Universidad Nacional Autónoma de México

Facultad de Ingeniería

Computación Gráfica e Interacción Humano-Computadora --- Laboratorio

Proyecto final

Alumno: Cuevas Lagos Jordi Octavio

Número de Cuenta: 316262615

Grupo Laboratorio: 09

Semestre: 2022-2

Manual técnico

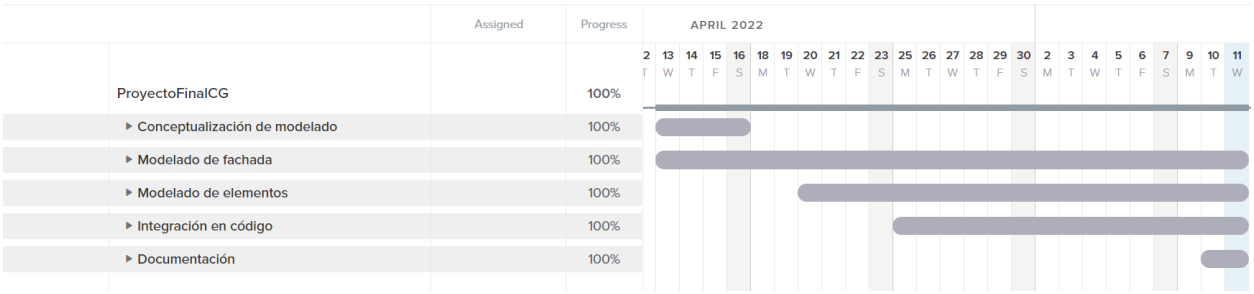
Proyecto Final

Índice

Objetivo

El objetivo de este proyecto es aplicar todo lo aprendido durante las prácticas de laboratorio con una recreación de una fachada y un interior seleccionados previamente.

Diagrama de Gantt



Alcance del proyecto

La finalidad de este proyecto es meramente educativa. No se busca recrear fielmente el entorno desarrollado ni mostrar maestría en el desarrollo de modelado en 3D, sino comprobar que el alumno ha aprendido con interés los contenidos del curso.

Limitantes

Solo se puede utilizar los recursos proporcionados durante el curso, esto con el objetivo de comprobar el aprendizaje durante el laboratorio. También evita que externos realicen el proyecto en lugar del alumno. Respecto al contenido del proyecto, se debe tratar de recrear un entorno de la vida real y se recreara un mínimo de objetos para considerar el proyecto como suficiente.

Por último, este proyecto fue realizado sin ánimos de lucro, por lo que la monetización de los resultados está fuera de consideración.

Documentación de código

Importación de modelos

```
Model Lavamanos((char*)"Models/Lavamanos/Lavamanos.obj");
Model Retrete((char*)"Models/Retrete/Toilet 03.obj");
Model Cortador((char*)"Models/Cortador/CortadorPizza.obj");
Model Pizza((char*)"Models/Pizza/Entera/PizzaEntera.obj");
Model Rebanada((char*)"Models/Pizza/Rebanada/RebanadaPizza.obj");
Model PuertaBano((char*)"Models/Puertas/Baño/PuertaBano.obj");
Model Refresco((char*)"Models/Refresco/cocaBotella.obj");
Model BoteBasura((char*)"Models/BoteBasura/Bote_Basura.obj");
Model Silla((char*)"Models/Silla/Silla.obj");
Model Mesa((char*)"Models/Mesa/Mesa.obj");
Model CajaPizza((char*)"Models/PizzaCaja/CajaPizza.obj");
Model BaseCaja((char*)"Models/CajaPizza/CajaAbierta/BaseCaja.obj");
Model TapaCaja((char*)"Models/CajaPizza/CajaAbierta/TapaCaja.obj");
Model Horno((char*)"Models/HornoCalentado/HornoCalentado.obj");
Model Fachada((char*)"Models/Fachada/Fachada.obj");
Model Tablero1((char*)"Models/TableroPromocion/Tablero1/Tablero1.obj");
Model Tablero2((char*)"Models/TableroPromocion/Tablero2/Tablero2.obj");
Model Tablero3((char*)"Models/TableroPromocion/Tablero3/Tablero3.obj");
```

La importación de modelos no fue complicada, solo llevó algo de esfuerzo el escribir todas las rutas para todos los objetos. Se separó en el código los modelos que llevarían algo de transparencia.

```
Model Vidrios((char*)"Models/Vidrios/Vidrios.obj");
Model PuertaVidrio1((char*)"Models/Puertas/Vidrio/PuertaVidrio.obj");
Model PuertaVidrio2((char*)"Models/Puertas/Vidrio/PuertaVidrio2.obj");
```

Manejo de transformaciones

La mayoría de los modelos solo requieren una traslación. Para el caso algunos objetos, como las sillas, fue necesario hacer cambios en el pivote para realizar rotaciones correctamente.

```
//Mesas
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(0, 0, 0));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Mesa.Draw(lightingShader);

model = glm::mat4(1);
model = glm::translate(model, glm::vec3(3.514, 0, 0));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Mesa.Draw(lightingShader);

model = glm::mat4(1);
model = glm::translate(model, glm::vec3(7.528, 0, 0));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Mesa.Draw(lightingShader);
```

Por ejemplo, como las mesas son iguales si son rotadas 90°, no fue necesario un a rotación y solamente se trasladó.

```
//Sillas
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(SillaInicialX, SillaInicialY, SillaInicialZ));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Silla.Draw(lightingShader);

model = glm::mat4(1);
model = glm::translate(model, glm::vec3(SillaInicialX + 2.0, SillaInicialY, SillaInicialZ));
model = glm::rotate(model, glm::radians(180.0f), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Silla.Draw(lightingShader);

model = glm::mat4(1);
model = glm::translate(model, glm::vec3(SillaInicialX + 3.5, SillaInicialY, SillaInicialZ));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Silla.Draw(lightingShader);
```

En el caso de las sillas, si hay una distinción al rotarlas 90°, por lo que se requiere de una rotación, después de trasladar.

Además, se crearon variables para almacenar las posiciones iniciales de algunos objetos, con la finalidad de facilitar la colocación de múltiples copias del mismo objeto, así como facilitar la animación de estos objetos. Esto se comentará más adelante en el apartado de animaciones.

Transparencias

```
//Traslucidez

glEnable(GL_BLEND);
glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
//model = glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, 0.0f, 0.0f));

//Vidrios
model = glm::mat4(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0);
Vidrios.Draw(lightingShader);

//Puertas
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(PuertaVidrio1X, PuertaVidrio1Y, PuertaVidrio1Z));
model = glm::rotate(model, glm::radians(-1.0f* RotPuerta), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0);
PuertaVidrio1.Draw(lightingShader);

model = glm::mat4(1);
model = glm::translate(model, glm::vec3(PuertaVidrio2X, PuertaVidrio2Y, PuertaVidrio2Z));
model = glm::rotate(model, glm::radians(RotPuerta), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0);
PuertaVidrio2.Draw(lightingShader);


glDisable(GL_BLEND);
glEnable(GL_DEPTH_TEST);
glBindVertexArray(0);
```

En esencia dibujamos los modelos con la característica de transparencia activada. Estos objetos tienen una textura de color gris oscuro, el cual se interpretará como un material traslúcido. Para evitar problemas con otras texturas oscuras, se apaga la transparencia al importar la fachada.

```
//Carga de modelo

//Fachada
glm::mat4 model(1);
model = glm::translate(model, glm::vec3(0, 0, 0));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(lightingShader.Program, "transparencia"), 0.0);
Fachada.Draw(lightingShader);
```

Pointlights

Se colocaron 4 PointLights para iluminar la fachada por dentro, uno en la cocina, uno en la planta superior y los dos restantes en los baños. Las luces de los baños son tenues, mientras que las otras son relativamente intensas.

```
// 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"), 0.05f, 0.05f, 0.05f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].diffuse"), 1.0f, 1.0f, 0.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].specular"), 1.0f, 1.0f, 0.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].constant"), 1.0f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].linear"), 0.09f);
glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[1].quadratic"), 0.032f);

// 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"), 0.05f, 0.05f, 0.05f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].diffuse"), 0.0f, 1.0f, 1.0f);
glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].specular"), 0.0f, 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);
```

Por ejemplo, la Pointlight 2 tiene un alcance decente por los valores de su lineal y cuadrática, mientras que la Pointlight 3 tiene un alcance bajo por dichos valores.

Animaciones

Para las animaciones se crearon variables de varios tipos: Booleanos para mantener un control de la animación con las teclas, flotantes que almacenan la posición inicial del modelo para evitar perderlo y flotantes que guardan valor que se agrega a la posición del objeto para modificarlo con una transformación:

```
//Cortador de pizza
float CortadorX = 0.0f;
float CortadorY = 0.0f;
float CortadorZ = 0.0f;
//Rebanada de pizza
float RebanadaX = -1.867;
float RebanadaY = 1.516;
float RebanadaZ = 6.699;
//Puertas de vidrio
//Puerta alta
float PuertaVidrio1X = 4.75;
float PuertaVidrio1Y = 5.059;
float PuertaVidrio1Z = -7.453;
```

```
//Puerta baja
float PuertaVidrio2X = 4.5;
float PuertaVidrio2Y = -0.062;
float PuertaVidrio2Z = -5.2;

//Banderas lógicas para controlar animaciones sencillas
bool PuertaAbierta = false;
bool CajaCerrada = false;
bool SillaMovida = false;
bool PizzaCortada = false;
bool TomarRebanada = false;

//Variables para controlar animaciones
float RotPuerta = 0;
float RotTapa = 0;
float desplazaSilla = 0;
float rebanadadesplX = 0;
float rebanadadesplY = 0;
```

Para animar, cambiamos dinámicamente una de las variables con ese propósito dentro de la función doMovement(). Para activar las animaciones, se hace uso de la función KeyCallback para controlar los booleanos. Los detalles se muestran en su modelo correspondiente.

Silla

```
model = glm::mat4(1); //Silla más próxima a la puerta
model = glm::translate(model, glm::vec3(SillaInicialX + 9.4 + SillaMovida, SillaInicialY, SillaInicialZ));
model = glm::rotate(model, glm::radians(180.0f+SillaMovida*3), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
Silla.Draw(lightingShader);
```

La animación de la silla es relativamente sencilla. Consiste en dos pasos: Primero se desplaza la silla a su posición correspondiente, la cual varía gracias a la variable “SillaMovida”. Esta variable se modifica en la función doMovement(): Luego se hace la rotación requerida para el objeto, la cual varía por un valor proporcional a la variable “SillaMovida”. Por la sencillez de estas dos transformaciones, considero que esta es una animación sencilla.

```
//Animaciones
//Silla
if ((SillaMovida) && (desplazaSilla < 3)) {
    desplazaSilla += 0.0000001;
}
else if ((!SillaMovida) && (desplazaSilla > 0)) {
    desplazaSilla -= 0.0000001;
}
```

```
if (keys[GLFW_KEY_I]) {
    SillaMovida = !SillaMovida;
}
```

De esta forma, si se pulsa la tecla I, se transita entre el estado inicial y final de la animación.

Puertas

```
//Puertas de Baño
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(PuertaBanoX, PuertaBanoY, PuertaBanoZ));
model = glm::rotate(model, glm::radians(-1.0f*RotPuerta), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
PuertaBano.Draw(lightingShader);
```

```
//Puertas
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(PuertaVidrio1X, PuertaVidrio1Y, PuertaVidrio1Z));
model = glm::rotate(model, glm::radians(-1.0f* RotPuerta), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
glUniform1f(glGetUniformLocation(lightingShader.Program, "transparencia"), 1.0);
PuertaVidrio1.Draw(lightingShader);
```

Esta animación sencilla solo utiliza una rotación, controlada por la variable “RotPuerta” tanto para las puertas de vidrio como las de los baños. La variable está limitada a 90°, y el sentido del giro se determina dentro de la transformación de rotación.

```
if (keys[GLFW_KEY_Y]) {
    PuertaAbierta = !PuertaAbierta;
}
```

```
//Puertas
if ((PuertaAbierta) && (RotPuerta < 80)) {
    RotPuerta += 1;
}
else if ((!PuertaAbierta) && (RotPuerta > 0)) {
    RotPuerta -= 1;
}
```

La lógica para activar y controlar la animación es similar a la de silla.

Caja

Para esta animación se requirió separar un modelo en dos partes para una animación correcta. La parte que se mueve es la parte superior (La tapa), la base es estática en todo momento.

```
Model BaseCaja((char*)"Models/CajaPizza/CajaAbierta/BaseCaja.obj");
Model TapaCaja((char*)"Models/CajaPizza/CajaAbierta/TapaCaja.obj");
```

Después de modificar el pivote de la tapa en Maya, simplemente se hace una rotación a la tapa respecto al eje Z para abrir y cerrar la caja. La lógica es muy similar a la de Silla y Puertas.

```
model = glm::mat4(1);
model = glm::translate(model, glm::vec3(TapaPizzaX, TapaPizzaY, TapaPizzaZ));
model = glm::rotate(model, glm::radians(RotTapa), glm::vec3(0.0f, 0.0f, 1.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
TapaCaja.Draw(lightingShader);
```

```
//Caja
if ((CajaCerrada) && (RotTapa < 115)) {
    RotTapa += 1;
}
else if ((!CajaCerrada) && (RotTapa > 0)) {
    RotTapa -= 1;
}
```

```
if (keys[GLFW_KEY_U]) {
    CajaCerrada = !CajaCerrada;
}
```

Cortador de pizza

El cortador de pizza fue animado por keyframes, por lo que se definen los elementos que van a cambiar dentro de la animación.

```
typedef struct _frame
{
    //Variables para GUARDAR Key Frames
    float incX;
    float incY;
    float incZ;

    float incRotX;
    float incRotY;
    float incRotZ;

    float rot1Y;
    float rot1X;
    float rot1Z;

    float pos1X;        //Variable para PosicionX
    float pos1Y;        //Variable para PosicionY
    float pos1Z;        //Variable para PosicionZ

}FRAME;
```

Luego de configurar las funciones requeridas para el funcionamiento de los keyframes, se guardan los valores correspondientes para hacer el movimiento:

```
CortadorPosX = 0;
CortadorPosY = 0;
CortadorPosZ = 0;
RotCortadorY = 0;
RotCortadorX = 0;
RotCortadorZ = 0;
saveFrame();

CortadorPosX = 1.5;
CortadorPosY = 0.1;
CortadorPosZ = -0.2;
//RotCortadorY = 0;
RotCortadorX = 90;
RotCortadorZ = 15;
saveFrame();

CortadorPosX = 0.5;
CortadorPosZ = -0.5;
saveFrame();
```

```
if (keys[GLFW_KEY_L])
{
    if (play == false && (FrameIndex > 1))
    {

        resetElements();
        //First Interpolation
        interpolation();

        play = true;
        playIndex = 0;
        i_curr_steps = 0;

        resetElements();

    }
    else
    {
        play = false;
    }
    PizzaCortada = true;
    rebanadadesplX = 2.2;

}
```

Luego de apretar la tecla L, la animación desplaza la rebanada de pizza dentro de la caja de pizza para simular un corte. Esto es importante pues encadena la última animación.

Rebanada de pizza

La rebanada de pizza tiene un desplazamiento parabólico, esto con el fin de simular la forma que tomamos la pizza de la caja una vez cortan la pizza en la caja. Para obtener el modelo que simula el comportamiento parabólico se consideraron tres puntos:

$$P_1(0,0), \qquad P_2(2.2,0), \quad P_3(1.1,1)$$

A partir de estos puntos se sustituye en la ecuación general de segundo grado, obteniendo un sistema de ecuaciones de dos variables

$$4.84a + 2.2b = 0$$
$$1.21a + 1.1b = 1$$

Al resolver el sistema de ecuaciones obtenemos:

$$a = -0.75, \quad b = 1.64$$

De esta forma, tenemos la ecuación que nos permite simular la parábola deseada.

```
//Pizza
if ((TomarRebanada) && rebanadadesplX > 0)  {
    rebanadadesplX -= 0.01;
    rebanadadesplY = -0.75 * rebanadadesplX * rebanadadesplX  + 1.64 * rebanadadesplX;
}
```

```
if (keys[GLFW_KEY_O]) {
    TomarRebanada = !TomarRebanada;
}
```

Por la forma que está configurado el objeto (su posición original se encuentra en la componente X negativa), la animación no puede empezar a menos que la animación del cortador desplace a la rebanada a una posición con una componente X mayor a 0. Es decir, se debe cortar la pizza antes de tomar la rebanada.