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 4

NOMBRE COMPLETO: Gómez Enríquez Agustín

Nº de Cuenta: 317031405

GRUPO DE LABORATORIO: 3

GRUPO DE TEORÍA: 5

SEMESTRE 2026-1

FECHA DE ENTREGA LÍMITE: 21 de septiembre del 2025

CALIFICACIÓN: _____

REPORTE DE PRÁCTICA:

Parte 1

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.

1.- Terminar la Grúa con:
-cuerpo(prisma rectangular)
-base (pirámide cuadrangular)
-4 llantas(4 cilindros) con teclado se pueden girar las 4 llantas por separado

```
183  // Overlay de "tread": rectangulitos muy delgados alrededor de la pared lateral.
184  // Se dibuja con color distinto encima del cilindro solido para ver el giro.
185  // parametros:
186  //   res: segmentos del cilindro (igual que el solido, p.ej. 64..128)
187  //   R:   radio base del cilindro
188  //   every: cada cuantos segmentos pongo una barra (1 = en todos, 2 = uno si/uno no, etc.)
189  //   width: ancho angular relativo de cada barra (0..1, siendo 1 el ancho de un segmento)
190  //   extrude: factor radial extra para que no se "pegue" con el solido (p.ej. 0.02 = 2%)
191  void CrearCilindroTreadOverlay(int res, float R, int every = 2, float width = 0.6f, float extrude = 0.02f) {
192      if (res < 3) res = 3;
193      if (every < 1) every = 1;
194      if (width < 0.05f) width = 0.05f;
195      if (width > 1.0f)  width = 1.0f;
196
197      const float y0 = -0.5f, y1 = 0.5f;
198      const float dt = 2.0f * PI / float(res);
199      const float R2 = R * (1.0f + extrude);     // un poco mas grande que el solido
200      const float halfFrac = 0.5f * width;
201
202      std::vector<GLfloat> vertices;   // posiciones
203      std::vector<unsigned int> idx;   // triangulos
204
205      // Recorremos los segmentos y, para los que seleccionemos, generamos una "barra" fina:
206      // Es un rectangulo sobre la pared (dos triangulos), extruido radialmente (R2),
207      // con altura completa (y de -0.5 a 0.5) y ancho angular reducido.
208      for (int i = 0; i < res; ++i) {
209          if ((i % every) != 0) continue;
210
211          float aCenter = (i + 0.5f) * dt;
212          float a0 = aCenter - halfFrac * dt;
```

Siguiendo el orden del código, comenzamos con una modificación al cilindro con el que trabajaremos las ruedas. Ya que hay dos tipos de cilindro, el que tiene textura y el que coloca un relleno. De esta manera se nos permite visualizar la rotación de las llantas y saber a que altura pegar la llanta al chasis.

```
// Controles
{
    auto keys = mainWindow.getsKeys();

    // canasta mas lenta
    if (keys[GLFW_KEY_J]) anguloCanastaDeg -= canastaSpeedDeg * deltaTime;
    if (keys[GLFW_KEY_K]) anguloCanastaDeg += canastaSpeedDeg * deltaTime;

    // ruedas (todas sobre eje Y)
    if (keys[GLFW_KEY_Z]) ruedaDeg[0] -= ruedaSpeedDeg * deltaTime;
    if (keys[GLFW_KEY_X]) ruedaDeg[1] -= ruedaSpeedDeg * deltaTime;
    if (keys[GLFW_KEY_C]) ruedaDeg[2] -= ruedaSpeedDeg * deltaTime;
    if (keys[GLFW_KEY_V]) ruedaDeg[3] -= ruedaSpeedDeg * deltaTime;

    // NUEVO: giro de la grua completa (excepto chasis/ruedas) sobre X con N/M
    if (keys[GLFW_KEY_N]) gruaXDeg -= gruaXSpeedDeg * deltaTime; // izquierda
    if (keys[GLFW_KEY_M]) gruaXDeg += gruaXSpeedDeg * deltaTime; // derecha
}

glClearColor(0.0f, 0.0f, 0.0f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

Aquí se muestra la configuración que tenemos para los movimientos de los brazos ruedas y la extensión de rotar la grúa

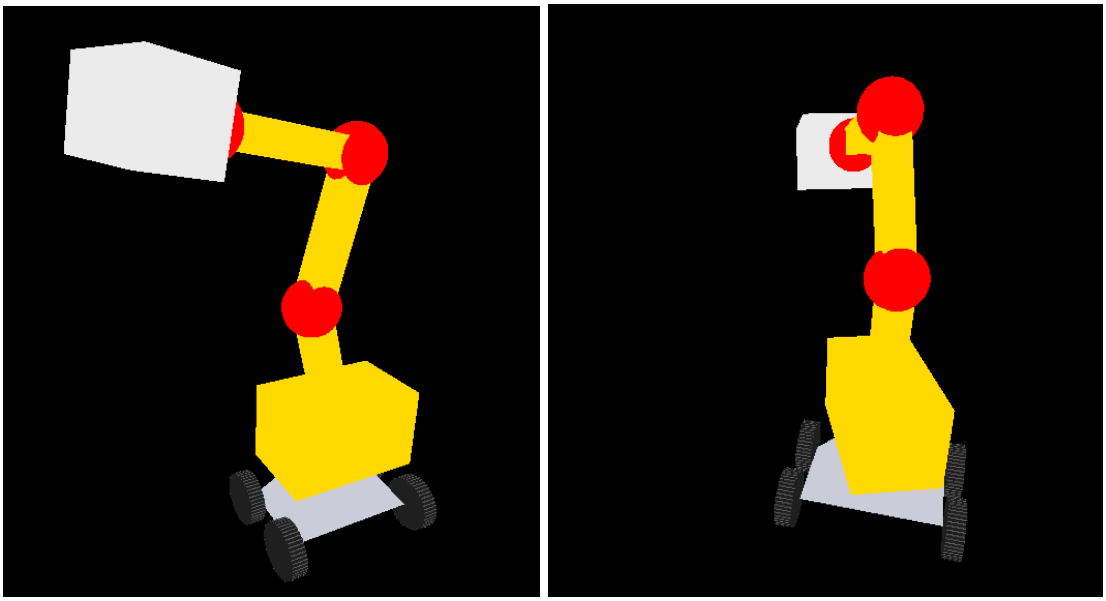
```
427
428     // ===== CHASIS (piramide)
429     {
430         const float chasisY = -1.8f;                // antes -2.2f
431         const glm::vec3 chasisScale(5.0f, 3.2f, 3.0f);
432
433         glm::mat4 m = root;                         // usar root (sin rotacion de grua)
434         m = glm::translate(m, glm::vec3(0.0f, chasisY, 0.0f));
435         m = glm::scale(m, chasisScale);
436
437         glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(m));
438         color = glm::vec3(0.80f, 0.80f, 0.85f);
439         glUniform3fv(uniformColor, 1, glm::value_ptr(color));
440         meshList[5]->RenderMesh();
441     }
442
443
444     {
445
446         const float chasisY = -1.8f;   // centro del chasis (piramide)
447         const float chasisScaleY = 3.2f;   // altura de la piramide
448
449         // base inferior de la piramide
450         const float yApex = chasisY + 0.5f * chasisScaleY;   // punta superior
451         const float yBase = chasisY - 0.5f * chasisScaleY;   // base inferior
452
```

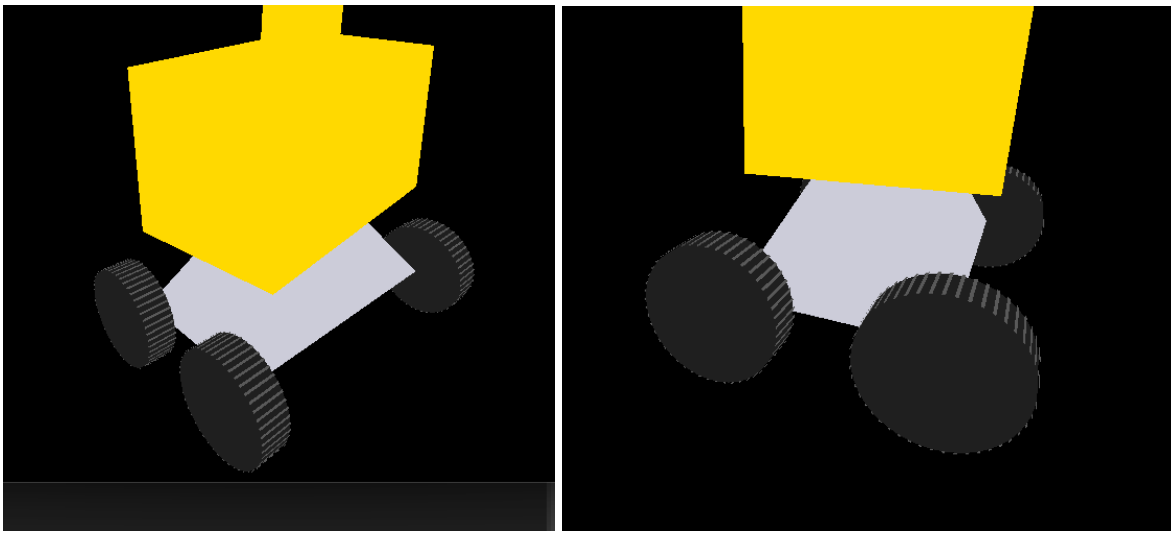
Añadimos lo solicitado en la practica que es la creación del chasis que es una pirámide de base cuadrada, que a su vez va a ir ligeramente dentro de la caja principal.

```
456             // radio y grosor (a lo largo del eje X tras la rotacion -90Z)
457             const float r = 1.0f;
458             const float t = 0.6f;
459
460             // separacion minima para que no se “meta” en el chasis
461             const float gapX = 0.06f;              // empuje extra hacia afuera en ±X
462
463             // medio ancho/profundidad de la base del chasis
464             const float baseHalfX = 2.5f;          // 5.0 / 2
465             const float baseHalfZ = 1.5f;          // 3.0 / 2
466
467             // como estamos en yW = yBase, la seccion del chasis ahi es la base completa
468             const float hxCur = baseHalfX;
469             const float hzCur = baseHalfZ;
470
471             struct Wheel {
472                 glm::vec3 baseCenter;  // punto de contacto con el chasis (sin empuje)
473                 float     signX;       // +1 derecha, -1 izquierda
474                 int       idx;
475             } wheels[4] = {
476                 { glm::vec3(-hxCur, yW, -hzCur), -1.0f, 0 },  // frente izq
477                 { glm::vec3(hxCur, yW, -hzCur), +1.0f, 1 },  // frente der
478                 { glm::vec3(-hxCur, yW,  hzCur), -1.0f, 2 },  // trasera izq
479                 { glm::vec3(hxCur, yW,  hzCur), +1.0f, 3 }   // trasera der
480             };
```

Colocamos las ruedas de manera que el centro de la circunferencia del cilindro se acomode al vértice que tiene la base del chasis y que este quede ligeramente dentro para dar la ilusión de estar unidos.

Ejecución





Parte 2

Seleccione hacer un gato con movimiento en las extremidades y las orejas.

Movimientos:

```
38  // ====== ROBOT CAT: estados ======
39  struct Joint1D { float ang = 0.0f, minDeg, maxDeg, speedDeg; };
40  struct Leg { Joint1D hip, knee; };
41
42  static Leg legs[4];               // 0=FL,1=FR,2=BL,3=BR
43  static Joint1D tail[3];           // cola: base, seg2, seg3
44
45  static float headYaw = 0.0f, headPitch = 0.0f;
46
47  static inline float clampf(float v, float a, float b) { return glm::min(glm::max(v, a), b); }
48
49  // Inicializa rangos/velocidades
50  static void initRobotCat()
51  {
52      auto setJ = [](Joint1D& j, float minD, float maxD, float speed) { j.minDeg = minD; j.maxDeg = maxD; j.speedDeg = speed; j.ang = 0.0f; };
53
54      for (int i = 0; i < 4; ++i) {
55          setJ(legs[i].hip, -40.0f, +30.0f, 45.0f);   // cadera
56          setJ(legs[i].knee, 0.0f, +70.0f, 60.0f);  // rodilla (solo flexiona hacia adelante)
57      }
58      setJ(tail[0], -35.0f, +35.0f, 50.0f);
59      setJ(tail[1], -35.0f, +35.0f, 50.0f);
60      setJ(tail[2], -35.0f, +35.0f, 50.0f);
61  }
62
```

Comenzamos definiendo la estructura para nuestro robot donde trabajaremos las articulaciones y las interacciones de cada grupo del robot.

```
303  int main()
304  {
305      // ====== Inicialización ======
306      mainWindow = Window(800, 600);
307      mainWindow.Initialise();
308
309      glEnable(GL_DEPTH_TEST);
310      glDisable(GL_CULL_FACE);
311      glPolygonMode(GL_FRONT_AND_BACK, GL_FILL);
312
313      // ====== Geometrías ======
314      CrearCubo();                        // [0] cubo
315      CrearPiramideTriangular();          // [1] orejas
316      CrearCilindro(96, 1.0f);            // [2] cuello
317      CreateShaders();
318
```

Vamos a trabajar de la siguiente manera:

Esfera: Articulaciones y cabeza

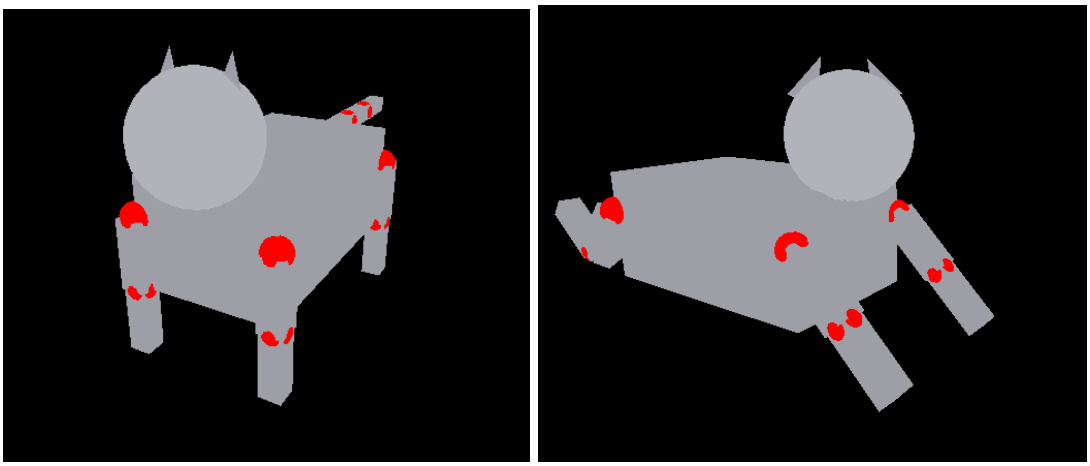
Cilindro: Cuello

Cubo / caja. Piernas (muslos y pantorrillas)

```
339      // Patas: mismas restricciones pero velocidades MÁS LENTAS
340      for (int i = 0; i < 4; ++i) {
341          legs[i].hip.minDeg = -45.0f;
342          legs[i].hip.maxDeg = +45.0f;
343          legs[i].hip.speedDeg = 25.0f;
344
345          legs[i].knee.maxDeg = 85.0f;
346          legs[i].knee.speedDeg = 30.0f;
347      }
348
349      // Cola: rango amplio
350      for (int i = 0; i < 3; ++i) {
351          tail[i].minDeg = -150.0f;
352          tail[i].maxDeg = +150.0f;
353          tail[i].speedDeg = 45.0f;
354      }
```

Vamos a usar algunas restricciones para limitar el movimiento de las extremidades, además de dar la ilusión de que nuestro animal se encuentra en reposo pero puede llegar a modificarse y dar la ilusión de correr.

Ejecución



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

Tuve problemas con las ruedas de la grúa pues el como acomodar el cilindro para dar la ilusión de que giraba me resulto complejo además de que no se lograba apreciar muy bien la llanta. Para no complicarme decidí hacer dos diseños de cilindro, el que estaba de base para dar esa textura y que se note el efecto de rotación y el formato que usamos en la práctica pasada para hacer un relleno del cilindro, esto me permitió entender bien la posición y el giro del cilindro. Tomando en cuenta lo aprendido me aventure a probar la rotación de la caja central y los brazos sin afectar al chasis y fue tan sencillo como usar de referencia la primera articulación y dar rotación a la caja. Con el gato investigue como marcar un tope en el movimiento y así poder hacer movimientos máximos en cada una de las articulaciones y brazos. Aunque considero que aun se pueden pulir los movimientos, especialmente el de la cola donde tuve que poder poner un movimiento libre en los 3 ejes pero se vuelve complejo y mas al trabajar 3 articulaciones.

3.- Conclusión:

La práctica me permitió conocer más sobre la jerarquía de objetos y también el como trabajar movimientos (rotaciones), teniendo puntos de referencia y que trabaje como una cadena que todos los estados dependen del anterior. En general fue una práctica desafiante, sobre todo en la segunda actividad donde teníamos que hacer un modelo desde cero; creo que se podría llegar a hacer mas movimientos en base a los vértices, pero para este primer acercamiento a las animaciones esta muy bien.

Bibliografía en formato APA

- Swiftless. (2010, March 25). *7. OpenGL Rotation and Translation (Version 2.0) – Swiftless Tutorials - Various older tutorials and newer programming musings*. Swiftless.com. https://www.swiftless.com/tutorials/opengl/rotation.html