Tarea8

Cube Texture Sky – Reflection From Sky

Instrucciones:

- Leer:

  https://docs.gameloft.org/3d-training/#Cubic_Mapping_and_Sky_Domes

Cargar la "cube texture":

- En la clase **Texture** agregar 2 funciones:
  - InitCubeTexture()
  - ExtraxtFace()

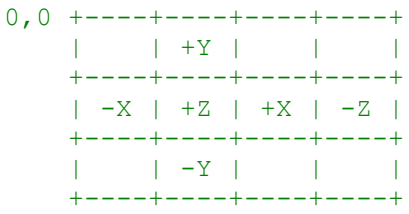
    (Este método se llamara desde **InitCubeTexture()** para extraer cada cara)
- En la función **InitCubeTexture()**:
  - Generar el recurso OpenGL (ejemplo: la textura cubo, cargarla a la VRAM) y hacerle un bind;
  - Usar de nuevo **LoadTGA** desde Utilities, función que cargara tu textura en un char *bufferTGA;
  - Ahora necesitaras extraer cada cara del cubo (son 6 caras) en un buffer y ese buffer se pasara **a glTextImage2D()** por cada cara.
  - Para extraer cada cara, se llamara a la función **ExtractFace()** con dos offsets:
    - Un offset horizontal (este dará el desplazamiento desde la posición 0,0 de tu textura en el eje horizontal).
    - Un offset Vertical (Similar al de arriba)
- Generar mipmaps (se necesitaran para la reflección).
- En **ExtractFace()**:
  - Determinar el comienzo de la cara dentro de bufferTGA y aplicar los offsets (necesitaras tambien el width/height y el bpp de la textura).

- Usar **memcpy_s** para copiar cada línea de la cara del cubo.

- Desde el paquete de recursos del proyecto, con algún programa de edición de imágenes modifica la textura **envMap.tga** volteándola verticalmente (quedara invertida verticalmente), eso es porque en OpenGL ES 2.0 hay un bug y 0,0 no tiene la posición en pantalla que debería de tener comúnmente.

- Se calcularan los offsets de la cara del cubo asumiendo que la posición 0,0 es la esquina superior izquierda.

  Ejemplo:



  +Z tiene los offsets:

  - Offset horizontal: widht/4
  - Offset vertical: height/3

  Tip: Todas las caras son cuadradas, por lo tanto widht/4 = height/3

"shader SkyBox":

- Usar **textureCube()** on el fragment shader para hacer el sampling de la textura cubo.

  "Necesitaras una dirección para tu sampling. Porque el archivo de geometría del cielo tiene un pivote en el centro del cubo, puedes usar directamente la posición en "local space" para hacer el sampling en el fragment shader"

"shader Reflection from Sky":

- Leer:

  https://docs.gameloft.org/3d-training/#Environment_Reflections

- Usar el modelo **Bila.nfg** para la reflexión (este modelo es una esfera).

Guía:

Para el reflejo:

- Todos los cálculos de la luz (el reflejo del cielo es un tipo de iluminación, se verá iluminación en siguientes tareas) se harán en el world space, por ser más conveniente (algunas de tus variables ya están expresadas en world space; y es más fácil ver los calculos);
- Debes poner atención al espacio de coordenadas en el que cada variable esta, por ejemplo, la posición de la cámara está en world space, la posición de tus vértices están en local space (por lo tanto, serán necesarias algunas transformaciones cuando tengas que hacer algún calculo entre los dos).
- Puedes calcular el vector reflexión en el vertex shader (se usara este vector reflexión para el sampling de tu cubo) , porque las direcciones se interpolan correctamente durante la interpolación lineal del pipeline. Además es más óptimo hacerlo en el vertex shader, porque si se hace en el fragment shader, tendremos más fragmentos que vértices y por lo tanto es más consumo de recursos de cómputo.
- Cuando se calcule el vector reflexión (usando ***toEye*** y la ***nolmal***), debes poner atención en que **toEeye** debe sufrir un cambio de signo al entrar al vértice (de acuerdo a la formula, **toEye** sale del vértice, por lo tanto se tiene que poner el signo “menos” en la función **reflect()**)

Para el cubo:

AVISO - CODIGO PARA SACAR CARA DEL CUBO DESDE EL BUFFER:

Esta parte debe ser omitida de preferencia, ya que pone a prueba tu lógica y habilidad para la solución de problemas complejos, si crees que es un problema demasiado complejo para ti, usa el siguiente código, estúdialo y compréndelo.

```
char * Texture::ExtractFace(char * bufferTGA, char * temp, int width, int
bpp, int offsetX, int offsetY)
{
      int w = width/4;
    int startRead = offsetY * width * bpp/8 + offsetX * bpp/8;

    for (int i = 0; i < w; i++)
    {
        memcpy_s(temp + i * w * bpp/8, w*w*bpp/8,bufferTGA + startRead,w *
bpp/8);
        startRead += width * bpp/8;
    }

    return temp;
}
```

```
bool Texture::InitCubeTexture()

{



int width, height, bpp;


      glGenTextures(1, &textureInfo.m_hTexture);
      glBindTexture(GL_TEXTURE_CUBE_MAP, textureInfo.m_hTexture);


    char * bufferTGA = LoadTGA(textureInfo.path, &width, &height, &bpp);



    int w = width / 4;
    int h = height / 3;



char* p[6];

      for(int i=0; i<6; i++)
            p[i] = new char[w*w*bpp/8];

      p[0]=ExtractFace(bufferTGA, p[0], width, bpp, 2 * w, w);
      p[1]=ExtractFace(bufferTGA, p[1], width, bpp, 0, w);
      p[2]=ExtractFace(bufferTGA, p[2], width, bpp, w, 0);
      p[3]=ExtractFace(bufferTGA, p[3], width, bpp, w, 2 * w);
```

```
      p[4]=ExtractFace(bufferTGA, p[4], width, bpp, w, w);
      p[5]=ExtractFace(bufferTGA, p[5], width, bpp, 3 * w, w);

       SafeDeleteArray(bufferTGA);

for(int i=0; i<6; i++)
      {

            if(bpp == 24)
            {
                 glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X+i, 0, GL_RGB,
w, h, 0, GL_RGB,GL_UNSIGNED_BYTE, p[i]);
            }
            else
            {
                  glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X+i, 0, GL_RGBA,
w, h, 0, GL_RGBA,GL_UNSIGNED_BYTE, p[i]);
            }
            SafeDelete(p[i]);

      }

      glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAG_FILTER, GL_LINEAR);

.

.

.

.

}
```

Resultado: