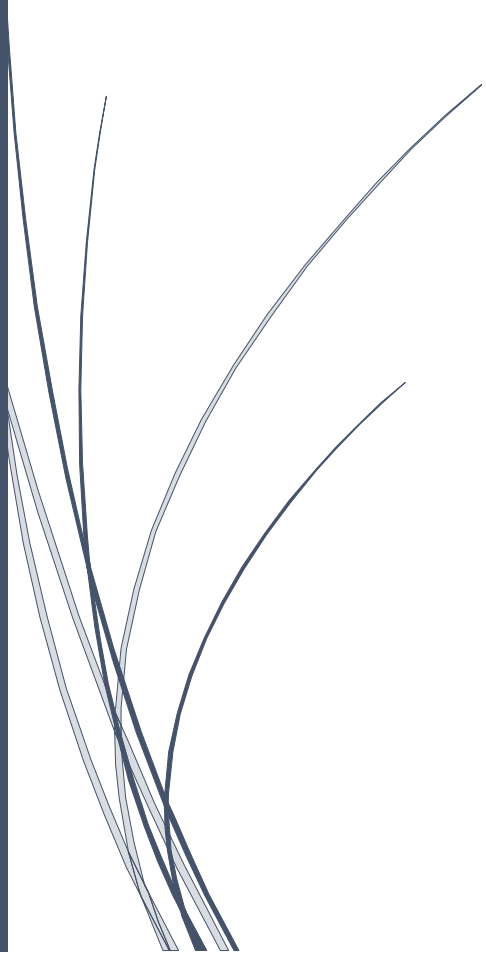
Laboratorio de computación grafica e interacción humano computadora

Manual técnico

Ambiente Virtual



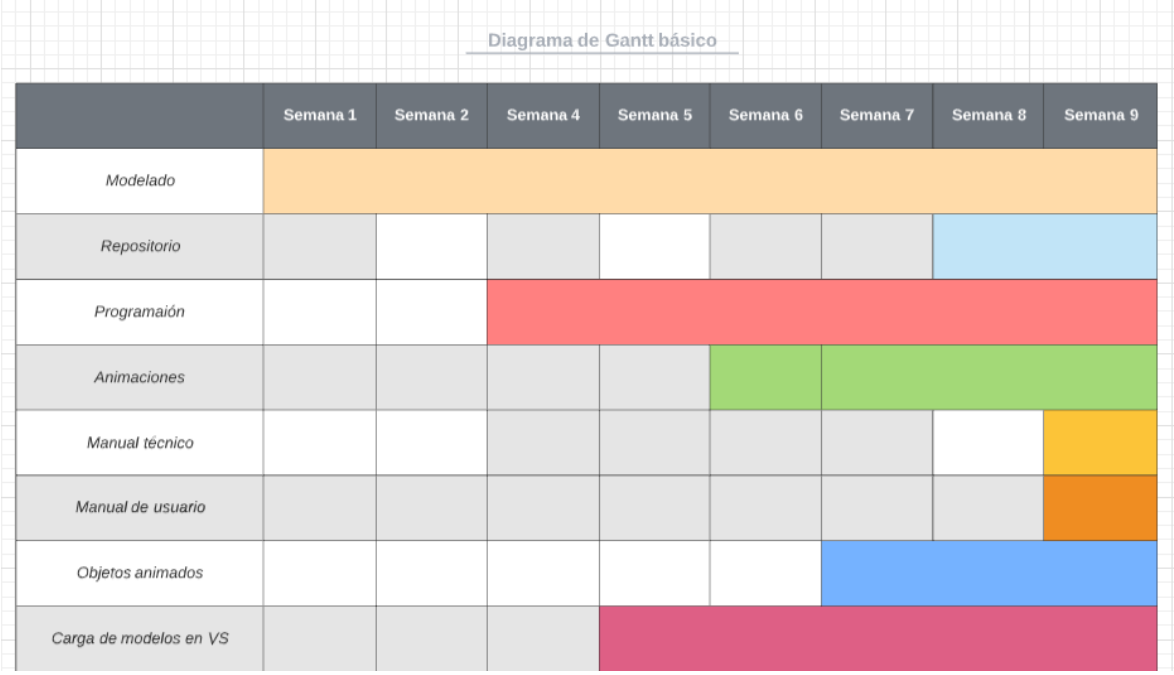
Valdelamar Tamez Valeria
GRUPO 04

Contenido

Objetivos

- El alumno aplicará los conocimientos adquiridos durante el semestre para la generación de un ambiente virtual.
- El alumno modelara y posicionara elementos de su propia autoría y bajo los recursos obtenidos de la red.
- El alumno documentará el proceso dentro de un repositorio de GitHub.

Diagrama de Gantt



Alcance del proyecto

El proyecto tiene como objetivo generar un ambiente virtual, del cual se toma como base el código que semana a semana durante todo el semestre, esto para que al final se obtenga un resultado que nos acerque a la realidad. Dicho programa se genera utilizando las herramientas para modelado MAYA, además del IDE de programación Visual Studio, utilizando el lenguaje de programación C++.

Dentro del proyecto se encuentran elementos de programación básicos, así como elementos nuevos que se tocaron a lo largo del curso.

Documentación de código

Dentro de las primeras líneas de código encontramos las bibliotecas que se encargan de generar lo necesario para las cargas de modelos, texturas y animaciones del proyecto.

```
//Load Models
#include "SOIL2/SOIL2.h"


// Other includes
#include "Shader.h"
#include "Camera.h"
#include "Model.h"
#include "Texture.h"
#include "modelAnim.h"
```

Generamos las variables globales de nuestro proyecto.

```
//variables globales dentro del entorno
float movX = 0.0;
float movZ = 0.0;
float rot = 0.0;
bool circuito = false;
bool recorrido1 = true;
bool recorrido2 = false;
bool recorrido3 = false;
bool recorrido4 = false;
bool recorrido5 = false;
```

Para la utilización de la técnica de animación por Keyframes se utilizan las siguientes líneas de código.

```
// Keyframes
float posX = PosIni.x, posY = PosIni.y, posZ = PosIni.z, rotder, rotizq, rotm;

#define MAX_FRAMES 9
int i_max_steps = 50;
int i_curr_steps = 0;
typedef struct _frame
{
    //Variables para GUARDAR Key Frames
    float posX;     //Variable para PosicionX
    float posY;     //Variable para PosicionY
    float posZ;     //Variable para PosicionZ
    float incX;     //Variable para IncrementoX
    float incY;     //Variable para IncrementoY
    float incZ;     //Variable para IncrementoZ
    float rotder;   //Variable para rotacion en puerta derecha
    float rotizq;   //Variable para rotacion en puerta izquierda
    float rotm;   //Variable para rotacion de pantalla Mac
    float rotInc;   //Variable para rotacion gradual
    float rotInc2;  //Variable para rotacion gradual
    float rotInc3;  //Variable para rotacion gradual
}FRAME;
```

Posteriormente al guardado de las variables para la animación, generamos la función resetElements, la cual nos ayudara a que dichas variables guarden los keyframes.

```
void resetElements(void)
{
    posX = KeyFrame[0].posX;
    posY = KeyFrame[0].posY;
    posZ = KeyFrame[0].posZ;

    rotder = KeyFrame[0].rotder;
    rotizq = KeyFrame[0].rotizq;
    rotm = KeyFrame[0].rotm;
}
```

La siguiente función para KeyFrames es la interpolación, es decir la que se encarga de utilizar la ecuación para la animación.

```
//Funcion de interpolacion de Keyframes
void interpolation(void)
{

    KeyFrame[playIndex].incX = (KeyFrame[playIndex + 1].posX - KeyFrame[playIndex].posX) / i_max_steps;
    KeyFrame[playIndex].incY = (KeyFrame[playIndex + 1].posY - KeyFrame[playIndex].posY) / i_max_steps;
    KeyFrame[playIndex].incZ = (KeyFrame[playIndex + 1].posZ - KeyFrame[playIndex].posZ) / i_max_steps;

    KeyFrame[playIndex].rotInc = (KeyFrame[playIndex + 1].rotder - KeyFrame[playIndex].rotder) / i_max_steps;
    KeyFrame[playIndex].rotInc2 = (KeyFrame[playIndex + 1].rotizq - KeyFrame[playIndex].rotizq) / i_max_steps;
    KeyFrame[playIndex].rotInc3 = (KeyFrame[playIndex + 1].rotm - KeyFrame[playIndex].rotm) / i_max_steps;
}
```

Inicializamos los frames de la animación.

```
//Inicializacion de los Keyframe para almacenar datos
for (int i = 0; i < MAX_FRAMES; i++)
{
    KeyFrame[i].posX = 0;
    KeyFrame[i].incX = 0;
    KeyFrame[i].incY = 0;
    KeyFrame[i].incZ = 0;
    KeyFrame[i].rotder = 0;
    KeyFrame[i].rotInc = 0;
    KeyFrame[i].rotizq = 0;
    KeyFrame[i].rotInc2 = 0;
    KeyFrame[i].rotm = 0;
    KeyFrame[i].rotInc3 = 0;

}
```

Se cargan sus valores

```
//Valores que toman los Keyframe para animacion
KeyFrame[0].rotder = -15;
KeyFrame[1].rotder = -40;
KeyFrame[2].rotder = -95;
KeyFrame[3].rotder = 0;
KeyFrame[4].rotder = -95;
KeyFrame[5].rotder = -40;
KeyFrame[6].rotder = -15;
KeyFrame[7].rotder = 0;
KeyFrame[0].rotizq = -15;
KeyFrame[1].rotizq = -40;
KeyFrame[2].rotizq = -95;
KeyFrame[3].rotizq = 0;
KeyFrame[4].rotizq = -95;
KeyFrame[5].rotizq = -40;
KeyFrame[6].rotizq = -15;
KeyFrame[7].rotizq = 0;
KeyFrame[0].rotm = 0;
KeyFrame[1].rotm = -90;
KeyFrame[2].rotm = -90;
KeyFrame[3].rotm = -0;
```

Para la carga de modelos utilizamos las siguientes líneas de código.

```
//Carga de modelos realizados externamente
Model casa((char*)"Models/completo/casa1.obj");
Model car((char*)"Models/car/car.obj");
Model ropero((char*)"Models/ropero/ropero.obj");
Model pder((char*)"Models/ropero/pder.obj");
Model pizq((char*)"Models/ropero/pizq.obj");
Model mac1((char*)"Models/mac/mac1.obj");
Model mac2((char*)"Models/mac/mac2.obj");

//Modelo de animación
ModelAnim animacionPersonaje1("animaciones/p1/waving.dae");
animacionPersonaje1.initShaders(animShader.Program);

ModelAnim animacionPersonaje2("animaciones/p2/clap.dae");
animacionPersonaje2.initShaders(animShader.Program);
```

Una vez cargados los modelos los dibujamos de la siguiente manera, cada modelo tiene una definición para dibujarse de manera diferente.

```
//casa
glm::mat4 model1(1);
model1 = glm::translate(model1, glm::vec3(0.0f, -50.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model1));
casa.Draw(lightingShader);

//suelo con primitivas
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texture1);
glBindVertexArray(VAO);
glm::mat4 model2(1);
model2 = glm::translate(model2, glm::vec3(0.0f, -70.0f, 11.5f));
model2 = glm::scale(model2, glm::vec3(500.0f, 0.2f, 500.0f));
model2 = glm::rotate(model2, glm::radians(90.0f), glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model2));
glDrawArrays(GL_TRIANGLES, 0, 36);
```

```
//carro
glm::mat4 model3(1);
model3 = glm::translate(model3, PosIni + glm::vec3(movX, 0.0f, movZ));
model3 = glm::scale(model3, glm::vec3(2.0f));
model3 = glm::rotate(model3, glm::radians(rot), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model3));
car.Draw(lightingShader);

//Ropero
glm::mat4 model4(1);
model4 = glm::translate(model4, glm::vec3(0.0f, -64.0f, -25.0f));
model4 = glm::scale(model4, glm::vec3(7.0f));
model4 = glm::rotate(model4, glm::radians(-90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model4));
ropero.Draw(lightingShader);
```

```
//puerta ropero derecha
glm::mat4 model5(1);
model5 = glm::translate(model5, glm::vec3(8.0f, -64.0f, -18.0f));
model5 = glm::scale(model5, glm::vec3(7.0f));
model5 = glm::rotate(model5, glm::radians(-90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
model5 = glm::rotate(model5, glm::radians(-rotizq), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model5));
pizq.Draw(lightingShader);

//puerta ropero izquierda
glm::mat4 model6(1);
model6 = glm::translate(model6, glm::vec3(-8.0f, -64.0f, -18.0f));
model6 = glm::scale(model6, glm::vec3(7.0f));
model6 = glm::rotate(model6, glm::radians(-90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
model6 = glm::rotate(model6, glm::radians(rotder), glm::vec3(0.0f, 1.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model6));
pder.Draw(lightingShader);
```

```
//Parte arriba Mac
glm::mat4 model7(1);
model7 = glm::translate(model7, glm::vec3(2.0f, -54.5f, 19.0f));
model7 = glm::rotate(model7, glm::radians(-rotm), glm::vec3(1.0f, 0.0f, 0.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model7));
mac1.Draw(lightingShader);

//Parte abajo Mac
glm::mat4 model8(1);
model8 = glm::translate(model8, glm::vec3(2.0f, -54.5f, 19.0f));
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model8));
mac2.Draw(lightingShader);

glBindVertexArray(0);
```

Para las animaciones de los personajes, tienen una definición de dibujado diferente.

```
/*_____________________________Personaje Animado 1_____________________________*/
animShader.Use();
modelLoc = glGetUniformLocation(animShader.Program, "model");
viewLoc = glGetUniformLocation(animShader.Program, "view");
projLoc = glGetUniformLocation(animShader.Program, "projection");

glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));

glUniform3f(glGetUniformLocation(animShader.Program, "material.specular"), 0.5f, 0.5f, 0.5f);
glUniform1f(glGetUniformLocation(animShader.Program, "material.shininess"), 28.0f);
glUniform3f(glGetUniformLocation(animShader.Program, "light.ambient"), 1.0f, 1.0f, 1.0f);
glUniform3f(glGetUniformLocation(animShader.Program, "light.diffuse"), 0.0f, 1.0f, 1.0f);
glUniform3f(glGetUniformLocation(animShader.Program, "light.specular"), 0.5f, 0.5f, 0.5f);
glUniform3f(glGetUniformLocation(animShader.Program, "light.direction"), 0.0f, -1.0f, -1.0f);
view = camera.GetViewMatrix();

model = glm::mat4(1);
model = glm::translate(model, glm::vec3(50.0f, -70.0f, 10.0f));
model = glm::scale(model, glm::vec3(0.07f));    // it's a bit too big for our scene, so scale it down
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
animacionPersonaje1.Draw(animShader);
```

```
/*_____________________________Personaje Animado 2_____________________________*/
animShader.Use();
modelLoc = glGetUniformLocation(animShader.Program, "model");
viewLoc = glGetUniformLocation(animShader.Program, "view");
projLoc = glGetUniformLocation(animShader.Program, "projection");

glUniformMatrix4fv(viewLoc, 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(projLoc, 1, GL_FALSE, glm::value_ptr(projection));

glUniform3f(glGetUniformLocation(animShader.Program, "material.specular"), 0.5f, 0.5f, 0.5f);
glUniform1f(glGetUniformLocation(animShader.Program, "material.shininess"), 28.0f);
glUniform3f(glGetUniformLocation(animShader.Program, "light.ambient"), 1.0f, 1.0f, 1.0f);
glUniform3f(glGetUniformLocation(animShader.Program, "light.diffuse"), 0.0f, 1.0f, 1.0f);
glUniform3f(glGetUniformLocation(animShader.Program, "light.specular"), 0.5f, 0.5f, 0.5f);
glUniform3f(glGetUniformLocation(animShader.Program, "light.direction"), 0.0f, -1.0f, -1.0f);
view = camera.GetViewMatrix();

model = glm::mat4(1);
model = glm::translate(model, glm::vec3(12.0f, -61.0f, 2.0f));
model = glm::rotate(model, glm::radians(-90.0f), glm::vec3(0.0f, 1.0f, 0.0f));
model = glm::scale(model, glm::vec3(0.12f));    // it's a bit too big for our scene, so scale it down
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
animacionPersonaje2.Draw(animShader);
glBindVertexArray(0);
```