UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

LAB. COMPUTACIÓN GRÁFICA E INTERACCIÓN HUMANO-COMPUTADORA

PROF: ING. CARLOS ALDAIR ROMAN BALBUENA

ALUMNO: GUERRERO FERRUSCA FRANCISCO

PROYECTO FINAL: MANUAL TÉCNICO

FECHA DE ENTREGA: 16/ NOVIEMBRE /2021

Objetivo:

Recrear a partir de modelos 3D un espacio que nos permita visualizar en OpenGL una fachada en la cual podremos interactuar con ella a partir de lo aprendido en el curso.

Diagrama de Gantt

ACTIVIDAD	4-8 De octubre	11 al 15 de ocutubre	18 a 22 de octubre	25 a 29 de octubre	1 a 5 de noviembre	8 a 12 de noviembre	15 al 19 de noviembre	21 de noviembre
Idea								
Propuesta								
Selección de objetos								
Entrega de propuesta								
Aprobación de propuesta								
entrega primer objeto								
entrega segundo objeto								
Entrega tercer objeto								
Entrega cuarto objeto								
Entrega quinto objeto								
creación de los objetos faltantes								
Animaciones sencillas								
Animaciones complejas								
integración de todos los objetos en el escenario								
Manual tecnico								
Manual de usuario								
Diagrama de Gant								
Entrega final								
Responsable	Guerrero	Ferrusca	Francisco					

Alcance del proyecto

Este proyecto abarca una parte de la casa que se ve en la serie “Los padrinos mágicos” así como el cuarto del protagonista Timmy Turner y el patio que se presenta en la serie. Además de poder interactuar con algunos de los objetos que se encuentran dentro o fuera de la casa.

Limitaciones

Al no tener un diagrama completo de como esta estructurada la casa, se recreo a partir de lo visto en los episodios, el usuario tiene libertad de ir a donde desee dentro del skybox,

Documentación

Durante la implementación de este proyecto se utilizó código visto a lo largo del semestre a continuación se presenta lo más importante del código que permite que visualmente se vea como en el ejecutable.

```
59      // Keyframes
60      float posX =PosIni.x, posY = PosIni.y, posZ = PosIni.z, rotRodIzq = 0 , rotRodDer, rotBraDer, RotBraIzq;
61
62      #define MAX_FRAMES 9
63      int i_max_steps = 190;
64      int i_curr_steps = 0;
65      typedef struct _frame
66      {
67          //Variables para GUARDAR Key Frames
68          float posX;     //Variable para PosicionX
69          float posY;     //Variable para PosicionY
70          float posZ;     //Variable para PosicionZ
71          float incX;     //Variable para IncrementoX
72          float incY;     //Variable para IncrementoY
73          float incZ;     //Variable para IncrementoZ
74          float rotRodDer;
75          float rotRodIzq;
76          float rotBraDer;
77          float RotBraIzq;
78          float rotInc3;
79          float rotInc4;
80          float rotInc2;
81          float rotInc;
82
83      }FRAME;
```

Figura 1: Declaración de las variables para utilizar Keyframes en la rotación de los objetos.

```
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].rotRodIzq - KeyFrame[playIndex].rotRodIzq) / i_max_steps;
    KeyFrame[playIndex].rotInc2 = (KeyFrame[playIndex + 1].rotRodDer - KeyFrame[playIndex].rotRodDer) / i_max_steps;
    KeyFrame[playIndex].rotInc3 = (KeyFrame[playIndex + 1].rotBraDer - KeyFrame[playIndex].rotBraDer) / i_max_steps;
    KeyFrame[playIndex].rotInc4 = (KeyFrame[playIndex + 1].RotBraIzq - KeyFrame[playIndex].RotBraIzq) / i_max_steps;
}
```

Figura 2: Calculo de interpolaciones.

```
Model escritorio((char*)"models/desk.obj");
Model buro((char*)"models/buro.obj");
Model cama((char*)"models/camacompleta.obj");
Model lamp((char*)"models/lampara.obj");
Model computadora((char*)"models/compu.obj");
Model silla((char*)"models/silla.obj");
Model pecera((char*)"models/pecera.obj");
Model casa((char*)"models/casa.obj");
Model puerta((char*)"models/puerta.obj");
Model agua((char*)"models/agua.obj");
Model tv((char*)"models/tv.obj");
Model arbol((char*)"models/arbol.obj");
Model star((char*)"models/cuadro.obj");
Model castle((char*)"models/castillo.obj");
Model trixie((char*)"models/trixie.obj");
Model vela((char*)"models/vela.obj");
//ANIMACIONES ROBOT
Model PiernaDer((char*)"models/pierna der.obj");
Model PiernaIzq((char*)"models/pierna izq.obj");
Model cuerpo((char*)"models/cuerporobot.obj");
Model BrazoDer((char*)"models/brazo derecho.obj");
Model BrazoIzq((char*)"models/brazo izq.obj");

//Dinosaurio Introducción
Model dino((char*)"models/dinosaurio.obj");
Model bcadino((char*)"models/bocadinosaurio.obj");
```

Figura 3: Carga de modelos.

```
299     GLfloat skyboxVertices[] = {
300         // Positions
301         -1.0f,  1.0f, -1.0f,
302         -1.0f, -1.0f, -1.0f,
303         1.0f, -1.0f, -1.0f,
304         1.0f, -1.0f, -1.0f,
305         1.0f,  1.0f, -1.0f,
306         -1.0f,  1.0f, -1.0f,
307 
308         -1.0f, -1.0f,  1.0f,
309         -1.0f, -1.0f, -1.0f,
310         -1.0f,  1.0f, -1.0f,
311         -1.0f,  1.0f, -1.0f,
312         -1.0f,  1.0f,  1.0f,
313         -1.0f, -1.0f,  1.0f,
314 
315         1.0f, -1.0f, -1.0f,
316         1.0f, -1.0f,  1.0f,
317         1.0f,  1.0f,  1.0f,
318         1.0f,  1.0f,  1.0f,
319         1.0f,  1.0f, -1.0f,
320         1.0f, -1.0f, -1.0f,
321 
322         -1.0f, -1.0f,  1.0f,
323         -1.0f,  1.0f,  1.0f,
324         1.0f,  1.0f,  1.0f,
325         1.0f,  1.0f,  1.0f,
326         1.0f, -1.0f,  1.0f,
327         -1.0f, -1.0f,  1.0f,
328 
329         -1.0f,  1.0f, -1.0f,
330         1.0f,  1.0f, -1.0f,
331         1.0f,  1.0f,  1.0f,
332         1.0f,  1.0f,  1.0f,
333         -1.0f,  1.0f,  1.0f,
334         -1.0f,  1.0f, -1.0f,
```

Figura 4: Vértices para la skybox.

```
    }
}
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
glTexParameteri(GL_TEXTURE_CUBE_MAP, GL_TEXTURE_WRAP_R, GL_CLAMP_TO_EDGE);
glBindTexture(GL_TEXTURE_CUBE_MAP, 0);
```

Figura 5: Mapeo de aristas para simular el ambiente en el cubo.

```
//SkyBox
GLuint skyboxVBO, skyboxVAO;
glGenVertexArrays(1, &skyboxVAO);
glGenBuffers(1,&skyboxVBO);
glBindVertexArray(skyboxVAO);
glBindBuffer(GL_ARRAY_BUFFER, skyboxVBO);
glBufferData(GL_ARRAY_BUFFER, sizeof(skyboxVertices),&skyboxVertices,GL_STATIC_DRAW);
glEnableVertexAttribArray(0);
glVertexAttribPointer(0, 3, GL_FLOAT,GL_FALSE, 3 * sizeof(GLfloat), (GLvoid *)0);

// Load textures
vector<const GLchar*> faces;
faces.push_back("SkyBox/right.tga");
faces.push_back("SkyBox/left.tga");
faces.push_back("SkyBox/top.tga");
faces.push_back("SkyBox/bottom.tga");
faces.push_back("SkyBox/back.tga");
faces.push_back("SkyBox/front.tga");

GLuint cubemapTexture = TextureLoading::LoadCubemap(faces);

glm::mat4 projection = glm::perspective(camera.GetZoom(), (GLfloat)SCREEN_WIDTH / (GLfloat)SCREEN_HEIGHT, 0.1f, 1000.0f);
```

Figura 6: Creación del VAO y VBO así como la carga de texturas al skybox a través de una lista.

```
    // Directional light
    glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.direction"), -0.2f, -1.0f, -0.3f);
    glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.ambient"), 1.0f, 1.0f, 1.0f);
    glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.diffuse"), 0.4f, 0.4f, 0.4f);
    glUniform3f(glGetUniformLocation(lightingShader.Program, "dirLight.specular"), 0.0f, 0.0f, 0.0f);


    // Point light 1
    glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].position"), pointLightPositions[0].x, pointLightPositions[0].y, pointLightPositio
    glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[0].ambient"), 1.0f, 1.0f, 1.0f);
    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, pointLightPositio
    glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].ambient"), 0.05f, 0.05f, 0.05f);
    glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].diffuse"), 0.0f, 0.0f, 0.0f);
    glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[1].specular"), 0.0f, 0.0f, 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, pointLightPositio
    glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].ambient"), 0.05f, 0.05f, 0.05f);
    glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].diffuse"), 0.0f, 0.0f, 0.0f);
    glUniform3f(glGetUniformLocation(lightingShader.Program, "pointLights[2].specular"), 0.0f, 0.0f, 0.0f);
    glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[2].constant"), 1.0f);
    glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[2].linear"), 0.09f);
    glUniform1f(glGetUniformLocation(lightingShader.Program, "pointLights[2].quadratic"), 0.032f);
```

Figura 7: Luces dentro del ambiente.

```
//Carga de modelo
//Escritorio
view = camera.GetViewMatrix();
glm::mat4 model(1);
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
escritorio.Draw(lightingShader);
//buro
view = camera.GetViewMatrix();
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
buro.Draw(lightingShader);
//cama
view = camera.GetViewMatrix();
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
cama.Draw(lightingShader);
//trixie
view = camera.GetViewMatrix();
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
trixie.Draw(lightingShader);
//vela
view = camera.GetViewMatrix();
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
vela.Draw(lightingShader);
//lampara
view = camera.GetViewMatrix();
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
lamp.Draw(lightingShader);
//cuadro
view = camera.GetViewMatrix();
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
star.Draw(lightingShader);
//compu
view = camera.GetViewMatrix();
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
computadora.Draw(lightingShader);

//silla
view = camera.GetViewMatrix();
glUniformMatrix4fv(modelLoc, 1, GL_FALSE, glm::value_ptr(model));
silla.Draw(lightingShader);
```

Figura 8: Carga de objetos recreados.

```
// Draw skybox as last
glDepthFunc(GL_LEQUAL);  // Change depth function so depth test passes when values are equal to depth buffer's content
SkyBoxshader.Use();
view = glm::mat4(glm::mat3(camera.GetViewMatrix()));    // Remove any translation component of the view matrix
glUniformMatrix4fv(glGetUniformLocation(SkyBoxshader.Program, "view"), 1, GL_FALSE, glm::value_ptr(view));
glUniformMatrix4fv(glGetUniformLocation(SkyBoxshader.Program, "projection"), 1, GL_FALSE, glm::value_ptr(projection));

// skybox cube
glBindVertexArray(skyboxVAO);
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_CUBE_MAP, cubemapTexture);
glDrawArrays(GL_TRIANGLES, 0, 36);
glBindVertexArray(0);
glDepthFunc(GL_LESS); // Set depth function back to default
```

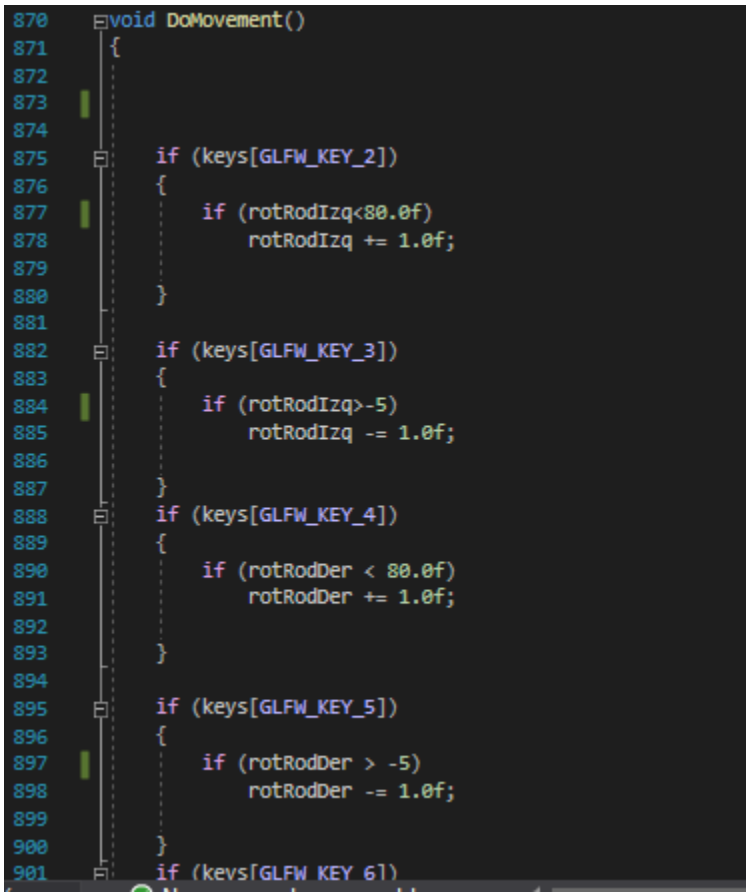
Figura 9: Llamada al shader del skybox.

```
//Movimiento del personaje

if (play)
{
    if (i_curr_steps >= i_max_steps) //end of animation between frames?
    {
        playIndex++;
        if (playIndex>FrameIndex - 2)   //end of total animation?
        {
            printf("termina anim\n");
            playIndex = 0;
            play = false;
        }
        else //Next frame interpolations
        {
            i_curr_steps = 0; //Reset counter
                              //Interpolation
            interpolation();
        }
    }
    else
    {
        //Draw animation
        posX += KeyFrame[playIndex].incX;
        posY += KeyFrame[playIndex].incY;
        posZ += KeyFrame[playIndex].incZ;

        rotRodIzq += KeyFrame[playIndex].rotInc;
        rotRodDer += KeyFrame[playIndex].rotInc2;
        rotBraDer += KeyFrame[playIndex].rotInc3;
        RotBraIzq += KeyFrame[playIndex].rotInc4;
        i_curr_steps++;
    }

}
}
```

Figura 10: Movimiento del personaje

```
791  void KeyCallback(GLFWwindow *window, int key, int scancode, int action, int mode)
792  {
793      if (keys[GLFW_KEY_L])
794      {
795          if (play == false && (FrameIndex > 1))
796          {
797
798              resetElements();
799              //First Interpolation
800              interpolation();
801
802              play = true;
803              playIndex = 0;
804              i_curr_steps = 0;
805          }
806          else
807          {
808              play = false;
809          }
810
811      }
812
813      if (keys[GLFW_KEY_K])
814      {
815          if (FrameIndex<MAX_FRAMES)
816          {
817              saveFrame();
818          }
819
820      }
```

Figura 11: Asignación de las letras L para correr la animación y K para guardar el movimiento.



```
870  void DoMovement()
871  {
872
873
874
875      if (keys[GLFW_KEY_2])
876      {
877          if (rotRodIzq<80.0f)
878              rotRodIzq += 1.0f;
879
880      }
881
882      if (keys[GLFW_KEY_3])
883      {
884          if (rotRodIzq>-5)
885              rotRodIzq -= 1.0f;
886
887      }
888      if (keys[GLFW_KEY_4])
889      {
890          if (rotRodDer < 80.0f)
891              rotRodDer += 1.0f;
892
893      }
894
895      if (keys[GLFW_KEY_5])
896      {
897          if (rotRodDer > -5)
898              rotRodDer -= 1.0f;
899
900      }
901      if (keys[GLFW_KEY_6])
```

Figura 12: Función que nos permite hacer movimientos a los objetos y asignación de teclas para el robot.

```
// Camera controls
if (keys[GLFW_KEY_W] || keys[GLFW_KEY_UP])
{
    camera.ProcessKeyboard(FORWARD, deltaTime);

}

if (keys[GLFW_KEY_S] || keys[GLFW_KEY_DOWN])
{
    camera.ProcessKeyboard(BACKWARD, deltaTime);


}

if (keys[GLFW_KEY_A] || keys[GLFW_KEY_LEFT])
{
    camera.ProcessKeyboard(LEFT, deltaTime);


}

if (keys[GLFW_KEY_D] || keys[GLFW_KEY_RIGHT])
{
    camera.ProcessKeyboard(RIGHT, deltaTime);
}
```

Figura 13: Asignación le teclas al movimiento de la cámara.