2021

Manual Técnico

Proyecto Final

COMPUTACIÓN GRÁFICA E INTERACCIÓN HUMANO COMPUTADORA

Carolina Kennedy Villa

Facultad de Ingeniería,
Universidad Nacional Autónoma de México

Objetivo

Crear y amueblar un cuarto nuevo y recrear nuevos elementos.

Alcance

Recrear y modernizar la casa de Malcolm el de en medio como proyecto final, utilizando todo lo aprendido en la materia de Computación Gráfica como aplicación de texturas y animaciones.

Introducción

Este manual, sirve para la explicación de la parte técnica del proyecto entregado.

Contenido

IDE

Se utiliza del programa Visual Studio Code 2017, el cual es compatible con varios lenguajes de programación y entre ellos C++, el cual es utilizado para este proyecto.

Bibliotecas

Glew: OpenGL Extension Wrangler Library es una biblioteca que ayuda en la carga y consulta de extensiones de OpenGL.

GLFW: Permite crear y dirigir ventanas y aplicaciones con OpenGL.

Iostream: Se utiliza para las operaciones de entrada y salida.

Definición de variables y funciones

Al inicio del programa se encuentra la declaración de funciones que son esenciales para el proyecto, tales como la cámara sintética (se establece el punto inicial de la casa), la luz del escenario (en general), el sonido, las animaciones, o la entrada del programa.

```
//#pragma comment(lib, "winmm.lib")

void framebuffer_size_callback(GLFWwindow* window, int width, int height);
void mouse_callback(GLFWwindow* window, double xpos, double ypos);
void scroll_callback(GLFWwindow* window, double xoffset, double yoffset);
//void my_input(GLFWwindow *window);
void my_input(GLFWwindow* window, int key, int scancode, int action, int mods);
void animate(void);

// settings
const unsigned int SCR_WIDTH = 1920;
const unsigned int SCR_HEIGHT = 1200;

// camera
Camera camera(glm::vec3(90.0f, 25.0f, 420.0f));
float MovementSpeed = 1.5f;
float lastX = SCR_WIDTH / 2.0f;
float lastY = SCR_HEIGHT / 2.0f;
bool firstMouse = true;

// timing
const int FPS = 60;
const int LOOP_TIME = 1000 / FPS; // = 16 milisec // 1000 millisec == 1 sec
double  deltaTime = 0.0f,
        lastFrame = 0.0f;

//Lighting
glm::vec3 lightPosition(20.0f, 30.0f, -200.0f);
glm::vec3 lightDirection(-1.0f, -1.0f, -1.0f);

//Sound
void sound();
```

Dentro de las variables, se inicializan aquellas que utilizaremos sobre todo para la manipulación de las animaciones.

```
float   movBanJx = 0.0f,
        movBanJz = 0.0f,
        movCama1 = 0.0f,
        movCama2 = 0.0f,
        movMueble1 = 0.0f,
        movMueble2 = 0.0f,
        movMueble3 = 0.0f,
        orienta = 180.0f;
bool    animacion = false,
        recorrido1 = true,
        recorrido2 = false,
        recorrido3 = false,
        recorrido4 = false,
        recorrido5 = false,
        recorrido6 = false,
        recorridoBJ1 = false,
        recorridoBJ2 = false,
        animacionBJ = false,
        recorridoBJR1 = false,
        recorridoBJR2 = false,
        animacionBJR = false,
        animacionCM = false,
        recorridoM1 = false,
        recorridoM2 = false,
        recorridoC1 = false,
        recorridoC2 = false,
        recorridoM3 = false,
        animacionCMR = false,
        recorridoMR1 = false,
        recorridoMR2 = false,
        recorridoCR1 = false,
        recorridoCR2 = false,
        recorridoMR3 = false;

float initBottle = 270.0f,
      initBottleY = 20.0f,
      rotateBottle = 0.0f,
      mariY = 13.0f,
      mariZ = 25.0f;
//Keyframes (Manipulación y dibujo)
float   movAuto_x = 0.0f,
        movAuto_y = 0.0f,
        movAuto_z = 0.0f,
        giroAuto_y = 0.0f;
float   incX = 0.0f,
        incY = 0.0f,
        incZ = 0.0f,
        rotInc = 0.0f,
        giroAuto_yInc = 0.0f;
```

Las siguientes funciones y estructura, nos sirven para realizar la animación del coche, la cual está hecha por KeyFrames. Aquí establecemos el número máximo de KeyFrames y los pasos intermedios por los que pasa nuestra animación.

```
#define MAX_FRAMES 9
int i_max_steps = 60;
int i_curr_steps = 0;
typedef struct _frame
{
    //Variables para GUARDAR Key Frames
    float movAuto_x;        //Variable para PosicionX
    float movAuto_y;        //Variable para PosicionY
    float movAuto_z;        //Variable para PosicionY
    float giroAuto_y;       //Variable para PosicionZ

}FRAME;

FRAME KeyFrame[MAX_FRAMES];
int FrameIndex = 5;         //introducir datos
bool play = false;
int playIndex = 0;

void saveFrame(void)
{
    //printf("frameindex %d\n", FrameIndex);
    std::cout << "Frame Index = " << FrameIndex << std::endl;
    std::cout << "X = " << movAuto_x << std::endl;
    std::cout << "Y = " << movAuto_y << std::endl;
    std::cout << "Z = " << movAuto_z << std::endl;
    std::cout << "GIRO Y = " << giroAuto_y << std::endl;

    KeyFrame[FrameIndex].movAuto_x = movAuto_x;
    KeyFrame[FrameIndex].movAuto_y = movAuto_y;
    KeyFrame[FrameIndex].movAuto_z = movAuto_z;
    KeyFrame[FrameIndex].giroAuto_y = giroAuto_y;

    FrameIndex++;
}

void resetElements(void)
{
    movAuto_x = KeyFrame[0].movAuto_x;
    movAuto_y = KeyFrame[0].movAuto_y;
    movAuto_z = KeyFrame[0].movAuto_z;
    giroAuto_y = KeyFrame[0].giroAuto_y;
}
```

También para los KeyFrames, se utiliza la función de interpolación, la cual nos ayudará para obtener los Frames intermedios entre cada uno de los que guardamos.

```
void interpolation(void)
{
    incX = (KeyFrame[playIndex + 1].movAuto_x - KeyFrame[playIndex].movAuto_x) / i_max_steps;
    incY = (KeyFrame[playIndex + 1].movAuto_y - KeyFrame[playIndex].movAuto_y) / i_max_steps;
    incZ = (KeyFrame[playIndex + 1].movAuto_z - KeyFrame[playIndex].movAuto_z) / i_max_steps;
    giroAuto_yInc = (KeyFrame[playIndex + 1].giroAuto_y - KeyFrame[playIndex].giroAuto_y) / i_max_steps;

}
```

La siguiente función, sirve para reproducir de manera automática el audio que se escucha en todo el proyecto.

```
// Sound
void sound() {
    if (soundon) {
        bool played = PlaySound("birds.wav", NULL, SND_LOOP | SND_ASYNC);
        cout << "Ambient:" << played << endl;
        soundon = false;
    }
}
```

La función anímate, contiene las 5 animaciones realizadas:

La primera es del coche elaborado por KeyFrames:

```
void animate(void)
{
    if (play)
    {
        if (i_curr_steps >= i_max_steps) //end of animation between frames?
        {
            playIndex++;
            if (playIndex > FrameIndex - 2) //end of total animation?
            {
                std::cout << "Animation ended" << std::endl;
                //printf("termina anim\n");
                playIndex = 0;
                play = false;
            }
            else //Next frame interpolations
            {
                i_curr_steps = 0; //Reset counter
                                  //Interpolation
                interpolation();
            }
        }
        else
        {
            //Draw animation
            movAuto_x += incX;
            movAuto_y += incY;
            movAuto_z += incZ;
            giroAuto_y += giroAuto_yInc;

            i_curr_steps++;
        }
    }
```

Las demás animaciones, se realizaron por estados:

```
//Botella
if (activeAnim)
{
    if (initBottle <= 275.0f)
    {
        initBottle += 0.1;
        if (rotateBottle <= 50.0f)
            rotateBottle += 1.0;
    }
    if (initBottleY >= -5.0f) {
        initBottleY -= 1.0f;
        if (initBottleY <= -6.0f)
            displayGlass = true;
    }
}

//////Mariposa
if (activeAnimM) {

    if (mari1) {
        mariY += 1.0f;
        mariZ -= 0.8f;
        if (mariY >= 14.0f) {
            mari2 = true;
            mari1 = false;
        }
    }
    if (mari2) {
        mariY -= 1.0f;
        if (mariY <= -6.0f) {
            mari1 = true;
            mari2 = false;
        }
    }

}
```

```
//Animación Banco de Jaimie
if (animacionBJ)//x = 14.0f, z=-24.5f
{
    if (recorridoBJ1)
    {
        movBanJz += 1.0f;
        if (movBanJz > 20.0f)
        {
            recorridoBJ1 = false;
            recorridoBJ2 = true;

        }
    }

    if (recorridoBJ2)
    {
        movBanJx += 1.0f;
        if (movBanJx > 13.0f)
        {
            recorridoBJ2 = false;
        }

    }
}
//Animación Banco de Jaimie Regreso a la Cuna
if (animacionBJR)//x = 14.0f, z=-24.5f
{
    if (recorridoBJR1)
    {
        movBanJx -= 1.0f;
        if (movBanJx < 0.0f)
        {
            recorridoBJR1 = false;
            recorridoBJR2 = true;

        }
    }

    if (recorridoBJR2)
    {
        movBanJz -= 1.0f;
        if (movBanJz < 0.0f)
        {
            recorridoBJR2 = false;
        }

    }
```

```
//Animación de mover los muebles
if (animacionCM)
{
    if (recorridoM1)
    {
        movMueble1 -= 1.0f;
        if (movMueble1 < -40.0f)
        {
            recorridoM1 = false;
            recorridoC1 = true;

        }
    }

    if (recorridoC1)
    {
        movCama1 -= 1.0f;
        if (movCama1 < -37.0f)
        {
            recorridoC1 = false;
            recorridoM2 = true;
        }

    }

    if (recorridoM2)
    {
        movMueble2 -= 1.0f;
        if (movMueble2 < -40.0f)
        {
            recorridoM2 = false;
            recorridoC2 = true;

        }
    }

    if (recorridoC2)
    {
        movCama2 -= 1.0f;
        if (movCama2 < -37.0f)
        {
            recorridoC2 = false;
            recorridoM3 = true;

        }

    }
    if (recorridoM3)
    {
```

```
//Animación de mover los muebles de regreso
if (animacionCMR)
{
    if (recorridoMR1)
    {
        movMueble1 += 1.0f;
        if (movMueble1 > 0.0f)
        {
            recorridoMR1 = false;
            recorridoCR2 = true;

        }
    }

    if (recorridoCR1)
    {
        movCama1 += 1.0f;
        if (movCama1 > 0.0f)
        {
            recorridoCR1 = false;
            recorridoMR1 = true;
        }

    }

    if (recorridoMR2)
    {
        movMueble2 += 1.0f;
        if (movMueble2 > 0.0f)
        {
            recorridoMR2 = false;
            recorridoMR3 = true;

        }
    }

    if (recorridoCR2)
    {
        movCama2 += 1.0f;
        if (movCama2 > 0.0f)
        {
            recorridoCR2 = false;
            recorridoMR2 = true;

        }

    }
    if (recorridoMR3)
```

Dentro de la función de main, se crean todos los objetos utilizados en el proyecto así como la posición donde se dibujarán, inicializamos la función animate, las posiciones de los KeyFrames y la configuración de la luz en el escenario :

```
int main()
{
    // glfw: initialize and configure
    //
    glfwInit();
    /*glfwWindowHint(GLFW_CONTEXT_VERSION_MAJOR, 3);
    glfwWindowHint(GLFW_CONTEXT_VERSION_MINOR, 3);
    glfwWindowHint(GLFW_OPENGL_PROFILE, GLFW_OPENGL_CORE_PROFILE);*/

#ifdef __APPLE__
    glfwWindowHint(GLFW_OPENGL_FORWARD_COMPAT, GL_TRUE);
#endif

    // glfw window creation
    GLFWwindow* window = glfwCreateWindow(SCR_WIDTH, SCR_HEIGHT, "Proyecto Final", NULL, NULL);
    if (window == NULL)
    {
        std::cout << "Failed to create GLFW window" << std::endl;
        glfwTerminate();
        return -1;
    }
    glfwMakeContextCurrent(window);
    glfwSetFramebufferSizeCallback(window, framebuffer_size_callback);
    glfwSetCursorPosCallback(window, mouse_callback);
    glfwSetScrollCallback(window, scroll_callback);
    glfwSetKeyCallback(window, my_input);

    // tell GLFW to capture our mouse
    glfwSetInputMode(window, GLFW_CURSOR, GLFW_CURSOR_DISABLED);

    // glad: load all OpenGL function pointers
    // ---------------------------------------
    if (!gladLoadGLLoader((GLADloadproc)glfwGetProcAddress))
    {
        std::cout << "Failed to initialize GLAD" << std::endl;
        return -1;
    }

    // configure global opengl state
    // -----------------------------
    glEnable(GL_DEPTH_TEST);

    // build and compile shaders
    // -------------------------
    //Shader staticShader("Shaders/lightVertex.vs", "Shaders/lightFragment.fs");
    Shader staticShader("Shaders/shader_Lights.vs", "Shaders/shader_Lights.fs");
    Shader skyboxShader("Shaders/skybox.vs", "Shaders/skybox.fs");
```

```
animate();

// render
// ------
glClearColor(0.3f, 0.3f, 0.3f, 1.0f);
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

// don't forget to enable shader before setting uniforms
staticShader.use();
//Setup Advanced Lights
staticShader.setVec3("viewPos", camera.Position);
staticShader.setVec3("dirLight.direction", lightDirection);
staticShader.setVec3("dirLight.ambient", glm::vec3(0.2f, 0.2f, 0.2f));
staticShader.setVec3("dirLight.diffuse", glm::vec3(1.0f, 1.0f, 1.0f));
staticShader.setVec3("dirLight.specular", glm::vec3(0.0f, 0.0f, 0.0f));

staticShader.setVec3("pointLight[0].position", lightPosition);
staticShader.setVec3("pointLight[0].ambient", glm::vec3(0.0f, 0.0f, 0.0f));
staticShader.setVec3("pointLight[0].diffuse", glm::vec3(0.0f, 0.0f, 0.0f));
staticShader.setVec3("pointLight[0].specular", glm::vec3(0.0f, 0.0f, 0.0f));
staticShader.setFloat("pointLight[0].constant", 0.8f);
staticShader.setFloat("pointLight[0].linear", 0.009f);
staticShader.setFloat("pointLight[0].quadratic", 0.032f);

staticShader.setVec3("pointLight[1].position", glm::vec3(0.0, 0.0f, 0.0f));
staticShader.setVec3("pointLight[1].ambient", glm::vec3(0.0f, 0.0f, 0.0f));
staticShader.setVec3("pointLight[1].diffuse", glm::vec3(0.0f, 0.0f, 0.0f));
staticShader.setVec3("pointLight[1].specular", glm::vec3(0.0f, 0.0f, 0.0f));
staticShader.setFloat("pointLight[1].constant", 1.0f);
staticShader.setFloat("pointLight[1].linear", 0.009f);
staticShader.setFloat("pointLight[1].quadratic", 0.032f);

staticShader.setFloat("material_shininess", 32.0f);
```

```
Model salaG("resources/objects/Kennedy/salaGrande.obj");
Model muebleTV("resources/objects/Kennedy/muebleTV.obj");
Model mesaR("resources/objects/Kennedy/mesa.obj");
Model sillaSala("resources/objects/Kennedy/sillaSala.obj");
Model sillon("resources/objects/Kennedy/sillon.obj");
Model libreroS("resources/objects/Kennedy/librero.obj");
Model cuna("resources/objects/Kennedy/cuna.obj");
Model cambiador("resources/objects/Kennedy/cambiadorP.obj");
Model oso("resources/objects/Kennedy/oso.obj");
Model ropero("resources/objects/Kennedy/ropero.obj");
Model estufaJ("resources/objects/Kennedy/estufaJ.obj");
Model techo("resources/objects/Kennedy/techo.obj");
//ModelAnim animacionPersonaje("resources/objects/Personaje1/PersonajeBrazo.dae");
//animacionPersonaje.initShaders(animShader.ID);

//Inicialización de KeyFrames
//for (int i = 0; i < MAX_FRAMES; i++)
//{
//  KeyFrame[i].movAuto_x = 0;
//  KeyFrame[i].movAuto_y = 0;
//  KeyFrame[i].movAuto_z = 0;
//  KeyFrame[i].giroAuto_y = 0;
//}

KeyFrame[0].movAuto_x = -150;
KeyFrame[0].movAuto_y = 0;
KeyFrame[0].movAuto_z = 0;
KeyFrame[0].giroAuto_y = 0;

KeyFrame[1].movAuto_x = 310;
KeyFrame[1].movAuto_y = 40;
KeyFrame[1].movAuto_z = 0;
KeyFrame[1].giroAuto_y = 0;

KeyFrame[2].movAuto_x = 850;
KeyFrame[2].movAuto_y = 0;
KeyFrame[2].movAuto_z = 0;
KeyFrame[2].giroAuto_y = 0;

KeyFrame[3].movAuto_x = 850;
KeyFrame[3].movAuto_y = 0;
KeyFrame[3].movAuto_z = 0;
KeyFrame[3].giroAuto_y = 90;
```

```
/*---------------- EXTERIOR ----------------*/
// PASTO
model = glm::mat4(1.0f);
model = glm::translate(model, glm::vec3(0.0f, -0.5f, 0.0f));
model = glm::scale(model, glm::vec3(2.0f, 1.0f, 2.0f));
staticShader.setMat4("model", model);
pasto.Draw(staticShader);

// BARDA EXTERIOR
model = glm::translate(glm::mat4(1.0f), glm::vec3(-200.0f, 0.5f, 0.0f));
model = glm::scale(model, glm::vec3(1000.0f, 1000.0f, 1000.0f));
staticShader.setMat4("model", model);
barda_exterior.Draw(staticShader);

model = glm::translate(glm::mat4(1.0f), glm::vec3(-100.0f, 0.5f, 0.0f));
model = glm::scale(model, glm::vec3(1000.0f, 1000.0f, 1000.0f));
staticShader.setMat4("model", model);
barda_exterior.Draw(staticShader);

model = glm::translate(glm::mat4(1.0f), glm::vec3(0.0f, 0.5f, 0.0f));
model = glm::scale(model, glm::vec3(1000.0f, 1000.0f, 1000.0f));
staticShader.setMat4("model", model);
barda_exterior.Draw(staticShader);

   //Sillon
   model = glm::rotate(glm::mat4(1.0f), glm::radians(270.0f), glm::vec3(0.0f, 1.0f, 0.0));
   model = glm::translate(model, glm::vec3(400.0f, 10.0f, -245.f));
   model = glm::scale(model, glm::vec3(0.7f,0.8f,0.7f));
   staticShader.setMat4("model", model);
   sillon.Draw(staticShader);

   //Sofá grande Sala
   model = glm::rotate(glm::mat4(1.0f), glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0));
   model = glm::translate(model, glm::vec3(-420.0f, 10.0f, 210.f));
   model = glm::scale(model, glm::vec3(1.0f));
   staticShader.setMat4("model", model);
   salaG.Draw(staticShader);

   //Mueble TV

   model = glm::translate(model, glm::vec3(-27.0f, 0.1f, -25.f));
   model = glm::rotate(model, glm::radians(90.0f), glm::vec3(0.0f, 1.0f, 0.0));
   model = glm::scale(model, glm::vec3(0.8f));
   staticShader.setMat4("model", model);
   muebleTV.Draw(staticShader);

   //Mesa Sala
   model = glm::translate(glm::mat4(1.0f), glm::vec3(190.0f, 18.0f, 260.f));
   model = glm::scale(model, glm::vec3(0.7f,0.72f,0.7f));
   staticShader.setMat4("model", model);
   mesaR.Draw(staticShader);

   //Silla Mesa Sala
   model = glm::scale(glm::mat4(1.0f), glm::vec3(1.0f,1.2f,1.0f));
   model = glm::translate(model, glm::vec3(185.0f, 8.0f, 240.f));
   staticShader.setMat4("model", model);
   sillaSala.Draw(staticShader);

   model = glm::scale(glm::mat4(1.0f), glm::vec3(1.0f, 1.2f, 1.0f));
   model = glm::translate(model, glm::vec3(175.0f, 8.0f, 260.f));
   model = glm::rotate(model, glm::radians(75.0f), glm::vec3(0.0f, 1.0f, 0.0));
   staticShader.setMat4("model", model);
   sillaSala.Draw(staticShader);

   model = glm::scale(glm::mat4(1.0f), glm::vec3(1.0f, 1.2f, 1.0f));
   model = glm::translate(model, glm::vec3(205.0f, 8.0f, 260.f));
   model = glm::rotate(model, glm::radians(-85.0f), glm::vec3(0.0f, 1.0f, 0.0));
   staticShader.setMat4("model", model);
   sillaSala.Draw(staticShader);
```

Por último, la función my_input, es importante, pues es donde establecemos las teclas que utilizaremos para las animaciones:

```
void my_input(GLFWwindow *window, int key, int scancode, int action, int mode)
{
    if (glfwGetKey(window, GLFW_KEY_ESCAPE) == GLFW_PRESS)
        glfwSetWindowShouldClose(window, true);
    //Camera movement
    if (glfwGetKey(window, GLFW_KEY_W) == GLFW_PRESS)
        camera.ProcessKeyboard(FORWARD, (float)deltaTime);
    if (glfwGetKey(window, GLFW_KEY_S) == GLFW_PRESS)
        camera.ProcessKeyboard(BACKWARD, (float)deltaTime);
    if (glfwGetKey(window, GLFW_KEY_A) == GLFW_PRESS)
        camera.ProcessKeyboard(LEFT, (float)deltaTime);
    if (glfwGetKey(window, GLFW_KEY_D) == GLFW_PRESS)
        camera.ProcessKeyboard(RIGHT, (float)deltaTime);
    //To Configure Model

    //Charger
    if (glfwGetKey(window, GLFW_KEY_G) == GLFW_PRESS)
        movAuto_x +=10;
    if (glfwGetKey(window, GLFW_KEY_H) == GLFW_PRESS)
        movAuto_x -=10;
    if (glfwGetKey(window, GLFW_KEY_J) == GLFW_PRESS)
        movAuto_y += 10;
    if (glfwGetKey(window, GLFW_KEY_K) == GLFW_PRESS)
        movAuto_y -= 10;
    if (glfwGetKey(window, GLFW_KEY_T) == GLFW_PRESS)
        movAuto_z += 10;
    if (glfwGetKey(window, GLFW_KEY_Y) == GLFW_PRESS)
        movAuto_z -= 10;
    if (glfwGetKey(window, GLFW_KEY_U) == GLFW_PRESS)
        giroAuto_y += 10;
    if (glfwGetKey(window, GLFW_KEY_I) == GLFW_PRESS)
        giroAuto_y -= 10;

    //Botella
    if (glfwGetKey(window, GLFW_KEY_M) == GLFW_PRESS) {
        activeAnim = true;
        initBottle++;
        initBottleY--;
        rotateBottle++;
    }
    //Mariposa
    if (glfwGetKey(window, GLFW_KEY_N) == GLFW_PRESS) {
        activeAnimM = true;
    }
```