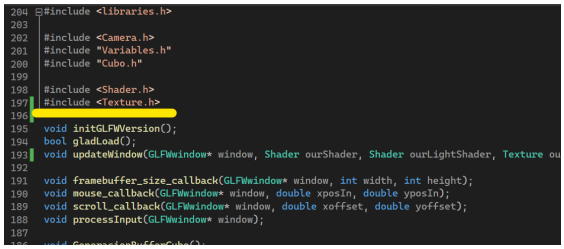
Programación de Videojuegos

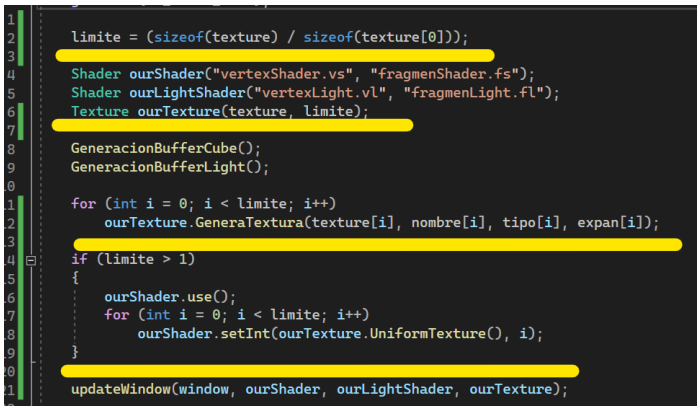
Universidad Autónoma de San Luis Potosí
Facultad de Ingeniería
Parcial 2
Angel de Jesús Maldonado Juárez
Actividad 1
14 de marzo del 2023

Incorporación de texturas al proyecto de iluminación

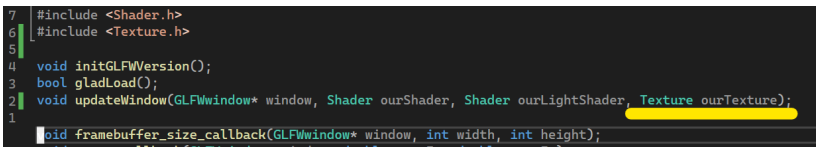
Como primer paso simplemente se agregó la directiva `#include` con la librería `Texture.h` en el archivo `main.cpp`, la cual tiene la clase `Texture` para poder importar, cargar, y manejar texturas:



En la función `main()` de este mismo archivo, se agregó la variable `ourTexture`, la cual se utiliza para cargar y guardar la textura en tiempo de ejecución:



En este mismo archivo, en la función `updateWindow()` se le agregó como parámetro un objeto `Texture` llamado `ourTexture`, el cual se utiliza para renderizar en tiempo de ejecución la(s) textura(s) que vayan a cargarse utilizando el método `ViewTexture()` del objeto del parámetro:



```cpp
void updateWindow(GLFWwindow* window, Shader ourShader, Shader ourLightShader, Texture ourTexture)
{
    while (!glfwWindowShouldClose(window))
    {
        float currentFrame = glfwGetTime();
        deltaTime = currentFrame - lastFrame;
        lastFrame = currentFrame;

        processInput(window);

        glClearColor(0.1f, 0.1f, 0.1f, 1.0f);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

        ourTexture.ViewTexture();

        ourShader.use();
        ourShader.setVec3("viewPos", camera.Position);
        ourShader.setVec3("objectColor", 0.19f, 0.59f, 1.0f);
```

En el archivo `Cube.h`, se le agregó al arreglo `vertices`, las coordenadas de la textura después de las normales:

```cpp
float vertices[] = {
    -0.5f, -0.5f, -0.5f,    -1.0f, 0.0f, 0.0f,    0.0f, 0.0f,
     0.5f, -0.5f, -0.5f,    -1.0f, 0.0f, 0.0f,    1.0f, 0.0f,
     0.5f,  0.5f, -0.5f,    -1.0f, 0.0f, 0.0f,    1.0f, 1.0f,
    - 0.5f,  0.5f, -0.5f,    -1.0f, 0.0f, 0.0f,    0.0f, 1.0f,
    - 0.5f, -0.5f,  0.5f,    1.0f, 0.0f, 0.0f,     0.0f, 0.0f,
     0.5f, -0.5f,  0.5f,     1.0f, 0.0f, 0.0f,     1.0f, 0.0f,
     0.5f,  0.5f,  0.5f,     1.0f, 0.0f, 0.0f,     1.0f, 1.0f,
    - 0.5f,  0.5f,  0.5f,    1.0f, 0.0f, 0.0f,     0.0f, 1.0f,
    - 0.5f,  0.5f,  0.5f,    0.0f, 0.0f, -1.0f,    1.0f, 0.0f,
    - 0.5f,  0.5f, -0.5f,    0.0f, 0.0f, -1.0f,    1.0f, 1.0f,
    - 0.5f, -0.5f, -0.5f,    0.0f, 0.0f, -1.0f,    0.0f, 1.0f,
     0.5f,  0.5f,  0.5f,     0.0f, 0.0f, -1.0f,    1.0f, 0.0f,
     0.5f, -0.5f, -0.5f,     0.0f, 0.0f, 1.0f,     0.0f, 1.0f,
     0.5f, -0.5f,  0.5f,     0.0f, 0.0f, 1.0f,     0.0f, 0.0f,
     0.5f, -0.5f, -0.5f,     0.0f, 0.0f, 1.0f,     1.0f, 1.0f,
    - 0.5f,  0.5f,  0.5f,    0.0f, 0.0f, 1.0f,     0.0f, 0.0f
};
```

Nuevamente en el archivo `main.cpp` se modificaron las funciones `GeneracionBufferCube()` y `GeneracionBufferLight()` , agregando la nueva distribución del arreglo `vertices`:

```cpp
void GeneracionBufferCube()
{
    glGenVertexArrays(1, &VAO);
    glGenBuffers(1, &VBO);
    glGenBuffers(1, &EBO);
    glBindVertexArray(VAO);

    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBufferData(GL_ARRAY_BUFFER, sizeof(vertices), vertices, GL_STATIC_DRAW);

    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);
    glBufferData(GL_ELEMENT_ARRAY_BUFFER, sizeof(indices), indices, GL_STATIC_DRAW);

    VertexAttribute(0, 3, 8, 0);
    VertexAttribute(1, 3, 8, 3);
    VertexAttribute(2, 2, 8, 6);

    glBindBuffer(GL_ARRAY_BUFFER, 0);
    glBindVertexArray(0);
}
```

```cpp
void GeneracionBufferLight()
{
    glGenVertexArrays(1, &lightVAO);
    glBindVertexArray(lightVAO);

    glBindBuffer(GL_ARRAY_BUFFER, VBO);
    glBindBuffer(GL_ELEMENT_ARRAY_BUFFER, EBO);

    VertexAttribute(0, 3, 8, 0);

    glBindBuffer(GL_ARRAY_BUFFER, 0);
    glBindVertexArray(0);
}
```

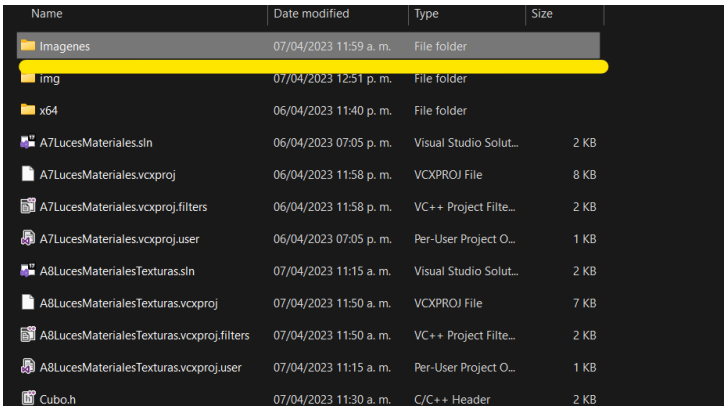
En el archivo `vertexShader.vs` se agregó el nuevo `layout` con las coordenadas de la textura llamado `aTexture1`, esta misma variable se declara como salida `ourTexture1` para que el *Fragment Shader* lo pueda recibir, finalmente, en el `main()` se iguala `ourTexture1` con el valor de `aTexture1`:

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aNormal;
layout (location = 2) in vec2 aTexture1;

out vec3 FragPos;
out vec3 Normal;
out vec2 ourTexture1;

uniform mat4 model;
uniform mat4 view;
uniform mat4 projection;

void main()
{
    FragPos = vec3(model * vec4(aPos, 1.0f));
    Normal = aNormal;
    gl_Position = projection * view * model * vec4(aPos, 1.0f);
    ourTexture1 = aTexture1;
}
```

En el archivo `fragmenShader.fs` se declara la variable de entrada `ourTexture1`, que representa las coordenadas de la textura, y la variable uniforme `texture1`, que será la imagen de la textura como tal. En el `main()` como prueba, simplemente se deja comentada la línea en donde se manda la salida `FragColor` con el resultado de las operaciones con la luz, y se iguala a la creación de una única textura:

```
#version 330 core
out vec4 FragColor;

in vec3 FragPos;
in vec3 Normal;
in vec2 ourTexture1;

struct Material {
    vec3 ambient;
    vec3 diffuse;
    vec3 specular;
    float shininght;
};

struct Light{
    vec3 ambient;
    vec3 diffuse;
    vec3 specular;

    vec3 pose;
};

uniform Material material;
uniform Light light;
uniform sampler2D texture1;

uniform vec3 lightColor;
uniform vec3 lightPos;
uniform vec3 viewPos;

void main()
{
    vec3 ambiental = material.ambient * light.ambient;

    vec3 norm = normalize(Normal);
    vec3 lightDir = normalize(light.pose - FragPos);
    float diff = max(dot(norm, lightDir), 0.0f);
    vec3 difuse = (diff * material.diffuse) * light.diffuse;

    vec3 viewDir = normalize(viewPos - FragPos);
    vec3 reflectDir = reflect(-lightDir, norm);
    float spec = pow(max(dot(viewDir, reflectDir), 0.0f), material.shininght);
    vec3 specular = spec * light.specular * material.specular;

    vec3 result = ambiental + difuse + specular;
    //FragColor = vec4(result, 1.0f);
    FragColor = texture(texture1, ourTexture1);
}
```

Finalmente, en el directorio del proyecto se agregó otro directorio llamado `Imagenes` el cual contendrá todos los archivos de imagen relacionados con las texturas de los objetos:



Al ejecutar el proyecto se puede observar el cubo cargado con la textura `caja.jpg` de manera exitosa junto con la luz: