UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DIVISIÓN DE INGENIERÍA ELÉCTRICA

INGENIERÍA EN COMPUTACIÓN

LABORATORIO DE COMPUTACIÓN GRÁFICA e INTERACCIÓN HUMANO COMPUTADORA

REPORTE DE PRÁCTICA Nº 06

NOMBRE COMPLETO: LOPEZ BETANCOURT MICHELLE

Nº de Cuenta: 318309028

GRUPO DE LABORATORIO: 02

GRUPO DE TEORÍA: 06

SEMESTRE 2025-1

FECHA DE ENTREGA LÍMITE: 28 SEPTIEMBRE DE 2024

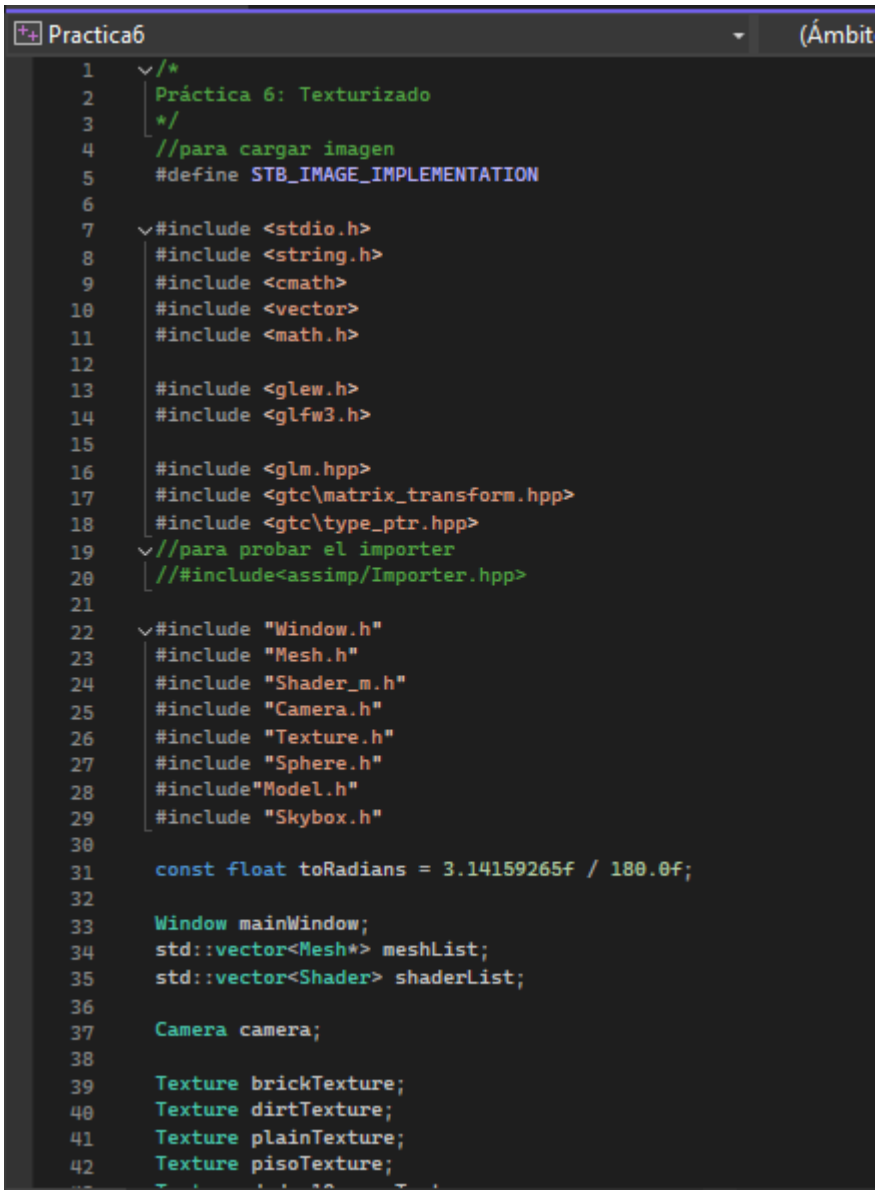
CALIFICACIÓN: _____

REPORTE DE PRÁCTICA:

1.- Ejecución de los ejercicios que se dejaron, comentar cada uno y capturas de pantalla de bloques de código generados y de ejecución del programa.

- Los ejercicios a realizar para esta práctica eran primero crear un dado que tuviera 10 caras (dodecaedro) y cada una de esas caras texturizarlas mediante código en OpenGL, como ejercicio 2 se tenía que importar el modelo de coche elegido anteriormente con sus respectivas 4 llantas acomodadas y texturizar las 4 llantas diferenciando el caucho y el rin; y por último, como tercer ejercicio se tenia que texturizar la cara del personaje de la imagen seleccionada anteriormente de tipo cars en el parabrisas (los ojos) y los detalles en el cofre y parrilla del modelo del coche.

  Código generado

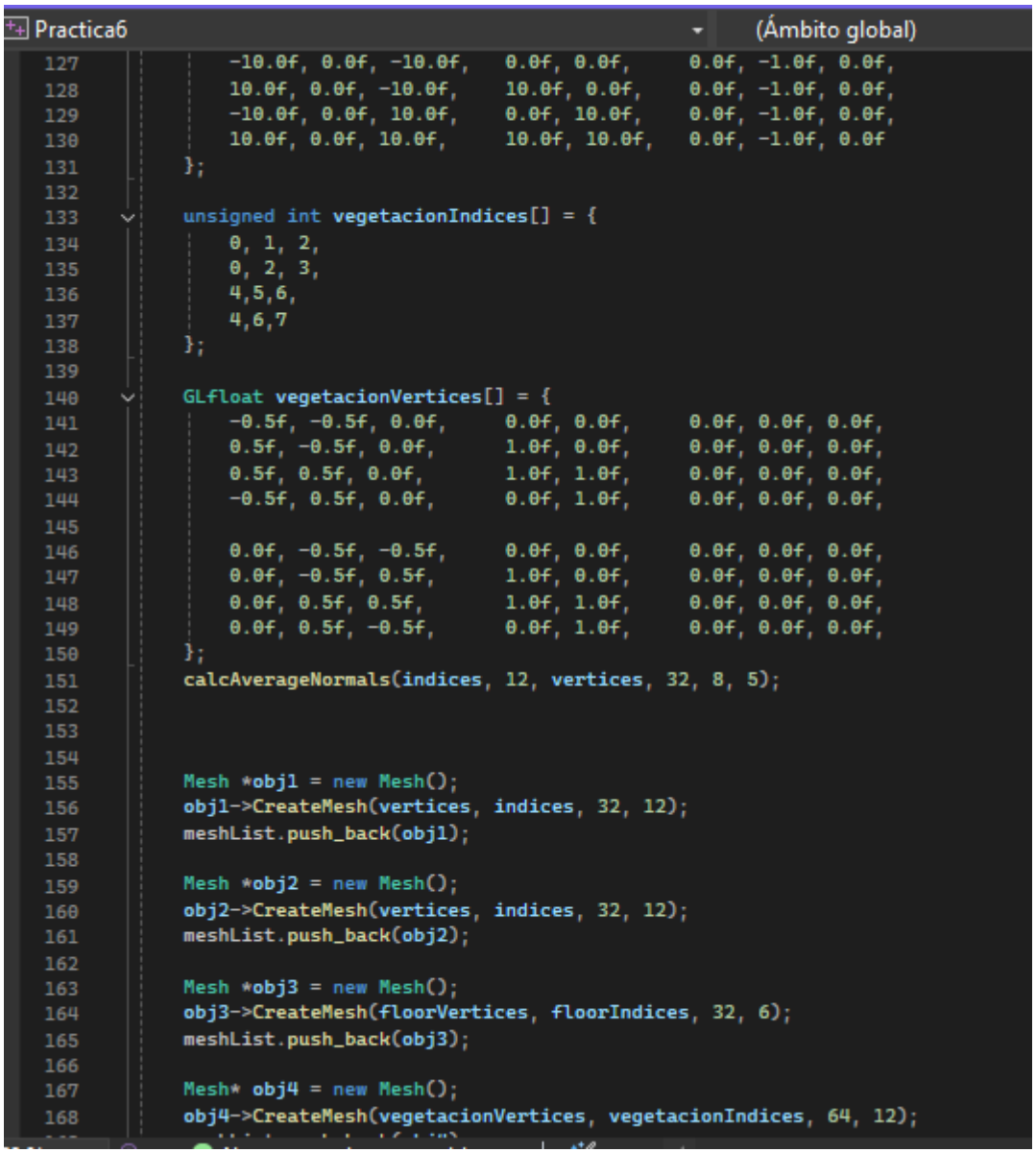

Practica6
(Ámbit
/*
Práctica 6: Texturizado
*/
//para cargar imagen
#define STB_IMAGE_IMPLEMENTATION
#include <stdio.h>
#include <string.h>
#include <cmath>
#include <vector>
#include <math.h>
#include <glew.h>
#include <glfw3.h>
#include <glm.hpp>
#include <gtc\matrix_transform.hpp>
#include <gtc\type_ptr.hpp>
//para probar el importer
//#include<assimp/Importer.hpp>
#include "Window.h"
#include "Mesh.h"
#include "Shader_m.h"
#include "Camera.h"
#include "Texture.h"
#include "Sphere.h"
#include"Model.h"
#include "Skybox.h"
const float toRadians = 3.14159265f / 180.0f;
Window mainWindow;
std::vector<Mesh*> meshList;
std::vector<Shader> shaderList;
Camera camera;
Texture brickTexture;
Texture dirtTexture;
Texture plainTexture;
Texture pisoTexture;

Practica6 (Ámbito global) main()

```
43    Texture dado_10carasTexture;
44    Texture logofiTexture;
45    Texture discoTexture;
46    Texture ruedaTexture;
47    Texture cofreTexture;
48    Texture parabrisasTexture;
49
50    Model carro_M;
51    Model Llanta_M;
52    Model Dado_M;
53    Model parabrisas_M;
54    Model cofre_M;
55
56    Skybox skybox;
57
58    //Sphere cabeza = Sphere(0.5, 20, 20);
59    GLfloat deltaTime = 0.0f;
60    GLfloat lastTime = 0.0f;
61    static double limitFPS = 1.0 / 60.0;
62
63
64    // Vertex Shader
65    static const char* vShader = "shaders/shader_texture.vert";
66
67    // Fragment Shader
68    static const char* fShader = "shaders/shader_texture.frag";
69
70
71
72
73    //cálculo del promedio de las normales para sombreado de Phong
74    void calcAverageNormals(unsigned int* indices, unsigned int indiceCount, GLfloat* vertices, unsigned int verticeCount,
75        unsigned int vLength, unsigned int normalOffset)
76    {
77        for (size_t i = 0; i < indiceCount; i += 3)
78        {
79            unsigned int in0 = indices[i] * vLength;
80            unsigned int in1 = indices[i + 1] * vLength;
81            unsigned int in2 = indices[i + 2] * vLength;
82            glm::vec3 v1(vertices[in1] - vertices[in0], vertices[in1 + 1] - vertices[in0 + 1], vertices[in1 + 2] - vertices[in0 + 2]);
83            glm::vec3 v2(vertices[in2] - vertices[in0], vertices[in2 + 1] - vertices[in0 + 1], vertices[in2 + 2] - vertices[in0 + 2]);
84            glm::vec3 normal = glm::cross(v1, v2);
```

Practica6 (Ámbito global)

```
85          normal = glm::normalize(normal);
86
87          in0 += normalOffset; in1 += normalOffset; in2 += normalOffset;
88          vertices[in0] += normal.x; vertices[in0 + 1] += normal.y; vertices[in0 + 2] += normal.z;
89          vertices[in1] += normal.x; vertices[in1 + 1] += normal.y; vertices[in1 + 2] += normal.z;
90          vertices[in2] += normal.x; vertices[in2 + 1] += normal.y; vertices[in2 + 2] += normal.z;
91      }
92
93      for (size_t i = 0; i < verticeCount / vLength; i++)
94      {
95          unsigned int nOffset = i * vLength + normalOffset;
96          glm::vec3 vec(vertices[nOffset], vertices[nOffset + 1], vertices[nOffset + 2]);
97          vec = glm::normalize(vec);
98          vertices[nOffset] = vec.x; vertices[nOffset + 1] = vec.y; vertices[nOffset + 2] = vec.z;
99      }
100 }
101
102
103
104 void CreateObjects()
105 {
106     unsigned int indices[] = {
107         0, 3, 1,
108         1, 3, 2,
109         2, 3, 0,
110         0, 1, 2
111     };
112
113     GLfloat vertices[] = {
114         //  x      y      z          u      v          nx     ny     nz
115             -1.0f, -1.0f, -0.6f,     0.0f, 0.0f,     0.0f, 0.0f, 0.0f,
116             0.0f, -1.0f, 1.0f,       0.5f, 0.0f,     0.0f, 0.0f, 0.0f,
117             1.0f, -1.0f, -0.6f,      1.0f, 0.0f,     0.0f, 0.0f, 0.0f,
118             0.0f, 1.0f, 0.0f,        0.5f, 1.0f,     0.0f, 0.0f, 0.0f
119     };
120
121     unsigned int floorIndices[] = {
122         0, 2, 1,
123         1, 2, 3
124     };
125
126     GLfloat floorVertices[] = {
```

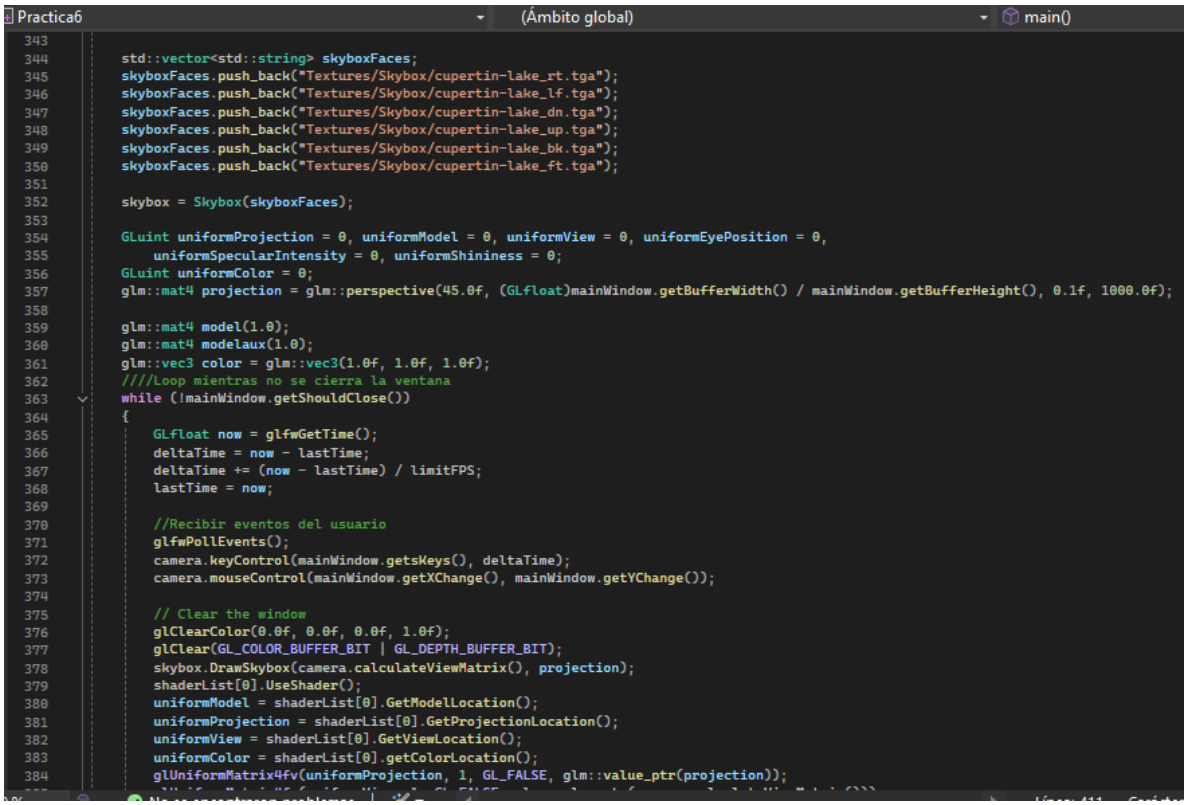

Practica6
(Ámbito global)
-10.0f, 0.0f, -10.0f, 0.0f, 0.0f, 0.0f, -1.0f, 0.0f,
10.0f, 0.0f, -10.0f, 10.0f, 0.0f, 0.0f, -1.0f, 0.0f,
-10.0f, 0.0f, 10.0f, 0.0f, 10.0f, 0.0f, -1.0f, 0.0f,
10.0f, 0.0f, 10.0f, 10.0f, 10.0f, 0.0f, -1.0f, 0.0f
};
unsigned int vegetacionIndices[] = {
0, 1, 2,
0, 2, 3,
4,5,6,
4,6,7
};
GLfloat vegetacionVertices[] = {
-0.5f, -0.5f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f,
0.5f, -0.5f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
0.5f, 0.5f, 0.0f, 1.0f, 1.0f, 0.0f, 0.0f, 0.0f,
-0.5f, 0.5f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
0.0f, -0.5f, -0.5f, 0.0f, 0.0f, 0.0f, 0.0f, 0.0f,
0.0f, -0.5f, 0.5f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f,
0.0f, 0.5f, 0.5f, 1.0f, 1.0f, 0.0f, 0.0f, 0.0f,
0.0f, 0.5f, -0.5f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f,
};
calcAverageNormals(indices, 12, vertices, 32, 8, 5);
Mesh *obj1 = new Mesh();
obj1->CreateMesh(vertices, indices, 32, 12);
meshList.push_back(obj1);
Mesh *obj2 = new Mesh();
obj2->CreateMesh(vertices, indices, 32, 12);
meshList.push_back(obj2);
Mesh *obj3 = new Mesh();
obj3->CreateMesh(floorVertices, floorIndices, 32, 6);
meshList.push_back(obj3);
Mesh* obj4 = new Mesh();
obj4->CreateMesh(vegetacionVertices, vegetacionIndices, 64, 12);

```
172
173
174     void CreateShaders()
175     {
176         Shader *shader1 = new Shader();
177         shader1->CreateFromFiles(vShader, fShader);
178         shaderList.push_back(*shader1);
179     }
180
181     void CrearDado()
182     {
183         unsigned int cubo_indices[] = {
184             //caras arriba
185             // front
186             0, 1, 2,
187
188             // back
189             3, 4, 5,
190
191             // left
192             6, 7 , 8,
193
194             // top
195             9, 10, 11,
196
197             // right
198             12, 13, 14,
199
200             //caras abajo
201             // front
202             15, 16, 17,
203
204             // back
205             18, 19, 20,
206
207             // left
208             21, 22, 23,
209
210             // top
211             24, 25, 26,
212
213             // right
```

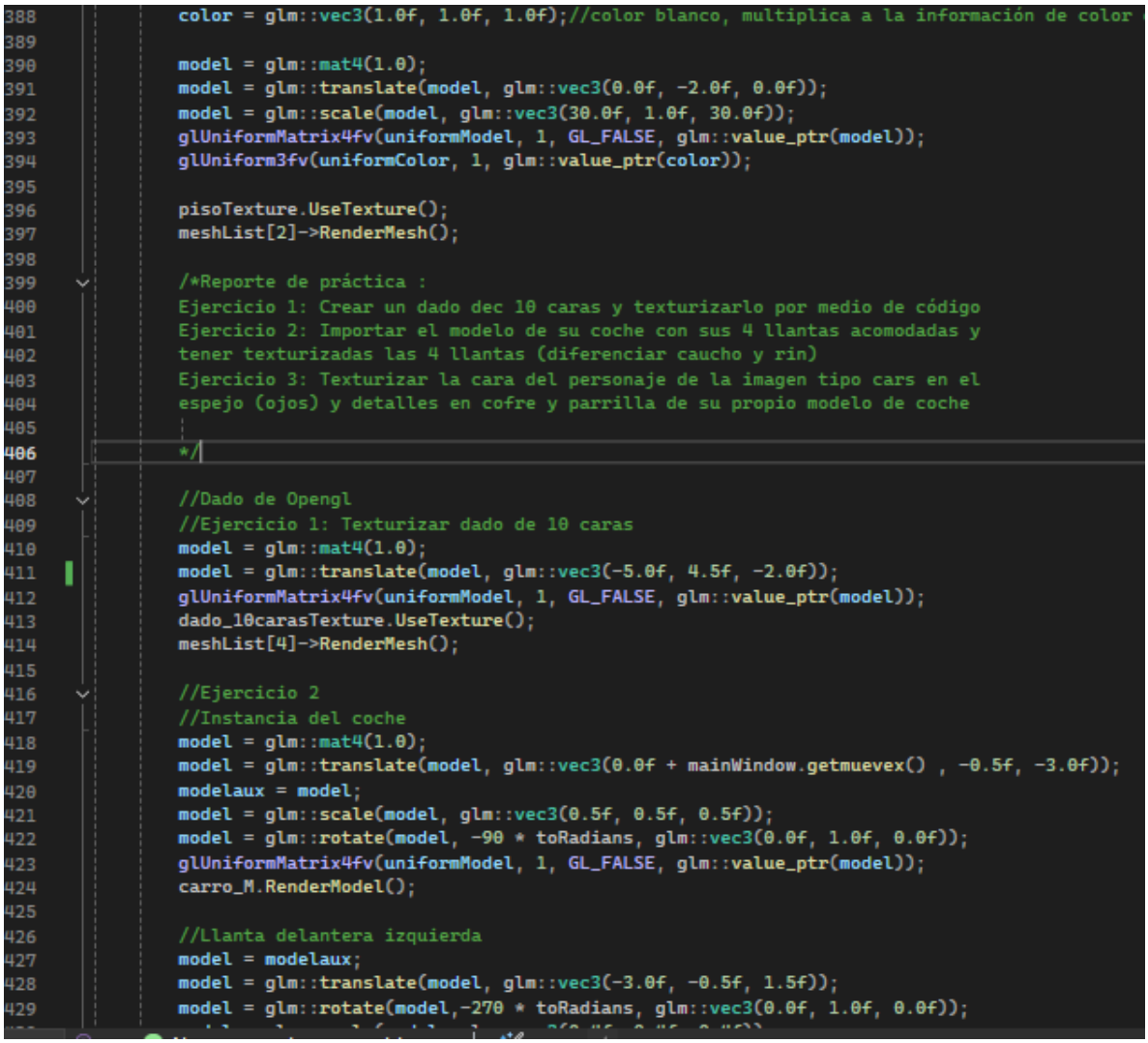
```
Practica6                                                    (Ámbito global)
217
218     };
219     //Ejercicio 1: reemplazar con sus dados de 6 caras texturizados, agregar normales
220 // average normals
221     GLfloat cubo_vertices[] = {
222
223         //caras arriba
224
225         // front
226         //x       y       z           S       T           NX      NY      NZ
227         0.0f,     2.0f,   0.0f,       0.25f,  0.85f,      0.0f,   0.0f,   -1.0f,  //0
228         0.0f,     0.0f,   -2.0f,      0.34f,  0.65f,      0.0f,   0.0f,   -1.0f,  //1
229         1.902f,   0.0f,   -0.618f,    0.15f,  0.65f,      0.0f,   0.0f,   -1.0f,  //2
230
231         // right
232         //x       y       z           S       T
233         0.0f,     2.0f,   0.0f,       0.39f,  0.85f,      -1.0f,  0.0f,   0.0f,
234         1.902f,   0.0f,   -0.618f,    0.51f,  0.65f,      -1.0f,  0.0f,   0.0f,
235         1.176f,   0.0f,   1.618f,     0.28f,  0.65f,      -1.0f,  0.0f,   0.0f,
236
237         // back
238         //x       y       z           S       T
239         0.0f,     2.0f,   0.0f,       0.57f,  0.85f,      0.0f,   0.0f,   1.0f,
240         1.176f,   0.0f,   1.618f,     0.68f,  0.65f,      0.0f,   0.0f,   1.0f,
241         -1.176f,  0.0f,   1.618f,     0.47f,  0.65f,      0.0f,   0.0f,   1.0f,
242
243
244         // left
245         //x       y       z           S       T
246         0.0f,     2.0f,   0.0f,       0.74f,  0.86f,      1.0f,   0.0f,   0.0f,
247         -1.176f,  0.0f,   1.618f,     0.85f,  0.65f,      1.0f,   0.0f,   0.0f,
248         -1.902f,  0.0f,   -0.618f,    0.62f,  0.65f,      1.0f,   0.0f,   0.0f,
249
250
251         // bottom
252         //x       y       z           S       T
253         0.0f,     2.0f,   0.0f,       0.25f,  0.65f,      0.0f,   1.0f,   0.0f,
254         -1.902f,  0.0f,   -0.618f,    0.37f,  0.45f,      0.0f,   1.0f,   0.0f,
255          0.0f,    0.0f,   -2.0f,      0.14f,  0.45f,      0.0f,   1.0f,   0.0f,
256
257          //caras abajo
258
```

```
Practica6                                                    (Ámbito global)
259         // front
260         //x       y       z          S       T          NX      NY      NZ
261         0.0f,     -2.0f,  0.0f,      0.4f,   0.65f,     0.0f,   0.0f,   -1.0f,  //0
262         0.0f,     0.0f,   -2.0f,     0.51f,  0.45f,     0.0f,   0.0f,   -1.0f,  //1
263         1.902f,   0.0f,   -0.618f,   0.3f,   0.45f,     0.0f,   0.0f,   -1.0f,  //2
264
265         // right
266         //x       y       z          S       T
267         0.0f,     -2.0f,  0.0f,      0.57f,  0.65f,     -1.0f,  0.0f,   0.0f,
268         1.902f,   0.0f,   -0.618f,   0.44f,  0.45f,     -1.0f,  0.0f,   0.0f,
269         1.176f,   0.0f,   1.618f,    0.68f,  0.45f,     -1.0f,  0.0f,   0.0f,
270
271         // back
272         //x       y       z          S       T
273         0.0f,     -2.0f,  0.0f,      0.73f,  0.65f,     0.0f,   0.0f,   1.0f,
274         1.176f,   0.0f,   1.618f,    0.84f,  0.45f,     0.0f,   0.0f,   1.0f,
275         -1.176f,  0.0f,   1.618f,    0.63f,  0.45f,     0.0f,   0.0f,   1.0f,
276
277
278         // left
279         //x       y       z          S       T
280         0.0f,     -2.0f,  0.0f,      0.392f, 0.438f,    1.0f,   0.0f,   0.0f,
281         -1.176f,  0.0f,   1.618f,    0.5f,   0.25f,     1.0f,   0.0f,   0.0f,
282         -1.902f,  0.0f,   -0.618f,   0.28f,  0.25f,     1.0f,   0.0f,   0.0f,
283
284
285         // bottom
286         //x       y       z          S       T
287         0.0f,     -2.0f,  0.0f,      0.55f,  0.44f,     0.0f,   1.0f,   0.0f,
288         -1.902f,  0.0f,   -0.618f,   0.75f,  0.25f,     0.0f,   1.0f,   0.0f,
289          0.0f,    0.0f,   -2.0f,     0.39f,  0.25f,     0.0f,   1.0f,   0.0f,
290
291
292
293     };
294
295     Mesh* dado = new Mesh();
296     dado->CreateMesh(cubo_vertices, cubo_indices, 240, 30);
297     meshList.push_back(dado);
298
299 }
300
```

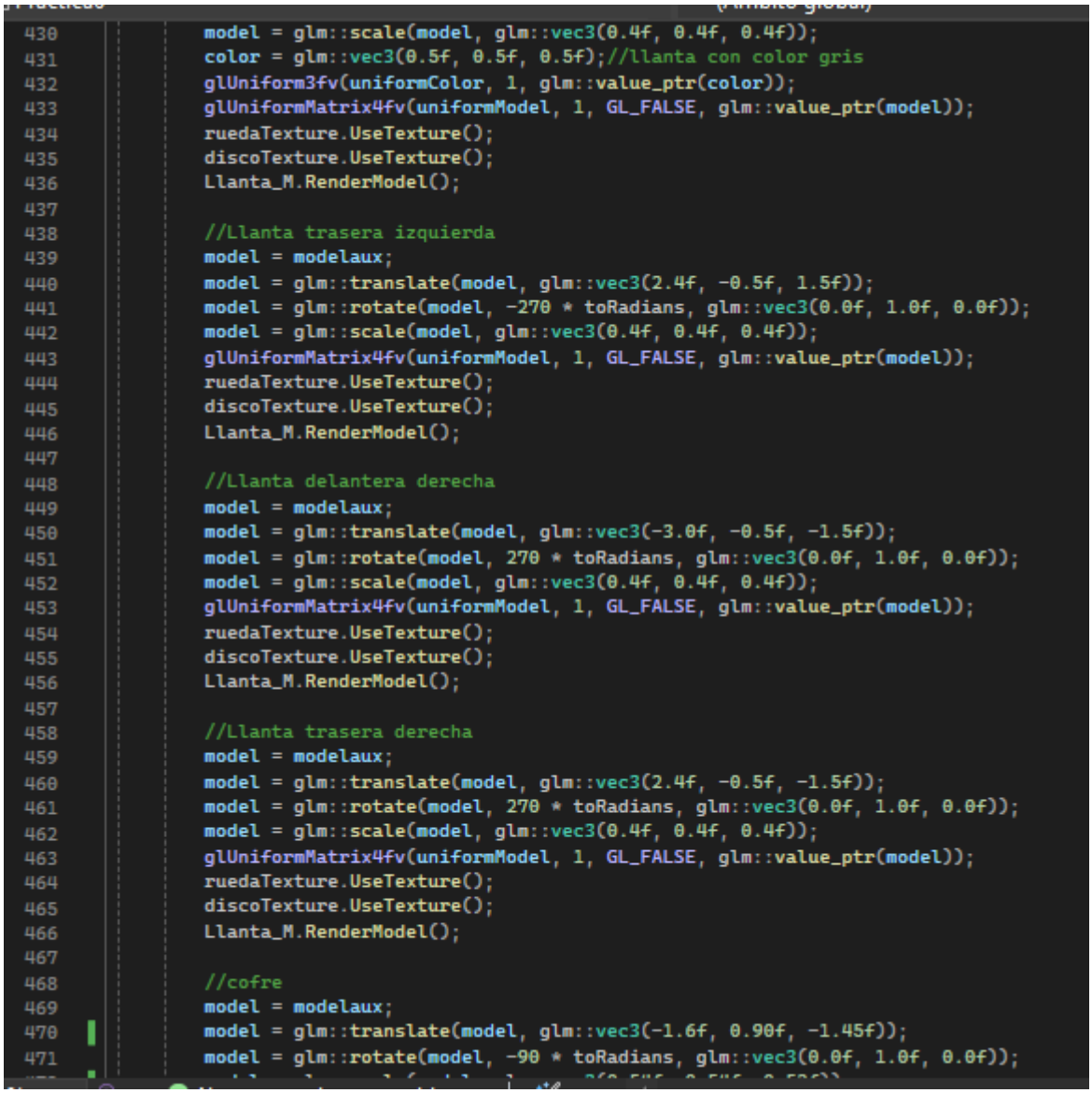
```
Practica0                                                    (Ámbito global)
301
302
303   int main()
304   {
305       mainWindow = Window(1366, 768); // 1280, 1024 or 1024, 768
306       mainWindow.Initialise();
307
308       CreateObjects();
309       CrearDado();
310       CreateShaders();
311
312       camera = Camera(glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(0.0f, 1.0f, 0.0f), -60.0f, 0.0f, 0.3f, 0.5f);
313
314       brickTexture = Texture("Textures/brick.png");
315       brickTexture.LoadTextureA();
316       dirtTexture = Texture("Textures/dirt.png");
317       dirtTexture.LoadTextureA();
318       plainTexture = Texture("Textures/plain.png");
319       plainTexture.LoadTextureA();
320       pisoTexture = Texture("Textures/piso.tga");
321       pisoTexture.LoadTextureA();
322       logofiTexture = Texture("Textures/escudo_fi_color.tga");
323       logofiTexture.LoadTextureA();
324       dado_10carasTexture = Texture("Textures/numeros-1-al-10.png");
325       dado_10carasTexture.LoadTextureA();
326       discoTexture = Texture("Textures/discoR.png");
327       discoTexture.LoadTextureA();
328       ruedaTexture = Texture("Textures/rueda.png");
329       ruedaTexture.LoadTextureA();
330       cofreTexture = Texture("Textures/cofre.png");
331       cofreTexture.LoadTextureA();
332       parabrisasTexture = Texture("Textures/cruz_ojos.png");
333
334       carro_M = Model();
335       carro_M.LoadModel("Models/carro.obj");
336       Llanta_M = Model();
337       Llanta_M.LoadModel("Models/llanta.obj");
338       parabrisas_M = Model();
339       parabrisas_M.LoadModel("Models/parabrisas.obj");
340       cofre_M = Model();
341       cofre_M.LoadModel("Models/cofre.obj");
342
```




```
343
344     std::vector<std::string> skyboxFaces;
345     skyboxFaces.push_back("Textures/Skybox/cupertin-lake_rt.tga");
346     skyboxFaces.push_back("Textures/Skybox/cupertin-lake_lf.tga");
347     skyboxFaces.push_back("Textures/Skybox/cupertin-lake_dn.tga");
348     skyboxFaces.push_back("Textures/Skybox/cupertin-lake_up.tga");
349     skyboxFaces.push_back("Textures/Skybox/cupertin-lake_bk.tga");
350     skyboxFaces.push_back("Textures/Skybox/cupertin-lake_ft.tga");
351
352     skybox = Skybox(skyboxFaces);
353
354     GLuint uniformProjection = 0, uniformModel = 0, uniformView = 0, uniformEyePosition = 0,
355         uniformSpecularIntensity = 0, uniformShininess = 0;
356     GLuint uniformColor = 0;
357     glm::mat4 projection = glm::perspective(45.0f, (GLfloat)mainWindow.getBufferWidth() / mainWindow.getBufferHeight(), 0.1f, 1000.0f);
358
359     glm::mat4 model(1.0);
360     glm::mat4 modelaux(1.0);
361     glm::vec3 color = glm::vec3(1.0f, 1.0f, 1.0f);
362     ////Loop mientras no se cierra la ventana
363     while (!mainWindow.getShouldClose())
364     {
365         GLfloat now = glfwGetTime();
366         deltaTime = now - lastTime;
367         deltaTime += (now - lastTime) / limitFPS;
368         lastTime = now;
369
370         //Recibir eventos del usuario
371         glfwPollEvents();
372         camera.keyControl(mainWindow.getsKeys(), deltaTime);
373         camera.mouseControl(mainWindow.getXChange(), mainWindow.getYChange());
374
375         // Clear the window
376         glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
377         glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
378         skybox.DrawSkybox(camera.calculateViewMatrix(), projection);
379         shaderList[0].UseShader();
380         uniformModel = shaderList[0].GetModelLocation();
381         uniformProjection = shaderList[0].GetProjectionLocation();
382         uniformView = shaderList[0].GetViewLocation();
383         uniformColor = shaderList[0].getColorLocation();
384         glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
```



```
388         color = glm::vec3(1.0f, 1.0f, 1.0f);//color blanco, multiplica a la información de color
389
390         model = glm::mat4(1.0);
391         model = glm::translate(model, glm::vec3(0.0f, -2.0f, 0.0f));
392         model = glm::scale(model, glm::vec3(30.0f, 1.0f, 30.0f));
393         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
394         glUniform3fv(uniformColor, 1, glm::value_ptr(color));
395
396         pisoTexture.UseTexture();
397         meshList[2]->RenderMesh();
398
399         /*Reporte de práctica :
400         Ejercicio 1: Crear un dado dec 10 caras y texturizarlo por medio de código
401         Ejercicio 2: Importar el modelo de su coche con sus 4 llantas acomodadas y
402         tener texturizadas las 4 llantas (diferenciar caucho y rin)
403         Ejercicio 3: Texturizar la cara del personaje de la imagen tipo cars en el
404         espejo (ojos) y detalles en cofre y parrilla de su propio modelo de coche
405
406         */
407
408         //Dado de Opengl
409         //Ejercicio 1: Texturizar dado de 10 caras
410         model = glm::mat4(1.0);
411         model = glm::translate(model, glm::vec3(-5.0f, 4.5f, -2.0f));
412         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
413         dado_10carasTexture.UseTexture();
414         meshList[4]->RenderMesh();
415
416         //Ejercicio 2
417         //Instancia del coche
418         model = glm::mat4(1.0);
419         model = glm::translate(model, glm::vec3(0.0f + mainWindow.getmuevex() , -0.5f, -3.0f));
420         modelaux = model;
421         model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
422         model = glm::rotate(model, -90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
423         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
424         carro_M.RenderModel();
425
426         //Llanta delantera izquierda
427         model = modelaux;
428         model = glm::translate(model, glm::vec3(-3.0f, -0.5f, 1.5f));
429         model = glm::rotate(model,-270 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
```



```
430         model = glm::scale(model, glm::vec3(0.4f, 0.4f, 0.4f));
431         color = glm::vec3(0.5f, 0.5f, 0.5f);//llanta con color gris
432         glUniform3fv(uniformColor, 1, glm::value_ptr(color));
433         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
434         ruedaTexture.UseTexture();
435         discoTexture.UseTexture();
436         Llanta_M.RenderModel();
437
438         //Llanta trasera izquierda
439         model = modelaux;
440         model = glm::translate(model, glm::vec3(2.4f, -0.5f, 1.5f));
441         model = glm::rotate(model, -270 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
442         model = glm::scale(model, glm::vec3(0.4f, 0.4f, 0.4f));
443         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
444         ruedaTexture.UseTexture();
445         discoTexture.UseTexture();
446         Llanta_M.RenderModel();
447
448         //Llanta delantera derecha
449         model = modelaux;
450         model = glm::translate(model, glm::vec3(-3.0f, -0.5f, -1.5f));
451         model = glm::rotate(model, 270 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
452         model = glm::scale(model, glm::vec3(0.4f, 0.4f, 0.4f));
453         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
454         ruedaTexture.UseTexture();
455         discoTexture.UseTexture();
456         Llanta_M.RenderModel();
457
458         //Llanta trasera derecha
459         model = modelaux;
460         model = glm::translate(model, glm::vec3(2.4f, -0.5f, -1.5f));
461         model = glm::rotate(model, 270 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
462         model = glm::scale(model, glm::vec3(0.4f, 0.4f, 0.4f));
463         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
464         ruedaTexture.UseTexture();
465         discoTexture.UseTexture();
466         Llanta_M.RenderModel();
467
468         //cofre
469         model = modelaux;
470         model = glm::translate(model, glm::vec3(-1.6f, 0.90f, -1.45f));
471         model = glm::rotate(model, -90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
```

```
468             //cofre
469             model = modelaux;
470             model = glm::translate(model, glm::vec3(-1.6f, 0.90f, -1.45f));
471             model = glm::rotate(model, -90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
472             model = glm::scale(model, glm::vec3(0.54f, 0.54f, 0.52f));
473             glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
474             cofreTexture.UseTexture();
475             cofre_M.RenderModel();
476
477             //parabrisas
478             model = modelaux;
479             model = glm::translate(model, glm::vec3(-1.1f, 1.42f, -0.15f));
480             model = glm::rotate(model, -90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
481             model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
482             glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
483             parabrisasTexture.UseTexture();
484             parabrisas_M.RenderModel();
485
486             glUseProgram(0);
487
488             mainWindow.swapBuffers();
489         }
490
491         return 0;
492     }
```

Ejecución del programa

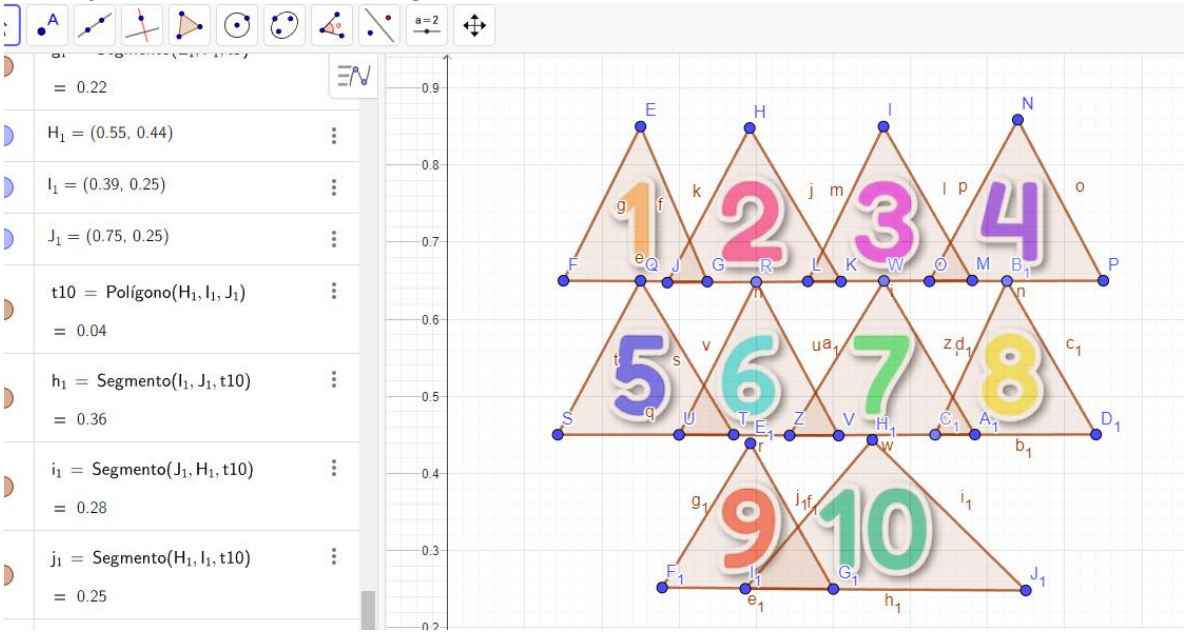

Practica 06: Texturizado
43
DINOCO
Practica 06: Texturizado
DINOCO

2.- Liste los problemas que tuvo a la hora de hacer estos ejercicios y si los resolvió explicar cómo fue, en caso de error adjuntar captura de pantalla

- El primer problema que tuve fue al momento de realizar el dado con forma de dodecaedro, ya que no sabia bien como acomodar y en que posición iban cada uno de los vértices, pero esto lo resolví mediante GeoGebra agregando una imagen de referencia y puniendo cada uno de los puntos en el plano, de esta manera logré definir cada vértice de la figura.

- El siguiente problema fue igual a la hora de darle textura al dado de 10 caras mediante una imagen, para lograr saber cuáles eran los vértices de S y T

agregue la imagen a GeoGebra y la escale quedando en (0,0) y (1,0) y fui agregando triángulos con sus respectivos vértices para saber cuáles iban a ser S y T, de esta manera logre definirlos.



- El siguiente problema fue al momento de crear los modelos en 3DMAX y modificarlos para agregarles la textura, ya que sobre todo para la parte del disco de las llantas no me cargaba la textura y lo tuve que repetir varias veces, hasta que logré realizarlo de manera correcta e importarlo para usarlo en el código.

3.- Conclusión:

a. Los ejercicios del reporte: Complejidad, Explicación.
Después de haber realizado los ejercicios propuestos para esta práctica puedo concluir que logre realizarlos con éxito, ya que comprendí la manera de dar textura a los objetos mediante la implementación de código y mediante 3DMAX así como pude observar la importancia de cada una de las partes usadas en el código, lo que hicimos durante estos ejercicios fue primero para el dado de 10 caras crear los vértices de la figura con ayuda de GeoGebra para mayor facilidad, después se eligió una imagen con números del 1 al 10 para que fuera la textura que se iba a ver en cada cara del dado y también en GeoGebra se obtuvieron los vértices para S y T que estos son los encargados de que se ve en cada cara.
Para el caso del segundo ejercicio lo que se hizo fue importar el modelo de coche elegido anteriormente y se texturizo las llantas del carro para que se observara el caucho y el rin, por ultimo para el caso del tercer ejercicio primero con la imagen tipo cars se texturizo

mediante 3DMAX el cofre y el parabrisas y se exportaron cada uno de estos objetos, en el código se crearon y cargaron cada uno de los modelos y texturas creados para que se lograran visualizar.

La complejidad al realizar los ejercicios para el dado y el coche creo no fue muy difícil ya que es algo que he realizado durante practicas y ejercicios anteriores, lo único es que si fue algo largo su procedimiento sobre todo para calcular todos los datos para realizar la textura mediante código.

b. Comentarios generales: Faltó explicar a detalle, ir más lento en alguna explicación, otros comentarios y sugerencias para mejorar desarrollo de la práctica
   Por mi parte puedo concluir que por el momento no tengo algún comentario adicional que brindar, ya que a mi punto de vista el profesor explico de manera bastante clara cada una de las instrucciones del código utilizado, asi como su funcionalidad.

c. Conclusión
   Después de a ver concluido con la practica puedo mencionar que se cumplió con los objetivos propuestos, debido a que comprendí el uso correcto de modelos y texturas, ya que mediante las actividades propuestas pude poner en practica estos conocimientos adquiridos y de esta manera cumplir con los ejercicios planteados.