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 04

NOMBRE COMPLETO: LOPEZ BETANCOURT MICHELLE

Nº de Cuenta: 318309028

GRUPO DE LABORATORIO: 02

GRUPO DE TEORÍA: 06

SEMESTRE 2025-1

FECHA DE ENTREGA LÍMITE: 7 DE SEPTIMEBRE DE 2024

CALIFICACIÓN: _____

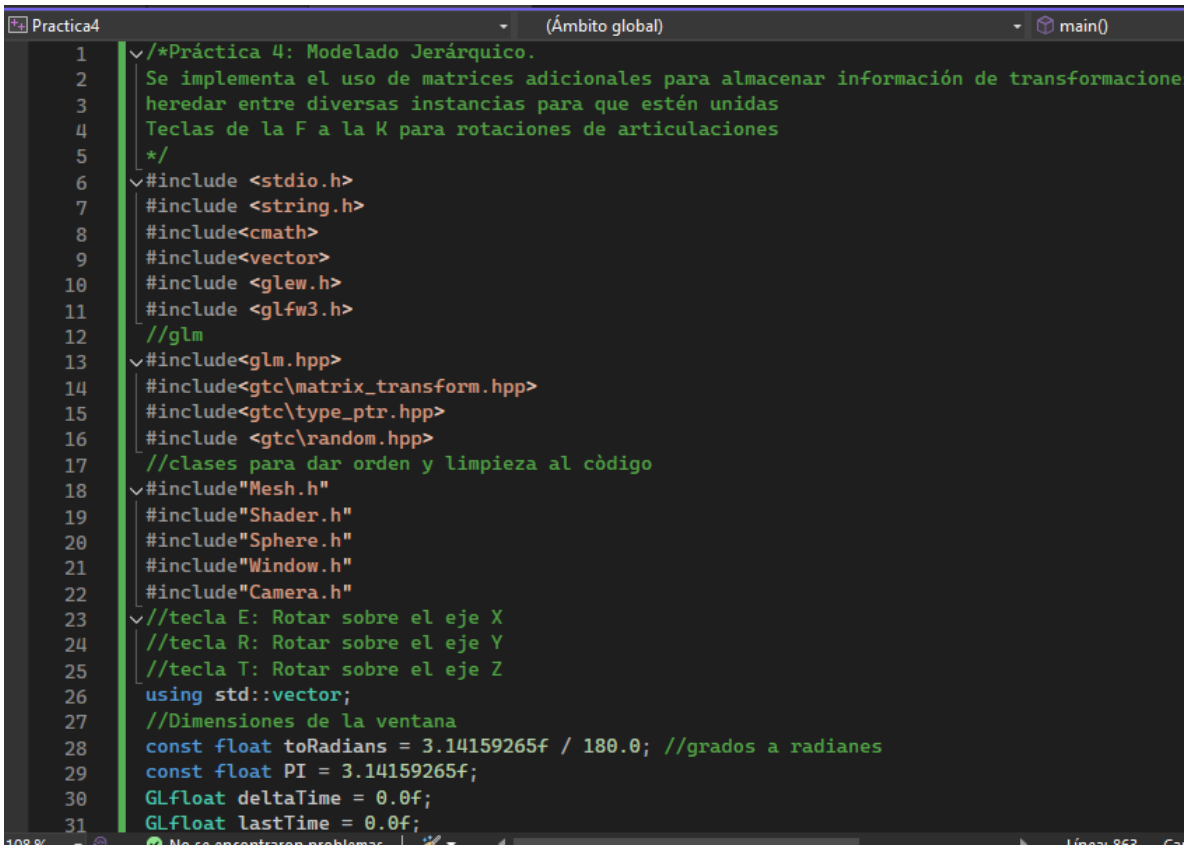
REPORTE DE PRÁCTICA:

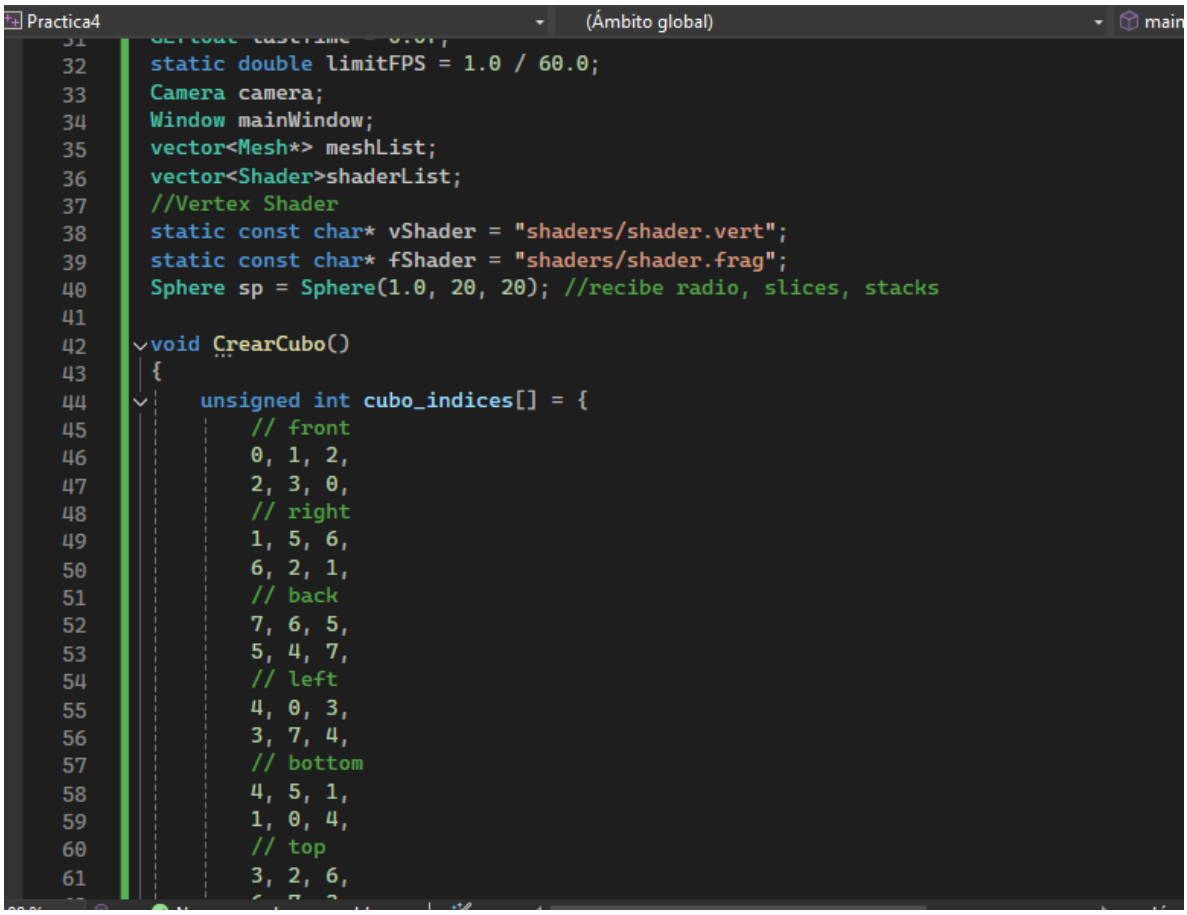
1.- Ejecución de los ejercicios que se dejaron, comentar cada uno y capturas de pantalla de bloques de código generados y de ejecución del programa.

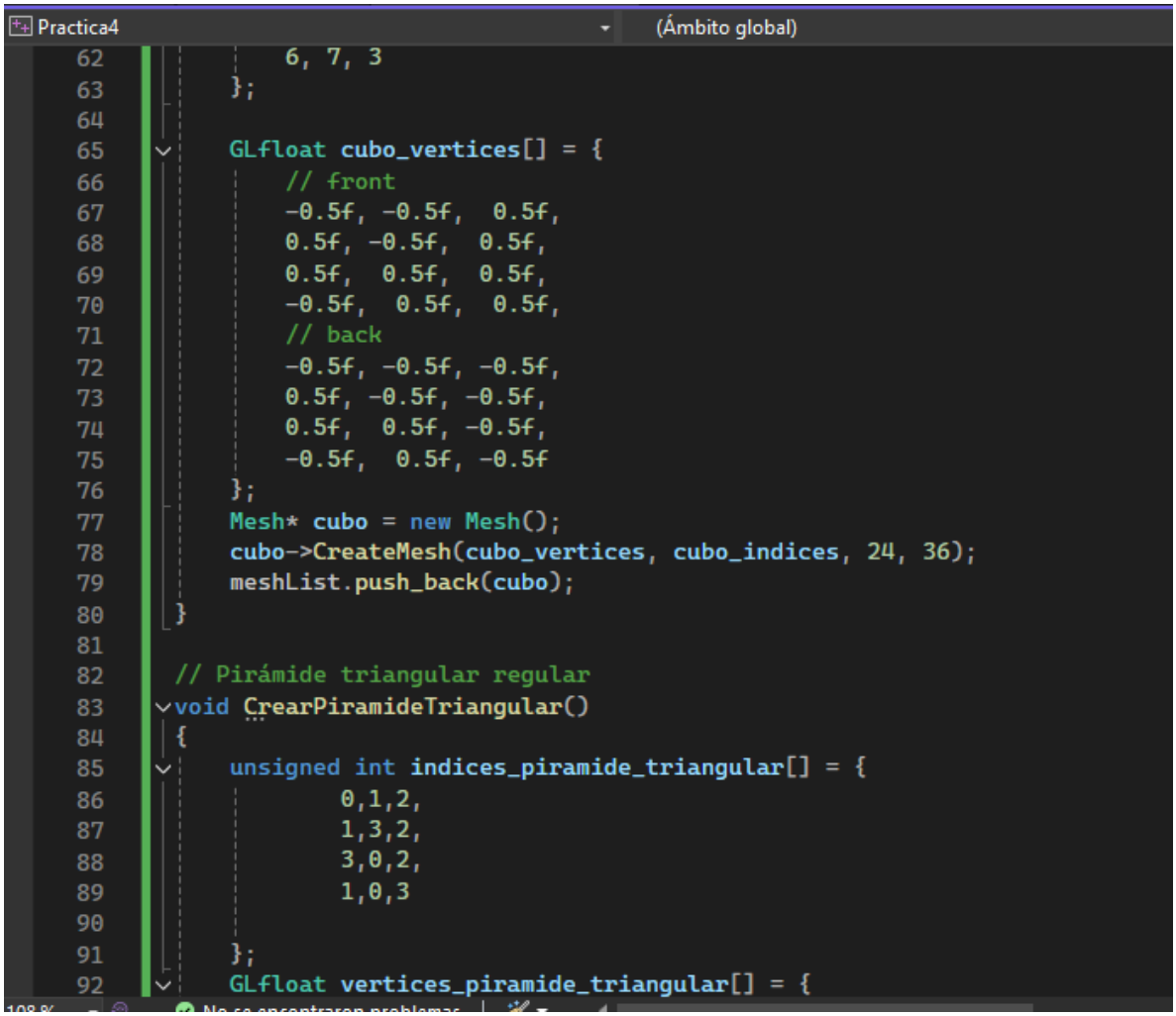
- Mediante el uso de jerarquías terminar de construir la siguiente grúa. Con ayuda de un prisma rectangular realizar el cuerpo de la grúa y construir tres brazos con sus 4 articulaciones para el movimiento, una canasta y sus respectivas 4 llantas en la parte inferior teniendo en cuenta que cada una debe tener movimiento, como se muestra en la siguiente imagen.

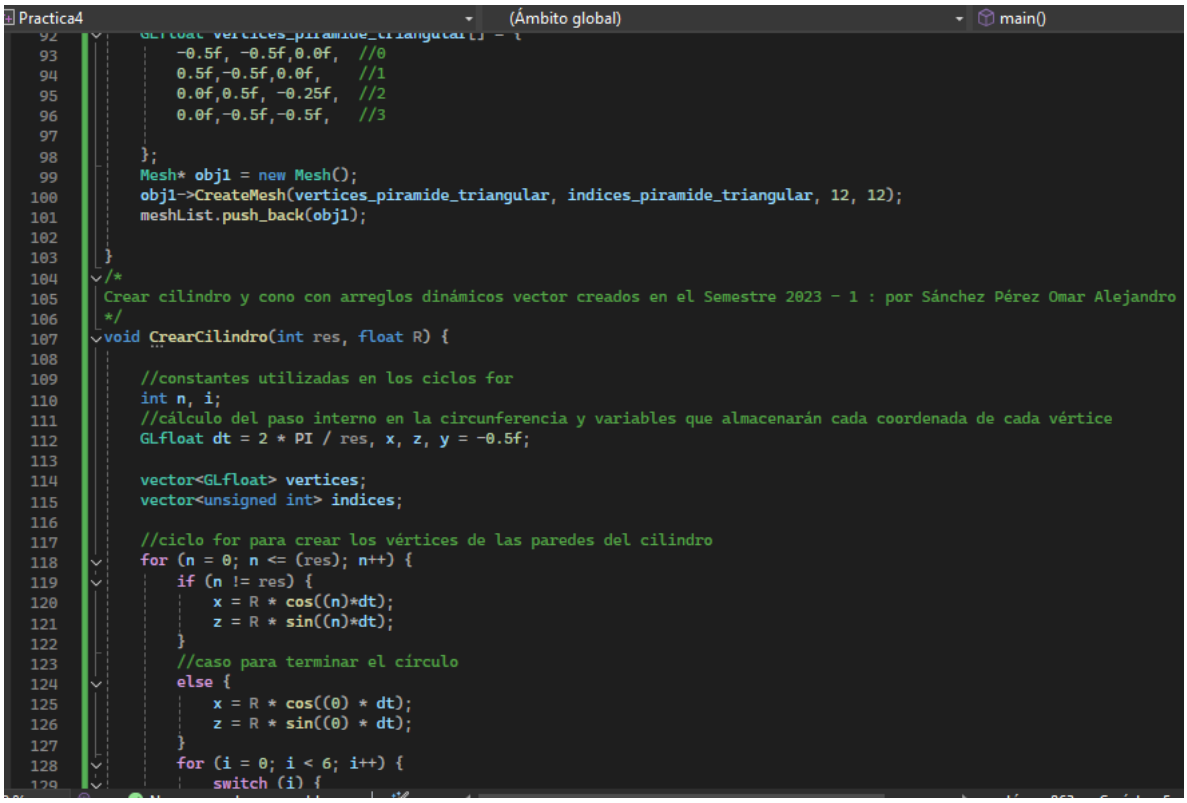
El segundo ejercicio consta en crear un animal robot en 3d instanciando cubos, pirámides, cilindros, conos, esferas, debe contar ya sea con 4 patas articuladas en 2 partes, las cuales se podrán mover mediante el teclado o con una cola y dos orejas articuladas que también se podrán mover mediante el teclado la cola y cada oreja.

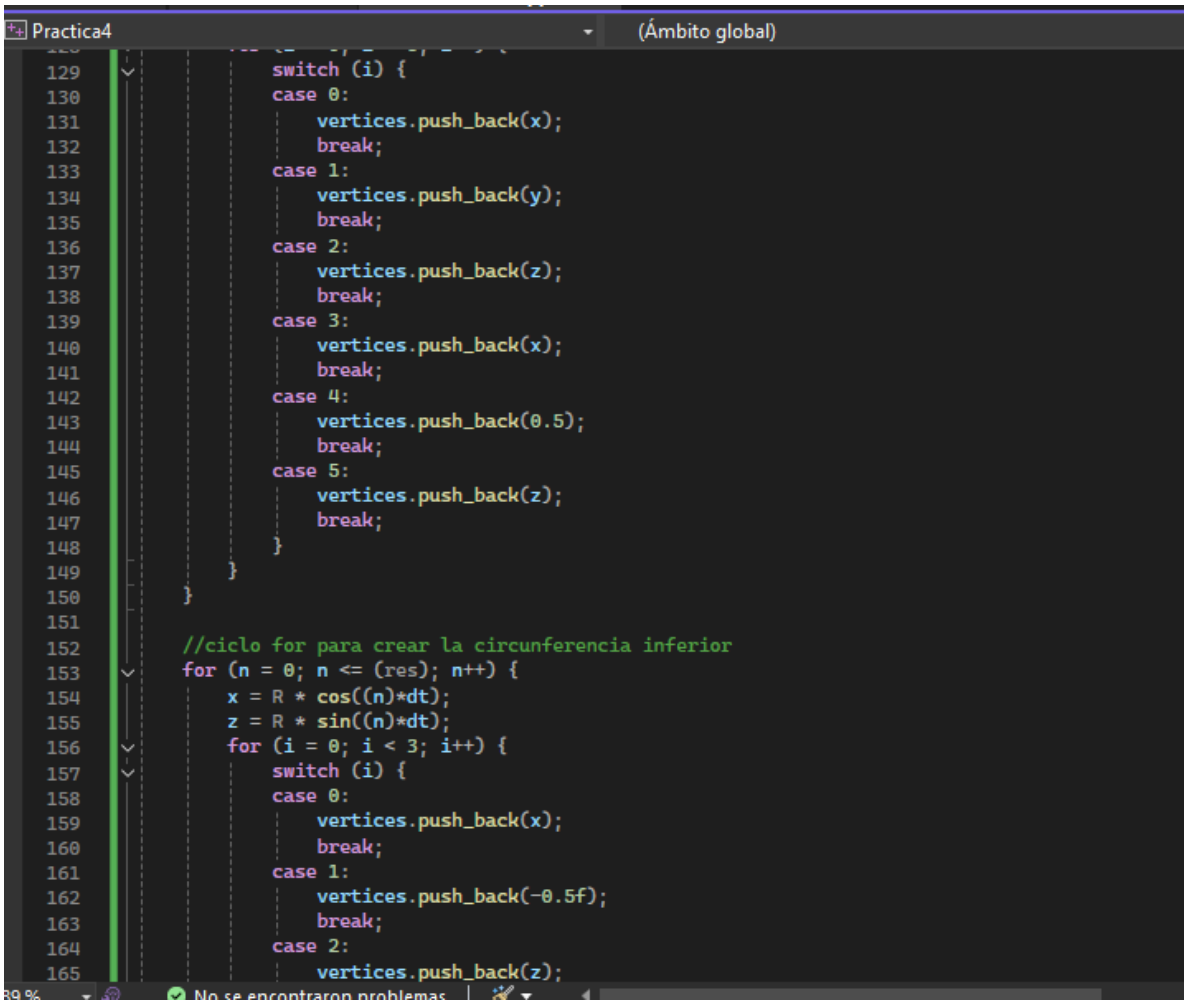
Código generado

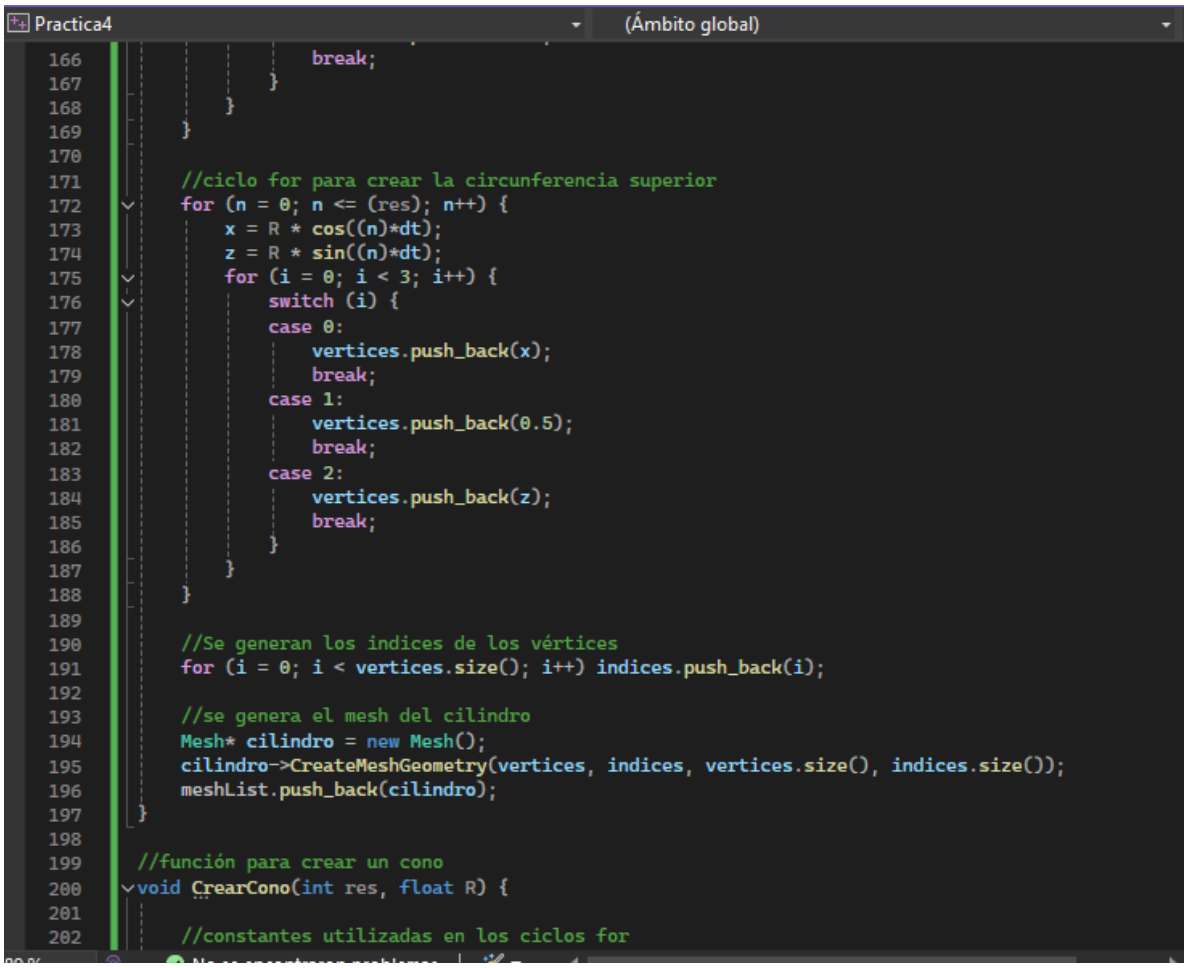

Practica4
(Ámbito global)
main()
/*Práctica 4: Modelado Jerárquico.
Se implementa el uso de matrices adicionales para almacenar información de transformacione
heredar entre diversas instancias para que estén unidas
Teclas de la F a la K para rotaciones de articulaciones
*/
#include <stdio.h>
#include <string.h>
#include<cmath>
#include<vector>
#include <glew.h>
#include <glfw3.h>
//glm
#include<glm.hpp>
#include<gtc\matrix_transform.hpp>
#include<gtc\type_ptr.hpp>
#include <gtc\random.hpp>
//clases para dar orden y limpieza al còdigo
#include"Mesh.h"
#include"Shader.h"
#include"Sphere.h"
#include"Window.h"
#include"Camera.h"
//tecla E: Rotar sobre el eje X
//tecla R: Rotar sobre el eje Y
//tecla T: Rotar sobre el eje Z
using std::vector;
//Dimensiones de la ventana
const float toRadians = 3.14159265f / 180.0; //grados a radianes
const float PI = 3.14159265f;
GLfloat deltaTime = 0.0f;
GLfloat lastTime = 0.0f;

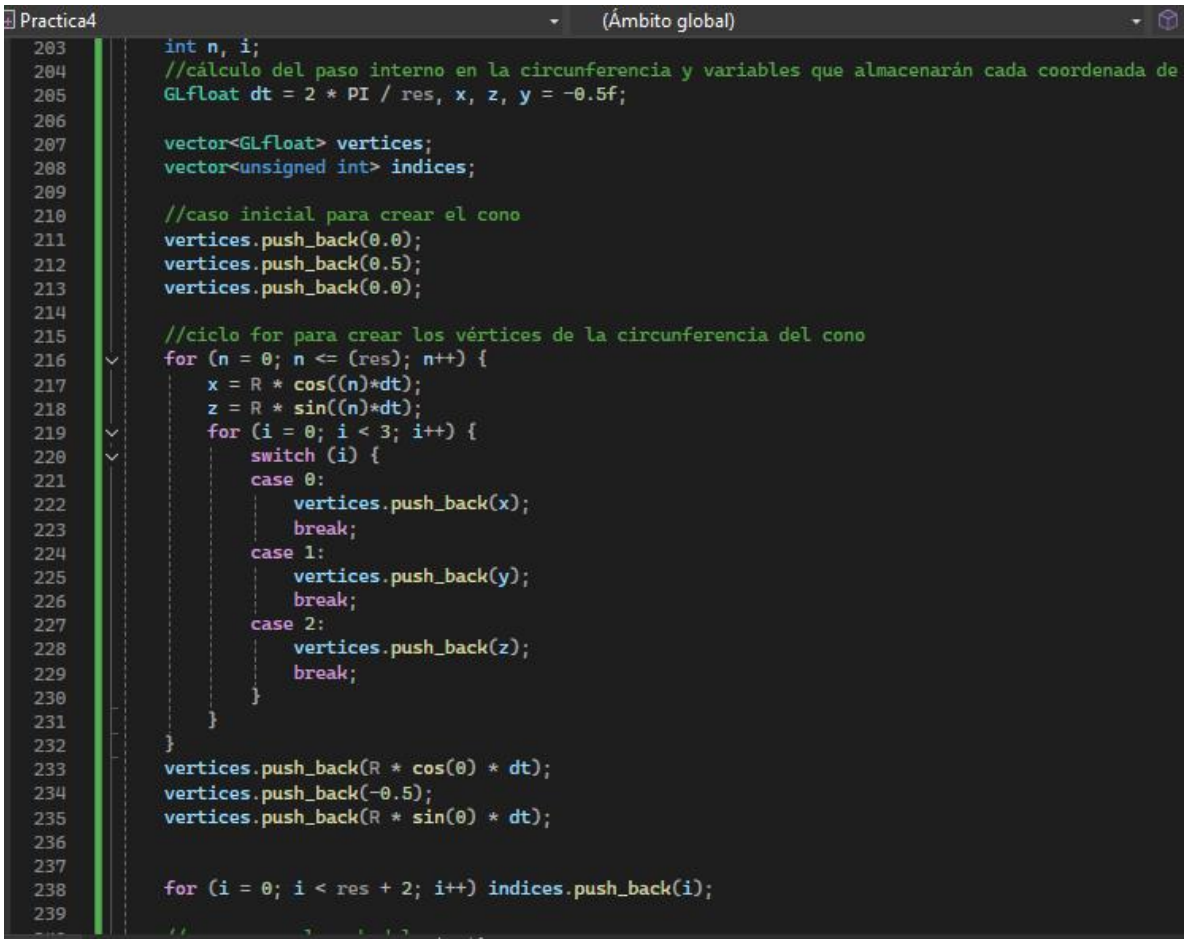

Practica4
(Ámbito global)
main
static double limitFPS = 1.0 / 60.0;
Camera camera;
Window mainWindow;
vector<Mesh*> meshList;
vector<Shader>shaderList;
//Vertex Shader
static const char* vShader = "shaders/shader.vert";
static const char* fShader = "shaders/shader.frag";
Sphere sp = Sphere(1.0, 20, 20); //recibe radio, slices, stacks
void CrearCubo()
{
unsigned int cubo_indices[] = {
// front
0, 1, 2,
2, 3, 0,
// right
1, 5, 6,
6, 2, 1,
// back
7, 6, 5,
5, 4, 7,
// left
4, 0, 3,
3, 7, 4,
// bottom
4, 5, 1,
1, 0, 4,
// top
3, 2, 6,


Practica4
(Ámbito global)
62 6, 7, 3
63 };
64
65 GLfloat cubo_vertices[] = {
66 // front
67 -0.5f, -0.5f, 0.5f,
68 0.5f, -0.5f, 0.5f,
69 0.5f, 0.5f, 0.5f,
70 -0.5f, 0.5f, 0.5f,
71 // back
72 -0.5f, -0.5f, -0.5f,
73 0.5f, -0.5f, -0.5f,
74 0.5f, 0.5f, -0.5f,
75 -0.5f, 0.5f, -0.5f
76 };
77 Mesh* cubo = new Mesh();
78 cubo->CreateMesh(cubo_vertices, cubo_indices, 24, 36);
79 meshList.push_back(cubo);
80 }
81
82 // Pirámide triangular regular
83 void CrearPiramideTriangular()
84 {
85 unsigned int indices_piramide_triangular[] = {
86 0,1,2,
87 1,3,2,
88 3,0,2,
89 1,0,3
90
91 };
92 GLfloat vertices_piramide_triangular[] = {


Practica4
(Ámbito global)
main()
-0.5f, -0.5f,0.0f, //0
0.5f,-0.5f,0.0f, //1
0.0f,0.5f, -0.25f, //2
0.0f,-0.5f,-0.5f, //3
};
Mesh* obj1 = new Mesh();
obj1->CreateMesh(vertices_piramide_triangular, indices_piramide_triangular, 12, 12);
meshList.push_back(obj1);
}
/*
Crear cilindro y cono con arreglos dinámicos vector creados en el Semestre 2023 - 1 : por Sánchez Pérez Omar Alejandro
*/
void CrearCilindro(int res, float R) {
//constantes utilizadas en los ciclos for
int n, i;
//cálculo del paso interno en la circunferencia y variables que almacenarán cada coordenada de cada vértice
GLfloat dt = 2 * PI / res, x, z, y = -0.5f;
vector<GLfloat> vertices;
vector<unsigned int> indices;
//ciclo for para crear los vértices de las paredes del cilindro
for (n = 0; n <= (res); n++) {
if (n != res) {
x = R * cos((n)*dt);
z = R * sin((n)*dt);
}
//caso para terminar el circulo
else {
x = R * cos((0) * dt);
z = R * sin((0) * dt);
}
for (i = 0; i < 6; i++) {
switch (i) {


Practica4
(Ámbito global)
switch (i) {
case 0:
vertices.push_back(x);
break;
case 1:
vertices.push_back(y);
break;
case 2:
vertices.push_back(z);
break;
case 3:
vertices.push_back(x);
break;
case 4:
vertices.push_back(0.5);
break;
case 5:
vertices.push_back(z);
break;
}
}
}
//ciclo for para crear la circunferencia inferior
for (n = 0; n <= (res); n++) {
x = R * cos((n)*dt);
z = R * sin((n)*dt);
for (i = 0; i < 3; i++) {
switch (i) {
case 0:
vertices.push_back(x);
break;
case 1:
vertices.push_back(-0.5f);
break;
case 2:
vertices.push_back(z);
No se encontraron problemas


Practica4
(Ámbito global)
166 break;
167 }
168 }
169 }
170
171 //ciclo for para crear la circunferencia superior
172 for (n = 0; n <= (res); n++) {
173 x = R * cos((n)*dt);
174 z = R * sin((n)*dt);
175 for (i = 0; i < 3; i++) {
176 switch (i) {
177 case 0:
178 vertices.push_back(x);
179 break;
180 case 1:
181 vertices.push_back(0.5);
182 break;
183 case 2:
184 vertices.push_back(z);
185 break;
186 }
187 }
188 }
189
190 //Se generan los indices de los vértices
191 for (i = 0; i < vertices.size(); i++) indices.push_back(i);
192
193 //se genera el mesh del cilindro
194 Mesh* cilindro = new Mesh();
195 cilindro->CreateMeshGeometry(vertices, indices, vertices.size(), indices.size());
196 meshList.push_back(cilindro);
197 }
198
199 //función para crear un cono
200 void CrearCono(int res, float R) {
201
202 //constantes utilizadas en los ciclos for




```
203     int n, i;
204     //cálculo del paso interno en la circunferencia y variables que almacenarán cada coordenada de
205     GLfloat dt = 2 * PI / res, x, z, y = -0.5f;
206
207     vector<GLfloat> vertices;
208     vector<unsigned int> indices;
209
210     //caso inicial para crear el cono
211     vertices.push_back(0.0);
212     vertices.push_back(0.5);
213     vertices.push_back(0.0);
214
215     //ciclo for para crear los vértices de la circunferencia del cono
216     for (n = 0; n <= (res); n++) {
217         x = R * cos((n)*dt);
218         z = R * sin((n)*dt);
219         for (i = 0; i < 3; i++) {
220             switch (i) {
221             case 0:
222                 vertices.push_back(x);
223                 break;
224             case 1:
225                 vertices.push_back(y);
226                 break;
227             case 2:
228                 vertices.push_back(z);
229                 break;
230             }
231         }
232     }
233     vertices.push_back(R * cos(0) * dt);
234     vertices.push_back(-0.5);
235     vertices.push_back(R * sin(0) * dt);
236
237
238     for (i = 0; i < res + 2; i++) indices.push_back(i);
239
```

```
Practica4                                   (Ámbito global)                    main()
240     //se genera el mesh del cono
241     Mesh* cono = new Mesh();
242     cono->CreateMeshGeometry(vertices, indices, vertices.size(), res + 2);
243     meshList.push_back(cono);
244 }
245
246 //función para crear pirámide cuadrangular unitaria
247 void CrearPiramideCuadrangular()
248 {
249     vector<unsigned int> piramidecuadrangular_indices = {
250         0,3,4,
251         3,2,4,
252         2,1,4,
253         1,0,4,
254         0,1,2,
255         0,2,4
256
257     };
258     vector<GLfloat> piramidecuadrangular_vertices = {
259         0.5f,-0.5f,0.5f,
260         0.5f,-0.5f,-0.5f,
261         -0.5f,-0.5f,-0.5f,
262         -0.5f,-0.5f,0.5f,
263         0.0f,0.5f,0.0f,
264     };
265     Mesh* piramide = new Mesh();
266     piramide->CreateMeshGeometry(piramidecuadrangular_vertices, piramidecuadrangular_indices, 15, 18);
267     meshList.push_back(piramide);
268 }
269
270
271
272 void CreateShaders()
273 {
274     Shader* shader1 = new Shader();
275     shader1->CreateFromFiles(vShader, fShader);
276     shaderList.push_back(*shader1);
277
```

```
Practica4                                   (Ámbito global)                    main()
280
281 int main()
282 {
283     mainWindow = Window(800, 600);
284     mainWindow.Initialise();
285     //Cilindro y cono reciben resolución (slices, rebanadas) y Radio de circunferencia de la base y tapa
286
287     CrearCubo();//indice 0 en MeshList
288     CrearPiramideTriangular();//índice 1 en MeshList
289     CrearCilindro(15, 1.0f);//índice 2 en MeshList
290     CrearCono(27, 2.0f);//indice 3 en MeshList
291     CrearPiramideCuadrangular();//indice 4 en MeshList
292     CreateShaders();
293
294
295
296     /*Cámara se usa el comando: glm::lookAt(vector de posición, vector de orientación, vector up));
297     En la clase Camera se reciben 5 datos:
298     glm::vec3 vector de posición,
299     glm::vec3 vector up,
300     GlFloat yaw rotación para girar hacia la derecha e izquierda
301     GlFloat pitch rotación para inclinar hacia arriba y abajo
302     GlFloat velocidad de desplazamiento,
303     GlFloat velocidad de vuelta o de giro
304     Se usa el Mouse y las teclas WASD y su posición inicial está en 0,0,1 y ve hacia 0,0,-1.
305     */
306     camera = Camera(glm::vec3(0.0f, 0.0f, 0.0f), glm::vec3(0.0f, 1.0f, 0.0f), -60.0f, 0.0f, 0.2f, 0.2f);
307     GLuint uniformProjection = 0;
308     GLuint uniformModel = 0;
309     GLuint uniformView = 0;
310     GLuint uniformColor = 0;
311     glm::mat4 projection = glm::perspective(glm::radians(60.0f), mainWindow.getBufferWidth() / mainWindow.getBufferHeight(), 0.1f, 100.0f);
312     //glm::mat4 projection = glm::ortho(-1, 1, -1, 1, 1, 10);
313
314     //Loop mientras no se cierra la ventana
315     sp.init(); //inicializar esfera
316     sp.load();//enviar la esfera al shader
317
```

```
Practica4                                   (Ámbito global)                    main()
318     glm::mat4 model(1.0);//Inicializar matriz de Modelo 4x4
319     glm::mat4 modelaux(1.0);//Inicializar matriz de Modelo 4x4 auxiliar para la jerarquía
320     glm::mat4 modelaux2(1.0);
321     glm::mat4 modelaux3(1.0);
322     glm::vec3 color = glm::vec3(0.0f, 0.0f, 0.0f); //inicializar Color para enviar a variable Uniform;
323
324     while (!mainWindow.getShouldClose())
325     {
326
327         GLfloat now = glfwGetTime();
328         deltaTime = now - lastTime;
329         deltaTime += (now - lastTime) / limitFPS;
330         lastTime = now;
331         //Recibir eventos del usuario
332         glfwPollEvents();
333         //Cámara
334         camera.keyControl(mainWindow.getsKeys(), deltaTime);
335         camera.mouseControl(mainWindow.getXChange(), mainWindow.getYChange());
336
337         //Limpiar la ventana
338         glClearColor(0.9f, 0.9f, 0.7f, 1.0f);
339         glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT); //Se agrega limpiar el buffer de profundidad
340         shaderList[0].useShader();
341         uniformModel = shaderList[0].getModelLocation();
342         uniformProjection = shaderList[0].getProjectLocation();
343         uniformView = shaderList[0].getViewLocation();
344         uniformColor = shaderList[0].getColorLocation();
345         //Grúa
346         {
347
348
349             // Creando el brazo de una grúa
350             //articulacion1 hasta articulación5 sólo son puntos de rotación o articulación, en este caso no dibujaremos esferas que
351
352             //para reiniciar la matriz de modelo con valor de la matriz identidad
353             model = glm::mat4(1.0);
354
```

```
Practica4                                   (Ámbito global)                    main()
355             ////AQUÍ SE DIBUJA LA CABINA, LA BASE, LAS 4 LLANTAS
356             //
357
358             //Cabina cuerpo
359             model = glm::translate(model, glm::vec3(0.0f, 0.0f, -4.0f));
360             modelaux = model;
361             model = glm::scale(model, glm::vec3(5.0f, 3.0f, 4.0f));
362             glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
363             glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
364             glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
365             color = glm::vec3(0.15f, 0.8f, 0.55f);
366             glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
367             meshList[0]->RenderMesh(); //dibuja cubo, pirámide triangular, pirámide base cuadrangular
368             //para descartar la escala que no quiero heredar se carga la información de la matrix auxiliar
369             model = modelaux;
370
371
372             {
373                 //base ruedas
374                 color = glm::vec3(0.44f, 0.48f, 0.48f);
375                 model = glm::translate(model, glm::vec3(0.0f, -1.1f, 0.0f));
376                 modelaux2 = model;
377                 model = glm::scale(model, glm::vec3(5.0f, 2.0f, 4.0f));
378                 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
379                 glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
380                 meshList[4]->RenderMesh();
381                 model = modelaux2;
382
383
384
385                 //rueda 1 atras derecha
386                 model = glm::translate(model, glm::vec3(2.0f, -1.0f, -2.3f));
387                 model = glm::scale(model, glm::vec3(1.0f, 1.0f, 0.5f));
388                 model = glm::rotate(model, glm::radians(90.0f), glm::vec3(1.0f, 0.0f, 0.0f));
389                 model = glm::rotate(model, glm::radians(mainWindow.getarticulacion5()), glm::vec3(0.0f, 1.0f, 0.0f));
390                 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
391                 glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
```

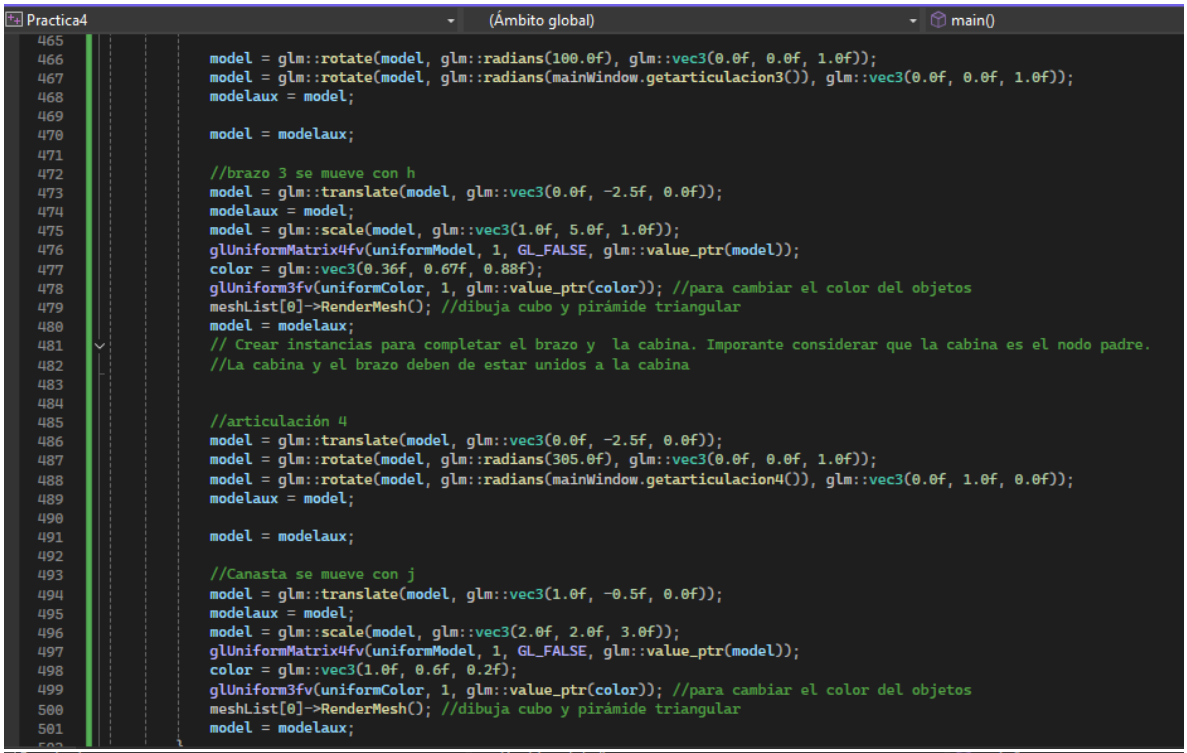
```
inline glm::highp_vec3::vec(float a, float b, float c)
19 sobrecargas más
Buscar en línea
```

Practica4 (Ámbito global) main()

```
392             meshList[2]->RenderMeshGeometry();
393             model = modelaux2;
394
395             //rueda 2 adelante derecha
396             model = glm::translate(model, glm::vec3(2.0f, -1.0f, 2.3f));
397             model = glm::scale(model, glm::vec3(1.0f, 1.0f, 0.5f));
398             model = glm::rotate(model, glm::radians(90.0f), glm::vec3(1.0f, 0.0f, 0.0f));
399             model = glm::rotate(model, glm::radians(mainWindow.getarticulacion6()), glm::vec3(0.0f, 1.0f, 0.0f));
400             glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
401             glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
402             meshList[2]->RenderMeshGeometry();
403             model = modelaux2;
404
405             //rueda 3 atras izquierda
406             model = glm::translate(model, glm::vec3(-2.0f, -1.0f, -2.3f));
407             model = glm::scale(model, glm::vec3(1.0f, 1.0f, 0.5f));
408             model = glm::rotate(model, glm::radians(90.0f), glm::vec3(1.0f, 0.0f, 0.0f));
409             model = glm::rotate(model, glm::radians(mainWindow.getarticulacion7()), glm::vec3(0.0f, 1.0f, 0.0f));
410             glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
411             glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
412             meshList[2]->RenderMeshGeometry();
413             model = modelaux2;
414
415             //rueda 4 adelante izquierda
416             model = glm::translate(model, glm::vec3(-2.0f, -1.0f, 2.3f));
417             model = glm::scale(model, glm::vec3(1.0f, 1.0f, 0.5f));
418             model = glm::rotate(model, glm::radians(90.0f), glm::vec3(1.0f, 0.0f, 0.0f));
419             model = glm::rotate(model, glm::radians(mainWindow.getarticulacion8()), glm::vec3(0.0f, 1.0f, 0.0f));
420             glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
421             glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
422             meshList[2]->RenderMeshGeometry();
423         }
424         model = modelaux;
425
426         //articulación 1
427         model = glm::translate(model, glm::vec3(-2.0f, 1.0f, 0.0f));
```

Practica4 (Ámbito global) main()

```
428         model = glm::rotate(model, glm::radians(225.0f), glm::vec3(0.0f, 0.0f, 1.0f));
429         model = glm::rotate(model, glm::radians(mainWindow.getarticulacion1()), glm::vec3(0.0f, 0.0f, 1.0f));
430         modelaux = model;
431
432
433         //brazo 1 se mueve con f
434         model = glm::translate(model, glm::vec3(0.0f, -2.5f, 0.0f));
435         modelaux = model;
436         model = glm::scale(model, glm::vec3(1.0f, 5.0f, 1.0f));
437         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
438         color = glm::vec3(0.95f, 0.81f, 0.24f);
439         glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
440         meshList[0]->RenderMesh(); //dibuja cubo y pirámide triangular
441         model = modelaux;
442
443         //articulación 2
444         model = glm::translate(model, glm::vec3(0.0f, -2.5f, 0.0f));
445
446         model = glm::rotate(model, glm::radians(270.0f), glm::vec3(0.0f, 0.0f, 1.0f));
447         model = glm::rotate(model, glm::radians(mainWindow.getarticulacion2()), glm::vec3(0.0f, 0.0f, 1.0f));
448         modelaux = model;
449
450
451         model = modelaux;
452
453         //brazo 2 se mueve con g
454         model = glm::translate(model, glm::vec3(0.0f, -2.5f, 0.0f));
455         modelaux = model;
456         model = glm::scale(model, glm::vec3(1.0f, 5.0f, 1.0f));
457         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
458         color = glm::vec3(0.60f, 0.34f, 0.71f);
459         glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
460         meshList[0]->RenderMesh(); //dibuja cubo y pirámide triangular
461         model = modelaux;
462
463         //articulación 3
464         model = glm::translate(model, glm::vec3(0.0f, -2.5f, 0.0f));
```



```
Practica4 (Ámbito global) main()
465
466         model = glm::rotate(model, glm::radians(100.0f), glm::vec3(0.0f, 0.0f, 1.0f));
467         model = glm::rotate(model, glm::radians(mainWindow.getarticulacion3()), glm::vec3(0.0f, 0.0f, 1.0f));
468         modelaux = model;
469
470         model = modelaux;
471
472         //brazo 3 se mueve con h
473         model = glm::translate(model, glm::vec3(0.0f, -2.5f, 0.0f));
474         modelaux = model;
475         model = glm::scale(model, glm::vec3(1.0f, 5.0f, 1.0f));
476         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
477         color = glm::vec3(0.36f, 0.67f, 0.88f);
478         glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
479         meshList[0]->RenderMesh(); //dibuja cubo y pirámide triangular
480         model = modelaux;
481         // Crear instancias para completar el brazo y  la cabina. Imporante considerar que la cabina es el nodo padre.
482         //La cabina y el brazo deben de estar unidos a la cabina
483
484
485         //articulación 4
486         model = glm::translate(model, glm::vec3(0.0f, -2.5f, 0.0f));
487         model = glm::rotate(model, glm::radians(305.0f), glm::vec3(0.0f, 0.0f, 1.0f));
488         model = glm::rotate(model, glm::radians(mainWindow.getarticulacion4()), glm::vec3(0.0f, 1.0f, 0.0f));
489         modelaux = model;
490
491         model = modelaux;
492
493         //Canasta se mueve con j
494         model = glm::translate(model, glm::vec3(1.0f, -0.5f, 0.0f));
495         modelaux = model;
496         model = glm::scale(model, glm::vec3(2.0f, 2.0f, 3.0f));
497         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
498         color = glm::vec3(1.0f, 0.6f, 0.2f);
499         glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
500         meshList[0]->RenderMesh(); //dibuja cubo y pirámide triangular
501         model = modelaux;
```

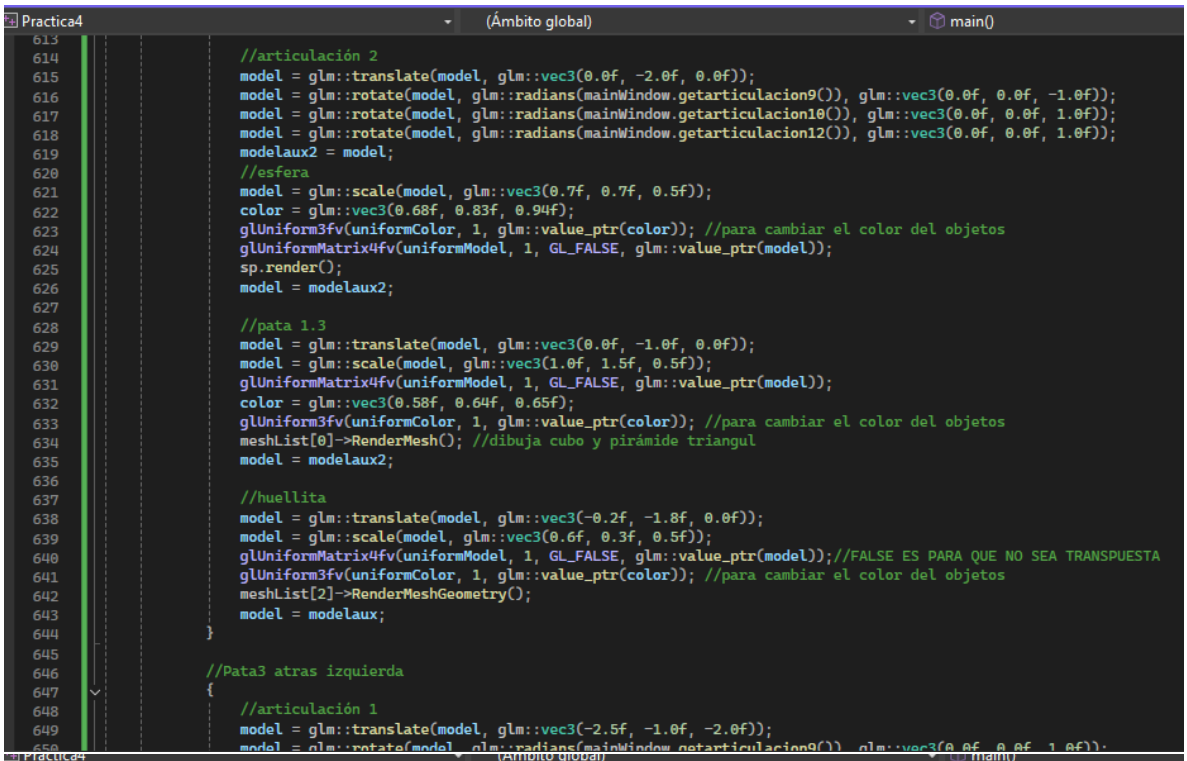
```
Practica4 (Ámbito global) main()
503
504     //Animal
505     {
506         // Creando el brazo de una grúa
507         //articulacion1 hasta articulación5 sólo son puntos de rotación o articulación, en este caso no dibujaremos es
508
509         //para reiniciar la matriz de modelo con valor de la matriz identidad
510         model = glm::mat4(1.0);
511
512
513         //
514         //Cuerpo perrito
515         model = glm::translate(model, glm::vec3(17.0f, 0.0f, -4.0f));
516
517         modelaux = model;
518         model = glm::scale(model, glm::vec3(4.0f, 2.0f, 3.0f));
519         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
520         glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
521         glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));
522         color = glm::vec3(0.58f, 0.64f, 0.65f);
523         glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
524         sp.render();
525         //para descartar la escala que no quiero heredar se carga la información de la matrix auxiliar
526         model = modelaux;
527
528         //Pata 1 adeante izquierda
529         {
530             //articulación 1
531             model = glm::translate(model, glm::vec3(-2.5f, -1.0f, 2.0f));
532             model = glm::rotate(model, glm::radians(mainWindow.getarticulacion9()), glm::vec3(0.0f, 0.0f, -1.0f));
533             model = glm::rotate(model, glm::radians(mainWindow.getarticulacion10()), glm::vec3(0.0f, 0.0f, 1.0f));
534             model = glm::rotate(model, glm::radians(mainWindow.getarticulacion11()), glm::vec3(0.0f, 0.0f, 1.0f));
535             modelaux2 = model;
536
537             model = glm::scale(model, glm::vec3(1.0f, 1.0f, 0.5f));
538             color = glm::vec3(0.68f, 0.83f, 0.94f);
539             glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
```

Practica4 (Ámbito global) main()

```
            glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
            sp.render();
            model = modelaux2;

            //pata 1.2
            model = glm::translate(model, glm::vec3(0.0f, -1.0f, 0.0f));
            model = glm::scale(model, glm::vec3(1.0f, 1.5f, 0.5f));
            glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
            color = glm::vec3(0.58f, 0.64f, 0.65f);
            glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
            meshList[0]->RenderMesh(); //dibuja cubo y pirámide triangul
            model = modelaux2;

            //articulación 2
            model = glm::translate(model, glm::vec3(0.0f, -2.0f, 0.0f));
            model = glm::rotate(model, glm::radians(mainWindow.getarticulacion9()), glm::vec3(0.0f, 0.0f, 1.0f));
            model = glm::rotate(model, glm::radians(mainWindow.getarticulacion10()), glm::vec3(0.0f, 0.0f, -1.0f));
            model = glm::rotate(model, glm::radians(mainWindow.getarticulacion12()), glm::vec3(0.0f, 0.0f, 1.0f));
            modelaux2 = model;

            //esfera
            model = glm::scale(model, glm::vec3(0.7f, 0.7f, 0.5f));
            color = glm::vec3(0.68f, 0.83f, 0.94f);
            glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
            glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
            sp.render();
            model = modelaux2;

            //pata 1.3
            model = glm::translate(model, glm::vec3(0.0f, -1.0f, 0.0f));
            model = glm::scale(model, glm::vec3(1.0f, 1.5f, 0.5f));
            glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
            color = glm::vec3(0.58f, 0.64f, 0.65f);
            glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
            meshList[0]->RenderMesh(); //dibuja cubo y pirámide triangul
            model = modelaux2;
```

Practica4 (Ámbito global) main()

```
            //huellita
            model = glm::translate(model, glm::vec3(-0.2f, -1.8f, 0.0f));
            model = glm::scale(model, glm::vec3(0.6f, 0.3f, 0.5f));
            glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
            glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
            meshList[2]->RenderMeshGeometry();


        }

        //Pata 2 adelante derecha
        {
            model = modelaux;
            //articulación 1
            model = glm::translate(model, glm::vec3(2.5f, -1.0f, 2.0f));
            model = glm::rotate(model, glm::radians(mainWindow.getarticulacion9()), glm::vec3(0.0f, 0.0f, 1.0f));
            model = glm::rotate(model, glm::radians(mainWindow.getarticulacion10()), glm::vec3(0.0f, 0.0f, -1.0f));
            model = glm::rotate(model, glm::radians(mainWindow.getarticulacion11()), glm::vec3(0.0f, 0.0f, 1.0f));
            modelaux2 = model;

            // esfera
            model = glm::scale(model, glm::vec3(1.0f, 1.0f, 0.5f));
            color = glm::vec3(0.68f, 0.83f, 0.94f);
            glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
            glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
            sp.render();
            model = modelaux2;

            //pata 1.2
            model = glm::translate(model, glm::vec3(0.0f, -1.0f, 0.0f));
            model = glm::scale(model, glm::vec3(1.0f, 1.5f, 0.5f));
            glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
            color = glm::vec3(0.58f, 0.64f, 0.65f);
            glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
            meshList[0]->RenderMesh(); //dibuja cubo y pirámide triangul
            model = modelaux2;
```



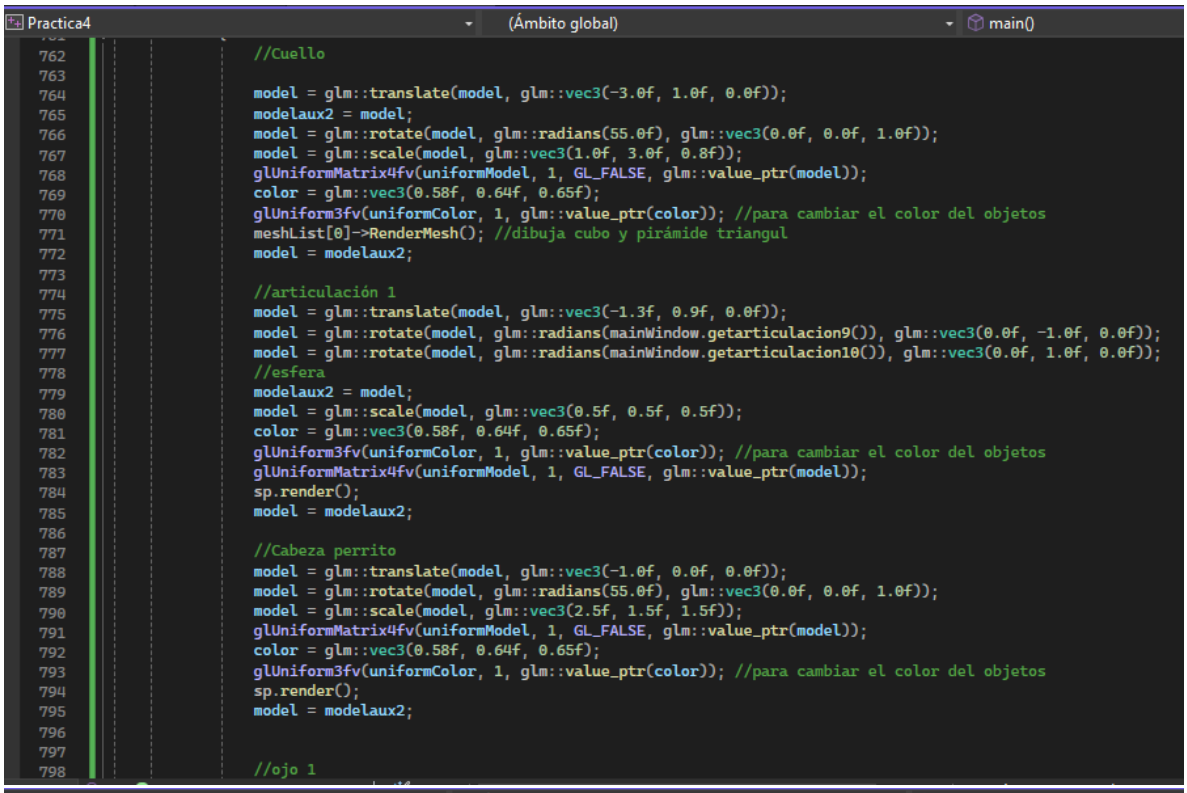
```
Practica4 | (Ámbito global) | main()

            //articulación 2
            model = glm::translate(model, glm::vec3(0.0f, -2.0f, 0.0f));
            model = glm::rotate(model, glm::radians(mainWindow.getarticulacion9()), glm::vec3(0.0f, 0.0f, -1.0f));
            model = glm::rotate(model, glm::radians(mainWindow.getarticulacion10()), glm::vec3(0.0f, 0.0f, 1.0f));
            model = glm::rotate(model, glm::radians(mainWindow.getarticulacion12()), glm::vec3(0.0f, 0.0f, 1.0f));
            modelaux2 = model;
            //esfera
            model = glm::scale(model, glm::vec3(0.7f, 0.7f, 0.5f));
            color = glm::vec3(0.68f, 0.83f, 0.94f);
            glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
            glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
            sp.render();
            model = modelaux2;

            //pata 1.3
            model = glm::translate(model, glm::vec3(0.0f, -1.0f, 0.0f));
            model = glm::scale(model, glm::vec3(1.0f, 1.5f, 0.5f));
            glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
            color = glm::vec3(0.58f, 0.64f, 0.65f);
            glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
            meshList[0]->RenderMesh(); //dibuja cubo y pirámide triangul
            model = modelaux2;

            //huellita
            model = glm::translate(model, glm::vec3(-0.2f, -1.8f, 0.0f));
            model = glm::scale(model, glm::vec3(0.6f, 0.3f, 0.5f));
            glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
            glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
            meshList[2]->RenderMeshGeometry();
            model = modelaux;
        }

        //Pata3 atras izquierda
        {
            //articulación 1
            model = glm::translate(model, glm::vec3(-2.5f, -1.0f, -2.0f));
            model = glm::rotate(model, glm::radians(mainWindow.getarticulacion9()), glm::vec3(0.0f, 0.0f, -1.0f));
            model = glm::rotate(model, glm::radians(mainWindow.getarticulacion10()), glm::vec3(0.0f, 0.0f, -1.0f));
            model = glm::rotate(model, glm::radians(mainWindow.getarticulacion11()), glm::vec3(0.0f, 0.0f, 1.0f));
            modelaux2 = model;
            //esfera
            model = glm::scale(model, glm::vec3(1.0f, 1.0f, 0.5f));
            color = glm::vec3(0.68f, 0.83f, 0.94f);
            glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
            glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
            sp.render();
            model = modelaux2;

            //pata 1.2
            model = glm::translate(model, glm::vec3(0.0f, -1.0f, 0.0f));
            model = glm::scale(model, glm::vec3(1.0f, 1.5f, 0.5f));
            glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
            color = glm::vec3(0.58f, 0.64f, 0.65f);
            glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
            meshList[0]->RenderMesh(); //dibuja cubo y pirámide triangul
            model = modelaux2;

            //articulación 2
            model = glm::translate(model, glm::vec3(0.0f, -2.0f, 0.0f));
            model = glm::rotate(model, glm::radians(mainWindow.getarticulacion9()), glm::vec3(0.0f, 0.0f, -1.0f));
            model = glm::rotate(model, glm::radians(mainWindow.getarticulacion10()), glm::vec3(0.0f, 0.0f, 1.0f));
            model = glm::rotate(model, glm::radians(mainWindow.getarticulacion12()), glm::vec3(0.0f, 0.0f, 1.0f));
            modelaux2 = model;
            //esfera
            model = glm::scale(model, glm::vec3(0.7f, 0.7f, 0.5f));
            color = glm::vec3(0.68f, 0.83f, 0.94f);
            glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
            glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
            sp.render();
            model = modelaux2;

            //pata 1.3
            model = glm::translate(model, glm::vec3(0.0f, -1.0f, 0.0f));
            model = glm::scale(model, glm::vec3(1.0f, 1.5f, 0.5f));
```

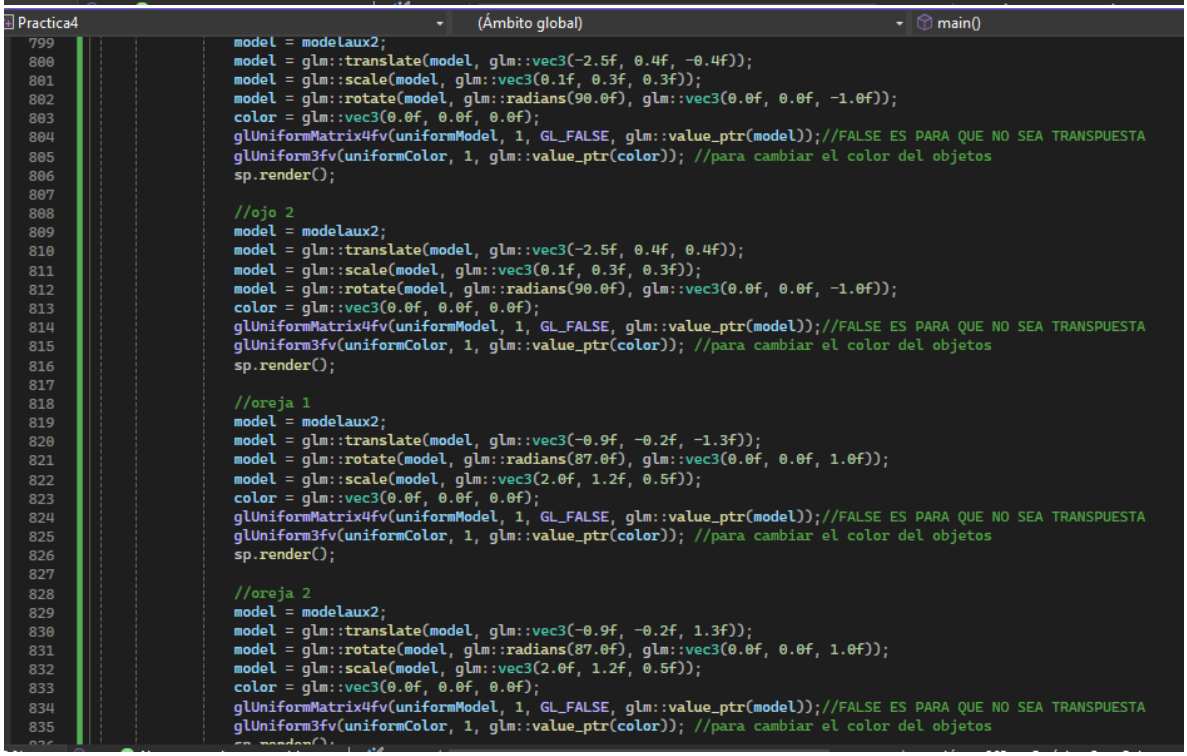
Practica4 (Ámbito global) main()

```
688             glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
689             color = glm::vec3(0.58f, 0.64f, 0.65f);
690             glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
691             meshList[0]->RenderMesh(); //dibuja cubo y pirámide triangul
692             model = modelaux2;
693
694             //huellita
695             model = glm::translate(model, glm::vec3(-0.2f, -1.8f, 0.0f));
696             model = glm::scale(model, glm::vec3(0.6f, 0.3f, 0.5f));
697             glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
698             glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
699             meshList[2]->RenderMeshGeometry();
700             model = modelaux;
701         }
702
703         //Pata4 atras derecha
704         {
705             //articulación 1
706             model = glm::translate(model, glm::vec3(2.5f, -1.0f, -2.0f));
707             model = glm::rotate(model, glm::radians(mainWindow.getarticulacion9()), glm::vec3(0.0f, 0.0f, -1.0f));
708             model = glm::rotate(model, glm::radians(mainWindow.getarticulacion10()), glm::vec3(0.0f, 0.0f, 1.0f));
709             model = glm::rotate(model, glm::radians(mainWindow.getarticulacion11()), glm::vec3(0.0f, 0.0f, 1.0f));
710             modelaux2 = model;
711             //esfera
712             model = glm::scale(model, glm::vec3(1.0f, 1.0f, 0.5f));
713             color = glm::vec3(0.68f, 0.83f, 0.94f);
714             glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
715             glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
716             sp.render();
717             model = modelaux2;
718
719             //pata 1.2
720             model = glm::translate(model, glm::vec3(0.0f, -1.0f, 0.0f));
721             model = glm::scale(model, glm::vec3(1.0f, 1.5f, 0.5f));
722             glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
723             color = glm::vec3(0.58f, 0.64f, 0.65f);
724             glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
```

Practica4 (Ámbito global) main()

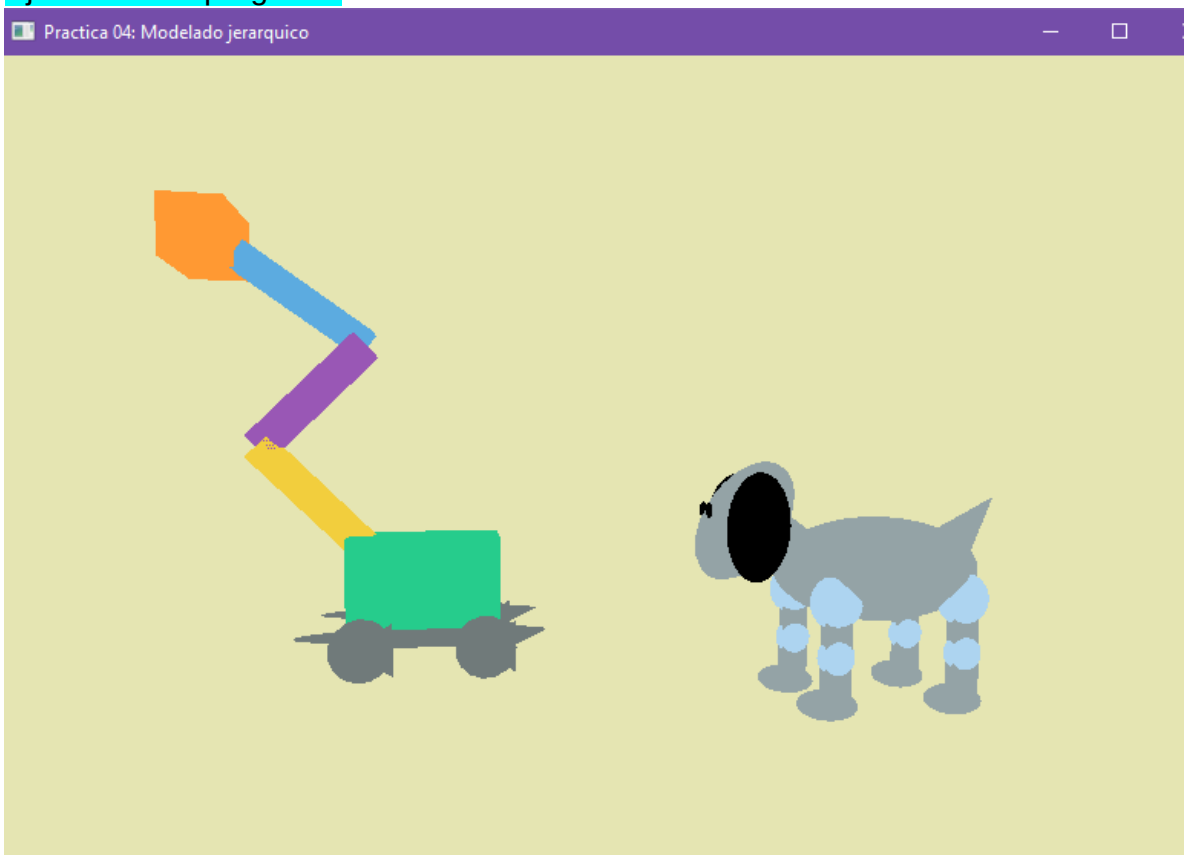
```
725             meshList[0]->RenderMesh(); //dibuja cubo y pirámide triangul
726             model = modelaux2;
727
728             //articulación 2
729             model = glm::translate(model, glm::vec3(0.0f, -2.0f, 0.0f));
730             model = glm::rotate(model, glm::radians(mainWindow.getarticulacion9()), glm::vec3(0.0f, 0.0f, 1.0f));
731             model = glm::rotate(model, glm::radians(mainWindow.getarticulacion10()), glm::vec3(0.0f, 0.0f, -1.0f));
732             model = glm::rotate(model, glm::radians(mainWindow.getarticulacion12()), glm::vec3(0.0f, 0.0f, 1.0f));
733             modelaux2 = model;
734             //esfera
735             model = glm::scale(model, glm::vec3(0.7f, 0.7f, 0.5f));
736             color = glm::vec3(0.68f, 0.83f, 0.94f);
737             glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
738             glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
739             sp.render();
740             model = modelaux2;
741
742             //pata 1.3
743             model = glm::translate(model, glm::vec3(0.0f, -1.0f, 0.0f));
744             model = glm::scale(model, glm::vec3(1.0f, 1.5f, 0.5f));
745             glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
746             color = glm::vec3(0.58f, 0.64f, 0.65f);
747             glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
748             meshList[0]->RenderMesh(); //dibuja cubo y pirámide triangul
749             model = modelaux2;
750
751             //huellita
752             model = glm::translate(model, glm::vec3(-0.2f, -1.8f, 0.0f));
753             model = glm::scale(model, glm::vec3(0.6f, 0.3f, 0.5f));
754             glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
755             glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
756             meshList[2]->RenderMeshGeometry();
757             model = modelaux;
758         }
759
760         //Parte de la cabeza
761         {
```

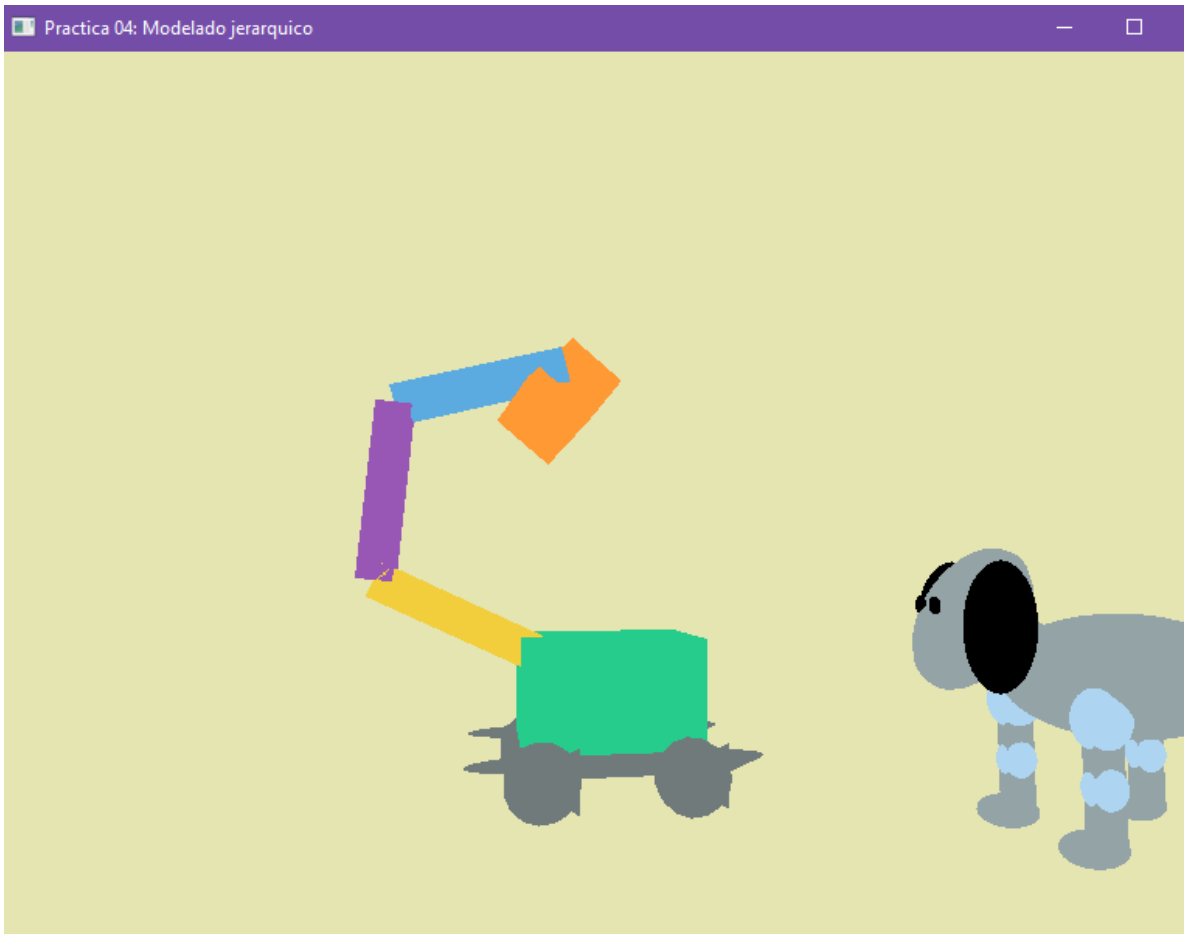

Practica4
(Ámbito global)
main()
762 //Cuello
763
764 model = glm::translate(model, glm::vec3(-3.0f, 1.0f, 0.0f));
765 modelaux2 = model;
766 model = glm::rotate(model, glm::radians(55.0f), glm::vec3(0.0f, 0.0f, 1.0f));
767 model = glm::scale(model, glm::vec3(1.0f, 3.0f, 0.8f));
768 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
769 color = glm::vec3(0.58f, 0.64f, 0.65f);
770 glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
771 meshList[0]->RenderMesh(); //dibuja cubo y pirámide triangul
772 model = modelaux2;
773
774 //articulación 1
775 model = glm::translate(model, glm::vec3(-1.3f, 0.9f, 0.0f));
776 model = glm::rotate(model, glm::radians(mainWindow.getarticulacion9()), glm::vec3(0.0f, -1.0f, 0.0f));
777 model = glm::rotate(model, glm::radians(mainWindow.getarticulacion10()), glm::vec3(0.0f, 1.0f, 0.0f));
778 //esfera
779 modelaux2 = model;
780 model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
781 color = glm::vec3(0.58f, 0.64f, 0.65f);
782 glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
783 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
784 sp.render();
785 model = modelaux2;
786
787 //Cabeza perrito
788 model = glm::translate(model, glm::vec3(-1.0f, 0.0f, 0.0f));
789 model = glm::rotate(model, glm::radians(55.0f), glm::vec3(0.0f, 0.0f, 1.0f));
790 model = glm::scale(model, glm::vec3(2.5f, 1.5f, 1.5f));
791 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
792 color = glm::vec3(0.58f, 0.64f, 0.65f);
793 glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
794 sp.render();
795 model = modelaux2;
796
797
798 //ojo 1

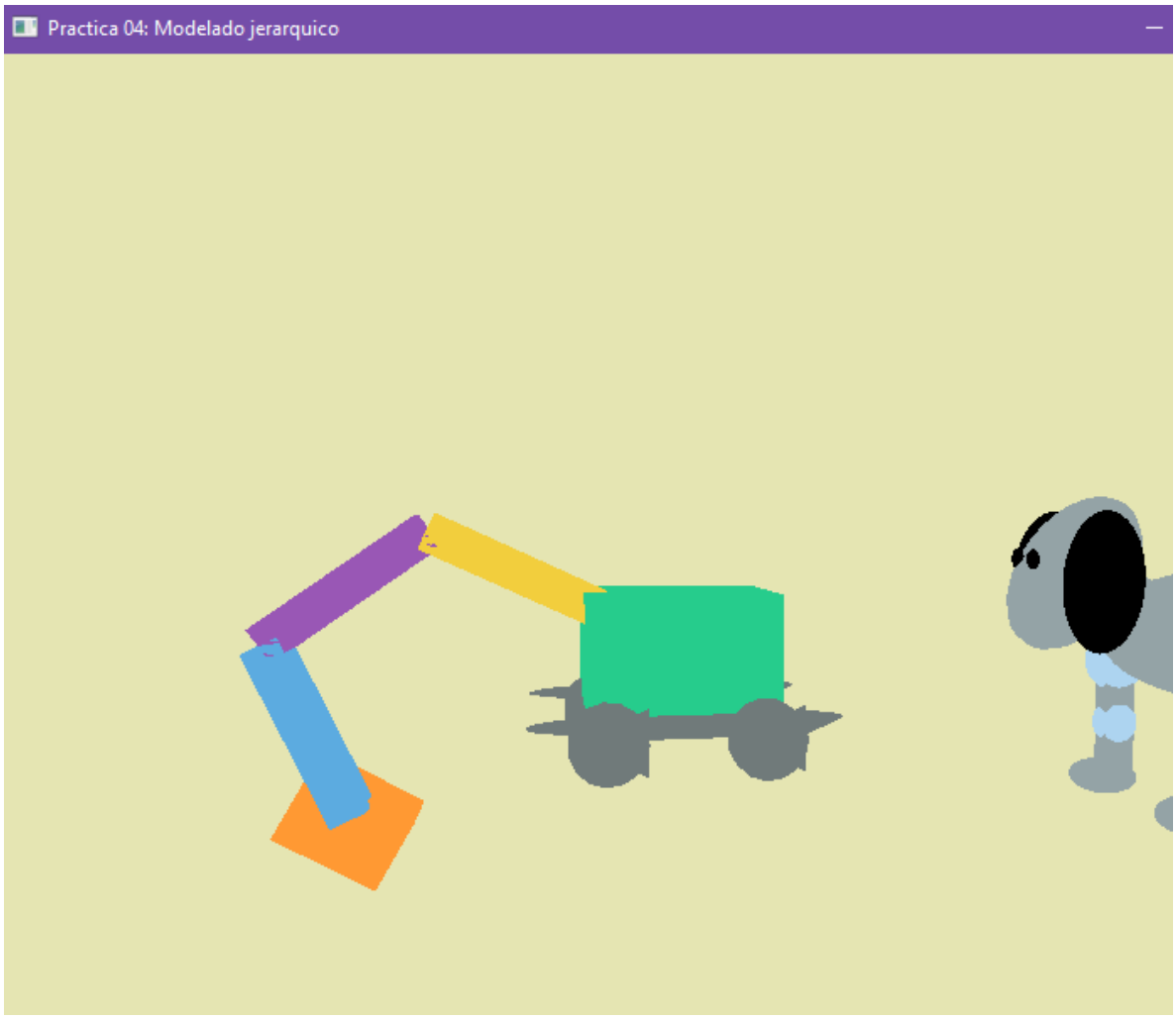

Practica4
(Ámbito global)
main()
799 model = modelaux2;
800 model = glm::translate(model, glm::vec3(-2.5f, 0.4f, -0.4f));
801 model = glm::scale(model, glm::vec3(0.1f, 0.3f, 0.3f));
802 model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 0.0f, -1.0f));
803 color = glm::vec3(0.0f, 0.0f, 0.0f);
804 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
805 glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
806 sp.render();
807
808 //ojo 2
809 model = modelaux2;
810 model = glm::translate(model, glm::vec3(-2.5f, 0.4f, 0.4f));
811 model = glm::scale(model, glm::vec3(0.1f, 0.3f, 0.3f));
812 model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 0.0f, -1.0f));
813 color = glm::vec3(0.0f, 0.0f, 0.0f);
814 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
815 glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
816 sp.render();
817
818 //oreja 1
819 model = modelaux2;
820 model = glm::translate(model, glm::vec3(-0.9f, -0.2f, -1.3f));
821 model = glm::rotate(model, glm::radians(87.0f), glm::vec3(0.0f, 0.0f, 1.0f));
822 model = glm::scale(model, glm::vec3(2.0f, 1.2f, 0.5f));
823 color = glm::vec3(0.0f, 0.0f, 0.0f);
824 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
825 glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
826 sp.render();
827
828 //oreja 2
829 model = modelaux2;
830 model = glm::translate(model, glm::vec3(-0.9f, -0.2f, 1.3f));
831 model = glm::rotate(model, glm::radians(87.0f), glm::vec3(0.0f, 0.0f, 1.0f));
832 model = glm::scale(model, glm::vec3(2.0f, 1.2f, 0.5f));
833 color = glm::vec3(0.0f, 0.0f, 0.0f);
834 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES PARA QUE NO SEA TRANSPUESTA
835 glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos

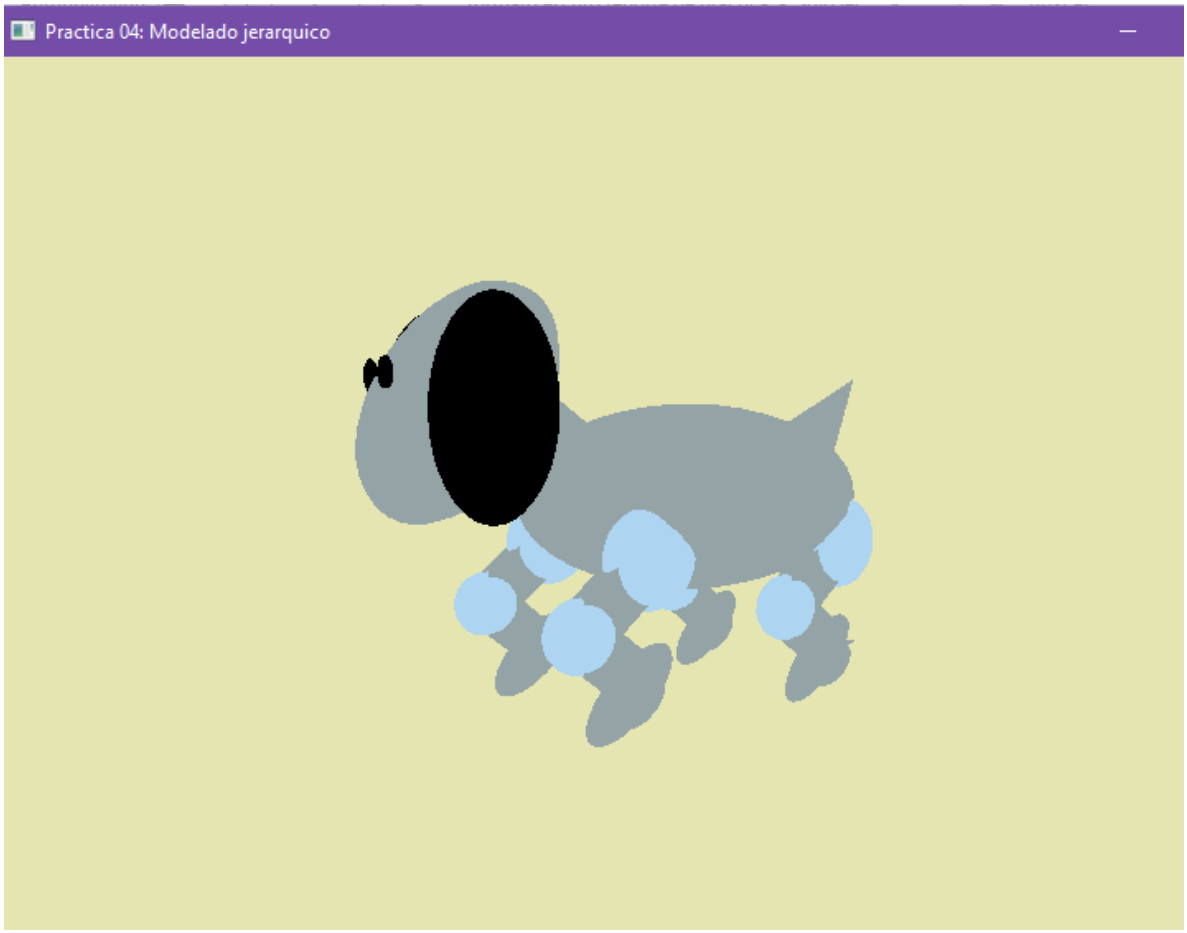
```
Practica4                                   (Ámbito global)                    main()
836                 sp.render();
837             }
838             //Cola
839             {
840                 model = modelaux;
841                 //articulación cola
842                 model = glm::translate(model, glm::vec3(3.0f, 0.7f, 0.0f));
843                 model = glm::rotate(model, glm::radians(mainWindow.getarticulacion9()), glm::vec3(0.0f, -1.0f, 0.0f));
844                 model = glm::rotate(model, glm::radians(mainWindow.getarticulacion10()), glm::vec3(0.0f, 1.0f, 0.0f));
845                 model = glm::rotate(model, glm::radians(mainWindow.getarticulacion13()), glm::vec3(0.0f, 1.0f, 0.0f));
846                 modelaux2 = model;
847                 //dibujar una pequeña esfera
848                 model = glm::scale(model, glm::vec3(0.5f, 0.5f, 0.5f));
849                 color = glm::vec3(0.68f, 0.83f, 0.94f);
850                 glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
851                 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
852                 sp.render();
853                 model = modelaux2;
854
855                 //cola
856                 model = glm::translate(model, glm::vec3(1.0f, 1.0f, 0.0f));
857                 model = glm::rotate(model, glm::radians(45.0f), glm::vec3(0.0f, 0.0f, -1.0f));
858                 model = glm::scale(model, glm::vec3(1.52f, 3.0f, 1.2f));
859                 glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
860                 color = glm::vec3(0.58f, 0.64f, 0.65f);
861                 glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
862                 meshList[4]->RenderMesh(); //dibuja cubo y pirámide triangular
863             }
864         }
865         glUseProgram(0);
866         mainWindow.swapBuffers();
867
868
869     }
870     return 0;
871 }
```

Ejecución del programa

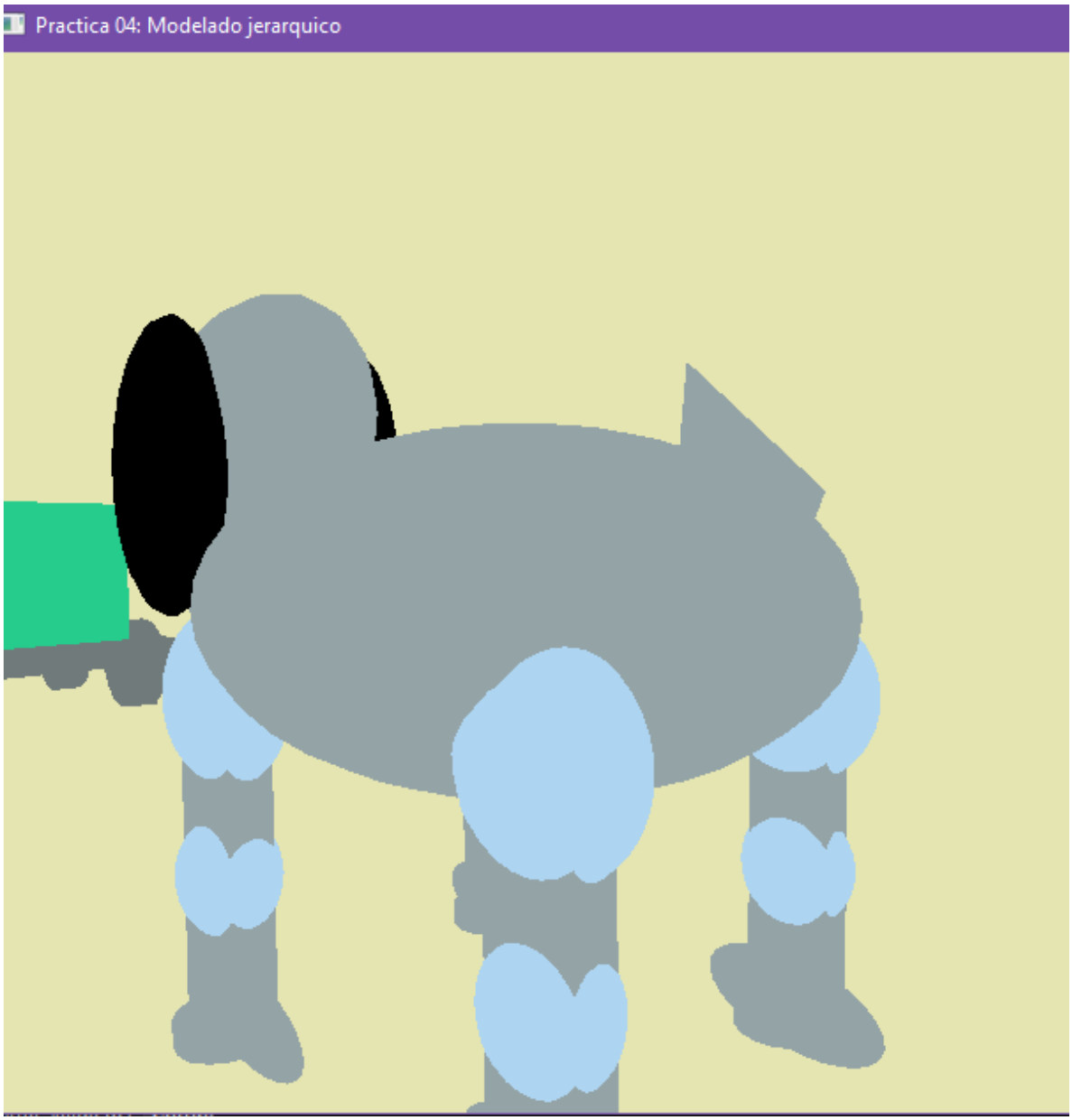



Practica 04: Modelado jerarquico


Practica 04: Modelado jerarquico


Practica 04: Modelado jerarquico


Practica 04: Modelado jerarquico


Practica 04: Modelado jerarquico

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

- Uno de los pequeños problemas que presenté al inicio fue al momento de darle movimiento a las llantas, ya que me confundí un poco al momento de donde definir las correctas articulaciones con las letras del teclado.
- Otro problema fue al momento de definir las jerarquías de las figuras geométricas de mi modelo.

3.- Conclusión:

a. Los ejercicios del reporte: Complejidad, Explicación.
   Después de haber realizado el ejercicio propuesto en clase puedo concluir que logré realizarlo con éxito, ya que comprendí la manera de

seguir creando figuras con la diferencia de que en este caso ahora se usa siguiendo una jerarquía, pude observar que es muy importante ya que a partir de este podemos hacer que nuestros modelos sigan cierta estructura, lo que hicimos durante estos ejercicios primero para la grúa fue crear para el area de la cabina un prisma rectangular, después el caso de los brazos con cada una de articulaciones para lograr que rotaran y al final la canasta y las ruedas de la grúa todas las partes con su respectivo movimiento de articulaciones. También aprendimos nuevas funciones para definir matrices auxiliares para tomar datos de una figura a otra, como en este caso lo hicimos con la escala.

Para el caso del segundo ejercicio fue realizar exactamente lo mismo, pero con alguna figura de un animal, en mi caso elegí un perro y lo primero fue hacer la base de su cuerpo mediante una esfera, después cada una de sus patas con sus articulaciones para el movimiento, finalmente para la cabeza, sus orejas y sus ojos mediante esferas y por ultimo la cola con un prisma.

La complejidad al realizar las figuras para hacer la grúa no fue difícil ya que es algo que he realizado desde practicas anteriores, lo único nuevo en este caso fue usar las jerarquías de manera correcta y comprender que es lo que hace esta jerarquía, asi como el uso de las matrices auxiliares.

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, así como su funcionalidad.

c. Conclusión
   Después de haber concluido con la práctica puedo mencionar que se cumplió con los objetivos propuestos, debido a que comprendí la representación de modelos jerárquicos mediante primitivas geométricas y al uso de un correcto de un diagrama de jerarquías, ya que mediante las actividades propuestas pude poner en práctica estos conocimientos adquiridos y de esta manera cumplir con los ejercicios planteados.