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 02

NOMBRE COMPLETO: Jiménez Enriquez Rubén Pedro

Nº de Cuenta: 318056832

GRUPO DE LABORATORIO: 03

GRUPO DE TEORÍA: 06

SEMESTRE 2026-2

FECHA DE ENTREGA LÍMITE: 1 de marzo del 2026

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.

I. **Ejercicio1.** Dibujar las iniciales de sus nombres, cada letra de un color diferente.

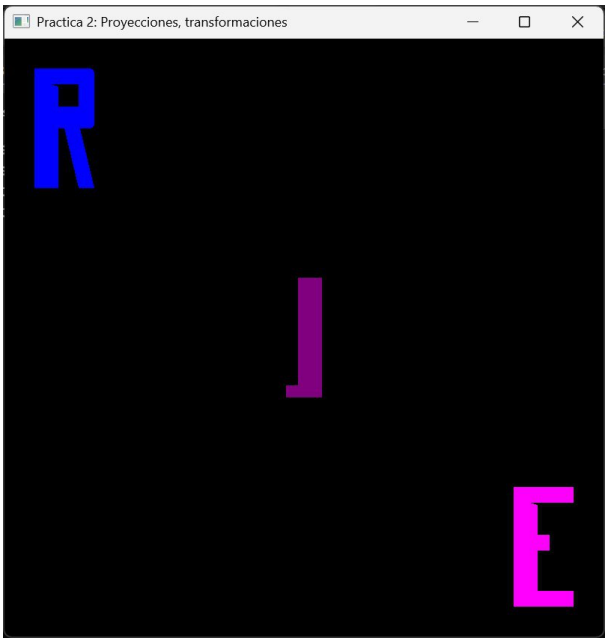
I.I. **Bloques de código generados.** Para llevar a cabo este ejercicio modifiqué dos secciones del programa. La primera fue la definición del vertice_letras[]. Únicamente colocaré la captura de un fragmento del código que modifiqué, ya que con varias líneas y realmente los vértices son los mismos que usé en la práctica 1, la diferencia radica en la segunda columna de flotantes, la cual permite asignar color a cada uno de los vértices.

```
void CrearLetrasyFiguras()
{
    GLfloat vertices_letras[] = {
            //X        Y          Z          R      G      B
            /*Letra R*/
            // t12 = Polygon(T, R, W1)
            -0.9f, 0.5f, 0.0f,              0.0f,  0.0f,  1.0f,
            -0.9f, 0.9f, 0.0f,              0.0f,  0.0f,  1.0f,
            -0.848f, 0.848f, 0.0f,          0.0f,  0.0f,  1.0f,

            // t13 = Polygon(W1, R, J1)
            -0.848f, 0.848f, 0.0f,          0.0f,  0.0f,  1.0f,
            -0.9f, 0.9f, 0.0f,              0.0f,  0.0f,  1.0f,
            -0.74f, 0.9f, 0.0f,             0.0f,  0.0f,  1.0f,

            // t14 = Polygon(W1, I1, J1)
            -0.848f, 0.848f, 0.0f,          0.0f,  0.0f,  1.0f,
            -0.82f, 0.848f, 0.0f,           0.0f,  0.0f,  1.0f,
            -0.74f, 0.9f, 0.0f,             0.0f,  0.0f,  1.0f,

            // t15 = Polygon(I1, J1, T1)
            -0.82f, 0.848f, 0.0f,           0.0f,  0.0f,  1.0f,
            -0.74f, 0.9f, 0.0f,             0.0f,  0.0f,  1.0f,
            -0.752f, 0.848f, 0.0f,          0.0f,  0.0f,  1.0f,

            // t16 = Polygon(T1, J1, K1)
            -0.752f, 0.848f, 0.0f,          0.0f,  0.0f,  1.0f,
            -0.74f, 0.9f, 0.0f,             0.0f,  0.0f,  1.0f,
            -0.72f, 0.9f, 0.0f,             0.0f,  0.0f,  1.0f,
            // t7 = Polygon(L, M, N)
            0.78f, -0.66f, 0.0f,            1.0f, 0.0f, 1.0f,
            0.82f, -0.66f, 0.0f,            1.0f, 0.0f, 1.0f,
            0.82f, -0.712f, 0.0f,           1.0f, 0.0f, 1.0f,

            // t8 = Polygon(O, P, G)
            0.78f, -0.712f, 0.0f,           1.0f, 0.0f, 1.0f,
            0.78f, -0.848f, 0.0f,           1.0f, 0.0f, 1.0f,
            0.7f, -0.9f, 0.0f,              1.0f, 0.0f, 1.0f,

            // t9 = Polygon(G, P, Q)
            0.7f, -0.9f, 0.0f,              1.0f, 0.0f, 1.0f,
            0.78f, -0.848f, 0.0f,           1.0f, 0.0f, 1.0f,
            0.9f, -0.848f, 0.0f,            1.0f, 0.0f, 1.0f,

            // t10 = Polygon(G, H, Q)
            0.7f, -0.9f, 0.0f,              1.0f, 0.0f, 1.0f,
            0.9f, -0.9f, 0.0f,              1.0f, 0.0f, 1.0f,
            0.9f, -0.848f, 0.0f,            1.0f, 0.0f, 1.0f,

            // t11 = Polygon(J, K, L)
            0.752f, -0.552f, 0.0f,          1.0f, 0.0f, 1.0f,
            0.78f, -0.56f, 0.0f,            1.0f, 0.0f, 1.0f,
            0.78f, -0.66f, 0.0f,            1.0f, 0.0f, 1.0f,
    };
    MeshColor *letras = new MeshColor();
    letras->CreateMeshColor(vertices_letras,702);
    meshColorList.push_back(letras);
```

La segunda sección modificada fue el while. Dentro de este se colocó el siguiente código para dibujar en pantalla las letras. Aunque este no fue el código correspondiente al archivo final ya que inicialmente dibujé las letras usando la proyección ortogonal, por lo tanto después tuve que hacer algunas modificaciones a la escala para que se muestre correctamente junto con el ejercicio 2

```
//Para las letras hay que usar el segundo set de shaders con índice 1 en ShaderL
shaderList[1].useShader();
uniformModel = shaderList[1].getModelLocation();
uniformProjection = shaderList[1].getProjectLocation();

model = glm::mat4(1.0);
model = glm::translate(model, glm::vec3(0.0f, 0.0f, -0.4f));
model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES
glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
meshColorList[0]->RenderMeshColor();
```

I.II. **Capturas de pantalla de la ejecución.**



I.III. **Comentarios.**

Para que una letra sea de un color en específico, todos los vértices que la componen deben tener asignado el mismo color.

Una vez creados los triángulos con el color de sus vértices asignados, lo único que hace falta es modificar el código dentro del while. Para poder dibujar las letras hay que usar el segundo set de shaders, que es shadercolor.vert y shadercolor.frag. Este par de shaders son los que nos permiten asignar a cada vértice un color específico. Respecto al color de las letras, yo coloqué Azul(0.0f, 0.0f, 1.0f) para la letra R, Morado(0.5f, 0.0f, 0.5f) para la letra J y Rosa(1.0f, 0.0f, 1.0f) para la letra E.

II. Ejercicio 2. Generar el dibujo de la casa de la clase, pero en lugar de instanciar triangulos y cuadrados será instanciando piramides y cubos, para esto se requiere crear shaders diferentes de los colores: rojo, verde, azul, café y verde oscuro en lugar de usar el shader con el color clamp

II.I. **Bloques de código generados**. EL primer bloque se encuentra casi inmediatamente después de los headers. Con estas primeras líneas especificamos la dirección dentro de nuestro proyecto donde están guardados los archivos de los nuevos shaders que se crearon para los diferentes colores que se usarán para dibujar la casa.

```
29    static const char* vShaderrojo = "shaders/shaderrojo.vert";
30    static const char* vShadercafe = "shaders/shadercafe.vert";
31    static const char* vShaderazul = "shaders/shaderazul.vert";
32    static const char* vShaderverdeclaro = "shaders/shaderverdeclaro.vert";
33    static const char* vShaderverdeoscuro = "shaders/shaderverdeoscuro.vert";
```

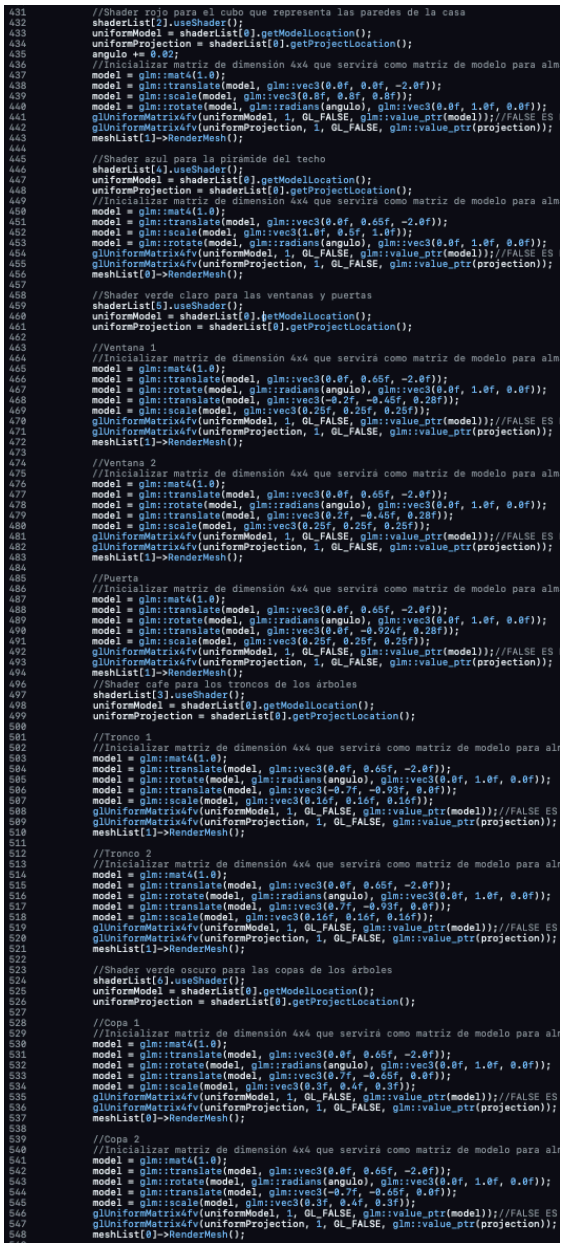
Lo siguiente que modifiqué fue el código de la función crea pirámide, ya que los vértices originales dibujaban una pirámide con base triangular y lo que necesitaba era una pirámide con base cuadrada para que quedara bien como el techo de la casa.

```
void CreaPiramide()
{
    unsigned int indices[] = {
        0, 1, 2,
        0, 2, 3,
        0, 1, 4,
        1, 2, 4,
        2, 3, 4,
        3, 0, 4

    };
    GLfloat vertices[] = {
        -0.5f, -0.5f,  0.5f,   // 0
         0.5f, -0.5f,  0.5f,   // 1
         0.5f, -0.5f, -0.5f,   // 2
        -0.5f, -0.5f, -0.5f,   // 3
         0.0f,  0.5f,  0.0f    // 4

    };
    Mesh* obj1 = new Mesh();
    obj1->CreateMesh(vertices, indices, 15, 18);
    meshList.push_back(obj1);
}
```

Posteriormente, dentro de la función CreateShaders() añadimos los nuevos shaders de colores a la shaderList para poder llamarlos y usarlos dentro del while.

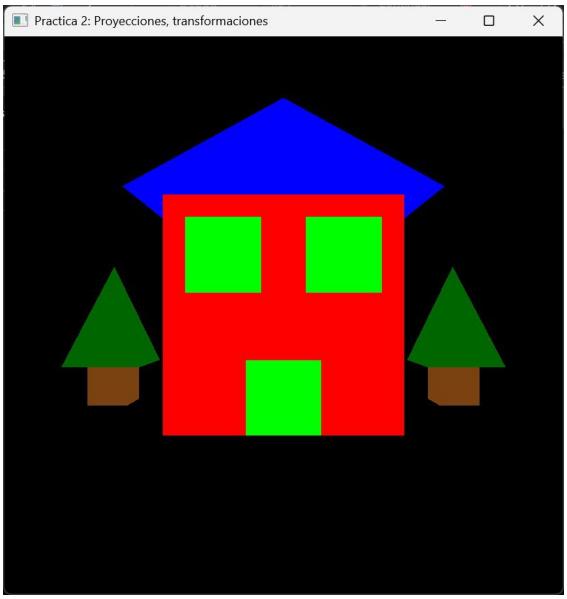
```
void CreateShaders()
{

    Shader *shader1 = new Shader(); //shader para usar indi
    shader1->CreateFromFiles(vShader, fShader);
    shaderList.push_back(*shader1);

    Shader *shader2 = new Shader();//shader para usar color
    shader2->CreateFromFiles(vShaderColor, fShaderColor);
    shaderList.push_back(*shader2);

    Shader* shader3 = new Shader(); //shader para usar indi
    shader3->CreateFromFiles(vShaderrojo, fShader);
    shaderList.push_back(*shader3);

    Shader* shader4 = new Shader(); //shader para usar indi
    shader4->CreateFromFiles(vShadercafe, fShader);
    shaderList.push_back(*shader4);

    Shader* shader5 = new Shader(); //shader para usar indi
    shader5->CreateFromFiles(vShaderazul, fShader);
    shaderList.push_back(*shader5);

    Shader* shader6 = new Shader(); //shader para usar indi
    shader6->CreateFromFiles(vShaderverdeclaro, fShader);
    shaderList.push_back(*shader6);

    Shader* shader7 = new Shader(); //shader para usar indi
    shader7->CreateFromFiles(vShaderverdeoscuro, fShader);
    shaderList.push_back(*shader7);

}
```

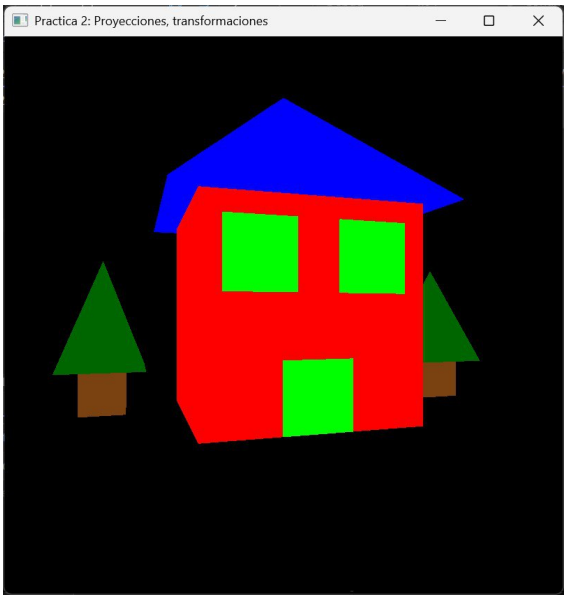
La tercera y última modificación que se le hizo al código, saltándome el cambio de proyección a perspective (ya que es sólo cambiar una línea por otra), es la del while. En este apartado ahora estaremos trabajando con los archivos de shader de colores que mandamos a traer. Las primeras tres líneas cada que se indica con un comentario el cambio de shader actúan en conjunto para cambiar el color con el que estaremos dibujando a partir de esa línea de código y hasta que se vuelva a cambiar de shader. Finalmente, para cada figura se inicializa una matriz de 4x4 que llamaremos “model”, en esta matriz se aplicarán las transformaciones geométricas (operaciones con matrices) que podremos después visualizar en las figuras que dibujemos.



```
        //Shader rojo para el cubo que representa las paredes de la casa
        shaderList[2].useShader();
        uniformModel = shaderList[0].getModelLocation();
        uniformProjection = shaderList[0].getProjectLocation();
        angulo += 0.02;
        //Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para alm
        model = glm::mat4(1.0);
        model = glm::translate(model, glm::vec3(0.0f, 0.0f, -2.0f));
        model = glm::scale(model, glm::vec3(0.8f, 0.8f, 0.8f));
        model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 1.0f, 0.0f));
        glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES
        glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
        meshList[1]->RenderMesh();

        //Shader azul para la piramide del techo
        shaderList[4].useShader();
        uniformModel = shaderList[0].getModelLocation();
        uniformProjection = shaderList[0].getProjectLocation();
        //Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para alm
        model = glm::mat4(1.0);
        model = glm::translate(model, glm::vec3(0.0f, 0.65f, -2.0f));
        model = glm::scale(model, glm::vec3(1.0f, 0.5f, 1.0f));
        model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 1.0f, 0.0f));
        glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES
        glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
        meshList[0]->RenderMesh();

        //Shader verde claro para las ventanas y puertas
        shaderList[5].useShader();
        uniformModel = shaderList[0].getModelLocation();
        uniformProjection = shaderList[0].getProjectLocation();

        //Ventana 1
        //Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para alm
        model = glm::mat4(1.0);
        model = glm::translate(model, glm::vec3(0.0f, 0.65f, -2.0f));
        model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 1.0f, 0.0f));
        model = glm::translate(model, glm::vec3(-0.2f, -0.45f, 0.28f));
        model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
        glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES
        glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
        meshList[1]->RenderMesh();

        //Ventana 2
        //Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para alm
        model = glm::mat4(1.0);
        model = glm::translate(model, glm::vec3(0.0f, 0.65f, -2.0f));
        model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 1.0f, 0.0f));
        model = glm::translate(model, glm::vec3(0.2f, -0.45f, 0.28f));
        model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
        glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES
        glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
        meshList[1]->RenderMesh();

        //Puerta
        //Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para alm
        model = glm::mat4(1.0);
        model = glm::translate(model, glm::vec3(0.0f, 0.65f, -2.0f));
        model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 1.0f, 0.0f));
        model = glm::translate(model, glm::vec3(0.0f, -0.92f, 0.28f));
        model = glm::scale(model, glm::vec3(0.25f, 0.25f, 0.25f));
        glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES
        glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
        meshList[1]->RenderMesh();
        //Shader cafe para los troncos de los árboles
        shaderList[3].useShader();
        uniformModel = shaderList[0].getModelLocation();
        uniformProjection = shaderList[0].getProjectLocation();

        //Tronco 1
        //Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para alm
        model = glm::mat4(1.0);
        model = glm::translate(model, glm::vec3(0.0f, 0.65f, -2.0f));
        model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 1.0f, 0.0f));
        model = glm::translate(model, glm::vec3(-0.7f, -0.93f, 0.0f));
        model = glm::scale(model, glm::vec3(0.1f, 0.16f, 0.16f));
        glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES
        glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
        meshList[1]->RenderMesh();

        //Tronco 2
        //Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para alm
        model = glm::mat4(1.0);
        model = glm::translate(model, glm::vec3(0.0f, 0.65f, -2.0f));
        model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 1.0f, 0.0f));
        model = glm::translate(model, glm::vec3(0.7f, -0.93f, 0.0f));
        model = glm::scale(model, glm::vec3(0.1f, 0.16f, 0.16f));
        glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES
        glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
        meshList[1]->RenderMesh();

        //Shader verde oscuro para las copas de los árboles
        shaderList[6].useShader();
        uniformModel = shaderList[0].getModelLocation();
        uniformProjection = shaderList[0].getProjectLocation();

        //Copa 1
        //Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para alm
        model = glm::mat4(1.0);
        model = glm::translate(model, glm::vec3(0.0f, 0.65f, -2.0f));
        model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 1.0f, 0.0f));
        model = glm::translate(model, glm::vec3(0.7f, -0.65f, 0.0f));
        model = glm::scale(model, glm::vec3(0.3f, 0.4f, 0.3f));
        glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES
        glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
        meshList[0]->RenderMesh();

        //Copa 2
        //Inicializar matriz de dimensión 4x4 que servirá como matriz de modelo para alm
        model = glm::mat4(1.0);
        model = glm::translate(model, glm::vec3(0.0f, 0.65f, -2.0f));
        model = glm::rotate(model, glm::radians(angulo), glm::vec3(0.0f, 1.0f, 0.0f));
        model = glm::translate(model, glm::vec3(-0.7f, -0.65f, 0.0f));
        model = glm::scale(model, glm::vec3(0.3f, 0.4f, 0.3f));
        glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));//FALSE ES
        glUniformMatrix4fv(uniformProjection, 1, GL_FALSE, glm::value_ptr(projection));
        meshList[0]->RenderMesh();
```

II.II.**Capturas de pantalla de la ejecución**.





II.III.**Comentarios**.

El cambio de color entre un dibujo y es gracias a la interacción entre el archivo de shader colorX.vert y el archivo shader.frag. El comportamiento de los shaders entre el ejercicio 1 y el ejercicio 2 radica en cómo el vertex maneja la información que le mandamos por medio del VBO.

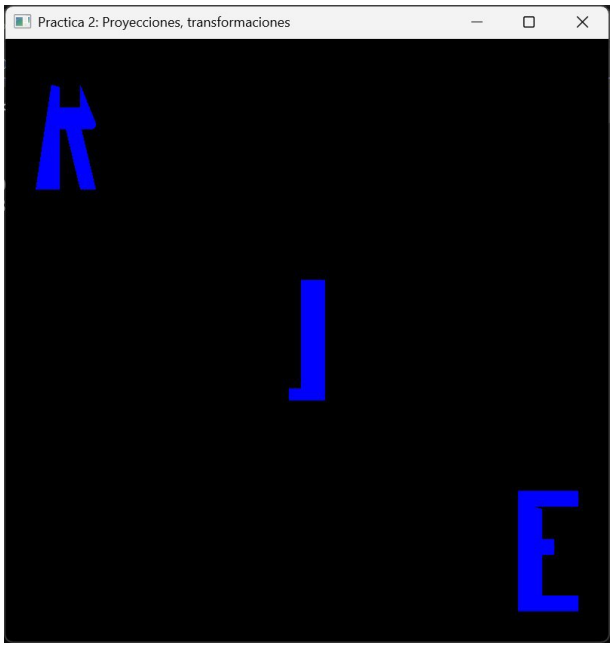
Dentro del archivo mesh.cpp indicamos la estructura con la que le vamos a mandar y almacenar los datos. En el caso del ejercicio 1, se le mandan al shader 6 flotantes. De estos 6 flotantes, los primeros tres se almacenarán en la primera posición de un arreglo vec3 y corresponden a la posición del pixel. Los siguientes tres flotantes se almacenarán en la segunda posición el arreglo y corresponden al color del pixel. Teniendo esta información, se le manda al shader.frag para que “pinte” el pixel.

Para el ejercicio 2 únicamente necesitamos mandarle 3 flotantes al shader, que son los de posición del pixel, ya que el color del que se pintará ya está definido en cada uno de los archivos que creamos.

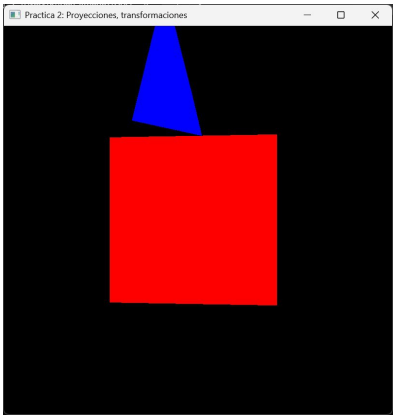
Cambiando a otro aspecto del ejercicio, la práctica no mencionaba que el dibujo debía de girar, sin embargo, me pareció que hacer que girara me permitiría comprender de mejor forma cómo funcionan las transformaciones geométricas y así fue, además, puede “apagarse” la rotación comentando el contador de ángulo. Detallaré más el proceso de cómo llegué al resultado final en el apartado de problemas.

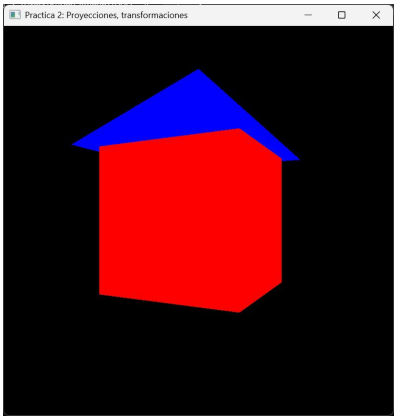
2.- Liste los problemas que tuvo a la hora de hacer estos ejercicios y si los resolvió explicar cómo fue, en caso de error adjuntar captura de pantalla

- El primer problema con el que me encontré fue en el ejercicio 1. Al momento de dibujar las letras, no recordaba que el eje Z (como lo tenemos configurado) sólo podemos trabajar con coordenadas negativas para indicar qué tan lejos de la cámara queremos que se dibuje nuestra figura. Yo estaba poniendo valores positivos en Z, por lo que no aparecían correctamente los dibujos, en la imagen que muestro puse valores negativos únicamente en los primeros triángulos de la letra R, por eso aparece incompleto.

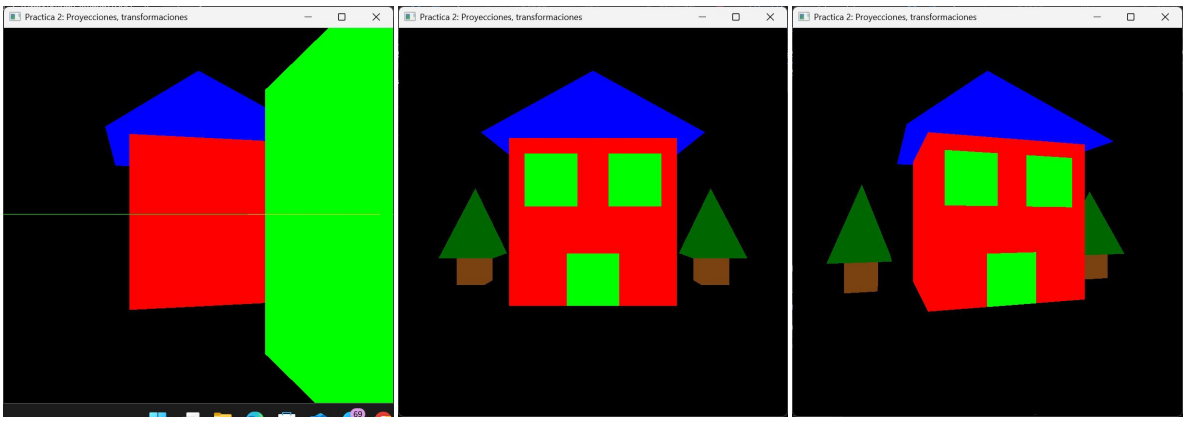


- El segundo problema que me encontré fue la pirámide. Estaba definida una pirámide con base triangular, lo que iba a hacer un poco complicado llegar a las transformaciones exactas para formar el techo con esta figura. Es por esto que hice la modificación que mostré en el apartado del ejercicio para tener directamente una pirámide de base cuadrada.



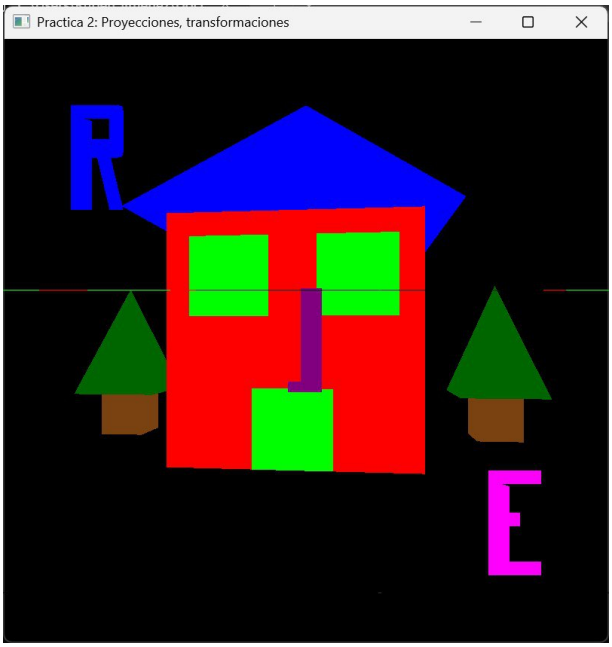


- El tercer y último problema que se me presentó fue al momento de implementar la rotación de la casa completa. En la clase nos mencionó el profesor que si hacíamos la rotación antes de la traslación la figura rotaría al rededor del origen. En un inicio entendí que esto era una especie de regla de OpenGL y que el origen estaba en el punto Z más cercano a la cámara, pero después de analizarlo un rato recordé que OpenGL trabaja siempre en el orden en el que se pongan las instrucciones y que el profesor nos explicó que las transformaciones no se trata más que de hacer operaciones con la matriz identidad. Entonces tuve que ir por pasos, al inicio quise asegurarme al rededor de qué eje giraban las figuras si primero hacía una traslación y luego rotación, entonces confirmé que giran al rededor del punto más cercano en Z, el cubo verde aparecía y desaparecía de la cámara. Pero si primero se hace la traslación y luego la rotación, el objeto gira sobre su propio eje Y.
  Mi conclusión fue que si todas las transformaciones se están aplicando sobre la misma matriz de 4x4, se van acumulando según el orden en el que se realicen. La forma de resolver este problema fue aplicar a todas las figuras que no quería que giraran sobre su propio eje la misma traslación y luego la misma rotación que a las paredes y el techo, que sí giran sobre su propio eje. Hacer estas dos primeras transformaciones es poner a la par a todas las figuras; todas las transformaciones que haga a partir de ese momento se agregarán a las ya existentes, en palabras simples esto lo interpreté como que estoy trasladando no solo la figura, sino también el origen del plano en el que voy a estar haciendo las siguientes transformaciones. Esto último puede notarse en el código para dibujar los cubos verdes de las ventanas y puerta ya que, como su nuevo origen está al centro de la casa, en esta ocasión fue necesario hacer una traslación positiva en el eje Z para que sobresalieran de la pared de enfrente.



3.- Conclusión:

a. Los ejercicios del reporte: La complejidad de los ejercicios de esta práctica considero que no fue demasiado alta en cuanto a implementar lo que se pedía en los ejercicios. Sin embargo, comprender realmente lo que estaba sucediendo con las transformaciones al momento de querer rotar la casa completa y cómo es que trabajan los shaders fue un poco más complicado, pero repasar los apuntes que hice en forma de comentarios el día de la clase conforme a la explicación que dio el profesor sobre el código fue de mucha ayuda, pues aunque no logré comprender por completo o lo comprendí de forma errónea en la clase las notas me permitieron replantearme lo que estaba haciendo y hacer las correcciones pertinentes.
   Como punto aparte, para el main final combiné ambos ejercicios para que las letras se muestren frente a la casa.



b. Comentarios generales: Aunque no es una práctica especialmente complicada, el contenido que abarca es realmente amplio y al momento de la clase es algo complicado captar todo, sin embargo, como ya mencioné, en una segunda visita al código y las notas de clase resulta más sencillo de comprender lo que estamos haciendo y por qué. Lo que complica un poco esta tarea es recordar en qué archivo del proyecto se realiza cada cosa.

c. Conclusión: La práctica me resultó bastante entretenida y me permitió afirmar muy bien los conocimientos básicos acerca del manejo de shaders, dibujo y aplicación de transformaciones sobre figuras en tres dimensiones

1. Bibliografía en formato APA