Universidad Nacional Autónoma de México.

Facultad de Ingeniería.

Laboratorio de Computación Gráfica e interacción Humano Computadora.

Nombre del alumno: Cervantes Rubí Brandon.

Número de Cuenta: 316136741.

Grupo de Laboratorio: 01.

Manual Técnico del proyecto

Semestre 2022-1.

Fecha de entrega límite: 22/11/2022.

INDICE

Mapa de referencia.

Modelos

Modelado geométrico.

En este tipo de modelado se parte de figuras primitivas a las cuales se les van a aplicar diferentes transformaciones geométricas; escalar, rotar o trasladar según sea el caso para formar una figura compleja a base de figuras simples. Se recomienda que los objetos modelados con esta técnica se encuentren estáticos, de lo contrario se tendrán que aplicar las translaciones pertinentes a cada una de las primitivas que forman parte del modelo.

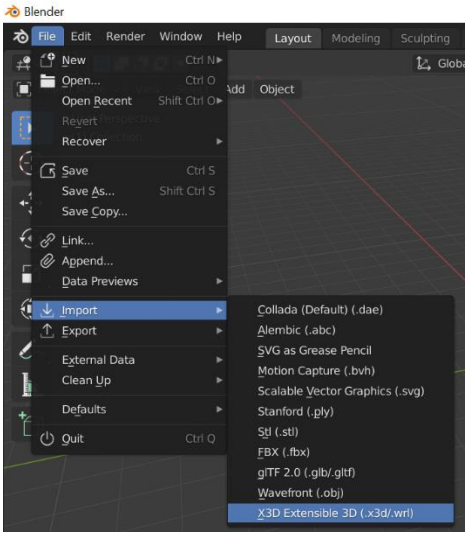
En este proyecto se crearon montículos de nieve esparcidos por el suelo y esferas que sirven de decoración en el árbol de la parte de arriba, adicional a esto se le aplico el material brilloso visto en clase a estas esferas, se tuvo el problema de que se le aplicaba también a todo lo que seguía por los mismos ciclos de reloj y para solucionarlo se opto por hacer un material neutro con valores de cero en la componente especular y el brillo.

Modelado jerárquico.

Este tipo de modelado no se utilizo en el proyecto mas que para unir al avatar, en este caso se tuvieron que separar las geometrías que lo componen y se juntaron con ayuda de traslaciones y de dos matrices auxiliares las cuales heredarán la posición del centro del cuerpo y del punto de articulación para las rodillas, en total se separo el cuerpo en 7 partes contando brazos piernas y cuerpo se darán mayores detalles en el apartado de modelado del avatar.

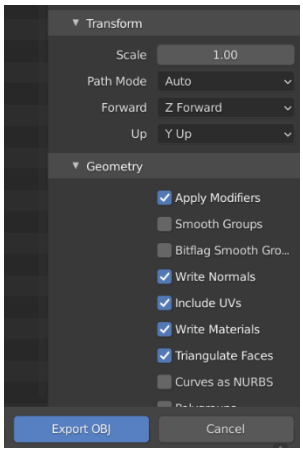
La parte que se tiene que tomar en cuenta para modelar con esta técnica es la posición en la que se encuentre la matriz de modelo considerando las rotaciones y traslaciones que se están haciendo, además de establecer bien cada una de las primitivas.

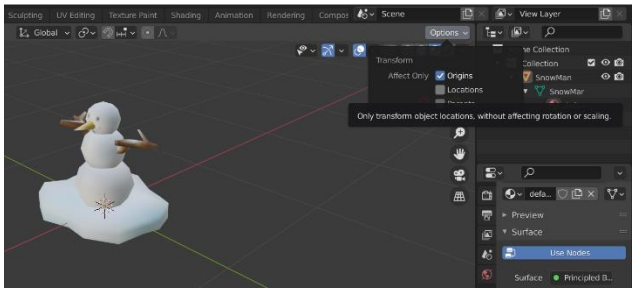
Importación de modelos.



Para la importación de modelos se tienen que buscar modelos de internet los cuales se puedan descargar, la pagina utilizada en este proyecto fue https://sketchfab.com/feed la cual ofrece un buen catálogo de descargables. El segundo paso consiste en importarlo como un modelo dentro de Blender (O algún otro software para el manejo de objetos 3D) cuidando la extensión del objeto que se haya descargado, las más populares son `.obj` y `.fbx` pero Blender ofrece una gran compatibilidad con todas las extensiones mostradas en la imagen a continuación.

Una vez importado nuestro modelo se tiene que buscar que su centro se encuentre en el origen, en caso de no estar centrado primero se deberán mover los ejes al centro del objeto para después mover el objeto junto con los ejes. Para mover solamente los ejes tendremos que seleccionar “Afectar solo los ejes” como se muestra en la imagen. Una vez centrado podemos exportar nuestro objeto como `.obj` con la opción `Export` mostrada en la imagen de arriba cuidando solo un par de detalles, el primero de ellos consiste en que el `vector Forward este en Z positivo` y el segundo que en el apartado de `Geometry` se marquen `caras triangulares`.





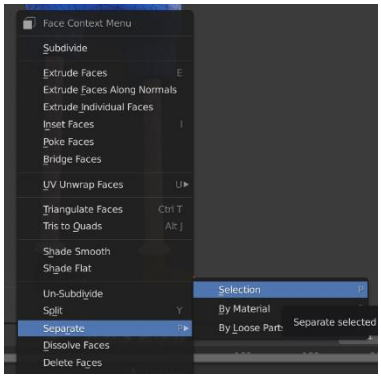
En caso de contener una imagen como textura se va a tener que aplicar dentro del mismo Blender, para ello es necesario que primero se convierta a un formato `tga` para que se integre correctamente con la biblioteca `stb image` que estamos usando para las texturas.

Para renderizar el objeto en el ambiente virtual es necesario declarar primero el objeto de la clase Model usada para los modelos, preparar el modelo asignándole la ruta desde la raíz del proyecto y por último aplicar las operaciones de rotación, traslación y escala con `glm`.

```
Model Terreno_Base;
...
Terreno_Base = Model();
    Terreno_Base.LoadModel("Models/terreno.obj");
...
//Terreno Base
model = glm::translate(model, glm::vec3(0.0f,  0.5f , -1.5f));
model_centro_auto = model;
model = glm::scale(model, glm::vec3(10.0f, 10.0f, 10.0f));
model = glm::rotate(model, 90 * toRadians, glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
Terreno_Base.RenderModel();
```

Modelo del avatar.

Para el avatar se busco un modelo en línea que sirviera como base, seguido de esto se uso Blender para importarlo y comenzar a separar las partes de este con el objetivo de tener las primitivas necesarias para aplicar un modelado Jerárquico en el avatar. Esta separación de partes se hizo con la herramienta `Separate` en el modo de edición, una vez separadas las primitivas del avatar se tienen que importar por separado siguiendo las instrucciones del apartado *importación de modelos* (Centrarlas, formatos, renderizado, etc), en mi caso fueron 7 primitivas al separar piernas, brazos y cuerpo.



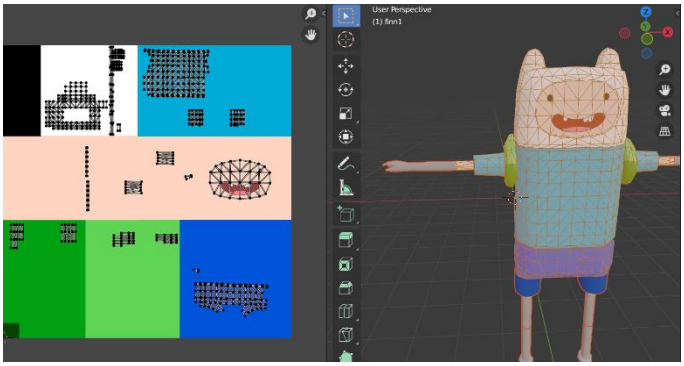
En este modelado se partió desde el cuerpo del avatar puesto que es la parte del cuerpo con más correcciones hacia las otras primitivas. Se deberán de utilizar matrices auxiliares para heredar las posiciones y rotaciones de interés. En este caso se usaron dos matrices auxiliares, la primera de ellas `matriz_cuerpo` para heredar la posición del centro del cuerpo y de esta manera ubicar piernas y manos a su alrededor. La segunda de las matrices auxiliares `matriz_pierna` fue para ubicar la posición donde terminaron las piernas y acomodar los pies (De las rodillas para abajo). En este paso lo mas importante es cuidar bien las posiciones y rotaciones que se están quedando en las matrices auxiliares ya que cada que se cambia de matriz auxiliar se está cambiando el contexto de punto de partida para las nuevas operaciones de traslación, rotación y escala. A continuacion se muestra una poscion del bloque de codigo utilizado para este modelado.

```
//BRAZO PIERNA DERECHA
		model = matriz_cuerpo;
		model = glm::translate(model, glm::vec3(2.5f, 10.92f, -45.23f));
		model = glm::rotate(model, rota_avatar_piernas * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
		matriz_pierna = model;
		model = glm::scale(model, glm::vec3(4.5f, 4.5f, 4.5f));
		glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
		FinnPiernaDerecha.RenderModel();
		//PIE DERECHO
		model = matriz_pierna;
		model = glm::translate(model, glm::vec3(0.0f, -5.0f, 0.25f));
		model = glm::scale(model, glm::vec3(4.5f, 4.5f, 4.5f));
		model = glm::rotate(model, rota_avatar_pies * toRadians, glm::vec3(1.0f, 0.0f, 0.0f));
		glUniformMatrix4fv(uniformModel, 1, GL_FALSE, glm::value_ptr(model));
		FinnPiernaDerechaBa.RenderModel();
```

Texturas

Texturizado del avatar

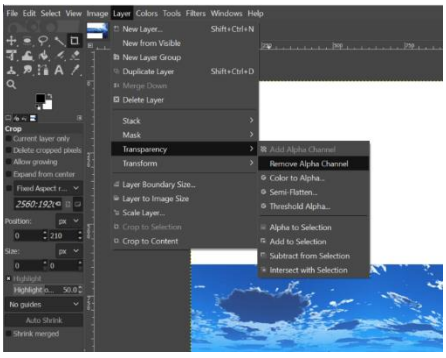
Puesto que el modelo que se importo del avatar fue descargado, solamente se tuvo que cambiar el formato de la imagen a un `.tga` y aplicarse en blender cuidando en la vista UV Editing que se aplicara correctamente.



Skybox

Para los skybox se tuvieron que buscar imágenes de internet las cuales tuvieran el formato de un cubo, una vez con la imagen se tiene que recortar cada uno de los 6 cuadrados que conforman el cubo cuidando en medida de lo posible que los recuadros sean del mismo tamaño y que se recorte sobre el área correcta ya que si nos pasamos demasiado se vera una parte del recuadro que tenga a un lado y en caso de no tener textura a un lado se verá una línea blanca la cual es aún más notoria. Por otro lado, si nos quedamos cortos en el recorte se verá discontinuo el skybox.

Estos recortes se pueden hacer con cualquier programa de edición de imagen, pero es importante que cada una de las caras que compone al recuadro este en formato `.tga` y `que no tengan Alpha channel`. En mi caso el recorte lo realice con GIMP y para remover el Alpha channel se tiene que ir a la pestaña de `Layer` → `Transparency` → `Remove Alpha Channel`. Al final se terminara con 12 imágenes para el Skybox contando el de día y el de noche.



Para agregarlo al ambiente, se tiene que utilizar la clase `Skybox` proporcionada, se tienen que instanciar dos objetos de esta clase (Uno para el de día y otro para el de noche). Se usa un `vector` para almacenar las rutas tomadas desde la raíz del proyecto las cuales van a dar a cada una de las caras del skybox, de igual manera se deben de tener un par de estos vectores ya que al momento de crear las instancias de los objetos skybox estos vectores son los parámetros del método constructor.

Para el intercambio entre los Skybox se tiene que establecer un contador de tiempo en el programa, para este contador se puede ayudar de la función `clock` del modulo `std` de `C++`, y donde cada ciertos segundos se cambie el modelo de skybox que se está mandando a dibujar.

```
std::vector<std::string> skyboxNightFaces;
skyboxNightFaces.push_back("Textures/Skybox/sky-night-right.tga");
skyboxNightFaces.push_back("Textures/Skybox/sky-night-left.tga");
skyboxNightFaces.push_back("Textures/Skybox/sky-night-bottom.tga");
skyboxNightFaces.push_back("Textures/Skybox/sky-night-up.tga");
skyboxNightFaces.push_back("Textures/Skybox/sky-night-front.tga");
skyboxNightFaces.push_back("Textures/Skybox/sky-night-back.tga");
...
skyboxNight = Skybox(skyboxNightFaces);
...
skyboxNight.DrawSkybox(camera.calculateViewMatrix(false), projection);
```

Animación

En todas las animaciones se necesitará una tecla de control para iniciar o pausar la animación. Para implementar esta tecla de control es necesario agregar una variable a la clase Window ya que esta es la que maneja el control en el teclado con glfw y con una condicional cambiar el valor de esta variable. Adicional a esto se necesitará un método get de la variable para poder mandarla a llamar desde el main del programa. A continuación, se muestra un ejemplo

```
//Window.cpp
    if (key == GLFW_KEY_K)
    {
        theWindow->mueve_munecos = 1.0;
    }

    //Window.h
    GLfloat mueve_munecos = 0.0f;
    GLfloat getMunecos() { return mueve_munecos; }

    //main
    if (mainWindow.getMunecos() == 1) {
        ...
    }
```

También se recomienda tener una tecla para poner todas estas variables en cero y de esta manera pausar las animaciones según se requieran.

Animación básica

Para la animación básica se consideraron desplazamientos y rotaciones lineales sobre cada eje las cuales se van a estar actualizando cada ciclo de reloj en base a condicionales y banderas, se tiene dos animaciones básicas una que describe el movimiento de un personaje caminando obre el agua y la segunda una animación de baile para los muñecos de nieve. Por ejemplo, la animación de los muñecos de nieve consta de un movimiento constante al rededor del eje 'X' incrementando y decrementando en 6 unidades hacia adelante y hacia atrás, esto con ayuda de una bandera llamada `mueve_muneco` la cual al momento de avanzar hasta el tope positivo (3 unidades) se pone en alto y cuando alcanza el tope negativo (-3 unidades) se pone en bajo marcando el cambio entre los incrementos y los decrementos.

Además, para las rotaciones se implementa algo similar solo que en este caso el tope va a ser el ángulo en grados que se va a rotar, por ejemplo, hacen rotaciones de lado a lado o al frente y atrás las cuales se pueden describir con rotaciones pequeñas yendo de -20 a 20 grados o tal vez un poco mas y rotaciones completas de 360 grados las cuales se ejecutan cada cierto numero de pasos adelante atrás, los pasos los podemos contar con una variable entera para controlar este giro completo. A continuación, se muestra el bloque de código como ejemplo.

```
//Animacion en los muñecos de nieve
        if (mainWindow.getMunecos() == 1) {
            inicio_munecos = true;
            if (adelante_munecos) { //Movimiento
                if (mueve_x_munecos < 3) {
                    mueve_x_munecos += offset_mueve_munecos * deltaTime;
                }
                else {
                    adelante_munecos = false;
                }
            }
            else {
                if (mueve_x_munecos > -3) {
                    mueve_x_munecos -= offset_mueve_munecos * deltaTime;
                }
                else {
                    adelante_munecos = true;
                }
            }
            if (pasos < 4) {//Rotacion
                if (angulo_z_munecos_ok) {
                    rota_z_munecos += offset_rota_munecos * deltaTime;
                    if (rota_z_munecos > 30) {
                        angulo_z_munecos_ok = false;
                        pasos += 1;
                    }
                }
                else {
                    rota_z_munecos -= offset_rota_munecos * deltaTime;
                    if (rota_z_munecos < -5) {
                        angulo_z_munecos_ok = true;
                    }
                }
            }
```

```
            else {
                //Giro completo
                if (pasos == 4) {
                    rota_y_munecos += offset_rota_munecos * 2 * deltaTime;
                    if (rota_y_munecos > 360) {
                        rota_y_munecos = 0;
                        pasos = 0;
                    }
                }
            }
        }
```

Se usaron estas técnicas de animación para el personaje que camina sobre el agua el cual toma la trayectoria de un polígono de 4 lados contenido en el agua controlando movimiento y una ligera rotación con una sentencia case, donde se tiene un caso para cada una de las rectas que recorre el personaje ya que los desplazamientos cambian de dirección con cada caso, se hace lo mismo con el personaje que no puede caminar sobre el agua y se va hundiendo. Lo único que se debe que tener presente en este tipo de animación es el estado de cada una de las banderas o las condiciones que afectan el flujo de la animación.

Por último, se recomienda manejar `Offsets` de movimiento y rotación para controlar la velocidad de cada uno de los incrementos, así como una variable que almacene el cambio en el tiempo similar a `deltaTime` para que sin importar las velocidades de reloj que alcance el procesador de las computadoras la animación se ejecute a la misma velocidad.

Animación compleja

Para este tipo de animación los movimientos ahora no serán descritos solo por incrementos lineales, sino que se usara una función matemática para describirlos. Algunos ejemplos son; Senos, cosenos, parábolas, elipses, etc. Cualquier función que nos describa un movimiento el cual se adecue al movimiento que queremos en la animación.

Como los incrementos son en cada una de las variables se recomienda que se usen las ecuaciones paramétricas de las funciones para que ambas estén en función de una sola variable la cual va a ser la que se esté incrementando, dependiendo del offset que declaremos.

En la animación de Rei bailando sobre la pista, se describe una trayectoria en forma de Lemniscata que también se puede ver como un numero ocho acostado o el símbolo de infinito para este movimiento se utilizaron las ecuaciones paramétricas de la curva, además de rotaciones en los ejes ‘Y’ y ‘Z’ que se controlan en base a condicionales. A continuación, se muestra el bloque de código como ejemplo.

```
//Animacion compleja para rei
    if (mainWindow.getRei() == 1.0f) {
        rota_y_rei += offset_rota_rei * 10 * deltaTime;
        if (angulo_z_rei_ok) {
            rota_z_rei += offset_rota_rei * deltaTime;
            if (rota_z_rei > 35) {
                angulo_z_rei_ok = false;
            }
        }
        else {
```

```
            rota_z_rei -= offset_rota_rei * deltaTime;
            if (rota_z_rei < -35) {
                angulo_z_rei_ok = true;
            }
        }
            if (angulo_x_rei_ok) {
            rota_x_rei += offset_rota_rei * deltaTime;
            if (rota_x_rei > 60) {
                angulo_x_rei_ok = false;
            }
        }
        else {
            rota_x_rei -= offset_rota_rei * deltaTime;
            if (rota_x_rei < -60) {
                angulo_x_rei_ok = true;
            }
        }

        num_lem_rei = 15 * sqrt(2) * cos(angulo_lemn);
        den_lem_rei = pow(sin(angulo_lemn), 2) + 1;
        mueve_x_rei = num_lem_rei / den_lem_rei;

        num_lem_rei = 15 * sqrt(2) * cos(angulo_lemn) * sin(angulo_lemn);
        mueve_z_rei = (num_lem_rei / den_lem_rei);
           angulo_lemn += 0.01 * deltaTime;
    }
```

Las otras animaciones descritas con esta técnica es el movimiento continuo de las nubes con senos y cosenos sobre el eje vertical y horizontal además de una rotación alrededor del escenario, la roca que con la tecla M libera el easter egg y aplasta a un muñeco de nieve, esta roca cada vez va girando más rápido y describe un movimiento parabólico en su caída.

Lo importante en esta técnica es buscar y adaptar las funciones las cuales nos describan el movimiento que se requiere.

Animación por keyframes

Esta técnica de animación no se implemento por motivos de tiempo pero consiste en establecer un conjunto de tantos frames como sea posible y donde cada frame contiene información de movimiento del objeto.

Animación del avatar

Para la animación del avatar lo que use fue animación básica y se dividió en dos partes, la traslación y la rotación de sus partes. Para la traslación completa del avatar solamente lo ligue a la posición de la cámara adelantándolo y bajándolo un poco para hacer el efecto de la cámara en tercera persona, tomando en cuenta que la clase Camera tiene su método get para la variable de la posición donde cada una de las componentes de la posición de la cámara se le aplican a la traslación del avatar.

Para las rotaciones se tomaron en cuenta dos, la del avatar completo sobre el eje vertical al momento de hacer los giros de cámara y la rotación de cada una de las partes ligadas al cuerpo. La primera de ellas se hizo adelantando el modelo del cuerpo (Sin brazos ni piernas) en blender para que en la rotación se apreciara un desplazamiento sobre los ejes 'X' y 'Z', además el cambio en el ángulo se hizo con la posición de la cámara en el eje 'Z'. La segunda rotación tiene que ver con la que esta asociada a cada una de las primitivas que forman el cuerpo del avatar, en el caso de los brazos es una sobre el eje 'X' para hacer el efecto del braceo los cuales van a ir en un ángulo de 75 ° a -75 ° cuando este en alguna cámara ligada al piso y de -30° a 30° cuando este en el modo de la cámara libre, en las piernas la rotación es robre el eje 'Z' con un angulo pequeño 30 ° y en los pies (De las rodillas para abajo) también en el eje 'Z' pero un poco mas marcada o veloz para hacer el efecto de que el avatar está caminando.

Estas animaciones se accionan con las teclas A, S, D, y W las cuales están asociadas también al movimiento de la cámara. A continuación, se muestra el bloque de código correspondiente a la animación del avatar.

```
//Animacion del avatar
   if (mainWindow.getMueveW() || mainWindow.getMueveS()
       || mainWindow.getMueveA() || mainWindow.getMueveD()) {
       if (mainWindow.getTipoCamara() != 5) { //Esta sobre algun plano
           //Brazos
           if (angulo_z_avatar_ok) {
               rota_avatar_brazos += 4.5 * deltaTime;
               if (rota_avatar_brazos > 75) {
                   angulo_z_avatar_ok = false;
               }
           }
           else {
               rota_avatar_brazos -= 4.5 * deltaTime;
               if (rota_avatar_brazos < -75) {
                   angulo_z_avatar_ok = true;
               }
           }
               //Piernas
           if (angulo_x_avatar_ok) {
               rota_avatar_piernas += 2.5 * deltaTime;
               if (rota_avatar_piernas > 30) {
                   angulo_x_avatar_ok = false;
               }
           }
           else {
               rota_avatar_piernas -= 2.5 * deltaTime;
               if (rota_avatar_piernas < -30) {
                   angulo_x_avatar_ok = true;
               }
           }
               //Pies
           if (angulo_x_pies_ok) {
               rota_avatar_pies += 2.5 * deltaTime;
               if (rota_avatar_pies > 0) {
                   angulo_x_pies_ok = false;
               }
           }
           else {
               rota_avatar_pies -= 2.5 * deltaTime;
               if (rota_avatar_pies < -25) {
                   angulo_x_pies_ok = true;
               }
           }
           }
```

```
        else { //Si esta volando
            if (angulo_z_avatar_ok) {
                rota_avatar_brazos += 3 * deltaTime;
                if (rota_avatar_brazos > 30) {
                    angulo_z_avatar_ok = false;
                }
            }
            else {
                rota_avatar_brazos -= 3 * deltaTime;
                if (rota_avatar_brazos < -30) {
                    angulo_z_avatar_ok = true;
                }
            }
        }
    }
```

Cámaras

En el escenario se tienen 5 cámaras, tres de ellas están ligadas a planos 'XZ' para cuando el avatar se este desplazando en sorbe el suelo, una cámara será aérea la cual nos proporcionará una vista desde arriba en el escenario y la última cámara será libre pudiéndose desplazar por cualquier eje del escenario.

El cambio de las cámaras esta descrito por una sentencia `switch case` para cambiar entre el estado de cada una de las cámaras, en esta sentencia case se guardan las posiciones anteriores en 4 vectores de tres componentes para cuando regrese cambie entre las cámaras retome en donde se quedó y además se avanza de posición siempre bloqueando algún eje de las cámaras a excepción de la cámara libre.

Para el proceso de guardado en cada ciclo de reloj se almacena la posición de la cámara en ese momento dependiendo del tipo de cámara que se tenga, es la cámara aérea y son tres regiones para las cámaras ligadas al piso; entrada, nieve y árbol. Para todos los casos solamente se guarda posición en el plano 'XZ' ya que la posición en 'Y' siempre será constante y variará dependiendo del tipo de cámara.

```
    switch (mainWindow.getTipoCamara()) //Guardado de posiciones anteriores
    {
    case 1:
        cam_entrada_anterior.x = camera.getCameraPosition().x;
        cam_entrada_anterior.z = camera.getCameraPosition().z;
        break;
    case 2:
        cam_nieve_anterior.x = camera.getCameraPosition().x;
        cam_nieve_anterior.z = camera.getCameraPosition().z;
        break;
    case 3:
        cam_arbol_anterior.x = camera.getCameraPosition().x;
        cam_arbol_anterior.z = camera.getCameraPosition().z;
        break;
    case 4:
        cam_aerea_anterior.x = camera.getCameraPosition().x;
        cam_aerea_anterior.z = camera.getCameraPosition().z;
        break;
    default:
        break;
    }
```

Los cambios en el tipo de Cámara se darán por teclado cambiando el valor de una variable controlada en la clase `Window.cpp` similar a como se tiene en el apartado de las animaciones teniendo por default el tipo de cámara 1 que es en el plano de la entrada.

Cámara al suelo

En el escenario se tienen tres cámaras ligadas al plano 'XZ'; entrada, nieve y árbol a las cuales les corresponden los tipos 1, 2 y 3 de las cámaras respectivamente. En cada ciclo de reloj se evalúa el tipo de cámara actual y dentro de cada uno de los primeros tres casos se tiene una condicional la cual evalúa si es la primera vez que se entró, si lo es entonces toma los valores de posición actual y anterior en los `spawns` de cada zona, estos valores se consiguieron mandando a imprimir la posición de la cámara en consola y acercándose al lugar donde se requería. En dado caso de que no sea la primera vez que entro, entonces se le van a asignar los valores de posición del ciclo de reloj anterior como punto de partida.

Se tiene una bandera para que cada que se cambia de cámara se detecte este primer cambio, si es el caso se toma como punto de partida los valores del respaldo para comenzar a desplazarse desde ahí, en caso de detectar un desplazamiento y no detectar el cambio de cámara entonces la posición avanzara con normalidad.

```
    Case 1: //Entrada
        if (inicio_entrada) { //Se toma la inicial
            camara_x_inicial = cam_entrada_anterior.x = -177.0f;
            camara_z_inicial = cam_entrada_anterior.x = 302.0f;
            inicio_entrada = false;
            rota_avatar_brazos = 0.0f;
        }
        else {  //Se toma la anterior
            camara_x_inicial = cam_entrada_anterior.x;
            camara_z_inicial = cam_entrada_anterior.z;
        }
        nueva_y_camara = -55.0f;
        break;
    case 2: //Nieve
        if (inicio_nieve) { //Se toma la inicial
            camara_x_inicial = cam_nieve_anterior.x = 116.0f;
            camara_z_inicial = cam_nieve_anterior.x = 413.0f;
            inicio_nieve = false;
            rota_avatar_brazos = 0.0f;
        }
        else {  //Se toma la anterior
            camara_x_inicial = cam_nieve_anterior.x;
            camara_z_inicial = cam_nieve_anterior.z;
        }
        nueva_y_camara = -75.0f;
        break;
    case 3: //Arbol
        if (inicio_arbol) { //Se toma la inicial
            camara_x_inicial = cam_arbol_anterior.x = 18.0f;
```

```
            camara_z_inicial = cam_arbol_anterior.z = 186.0f;
            inicio_arbol = false;
            rota_avatar_brazos = 0.0f;
        }
        else {  //Se toma la anterior
            camara_x_inicial = cam_arbol_anterior.x;
            camara_z_inicial = cam_arbol_anterior.z;
        }
        nueva_y_camara = 35.0f;
        break;
        ...
if (mainWindow.getCameraFirstChange()) {
            nueva_x_camara = camara_x_inicial;
            nueva_z_camara = camara_z_inicial;
            mainWindow.setCameraFirstChange(false);
        }
        else {
            nueva_x_camara = camera.getCameraPosition().x;
            nueva_z_camara = camera.getCameraPosition().z;
            //Respaldo de camaras
        ...
```

En este tipo de cámaras el principal factor que las diferencia son su posición en 'Y', se tiene que están ligadas a -55, -75 y 35 unidades, además de sus posiciones iniciales en los otros ejes.

Cámara aérea

Para la cámara aérea se tiene un sistema de almacenamiento de posiciones basado en vectores que guardas la posición del ciclo de reloj anterior. También se va a estar moviendo sobre 'XZ' pero en esta ocasión su ubicación en 'Y' va a ser muy grande (180 unidades).

Se hizo una modificación a la clase camera cuando llama al método `calculateViewMatrix` que es el que se encarga de controlar hacia donde y como mira la cámara con `lookAt`. Pues en esta parte es importante que el vector `view` siempre este mirando en el eje 'Y' negativo y para las otras dos componentes dependerá de la posición de la cámara para que el vector de vista se desplace junto con la cámara. Además, el vector Up debe de quedar apuntando hacia 'X' positivo en todo momento. Como este comportamiento solo lo queremos para este tipo de cámara se implementó una bandera que evaluara si se esta llamando a la cámara aérea o no y dependiendo de ello calcule `lookAt` de la cámara como lo hacíamos en las cámaras al suelo, es decir dinámicamente o lo bloque para la cámara aérea.

El método `calculateViewMatrix` para esta cámara quedo de la siguiente manera, donde como parámetro ca a true si la cámara a visualizar es la cámara aérea o a false si la cámara es libre o ligada al suelo.

```
glm::mat4 Camera::calculateViewMatrix(bool aerea)
{
    if (aerea) {
        return glm::lookAt(position, glm::vec3(position.x, -1.0f, position.z),
glm::vec3(1.0f, 0.0f, 0.0f));
    }
    else {
        return glm::lookAt(position, position + front, up);
    }

}
```

Luces

En el escenario se declararon los tres tipos de luces vistas en clase; ambiental (Dos colores), spotlight (Moviéndose) y puntuales las cuales encienden y apagan. Los detalles se darán a continuación.

Luces ambientales

Este tipo de luces es la que ilumina al escenario y va a estar cambiando acorde al ciclo de día y noche, en este caso implemente un método `setColor` en la sueprclase Light ya que se usó también el show de luces spotlights, el cual le cambiara el color a este tipo de luz, de esta manera en el día se tiene la luz blanca que ilumina de buena manera todo el escenario con componentes RGB de (1.0, 1.0, 1.0) y al momento de que se llega al tiempo de cambiar a la noche en la parte donde se dibuja el skybox de noche entonces se cambia el color con el método `setColor` a uno azul con componentes de color (0.337, 0.490, 0.623). A continuación, se muestra el bloque de código donde se hace el cambio.

```
if (duration <= 10) { //Dia
        mainLight.setLight(glm::vec3(1.0f, 1.0f, 1.0f));
        if (mainWindow.getTipoCamara() != 4) {
            skyboxDay.DrawSkybox(camera.calculateViewMatrix(false), projection);
        }
        else { //Camara aerea
            skyboxDay.DrawSkybox(camera.calculateViewMatrix(true), projection);
        }
        dia = true;
        nightLights = 2;
    }
    else { //Noche
        mainLight.setLight(glm::vec3(0.3373f, 0.4902f, 0.6235f)); //Color de luz
ambiental noche
        if (mainWindow.getTipoCamara() != 4) {
            skyboxNight.DrawSkybox(camera.calculateViewMatrix(false), projection);
        }
        else { //Camara aerea
            skyboxNight.DrawSkybox(camera.calculateViewMatrix(true), projection);
        }
        dia = false;
        nightLights = 0;
        if (duration >= 20) { //Regresar al dia
            start = std::clock();
        }
    }
```

Luces spotlight

Con este tipo de luces se hizo el espectáculo de luces que esta en la pista de baile, para hacerlo se declaro que en todo momento estas luces apuntaran hacia el eje 'Y' negativo, y se fue modificando la posición de la luz con el método `setPos` de estas luces. Para la primera de ellas se describe una circunferencia pequeña, para la segunda de ellas una elipse que abarca una sección de la pista de baile. Y la ultima luz `spotlight` va modificando sus componentes de posición describiendo una línea diagonal la cual va aumentando su velocidad para hacer el efecto, este cambio en las componentes de posición se hizo segun condicionales similar a la animación básica.

Además, con el método `setColor` descrito en las luces ambientales se cambia el color de cada una de las luces cada 0.5 segundos que dure la animación con un mecanismo de control del tiempo similar al que se implementó en el ciclo de día y noche, donde cada que se alcance cierto tiempo se cambia el color y se aumenta el tiempo que se tiene que alcanzar para el siguiente cambio. A continuación, se muestra el bloque que describe este cambio en la posición de las luces.

```
//Show de luces spotlight
        if (mainWindow.getBaile() == 1.0) {
            if (inicia_baile) {
                start_baile = std::clock();
                inicia_baile = false;
            }
            tiempo_baile = (std::clock() - start_baile) / (double)CLOCKS_PER_SEC;
            printf("Tiempo: %f\n", tiempo_baile);

            if (tiempo_baile > edge_baile) {
                for (int i = 0; i < 3; i++) {
                    color_pista_x = (float)rand() / (float)RAND_MAX;
                    color_pista_y = (float)rand() / (float)RAND_MAX;
                    color_pista_z = (float)rand() / (float)RAND_MAX;
                    spotLights[i].SetColor(glm::vec3(color_pista_x, color_pista_y,
color_pista_z));
                }
                edge_baile += 0.5;
            }

            //Movimiento de la primera luz

            pos_luz_gamma_x = 4 * sin(angulo_luz_gamma * toRadians) - 19.0;
            pos_luz_gamma_z = 7 * cos(angulo_luz_gamma * toRadians) + 84.0;
            angulo_luz_gamma += 35 * deltaTime;
            spotLights[1].SetPos(glm::vec3(pos_luz_gamma_x, 60.0f, pos_luz_gamma_z));
```

```
            pos_luz_alpha_x = 20 * sin(angulo_luz_alpha * toRadians) + 3.0;
            pos_luz_alpha_z = 5 * cos(angulo_luz_alpha * toRadians) + 130.0;
            angulo_luz_alpha += 25 * deltaTime;
            spotLights[0].SetPos(glm::vec3(pos_luz_alpha_x, 60.0f, pos_luz_alpha_z));

            pos_luz_beta_x = 5 * sin(angulo_luz_beta * toRadians) + 21.0;
            pos_luz_beta_z = 5 * cos(angulo_luz_beta * toRadians) + 94.0;
            angulo_luz_beta += 10 * deltaTime;

             //Movimiento de la tercera luz
            if (sube) {
                pos_luz_beta_x += (0.5 + offset_luz_vertical) * deltaTime;
                offset_luz_vertical += 0.5;
                if (pos_luz_beta_x > 20.0) {
                    sube = false;
                    offset_luz_vertical = 0.5;
                }
            }
            else {
                pos_luz_beta_x -= (0.5 + offset_luz_vertical) * deltaTime;
                offset_luz_vertical += 0.5;
                if (pos_luz_beta_x < -26.0) {
                    sube = true;
                    offset_luz_vertical = 0.5;
                }
            }

            if (derecha) {
                pos_luz_beta_z += (0.5 + offset_luz_vertical) * deltaTime;
                offset_luz_vertical += 0.5;
                if (pos_luz_beta_z > 138.0) {
                    derecha = false;
                    offset_luz_vertical = 0.5;
                }
            }
            else {
                pos_luz_beta_z -= (0.5 + offset_luz_vertical) * deltaTime;
                offset_luz_vertical += 0.5;
                if (pos_luz_beta_z < 80.0) {
                    derecha = true;
                    offset_luz_vertical = 0.5;
                }
            }

            spotLights[2].SetPos(glm::vec3(pos_luz_beta_x, 60.0f, pos_luz_beta_z));


        }
        else {
            tiempo_baile = 0.0;
            start_baile = 0.0;
            inicia_baile = true;
            edge_baile = 0.5;
        }
```

Se implementa un ciclo repetitivo el cual se encarga de cambiar el color de cada una de las luces por eso es que se repite tres veces, en cada iteración se obtienen tres números aleatorios del 1 a cero los cuales corresponden a cada una de las componentes de color de las cuales se va a pintar la luz.

Luces puntuales

En el caso de las luces puntuales se implementaron tres de ellas, dos se van a encender automáticamente cuando se haga de noche con una bandera que se pone en alto, y la tercera de ellas se enciende y apaga con teclado implementando dos variables en la clase `Window`, una para encender y la otra para apagar la luz.

Sin embargo, la lógica cambia cuando es de noche o de día, puesto que para encenderlas y apagarlas se decrementa el contador que se tiene para las luces puntuales de tal manera que la luz no se mande al `shader`. Cuando es de día se necesita que la luz que no se va a mandar este en la primera posición del arreglo puesto que las primeras dos definitivamente no se van a mandar, y cuando sea de noche se necesita que esta luz que se enciende y apaga necesita estar al final del arreglo puesto que las otras dos luces siempre se van a mandar. Entonces para implementarlo se tienen dos arreglos de luces `spotlight` y dependiendo si es de día o de noche se manda a llamar el que tiene la luz que enciende por teclado hasta el inicio del teclado o hasta el final del teclado.

Para implementar el apagado automático de las luces dependiendo del ciclo de día y de noche, se tiene un contador de luces de noche el cual cambia a cero cuando es de noche de tal manera que si se manden al `shader` y tiene valor de 2 cuando es de día restándole estas unidades al contador de luces que se va a mandar al `shader` junto con su arreglo.

```
if (dia) { //Luz que enciende y apaga
        if (mainWindow.getapaga_luz_pinata()) {
            shaderList[0].SetPointLights(pointLightsNight, pointLightCount -
nightLights - 1);
        }
        else {
            shaderList[0].SetPointLights(pointLightsNight, pointLightCount -
nightLights);
        }
    }
    else {
        if (mainWindow.getapaga_luz_pinata()) {
            shaderList[0].SetPointLights(pointLights, pointLightCount - nightLights -
1);
        }
        else {
            shaderList[0].SetPointLights(pointLights, pointLightCount - nightLights);
        }
    }
```

Referencias

Modelos.

"Knuckes Meme" (https://skfb.ly/6vHCR) by xxxhacker573xxx is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).

"Cyberpunk Room Flooring" (https://skfb.ly/6XVFI) by riach is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).

"Serene Lake" (https://skfb.ly/6Wvsn) by Julia Dzindzio is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).

"Oozora Subaru- Shuba Shuba Rigged" (https://skfb.ly/onwAB) by zackw is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).

"Wataten - Bang meme" (https://skfb.ly/o6ntE) by hogi111222 is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).

"Knuckes Meme" (https://skfb.ly/6vHCR) by xxxhacker573xxx is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).

"It's just Bernie being Bernie" (https://skfb.ly/6YGSL) by James Bayer is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).

"Finn the Human (Adventure Time ^__^)" (https://skfb.ly/6UEXA) by Snicks is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).

"Adventure Time drafts" (https://skfb.ly/6RunZ) by ninel_frolova is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).

"Among Us Killed Character" (https://skfb.ly/6UZFw) by oriolfrancin99 is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).

"Cheems" (https://skfb.ly/ooCIP) by feverpepper is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).

"Rei Ayanami Plushie" (https://skfb.ly/onVKF) by timeforrick is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).

"Marceline the Vampire Queen" (https://skfb.ly/68YqT) by snorriholm is licensed under Creative Commons Attribution (http://creativecommons.org/licenses/by/4.0/).

Skybox.

https://images.gamebanana.com/img/ss/mods/5ce4681ff32f8.webp

https://images.gamebanana.com/img/ss/mods/5cdfe4920f8a4.webp

Otros.

Home. (2017). Opengl-Tutorial. Recuperado 2021, de http://www.opengl-tutorial.org/es/

LearnOpenGL - Shaders. (2018). LearnOpenGl. Recuperado 2021, de https://learnopengl.com/Getting-started/Shaders