Tarea4

Cargar archivo .nfg

Instrucciones:

- Crear clase "**Model**", en la cual servirá solo para leer la información del modelo (información contenida en los .nfg), mandarla a la GPU y guardar los handle para los vértices y los índices (los handle son los identificadores que se les asigna el valor del buffer cuando se crea un nuevo buffer en la GPU, ejem. `glGenBuffers(1, &m_hVertexBuffer)`).
- Usar buffer objects (vertex buffer object y index buffer object).
- Mantener en la clase solo los atributos necesarios; por ejemplo necesitamos conservar los handles de los buffer objects, pero no los heap buffers que se usan para cargar la información que mandamos al GPU, esa información solo se mantendrá temporalmente.
- Para leer la información de geometría de los archivos utilizar "fscanf_s"
- Dibujar el modelo.

Guía:

- Copia las carpetas que están en NewResourcesPacket\ en donde está tu código para tener una ruta de acceso más cómoda.
- Accesa a la carpeta Models y abre el archivo Woman1.nfg con el notepad, te darás cuenta que viene información sobre los vértices y los índices, la información de los vértices es:

  pos → Posición del vértice, consta de 3 valores flotantes
  norm → Normal, consta de 3 valores flotantes
  binorm → Bitangente o Binormal, consta de 3 valores flotantes
  tgt → Tangente, consta de 3 valores flotantes
  uv → Coordenadas de textura, consta de 2 valores flotantes

Los nuevos atributos los declararemos en la estructura Vertex para poderlos capturar más facil, en Vertex.h:

```
struct Vertex //STEP2
{
Vector4 color;
Vector3 pos;
Vector3 normal;
Vector3 binormal;
Vector3 tgt;
Vector2     uvs;
};
```

*para los índices en la clase Model declararemos un Array de enteros, ya que los índices son consecutivos y no están dentro de los atributos de los vertex.

- En la clase **Model**, donde se mantendrá toda la información será algo parecido a esto:

class Model

{

Model(); // you need to keep always constructors very simple (only code like variable = 0, pointer = NULL).

// That's because on some platforms you can't access the call stack before main(), so, if you have global objects (constructor will be called before main()), it's a nightmare to identify which of those global objects has problems.

~Model(); //Don't forget to put on NULL the pointers after "delete pointer"

void InitModel(char * filename); // You read .nfg file using fscanf_s(). You can "jump" over some values which are not relevant for you. Example:

```
// fscanf_s( pFile, "   %*d. pos:[%f, %f, %f]; norm:[%f, %f, %f]; binorm:[%f, %f, %f]; tgt:[%f, %f, %f]; uv:[%f, %f];",&verticesData.pos.x, &verticesData.pos.y, & verticesData.pos.z,…….);

// Please notice that "*" allowed us to jump over a value (it is read from your file, but not assigned to a variable".

// You create your vertex buffer object and your index buffer object (glGenBuffers() etc.)
```

glGenBuffers(1, &m_hVertexBuffer);
glBindBuffer(GL_ARRAY_BUFFER, m_hVertexBuffer);

glBufferData(GL_ARRAY_BUFFER, maxVertices * sizeof(Vertex), bufferVertex, GL_STATIC_DRAW);

glGenBuffers(1, &m_hIndexBuffer);
glBindBuffer(GL_ARRAY_BUFFER, m_hIndexBuffer);
glBufferData(GL_ARRAY_BUFFER, (m_noIndices)*sizeof(int), bufferIndices, GL_STATIC_DRAW);

GLuint m_hVertexBuffer, m_hIndexBuffer; // you keep handles to your buffer objects

unsigned int m_noIndices; // you need it in glDrawElements().

unsigned int m_ID; // You don't use for the moment this data member, but when you will write the engine, here it will be kept the ID of the model from Resource Manager file.

};

- Si quisiéramos leer el número entero de un enunciado con fscanf_s seria de la siguiente forma:

  "1. Numero 69"

  fscanf_s(pFile, "%*d. %*s %d\n",&var);

  Donde "pFile" es de tipo FILE y contiene el buffer del archivo, y "var" es una variable entera. El "*" nos sirve para omitir información.

- Una vez leída la información de los vértices y de los índices mandaremos esa información a la VRam:

```
	//Generamos  el handle para el vertex buffer
	glGenBuffers(1, &m_hVertexBuffer);
	//Hacemos bind al buffer para decir que lo vamos a utilizar
	glBindBuffer(GL_ARRAY_BUFFER, m_hVertexBuffer);
	//Mandamos la informacion contenida en un array de tipo Vertex
	glBufferData(GL_ARRAY_BUFFER, maxVertices * sizeof(Vertex), bufferVertex, GL_STATIC_DRAW);

	//Generamos  el handle para el index buffer
	glGenBuffers(1, &m_hIndexBuffer);
	//Hacemos bind al buffer para decir que lo vamos a utilizar
	glBindBuffer(GL_ARRAY_BUFFER, m_hIndexBuffer);
	//Mandamos la informacion contenida en un array de tipo int
	glBufferData(GL_ARRAY_BUFFER, (m_noIndices)*sizeof(int), bufferIndices, GL_STATIC_DRAW);
	//Indicamos que ya no vamos a utilizar ningun buffer
	glBindBuffer(GL_ARRAY_BUFFER, 0);
```

```
	//Borramos los arrays temporales donde guardamos la informacion leida del .nfg ya que la informacion esta en la VRam
	delete[] bufferVertex;
	delete[] bufferIndices;
```

- Recuerda hacer un glBindBuffer al handle (que ahora están en tu clase modelo) de tus vertex antes de mandar los atributos.

- Necesitaras usar **glDrawElements()** para renderiar, por que ya necesitamos trabajar con **indices**.

  Casi al final de la función Draw(), tenemos lo siguiente:

```
	glDrawArrays(GL_TRIANGLES, 0, 3);
```

  Sustituyelo por:

```
	glBindBuffer( GL_ELEMENT_ARRAY_BUFFER, model_load->getm_hIndexBuffer());
	glDrawElements( GL_TRIANGLES, model_load->getIndicesNum(), GL_UNSIGNED_INT, (void*)0 );
```

  Donde:

  - `model_ptr->getm_hIndexBuffer()` Devuelve el handle del Index Buffer Object

  - `model_ptr->getIndicesNum()` Devuelve el numero de indices

- Activa el depth test (en la función **`int Init(ESContext *esContext)`**)

```
	glEnable(GL_DEPTH_TEST);
```

- En la función **`int Init (ESContext *esContext)`** de NewTrainingFramework, has una instancia de Model, y llama a la función **Load()** de Model.

```
	model_ptr->InitModel(".\\Models\\Woman1.nfg");
```

- Por el momento “hardcodea” la posición, escala y rotación para calcular la worldMatrix (esto es para describir el objeto que pondremos en escena, que usara la geometría del modelo).

  Por ejemplo:

```
Matrix scaleMatrix;
Matrix rotationXMatrix;
Matrix rotationYMatrix;
Matrix rotationZMatrix;
Matrix translationMatrix;

scaleMatrix.SetIdentity();
rotationXMatrix.SetIdentity();
rotationYMatrix.SetIdentity();
rotationZMatrix.SetIdentity();
translationMatrix.SetIdentity();

//rotationXMatrix.SetRotationX(angle);
scaleMatrix.SetScale(0.8f);
```

- En la función **Draw(),** usa la world matrix anteriormente calculada para calcular la WordViewProjection Matrix

  WVP = world matrix of your scene object * view matrix * projection matrix.

  Donde:

  WoldMatrix → es la del objeto, la que está en este momento en NewTrainingFramework

  View Matrix → Es tomada de la instancia de la cámara

- En el Fragment Shader, asigna el color rojo para glFragColor.

```
gl_FragColor = vec4(1.0, 0.0, 0.0, 1.0);
                    R    G    B    A
```

- No olvidar en el destructor de la clase “Model”, hacer delete de los buffers objects:

```
if(m_hVertexBuffer)
    {
        glDeleteBuffers(1,&m_hVertexBuffer);
        m_hVertexBuffer = NULL;
    }
```