Universidad Nacional Autónoma de México

Facultad de Ingeniería

Computación Gráfica e Interacción Humano-Computadora --- Laboratory

Final Proyect

Student: Cuevas Lagos Jordi Octavio

Acoount Number: 316262615

Laboratory Group: 09

Semester: 2022-2

Tecnical Manual

Final Proyect

Índex

Objetive……..3

Gantt Diagramm……..3

Proyect Scope……..3

Restrictions……..3

Code Documentation……..3

Model Importing……..3

Transformation Management……4

Transparencies……5

Point Lights……..5

Animations……6

Chair……6

Doors……7

Pizza box……7

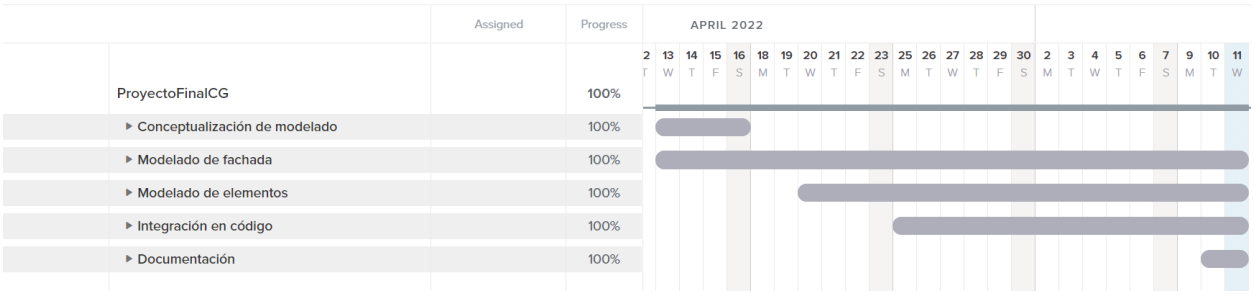
Pizza Cutter……8

Pizza Slice……9

Objetive

The objective of this project is to apply everything learned during the laboratory practices with the recreation of a previously selected facade and interior.

Gantt Diagram



Project Scope

The purpose of this project is purely educational. It does not seek to faithfully recreate the developed environment or show mastery in the development of 3D modeling, but to verify that the student has learned with interest the contents of the course.

Restrictions

Only the resources provided during the course can be used, this with the objective of testing the learning during the laboratory. It also prevents outsiders from doing the project in place of the student. Regarding the content of the project, it should try to recreate a real-life environment and a minimum of objects will be recreated to consider the project as sufficient.

Finally, this project was conducted on a non-profit basis, so monetization of the results is out of the question.

Code Documentation

Model Importing

```
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");
```

Importing models was not complicated, it just took some effort to write all the paths for all the objects. The models that would carry some transparencies were separated in the code.

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

Transformation Management

Most of the models only require one translation. In the case of some objects, such as chairs, it was necessary to make changes in the pivot to perform rotations correctly.

```
//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);
```

For example, since the tables are the same if rotated 90°, no rotation was necessary, and they were only moved.

```
//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);
```

In the case of chairs, if there is a distinction when rotating them 90°, so a rotation is required, after translating.

In addition, variables were created to store the initial positions of some objects, in order to facilitate the placement of multiple copies of the same object, as well as to facilitate the animation of these objects. This will be discussed later in the section on animations.

Transparencies

```
//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);
```

In essence we draw the models with the transparency feature activated. These objects have a dark gray texture, which will be interpreted as a translucent material. To avoid problems with other dark textures, the transparency is turned off when importing the facade.

```
//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

Four PointLights were placed to illuminate the facade from the inside, one in the kitchen, one on the upper floor and the other two in the bathrooms. The bathroom lights are dim, while the others are relatively bright.

```
// 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);
```

For example, Pointlight 2 has a decent range for its linear and quadratic values, while Pointlight 3 has a low range for those values.

Animations

Several types of variables were created for the animations: Booleans to keep control of the animation with the keys, floats that store the initial position of the model to avoid losing it and floats that store a value that is added to the position of the object to modify it with a transformation:

```
//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;
```

To animate, we dynamically change one of the variables for that purpose within the doMovement() function. To activate the animations, we make use of the KeyCallback function to control the booleans. The details are shown in its corresponding model.

Chair

```
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);
```

The animation of the chair is relatively simple. It consists of two steps: First, the chair is moved to its corresponding position, which varies thanks to the variable “SillaMovida”. This variable is modified in the function doMovement(): Then the required rotation is made for the object, which varies by a value proportional to the variable “SillaMovida”. Because of the simplicity of these two transformations, I consider this to be a simple animation.

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

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

In this way, pressing the I key transitions between the initial and final state of the animation.

Doors

```
//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);
```

This simple animation uses only one rotation, controlled by the variable "RotPuerta" for both glass and bathroom doors. The variable is limited to 90°, and the direction of rotation is determined within the rotation transformation.

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

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

The logic for activating and controlling the animation is like the chair model.

Pizza Box

For this animation it was required to separate a model into two parts for a correct animation. The part that moves is the top part (the lid), the base is static at all times.

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

After modifying the lid pivot in Maya, simply rotate the lid with respect to the Z axis to open and close the box. The logic is very similar to that of Chair and Doors.

```
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;
}
```

Pizza Cutter

The pizza cutter was animated by keyframes, so the elements that are going to change within the animation are defined.

```
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;
```

After configuring the required functions for keyframe operation, the corresponding values are saved to make the movement:

```
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;

}
```

After pressing the L key, the animation moves the pizza slice inside the pizza box to simulate a cut. This is important because it chains the last animation.

Pizza Slice

The pizza slice has a parabolic displacement, this in order to simulate the shape we take the pizza from the box once we cut the pizza in the box. To obtain the model that simulates parabolic behavior, three points were considered:

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

From these points we substitute in the general equation of second degree, obtaining a system of equations of two variables.

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

By solving the system of equations, we obtain:

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

Thus, we have the equation that allows us to simulate the desired parabola.

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

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

Because of the way the object is set up (its original position is in the negative X component), the animation cannot start unless the slicer animation moves the slice to a position with an X component greater than 0. In other words, the pizza must be cut before the slice is taken.