Realidad Virtual. Práctica 6.

TEXTURAS

Índice

1. Objetivos

- En esta práctica se describe la forma de añadir texturas a los objetos.
- Una textura describe la imagen que se proyecta sobre la superficie de los objetos.
- Para definir la forma de proyectar la imagen hay que añadir a los vértices un nuevo atributo que representa la posición del vértice sobre la imagen de textura.
  - Este atributo se conoce como “coordenadas de textura”.
- El resultado final de esta práctica es similar al presentado en la práctica anterior, pero añadiendo las texturas a los cuerpos representados.

- Código de la práctica:
  https://www.uhu.es/francisco.moreno/gii_rv/practicas/practica06/practica06.rar

2. La biblioteca FreeImage

- OpenGL utiliza su propio formato de almacenamiento de imágenes y no dispone de funciones que permitan leer ficheros de imágenes en los formatos comunes (GIF, JPG, PNG, etc).
- Para resolver este problema es habitual utilizar alguna biblioteca auxiliar dedicada al tratamiento de imágenes en distintos formatos.
- A lo largo de las prácticas vamos a utilizar una de estas bibliotecas denominada FreeImage.
- Al igual que hemos hecho con las bibliotecas glfw, glew y glm, la biblioteca FreeImage se ha incluido en la distribución del código de la práctica en el directorio 'C:\ComputerGraphics\Tools'.
- A lo largo de todas las prácticas hemos ido incluyendo la biblioteca FreeImage en las propiedades de los proyectos, tanto en los directorios de búsqueda como en las dependencias adicionales.

2. La biblioteca FreeImage

- En esta práctica vamos a hacer uso por primera vez de esta biblioteca.
- Para utilizar sus funciones hay que incluir en el código el archivo de cabecera <FreeImage.h>.
- Hay que tener en cuenta además que el fichero FreeImage.dll debe copiarse en el directorio de trabajo de la aplicación.

3. La clase CGMaterial

- Las texturas se van a configurar como parte de la descripción del material asociado a cada figura.
- Para ello se ha añadido un nuevo campo a la clase CGMaterial. Denominado textureId, que almacenará la referencia a la textura incluida en el material.
- Para almacenar una textura en la tarjeta gráfica se utilizará el método InitTexture(), que puede leer la textura desde un fichero externo o desde un recurso de la aplicación.
- Para poder utilizar la misma textura en varios materiales, se han añadido también los métodos GetTexture() y SetTexture() que permiten editar la referencia a la textura (textureId) sin necesidad de cargar una nueva imagen.

3. La clase CGMaterial

- Cabecera CGMaterial.h

```
#pragma once
#include <glm/glm.hpp>
#include "CGShaderProgram.h"
class CGMaterial {
private:
  glm::vec3 Ka;        // Reflectividad ambiental (color ante la luz ambiental)
  glm::vec3 Kd;        // Reflectividad difusa    (color ante la luz difusa)
  glm::vec3 Ks;        // Reflectividad especular (color ante la luz especular)
  GLfloat Shininess;   // Factor de brillo (comportamiento ante la luz especular)
  GLuint textureId;    // Identificador de la textura básica
public:
  CGMaterial();
  void SetAmbientReflect(GLfloat r, GLfloat g, GLfloat b);
  void SetDifusseReflect(GLfloat r, GLfloat g, GLfloat b);
  void SetSpecularReflect(GLfloat r, GLfloat g, GLfloat b);
  void SetShininess(GLfloat f);
  void SetUniforms(CGShaderProgram* program);
  void SetTexture(GLuint id);
  void InitTexture(const char* filename);
  void InitTexture(int idr);
  GLuint GetTexture();
};
```

3. La clase CGMaterial

- InitTexture(const char* filename): se encarga de leer el archivo de textura, almacenarlo en la tarjeta gráfica y configurar el modo de aplicación de la textura.
  - Este método utiliza las funciones de FreeImage para determinar el formato de la imagen a cargar (FreeImage_GetFileType)
  - Leer el fichero de imagen (FreeImage_Load)
  - Y convertirlo a un formato compatible con OpenGL (FreeImage_ConvertTo32Bits.
  - También utiliza funciones para obtener la anchura y altura de la imagen (FreeImage_GetWidth, FreeImage_GetHeight)
  - Y para eliminarla (FreeImage_Unload)
- El método InitTexture() se encarga también de crear un objeto textura (glGenTextures), seleccionar la textura (glBindTexture). Copiar los datos de la imagen (glTextImage2D) y configurar el proceso de aplicación de dicha textura (glTextParameter)

3. La clase CGMaterial

- InitTexture(int idr): similar al anterior, pero en este caso se carga el contenido de la textura desde un recurso de aplicación.
  - Para ello en primer lugar almacena en memoria el contenido del recurso de forma similar a la que se almacena el contenido de los recursos de los shaders.
  - A continuación, se utilizan las funciones de FreeImage para abrir como fichero el contenido almacenado en memoria (FreeImage_OpenMemory)
  - Determinar el formato de la imagen (FreeImage_GetFileTypeFromMemory)
  - Y cargar su contenido en la estructura de datos utilizada en la biblioteca (FreeImage_LoadFromMemory).
  - A partir de ahí el código es idéntico a la versión anterior.

3. La clase CGMaterial

- El método Draw() debe modificarse para utilizar la textura.
  - Para eso hay que activarla como GL_TEXTURE0 (glActiveTexture)
  - Seleccionarla (glBindTexture)
  - Y asignar el valor 0 a la variable uniforme BaseTex que se utiliza en el FragmentShader para acceder a la textura.

3. La clase CGMaterial

- Contenido CGMaterial.cpp

```
#include <Windows.h>
#include "CGMaterial.h"
#include <GL/glew.h>
#include <FreeImage.h>


 CGMaterial::CGMaterial()
{
  Ka = glm::vec3(1.0f, 1.0f, 1.0f);
  Kd = glm::vec3(1.0f, 1.0f, 1.0f);
  Ks = glm::vec3(0.8f, 0.8f, 0.8f);
  Shininess = 16.0f;
  textureId = 0;
}


void CGMaterial::SetAmbientReflect(GLfloat r,
GLfloat g, GLfloat b)
{
  Ka = glm::vec3(r, g, b);
}
```

3. La clase CGMaterial

```
void CGMaterial::SetDifusseReflect(GLfloat r,
GLfloat g, GLfloat b)
{
  Kd = glm::vec3(r, g, b);
}

void CGMaterial::SetSpecularReflect(GLfloat r,
GLfloat g, GLfloat b)
{
  Ks = glm::vec3(r, g, b);
}

void CGMaterial::SetShininess(GLfloat f)
{
  Shininess = f;
}
```

3. La clase CGMaterial

```
void CGMaterial::SetUniforms(CGShaderProgram* program)
{
  program->SetUniformVec3("Material.Ka", Ka);
  program->SetUniformVec3("Material.Kd", Kd);
  program->SetUniformVec3("Material.Ks", Ks);
  program->SetUniformF("Material.Shininess", Shininess);
  program->SetUniformI("BaseTex", 0);
  glActiveTexture(GL_TEXTURE0);
  glBindTexture(GL_TEXTURE_2D, textureId);
}

void CGMaterial::SetTexture(GLuint id)
{
  textureId = id;
}
```

3. La clase CGMaterial

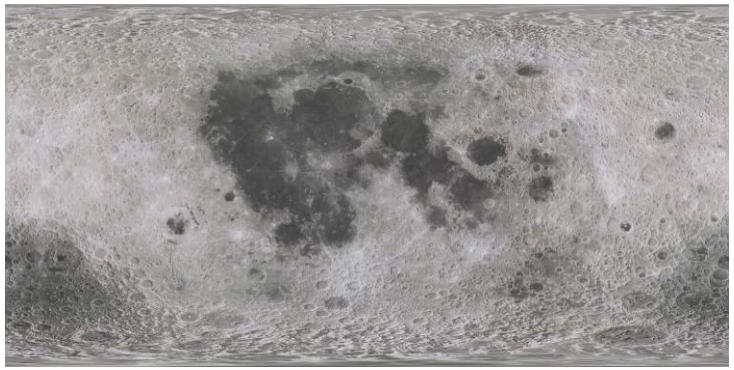
```
GLuint CGMaterial::GetTexture()
{
  return textureId;
}
void CGMaterial::InitTexture(const char* filename)
{
  FREE_IMAGE_FORMAT format = FreeImage_GetFileType(filename, 0);
  FIBITMAP* bitmap = FreeImage_Load(format, filename);
  FIBITMAP* pImage = FreeImage_ConvertTo32Bits(bitmap);
  int nWidth = FreeImage_GetWidth(pImage);
  int nHeight = FreeImage_GetHeight(pImage);
  glGenTextures(1, &textureId);
  glBindTexture(GL_TEXTURE_2D, textureId);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
  glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA8, nWidth, nHeight,
    0, GL_BGRA, GL_UNSIGNED_BYTE, (void*)FreeImage_GetBits(pImage));
  FreeImage_Unload(pImage);
}
```

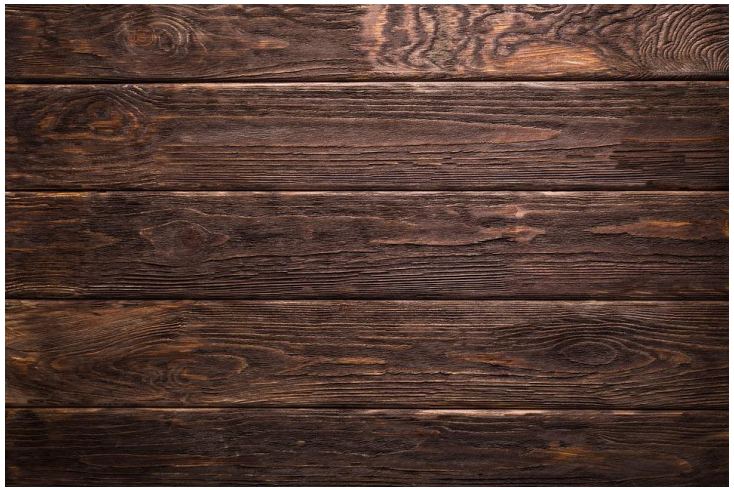
3. La clase CGMaterial

```
void CGMaterial::InitTexture(int idr)
{
  HRSRC handle = FindResource(NULL, MAKEINTRESOURCE(idr), L"IMAGE");
  HGLOBAL hGlobal = LoadResource(NULL, handle);
  LPCTSTR rsc_ptr = static_cast<LPCTSTR>(LockResource(hGlobal));
  DWORD mem_size = SizeofResource(NULL, handle);
  BYTE* mem_buffer = (BYTE*)malloc((mem_size) * sizeof(BYTE));
  memcpy(mem_buffer, rsc_ptr, mem_size * sizeof(BYTE));
  FreeResource(hGlobal);
  FIMEMORY* hmem = FreeImage_OpenMemory(mem_buffer, mem_size * sizeof(BYTE));
  FREE_IMAGE_FORMAT fif = FreeImage_GetFileTypeFromMemory(hmem, 0);
  FIBITMAP* check = FreeImage_LoadFromMemory(fif, hmem, 0);
  FIBITMAP* pImage = FreeImage_ConvertTo32Bits(check);
  int nWidth = FreeImage_GetWidth(pImage);
  int nHeight = FreeImage_GetHeight(pImage);
  glGenTextures(1, &textureId);
  glBindTexture(GL_TEXTURE_2D, textureId);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_MIRRORED_REPEAT);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_MIRRORED_REPEAT);
  glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA8, nWidth, nHeight,
   0, GL_BGRA, GL_UNSIGNED_BYTE, (void*)FreeImage_GetBits(pImage));
  FreeImage_Unload(pImage);
  FreeImage_CloseMemory(hmem);
  free(mem_buffer);
}
```

4. Texturas

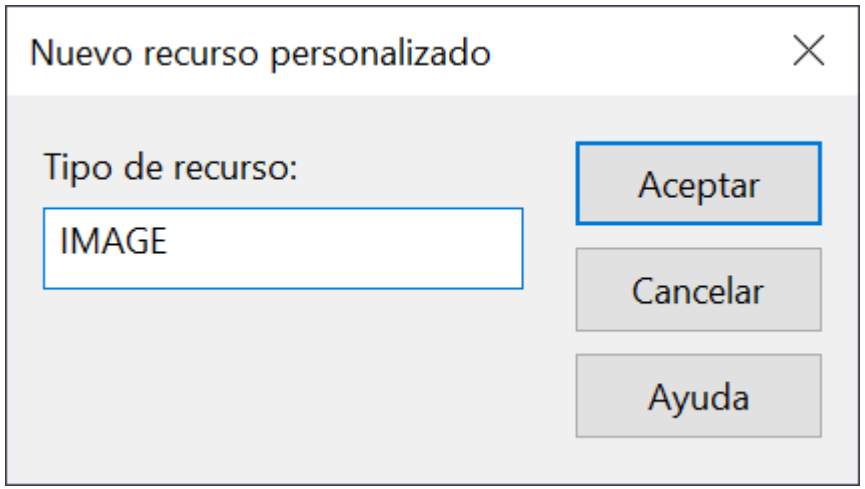
- Los ficheros de textura que vamos a utilizar en esta práctica se llaman “stone.jpg”, “moon.jpg” y “wood.jpg”.
  - Estas se encontrarán en el directorio “textures”.





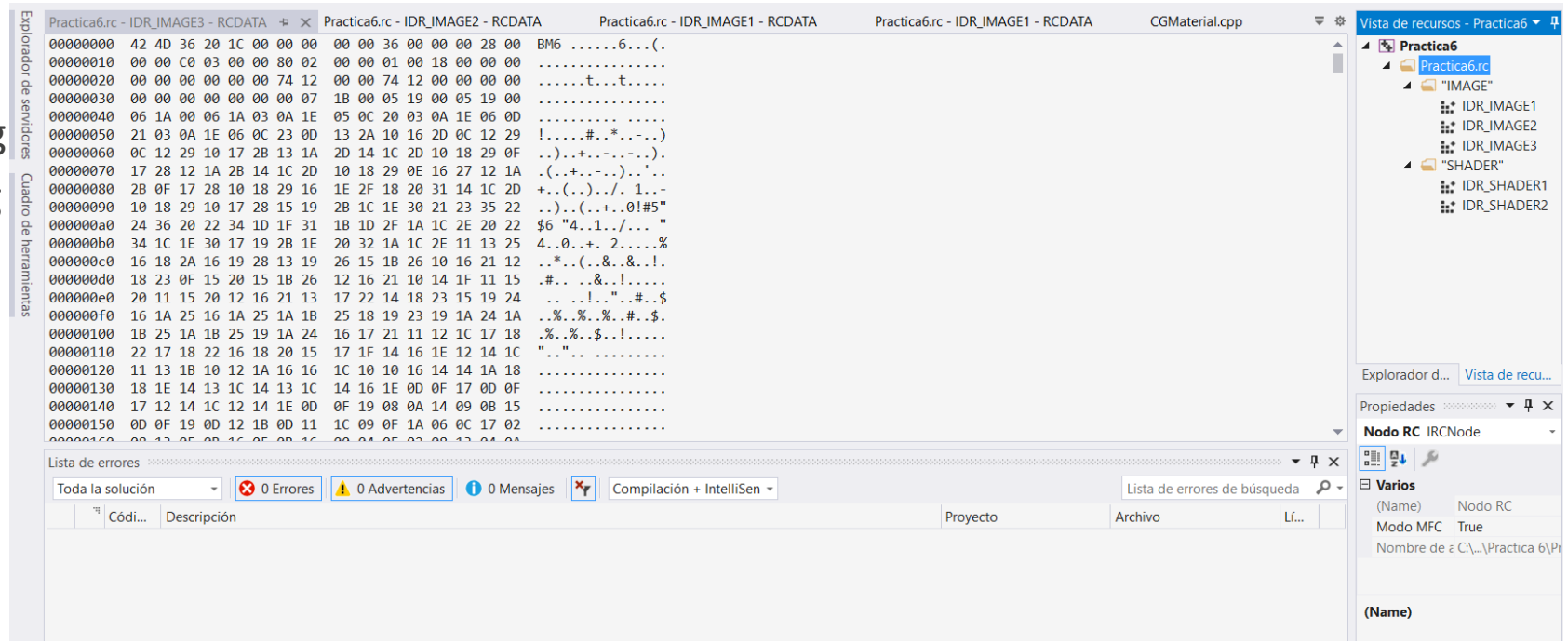
4. Texturas

- Para cargar las texturas como recursos en VisualStudio se procede de la misma forma que se indicó en la práctica 2.
  - Explorador de soluciones --> Archivos de recursos --> (sobre este botón derecho) --> Agregar --> Recurso… --> Personalizar --> Nombre IMAGE.



4. Texturas

- Cargamos las texturas como cargamos el VertexShader y el FragmentShader.
- En Practica6.rc --> click derecho --> Agregar recurso… --> Importar --> cargar las imágenes en el siguiente orden:
  - IDR_IMAGE1 --> stone.jpg
  - IDR_IMAGE2 --> moon.jpg
  - IDR_IMAGE3 --> wood.jpg



5. El programa gráfico

- Para incorporar texturas en la generación de la imagen es necesario añadir a los vértices un nuevo atributo que indica la posición de cada vértice sobre la textura.
  - Este atributo se conoce como “coordenadas de textura” y está formado por dos componentes definidas en un rango entre 0.0 y 1.0.
- La incorporación de estos nuevos atributos suponen modificar el VertexShader de manera que incluya una nueva entrada (VertexTexCoord) y una nueva salida (TexCoord) de tipo vec2.
- Las coordenadas de textura no dependen de la posición del observador así que no es necesario transformarlas dentro del VertexShader.
  - Por tanto, el código del VertexShader se limita a copiar las coordenadas de textura como variable de salida para que puedan ser procesadas en el FragmentShader.

5. El programa gráfico

- Contenido VertexShader.glsl

```
#version 400
layout(location = 0) in vec3 VertexPosition;
layout(location = 1) in vec3 VertexNormal;
layout(location = 2) in vec2 VertexTexCoord;
uniform mat4 MVP;
uniform mat4 ViewMatrix;
uniform mat4 ModelViewMatrix;
out vec3 Position;
out vec3 Normal;
out vec2 TexCoord;

void main()
{
  vec4 n4 = ModelViewMatrix*vec4(VertexNormal, 0.0);
  vec4 v4 = ModelViewMatrix*vec4(VertexPosition, 1.0);
  Normal = vec3(n4);
  Position = vec3(v4);
  TexCoord = VertexTexCoord;
  gl_Position = MVP * vec4(VertexPosition, 1.0);
}
```

5. El programa gráfico

- Para incorporar la textura al cálculo del color de cada pixel es necesario incluir en el FragmentShader tanto las coordenadas de textura del pixel (TexCoord) como una nueva variable uniforme que permita acceder a la imagen de textura (BaseTex).
- Las texturas se tratan en GLSL como tipos de datos opacos denominados sampler2D (para texturas 2D).
  - En realidad, la variable uniforme BaseTex contiene el identificador que permite referenciar la textura activa.
  - Por ejemplo, el valor 0 permite referenciar a la textura activa GL_TEXTURE0.
- Para acceder al color correspondiente a una cierta coordenada de textura (lo que se conoce como un texel) se utiliza la función predefinida texture() que devuelve un vec4 que contiene el color del texel en formato RGBA.

5. El programa gráfico

- A continuación, se muestra el contenido del FragmentShader incorporando el modelo de iluminación de Phong junto al uso de texturas.
  - Para ello se ha modificado la función ads() para incluir como parámetro el color de textura del pixel.
  - El color de la textura influye en el efecto de la luz ambiental y difusa, pero no en el de la luz especualr.
  - La razón por la que se trata de forma diferente la luz especular es que ésta es la que provoca un efecto de brillo o de destello.
  - Si un pixel tiene un color de textura oscuro y multiplicamos este color por el efecto de la luz especular, como mucho el color seguirá siendo oscuro.
  - Al no incluir la textura sobre el efecto especular se consigue mantener los efectos de brillo.

5. El programa gráfico

```
#version 400

in vec3 Position;
in vec3 Normal;
in vec2 TexCoord;

uniform sampler2D BaseTex;

uniform mat4 ViewMatrix;

struct LightInfo {
  vec3 Ldir;
  vec3 La;
  vec3 Ld;
  vec3 Ls;
};
uniform LightInfo Light;

struct MaterialInfo{
  vec3 Ka;
  vec3 Kd;
  vec3 Ks;
  float Shininess;
};
uniform MaterialInfo Material;

out vec4 FragColor;

vec3 ads(vec3 TexColor) {
  vec4 s4 = ViewMatrix*vec4(Light.Ldir, 0.0);
  vec3 n = normalize(Normal);
  vec3 v = normalize(-Position);
  vec3 s = normalize(-vec3(s4));
  vec3 r = reflect(-s, n);
  float dRate = max(dot(s, n), 0.0);
  float sRate = pow(max(dot(r, v), 0.0), Material.Shininess);
  vec3 ambient = Light.La * Material.Ka;
  vec3 difusse = Light.Ld * Material.Kd * dRate;
  vec3 specular = Light.Ls * Material.Ks * sRate;
  return (ambient + difusse)*TexColor + specular;
}

void main()
{
  vec3 TexColor = vec3( texture(BaseTex,TexCoord) );
  vec3 Color = ads(TexColor);
  FragColor = vec4(Color,1.0);
}
```

6. La clase CGFigure

- La clase CGFigure describe el comportamiento general de cualquiera de las figuras geométricas que vamos a incluir en la escena.
- Con respecto a la práctica anterior, la modificación principal consiste en añadir un nuevo atributo al VAO asociado a la figura.
- Este atributo contendrá las coordenadas de textura de cada vértice y requiere su propio VBO para almacenar sus datos.
- Estos datos se incluirán inicialmente en un campo denominado textures (los valores de este array dependen de cada figura geométrica concreta) cuyo contenido se almacenará en el VBO correspondiente al ejecutar el método InitBuffers().

6. La clase CGFigure

- Cabecera CGFigure.h

```
#pragma once

#include <GL/glew.h>
#include <glm/glm.hpp>
#include "CGMaterial.h"
#include "CGShaderProgram.h"

#define VERTEX_DATA 0
#define INDEX_DATA 1
#define NORMAL_DATA 2
#define TEXTURE_DATA 3

class CGFigure {
protected:
  GLushort* indexes; // Array of indexes
  GLfloat* vertices; // Array of vertices
  GLfloat* normals; // Array of normals
  GLfloat* textures; // Array of texture coordinates

  GLuint numFaces; // Number of faces
  GLuint numVertices; // Number of vertices
  GLuint VBO[4];
  GLuint VAO;

  glm::mat4 location; // Model matrix
  CGMaterial* material;

public:
  ~CGFigure();
  void InitBuffers();
  void SetMaterial(CGMaterial* mat);
  void ResetLocation();
  void Translate(glm::vec3 t);
  void Rotate(GLfloat angle, glm::vec3 axis);
  void Draw(CGShaderProgram* program, glm::mat4 projection, glm::mat4 view);
};

#endif
```

6. La clase CGFigure

- Contenido CGFigure.cpp

```
#include "CGFigure.h"
#include <GL/glew.h>
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>

//
CGFigure::~CGFigure()
{
    if (vertices != NULL) delete[] vertices;
    if (indexes != NULL) delete[] indexes;
    if (normals != NULL) delete[] normals;
    if (textures != NULL) delete[] textures;

    // Delete vertex buffer objects
    glDeleteBuffers(4, VBO);
    glDeleteVertexArrays(1, &VAO);
}
```

6. La clase CGFigure

```
void CGFigure::InitBuffers()
{
  // Create the Vertex Array Object
  glGenVertexArrays(1, &VAO);
  glBindVertexArray(VAO);

  // Create the Vertex Buffer Objects
  glGenBuffers(4, VBO);

  // Copy data to video memory
  // Vertex data
  int buffsize = sizeof(GLfloat) * numVertices * 3;
  glBindBuffer(GL_ARRAY_BUFFER, VBO[VERTEX_DATA]);
  glBufferData(GL_ARRAY_BUFFER, buffsize, vertices, GL_STATIC_DRAW);

  // Normal data
  glBindBuffer(GL_ARRAY_BUFFER, VBO[NORMAL_DATA]);
  glBufferData(GL_ARRAY_BUFFER, buffsize, normals, GL_STATIC_DRAW);

  // Texture coordinates
  buffsize = sizeof(GLfloat) * numVertices * 2;
  glBindBuffer(GL_ARRAY_BUFFER, VBO[TEXTURE_DATA]);
  glBufferData(GL_ARRAY_BUFFER, buffsize, textures, GL_STATIC_DRAW);

  // Indexes
  buffsize = sizeof(GLushort) * numFaces * 3;
  glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, VBO[INDEX_DATA]);
  glBufferData(GL_ELEMENT_ARRAY_BUFFER, buffsize, indexes, GL_STATIC_DRAW);

  delete[] vertices;
  delete[] indexes;
  delete[] normals;
  delete[] textures;

  vertices = NULL;
  indexes = NULL;
  normals = NULL;
  textures = NULL;

  glEnableVertexAttribArray(0); // Vertex position
  glBindBuffer(GL_ARRAY_BUFFER, VBO[VERTEX_DATA]);
  glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);

  glEnableVertexAttribArray(1); // Vertex normals
  glBindBuffer(GL_ARRAY_BUFFER, VBO[NORMAL_DATA]);
  glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 0, 0);

  glEnableVertexAttribArray(2); // Vertex textures
  glBindBuffer(GL_ARRAY_BUFFER, VBO[TEXTURE_DATA]);
  glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 0, 0);

  location = glm::mat4(1.0f);
}
```

6. La clase CGFigure

```
void CGFigure::SetMaterial(CGMaterial* mat)
{
  material = mat;
}

void CGFigure::ResetLocation()
{
  location = glm::mat4(1.0f);
}
```

```
void CGFigure::Translate(glm::vec3 t)
{
  location = glm::translate(location, t);
}

void CGFigure::Rotate(GLfloat angle, glm::vec3 axis)
{
  location = glm::rotate(location, glm::radians(angle), axis);
}
```

6. La clase CGFigure

```
void CGFigure::Draw(CGShaderProgram* program, glm::mat4 projection, glm::mat4 view)
{
  glm::mat4 mvp = projection * view * location;
  program->SetUniformMatrix4("MVP", mvp);
  program->SetUniformMatrix4("ViewMatrix", view);
  program->SetUniformMatrix4("ModelViewMatrix", view * location);
  material->SetUniforms(program);

  glBindVertexArray(VAO);
  glDrawElements(GL_TRIANGLES, numFaces * 3, GL_UNSIGNED_SHORT, NULL);
}
```

7. La clase CGScene

- La escena a representar en el modelo (descrita en la clase CGScene) es muy parecida a la versión de la práctica anterior.
- La única diferencia consiste en que se han definido materiales diferentes para cada figura para que puedan dibujarse en diferentes colores y que se han añadido las texturas a estos materiales.
- La textura Stone se va a utilizar en diferentes materiales.
- Para no cargarla varias veces se ha cargado una única vez en el material ‘mat0’ y se ha utilizado el mismo identificador de textura (textureId) en el resto de los materiales que la utilizan.
- El fichero de cabecera de la clase incorpora ahora los nuevos materiales.

7. La clase CGScene

- Cabecera CGScene.h

```
#pragma once


#include <GL/glew.h>
#include <glm/glm.hpp>
#include "CGShaderProgram.h"
#include "CGLight.h"
#include "CGMaterial.h"
#include "CGFigure.h"


class CGScene {
public:
  CGScene();
  ~CGScene();
  void Draw(CGShaderProgram* program, glm::mat4 proj, glm::mat4 view);


private:
  CGFigure* ground;
  CGFigure* fig0;
  CGFigure* fig1;
  CGFigure* fig2;
  CGFigure* fig3;
  CGFigure* fig4;
  CGFigure* fig5;
  CGLight* light;
  CGMaterial* matg;
  CGMaterial* mat0;
  CGMaterial* mat1;
  CGMaterial* mat2;
  CGMaterial* mat3;
  CGMaterial* mat4;
  CGMaterial* mat5;
};
```

7. La clase CGScene

- Contenido CGScene.cpp

```
#include "CGScene.h"
#include <GL/glew.h>
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include "CGShaderProgram.h"
#include "CGFigure.h"
#include "CGLight.h"
#include "CGMaterial.h"
#include "CGCube.h"
#include "CGCone.h"
#include "CGCylinder.h"
#include "CGSphere.h"
#include "CGTorus.h"
#include "CGIcosahedron.h"
#include "CGGround.h"
```

7. La clase CGScene

```
CGScene::CGScene()
{
  glm::vec3 Ldir = glm::vec3(1.0f, -0.8f, -1.0f);
  Ldir = glm::normalize(Ldir);
  light = new CGLight();
  light->SetLightDirection(Ldir);
  light->SetAmbientLight(glm::vec3(0.2f, 0.2f, 0.2f));
  light->SetDifusseLight(glm::vec3(0.8f, 0.8f, 0.8f));
  light->SetSpecularLight(glm::vec3(1.0f, 1.0f, 1.0f));

  matg = new CGMaterial();
  matg->SetAmbientReflect(1.0f, 1.0f, 1.0f);
  matg->SetDifusseReflect(1.0f, 1.0f, 1.0f);
  matg->SetSpecularReflect(0.8f, 0.8f, 0.8f);
  matg->SetShininess(16.0f);
  //matg->InitTexture("textures/wood.jpg");
  matg->InitTexture(IDR_IMAGE2);

  ground = new CGGround(50.0f, 50.0f);
  ground->SetMaterial(matg);

  mat0 = new CGMaterial();
  mat0->SetAmbientReflect(1.0f, 0.0f, 1.0f);
  mat0->SetDifusseReflect(1.0f, 0.0f, 1.0f);
  mat0->SetSpecularReflect(0.8f, 0.8f, 0.8f);
  mat0->SetShininess(16.0f);
  //mat0->InitTexture("textures/stone.jpg");
  mat0->InitTexture(IDR_IMAGE1);
```

7. La clase CGScene

```
GLuint textureId = mat0->GetTexture();

fig0 = new CGCone(5, 20, 5.0f, 5.0f);
fig0->SetMaterial(mat0);
fig0->Translate(glm::vec3(25.0f, 5.0f, 25.0f));
fig0->Rotate(-90.0f, glm::vec3(1.0f, 0.0f, 0.0f));

mat1 = new CGMaterial();
mat1->SetAmbientReflect(1.0f, 1.0f, 1.0f);
mat1->SetDifusseReflect(1.0f, 1.0f, 1.0f);
mat1->SetSpecularReflect(0.8f, 0.8f, 0.8f);
mat1->SetShininess(16.0f);
```

```
mat1->SetTexture(textureId);

fig1 = new CGCube(5.0f);
fig1->SetMaterial(mat1);
fig1->Translate(glm::vec3(-25.0f, 5.0f, 25.0f));
fig1->Rotate(-90.0f, glm::vec3(1.0f, 0.0f, 0.0f));

mat2 = new CGMaterial();
mat2->SetAmbientReflect(0.2f, 0.2f, 1.0f);
mat2->SetDifusseReflect(0.2f, 0.2f, 1.0f);
mat2->SetSpecularReflect(0.8f, 0.8f, 0.8f);
mat2->SetShininess(16.0f);
```

```
mat2->SetTexture(textureId);

fig2 = new CGCylinder(20, 20, 5.0f, 5.0f);
fig2->SetMaterial(mat2);
fig2->Translate(glm::vec3(25.0f, 5.0f, 0.0f));
fig2->Rotate(90.0f, glm::vec3(1.0f, 0.0f, 0.0f));
```

7. La clase CGScene

```
mat3 = new CGMaterial();
 mat3->SetAmbientReflect(1.0f, 0.0f, 0.0f);
 mat3->SetDifusseReflect(1.0f, 0.0f, 0.0f);
 mat3->SetSpecularReflect(0.8f, 0.8f, 0.8f);
 mat3->SetShininess(16.0f);
 mat3->SetTexture(textureId);

 fig3 = new CGTorus(20, 40, 3.0f, 5.0f);
 fig3->SetMaterial(mat3);
 fig3->Translate(glm::vec3(-25.0f, 8.0f, 0.0f));

 mat4 = new CGMaterial();
 mat4->SetAmbientReflect(1.0f, 1.0f, 0.0f);
 mat4->SetDifusseReflect(1.0f, 1.0f, 0.0f);
 mat4->SetSpecularReflect(0.8f, 0.8f, 0.8f);
 mat4->SetShininess(16.0f);
 //mat4->InitTexture("textures/moon.jpg");
 mat4->InitTexture(IDR_IMAGE3);

 fig4 = new CGSphere(20, 40, 8.0f);
 fig4->SetMaterial(mat4);
 fig4->Translate(glm::vec3(25.0f, 8.0f, -25.0f));

 mat5 = new CGMaterial();
 mat5->SetAmbientReflect(0.0f, 1.0f, 1.0f);
 mat5->SetDifusseReflect(0.0f, 1.0f, 1.0f);
 mat5->SetSpecularReflect(0.8f, 0.8f, 0.8f);
 mat5->SetShininess(16.0f);
 mat5->SetTexture(textureId);

 fig5 = new CGIcosahedron(5.0f);
 fig5->SetMaterial(mat5);
 fig5->Translate(glm::vec3(-25.0f, 8.0f, -
25.0f));
}
```

7. La clase CGScene

```
CGScene::~CGScene()
{
  delete ground;
  delete fig0;
  delete fig1;
  delete fig2;
  delete fig3;
  delete fig4;
  delete fig5;
  delete light;
  delete matg;
  delete mat0;
  delete mat1;
  delete mat2;
  delete mat3;
  delete mat4;
  delete mat5;
}
void CGScene::Draw(CGShaderProgram* program, glm::mat4 proj, glm::mat4 view)
{
  light->SetUniforms(program);

  ground->Draw(program, proj, view);
  fig0->Draw(program, proj, view);
  fig1->Draw(program, proj, view);
  fig2->Draw(program, proj, view);
  fig3->Draw(program, proj, view);
  fig4->Draw(program, proj, view);
  fig5->Draw(program, proj, view);
}
```

8. Figuras geométricas

- Con respecto a las diferentes figuras geométricas, es necesario modificar su código para incorporar a los atributos de los vértices el valor de las coordenadas de textura en cada vértice.