Práctica 5 – Modelado Jerárquico

Para esta práctica el archivo comprimido tiene:

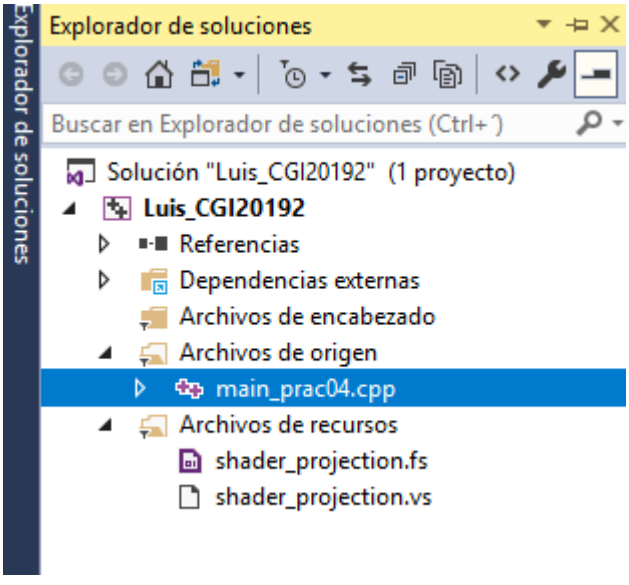
- Main_prac05.cpp
- Carpeta “shaders”

***La carpeta “shaders” contiene el shader que permite asignar un color de forma individual a cada uno de los cubos. Se supone que en la práctica 4 se debió haber construido. La práctica 4 hace uso de esa modificación de ese shader.

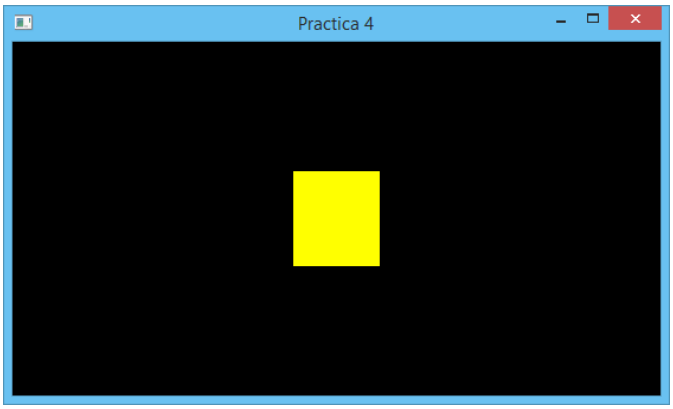
Se debe descomprimir en la carpeta del proyecto (si pide sobre escribir archivos indicar que Sí) y debe quedar así la estructura:

Debug	19/02/2019 20:13	Carpeta de archivos	
include	19/02/2019 20:12	Carpeta de archivos	
lib	29/01/2019 14:02	Carpeta de archivos	
shaders	19/02/2019 20:12	Carpeta de archivos	
CGeIHC20192.vcxproj	19/02/2019 20:13	VC++ Project	7 KB
CGeIHC20192.vcxproj.filters	19/02/2019 20:13	Archivo FILTERS	1 KB
CGeIHC20192.vcxproj.user	29/01/2019 14:01	Archivo USER	1 KB
Configuración.docx	29/01/2019 14:35	Documento de Mi...	498 KB
glew32.dll	09/01/2019 21:55	Extensión de la apl...	381 KB
glfw3.dll	09/01/2019 21:56	Extensión de la apl...	70 KB
main_prac01.cpp	18/02/2019 12:55	C++ Source	7 KB
main_prac02_lore.cpp	19/02/2019 11:29	C++ Source	9 KB
main_prac03.cpp	19/02/2019 20:13	C++ Source	7 KB
practica01_v02.cpp	26/01/2019 14:34	C++ Source	7 KB

En Visual Studio, en el Explorador de Soluciones, se debe agregar el main_prac05.cpp y quitar el de prácticas pasadas,



El código no debería de dar problemas y lo que se debe apreciar al ejecutar el programa es:



En la práctica 3 ya le había enseñado la operación de Traslación. Esta operación se sigue realizando con las teclas W, A, S, D, Av Pag y Re Pág.

Ahora les enseñaré la transformación geométrica de Rotación, así que al código de dibujo:

```
130     //Use "view" in order to affect all models
131     view = glm::translate(view, glm::vec3(movX, movY, movZ));
132
```

Se le adiciona una rotación, en mi caso puse una rotación con una variable sobre el eje X:

```
136     //Use "view" in order to affect all models
137     view = glm::translate(view, glm::vec3(movX, movY, movZ));
138     view = glm::rotate(view, glm::radians(rotX), glm::vec3(1.0f, 0.0f, 0.0f));
```

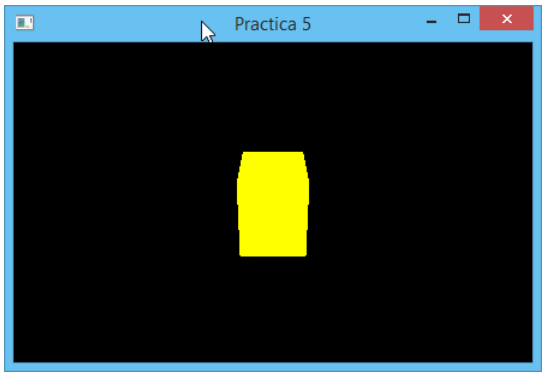
Recordar que se debe declarar una variable global:

```
31      //For Keyboard
32      float  movX = 0.0f,
33             movY = 0.0f,
34             movZ = -5.0f,
35             rotX = 0.0f;
```

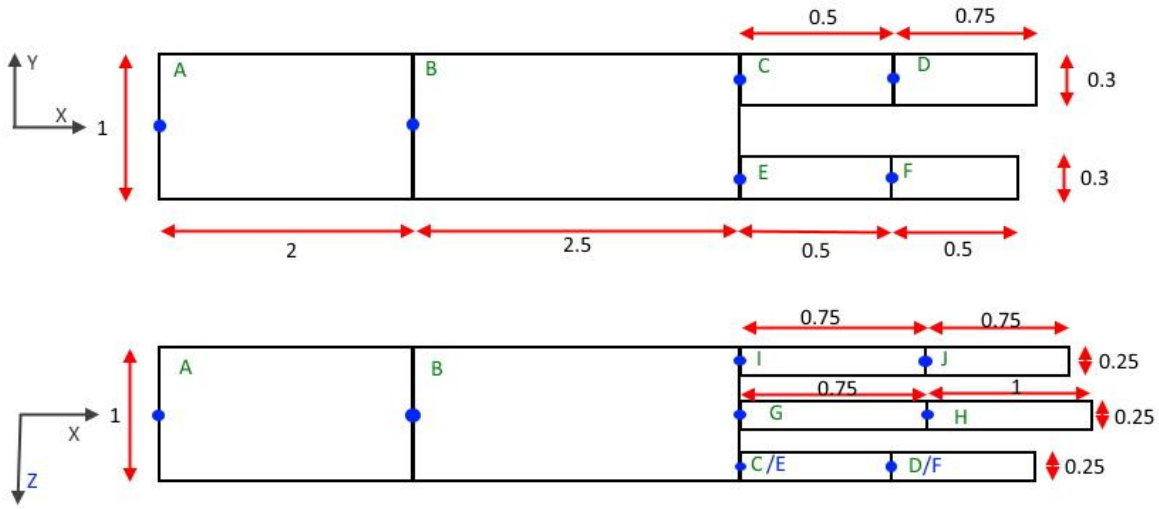
Y se le asigna teclas para modificar la variable, yo utilizaré las teclas de flechas arriba y abajo, dentro de la función *my_input()*:

```
309     if (glfwGetKey(window, GLFW_KEY_DOWN) == GLFW_PRESS)
310         rotX += 0.18f;
311     if (glfwGetKey(window, GLFW_KEY_UP) == GLFW_PRESS)
312         rotX -= 0.18f;
```

El resultado es:



El objetivo de la práctica es crear el brazo robot:



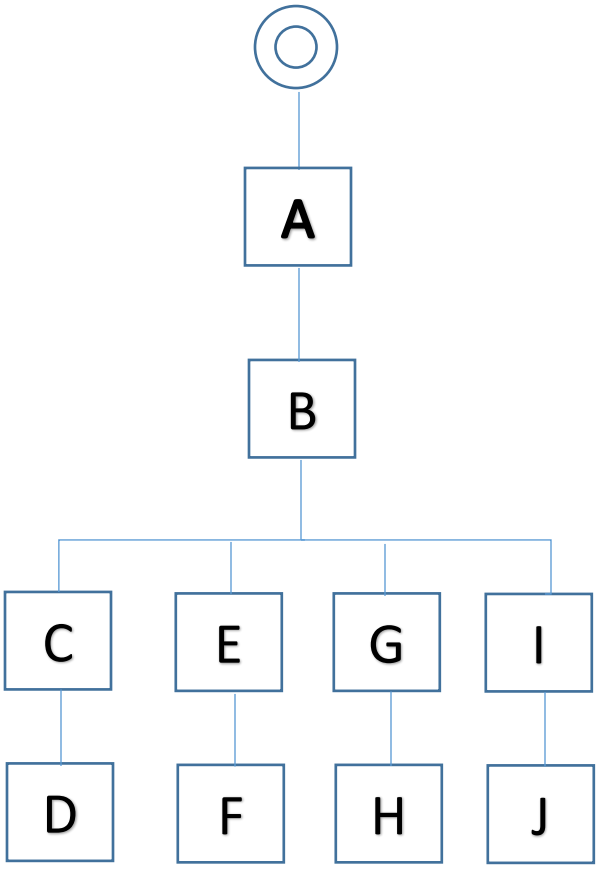
Algunas observaciones:

Por la complejidad del elemento se tienen dos vistas, primero está la vista frontal (ejes X, Y) y la vista superior (ejes X, Z).

El brazo robot tendrá asociadas algunas “animaciones”, las cuales consisten en las articulaciones del elemento, en este caso representadas por los puntos de color azul, simulando ser los movimientos del hombro, codo, y de los dedos.

Debido a estas mismas articulaciones, se debe definir la jerarquía de la construcción, es decir, la forma en que se transmite el movimiento. Poniendo la analogía del brazo humano podemos ver que al mover la articulación del hombro, ésta articulación afecta a todos los elementos del brazo, por lo cual se le considera de mayor jerarquía, en cambio, al mover una articulación que corresponda a los dedos, los elementos que son afectados son menos,

por lo cual está más abajo en la jerarquía. Esto se representa por el siguiente diagrama jerárquico.



Recomiendo comenzar declarando las variables que servirán para las rotaciones (hombro, codo, etc.) como globales:

```
31      //For Keyboard
32      float   movX = 0.0f,
33              movY = 0.0f,
34              movZ = -5.0f,
35              rotX = 0.0f;
36
37      float   hombro = 0.0f,
38              codo = 0.0f,
39              muneca = 0.0f,
40              dedo1 = 45.0f,
41              dedo2 = -45.0f;
```

Para trabajar con Jerarquía, YO recomiendo utilizar matrices temporales que ayudarán a determinar el lugar a partir del cual vamos realizar las operaciones, según yo, para el brazo necesitamos dos matrices temporales, las creo en la función *display()*:

```
124     // create transformations and Projection
125     glm::mat4 modelTemp = glm::mat4(1.0f); //Temp
126     glm::mat4 modelTemp2 = glm::mat4(1.0f); //Temp
127     glm::mat4 model = glm::mat4(1.0f);      //initialize Matrix, Use this matrix f
128     glm::mat4 view = glm::mat4(1.0f);       //Use this matrix for ALL models
129     glm::mat4 projection = glm::mat4(1.0f);//This matrix is for Projection
```

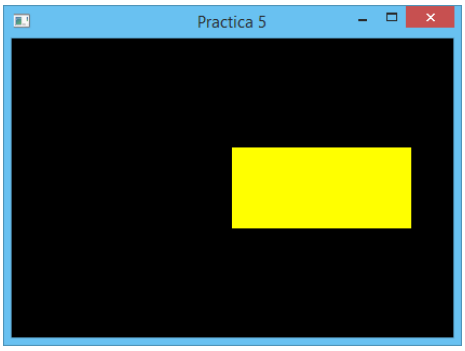
Comienzo la construcción de la figura con la rotación que simulará ser el **Hombro**, para ello creo una rotación sobre el eje Z:

```
145     glBindVertexArray(VAO);
146     model = glm::rotate(model, glm::radians(hombro), glm::vec3(0.0f, 0.0, 1.0f)); //hombro
147
148     projectionShader.setMat4("model", model);
149     projectionShader.setVec3("aColor", glm::vec3(1.0f, 1.0f, 0.0f));
150     glDrawArrays(GL_QUADS, 0, 24); //A
```

Ya con la rotación que simula ser la articulación, procedo a construir mi objeto A, para ello debo llegar a su centro geométrico y aplicar la escala:

```
145     glBindVertexArray(VAO);
146     model = glm::rotate(model, glm::radians(hombro), glm::vec3(0.0f, 0.0, 1.0f)); //hombro
147     model = glm::translate(model, glm::vec3(1.5f, 0.0f, 0.0f));
148     model = glm::scale(model, glm::vec3(3.0f, 1.0f, 1.0f));
149     projectionShader.setMat4("model", model);
150     projectionShader.setVec3("aColor", glm::vec3(1.0f, 1.0f, 0.0f));
151     glDrawArrays(GL_QUADS, 0, 24); //A
```

Con eso obtengo la figura:



Falta hacer que la articulación se mueva con ayuda de una tecla, para ello se debe ir ala función de *my_Input():*

```
310     if (glfwGetKey(window, GLFW_KEY_DOWN) == GLFW_PRESS)
311         rotX += 0.18f;
312     if (glfwGetKey(window, GLFW_KEY_UP) == GLFW_PRESS)
313         rotX -= 0.18f;
314     if (glfwGetKey(window, GLFW_KEY_R) == GLFW_PRESS)
315         hombro += 0.18f;
316     if (glfwGetKey(window, GLFW_KEY_F) == GLFW_PRESS)
317         hombro -= 0.18f;
```

Ahora viene la parte, según yo, importante y difícil de explicar, que es el uso de la matriz temporal.

Se supone que en la sesión anterior vieron que las operaciones se acumulan y que también se puede “resetear”, utilizando la matriz identidad, sin embargo, ahora no nos interesa:

1. Conservar la escala de la Figura A, ni
2. Regresar al origen.

Lo que necesitamos es **Conservar la rotación del hombro**, por lo cual la solución es utilizar una matriz temporal en un valor donde se tenga ya aplicada la rotación del hombro. En mi caso se me hace sencillo el conservar hasta donde llegamos al centro de la figura A, ya que es antes de la escala pero después de la rotación, así que ahí utilizaré la primera matriz temporal:

```
145     glBindVertexArray(VAO);
146     model = glm::rotate(model, glm::radians(hombro), glm::vec3(0.0f, 0.0, 1.0f)); //hombro
147     modelTemp = model = glm::translate(model, glm::vec3(1.5f, 0.0f, 0.0f));
148     model = glm::scale(model, glm::vec3(3.0f, 1.0f, 1.0f));
149     projectionShader.setMat4("model", model);
150     projectionShader.setVec3("aColor", glm::vec3(1.0f, 1.0f, 0.0f));
151     glDrawArrays(GL_QUADS, 0, 24); //A
```

Así que la construcción del siguiente elemento lo hago desde esa posición así que pongo en el código:

```
145     glBindVertexArray(VAO);
146     model = glm::rotate(model, glm::radians(hombro), glm::vec3(0.0f, 0.0, 1.0f)); //hombro
147     modelTemp = model = glm::translate(model, glm::vec3(1.5f, 0.0f, 0.0f));
148     model = glm::scale(model, glm::vec3(3.0f, 1.0f, 1.0f));
149     projectionShader.setMat4("model", model);
150     projectionShader.setVec3("aColor", glm::vec3(1.0f, 1.0f, 0.0f));
151     glDrawArrays(GL_QUADS, 0, 24); //A
152
153     model = glm::translate(modelTemp, glm::vec3(1.5f, 0.0f, 0.0f));
```

Esa traslación, la hago desde la primera matriz temporal y me ayuda a ubicarme entre la figura A y B, para ahí colocar la articulación del **Codo:**

```
152
153     model = glm::translate(modelTemp, glm::vec3(1.5f, 0.0f, 0.0f));
154     model = glm::rotate(model, glm::radians(codo), glm::vec3(0.0f, 1.0, 0.0f));
```

Dicha articulación quiero que gire sobre el eje Y.

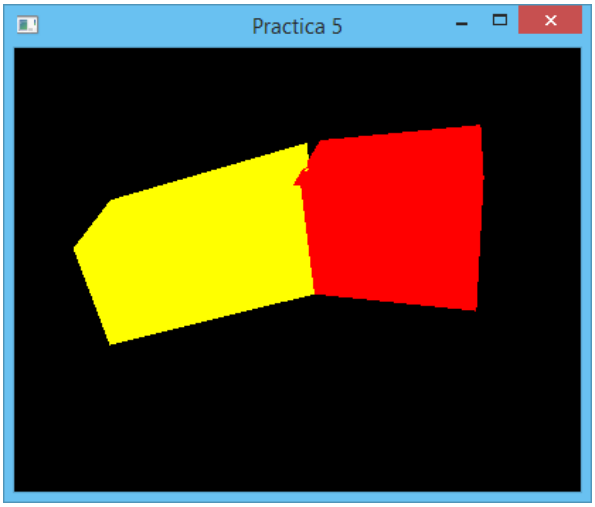
Ya con la articulación del codo, procedo a moverme al centro de la figura B, colocar su escala y dibujar:

```
149     projectionShader.setMat4("model", model);
150     projectionShader.setVec3("aColor", glm::vec3(1.0f, 1.0f, 0.0f));
151     glDrawArrays(GL_QUADS, 0, 24); //A
152
153     model = glm::translate(modelTemp, glm::vec3(1.5f, 0.0f, 0.0f));
154     model = glm::rotate(model, glm::radians(codo), glm::vec3(0.0f, 1.0, 0.0f));//Codo
155     model = glm::translate(model, glm::vec3(1.0f, 0.0f, 0.0f));
156     model = glm::scale(model, glm::vec3(2.0f, 1.0f, 1.0f));
157     projectionShader.setMat4("model", model);
158     projectionShader.setVec3("aColor", glm::vec3(1.0f, 0.0f, 0.0f));
159     glDrawArrays(GL_QUADS, 0, 24); //B
```

La variable **codo** se debe modificar también desde teclado:

```
316     if (glfwGetKey(window, GLFW_KEY_F) == GLFW_PRESS)
317         hombro -= 0.18f;
318     if (glfwGetKey(window, GLFW_KEY_T) == GLFW_PRESS)
319         codo += 0.18f;
320     if (glfwGetKey(window, GLFW_KEY_G) == GLFW_PRESS)
321         codo -= 0.18f;
```

Hasta aquí se obtiene:



Recomendación: invitar a que los alumnos limiten el movimiento del *codo* y *hombro*, hombre hasta 90 grados hacia arriba y 110 grados hacia abajo. El codo 90 grados hacia adentro, y 5 grados hacia afuera.

Se vuelve a utilizar la primera matriz temporal, para determinar el lugar desde el cual se desea iniciar las siguientes operaciones, yo recomiendo que sea antes de la escala de la figura B, y después de la rotación del codo, así que propongo:

```
153     model = glm::translate(modelTemp, glm::vec3(1.5f, 0.0f, 0.0f));
154     model = glm::rotate(model, glm::radians(codo), glm::vec3(0.0f, 1.0, 0.0f));//Codo
155     modelTemp = model = glm::translate(model, glm::vec3(1.0f, 0.0f, 0.0f));
156     model = glm::scale(model, glm::vec3(2.0f, 1.0f, 1.0f));
157     projectionShader.setMat4("model", model);
158     projectionShader.setVec3("aColor", glm::vec3(1.0f, 0.0f, 0.0f));
159     glDrawArrays(GL_QUADS, 0, 24); //B
```

Realizo algo parecido para el elemento C, es decir, me muevo al punto donde quiero la rotación, coloco la rotación (**muneca**), me muevo al centro de C, aplico la escala y dibujo:

```
157     projectionShader.setMat4("model", model);
158     projectionShader.setVec3("aColor", glm::vec3(1.0f, 0.0f, 0.0f));
159     glDrawArrays(GL_QUADS, 0, 24); //B
160
161     model = glm::translate(modelTemp, glm::vec3(1.0f, 0.0f, 0.0f));
162     model = glm::rotate(model, glm::radians(muneca), glm::vec3(1.0f, 0.0, 0.0f));//muñeca
163     model = glm::translate(model, glm::vec3(0.25f, 0.0f, 0.0f));
164     model = glm::scale(model, glm::vec3(0.5f, 1.0f, 1.0f));
165     projectionShader.setMat4("model", model);
166     projectionShader.setVec3("aColor", glm::vec3(1.0f, 1.0f, 1.0f));
167     glDrawArrays(GL_QUADS, 0, 24); //C
168
```

A la variable **muñeca** debo modificarla desde teclado:

```
320         codo += 0.18f;
321     if (glfwGetKey(window, GLFW_KEY_G) == GLFW_PRESS)
322         codo -= 0.18f;
323     if (glfwGetKey(window, GLFW_KEY_Y) == GLFW_PRESS)
324         muneca += 0.18f;
325     if (glfwGetKey(window, GLFW_KEY_H) == GLFW_PRESS)
326         muneca -= 0.18f;
```

Vuelvo a utilizar mi primera matriz temporal:

```
161     model = glm::translate(modelTemp, glm::vec3(1.0f, 0.0f, 0.0f));
162     model = glm::rotate(model, glm::radians(muneca), glm::vec3(1.0f, 0.0, 0.0f));//muñeca
163     modelTemp = model = glm::translate(model, glm::vec3(0.25f, 0.0f, 0.0f));
164     model = glm::scale(model, glm::vec3(0.5f, 1.0f, 1.0f));
165     projectionShader.setMat4("model", model);
166     projectionShader.setVec3("aColor", glm::vec3(1.0f, 1.0f, 1.0f));
167     glDrawArrays(GL_QUADS, 0, 24); //C
```

Y construyo dedos:

```
172     //Dedo 1
173     model = glm::translate(modelTemp, glm::vec3(0.25f, 0.35f, 0.375f));
174     model = glm::rotate(model, glm::radians(dedo1), glm::vec3(0.0f, 0.0, 1.0f));
175     modelTemp = model = glm::translate(model, glm::vec3(0.5f, 0.0f, 0.0f));
176     model = glm::scale(model, glm::vec3(1.0f, 0.3f, 0.25f));
177     projectionShader.setMat4("model", model);
178     projectionShader.setVec3("aColor", glm::vec3(0.0f, 1.0f, 0.0f));
179     glDrawArrays(GL_QUADS, 0, 24); //Primera Parte
180
181     model = glm::translate(modelTemp, glm::vec3(0.5f, 0.0f, 0.0f));
182     model = glm::rotate(model, glm::radians(dedo2), glm::vec3(0.0f, 0.0, 1.0f));
183     model = glm::translate(model, glm::vec3(0.5f, 0.0f, 0.0f));
184     model = glm::scale(model, glm::vec3(1.0f, 0.3f, 0.25f));
185     projectionShader.setMat4("model", model);
186     projectionShader.setVec3("aColor", glm::vec3(1.0f, 1.0f, 1.0f));
187     glDrawArrays(GL_QUADS, 0, 24); //Segunda parte
```

Importante 2: Aquí viene el uso de la segunda matriz temporal. ¿Por qué no utilizar la primera matriz temporal? La primera matriz temporal, si se revisa el código, se utilizó para construir la segunda parte del dedo a partir de la primera parte. Si para construir el siguiente dedo se utiliza la primera matriz temporal lo que estaríamos haciendo es que el segundo dedo dependerá de la primer parte del primer dedo. *Espero que esto no suene confuso.*

Como los dedos son independientes entre sí, o al menos eso se intentaría, se debe construir el siguiente dedo, sin tomar en cuenta nada del primer dedo, así que yo recomiendo hacer:

```
158     projectionShader.setVec3("aColor", glm::vec3(1.0f, 0.0f, 0.0f));
159     glDrawArrays(GL_QUADS, 0, 24); //B
160
161     model = glm::translate(modelTemp, glm::vec3(1.0f, 0.0f, 0.0f));
162     model = glm::rotate(model, glm::radians(muneca), glm::vec3(1.0f, 0.0, 0.0f));//muñeca
163     modelTemp2 = modelTemp = model = glm::translate(model, glm::vec3(0.25f, 0.0f, 0.0f));
164     model = glm::scale(model, glm::vec3(0.5f, 1.0f, 1.0f));
165     projectionShader.setMat4("model", model);
166     projectionShader.setVec3("aColor", glm::vec3(1.0f, 1.0f, 1.0f));
167     glDrawArrays(GL_QUADS, 0, 24); //C
168
169     //Dedo 1
170     model = glm::translate(modelTemp, glm::vec3(0.25f, 0.35f, 0.375f));
```

Retornar hasta el centro de la muñeca, y ahí colocar ambas matrices temporales, y recuperar la matriz temporal dos, en la construcción de los demás dedos, es decir algo como:

```
186     //Dedo 2
187     model = glm::translate(modelTemp2, glm::vec3(0.25f, 0.35f, 0.0f));
188     model = glm::rotate(model, glm::radians(dedo1), glm::vec3(0.0f, 0.0, 1.0f));
189     modelTemp = model = glm::translate(model, glm::vec3(0.75f, 0.0f, 0.0f));
190     model = glm::scale(model, glm::vec3(1.5f, 0.3f, 0.25f));
191     projectionShader.setMat4("model", model);
192     projectionShader.setVec3("aColor", glm::vec3(0.0f, 1.0f, 0.0f));
193     glDrawArrays(GL_QUADS, 0, 24); //dedo1
194
195     model = glm::translate(modelTemp, glm::vec3(0.75f, 0.0f, 0.0f));
196     model = glm::rotate(model, glm::radians(dedo2), glm::vec3(0.0f, 0.0, 1.0f));
197     model = glm::translate(model, glm::vec3(0.5f, 0.0f, 0.0f));
198     model = glm::scale(model, glm::vec3(1.0f, 0.3f, 0.25f));
199     projectionShader.setMat4("model", model);
200     projectionShader.setVec3("aColor", glm::vec3(1.0f, 1.0f, 1.0f));
201     glDrawArrays(GL_QUADS, 0, 24); //dedo2
```

```
//Dedo 3
model = glm::translate(modelTemp2, glm::vec3(0.25f, 0.35f, -0.375f));
model = glm::rotate(model, glm::radians(dedo1), glm::vec3(0.0f, 0.0, 1.0f));
modelTemp = model = glm::translate(model, glm::vec3(0.5f, 0.0f, 0.0f));
model = glm::scale(model, glm::vec3(1.0f, 0.3f, 0.25f));
projectionShader.setMat4("model", model);
projectionShader.setVec3("aColor", glm::vec3(0.0f, 1.0f, 0.0f));
glDrawArrays(GL_QUADS, 0, 24); //parte1

model = glm::translate(modelTemp, glm::vec3(0.5f, 0.0f, 0.0f));
model = glm::rotate(model, glm::radians(dedo2), glm::vec3(0.0f, 0.0, 1.0f));
model = glm::translate(model, glm::vec3(0.375f, 0.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.75f, 0.3f, 0.25f));
projectionShader.setMat4("model", model);
projectionShader.setVec3("aColor", glm::vec3(1.0f, 1.0f, 1.0f));
glDrawArrays(GL_QUADS, 0, 24); //parte2*/
```

Me queda: