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 03

NOMBRE COMPLETO: Vázquez Reyes Sebastián

Nº de Cuenta: 318150923

GRUPO DE LABORATORIO: 11

GRUPO DE TEORÍA: 6

SEMESTRE 2024-2

FECHA DE ENTREGA LÍMITE: miércoles 28 de febrero de 2024

CALIFICACIÓN: _____

REPORTE DE PRÁCTICA:

1.- Generar una pirámide rubik (pyraminx) de 9 pirámides por cara. Cada cara de la pyraminx que se vea de un color diferente y que se vean las separaciones entre instancias (las líneas oscuras son las que permiten diferenciar cada pirámide pequeña)

Antes de comenzar el reporte, debo comentar algo. Por error, confundí la figura que debíamos formar. Pensé que el pyraminx tenia 5 caras en realidad y no 4, es decir, era una pirámide cuadrangular en vez de triangular (la verdad nunca he visto una en vivo y directo y realmente no conocía la figura, y el video de la figura proporcionado no me cargo hasta unos cuantos días después de empezar la práctica). Me di cuenta de mi error muy tarde como para poder corregirlo, por lo que decidí mejor continuar con el trabajo que ya tenía hecho.

Para este trabajo, coloque diversas pirámides dentro de una sola pirámide enorme de color negro, de esta forma cree las separaciones entre cada triangulo pequeño.

Para empezar, cree la piramide enorme de color negro en el siguiente codigo:

```
model = glm::mat4(1.0);
model = glm::translate(model, origen);//Trasladamos la figura al punto de origen, luego la devolvem
model = glm::rotate(model, glm::radians(mainWindow.getrotax()), glm::vec3(1.0f, 0.0f, 0.0f)); //E
model = glm::rotate(model, glm::radians(mainWindow.getrotay()), glm::vec3(0.0f, 1.0f, 0.0f)); //R /
model = glm::rotate(model, glm::radians(mainWindow.getrotaz()), glm::vec3(0.0f, 0.0f, 1.0f)); //T
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -8.0f));
model = glm::scale(model, glm::vec3(30.0f, 30.0f, 30.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
//la línea de proyección solo se manda una vez a menos que en tiempo de ejecución
//se programe cambio entre proyección ortogonal y perspectiva
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
glUniformMatrix4fv(uniformView, 1, GL_FALSE, glm::value_ptr(camera.calculateViewMatrix()));

//BASE PIRAMIDE NEGRA

color = glm::vec3(0.0f, 0.0f, 0.0f);
glUniform3fv(uniformColor, 1, glm::value_ptr(color)); //para cambiar el color del objetos
meshList[4]->RenderMesh(); //dibuja cubo y pirámide triangular Y pirámide base cuadrangular
```

Luego, cree un total de 9 piramides por lado, y por cara se coloreaban de un color distinto. El proceso de creacion de estas 9 piramides es el mismo para todas las caras, asi que solo explicare y mostraré el codigo de una cara en este documento. Primero cree la punta de cada cara:

```
//CARA FRONTAL ROJA
//PUNTA
model = glm::mat4(1.0);
model = glm::translate(model, origen);
model = glm::rotate(model, glm::radians(mainWindow.getrotax()), glm::vec3(1.0f, 0.0f, 0.0f)); //E
model = glm::rotate(model, glm::radians(mainWindow.getrotay()), glm::vec3(0.0f, 1.0f, 0.0f)); //R //a
model = glm::rotate(model, glm::radians(mainWindow.getrotaz()), glm::vec3(0.0f, 0.0f, 1.0f)); //T
color = glm::vec3(1.0f, 0.0f, 0.0f);
model = glm::translate(model, glm::vec3(0.0f, +8.5f, -6.8f));
model = glm::scale(model, glm::vec3(9.0f, 9.0f, 9.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[4]->RenderMesh();
```

Después, cree los 3 triangulos que componen el cuerpo medio de la cara:

```
//MEDIO
model = glm::mat4(1.0);
model = glm::translate(model, origen);
model = glm::rotate(model, glm::radians(mainWindow.getrotax()), glm::vec3(1.0f, 0.0f, 0.0f)); //E
model = glm::rotate(model, glm::radians(mainWindow.getrotay()), glm::vec3(0.0f, 1.0f, 0.0f)); //R //
model = glm::rotate(model, glm::radians(mainWindow.getrotaz()), glm::vec3(0.0f, 0.0f, 1.0f)); //T
model = glm::translate(model, glm::vec3(0.0f, -2.8f, -1.2f));
model = glm::rotate(model, glm::radians(126.5f), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::scale(model, glm::vec3(8.8f, 8.8f, 8.8f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[4]->RenderMesh();

model = glm::mat4(1.0);
model = glm::translate(model, origen);
model = glm::rotate(model, glm::radians(mainWindow.getrotax()), glm::vec3(1.0f, 0.0f, 0.0f)); //E
model = glm::rotate(model, glm::radians(mainWindow.getrotay()), glm::vec3(0.0f, 1.0f, 0.0f)); //R //
model = glm::rotate(model, glm::radians(mainWindow.getrotaz()), glm::vec3(0.0f, 0.0f, 1.0f)); //T
model = glm::translate(model, glm::vec3(-4.8f, -1.2f, -2.0f));
model = glm::scale(model, glm::vec3(8.5f, 8.5f, 8.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[4]->RenderMesh();

model = glm::mat4(1.0);
model = glm::translate(model, origen);
model = glm::rotate(model, glm::radians(mainWindow.getrotax()), glm::vec3(1.0f, 0.0f, 0.0f)); //E
model = glm::rotate(model, glm::radians(mainWindow.getrotay()), glm::vec3(0.0f, 1.0f, 0.0f)); //R //
model = glm::rotate(model, glm::radians(mainWindow.getrotaz()), glm::vec3(0.0f, 0.0f, 1.0f)); //T
model = glm::translate(model, glm::vec3(+4.8f, -1.2f, -2.0f));
model = glm::scale(model, glm::vec3(8.5f, 8.5f, 8.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[4]->RenderMesh();
```

Y por ultimo, los triangulos que conforman la parte inferior de la cara:

```
//BASE
model = glm::mat4(1.0);
model = glm::translate(model, origen);
model = glm::rotate(model, glm::radians(mainWindow.getrotax()), glm::vec3(1.0f, 0.0f, 0.0f)); //E
model = glm::rotate(model, glm::radians(mainWindow.getrotay()), glm::vec3(0.0f, 1.0f, 0.0f)); //R /
model = glm::rotate(model, glm::radians(mainWindow.getrotaz()), glm::vec3(0.0f, 0.0f, 1.0f)); //T
model = glm::translate(model, glm::vec3(-9.3f, -10.0f, +2.5f));
model = glm::scale(model, glm::vec3(8.5f, 8.5f, 8.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[4]->RenderMesh();

model = glm::mat4(1.0);
model = glm::translate(model, origen);
model = glm::rotate(model, glm::radians(mainWindow.getrotax()), glm::vec3(1.0f, 0.0f, 0.0f)); //E
model = glm::rotate(model, glm::radians(mainWindow.getrotay()), glm::vec3(0.0f, 1.0f, 0.0f)); //R /
model = glm::rotate(model, glm::radians(mainWindow.getrotaz()), glm::vec3(0.0f, 0.0f, 1.0f)); //T
model = glm::translate(model, glm::vec3(+9.3f, -10.0F, +2.5f));
model = glm::scale(model, glm::vec3(8.5f, 8.5f, 8.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[4]->RenderMesh();

model = glm::mat4(1.0);
model = glm::translate(model, origen);
model = glm::rotate(model, glm::radians(mainWindow.getrotax()), glm::vec3(1.0f, 0.0f, 0.0f)); //E
model = glm::rotate(model, glm::radians(mainWindow.getrotay()), glm::vec3(0.0f, 1.0f, 0.0f)); //R /
model = glm::rotate(model, glm::radians(mainWindow.getrotaz()), glm::vec3(0.0f, 0.0f, 1.0f)); //T
model = glm::translate(model, glm::vec3(0.0F, -10.0F, +2.5f));
model = glm::scale(model, glm::vec3(8.5f, 8.5f, 8.5f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[4]->RenderMesh();
```

```
model = glm::mat4(1.0);
model = glm::translate(model, origen);
model = glm::rotate(model, glm::radians(mainWindow.getrotax()), glm::vec3(1.0f, 0.0f, 0.0f)); //E
model = glm::rotate(model, glm::radians(mainWindow.getrotay()), glm::vec3(0.0f, 1.0f, 0.0f)); //R //
model = glm::rotate(model, glm::radians(mainWindow.getrotaz()), glm::vec3(0.0f, 0.0f, 1.0f)); //T
model = glm::translate(model, glm::vec3(-4.65f, -11.7f, 3.4f));
model = glm::rotate(model, glm::radians(126.8f), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::scale(model, glm::vec3(8.2f, 8.2f, 8.2f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[4]->RenderMesh();

model = glm::mat4(1.0);
model = glm::translate(model, origen);
model = glm::rotate(model, glm::radians(mainWindow.getrotax()), glm::vec3(1.0f, 0.0f, 0.0f)); //E
model = glm::rotate(model, glm::radians(mainWindow.getrotay()), glm::vec3(0.0f, 1.0f, 0.0f)); //R //
model = glm::rotate(model, glm::radians(mainWindow.getrotaz()), glm::vec3(0.0f, 0.0f, 1.0f)); //T
model = glm::translate(model, glm::vec3(+4.65f, -11.7f, 3.4f));
model = glm::rotate(model, glm::radians(126.8f), glm::vec3(1.0f, 0.0f, 0.0f));
model = glm::scale(model, glm::vec3(8.2f, 8.2f, 8.2f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[4]->RenderMesh();
```

Como dije, este proceso se repite para las 4 caras de forma triangular en la piramide. Lo unico que cambia son las coordenadas de traslado y el color.

Posteriormente, tambien coloque cubos blancos en la parte inferior de la piramide, la base con forma de cuadrado. Para esta parte, primero coloque un cuadrado negro cubriendo la parte inferior de la piramide, porque se alcanzaban a ver las figuras que utilice para elaborar las caras triangulares. Este cuadrado se define en el siguiente codigo:

```
805     //PARTE INFERIOR DE LA PIRAMIDE
806
807     model = glm::mat4(1.0);
808     model = glm::translate(model, origen);
809     model = glm::rotate(model, glm::radians(mainWindow.getrotax()), glm::vec3(1.0f, 0.0f, 0.0f)); //E
810     model = glm::rotate(model, glm::radians(mainWindow.getrotay()), glm::vec3(0.0f, 1.0f, 0.0f)); //R //
811     model = glm::rotate(model, glm::radians(mainWindow.getrotaz()), glm::vec3(0.0f, 0.0f, 1.0f)); //T
812     color = glm::vec3(0.0f, 0.0f, 0.0f);
813     model = glm::translate(model, glm::vec3(0.0f, -15.25f, -8.0f));
814     model = glm::scale(model, glm::vec3(29.0f, 1.0f, 29.0f));
815     glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
816     glUniform3fv(uniformColor, 1, glm::value_ptr(color));
817     meshList[0]->RenderMesh();
```

Luego, cree 9 cuadrados blancos que fueron traslados de forma unica para que la parte inferior de esta piramide tambien se asemejara al resto de las caras. Para los cuadrados blancos, se uso el siguiente codigo, unicamente cambian las coordenadas de traslado para cada figura:

```
model = glm::mat4(1.0);
model = glm::translate(model, origen);
model = glm::rotate(model, glm::radians(mainWindow.getrotax()), glm::vec3(1.0f, 0.0f, 0.0f)); //E
model = glm::rotate(model, glm::radians(mainWindow.getrotay()), glm::vec3(0.0f, 1.0f, 0.0f)); //R //
model = glm::rotate(model, glm::radians(mainWindow.getrotaz()), glm::vec3(0.0f, 0.0f, 1.0f)); //T
color = glm::vec3(1.0f, 1.0f, 1.0f);
model = glm::translate(model, glm::vec3(+9.6f, -15.0f, +1.6f));
model = glm::scale(model, glm::vec3(8.0f, 2.0f, 8.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
meshList[0]->RenderMesh();
```

Por último, también cree un fondo blanco para observar mejor las divisiones negras entre cada figura. El código que crea este fondo es el siguiente:

```
//FONDO
model = glm::mat4(1.0);
color = glm::vec3(1.0, 1.0, 1.0);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
model = glm::translate(model, glm::vec3(-30.0f, -30.0f, -30.0f));
model = glm::scale(model, glm::vec3(200.0f, 200.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
meshList[0]->RenderMesh();

model = glm::mat4(1.0);
color = glm::vec3(1.0, 1.0, 1.0);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
model = glm::translate(model, glm::vec3(-30.0f, -30.0f, +30.0f));
model = glm::scale(model, glm::vec3(200.0f, 200.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
meshList[0]->RenderMesh();

model = glm::mat4(1.0);
color = glm::vec3(1.0, 1.0, 1.0);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
model = glm::translate(model, glm::vec3(+50.0f, -10.0f, +00.0f));
model = glm::scale(model, glm::vec3(0.0f, 200.0f, 200.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
meshList[0]->RenderMesh();

model = glm::mat4(1.0);
color = glm::vec3(1.0, 1.0, 1.0);
glUniform3fv(uniformColor, 1, glm::value_ptr(color));
model = glm::translate(model, glm::vec3(+0.0f, -50.0f, +00.0f));
model = glm::scale(model, glm::vec3(200.0f, 0.0f, 200.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
meshList[0]->RenderMesh();
```

La ejecución del programa se encuentra en el video adjunto.

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

Para no complicar mi existencia esta vez, decidí copiar las líneas que permitían rotar con las teclas r, t y e a la figura sobre un determinado eje para todas las figuras creadas. El código quedo muy largo gracias a esto, pero trasladar y rotar las figuras esta vez fue más fácil, y también interactuar en el ambiente gráfico fue más sencillo. Además, obviamente no es lo que se pidió exactamente, pero se asemeja un poco.

3.- Conclusión:

a. Los ejercicios del reporte: Complejidad, Explicación.

Creo que realizar el ejercicio con una pirámide cuadrangular como base que con una triangular es algo más fácil, cuando creas el código que coloca los triángulos en una de las caras, el código de la contraparte de dicha cara es básicamente el mismo, únicamente cambiando las coordenadas del eje X o Z, ya sea disminuyéndolas o aumentándolas; básicamente solo creas el código

de 2 caras en vez de 4 como imagino que es en una pirámide triangular. Por otro lado, fue entretenido crear la figura, fue complicado en un inicio averiguar las coordenadas correctas para incrustar las pirámides en la pirámide grandota negra, y también fue un lio averiguar el ángulo exacto en que rotar las pirámides que deben de estar de cabeza en las caras, pero después de construir una sola cara, el resto fue mas sencillo. La parte inferior de la pirámide no fue complicada, pues solo fue crear cubos y colocarlos en cierta posición, menos difícil de averiguar que la de las pirámides pequeñas. Lo mismo pasa con los fondos blancos que agregué.

b. Comentarios generales: Faltó explicar a detalle, ir más lento en alguna explicación, otros comentarios y sugerencias para mejorar desarrollo de la práctica

Explicar el código durante la sesión me ahorro muchísimo tiempo para la realización de la práctica y me ayudo a entenderla mucho mejor, gracias a eso no me tomo mucho tiempo entender la construcción de las figuras en el entorno 3D y el manejo de la cámara y las rotaciones sobre los ejes en las figuras.

c. Conclusión

Al finalizar esta práctica aprendí a rotar, trasladar y escalar figuras en un ambiente 3D, también aprendí como interactúan unas con otras al ocupar la misma posición. Además, aprendí a crear fondos para los ambientes 3D mediante trasladado y escalado

Bibliografía

- Navarro, J., (2018). “SUPERFICIES CILÍNDRICAS Y CÓNICAS”. Recuperado el 21 de febrero de 2024 de https://proyectodescartes.org/uudd/materiales_didacticos/superficies_curiosas-1_JS/cono-cilindro.html