Proyecto final cubo de rubik

Carlos G. Morales - Cristhian O. Pinto - Aaron B. Ramirez
Departamento de Computacion
Universidad Catolica San Pablo
Email's: carlos.morales@ucsp.edu.pe - cristhian.ocola@ucsp.edu.pe - aaron.blanco@ucsp.edu.pe

I. Introduction

El objetivo de este proyecto es implementar un cubo de Rubik utilizando OpenGL. El cubo de Rubik es un famoso rompecabezas tridimensional que consiste en un cubo compuesto por varias piezas móviles. Cada pieza tiene un color y el objetivo del juego es resolver el cubo girando las diferentes capas para que cada cara del cubo tenga un solo color.

Para lograr esto, se utilizará la biblioteca GLM Graphics para manejar la representación gráfica en 3D del cubo y GLFW para crear la ventana de la aplicación. También se utilizarán las estructuras "Cube" y "Rubik" para representar las piezas individuales y el cubo de Rubik en su conjunto.

La estructura "Rubik" contendrá una colección de objetos "Cube", donde cada objeto representa una pieza individual del cubo. Además, se utilizarán vectores y matrices de la biblioteca GLM para manipular y transformar las piezas en respuesta a las interacciones del usuario.

El cubo de Rubik consta de diferentes capas que se pueden girar en diferentes direcciones. Se utilizarán las funciones de transformación implementadas en la estructura "Rubik" para girar las capas del cubo en respuesta a los movimientos del usuario. Esto permitirá simular los movimientos del cubo de Rubik y visualizar los cambios en tiempo real.

Además de la funcionalidad básica de girar las capas del cubo, se implementarán animaciones para mejorar la experiencia visual. Estas animaciones permitirán que las capas giren suavemente y de manera controlada, lo que dará una sensación más realista al resolver el cubo de Rubik.

II. DESCRIPCIÓN DE LA ANIMACIÓN Y ESTRUCTURA

Hemos implementado una simulación de efecto de respiración y animación en un cubo de Rubik utilizando mi lenguaje de programación. Este código nos permite crear un programa que simula un cubo de Rubik en movimiento, generando una ilusión de expansión y contracción del cubo, como si estuviera respirando.

Para lograr este efecto, he utilizado varias funciones dentro del código. Por ejemplo, la función 'respirar' se encarga de generar el movimiento de respiración en el cubo, donde el cubo se expande y contrae de manera suave y continua. Además, hemos implementado la función 'mover_cubito' que me permite mover un cubito individual en una dirección específica.

Estas funciones, junto con otras partes del código, trabajan en conjunto para crear una animación realista del cubo de Rubik en movimiento. A medida que exploramos las explicaciones más detalladas, podrás comprender mejor cómo funciona cada parte del código y cómo contribuye al efecto final.

A. ESTRUCTURA

1) Rubik: : La estructura Rubik representa un cubo de Rubik y contiene varias variables y funciones miembro. Las variables miembro incluyen cubos (vector de cubos individuales que forman el cubo de Rubik), IPT (vector de índices para dibujar el cubo), ind_tam (tamaño del vector de índices), num_cubos (número total de cubos en el cubo de Rubik), centros (vector de coordenadas del centro de cada cubo), puntos (vector de puntos), camadas (vector de vectores de pares que representan las capas del cubo de Rubik), angulo_i y angulo_f (ángulos inicial y final para la animación), angulo_i_animacion y angulo_f_animacion (ángulos inicial y final para la animación del átomo), pasos_animacion (cantidad de pasos para la animación del átomo), pasos (cantidad de pasos para la animación del cubo de Rubik), sentido (dirección de rotación), y eje (eje de rotación).

El constructor Rubik() inicializa el cubo de Rubik y genera los centros de los cubos, así como los índices y los colores de las capas del cubo.

2) Rubik::TransformBloques(): : Esta función se encarga de aplicar una transformación a un conjunto de cubos específicos. Recibe como parámetros una matriz de transformación, un vector de desplazamiento y un entero que representa el tipo de operación a realizar en los cubos (rotación en X, Y o Z). La función itera sobre los cubos en la camada especificada y aplica la transformación a cada uno de ellos.

3) Rubik::cambiar_centros(): : Esta función se encarga de cambiar los centros de los cubos en el objeto Rubik. Genera los nuevos centros de los cubos en un patrón específico y los asigna a la variable centros del objeto Rubik.

4) Rubik::TransformAnimado(): : Esta función realiza una transformación animada en un conjunto de cubos específicos. Recibe una matriz de transformación, un vector de desplazamiento, un entero que representa el tipo de operación a realizar en los cubos y una lista de pares que indica qué cubos de la camada deben ser transformados. La función verifica el valor de ind para determinar qué cubos deben ser transformados en función de la camada específica, y luego aplica la transformación a los cubos correspondientes.

5) Rubik::Transform(): : Esta función aplica una transformación a todos los cubos en el objeto Rubik. Recibe una

matriz de transformación, un vector de desplazamiento y un entero que representa el tipo de operación a realizar en los cubos (rotación en X, Y o Z). La función itera sobre todos los cubos y aplica la transformación a cada uno de ellos.

6) Rubik::GenEbo(): : Esta función genera un vector de índices (Element Buffer Object) que se utiliza para renderizar los cubos. Itera sobre todos los cubos y utiliza la función Cube::GenEbo() para obtener los índices de cada cubo. Luego, agrega estos índices al vector de salida.

7) Rubik::PushVector(): : Esta función agrega los elementos de un vector B a otro vector A. Es utilizada por la función GenEbo() para agregar los índices de los cubos al vector de salida.

8) Rubik::GenCentrosCubos(): : Esta función genera los centros de los cubos en el objeto Rubik. Calcula los centros en función del tamaño de los cubos y los espacios entre ellos, y los agrega al vector centros del objeto Rubik.

9) Rubik::GenIndicesCamadas(): : Esta función genera los índices de las diferentes capas del cubo de Rubik. Crea una lista de vectores de pares, donde cada par contiene el índice de un cubo y un carácter que representa el color de ese cubo. Cada vector de pares representa una capa del cubo (frente, trasera, superior, inferior, izquierda o derecha). Los índices se generan en función del tamaño del cubo y se almacenan en la variable camadas del objeto Rubik.

10) Rubik::animacion_atomo(): : Esta función realiza una animación del átomo en el cubo de Rubik. Realiza una rotación en el eje Z de una capa de cubos seleccionada, generando una animación en la que los cubos se mueven alrededor del centro de la capa.

11) Rubik::MoverCamadaFrontal(): : Esta función realiza un movimiento de la capa frontal del cubo de Rubik. Recibe como parámetros un entero que representa la dirección del movimiento (1 o -1) y un entero que representa la cantidad de pasos para la animación. La función aplica una transformación a la capa frontal y actualiza los centros de los cubos.

12) Rubik::MoverCamadaLateral(): : Esta función realiza un movimiento de la capa lateral del cubo de Rubik. Recibe como parámetros un entero que representa la dirección del movimiento (1 o -1) y un entero que representa la cantidad de pasos para la animación. La función aplica una transformación a la capa lateral y actualiza los centros de los cubos.

B. ANIMACIONES

1) Respirar: : Esta función se encarga de realizar el movimiento de respiración del cubo Rubik. Toma como entrada un parámetro f que indica la dirección del movimiento. La función genera una serie de números aleatorios y aplica una transformación de translación a cada uno de los cubos del cubo Rubik. Dependiendo del número aleatorio generado y la dirección del movimiento, se desplaza cada cubo en una dirección determinada. La diferencia entre los cubitos con números aleatorios, es mínima, de esta forma no se nota a simple vista que todos están yendo a diferentes posiciones.

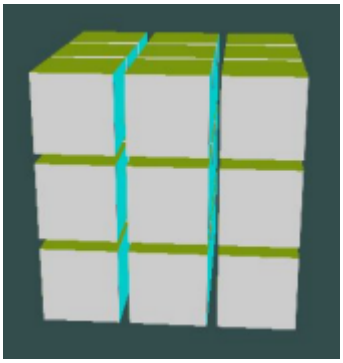

Fig. 1. Cubo rubik inicial.

2) Mover_cubito: : Esta función se utiliza para mover un cubito específico del cubo Rubik en una dirección determinada. Toma como entrada un parámetro mov que indica la cantidad de movimiento y un parámetro ax que indica el eje en el que se realizará el movimiento (X, Y o Z). La función aplica una transformación de translación al cubito seleccionado en la dirección especificada.

3) Animacion_pal: : Esta función se utiliza para realizar la animación de los cubitos que forman una palabra en el cubo Rubik. Toma como entrada un parámetro mov que indica la cantidad de movimiento y un parámetro eje que indica el eje en el que se realizará el movimiento (X, Y o Z). La función aplica una transformación de translación a los cubitos de la palabra en la dirección especificada.

4) Animacion_palabra: : Esta función controla la animación de la palabra formada por los cubitos del cubo Rubik. Realiza la animación de los cubitos uno por uno en orden, siguiendo las coordenadas especificadas en el vector letras_puntos. Utiliza las funciones animacion_pal() y reiniciar_estados_cubo() para realizar la animación de cada cubito y reiniciar los estados después de completar una animación.

5) Reiniciar_estados_cubo: : Esta función se utiliza para reiniciar los estados de los cubitos del cubo Rubik. Establece todos los estados de los cubitos en 0, lo que indica que no han sido animados.

III. FUNCIONALIDADES

Instrucciones:

- **F** → Camada frontal
- **B** → Camada trasera
- **R** → Camada derecha
- **L** → Camada izquierda
- **U** → Camada superior
- **D** → Camada inferior

Para ingresar movimiento:

- **W** → Para ingresar movimiento

Movimientos inversos:

- **S** → Movimientos inversos

Generar solución:

- **E** → Para generar solve

Animaciones:

- **Q** → Animación de la solución
- **A** → Animación de átomo
- **Z** → Animación para la letra, pero el cubo debe estar armado de forma original
- **X** → Cambiar sentido para reconstruir cubo después de formar la palabra
- **P** → Parar animación de átomo y retornarla a la forma original de cubo

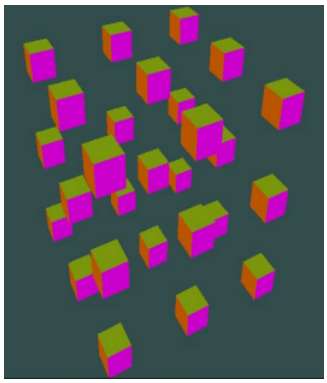

Fig. 2. Respiración.

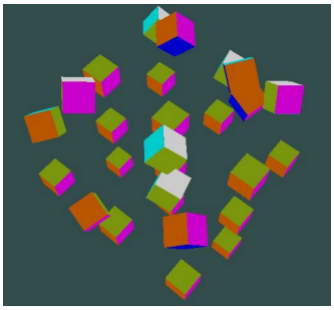

Fig. 3. Movimiento átomo.

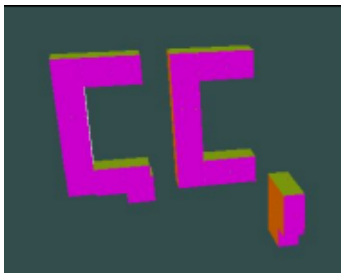

Fig. 4. Formación de letras 1.

IV. Problemas encontrados en la implementación

A medida que te sumerges en la creación de esta animación única, que combina la estructura de un átomo con los movimientos de un Cubo de Rubik, es natural encontrarse con desafíos al tratar de fusionar estas dos implementaciones

Fig. 5. Formación de letras 2.

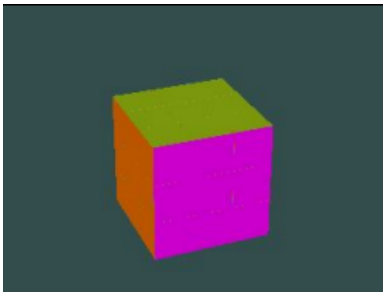

Fig. 6. Formación de letras 3.

distintas. La transición fluida de la representación del átomo hacia la formación de palabras requiere una atención especial, ya que ambos elementos poseen características diferentes que deben amalgamarse de manera coherente y visualmente atractiva.

Uno de los desafíos principales es establecer una conexión lógica y comprensible entre las rotaciones que simulan los movimientos de los átomos y las traslaciones necesarias para formar palabras. En la animación del átomo, las rotaciones se inspiran en los movimientos de los electrones y las órbitas atómicas. Estos movimientos suelen ser suaves y fluidos, con una sensación de ingravidez y armonía. Por otro lado, al formar palabras, las traslaciones de los cubos de Rubik deben ser más definidas y precisas, siguiendo un patrón específico para ensamblar las letras y palabras deseadas.

El primer problema que surge es cómo lograr una transición coherente entre las rotaciones suaves y las traslaciones más definidas. Si las rotaciones del átomo se detienen bruscamente y se inician las traslaciones, la animación puede perder su fluidez y parecer abrupta e incoherente. Por otro lado, si se intenta mantener la suavidad de las rotaciones al formar palabras, es posible que los cubos de Rubik no encajen correctamente o que las letras no se formen de manera clara y legible.

Otro desafío importante es la sincronización precisa de los movimientos. Dado que los cubos de Rubik deben girar y desplazarse en momentos específicos para formar palabras, es crucial establecer una coordinación perfecta entre las rotaciones y las traslaciones. Si los movimientos no están sincronizados correctamente, se corre el riesgo de que las palabras no se formen de manera fluida o que los cubos se

muevan de manera errática e inconsistente, dificultando la comprensión y la experiencia visual para el espectador.

Además, debes considerar la complejidad de la coreografía entre los cubos de Rubik mientras se realizan las traslaciones. Asegurarse de que cada cubo se mueva de manera precisa y en el orden correcto para formar las palabras puede ser un desafío técnico significativo. La interacción entre los diferentes cubos y su relación espacial también debe ser cuidadosamente planificada y ejecutada, para evitar colisiones no deseadas o una apariencia caótica durante el proceso de formación de palabras.

V. Próximos pasos

A continuación, se presentan los próximos pasos para mejorar el código y ampliar la funcionalidad del programa:

- **Optimización del código:** Se puede revisar el código en busca de oportunidades de optimización para mejorar el rendimiento y la eficiencia del programa.
- **Mejora de la animación:** Se puede trabajar en mejorar las animaciones para que sean más suaves y realistas. Esto podría implicar ajustar los valores de sensibilidad del ratón y otras configuraciones relacionadas con el movimiento de la cámara y los cubos.
- **Implementación de algoritmos para resolver el cubo automáticamente:** Se puede incorporar algoritmos populares de resolución del cubo de Rubik, como el método de capas o el método de Fridrich, para permitir que el programa resuelva automáticamente el cubo a partir de un estado desordenado.
- **Interfaz de usuario mejorada:** Se puede trabajar en mejorar la interfaz de usuario para que sea más intuitiva y fácil de usar. Esto puede incluir agregar botones y controles para ejecutar diferentes funciones y proporcionar retroalimentación visual sobre el estado del cubo.
- **Personalización de la secuencia de movimientos:** Se podría permitir al usuario ingresar sus propias secuencias de movimientos para resolver el cubo o realizar patrones específicos.
- **Guardar y cargar estados del cubo:** Implementar la funcionalidad para guardar el estado actual del cubo en un archivo y cargarlo posteriormente para continuar con la resolución o la animación.
- **Añadir soporte para otros tipos de cubos:** Ampliar el programa para que admita cubos de diferentes tamaños y configuraciones, lo que permitiría a los usuarios explorar y resolver cubos más complejos.
- **Mejora de la documentación:** Escribir una documentación detallada para el código, explicando su estructura y funcionamiento interno. Esto facilitará el mantenimiento y la colaboración con otros desarrolladores.
- **Pruebas y depuración:** Realizar pruebas exhaustivas para asegurarse de que todas las funcionalidades funcionen correctamente y solucionar cualquier error o bug que se encuentre.

Referencias

1) Goel, A., Sharma, S., & Gupta, P. (2018). A Comprehensive Study of Rubik's Cube and its Algorithms. In 2018 9th International Conference on Computing, Communication and Networking Technologies (ICCCNT) (pp. 1-6). IEEE. doi: 10.1109/ICCCNT.2018.8494070
2) Korf, R. E. (1997). Finding Optimal Solutions to Rubik's Cube Using Pattern Databases. In AAAI/IAAI (Vol. 97, pp. 700-705).
3) Chang, E. Y., & Zheng, L. (2009). Rubik's cube solvers and the robot. In 2009 IEEE International Conference on Robotics and Automation (pp. 2841-2846). IEEE. doi: 10.1109/ROBOT.2009.5152656
4) Ortuño, M. T., Ramírez, A. J., & Tirado, G. (2015). Using an ant colony optimization algorithm to solve the Rubik's cube. Swarm and Evolutionary Computation, 24, 1-13. doi: 10.1016/j.swevo.2015.02.003
5) Singmaster, D. (1981). Notes on Rubik's "Magic Cube". Enslow Publishers.
6) Russell, S. J., & Norvig, P. (2009). Artificial Intelligence: A Modern Approach (3rd ed.). Pearson.
7) Köksal, E. (2016). A Computational Approach to Rubik's Cube. In 2016 24th Signal Processing and Communication Application Conference (SIU) (pp. 1067-1070). IEEE. doi: 10.1109/SIU.2016.7496141
8) Documentación de GLFW: https://www.glfw.org/documentation.html
9) Documentación de GLM: https://github.com/g-truc/glm/blob/master/manual.md
10) Tutorial de OpenGL: https://learnopengl.com/