**UNIVERSIDAD DE LAS FUERZAS ARMADAS**

**ESPE**

**DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN**

**CARRERA DE TECNOLOGÍAS DE LA INFORMACIÓN**

**ESTRUCTURA DE DATOS - NRC: 14128**

**TEMA: PROYECTO DEL TERCER PARCIAL – CUBO RUBIK**

**GRUPO N.-**

**INTEGRANTES:**

[JEAN PIERRE PEÑARANDA GUEVARA](https://micampus.espe.edu.ec/user/view.php?id=690&course=5534)

[FRANCO ANTONIO PILLAJO PERALTA](https://micampus.espe.edu.ec/user/view.php?id=1628&course=5534)

[KEVIN PAÚL GUALOTUÑA CRISANTO](https://micampus.espe.edu.ec/user/view.php?id=4088&course=5534)

[CESAR OCTAVIO ARICO MURILLO](https://micampus.espe.edu.ec/user/view.php?id=4220&course=5534)

**DOCENTE:**

ING. MAYRA ALVAREZ

**08 - 03 - 2024**

**Índice de contenidos**

[1. Introducción 3](#_Toc160746592)

[2. Desarrollo 4](#_Toc160746593)

[2.1 Fases del proyecto: 4](#_Toc160746594)

[2.2.1 Preparación del entorno de trabajo 4](#_Toc160746595)

[2.2.2 Creación y modelado del cubo: 5](#_Toc160746596)

[2.2.3 Actualización a un cubo de 3x3x3: 9](#_Toc160746597)

[2.2.4 Adición de lineas negras: 10](#_Toc160746598)

[2.2.5 Organización del proyecto 12](#_Toc160746599)

[2.2.6 Organización del código: 15](#_Toc160746600)

[2.2.8. Corrección de errores y nuevas funciones: 15](#_Toc160746601)

[2.2 Algoritmos empleados: 16](#_Toc160746602)

[3. Conclusiones 25](#_Toc160746603)

[4. Recomendaciones 25](#_Toc160746604)

[5. Referencias Bibliograficas 27](#_Toc160746605)

**Tabla de figuras**

[Figura 2.2.1.1. Archivos glut 6](#_Toc160772015)

[Figura 2.2.1.2. Librerías glut 6](#_Toc160772016)

[Figura 2.2.1.*3.* Archivo dll 7](#_Toc160772017)

[Figura 2.2.1.4. Archivo glut.cbp 7](#_Toc160772018)

[Figura 2.2.1.5. Archivos Script 7](#_Toc160772019)

[Figura 2.2.2.1*.* Librerías 8](#_Toc160772020)

[Figura 2.2.2.2. OpenGL y GLUT 8](#_Toc160772021)

[Figura 2.2.2.3. Estructura CuboRubik 9](#_Toc160772022)

[Figura 2.2.2.4*.* Dibujo del cubo 10](#_Toc160772023)

[Figura 2.2.2.5*.* Manejo Cámara 11](#_Toc160772024)

[Figura 2.2.2.6. Perspectiva 11](#_Toc160772025)

[Figura 2.2.2.7. Creación de cubo de colores 12](#_Toc160772026)

[Figura 2.2.3.1. Actualización Display 13](#_Toc160772027)

[Figura 2.2.3.2. Cubo Rubik 3x3x3 14](#_Toc160772028)

[Figura 2.2.3.3. Cubo Rubik estilizado 14](#_Toc160772029)

[Figura 2.2.4.1. Líneas Negras Display 15](#_Toc160772030)

[Figura 2.2.4.2. Líneas negras en caras para visualizar las separaciones del Rubik 16](#_Toc160772031)

[Figura 2.2.5.1. Estructura Rubik en Rubik.h 16](#_Toc160772032)

[Figura 2.2.5.2. Cubito.h 17](#_Toc160772033)

[Figura 2.2.5.3. Rotaciones 18](#_Toc160772034)

[Figura 2.2.5.4. Cubo Rubik mejora de movilidad 19](#_Toc160772035)

[Figura 2.2.6.1. Documentación 19](#_Toc160772036)

[Figura 2.2.7.1. Cubo rubik colores aleatorios por teclado y giros correctos 20](#_Toc160772037)

[Figura 2.2.7.2. Cubo rubik colores iniciales mediante teclado 20](#_Toc160772038)

[Figura 2.2.1. Función de asignar colores al cubo 21](#_Toc160772039)

[Figura 2.2.3. Mostrar 23](#_Toc160772040)

[Figura 2.2.4. Redimensión 24](#_Toc160772041)

[Figura 2.2.5. menuCallback 24](#_Toc160772042)

[Figura 2.2.6. Mostrar Timer 25](#_Toc160772043)

[Figura 2.2.7. void Raton 25](#_Toc160772044)

[Figura 2.2.8. Movimiento Ratón 26](#_Toc160772045)

[Figura 2.2.9. void teclado 27](#_Toc160772046)

[Figura 2.2.10. void incializarCubo 27](#_Toc160772047)

[Figura 2.2.11. Main 28](#_Toc160772048)

[Figura 2.2.12. RotarCapaDerecha 29](#_Toc160772049)

[Figura 2.2.13. RotarCaraFrontal 30](#_Toc160772050)

# 1. Introducción

El proyecto del Cubo de Rubik en OpenGL es un ambicioso esfuerzo para desarrollar una aplicación interactiva que emule la experiencia física del famoso rompecabezas tridimensional inventado por Erno Rubik en 1974. Este cubo, con sus seis caras de colores, ha desafiado a generaciones con sus combinaciones aparentemente infinitas y sus soluciones ingeniosas (*Historia Del Cubo de Rubik | Biblioteca ULPGC*, 2024).

El propósito principal de esta iniciativa es proporcionar a los entusiastas del Cubo de Rubik una plataforma virtual donde puedan explorar, aprender y mejorar sus habilidades para resolver este intrigante puzzle. Además de ser una herramienta de entretenimiento, aspiramos a que sea un recurso educativo valioso que fomente la comprensión de los principios detrás de la resolución del cubo y estimule el desarrollo del pensamiento lógico y estratégico.

El alcance del proyecto abarca la creación de una aplicación en OpenGL que permita a los usuarios interactuar con un Cubo de Rubik virtual, rotando sus caras y aplicando las reglas establecidas del juego. Con una interfaz gráfica intuitiva y controles accesibles, buscamos crear una experiencia inmersiva que capture la esencia del Cubo de Rubik físico mientras aprovecha las capacidades de la tecnología digital.

El equipo detrás de este proyecto está compuesto por un grupo de estudiantes de la Carrera de Tecnologías de la Información de la Universidad de las Fuerzas Armadas - ESPE, quienes asumen el desafío de fusionar la creatividad, el conocimiento técnico y la pasión por los rompecabezas para llevar este proyecto a la realidad.

Con el Cubo de Rubik en OpenGL, no solo buscamos ofrecer una aplicación funcional, sino también celebrar la fascinación humana por los desafíos intelectuales y la resolución de problemas. Este proyecto representa un tributo a la mente humana y su capacidad para abordar y superar obstáculos aparentemente insuperables.

# 2. Desarrollo

Se decidió dividir el proyecto en varias etapas para facilitar su desarrollo:

* Instalación de OpenGL, GLUT y configuración del entorno de desarrollo C++.
* Implementación de la estructura de datos del cubo de Rubik.
* Desarrollo de la representación gráfica del cubo.
* Implementación de la interacción del usuario para rotar la vista.

## 2.1 Fases del proyecto:

### 2.2.1 Preparación del entorno de trabajo

Inicialmente preparamos el Codeblocks para que pueda funcionar con las librerías que permitan usar OPENGL. Copiamos los archivos en la carpeta de instalación del programa. En “C:\ProgramFiles\CodeBlocks\MinGW\x86\_64-w64-mingw32\include\GL”.



Figura 2.2.1.1. Archivos glut

Luego copiamos los archivos en “C:\Program Files\CodeBlocks\MinGW\x86\_64-w64-mingw32\lib”.



Figura 2.2.1.2. Librerías glut

Copiar el archivo dll en “C:\Windows\System32 ”



Figura 2.2.1.*3.* Archivo dll

Copiamos y reemplazamos en “C:\Program Files\CodeBlocks\share\CodeBlocks\templates”.



Figura 2.2.1.4. Archivo glut.cbp

Y por último copiamos y reemplazamos está archivos en “C:\Program Files\CodeBlocks\share\CodeBlocks\templates\wizard\glut”



Figura 2.2.1.5. Archivos Script

Una vez hecho lo anterior procedimos al desarrollo del cubo de Rubik empezando en la creación y el moldeo de un cubo de colores inicialmente.

### Creación y modelado del cubo:

**Llamado a librerías y variables globales definidas**: Se llama a las librerías necesarias para el proyecto y se inicializan variables globales que serán usadas dentro de funciones para realizar acciones específicas.

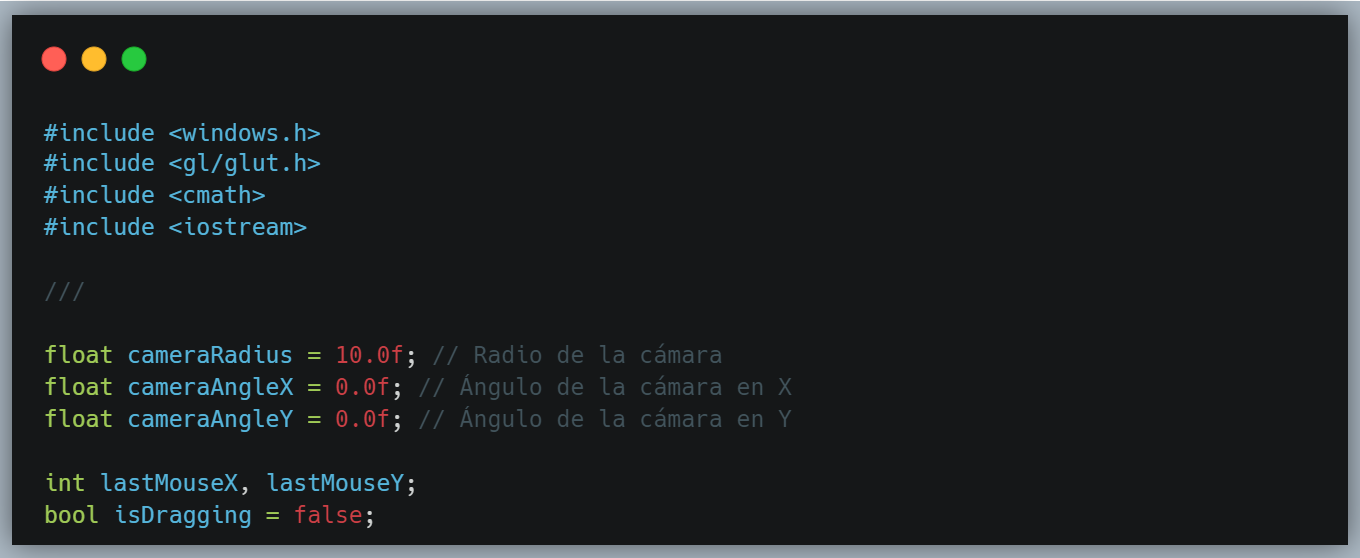


Figura 2.2.2.1*.* Librerías

**Inicialización de OpenGL y GLUT:** Se inicia el entorno de OpenGL y GLUT en la función main(). Se establecen el modo de visualización, el tamaño de la ventana y se crea la ventana principal

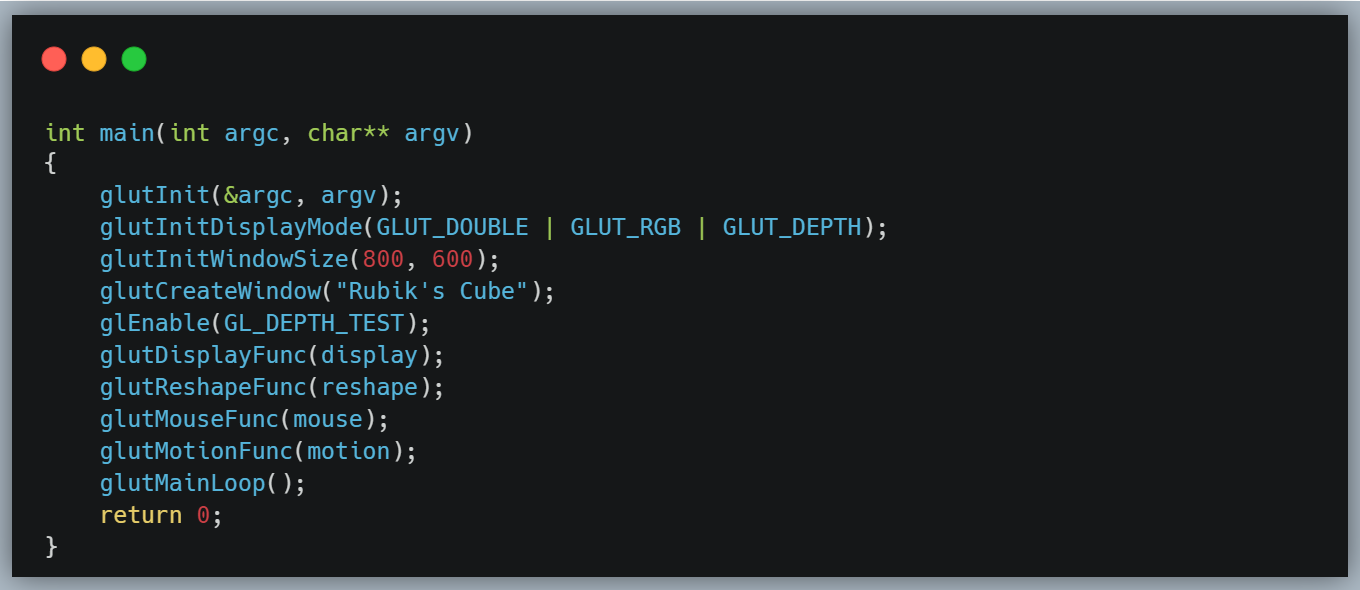


Figura 2.2.2.2. OpenGL y GLUT

**Definición de la Estructura del Cubo de Rubik:** Se define la estructura *RubiksCube* para representar el cubo de Rubik y sus cubos más pequeños. Esta estructura contiene un arreglo tridimensional de cubos más pequeños, cada uno representado por su color en cada una de sus seis caras.

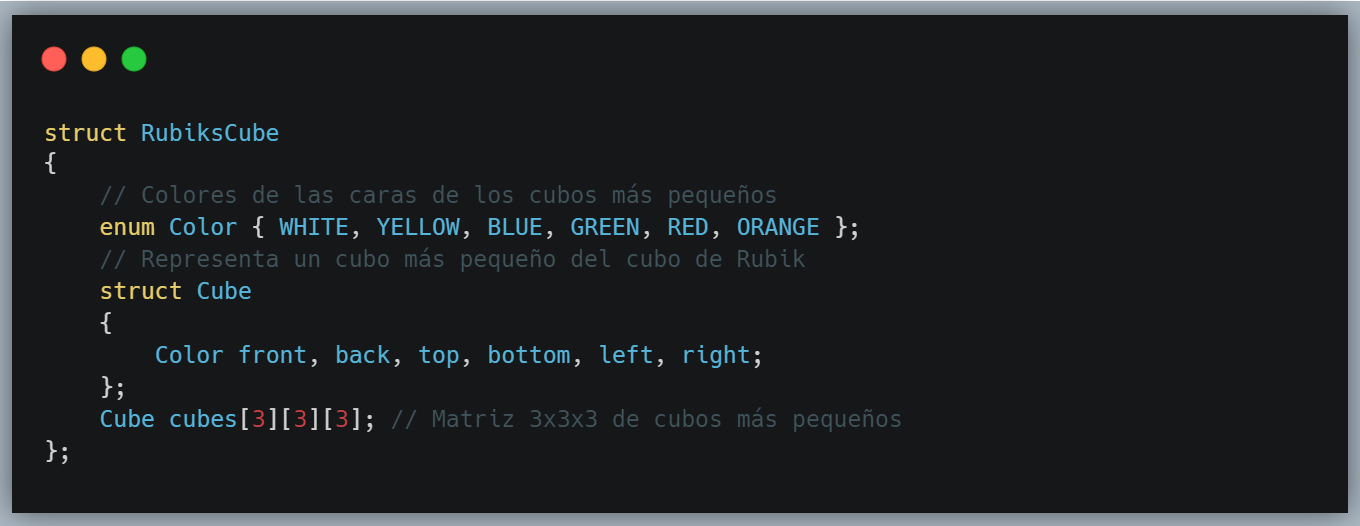


Figura 2.2.2.3. Estructura CuboRubik

**Dibujo del Cubo de Rubik:** Se utiliza la función display() para dibujar el cubo de Rubik. Dentro de esta función, se establecen las transformaciones de la cámara y se dibujan los cubos más pequeños con colores diferentes para representar las caras del cubo.

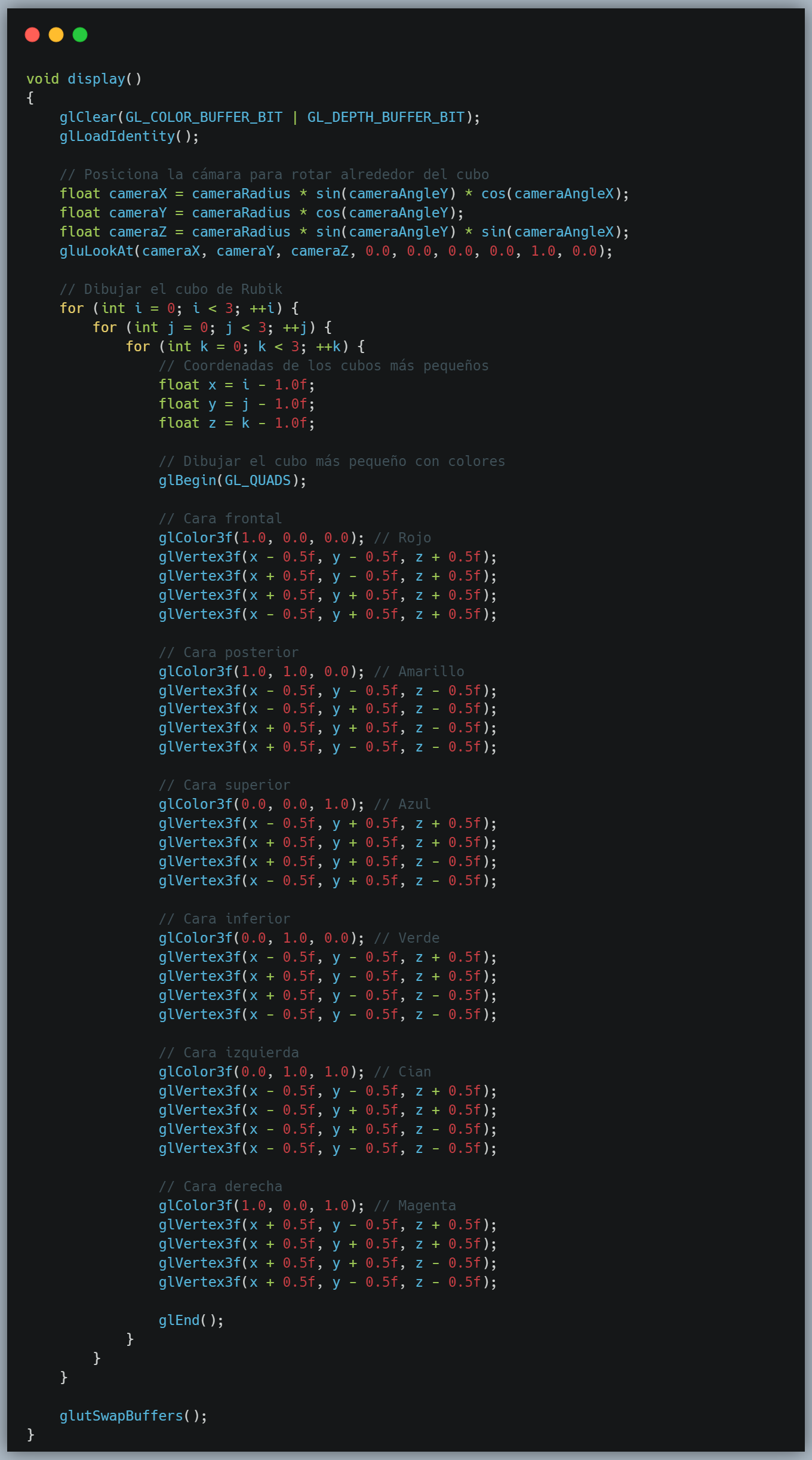


Figura 2.2.2.4*.* Dibujo del cubo

**Manejo de la Cámara**: Se implementa la funcionalidad de rotación de la cámara utilizando el mouse. Las funciones mouse() y motion() se encargan de detectar los eventos del mouse y calcular los cambios en los ángulos de la cámara en función del movimiento del mouse.



Figura 2.2.2.5*.* Manejo Cámara

**Configuración de la Perspectiva**: La función reshape() se utiliza para configurar la proyección perspectiva, asegurándose de que los objetos se rendericen correctamente en la ventana en función del tamaño y la relación de aspecto.

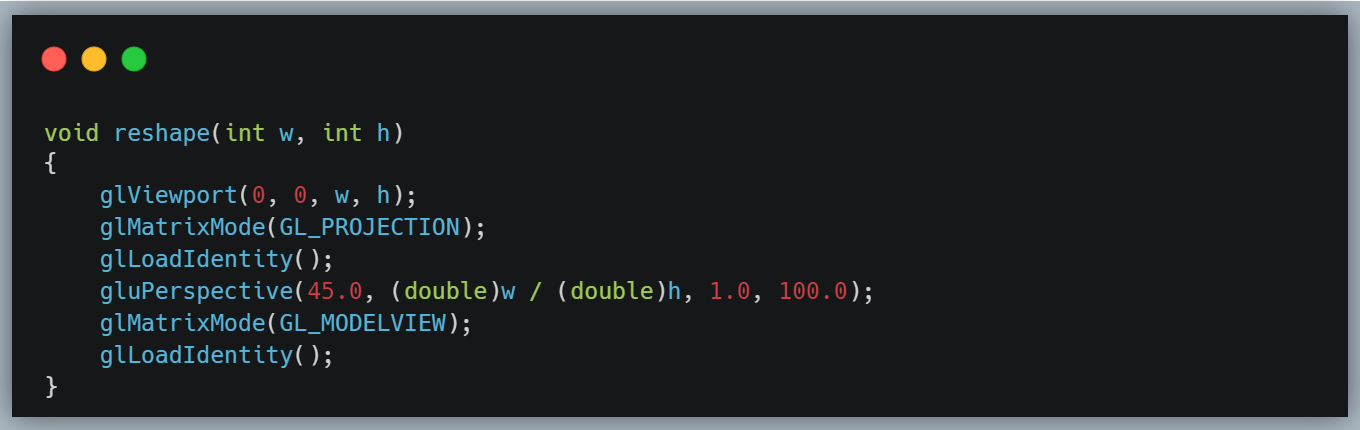


Figura 2.2.2.6. Perspectiva

**Ejecución del programa**: Dado el código, procedemos a ejecutar para verificar si existen errores, en nuestro caso la ejecución de dio con éxito mostrándonos un cubo tridimensional.

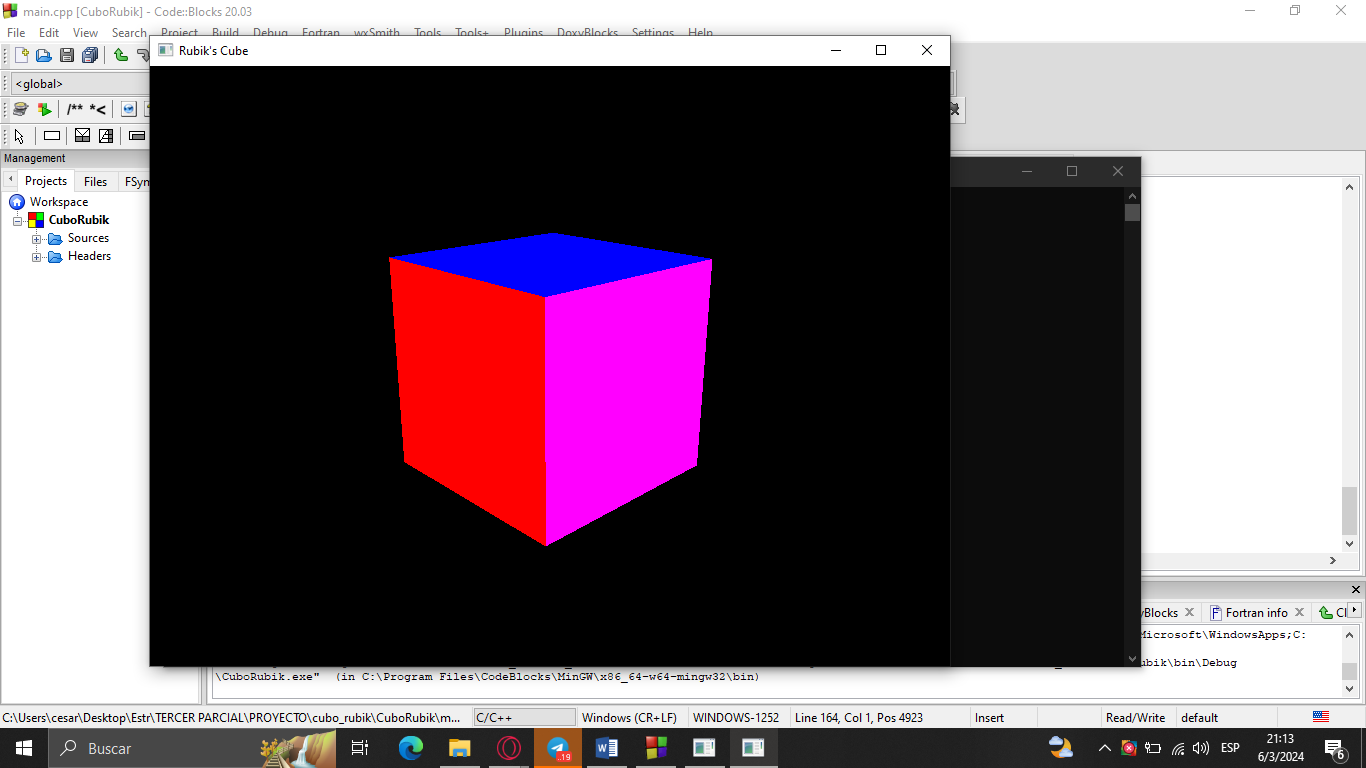


Figura 2.2.2.7. Creación de cubo de colores

### 2.2.3 Actualización a un cubo de 3x3x3:

**Actualización de Display():** Se modificó el modelo del cubo para tener 3 capas en cada eje.

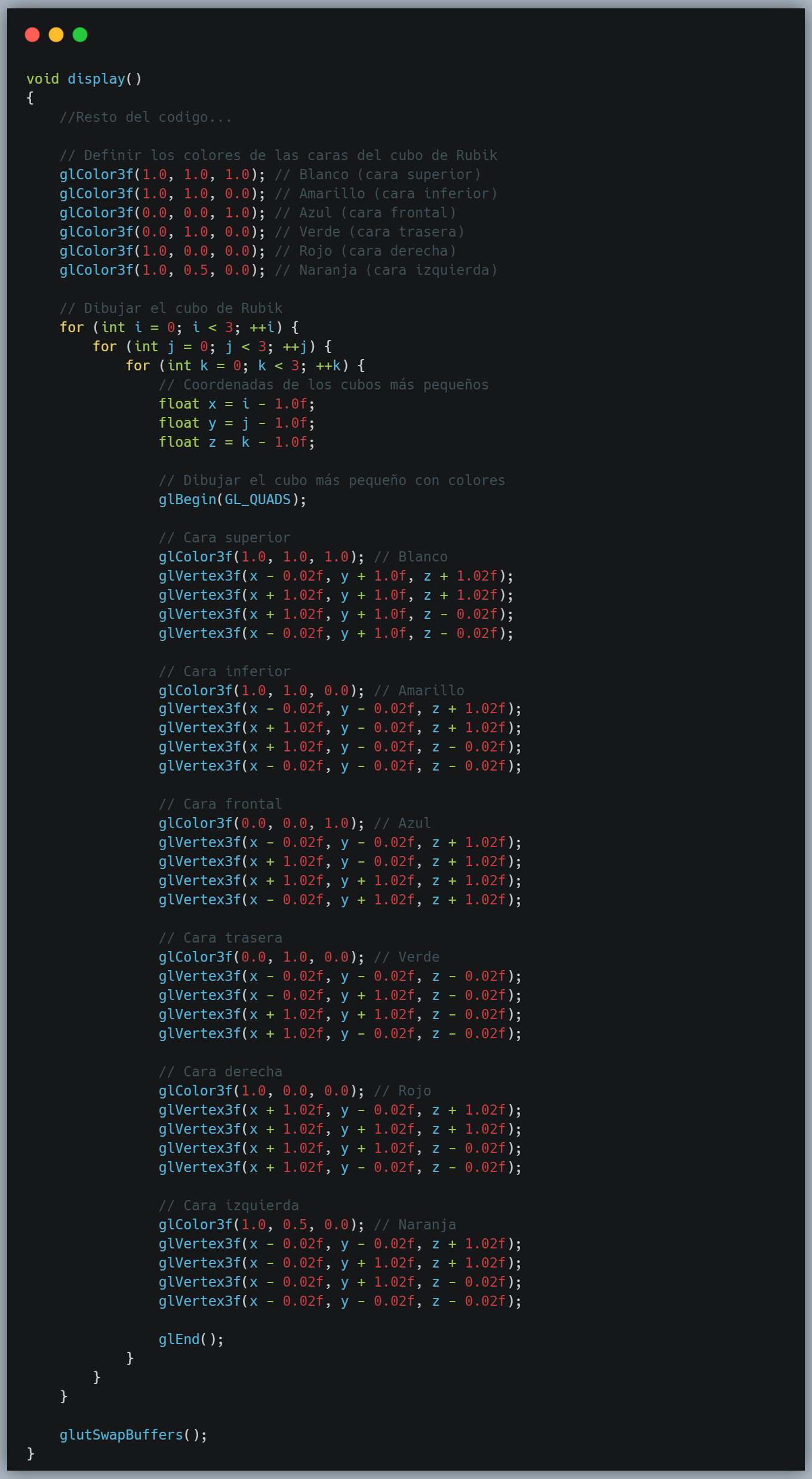


Figura 2.2.3.1. Actualización Display

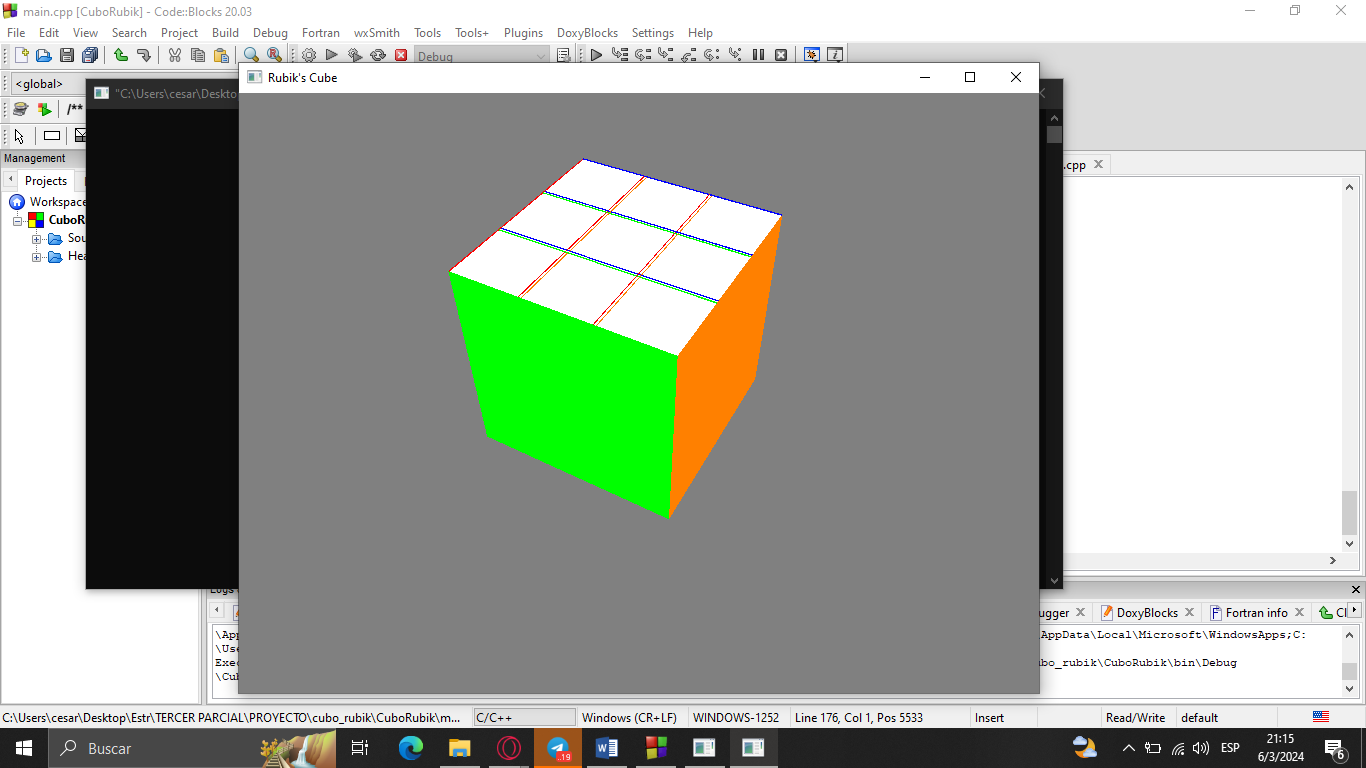


Figura 2.2.3.2. Cubo Rubik 3x3x3

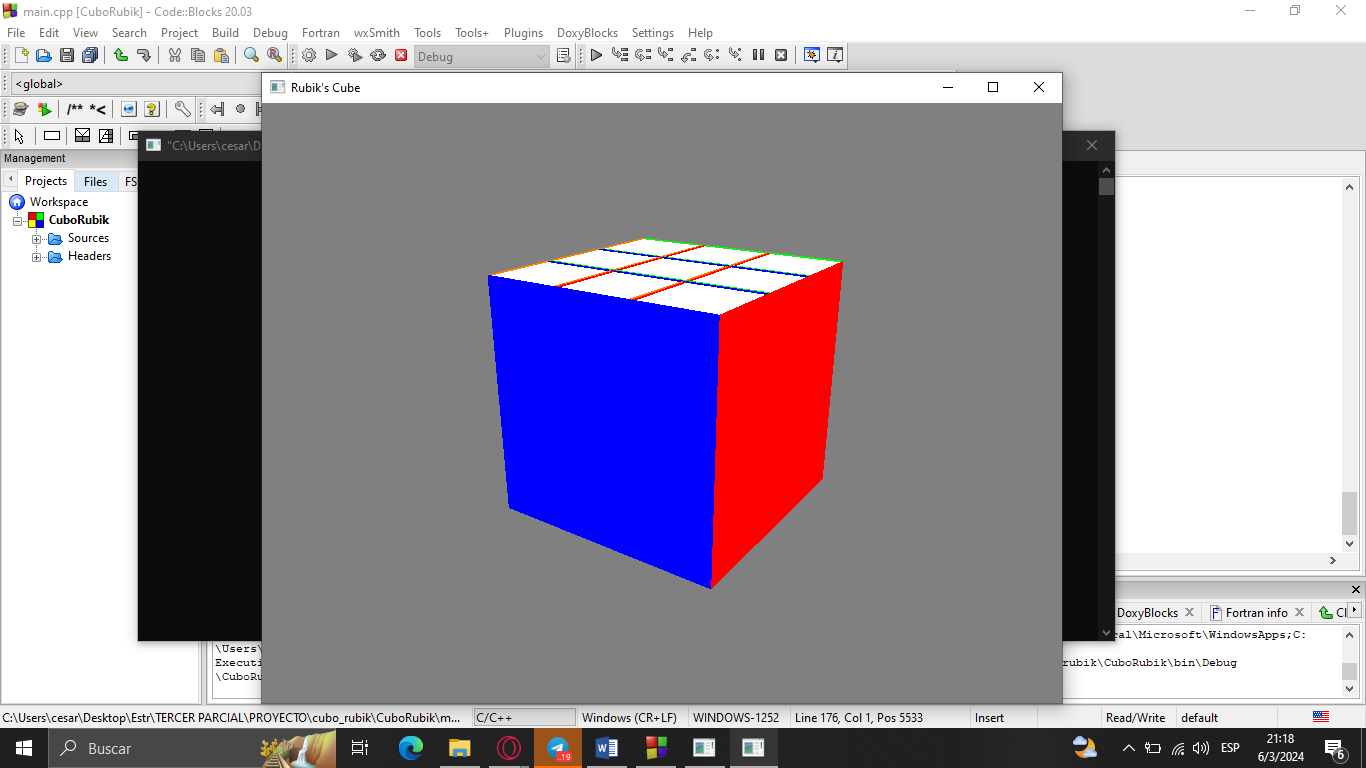


Figura 2.2.3.3. Cubo Rubik estilizado

### 2.2.4 Adición de lineas negras:

**Actualización de Display():** Se agregaron caras negras a las caras del cubo que no se veían.

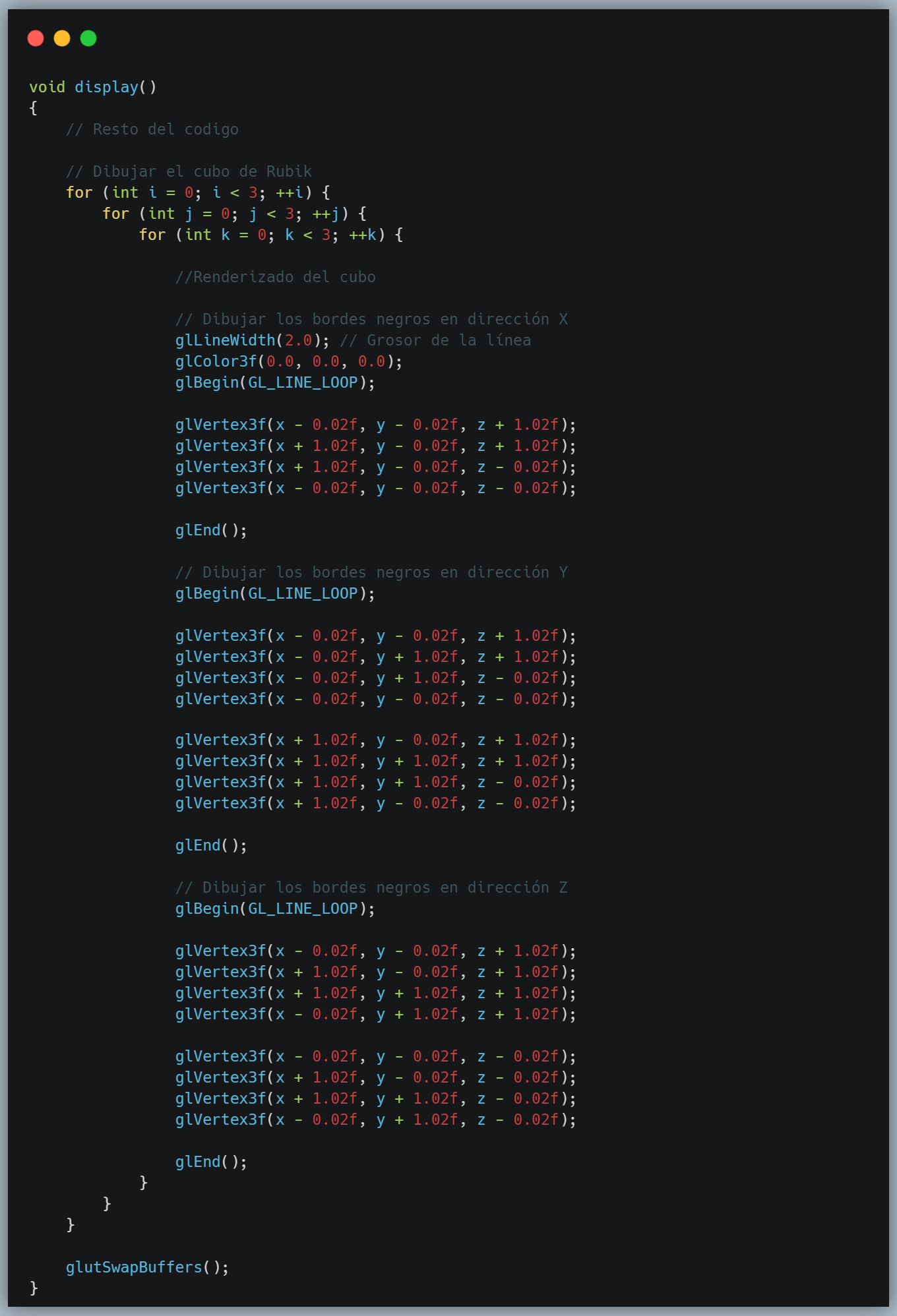


Figura 2.2.4.1. Líneas Negras Display



Figura 2.2.4.2. Líneas negras en caras para visualizar las separaciones del Rubik

### 2.2.5 Organización del proyecto

**Rubik.h:** Se crea la estructura Rubik en un archivo separada para manejar el arreglo tridimensional

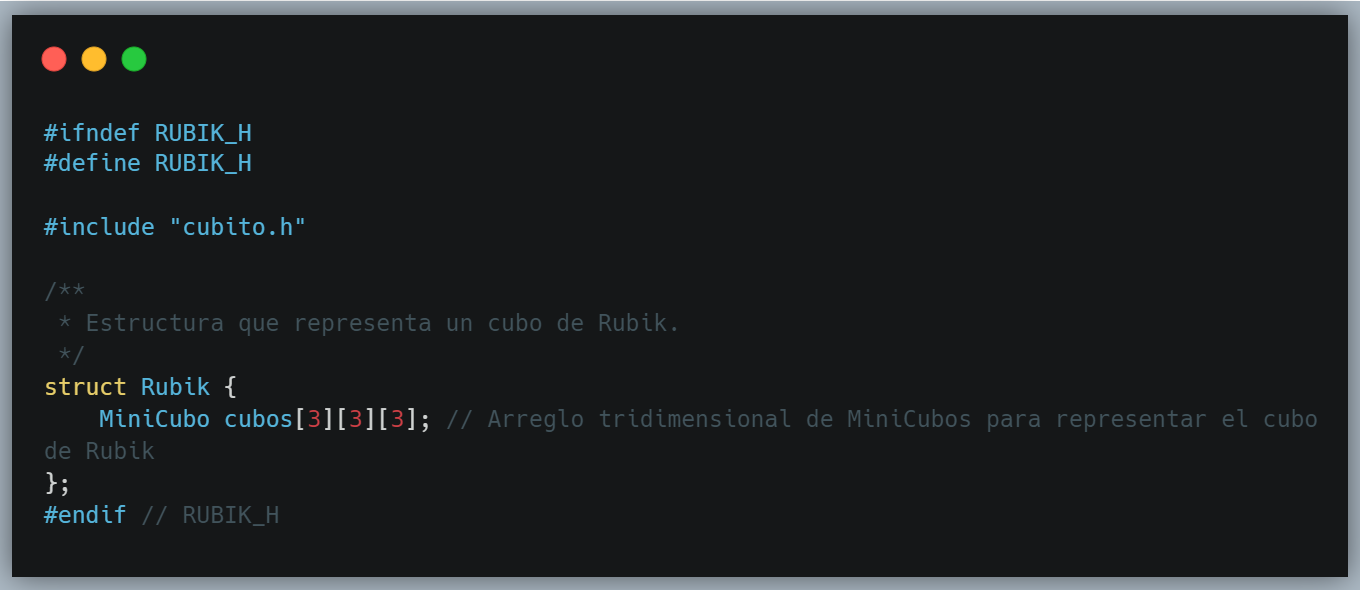


Figura 2.2.5.1. Estructura Rubik en Rubik.h

**Cubito.h:** Se crea el header cubito.h para manejar los minicubos que conforman el Rubik y se agrega las rotaciones en su propio eje para visualizar las rotaciones de las capas.

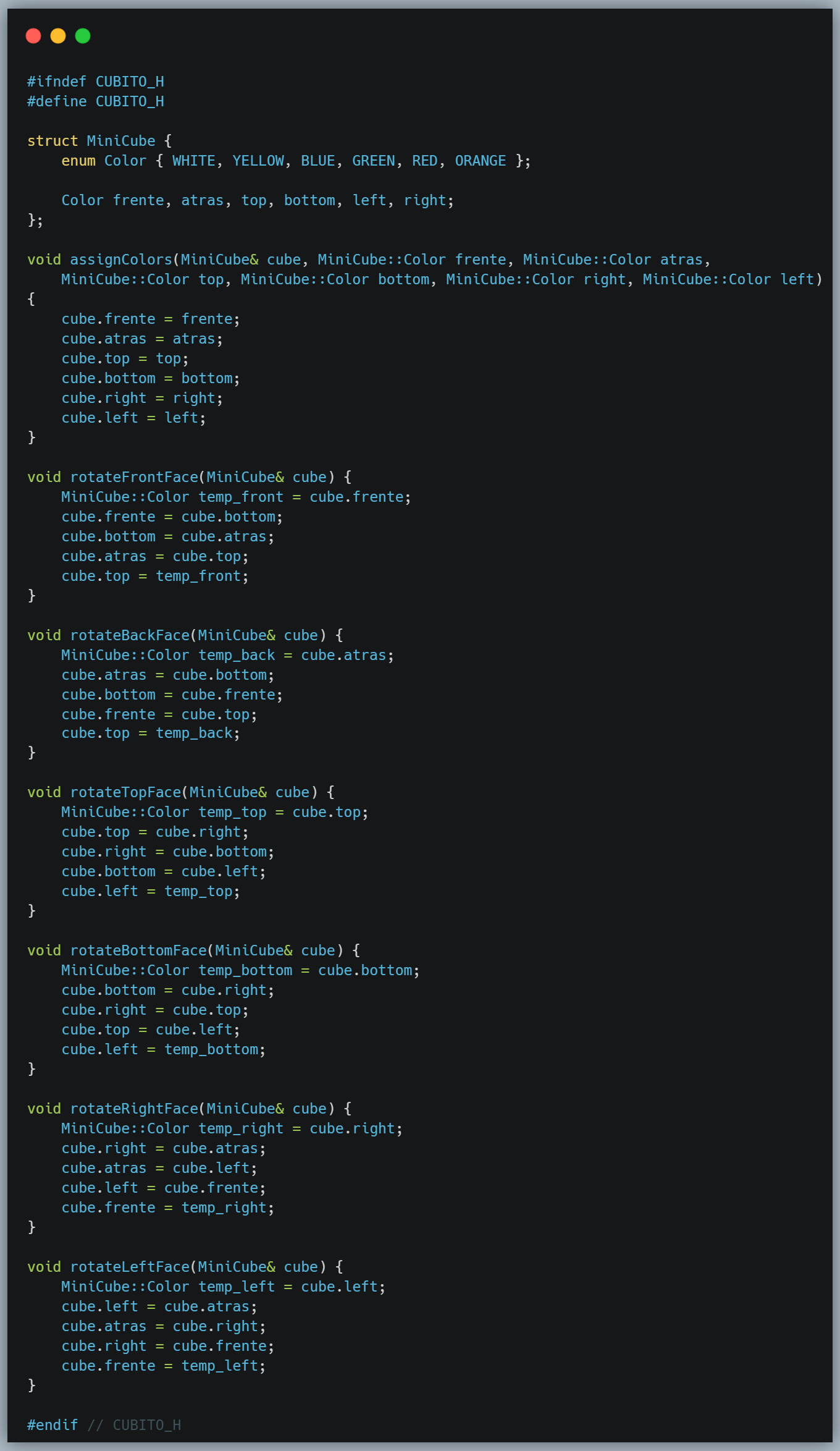


Figura 2.2.5.2. Cubito.h

**Rotaciones.h:** Se crea el header que controla las rotaciones de las capas del Rubik, cada funcion mueve de posicion los minicubos y llama a rotar a estos para simular el giro de un Rubik.



Figura 2.2.5.3. Rotaciones

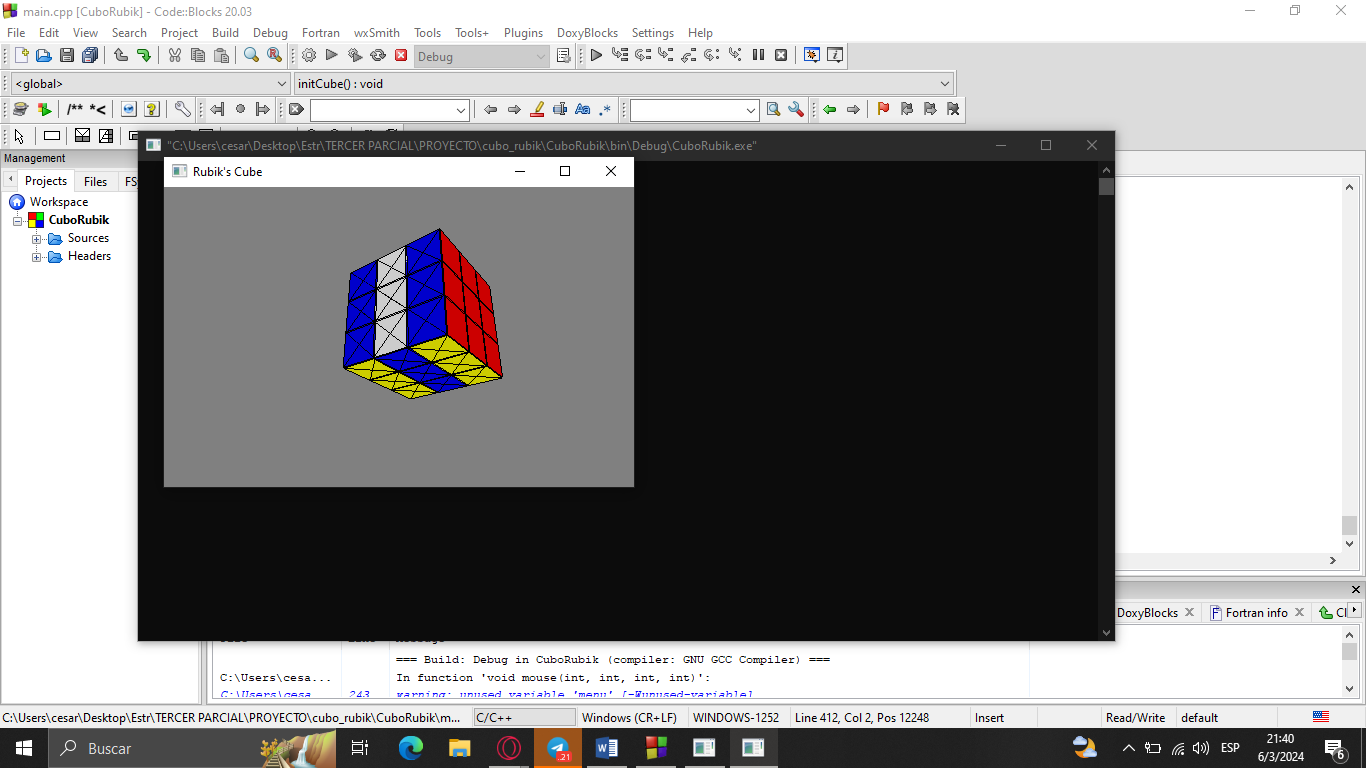


Figura 2.2.5.4. Cubo Rubik mejora de movilidad

### 2.2.6 Organización del código:

**Documentación:** Se documentó el código para facilitar su comprensión y mantenimiento.

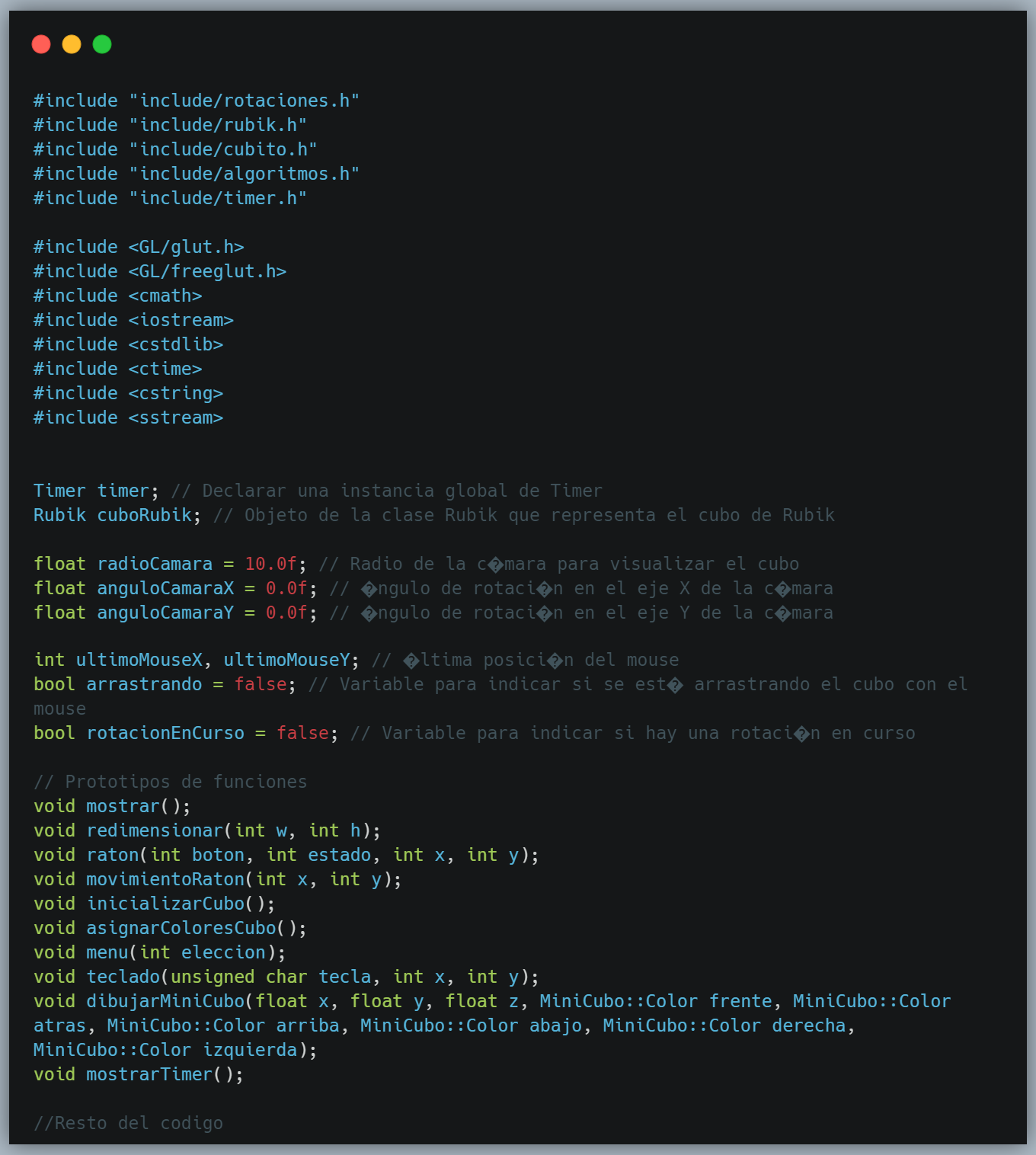


Figura 2.2.6.1. Documentación

### 2.2.7. Corrección de errores y nuevas funciones:

Se agregaron nuevas teclas para regresar al inicio y generar colores aleatorios.

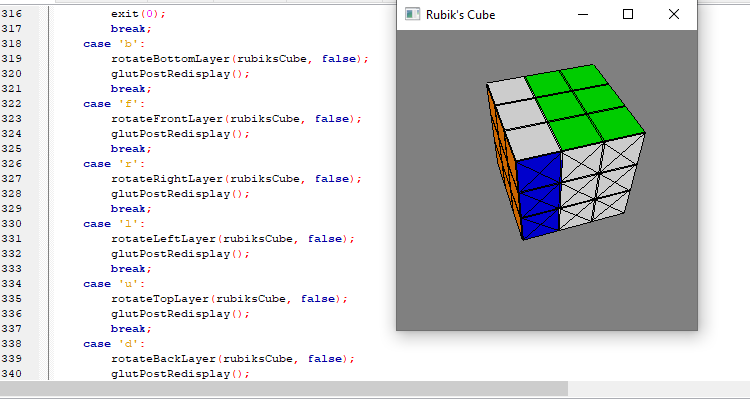


Figura 2.2.7.1. Cubo rubik colores aleatorios por teclado y giros correctos

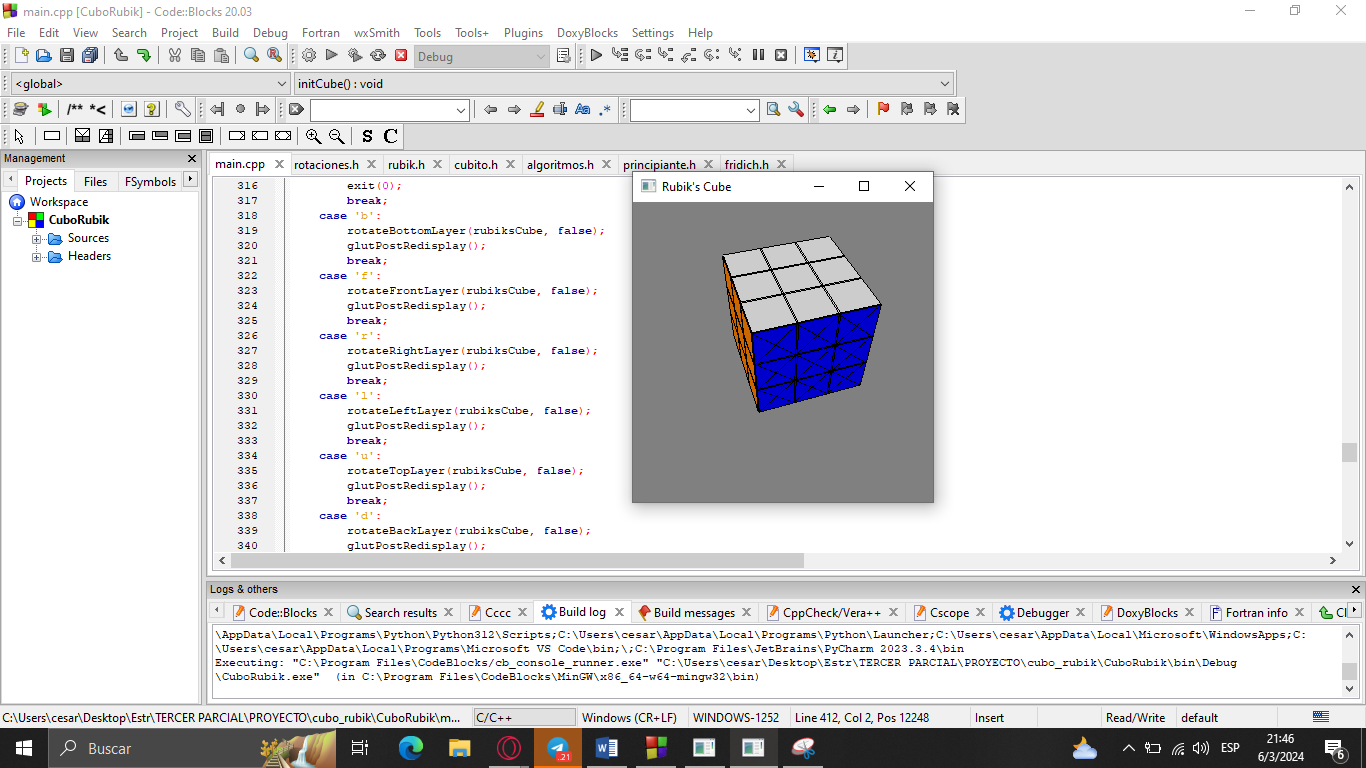


Figura 2.2.7.2. Cubo rubik colores iniciales mediante teclado

## 2.2 Algoritmos empleados:

**Asignación de colores del Rubik:** Este algoritmo asigna colores a cada cara de un cubo de Rubik utilizando una matriz tridimensional. Recorre cada posición del cubo y asigna colores según su ubicación en la matriz, siguiendo una convención estándar de colores para un cubo de Rubik.



Figura 2.2.1. Función de asignar colores al cubo

**Dibujar el cubo de Rubik:** Esta función dibuja un mini cubo en una escena 3D utilizando OpenGL. Toma como entrada las coordenadas (x, y, z) de la posición del cubo y los colores de cada una de sus caras Luego, dibuja cada cara del cubo con los colores proporcionados.

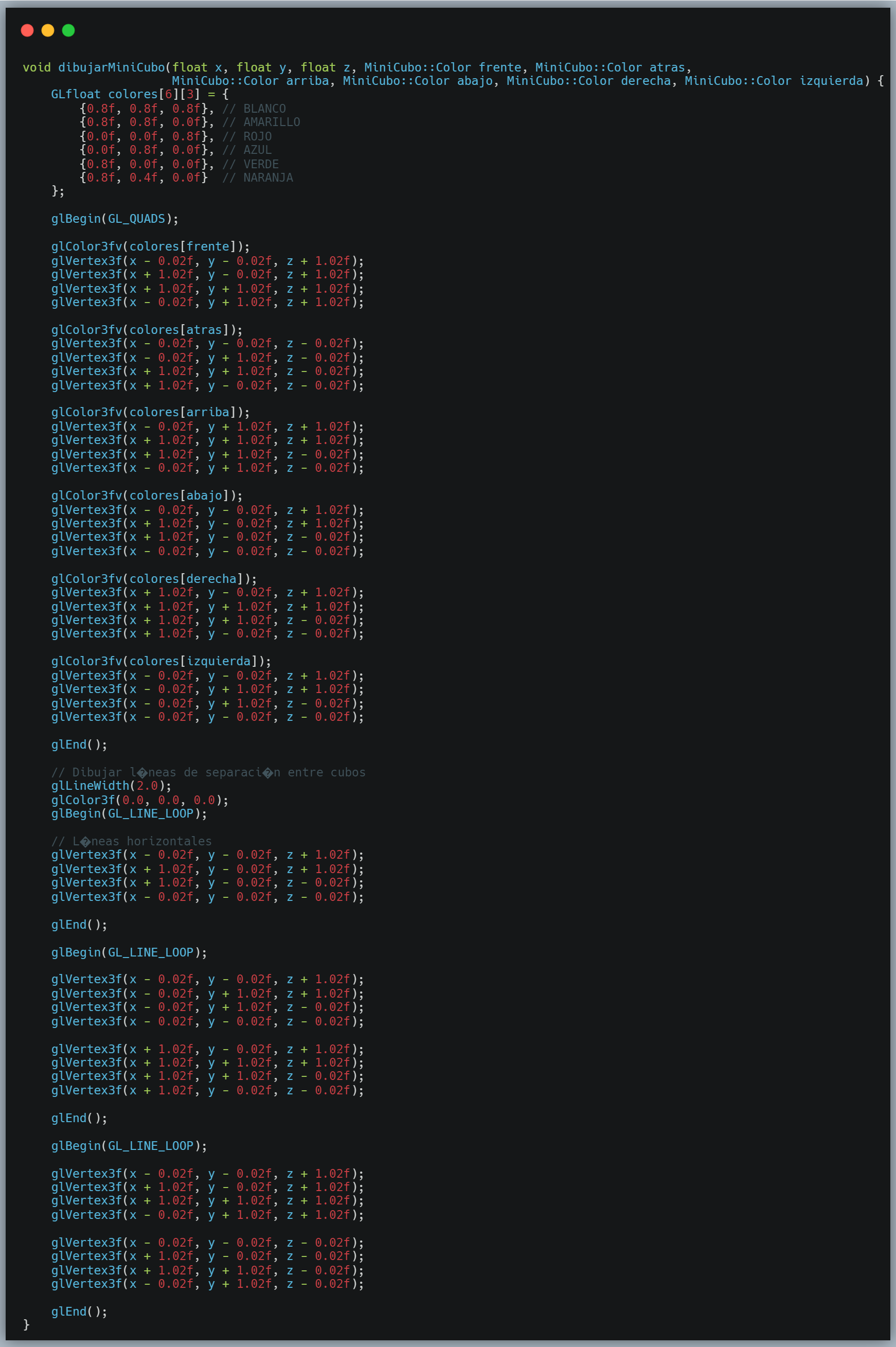


Figura 2.2.2. Dibujar Cubo

**Mostrar:** La función mostrar() utiliza OpenGL para representar el cubo de Rubik en la pantalla. Primero, limpia los búferes de color y profundidad, ajusta la posición de la cámara y establece el color de fondo. Luego, recorre cada mini cubo del cubo Rubik para dibujarlo con sus respectivos colores. Finalmente, muestra un temporizador y realiza el intercambio de búferes para visualizar los cambios en la pantalla.



Figura 2.2.3. Mostrar

**Redimensión:** La función redimensionar() se utiliza para ajustar la visualización en caso de que la ventana cambie de tamaño. Primero, establece el área de visualización utilizando glViewport con las dimensiones proporcionadas. Luego, configura la matriz de proyección utilizando glMatrixMode(GL\_PROJECTION) y gluPerspective para definir la perspectiva de la cámara con un ángulo de visión de 45 grados, una relación de aspecto basada en el ancho y alto de la ventana, y un rango de visión entre 1.0 y 100.0 unidades. Finalmente, se restaura la matriz de vista del modelo utilizando glMatrixMode(GL\_MODELVIEW) y glLoadIdentity().



Figura 2.2.4. Redimensión

**MenuCallback:** La función menuCallback() es un callback utilizado para manejar las selecciones realizadas en un menú. Dependiendo de la opción seleccionada, ejecuta acciones específicas.

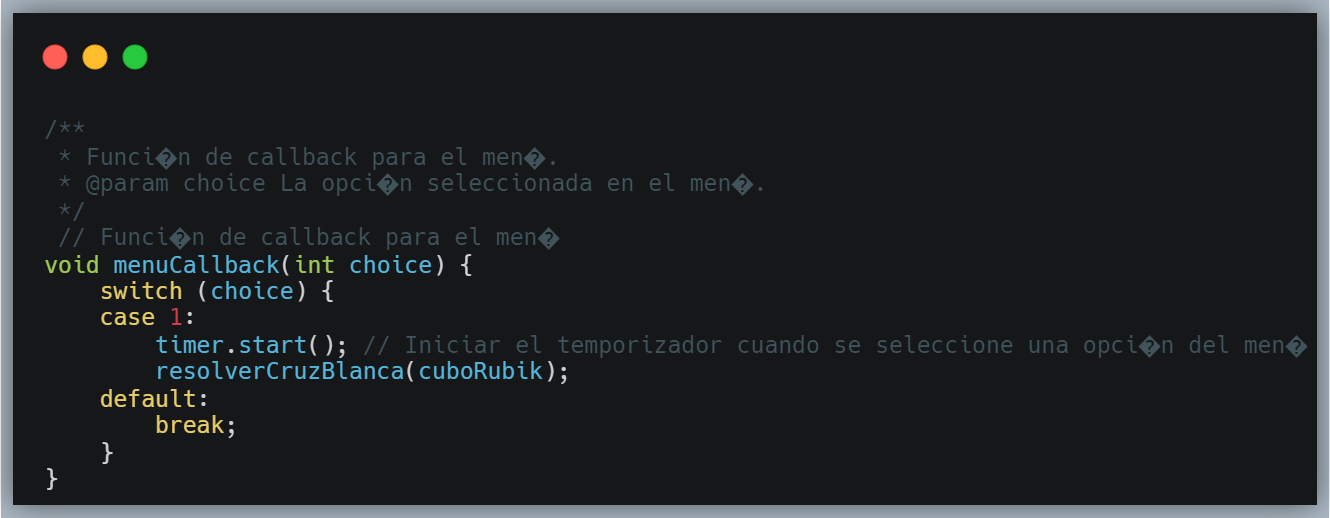


Figura 2.2.5. menuCallback

**Mostrar timer:** La función mostrarTimer() se encarga de mostrar el temporizador en la pantalla utilizando OpenGL. Primero, configura la matriz de proyección para permitir la colocación del texto en cualquier posición de la ventana. Luego, dibuja el texto del temporizador en la esquina superior derecha de la ventana, utilizando el tiempo transcurrido obtenido del temporizador. Después, dibuja un fondo semitransparente detrás del texto para mejorar la legibilidad. Finalmente, restaura las matrices de proyección y modelo al estado previo.



Figura 2.2.6. Mostrar Timer

**Raton:** La función raton() maneja los eventos del ratón en la ventana. Si el botón izquierdo del ratón se presiona, guarda la posición del ratón (x y y) y establece la bandera arrastrando en verdadero. Si el botón izquierdo se libera, la bandera arrastrando se establece en falso. Además, si el botón derecho del ratón se presiona, se crea un menú contextual con una opción "Resolver" y se adjunta al botón derecho del ratón, utilizando la función menuCallback para manejar las selecciones del menú.



Figura 2.2.7. void Raton

**Movimiento del ratón:** La función movimientoRaton() se encarga de manejar los movimientos del ratón cuando se está arrastrando. Calcula la diferencia de movimiento del ratón desde la última posición y actualiza los ángulos de rotación de la cámara (anguloCamaraX y anguloCamaraY) según estos movimientos. Luego, actualiza la última posición del ratón y fuerza el redibujado de la escena para reflejar los cambios. Esto permite que el usuario pueda rotar la cámara arrastrando el ratón.



Figura 2.2.8. Movimiento Ratón

**Teclado**: La función teclado() maneja los eventos del teclado en la ventana. Primero, verifica si no hay una rotación en curso antes de procesar la entrada del teclado. Luego, dependiendo de la tecla presionada, realiza diversas acciones. Por ejemplo, si se presiona la tecla 'q' o 'Q', sale del programa. Si se presiona una tecla correspondiente a una rotación específica ('b', 'f', 'r', 'l', 'u', 'd', 'B', 'F', 'R', 'L', 'U', 'D', 'm', 'M', 'e', 'E', 's', 'S'), se inicia la rotación correspondiente en el cubo de Rubik. Después de realizar la rotación, se establece una bandera rotacionEnCurso en verdadero y se programa un temporizador para que esta bandera se restablezca después de un breve período de tiempo, permitiendo así que se realicen rotaciones consecutivas. Finalmente, si se presiona la tecla 'o' o 'O', se llama a la función asignarColoresCubo() para restablecer los colores del cubo de Rubik a su estado inicial.



Figura 2.2.9. void teclado

**Inicializar el cubo:** La función inicializarCubo() se encarga de configurar y preparar el entorno de renderizado en 3D antes de mostrar el cubo de Rubik en la pantalla. Primero, configura el modelo de sombreado utilizando glShadeModel(GL\_SMOOTH) para obtener sombras suaves. Luego, habilita la prueba de profundidad para el renderizado en 3D utilizando glEnable(GL\_DEPTH\_TEST), lo que asegura que los objetos se rendericen correctamente en relación con su profundidad en la escena. Finalmente, llama a la función asignarColoresCubo() para inicializar los colores del cubo de Rubik.

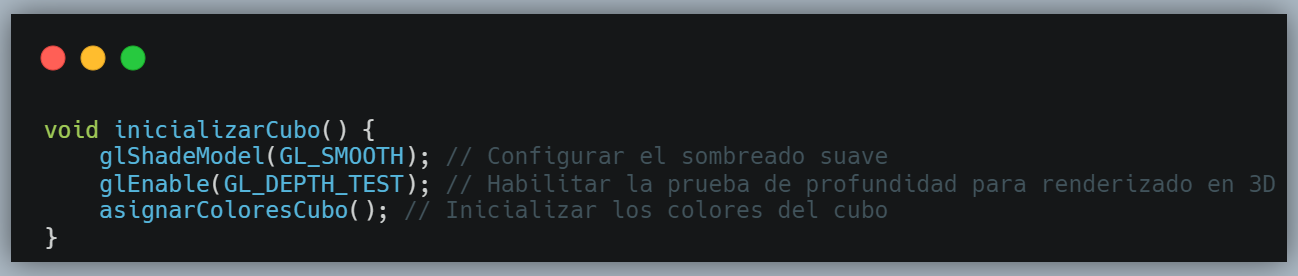
****

Figura 2.2.10. void incializarCubo

**Main:** El main() es la función principal del programa. En primer lugar, se inicializa el cubo de Rubik llamando a la función inicializarCubo(). Luego, se inicializa GLUT y se configura la ventana de visualización. Se habilita la prueba de profundidad y el blending para permitir transparencias. A continuación, se registran las funciones de callback para eventos de OpenGL, como el renderizado, el redimensionamiento de la ventana, el manejo de eventos del ratón, el movimiento del ratón, el manejo de eventos de teclado normal y el manejo de eventos de teclas especiales. Finalmente, se inicia el bucle principal de GLUT con glutMainLoop(). Una vez que el bucle principal finaliza, el programa devuelve 0 como código de salida.

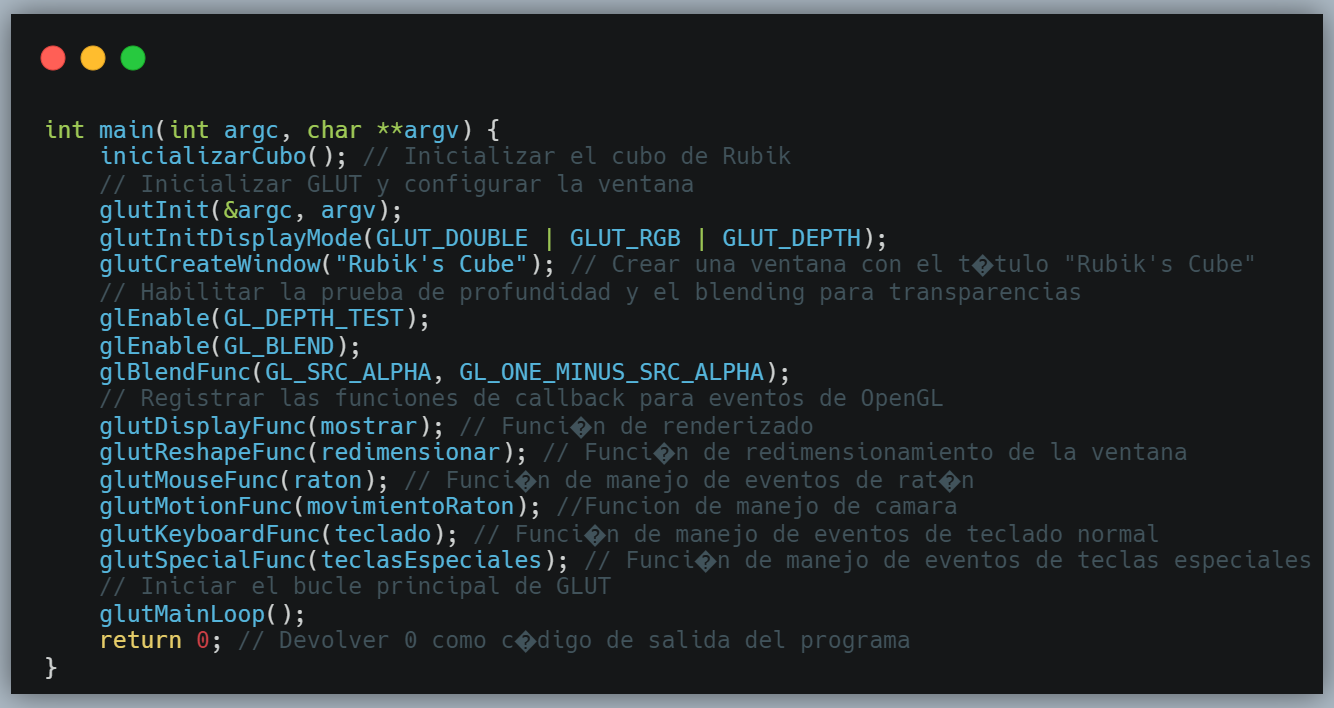


Figura 2.2.11. Main

**Rotaciones de capas:** La función rotarCapa…() se encarga de realizar una rotación en una capa específica del cubo de Rubik en el sentido indicado, ya sea en sentido horario o antihorario. Para esto, se crea una capa temporal para almacenar los mini cubos de la capa que se va a rotar. Posteriormente, se copian los mini cubos de la capa original a la capa temporal. Luego, se procede a realizar la rotación de los mini cubos en la capa según el sentido especificado. Si la rotación es en sentido horario, los mini cubos se desplazan en una dirección específica (por ejemplo, hacia la derecha para la capa derecha), mientras que si es en sentido antihorario, se desplazan en la dirección opuesta. Además de rotar los mini cubos, también se rota la cara correspondiente de cada mini cubo individualmente para mantener la coherencia de los colores en el cubo. Este proceso se repite para todas las capas del cubo de Rubik que pueden ser rotadas (superior, inferior, frontal, trasera, izquierda, derecha, media, equatorial y standing).



Figura 2.2.12. RotarCapaDerecha

Rotar caras: La función rotarCara() realiza la rotación de una cara específica de un mini cubo del cubo de Rubik. Para ello, se intercambian los colores de las posiciones que conforman la cara en un patrón específico que mantiene la consistencia de los colores en el cubo. Primero, se guarda el color de la cara en una variable temporal. Luego, se actualizan los colores de las posiciones que conforman la cara, intercambiándolos entre sí en un patrón determinado. De esta manera, se simula la rotación de la cara del mini cubo, ya sea en sentido horario o antihorario, dependiendo de la dirección de la rotación especificada.

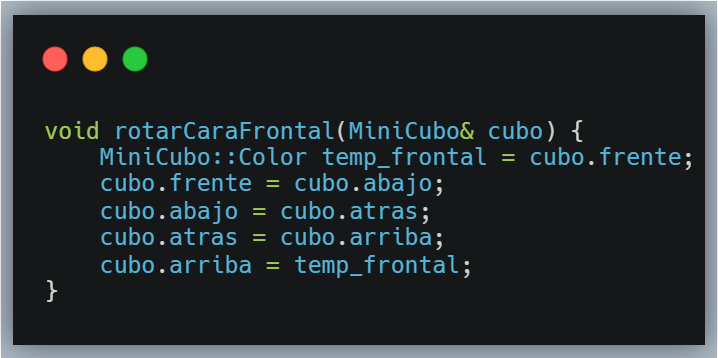


Figura 2.2.13. RotarCaraFrontal

# 3. Conclusiones

* **Interactividad y Experiencia de Usuario:** El proyecto logra proporcionar una experiencia interactiva satisfactoria al permitir al usuario manipular el cubo de Rubik a través del teclado y el ratón. Esto añade un elemento de interactividad que mejora la experiencia de usuario y hace que el proyecto sea más atractivo.
* **Implementación de Algoritmos:** La implementación de algoritmos para resolver el cubo de Rubik demuestra un buen nivel de comprensión y habilidad para aplicar algoritmos complejos en un entorno gráfico. Esto muestra la capacidad del desarrollador para abordar desafíos algorítmicos y su habilidad para traducirlos en código funcional.
* **Organización y Modularidad del Código:** La estructura del código está organizada en diferentes archivos y clases, lo que facilita su comprensión y mantenimiento. La división en archivos de encabezado e implementación, junto con el uso de clases, mejora la modularidad del código y lo hace más escalable y fácil de mantener.

# 4. Recomendaciones

* Optimización del Renderizado: Considerar técnicas de optimización para mejorar el rendimiento del renderizado, especialmente cuando se manipula el cubo de Rubik en tiempo real. Esto puede incluir técnicas como el uso de listas de visualización o buffers de vértices para reducir la carga de renderizado.
* Mejora de la Usabilidad: Añadir comentarios en el código para mejorar la comprensión y la mantenibilidad del mismo. Además, se podría implementar una interfaz de usuario más intuitiva, como menús desplegables o botones, para facilitar la interacción del usuario con el cubo de Rubik.
* Ampliación de Funcionalidades: Explorar la posibilidad de añadir características adicionales, como la generación aleatoria de configuraciones de cubo de Rubik para resolver, o la implementación de algoritmos más avanzados para la resolución automática del cubo. Esto podría aumentar la utilidad y la diversión del proyecto.

# 5. Referencias Bibliograficas

deckerix. (2024). *Primeros pasos con OpenGL en Windows y Code::Blocks – El mundo de deckerix*. Deckerix.com. http://deckerix.com/blog/primeros-pasos-con-opengl-en-windows-y-codeblocks/#google\_vignette

*Historia del cubo de Rubik | Biblioteca ULPGC*. (2024). Ulpgc.es. https://biblioteca.ulpgc.es/exposicion-ingenio3-rubik-historia

*Code With C*. (2014, August 15). Code with C. https://www.codewithc.com/how-to-setup-opengl-glut-in-codeblocks/

‌