

HAI719I – Programmation 3D TP Textures

Fabien Caballero

December 23, 2022

Contents

1	Plaquer une texture	2
2	Plaquage d'une texture avec utilisation d'une normalMap	3
3	SkyBox	4
4	Objet reflété	6
5	Remarques	7

1 Plaquer une texture

On crée un objet Gluint texture que l'on initialise grâce à notre image texture en utilisant la fonction stbi_load. Puis on utilise glTexImage2D et glGenerateMipmap pour générer notre objet texture et on l'envoie à notre shader avec glUniform.

```
1 // Material.h
2     GLuint m_texture;
3 // Material.cpp
4     m_texture = loadTexture2DFromFilePath("data/brickWall.png");
5
6 // Texture.cpp
7 GLuint loadTexture2DFromFilePath( const std::string &path)
8 {
9     GLuint texture;
10    glGenTextures(1, &texture);
11    glBindTexture(GL_TEXTURE_2D, texture);
12
13    int width, height, nrChannels;
14    unsigned char *data = stbi_load(path.c_str(), &width, &height, &nrChannels, 3);
15
16    if (!data)
17    {
18        stbi_image_free(data);
19        throw std::runtime_error("Failed to load texture: " + path);
20    }
21
22    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data)
23    ;
24    glGenerateMipmap(GL_TEXTURE_2D);
25    stbi_image_free(data);
26    setDefaultTexture2DParameters(texture);
27    return texture;
28 }
29 \\ Material.cpp
30 glBindActiveTexture(GL_TEXTURE0);
31     glUniform1i(getUniform("colorText"), 0);
32     glBindTexture(GL_TEXTURE_2D, m_texture);
```

Puis finalement dans notre shader on utilise la fonction texture pour attribuer la texture à notre fragment.

```
1 texture(colorText, o_uv0)
```

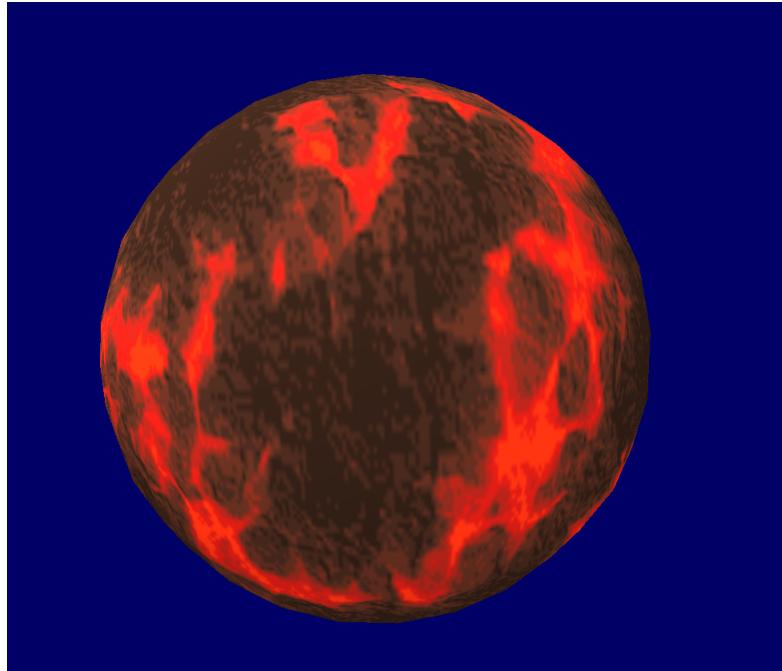


Figure 1: Sphere avec texture lave

2 Plaqueage d'une texture avec utilisation d'une normalMap

On fait pareil que précédemment pour avoir une texture, mais on ajoute cette fois ci un autre objet texture qui est notre texture normalMap, en faisant exactement pareil que précédemment mais en donnant en paramètre de loadTexture2DFromFilePath la path de notre image normalMap. Il faut pour cela calculer un rendu de lumière avec Phong, das le vertex shader, par exemple pour cela on utilise la normale, pour avoir la tangeante et la bitangeante puis dans le fragment shader on calcule notre phong en utilisant comme normal celle de notre texture NormalMap.

```

1      #version 330 core
2
3  out vec4 FragColor;
4
5  in vec3 o_positionWorld;
6  in vec2 o_uv0;
7  in vec3 tangentLightPos;
8  in vec3 tangentViewPos;
9  in vec3 tangentFragmentPos;
10
11 uniform sampler2D colorText;
12 uniform sampler2D normalText;
13 uniform vec3 lightPos;
14 uniform vec3 viewPos;
15
16 void main() {
17
18     vec3 normal=texture(normalText,o_uv0).rgb;
19     normal=normalize(normal*2.0-1.0);
20
21     vec3 color=texture(colorText,o_uv0).rgb;
22
23     vec3 ambiant=1.0*color;

```

```

24     vec3 lightDir=normalize(tangentLightPos - tangentFragmentPos);
25     float diff=max(dot(lightDir,normal),0.0);
26     vec3 diffuse=diff*color;
27
28     vec3 viewDir=normalize(tangentViewPos-tangentFragmentPos);
29     vec3 reflex=reflect(-lightDir,normal);
30     vec3 halfway=normalize(lightDir+viewDir);
31     float spec=pow(max(dot(normal,halfway),0.0),32.0);
32     vec3 specular=vec3(0.2)*spec;
33
34     FragColor=vec4(ambient+diffuse+specular,1.0);
35 }

```

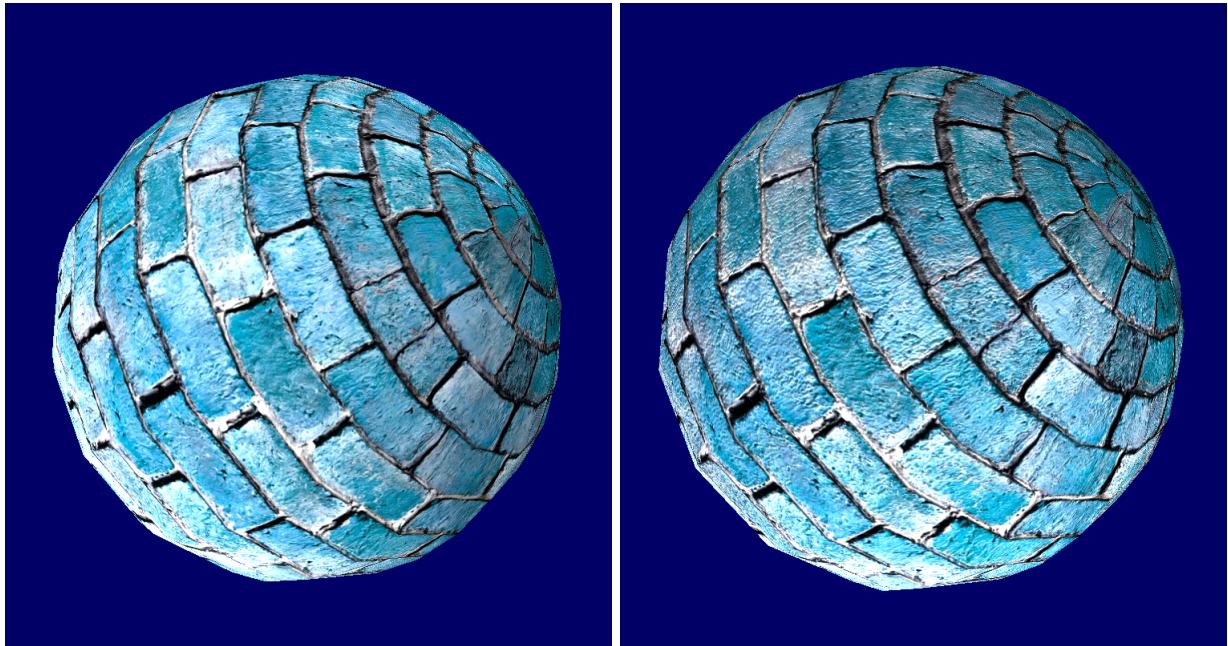


Figure 2: Sphere avec texture brick, sans normalMap à gauche et avec normalMap à droite

3 SkyBox

Pour réaliser la skyBox on définit un objet vertex array sur notre GPU puis grâce à un buffer on lui envoi les données des vertices de notre cube. Puis on crée et on load une texture cube (plusieurs images), et on envoi celle-ci au fragment shader, pour faire la même chose que pour une texture classique mais avec cette fois-ci une texture cube et des coordonées 3D.

```

1 // Material.cpp
2
3 glGenVertexArrays(1, &skyboxVAO);
4 glGenBuffers(1, &skyboxVBO);
5 glBindVertexArray(skyboxVAO);
6 glBindBuffer(GL_ARRAY_BUFFER, skyboxVBO);
7 glBufferData(GL_ARRAY_BUFFER, sizeof(skyboxVertices), &skyboxVertices, GL_STATIC_DRAW);

```

```

9     glEnableVertexAttribArray(0);
10    glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 3 * sizeof(float), (void *)0);
11
12    m_texture = loadSkybox(faces);
13
14 // Texture.cpp
15
16 GLuint loadSkybox(std::vector<std::string> faces){
17    GLuint texture;
18    glGenTextures(1, &texture);
19    glBindTexture(GL_TEXTURE_CUBE_MAP, texture);
20
21    int width, height, nrChannels;
22    unsigned char *data;
23    for(unsigned int i=0; i<faces.size(); i++){
24        // faces est un tableau de string contenant les chemins des images textures de la
25        // skybox
26        data = stbi_load(faces[i].c_str(), &width, &height, &nrChannels, 0);
27        if(data){
28            stbi_set_flip_vertically_on_load(false);
29            glTexImage2D(GL_TEXTURE_CUBE_MAP_POSITIVE_X + i, 0, GL_RGB, width, height, 0,
30            GL_RGB, GL_UNSIGNED_BYTE, data);
31            stbi_image_free(data);
32        }
33        else{
34            std::cout<<"ERROR!"<<std::endl;
35            stbi_image_free(data);
36        }
37    }
38    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
39    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
40    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
41    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
42    glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_R, GL_CLAMP_TO_EDGE);
43
44    return texture;
45}

```



Figure 3: Skybox

4 Objet reflété

Dans un premier temps il faut créer 2 couples de shader un qui générera la skybox et l'autre l'objet dans la skybox. Pour cela on créé 2 programmes, et on rajoute un attribut à la fonction bind de material pour qu'il sache si il s'agit de la procédure pour la skybox ou pour l'objet on modifie l'id du program en fonction (pour cela j'ai utilisé un id utilisé et 2 autres qui correspondent aux 2 cas de figure et qui seront attribué selon le cas).

Pour le 2e couple de shader (celui de l'objet) on récupère la normale (en pensant à transformer les normales pour qu'elles soient adaptées aux transformations appliquées), on envoi la normale et la position au fragment shader qui lui calcule le rayon incident grâce à la position de la caméra. Avec la fonction reflex et en donnant en paramètre le rayon incident et la normale tout eux normalisés, cela nous donne le rayon de réflexion et on s'en sert pour récupérer la valeur du fragment à refléter.

```
1 void Material::bind(bool cube)
2 {
3     check();
4     ObjetSkybox = cube;
5     if (cube)
6     {
7         m_program = skyboxprogram;
8     }
9     else
10    {
11        m_program = sphereprogram;
12    }
13    glUseProgram(m_program);
14
15    internalBind();
16 }
17
18
19 void Material::internalBind()
20 {
21     // bind parameters
22
23     if (ObjetSkybox && m_texture != -1)
24     {
25         glDepthMask(GL_FALSE);
26         glDepthFunc(GL_LEQUAL);
27         glBindVertexArray(skyboxVAO);
28         glActiveTexture(GL_TEXTURE0);
29         glBindTexture(GL_TEXTURE_CUBE_MAP, m_texture);
30         glDrawArrays(GL_TRIANGLES, 0, 36);
31         glBindVertexArray(0);
32         glDepthFunc(GL_LESS);
33         glDepthMask(GL_TRUE);
34     }
35     else if (m_textureSphere != -1)
36     {
37         glUniform1i(getUniform("Texture"), 0);
38         glActiveTexture(GL_TEXTURE0);
39         glBindTexture(GL_TEXTURE_CUBE_MAP, m_textureSphere);
40     }
41     GLuint c_p = glGetUniformLocation(sphereprogram, "cameraPos");
42     glUniform3f(c_p, Context::camera.position[0], Context::camera.position[1], Context::camera.position[2]);
43 }
44 }
```

```

1 // vertex shader
2
3 #version 330 core
4
5 layout(location = 0) in vec3 position;
6 layout(location = 1) in vec3 normal;
7 layout(location = 2) in vec2 texCoord;
8
9 uniform mat4 model;
10 uniform mat4 view;
11 uniform mat4 projection;
12
13 out vec3 coordonneesText;
14 out vec3 normalText;
15
16 void main() {
17     normalText=mat3(transpose(inverse(model)))*normal;
18     coordonneesText=vec3(model*vec4(position,1.0));
19     gl_Position = projection * view * vec4(position.x,-position.y,-position.z,1.0);
20 }
21
22 //fragment shader
23
24 #version 330 core
25
26 out vec4 FragColor;
27
28 in vec3 coordonneesText;
29 in vec3 normalText;
30
31 uniform vec3 cameraPos;
32
33 uniform samplerCube Texture;
34
35 void main()
36 {
37     vec3 I = normalize(coordonneesText - cameraPos);
38     vec3 R = reflect(I, normalize(normalText));
39     FragColor = vec4(texture(Texture, R).rgb, 1.0);
40 }
```

5 Remarques

Ce Tp était assez long et assez compliqué à comprendre bien qu'on maîtrisait les différents éléments utilisés comme les shaders ou les buffers, des choses précises comme l'utilisation de 2 shaders. Le conseil de l'utilisation de LearOpenGL a été très utile.

Cependant long et peu de temps avec les autres travaux et les révisions je n'ai pas eu le temps d'aborder le second TP.



Figure 4: Skybox avec objet reflété