Cubo de Rubik en OpenGL

Integrantes: Saul Andersson Rojas Coila (saul.rojas@ucsp.edu.pe)
Curso: Computación Gráfica CCOMP-1 2021
Departamento: Ciencia de la Computación.
Universidad: Universidad Católica San Pablo, Arequipa - Perú.

Introducción

El trabajo final consiste en el modelado, renderizado y animación de un cubo de Rubik en OpenGL, se implementaron los conceptos básicos para el desarrollo de aplicativos gráficos tridimensionales en OpenGL como:

- Modelado con *vertex buffer objects(VBOs), vertex array objects (VAOs)* y *element buffer objects (EBOs)* **[1].**
- Programación de **Shaders** *vertex shader* y *fragment shader*, además del uso de variables *in*,*out*,*uniform*, y distintos tipos de datos *float, int, veci, mati, Sampler2D (para texturas)*, etc **[2].**
- Texturas aplicadas a la superficie de los objetos, *texture units, sampler2D, Filtering, Wrapping, Minmaps* **[3].**
- Programación de una cámara *lookAt* que permite la visualización del escenario **[6].**
- Programación de la matriz de proyección *ortogonal* y en *perspectiva* que nos permite visualizar de distintos modos nuestro objeto **[5].**
- Animación del cubo de Rubik haciendo uso de funciones trigonométricas, traslación de vectores (*álgebra lineal*), uso de Cuaterniones para la rotaciones, etc **[4],[7],[8].**

Animación escogida

Las animaciones escogidas fueron dos; la primera es llamada *Respiración Continua*, la cual repele y atrae los pequeños cubos que componen el cubo de Rubik, esta simulación de respiración puede controlarse como se verá más adelante; la segunda animación es llamada *Interpolación continua de texturas*, esta animación lo que hace es intercalar dos texturas sobre la superficie de cada cubo, la intercalación es continua durante la ejecución del programa.

Funcionalidades

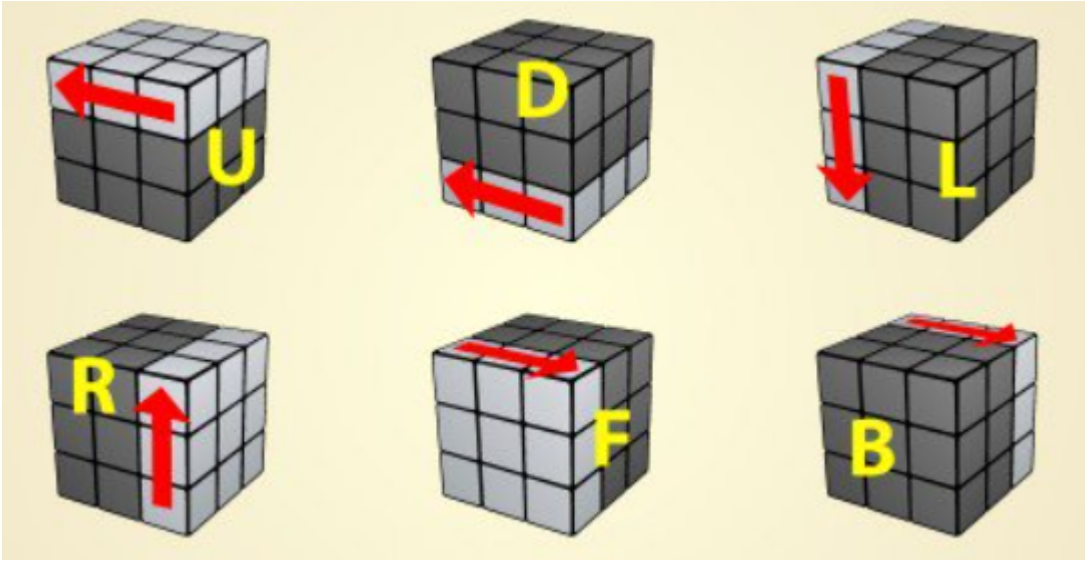
El programa tiene distintas funcionalidades tanto a nivel del proyecto en sí (cubo de Rubik), como también en relación a los conceptos de computación gráfica como son el uso de la cámara, la proyección de la forma, etc.

Movimiento de las caras del cubo

Empecemos por los controles para realizar movimientos sobre el cubo de Rubik, se implementó en total 12 movimientos para poder rotar las caras del cubo, hay rotaciones 6 para las rotaciones horarias, y 6 para las antihorarias (una por cada cara del cubo respectivamente), estas rotaciones se pueden controlar mediante el teclado alfanumérico:

PAD 0: Cara Frontal (front) sentido horario
PAD 1: Cara Frontal (front) sentido antihorario
PAD 2: Cara Derecha (right) sentido horario
PAD 3: Cara Derecha (right) sentido antihorario
PAD 4: Cara Superior (up) sentido horario
PAD 5: Cara Superior (up) sentido antihorario
PAD 6: Cara Trasera (back) sentido horario
PAD 7: Cara Trasera (back) sentido antihorario
PAD 8: Cara Izquierda (left) sentido horario
PAD 9: Cara Izquierda (left) sentido antihorario
PAD +: Cara Abajo (down) sentido horario
PAD -: Cara Abajo (down) sentido antihorario

En la imagen de abajo se muestra las rotaciones que realizará el programa.



Ejecución del *solver* del cubo de Rubik

Para realizar la resolución del cubo simplemente debe apretar la tecla ESPACIO (SPACE) de su ordenador, esto automáticamente solucionará el cubo de Rubik, también se imprimirá en consola la forma inicial del cubo, de cómo fué enviado al solver, para más información vea **[9]**.

Manejo de las *animaciones* para el cubo de Rubik

El manejo de las animaciones para el cubo de Rubik solo consta de dos teclas *B* o *b* para activar la animación *Respiración Continua* y *V*

o *v* para detenerla, en cuanto a la animación *Interpolación de texturas* ésta es continua durante la ejecución del programa.

Manejo de la cámara

El manejo de la cámara se realiza con el *mouse* (para cambiar la orientación y el *zoom*), deslice la rueda del *mouse* hacia arriba para aumentar el *zoom* y hacia abajo para reducirlo, también use las teclas:

- A (moverse a la izquierda).
- W (moverse hacia adelante).
- S (moverse a la derecha).
- D (moverse hacia atrás).

Manejo de la matriz de proyección

El manejo de la matriz de proyección se controla mediante dos teclas, *P* o *p* para cambiar a la proyección en perspectiva y las teclas *O* o *o* para la ortogonal, la rueda del ratón cuando se mueve lo que hace es incrementar el ángulo del *FOV* de la matriz de proyección en perspectiva.

Problemas encontrados en la implementación

El problema principal encontrado en la implementación fué la imposibilidad del mapeo completo de una textura sobre una cara del cubo de Rubik, esto sucede por la forma en que se realiza el modelado del cubo, en este caso yo solo modelo un cubo y uso una variable *uniform* para trasladarlo y dibujarlo nuevamente en pantalla, básicamente el programa solo usa 24 vértices, los cuales son dibujados 26 veces (la cantidad de cubos que componen el cubo de Rubik).

```
class Cube
{
private:
    static const int NFACES = 6;
    void chooseColor(const Shader& program, GLint i);
public:
    char colors[NFACES];

    glm::vec3 pos;
    glm::mat4 model;
    Cube();
    Cube(const Cube& another_cube);
    Cube(glm::mat4 model_, glm::vec3 pos_);
    void draw(const Shader& program);
    ~Cube();
};
```

Como se ve en el código tenemos una clase **Cube** la cual tiene como atributo una matriz 4x4 que es usada como una matriz modelo que será enviada a la variable *uniform* ***model*** en mi *vertex shader*. Para

poder renderizar toda una imagen sobre una cara del cubo de Rubik se necesitaría generar todos los vértices que componen el cubo de Rubik, lo cual requiere una refactorización total del programa, es por eso que debido a la limitación del tiempo no se logró el primer punto a saber *"mostrar el cubo de rubik ordenado con las imágenes de las letras pasadas como textura para cada cara del cubo"*.

Conclusiones

Se ha realizado un aplicativo que utiliza los conceptos básicos en computación gráfica, se ha logrado el desarrollo en su mayoría de los puntos especificados, tanto en la aplicación de los conceptos matemáticos como los aspectos más técnicos del lado de la programación, considero que el trabajo ha sido de mucha ayuda en cuanto me ha llevado a investigar las piezas claves para el desarrollo de programas gráficos.

Referencias

[1] J. Vries, "LearnOpenGL - Hello Triangle", Learnopengl.com, 2021. [Online]. Available: https://learnopengl.com/Getting-started/Hello-Triangle. [Accessed: 05- Dec- 2021].

[2] J. Vries, "LearnOpenGL - Shaders", Learnopengl.com, 2021. [Online]. Available: https://learnopengl.com/Getting-started/Shaders. [Accessed: 05- Dec- 2021].

[3] Vriez, J., 2021. LearnOpenGL - Textures. [online] Learnopengl.com. Available at: <https://learnopengl.com/Getting-started/Textures> [Accessed 5 December 2021].

[4] J. Vries, "LearnOpenGL - Transformations", Learnopengl.com, 2021. [Online]. Available: https://learnopengl.com/Getting-started/Transformations. [Accessed: 05- Dec- 2021].

[5] J. Vries, "LearnOpenGL - Coordinate Systems", Learnopengl.com, 2021. [Online]. Available: https://learnopengl.com/Getting-started/Coordinate-Systems. [Accessed: 05- Dec- 2021].

[6] J. Vries, "LearnOpenGL - Camera", Learnopengl.com, 2021. [Online]. Available: https://learnopengl.com/Getting-started/Camera. [Accessed: 05- Dec- 2021].

[7] S. Ahn, "Quaternion", Songho.ca, 2021. [Online]. Available: http://www.songho.ca/math/quaternion/quaternion.html#reference. [Accessed: 06- Dec- 2021].

[8] S. Ahn, "Quaternion to Rotation Matrix", Songho.ca, 2021. [Online]. Available: http://www.songho.ca/opengl/gl_quaternion.html. [Accessed: 06- Dec- 2021].

[9] R. Morales Pérez, "GitHub - Rubenmp/Rubik: This program solves Rubik's cube 3x3", GitHub, 2021. [Online]. Available: https://github.com/Rubenmp/Rubik. [Accessed: 06- Dec- 2021].