Universidad Nacional Autónoma de México

Facultad de Ingeniería

Laboratorio de Computación Gráfica e

Interacción Humano Computadora

Manual Técnico

Profesor: Ing. Carlos Aldair Roman Balbuena

Alumno: Chávez Sánchez Juan Daniel

Número de cuenta: 316350866

Grupo de laboratorio: 9

Grupo de teoría: 2

Semestre 2022-2

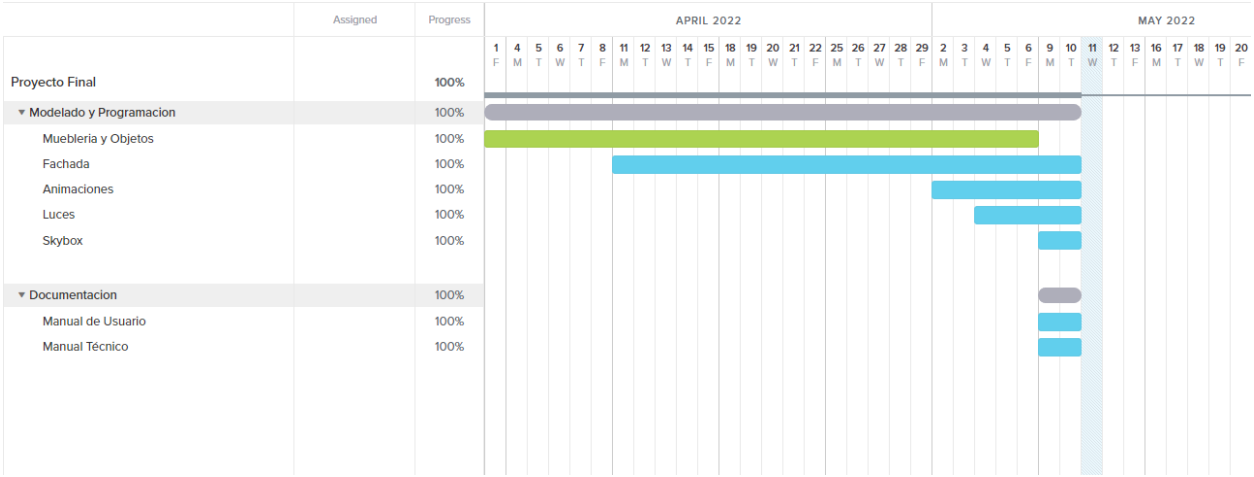
11 de mayo del 2022, Ciudad Universitaria, CDMX.

Contenido

Objetivos

Con el siguiente documento se intentará explicar la forma de trabajo, las herramientas empleadas, la programación y cálculos que fueron utilizados a lo largo de la realización del proyecto final con la finalidad de que el usuario pueda conocer más a profundidad el software.

Diagrama de Gantt



Alcance del Proyecto

El proyecto final de la materia “Computación Gráfica e Interacción Humano Computadora” tiene como público final al profesor de la misma, el Ing. Carlos Aldair Roman Balbuena así como a todos aquellos que tengan acceso al software realizado con la finalidad de comprobar únicamente los conocimientos alcanzados en el curso limitándose así al uso nativo meramente conceptual de OpenGL, modelado, texturizado y exportación de los modelos realizados en formato OBJ así como su importación, muestra y animación haciendo uso de Visual Studio 2022.

Limitantes

El proyecto final presentado únicamente puede ser visto en sistemas operativos Windows y con los requisitos mínimos planteados en el manual de usuario. Así mismo, dentro del proyecto, las acciones permitidas haciendo uso de las teclas ahí establecidas para poder visualizar las animaciones son las únicas acciones permitidas fuera del control de la cámara en primera persona.

Documentación del código

Para la documentación del código se tiene que tomar en cuenta que para este proyecto se decidió representar un centro *pokemón* del juego en sus versiones *Omega Ruby* y *Alpha Saphire* siendo así que todos los modelos correspondientes a los muebles y la fachada debieron haber cumplido con esta característica. Además, todas y cada una de las animaciones debe corresponder a una animación vista dentro del mismo juego.

El software contiene 5 animaciones, 3 sencillas correspondientes a los modelados de las pokebolas, la televisión y el cartel en LED's que se muestra girando en la entrega vista para *Omega Ruby*, y dos complejas correspodientes a las burbujas que salen de los tubos a los laterales de la estructura y a la apertura de un botiquín que saca un spray de primeros auxilios y lo pone sobre la mesa.

Para comenzar entonces, se mostrará la forma en que se importaron los modelos dentro del código los cuales se encuentran con sus respectivas texturas dentro de la carpeta "Models" que se encuentra dentro del mismo proyecto de Visual Studio:

```
//Inclusion de modelos del proyecto
Model pokeFachada((char*)"Models/proyectoPokemon/pokeFachada.obj");
Model pokeSillon((char*)"Models/proyectoPokemon/pokeSillon.obj");
Model pokeMesa((char*)"Models/proyectoPokemon/pokeMesa.obj");
Model pokeLibrero((char*)"Models/proyectoPokemon/pokeLibrero.obj");
Model pokeMaquina((char*)"Models/proyectoPokemon/pokeMaquina.obj");
Model pokeLetrero((char*)"Models/proyectoPokemon/pokeLetrero.obj");
Model pokeBancoAmarillo((char*)"Models/proyectoPokemon/pokeBancoAmarillo.obj");
Model pokeBancoRosa((char*)"Models/proyectoPokemon/pokeBancoRosa.obj");
Model pokeMaceta((char*)"Models/proyectoPokemon/pokeMaceta.obj");
Model pokeBanda((char*)"Models/proyectoPokemon/pokeBanda.obj");
Model pokeTele((char*)"Models/proyectoPokemon/pokeTele.obj");
Model basePokeBola((char*)"Models/proyectoPokemon/basePokeBola.obj");
Model tapaPokeBola((char*)"Models/proyectoPokemon/tapaPokeBola.obj");
Model basePokeBotiquin((char*)"Models/proyectoPokemon/basePokeBotiquin.obj");
Model tapaPokeBotiquin((char*)"Models/proyectoPokemon/tapaPokeBotiquin.obj");
Model pokeSpray((char*)"Models/proyectoPokemon/pokeSpray.obj");
Model pokeBurbujas((char*)"Models/proyectoPokemon/pokeBurbujas.obj");
Model pokeVidrio((char*)"Models/proyectoPokemon/pokeVidrio.obj");
Model pokeTubos((char*)"Models/proyectoPokemon/pokeTubos.obj");
```

Figura 1. Importación de modelos dentro de Visual Studio.

Es bajo su dirección dentro del proyecto cómo es que Visual Studio 2022 encuentra los modelos y logra importar el OBJ dentro del trabajo. Une vez hecho esto se dibujan cada uno de los modelos dentro de nuestra escena que va a representar el trabajo. En esta ocasión el presente documento hará muestra de aquellos objetos que tienen una animación ya que la importación en los demás modelos y su dibujado es exactamente igual que a estos últimos.

Animaciones Sencillas

Movimiento de la televisión

```
//Tele
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-5.7, 4.2, -0.2));
model = glm::rotate(model, glm::radians(teleRot), glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
pokeTele.Draw(lightingShader);
```

Figura 2. Dibujado de modelos dentro del proyecto e imposición de comportamientos.

De forma general, todos los objetos con excepción de los repetidos y que forman parte de la ambientación, se le hace un reinicio a la matriz de posiciones para ajustarla a la posición (0,0,0) es decir, en el centro de la escena para poder hacer uso correcto de su pivote una vez que están dibujados dentro de nuestro trabajo. Es con la traslación que establecemos la posición que queremos que tenga dentro de nuestro espacio y con la rotación la animación que se desea que haga. En esta ocasión la televisión se encuentra colgando del techo con ayuda de dos agarraderas las cuales permiten deslizarse de forma horizontal; para lograr lo anterior se requirió una variable que representara la rotación deseada para la televisión y una segunda variable booleana para saber cuándo es que la tele está en movimiento.

```
bool teleMov = false;
float teleRot = 0.0;
```

Figura 2.1. Variables creadas para la animación.

Los condicionales establecen los puntos de inicio y de término de la animación y también el reinicio de la mismo en su posición original, en este caso, se hace uso de la tecla M para realizar lo anterior:

```
if (keys[GLFW_KEY_M]) //MOVIMIENTO TELEVISION
{
    if (teleMov == false)
    {

        teleMov = true;

    }
    else
    {
        teleRot = 0.0f;
        teleMov = false;
    }
}
```

Figura 2.2. Condicionales del movimiento de la televisión.

La animación se basa en la rotación del eje Z y que no sea mayor a 30 grados la cual irá aumentando de 0.5 en 0.5 hasta que llegue a los dichos 30 grados y entonces se reinicien los grados a ceros.

```
if (teleMov) //MOVIMIENTO DE LA TELE
{

    if (teleRot < 30.0f)
    {
        teleRot += 0.5f;
    }

}
```

Figura 2.3. Constante de aumento en la rotación de la televisión.

Figura 2.4. Animación de la televisión.

Movimiento del cartel

```
//Banda
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-6.5, 6.25, -0.3));
model = glm::rotate(model, glm::radians(-rot), glm::vec3(0.0f, 1.0f, 0.0));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
pokeBanda.Draw(lightingShader);
```

Figura 3. Dibujado de modelos dentro del proyecto e imposición de comportamientos.

Para el movimiento de la banda-cartel que se encuentra girando de forma constante dentro del videojuego, sin embargo, por las condiciones establecidas para este trabajo, dicho movimiento debe ser activado por una tecla la cual comienza una rotación sobre el eje Y que sólo puede ser interrumpida apretando la misma tecla una vez más, pero es necesario que vaya en una rotación inversa a la acostumbrada ya que por la distribución del cartel éste tendrá que ser leída de izquierda a derecha haciendo uso de la tecla O.

Al igual que con la animación de la televisión, se requirió una variable que representara la rotación deseada para la televisión y una segunda variable booleana para saber cuándo es que la tele está en movimiento para poder paso al siguiente incremento en la rotación de uno en uno.

```
if (bandaMov == true)
{
    rot += 1.0f;
}
```

Figura 3.1. Función que permite mover la banda del cartel constantemente.

Figura 3.2. Animación para el cartel.

Movimiento de pokebolas

El movimiento de las pokebolas buscó parecerse al original visto en videojuegos el cual tanto la base blanca y roja se abren hasta cierto ángulo para poder desplegar uno de los monstruos que tienen en su interior. En las versiones del centro pokemon en *Omega Ruby* es común ver encima de las mesas un par de pokebolas que sirven para poder ambientar el escenario.

```
//Base pokebola
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-4, 2.5, -3.5));
model = glm::rotate(model, glm::radians(45.0f), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(pokeRotBase), glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
basePokeBola.Draw(lightingShader);

//Tapa pokebola
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-4, 2.5, -3.5));
model = glm::rotate(model, glm::radians(45.0f), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::rotate(model, glm::radians(pokeRot), glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
tapaPokeBola.Draw(lightingShader);
```

Figura 4. Función que permite mover la banda del cartel constantemente.

La animación que permite cerrar y abrir requirió de 5 variables booleanas y dos flotantes que pudieran determinar el ángulo sobre el cual cada una de las tapas de la pokebola rotaran haciendo uso de la tecla P.

```
bool abierta = false;
bool cerrada = true;
bool seMueve = false;
float pokeRot = 0.0f;

bool abiertaBase = false;
bool cerradaBase = true;
float pokeRotBase = 0.0f;
```

Figura 4.1. Variables utilizadas para la animación de las pokebolas.

```
if (seMueve)
{
    if (cerrada && cerradaBase)
    {
        if (pokeRot > -45.0f)
        {
            pokeRot -= 0.1f;
        }
        else
        {
            cerrada = false;
            abierta = true;
            seMueve = false;
        }

        if (pokeRotBase < 45.0f)
        {
            pokeRotBase += 0.1f;
        }
        else
        {
            cerradaBase = false;
            abiertaBase = true;
            seMueve = false;
        }

    }
    if (abierta && abiertaBase)
    {
        if ((pokeRot < 0.0f))
        {
            pokeRot += 0.1f;
        }
        else
        {
            cerrada = true;
            abierta = false;
            seMueve = false;
        }

        if ((pokeRotBase > 0.0f))
        {
            pokeRotBase -= 0.1f;
        }
        else
        {
            cerradaBase = true;
            abiertaBase = false;
            seMueve = false;
        }

    }
}
```

Figura 4.2. Función y lógica utilizada para el movimiento de las pokebolas.

En primera instancia se comprueba si la pokebola está en movimiento, si no lo hace, entonces se comprueba que ambas tapas estén cerradas, si lo están, entonces el usuario desea que la animación haga que las pokebolas se abran. Ambas tapas rotarán hasta llegar a un ángulo de 45 grados, en el caso de la tapa, el valor de la rotación irá decrementando hasta llegar a -45 grados y la base irá incrementando hasta llegar al dicho ángulo. En caso de que ambas tapas estén abiertas, entonces la animación lo que buscará es que se cierren y al igual que con el caso anterior los valores de la rotación irán decrementando y aumentando dependiendo el caso hasta que ambas tapas estén en cero grados simbolizando el cierre de la pokebola.

Figura 4.3. Animación de apertura de las pokebolas.

Figura 4.4. Animación de cerrado de las pokebolas.

Animaciones Complejas

Movimiento de las burbujas

La idea del movimiento de las burbujas nació de los tubos que se encuentran a los laterales del centro pokemon, en el videojuego, estas burbujas salen en tiempos aleatorios, pero únicamente son desplazadas verticalmente, sin embargo, el movimiento de las burbujas suele ser muy erráticos y no son desplazadas únicamente de forma vertical ya que al mismo tiempo van oscilando o irse separando una burbuja de otra. En el caso del proyecto final, se optó por que las burbujas fuesen oscilando a medida que fuesen subiendo por el tubo.

Este oscilamiento bien puede ser representado por una función senoidal, en donde la amplitud de la misma representa el intervalo sobre el cual las burbujas se van a mover de lado a lado o de izquierda a derecha realizando dicho movimiento en un intervalo de altura de 0 a 6 la cual es aproximadamente el alto del modelo del tubo.

Siguiendo entonces la siguiente función y su representación en Geogebra es como comienza su implementación en código.

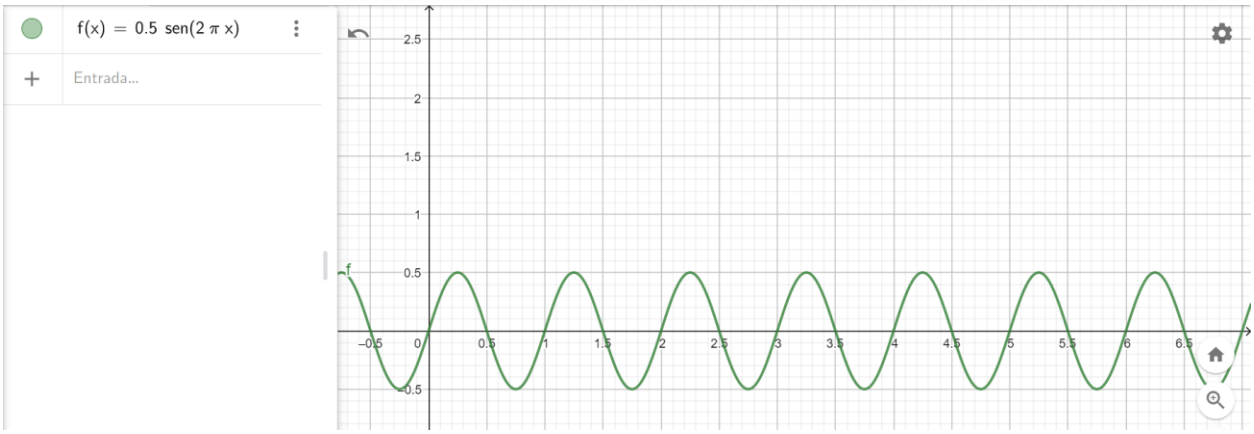


Figura 5. Función que representa el oscilamiento senoidal de las burbujas.

Para comenzar la explicación de la implementación de las burbujas es importante recalcar que estas fueron uno de los objetos translúcidos que existen dentro de la escena, y por ello, requieren de un agregado de líneas de código adicionales que les permita tener esta característica visual.

```
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-5.65, 0, -8.9));
model = glm::translate(model, glm::vec3(0, burbujaVert, 0.5 * sin(2 * 3.1416 * burbujaVert)));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0);
pokeBurbujas.Draw(lightingShader);
```

Figura 5.1. Dibujado y asignación de translucidez como propiedad.

Continuando con la animación, para poder realizarla entonces se hace uso de una variable booleana que permita saber si las burbujas se encuentran en movimiento y que también permita que a medida que estén oscilando o moviéndose de lado a lado vayan ascendiendo. El traslado vertical es controlado por la variable *burbujaVert*.

```
if (burbujaMov == true)
{
    if (burbujaVert < 6.0f)
    {
        burbujaVert += 0.01f;

    }
    else
    {
        burbujaVert = 0.0f;
        burbujaMov = false;
    }

}
```

Figura 5.2. Función que limita la ascensión de las burbujas.

La cual establece que cuando se llegue a una altura de 6, la animación termine mientras que la oscilación continuará mientras la variable burbujaVert siga recibiendo valores que asignar a la función vista en Geogebra con la tecla K.

Figura 5.3. Animación de las burbujas.

Movimiento del botiquín

El movimiento del botiquín es una animación hecha a partir de *keyframes*, en donde se busca que el modelo del botiquín abra su tapa y de su interior salga un spray de primeros auxilios para los pokemons que vayan al centro pokemon y se ponga sobre la mesa durante unos segundos para después volver adentro del botiquín.

Explicado lo anterior, entonces la tapa del botiquín únicamente rotará mientras que el spray tendrá un desplazamiento y una rotación para que se puesto sobre la mesa, es entonces que se crearon las siguientes variables que permitirán dichos movimientos.

```
// Keyframes
float sprayX =PosIni.x, sprayY = PosIni.y, sprayZ = PosIni.z, rotSpray = 0, rotTapa = 0;
```

Figura 6. Variables que manejan los movimientos del botiquín y el spray.

En donde las variables *sprayX*, *sprayY* y *sprayZ* son variables que nos permitirán manipular la posición del spray dentro de la animación mientras que rotSpray permitirá el manejo de su rotación y rotTapa la del botiquín. Posteriormente, estas variables las incluimos dentro de nuestra estructura que irá capturando los frames que deseamos que se incluyan dentro de nuestra animación.

```
typedef struct _frame
{
    //Variables para GUARDAR Key Frames
    float sprayX;       //Variable para PosicionX
    float sprayY;       //Variable para PosicionY
    float sprayZ;       //Variable para PosicionZ
    float incX;     //Variable para IncrementoX
    float incY;     //Variable para IncrementoY
    float incZ;     //Variable para IncrementoZ
    float rotSpray;
    float rotIncSpray;
    float rotTapa;
    float rotInc;

}FRAME;
```

Figura 6.1 Inclusión de las variables para ser capturadas por el keyframe.

Así mismo, debido a que estas nuevas variables fueron creadas, igualmente tienen que ser incluidas dentro de las funciones que permiten realizar la toma de keyframes de forma manual siendo estas *saveFrame()*, *resetElements()*, *interpolation()*.

```
void saveFrame(void)
{

    printf("posx %f\n", sprayX);

    KeyFrame[FrameIndex].sprayX = sprayX;
    KeyFrame[FrameIndex].sprayY = sprayY;
    KeyFrame[FrameIndex].sprayZ = sprayZ;

    KeyFrame[FrameIndex].rotSpray = rotSpray;
    KeyFrame[FrameIndex].rotTapa = rotTapa;


    FrameIndex++;
}

void resetElements(void)
{
    sprayX = KeyFrame[0].sprayX;
    sprayY = KeyFrame[0].sprayY;
    sprayZ = KeyFrame[0].sprayZ;

    rotSpray = KeyFrame[0].rotSpray;
    rotTapa = KeyFrame[0].rotTapa;

}

void interpolation(void)
{

    KeyFrame[playIndex].incX = (KeyFrame[playIndex + 1].sprayX - KeyFrame[playIndex].sprayX) / i_max_steps;
    KeyFrame[playIndex].incY = (KeyFrame[playIndex + 1].sprayY - KeyFrame[playIndex].sprayY) / i_max_steps;
    KeyFrame[playIndex].incZ = (KeyFrame[playIndex + 1].sprayZ - KeyFrame[playIndex].sprayZ) / i_max_steps;


    KeyFrame[playIndex].rotIncSpray = (KeyFrame[playIndex + 1].rotSpray - KeyFrame[playIndex].rotSpray) / i_max_steps;
    KeyFrame[playIndex].rotInc = (KeyFrame[playIndex + 1].rotTapa - KeyFrame[playIndex].rotTapa) / i_max_steps;

}
```

Figura 6.2 Inclusión de las variables para ser capturadas por el keyframe.

Además de todas estas funciones, se añadió una más correspondiente al traspaso de información de cada uno de las posiciones y rotaciones deseadas dentro de nuestra animación para así cambiar el valor de las variables globales correspondientes a dichas acciones.

```
void setFramesValues(float posSprayX, float posSprayY, float posSprayZ, float rotRotTapa, float rotRotSpray)
{
    sprayX = posSprayX;
    sprayY = posSprayY;
    sprayZ = posSprayZ;
    rotSpray = rotRotSpray;
    rotTapa = rotRotTapa;
}
```

Figura 6.3. Manejo del valor de las variables mediante una función.

Para la animación se pensó entonces realizar la apertura del botiquín para posteriormente desplazar el spray lentamente, en primera instancia, fuera del botiquín, y después rotarlo para que este quede verticalmente asentado encima de la mesa.

Sabiendo la anterior, se ocuparon los siguientes keyframes para su realización:

setFramesValues(-5.3, 1, -1.5,0,0); //Posiciones iniciales de los objetos

setFramesValues(-5.3, 1, -1.5, 30,0); //Apertura de la tapa del botiquín

setFramesValues(-5.3, 1.5, -1.5, 30,0); //Alza del spray dentro del botiquín

setFramesValues(-5.0, 1.5, -0.5, 30,0); //Desplazamiento horizontal del spray fuera del botiquín

setFramesValues(-5.0, 2.5, -0.5, 30, 0); //Alza nuevamente del spray una vez fuera del botiquín

setFramesValues(-3.5, 2.5, -0.5, 30, 90); //Rotación del spray y puesta del mismo sobre la mesa.

```
//Animacion por keyframe

setFramesValues(-5.3, 1, -1.5,0,0);
saveFrame();


setFramesValues(-5.3, 1, -1.5, 30,0);
saveFrame();

setFramesValues(-5.3, 1.5, -1.5, 30,0);
saveFrame();

setFramesValues(-5.0, 1.5, -0.5, 30,0);
saveFrame();

setFramesValues(-5.0, 2.5, -0.5, 30, 0);
saveFrame();

setFramesValues(-3.5, 2.5, -0.5, 30, 90);
saveFrame();

setFramesValues(-3.5, 2.5, -0.5, 30, 90);
saveFrame();

setFramesValues(-5.3, 1, -1.5, 0,0);
saveFrame();
```

Figura 6.4 Keyframes tomados con las posiciones de nuestra animación.

Figura 6.5 Primer keyframe con posiciones iniciales.

Figura 6.6 Segundo keyframe con la tapa abierta.

Figura 6.7 Tercer keyframe levantando el spray con la tapa abierta.

Figura 6.8 Cuarto keyframe con el spray fuera del botiquín.

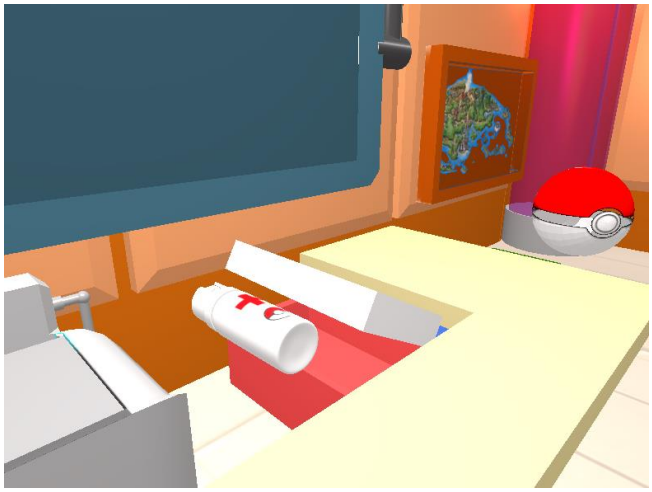


Figura 6.9 Quinto keyframe alza de el spray fuera del botiquín.

Figura 6.10 Sexto keyframe puesta de el spray en la mesa.

Figura 6.11 Séptimo y octavo keyframe de el spray en la mesa.

Hecho lo anterior, solo resta controlar a partir de la transformación de rotación de OpenGL el modelo del botiquín y el spray mediante la tecla L.

```
//Tapa botiquin
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(-5.3, 1.5, -2.1));
model = glm::rotate(model, glm::radians(-rotTapa), glm::vec3(1.0f, 0.0f, 0.0));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
tapaPokeBotiquin.Draw(lightingShader);

//Spray
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(sprayX, sprayY, sprayZ));
model = glm::rotate(model, glm::radians(-rotSpray), glm::vec3(0.0f, 0.0f, 1.0));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
pokeSpray.Draw(lightingShader);
```

Figura 6.12 Manejo de las transformaciones de OpenGL mediante nuestras varibles controladas por keyframe.

Luces

Cómo última parte de este manual técnico, se hará mención a las luces que fueron utilizadas para ambientar la escena con un total de cuatro.

```
// Point light 1
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].position"), pointLightPositions[0].x, pointLightPositions[0].y, pointLightPositions[0].z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].ambient"), 0.05f, 0.05f, 0.05f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].diffuse"), LightP1.x, LightP1.y, LightP1.z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].specular"), LightP1.x, LightP1.y, LightP1.z);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].linear"), 0.09f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[0].quadratic"), 0.032f);

// Point light 2
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].position"), pointLightPositions[1].x, pointLightPositions[1].y, pointLightPositions[1].z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].ambient"), 1.0f, 0.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].diffuse"), 1.0f, 0.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].specular"), 1.0f, 1.0f, 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].linear"), 0.7f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].quadratic"), 1.8f);

// Point light 3
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].position"), pointLightPositions[2].x, pointLightPositions[2].y, pointLightPositions[2].z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].ambient"), 1.0f, 0.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].diffuse"), 1.0f, 0.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].specular"), 1.0f, 1.0f, 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[2].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[2].linear"), 0.7f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[2].quadratic"), 1.8f);

// Point light 4
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[3].position"), pointLightPositions[3].x, pointLightPositions[3].y, pointLightPositions[3].z);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[3].ambient"), 0.4f, 0.4f, 0.4f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[3].diffuse"), 0.4f, 0.4f, 0.4f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[3].specular"), 1.0f, 1.0f, 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[3].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[3].linear"), 0.045f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[3].quadratic"), 0.0075f);
```

Figura 7 Definición de las luces del escenario

Las luces utilizadas se encuentran en 4 puntos en concreto dentro de la estructura, una debajo del cartel circular del centro, una el techo y las otras dos dentro de cada uno de los tubos para iluminar las burbujas siendo estas dos últimas las de menor dispersión por la menor cantidad de área que tienen que iluminar y la del techo, la de mayor dispersión siendo conscientes que esta luz se encarga de iluminar todo el centro pokemon. Para

lograr estos efectos entonces se asignaron valores a las cantidades lineales y cuadráticas de las luces con base en la siguiente tabla.

Distance	Constant	Linear	Quadratic
7	1.0	0.7	1.8
13	1.0	0.35	0.44
20	1.0	0.22	0.20
32	1.0	0.14	0.07
50	1.0	0.09	0.032
65	1.0	0.07	0.017
100	1.0	0.045	0.0075
160	1.0	0.027	0.0028
200	1.0	0.022	0.0019
325	1.0	0.014	0.0007
600	1.0	0.007	0.0002
3250	1.0	0.0014	0.000007

Conclusión

Con la realización del presente trabajo se logró poner en práctica todos y cada uno de los conceptos vistos en el curso para poder manejar de forma adecuada cada uno de los conceptos de OpenGL así como conocer sus alcances y limitaciones de este así como las posibilidades que tiene un alumno iniciado en el área para poder representar proyectos desde perspectivas distintas, esperando que alguno de los conocimientos obtenidos en la materia lleguen a ser de utilidad en el campo laboral.