la librería GLM:

glm::frustum(left, right, bottom, top, near, far)

glm::perspective(fovy,aspect,near,far)

- Generalmente los atributos de los vértices de los objetos se expresan en un sistema de coordendas propio del objeto (sistema de coordenadas local). Este sistema de coordenadas suele tener su origen en un punto característico del objeto (por ejemplo, en su centro) y sus ejes de coordenadas suelen ser direcciones características del objeto.



- Para realizar la proyección, las posiciones de los vértices deben estar descritas en el sistema de coordenadas del observador. Por tanto, es necesario realizar una transformación entre el sistema de coordenadas local y el sistema de coordenadas del observador.

- Normalmente esta trasformación se realiza en dos pasos. En primer lugar las posiciones de todos los objetos se transforman a un sistema de coordenadas fijo del modelo. De esta forma las posiciones de todos los objetos quedan expresadas en un mismo sistema de coordenadas. Para transformar las coordenadas locales en coordenadas de modelo se utiliza una matriz de transformación conocida como *Model*.
- El segundo paso es transformar estas coordenadas de modelo al sistema de coordenadas del observador. Para esto se utiliza una matriz de transformación conocida como *View*.
- La transformación completa se conoce como *Model-View-Projection*.

- Si queremos que realizar una traslación, la matriz de transformación debe ser

$$M_{Tr} = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

De esta forma

$$v_e = M_{Tr} \cdot v_0 = \begin{bmatrix} 1 & 0 & 0 & a \\ 0 & 1 & 0 & b \\ 0 & 0 & 1 & c \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{bmatrix} = \begin{bmatrix} x_0 + a \\ y_0 + b \\ z_0 + c \\ 1 \end{bmatrix}$$

- Si queremos que realizar una transformación de escalado sobre el modelo, la matriz debe ser

$$M_{Sc} = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

De esta forma

$$v_e = M_{Sc} \cdot v_0 = \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{bmatrix} = \begin{bmatrix} a \cdot x_0 \\ b \cdot y_0 \\ c \cdot z_0 \\ 1 \end{bmatrix}$$

- Las rotaciones afectan a las tres primeras filas y columnas de la matriz. Por ejemplo, para girar un ángulo α sobre el eje Z la matriz debe ser

$$M_{Rot} = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- De esta forma

$$v_e = M_{Rot} \cdot v_0 = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) & 0 & 0 \\ \sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x_0 \\ y_0 \\ z_0 \\ 1 \end{bmatrix} = \begin{bmatrix} x_0 \cdot \cos(\alpha) - y_0 \cdot \sin(\alpha) \\ x_0 \cdot \sin(\alpha) + y_0 \cdot \cos(\alpha) \\ z_0 \\ 1 \end{bmatrix}$$

- La matriz de rotación sobre un eje arbitrario (a_x, a_y, a_z) es la siguiente:

$$c_\theta = \cos(\theta) \quad s_\theta = \sin(\theta)$$

$$M_{Ra}(\theta) = \begin{bmatrix} a_x^2 + c_\theta(1-a_x^2) & a_x a_y(1-c_\theta) - a_z s_\theta & a_x a_z(1-c_\theta) + a_y s_\theta & 0 \\ a_x a_y(1-c_\theta) + a_z s_\theta & a_y^2 + c_\theta(1-a_y^2) & a_y a_z(1-c_\theta) - a_x s_\theta & 0 \\ a_x a_z(1-c_\theta) - a_y s_\theta & a_y a_z(1-c_\theta) + a_x s_\theta & a_z^2 + c_\theta(1-a_z^2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- Se pueden generar facilmente las matrices de transformación por medio de funciones de la biblioteca GLM.

glm::translate(glm::mat4 M, glm::vec3 disp)

glm::scale(glm::mat4 M, glm::vec3 scale)

glm::rotate(glm::mat4 M, float angle, glm::vec3 axis)

- Estas funciones generan una modificación de la matriz *M* multiplicándola por la derecha por la matriz de desplazamiento, escalado o rotación correspondiente.

$$M_{new} = M_{old} \cdot M_{Tr}$$

- Ejemplo de VertexShader que contiene la transformación Model-View-Projection

```
#version 450
#extension GL_ARB_separate_shader_objects : enable

layout(binding = 0) uniform UniformBufferObject {
    mat4 MVP;
} ubo;

layout(location = 0) in vec3 inPosition;

void main()
{
    gl_Position = ubo.MVP * vec4(inPosition, 1.0);
}
```