CIENCIA DE LA COMPUTACIÓN

“Proyecto final de programación: Cubo Rubick”

COMPUTACIÓN GRÁFICA

Andrés Cusirramos Márquez Mares

andres.cusirramos@ucsp.edu.pe

Anthony Paolo Fernández Sardón

anthony.fernandez@ucsp.edu.pe

SEMESTRE VII

AÑO 2022

“Los alumnos declaran haber realizado el presente trabajo de acuerdo a las normas de la Universidad Católica San Pablo”

INTRODUCCIÓN DEL PROYECTO

La finalidad del proyecto consiste en la aplicación de conceptos de modelado, renderizado y animación en la elaboración de un cubo de Rubick con la capacidad de moverse por camadas (movimientos horizontales y verticales de los pequeños cubos que lo componen) y que pueda solucionarse automáticamente. Por otro lado, la visualización del mismo se hará mediante una cámara virtual que permita observarlo de diferentes ángulos y que a su vez tenga animaciones para verlo de forma dinámica.

La elaboración del proyecto muestra de forma aplicativa los conceptos aprendidos relacionados al entorno de OpenGL: utilización de shaders, aplicación de colores y texturas, elaboración y utilización de matrices de transformación, perspectiva y modelado en tres dimensiones, además de estructuras de datos y conocimientos de álgebra. Todo ello ha permitido modelar vectores organizados en una estructura que adopten la forma del cubo de Rubick, renderizar tanto sus colores correspondientes a los cubitos como una textura para cada una de las caras, manipular los vectores y transformarlos para darles animación al moverlos en camadas y generar que el cubo se desordene y arme haciendo uso de un algoritmo para dicho fin (algoritmo de Kociemba). Para concluir se agregaron animaciones en la cámara que permita la visualización del cubo dentro del entorno, y para el cubo mismo para hacerlo más dinámico: respiración, donde cada cubo se expande en una determinada dirección dando el efecto de respiración, y desorden, donde cada cubo se separa de la estructura principal y luego se reordena a su posición original.

IMPLEMENTACIÓN DE LA ANIMACIÓN PROPUESTA

1ra Animación: Respiración de cubos

En esta animación tenemos que simular las respiracion del cubo, para esto tenemos que incrementar la distancia de cada cubo respecto a los demás cubitos, para generar una sensación que los cubitos respiran, pero para agregarle valor a esta animación tambien al momento de hacer la separación de los cubitos, haremos que estos incrementen su tamaño uno a uno para así poder tener una animación que de la sensación de que el cubo en conjunto de los cubos más pequeños está respirando.

Para poder hacer que los cubitos se separen entre ellos, fue necesario usar una translación para lo cual se aplicó la función que nos ofrece la librería matemática GLM, en esta función usamos dos parámetros, el primero una matriz identidad y luego accedemos a cada cubito a través de la estructura de Cubo para así llegar a cada posición inicial y multiplicar cada vector por una distancia la cual puede ser positiva o negativa, para así incrementar o decrementar la distancia de los cubitos.

model=glm::translate(glm::mat4(1.0),(MagicCube.littleCubes[i].initialPosition)*distancia);

Para lograr escalar los cubitos mientras estos se trasladan, usamos la función Scale de la librería matemática GLM, donde tenemos una matriz identidad y un vector de escala el cual le sumamos o restamos otro vector para así tener un aumento y decremento de la escala.

model = glm::scale(glm::mat4(1.0), ScaleVector);

Cabe resaltar que estas dos funciones están siendo utilizadas dentro del render loop para así poder generar las animaciones cada vez que los cubitos están siendo renderizados.

2da Animación: Cubos desordenados

Esta animación parte de los pequeños cubos formando el cubo de Rubick. Al momento de activar esta funcionalidad, los cubitos se dispersan y caen a una cierta distancia con un movimiento elíptico. Nuevamente al activar la funcionalidad, los cubitos regresan a su posición original siguiendo el recorrido contrario hasta armar adecuadamente el cubo.

Para lograr esta animación se hizo uso de la fórmula del elipsoide para seguir la trayectoria en el movimiento de los cubitos:

$$\frac{x - xo}{a^2} + \frac{y - yo}{b^2} + \frac{z - zo}{c^2} = 1$$

Donde, (xo, yo, zo) son las coordenadas del centro de la elipsoide en tres dimensiones. Los valores de "a,b,c" corresponden al radio de la elipse en cada una de las dimensiones correspondientes.

Para poder utilizar una variables que no dependa de "x,y,z" al momento de aplicar esta fórmula para variar las posiciones de los vértices se utilizó la fórmula parametrizada del elipsoide:

$$x = xo + a \cdot sen\alpha \cdot sen\theta$$

$$y = yo + b \cdot \ cos\theta$$

$$z = zo + c \cdot \ cos\alpha \cdot sen\theta$$

Donde: $\alpha \ y \ \theta$ corresponden a los ángulos de las coordenadas esféricas.

Debido a que ya se tenía implementada la transformación de traslación de vectores, era necesario adaptar los cálculos para que los vértices sigan la función del elipsoide en cada iteración. Para ello se derivó la fórmula parametrizada de la elipse para que el resultado sirva de parámetro al momento de aplicar la matriz de traslación cada vez que el ángulo θ varíe a lo largo del tiempo:

$$\frac{dx}{d\theta} = (a \cdot sen\alpha \cdot cos\theta)$$

$$\frac{dy}{d\theta} = (-b \cdot sen\theta)$$

$$\frac{dz}{d\theta} = (c \cdot cos\alpha \cdot cos\theta)$$

Para la aplicación de esta fórmula en la animación, los valores de a, c, α son constantes, mientras que el valor de b depende del valor máximo de "y" de los vértices de cada cubito. En cuanto al valor de θ , este varía cada que se aplica la animación.

Cada cubito tiene el centro de su elipsoide al mismo nivel para que al momento de incrementar el valor de θ todos lleguen a una posición igual en el eje "y". Cuando se activa la animación, el ángulo θ aumenta en una unidad hasta el valor de 90 lo que genera que los cubos se desplacen siguiendo la función del elipsoide. Cuando se vuelve a activar la animación, pasa lo contrario y el ángulo θ decrementa su valor hasta 0 ocasionando que los cubitos vuelvan a formar el cubo de Rubick.

FUNCIONALIDADES DEL PROGRAMA

Para las funcionalidades de nuestro cubo Rubick, tenemos diferentes funciones las cuales se ingresan por teclado a través de la función "key_callback" y para capturar la letra que estamos presionando "glfwGetKey".

Para los **Movimientos del Cubo** tenemos las letras R, L, U, D ,F , B. Las cuales realizan los movimientos clásicos del cubo, como por ejemplo al presionar la letra R, se realiza el movimiento de la camada de la derecha y así respectivamente con cada uno de las letras especificadas. Con esto podemos desarmar el cubo haciendo el movimiento de las camadas.

Como **funcionalidades de los movimientos** también tenemos dos funciones, la primera función tenemos al presionar la letra A en la cual se realizarán 20 movimientos de forma aleatoria para así desarmar el cubo y como segunda función tenemos la letra S, la cual activará al Solver de nuestro cubo y se resolverá el cubo con los movimientos dado por el Solver, esta tecla puede ser activada luego de desordenar el cubo manualmente con los movimientos clásicos o con luego de presionar la tecla A que hace movimientos aleatorios.

Como funcionalidades de **movimiento de cámara**, tenemos movimientos manuales para mover la cámara en cada uno de sus ejes X, Y y Z. Para mover la cámara en el eje X tenemos la tecla LEFT, la cual hace una suma 0.3f en x para el vector "cameraPos" y tenemos la tecla RIGHT para realizar el efecto contrario en el eje x. Para el eje Y tenemos las teclas UP y DOWN, UP para aumentar 0.3f a y en el vector "cameraPos" y DOWN la cual realizar el efecto contrario. Y por último tenemos el movimiento en eje Z para los tenemos la tecla PAGE_UP para realizar la suma de 0.3f en z para el vector "cameraPos" y PAGE_DOWN para realizar el efecto contrario.

Para los **movimientos de la cámara automáticos** en los cuales se mueve la cámara en el eje que escojamos y tenemos 4, el primero en x para el cual presionamos la tecla X, el segunda

en y para el cual presionamos la tecla Y, el tercero en Z para el cual presionamos la tecla Z y por último tenemos un movimiento en espiral de la cámara la cual se activa con la letra W ocasionando movimientos elípticos a lo largo del eje y.

Para las **animaciones de los cubos que componen al Rubick**, con el número 1 se activa y desactiva los Cubos Desordenados: al presionar una vez los cubos se dispersan cayendo en el eje "y" con un movimiento elíptico, y al volver a presionar estos se reordenan en la estructura inicial. Con el número 2 y 3 se activa la animación de Respiración donde se expanden y contraen los cubos respectivamente.

PROBLEMAS ENCONTRADOS EN LA IMPLEMENTACIÓN

- Como primer problema en la implementación encontramos complicado la implementación de las texturas a las caras del cubo, dado que teníamos como primera idea poner a cada cubo con una imagen, pero luego nos dimos cuenta que solo teníamos que usar una imagen por cara y así definir a cada cubo la parte de la imagen que estábamos cargando, así que modificados los vértices donde definimos primero los colores y agregamos las coordenadas de las texturas.

- Como segundo problema, tambien relacionado con las texturas, al momento de desarmar el cubo y luego realizar la solución, las texturas del centro no quedaban en su posición correcta. Para solucionar esto tuvimos que establecer la posición inicial de las texturas de los cubitos del centro y así después de terminar de resolver el cubo, hacemos que esas texturas regresen a su posición inicial y así el cubo queda correctamente ordenado.

- Como ultimo problema lo tuvimos en las animaciones dado que para la respiración del cubo tenemos que realizar una escala de los cubitos, pero al realizar la translación los cubos se movían en una sola dirección. Para la animación de desordenar los cubos, tuvimos impedimentos al determinar la fórmula para que los cubos dispersados sigan un patrón, y después el poder aplicar adecuadamente esta fórmula en las transformaciones de los vectores.

CONCLUSIONES

- ✓ El entorno de OpenGL es bastante amigable para poder generar elementos visuales y realizar en ellos transformaciones pudiendo modelar, renderizar y animar objetos en dos y tres dimensiones. El conocimiento de conceptos de álgebra y geometría ayuda bastante a que dicho entorno sea más aprovechable y encontrar soluciones a problemas propios de este ámbito.

- ✓ La utilización de estructuras de datos permite encapsular valores comunes entre los objetos y que sea más sencilla su manipulación, así como la aplicación de funciones.
- ✓ El modelado y renderizado se encuentran bastante relacionados. La disposición de los vectores debe considerar la apariencia que tendrá cada uno de ellos. Los shaders hacen posible que estos dos aspectos se relacionen y puedan ser aplicados.
- ✓ La animación requiere una claridad en la manipulación de los vectores que se desean animar. OpenGL ofrece librerías como GLM con bastantes funciones que ayudan a este fin, aunque de igual forma se pueden aplicar transformaciones elaborando las operaciones con sus matrices correspondientes.

REFERENCIAS

J. De Vries. "Learn OpenGL, extensive tutorial resource for learning Modern OpenGL". Learn OpenGL, extensive tutorial resource for learning Modern OpenGL. https://learnopengl.com/ (accedido el 30 de junio de 2022).

"GitHub - muodov/kociemba: A pure Python and pure C ports of Kociemba's algorithm for solving Rubik's cube". GitHub. https://github.com/muodov/kociemba (accedido el 1 de julio de 2022).

“Elipsoide”. Hmong wiki. https://hmong.es/wiki/Ellipsoid (accedido el 01 de julio de 2022).

“Sistema de coordenadas esféricas”. Hmong wiki. https://hmong.es/wiki/Spherical_coordinates (accedido el 01 de julio de 2022).