Apellidos: Fernández Miguel Del Corral	Nombre: Oscar
Apellidos: Del Cerro Morera	Nombre: Gonzalo

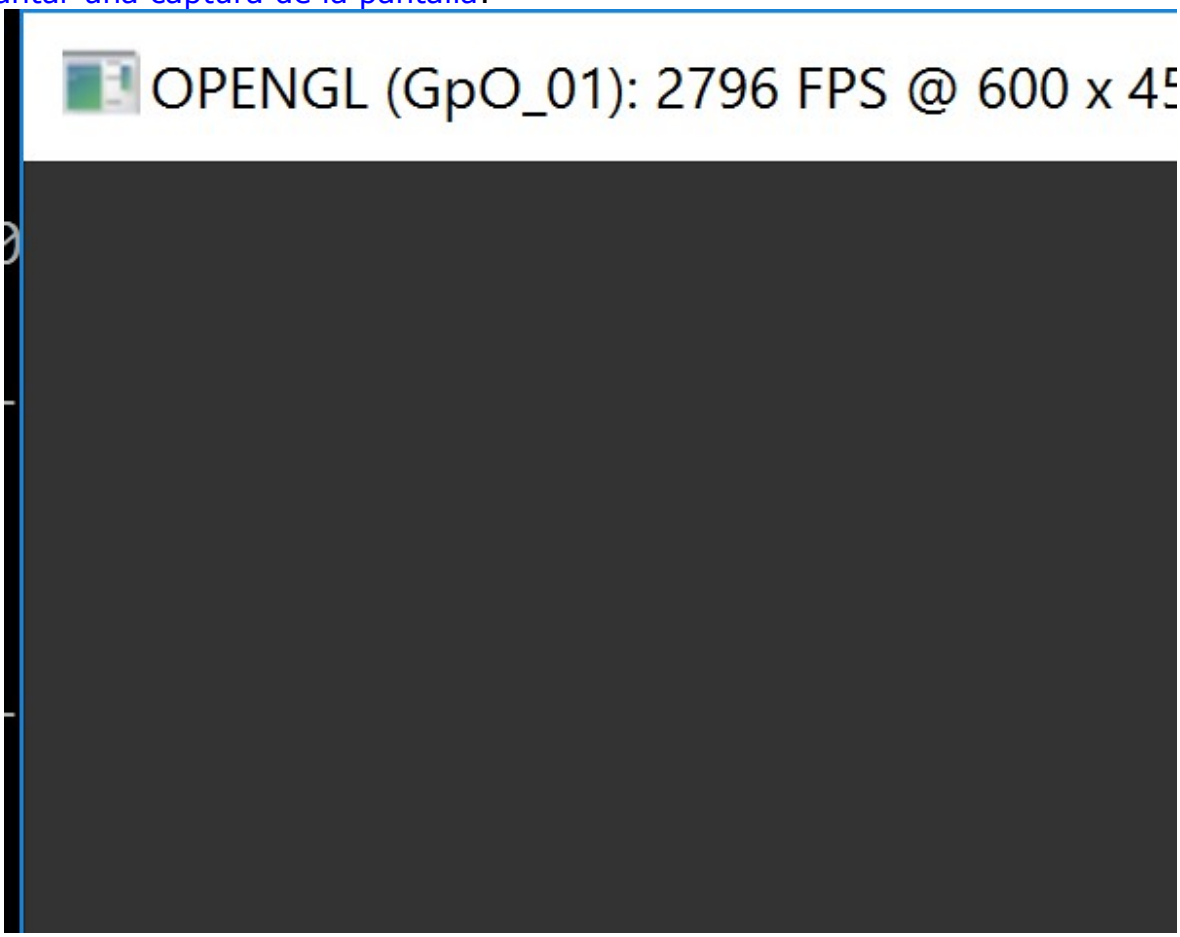
Responder a las preguntas y adjuntar los "trozos" de código pedidos. Entregad fichero de respuestas junto con los ficheros fuente pedidos.

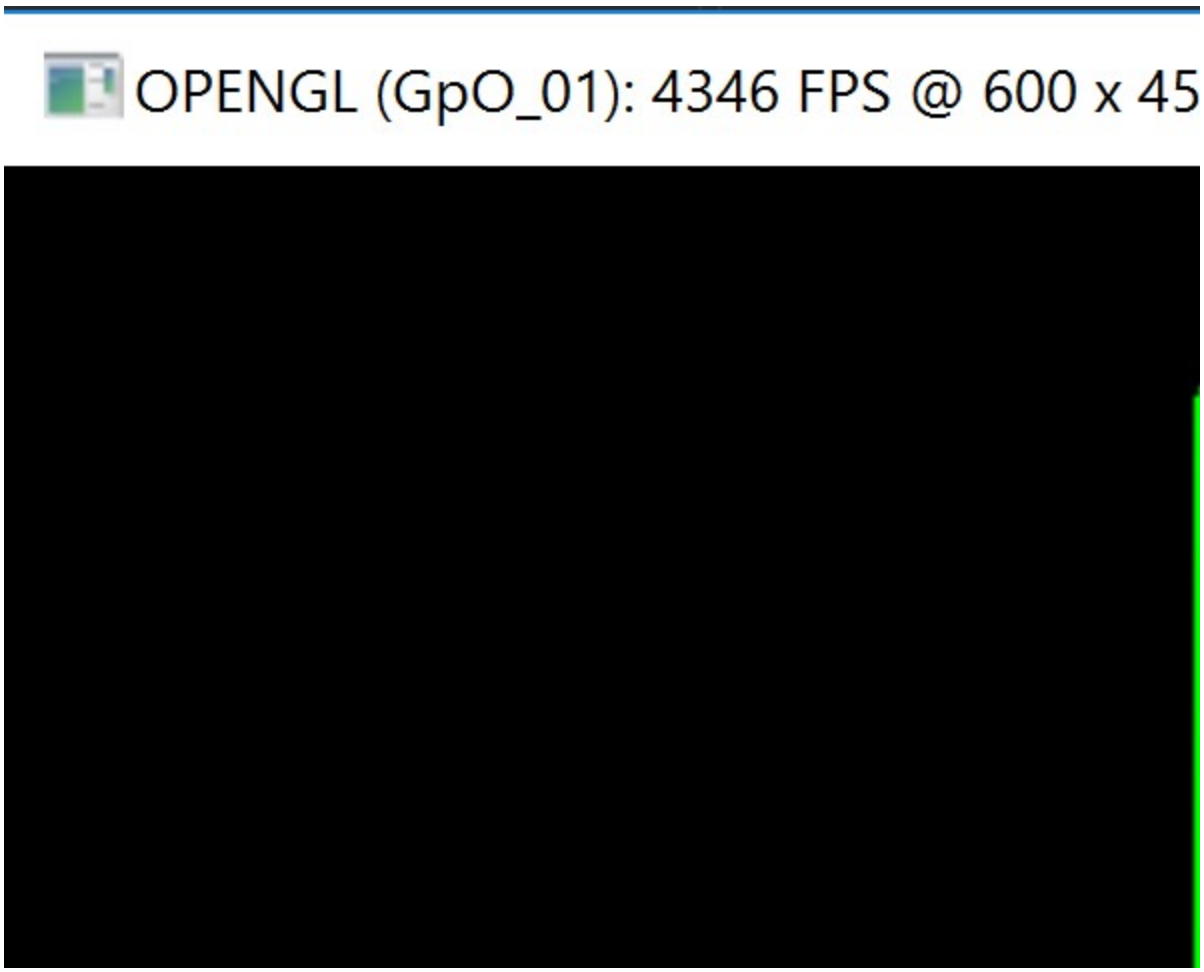
Objetivos:

- Conocer el entorno de VisualStudio y las librerías auxiliares: freeglut para manejo de ventanas/eventos y glm para crear/operar con matrices
- Entender la descripción de una escena, disposición ejes, etc. en OpenGL
- Cambiar la posición/orientación de la cámara con el teclado y observar los resultados.
- Combinar el uso de variables "uniform" con las rutinas de manejo de eventos para mover objetos por la escena o cambiar el punto de vista.
- Identificar que partes del código deben ejecutarse una vez (inicialización escena) y cuáles deben ejecutarse cada vez que regeneramos un frame.

LAB 1:

1)Adjuntar una captura de la pantalla.

1) 



Porque no se limpia el buffer antes de pintar la siguiente iteración.

2) Indicar la relación de aspecto y campo de visión vertical usado en dicha matriz de proyección.
4:3. Y 35.0f.

3) ¿Qué le pasa al triángulo? Adjuntar código añadido (1 línea)
Mantiene su relación de aspecto.

```
aspect = (float) CurrentWidth / CurrentHeight;
```

4) Código añadido

```
fov = (float) CurrentHeight * 35 / 450;
```

5) Adjuntar código de la función mouse().

```
void mouse(int but,int state, int x, int y)
{

  printf("Estado %d\n",state);
  if (but  == 0 || but == 3)
  {
         CurrentHeight = CurrentHeight + 10;
  }
  else
         CurrentHeight = CurrentHeight - 10;

       glutReshapeWindow(CurrentWidth, CurrentHeight);

}
```

LAB 2:

1) NTBD

2) ¿Qué variable debéis cambiar? ¿En qué función debéis hacerlo? Adjuntar el código añadido a vuestro programa (2/3 líneas).

```
void key_special(int key, int x, int y)
{
  switch (key)
       {
         case GLUT_KEY_F1:  // Teclas de Funcion

           break;
         case GLUT_KEY_UP:  //Teclas cursor;
                printf("Posicion X ini %f\n", pos_obs.r);
                pos_obs += vec3(0.1f, 0.0f, 0.0f);
                printf("Posicion X after %f", pos_obs.r);
                break;
         case GLUT_KEY_DOWN:
                printf("Posicion X %f \n", pos_obs.r);
                pos_obs -= vec3(0.1f, 0.0f, 0.0f);
                printf("Posicion X after %f", pos_obs.r);
         break;
         case GLUT_KEY_LEFT:
            break;
      case GLUT_KEY_RIGHT:
             break;
       }
}
```

3) Poneos a X=0.7 y luego a X=0.3 del origen. ¿Qué diferencia apreciáis? ¿Por qué?
Se hace más grande o más pequeño. Cambia el punto del observador.
Ahora alejaos. Observar como el objeto se va reduciendo.
¿Notáis algo extraño si os alejáis mucho?
El triángulo desaparece porque nos encontramos más lejos o más cerca del plano lejano o el cercano.
¿A qué distancia del origen estabais? JUSTIFICAR.
Esto puede ser resuelto modificando las variables: `Znear=0.5, Zfar=20`

4) Adjuntar código añadido (basta con modificar una línea).

```
T = translate(3 * sin(tt), 3 * cos(tt),0.0f );  M = T;
```

¿Cambia el tamaño del objeto?

Sì.

¿Por qué si no se cambia su escala? Porque el objeto se acerca / aleja desde el punto del observador.

Acercaos con los cursores. ¿Qué pasa si os ponéis en X=3.5?

El triángulo pasa por delante del observador y se ve correctamente. ¿Y en X=2.5?

El triángulo pasa por detrás de donde estaría la cámara.

¿Qué se observa si os colocáis en X=20?

Solo se ve cuando está más cerca del X=20.

LAB 3:

Adjuntad código de vuestra función de render (solo la parte de crear las matrices M y dibujar ambos objetos). Compilar y ver el resultado.

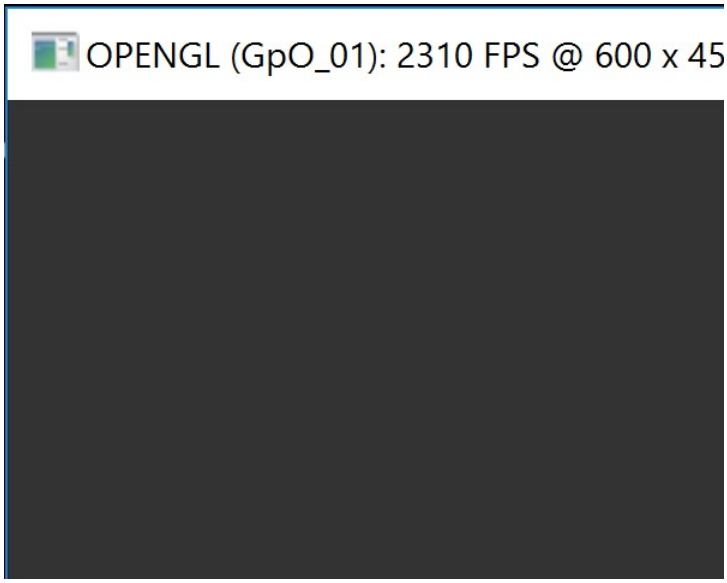
```
mat4 P,V,M,T,R,S;
//T=translate(0.0f,0.0f,3*sin(tt));  M=T;
T = translate(3 * sin(tt), 3 * cos(tt),0.0f );  M = T;
P = perspective(fov, aspect, 0.5f, 20.0f);  //40º FOV,  4:3 ,  Znear=0.5, Zfar=20
V = lookAt(pos_obs, target, up  );  // Pos camara, Lookat, head up

mat4 Q=P*V*M;
transfer_mat4("MVP",Q);

glBindVertexArray(triangulo.VAO);              // Activamos VAO asociado al objeto
glDrawArrays(GL_TRIANGLES, 0, triangulo.Nv);   // Orden de dibujar (Nv vertices)
glBindVertexArray(0);
T=translate(0.0f,0.0f,3*cos(tt));  M=T;
M = T;
Q = P*V*M;
transfer_mat4("MVP", Q);
glBindVertexArray(triangulo.VAO);              // Activamos VAO asociado al objeto
glDrawArrays(GL_TRIANGLES, 0, triangulo.Nv);   // Orden de dibujar (Nv vertices)
glBindVertexArray(0);
```

¿Apreciáis algo incorrecto en la evolución de la escena mostrada? Si, el Ultimo triangulo que se dibuja prevalece sobre el otro, es decir, El segundo triangulo esta siepre por encima aunque el primero esta mas cerca del observador.

En la entrega, adjuntar a este fichero doc el código de este programa (con nombre lab123.cpp, incorporando las modificaciones de los 3 ejercicios).

LAB 4:

Adjuntar una captura de pantalla y el código usado en render_scene().


OPENGL (GpO_01): 2310 FPS @ 600 x 45

```
        T = translate(3 * sin(tt), 3 * cos(tt), 0.0f);  M = T;
        R = glm::rotate(80 * tt, vec3(1.0f, 0.0f, 0.0f));
        S = glm::scale(vec3(1.0f, 0.5f, 1.5f));

        P = perspective(fov, aspect, 0.5f, 20.0f);  //40º FOV,  4:3 ,  Znear=0.5, Zfar=20
        V = lookAt(pos_obs, target, up);  // Pos camara, Lookat, head up
        M = T*R*S;
        mat4 Q = P*V*M;
        transfer_mat4("MVP", Q);

        glBindVertexArray(triangulo.VAO);              // Activamos VAO asociado al objeto
        glDrawArrays(GL_TRIANGLES, 0, triangulo.Nv);   // Orden de dibujar (Nv vertices)
        glBindVertexArray(0);
        //T = translate(0.0f, 0.0f, 3 * cos(tt));  M = T;
        M = S*R*T;
        Q = P*V*M;
        transfer_mat4("MVP", Q);
        glBindVertexArray(triangulo.VAO);              // Activamos VAO asociado al objeto
        glDrawArrays(GL_TRIANGLES, 0, triangulo.Nv);   // Orden de dibujar (Nv vertices)
        glBindVertexArray(0);

        M = R*T*S;
        Q = P*V*M;
        transfer_mat4("MVP", Q);
        glBindVertexArray(triangulo.VAO);              // Activamos VAO asociado al objeto
        glDrawArrays(GL_TRIANGLES, 0, triangulo.Nv);   // Orden de dibujar (Nv vertices)
        glBindVertexArray(0);
```

LAB 5:

Adjuntar una captura de pantalla con el resultado en este doc y el código de vuestro programa (lab5.cpp).

```
                                                        ///////// Actializaciuon
matrices M, V, P  ////////

	mat4 P, V, M, T, R, S;
	//T=translate(0.0f,0.0f,3*sin(tt));  M=T;
	T = translate( 0.0f,2 * sin(tt * 3.14f * 2 / 5), 2 * cos(tt * 3.14f * 2 / 5));  M =
T;
	R = glm::rotate(50 * tt, vec3(1.0f, 0.0f, 0.0f));
	//R = glm::rotate(80 * tt, vec3(1.0f, 0.0f, 0.0f));
	//S = glm::scale(vec3(1.0f, 0.5f, 1.5f));

	P = perspective(fov, aspect, 0.5f, 20.0f);  //40º FOV,  4:3 ,  Znear=0.5, Zfar=20
	V = lookAt(pos_obs, target, up);  // Pos camara, Lookat, head up
	M = T*R;
	mat4 Q = P*V*M;
	transfer_mat4("MVP", Q);

	glBindVertexArray(triangulo.VAO);              // Activamos VAO asociado al objeto
	glDrawArrays(GL_TRIANGLES, 0, triangulo.Nv);   // Orden de dibujar (Nv vertices)
	glBindVertexArray(0);
	float fase = (float)(tt * 3.14  * 2 /5  ) - (2 * 3.14) / 3;
	T = translate(0.0f, 2 * sin( fase ), 2 * cos(fase));  M = T;
	R = glm::rotate(50 * tt, vec3(0.0f, 1.0f, 0.0f));
	M = T*R;
	Q = P*V*M;
	transfer_mat4("MVP", Q);
	glBindVertexArray(triangulo.VAO);              // Activamos VAO asociado al objeto
	glDrawArrays(GL_TRIANGLES, 0, triangulo.Nv);   // Orden de dibujar (Nv vertices)
	glBindVertexArray(0);
	float faseTrianguloTres = (float)(tt * 3.14 * 2 / 5) - (4 * 3.14) / 3;
	T = translate(0.0f, 2 * sin(faseTrianguloTres), 2 * cos(faseTrianguloTres));  M = T;
	R = glm::rotate(50 * tt, vec3(0.0f, 0.0f, 1.0f));
	M = T*R;
	Q = P*V*M;
	transfer_mat4("MVP", Q);
	glBindVertexArray(triangulo.VAO);              // Activamos VAO asociado al objeto
	glDrawArrays(GL_TRIANGLES, 0, triangulo.Nv);   // Orden de dibujar (Nv vertices)
	glBindVertexArray(0);
```

d

LAB 6:

Volver a compilar el programa. ¿Aparecen errores? ¿Por qué?
Ejecutarlo. ¿Funciona? Observar los avisos que se vuelcan a la consola.

Adjuntar código del "fragment shader" modificado y una captura de la imagen resultante donde se observe el efecto del cambio de código.

LAB 7:

Adjuntad código de vuestra nueva función crear_triangulo().

Adjuntad nuevo código para crear vertex_data.

¿Cuál sería ahora vuestro vector de índices?. Adjuntadlo.
¿Cuántos vértices e índices tiene nuestro objeto?

Adjuntad código completo de vuestro programa incorporando el uso de índices y mostrando un cuadrado en vez de un triángulo (llamadlo lab7.cpp)