Renata Castro Olmos

S22019640

Ingeniería Informática

Facultad de Ingeniería Eléctrica y Electrónica

Universidad veracruzana

Docente: Yuliana Berumen Diaz

Graficación por computadora

Proyecto final

09 de diciembre del 2025

Veracruz, Boca del Rio

Contenido

Introducción

En este código se verá la implementación de todo lo visto a lo largo del semestre de la experiencia educativa de Graficación por Computadora como las funciones primitivas de OpenGL (glTranslate, glRotate, glLines, glColor3f, etc) y con los temas de estructura de datos, por ejemplo, colas, pilas , listas y árboles.

Con el objetivo de lograr demostrar mis capacidades en la programación de una animación con un personaje articulado.

Requisitos

- Tener OpenGL en tu dispositivo.
- Memoria suficiente para poder descargar el proyecto.
- Saber ejecutarlo en consola

Diseño

Estructuras de Datos Utilizadas

- Árbol jerárquico
- Lista enlazada
- Cola

Algoritmos Principales

- Interpolación Lineal de Keyframes
- Recorrido de Árbol

Implementación

Librerías

```
#include <GL/glut.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <windows.h>
```

#include <GL/glut.h>: proporciona las funciones para crear ventanas y manjear evntos del teclasm mouse y dibujar formas en Opengl.

#include <GL/glut.h>: Proporciona funciones como sin(), cos(), sqrt() que son fundamentales para calcular rotaciones y posiciones de los personajes.

#include <stdio.h>: básica de c , para printf.

#include <stdlib.h>: proporciona funciones para gestión de memoria dinámica (malloc, free).

#include <string.h>: Incluye la librería de manejo de cadenas de texto. Proporciona funciones como strcpy() y strcmp() para copiar y comparar strings.

Código

Se define el formato RGB usado para cargar las texturas BMP correctamente.

```
#ifndef GL_BGR_EXT
#define GL_BGR_EXT 0x80E0
#endif
```

Define la constante PI para los cálculos trigonométricos que se necesiten.

Se definen los factores de conversión de radianes a grados y de grados a radianes para los cálculos de los ángulos.

```
#define PI 3.14159265359
#define RAD_TO_DEG (180.0/PI)
#define DEG_TO_RAD (PI/180.0)
```

Se definen los ángulos máximos y mínimos de cada extremidad en grados.

```
#define MUSLO_MIN 1.67
#define MUSLO_MAX 87.62
#define RODILLA_MIN -78.54
#define RODILLA_MAX 1.67
#define BRAZO_MIN 25.99
#define BRAZO_MAX 169.23
#define ANTEBRAZO_MIN 25.99
#define ANTEBRAZO_MAX 82.17
```

Estructura de Nodo de nuestro árbol, que será nuestras articulaciones, contando con los hijo y hermanos,

```
typedef struct Nodo {
    char nombre[50];
    double tx, ty;
    double angulo;
    double longitud;
    struct Nodo* padre;
    struct Nodo* primerHijo;
    struct Nodo* siguienteHermano;
} Nodo;
```

Estructura para el personaje principal de nuestra animación en mi caso es una mujer, cuenta con la posición , tamaño, colores de todo lo que porta y los ángulos correspondientes, nuestro nodos raíz que en este caso es el torso y lo correspondiente a las animaciones.

```
typedef struct Personaje {
    char nombre[50];
    double posX, posY;            // la pos del per
    double escala;                // tamaño del per
    float colorVestido[3];
    float colorVestidoOscuro[3];
    float colorCuello[3];
    double anguloBrazoDer, anguloAntebrazoDer;
    double anguloBrazoIzq, anguloAntebrazoIzq;
    double anguloMusloDer, anguloPantorrillaDer;
    double anguloMusloIzq, anguloPantorrillaIzq;
    Nodo* torso;                  // Raiz del árbol

    float tiempoAnimacion;        // Tiempo acumula
    int tipoAnimacion;            // es q tengo pen
    int animando;                 // 1=animando, 0=
} Personaje;
```

Estructura para cada frame que va a tener nuestra animación, aquí se forma la lista enlazada, ya que una frame se une con otro frame , almacena la posición y ángulos del personaje.

```
typedef struct Keyframe {
    //el moment en seg donde desbe
    float tiempo;

    // BRAZOS
    double anguloBrazoDer;
    double anguloAntebrazoDer;
    double anguloBrazoIzq;
    double anguloAntebrazoIzq;
    // PIERNAS
    double anguloMusloDer;
    double anguloPantorrillaDer;
    double anguloMusloIzq;
    double anguloPantorrillaIzq;

    // POS DEL PERSONAJE COMPLETO
    float posX;
    float posY;

    struct Keyframe* siguiente;
} Keyframe;
```

Este es el conjunto de los frames que hicimos , es como el libro en el cual se va todo el conjunto de fotos para la animación.

```
typedef struct AnimacionLista{
    Keyframe* primerKeyframe;
    Keyframe* keyframeActual;
    int numKeyframes;
    float tiempoAnimacion;
} AnimacionLista;
```

Estructura para hacer la cola de las escenas, es solo una parte, pero con información de cuanto va a durar la escena , que numero es y a que escena se une.

```
typedef struct Escena {
    int numero;
    float duracionInicio;
    float duracionFin;
    char nombre[50];
    void (*renderizar)(void);
    struct Escena* siguiente;
} Escena;
```

Cola de escenas porque van de la primera a la última, se identifica la primera y la ultima para poder ir insertando escenas en orden.

```
typedef struct ColaEscenas{
    Escena* frente;
    Escena* final;
    int totalEscenas;
    float tiempoTotal;
} ColaEscenas;
```

Para los fondes de cada escena implemente una función para poder cargar la textura.

```
typedef struct Textura{
    // Es un array unidime
    unsigned char* data;
    int width;      //anch
    int height;     //alto
} Textura;
```

Vectores de las partes del cuerpo del personaje para saber la dirección y la longitud de cualquier extremidad, también para saber la ubicación de las articulaciones.

```
double centroTorso[2] = {0.0, 0.069};
double hombroDer[2] = {0.12, 0.2};
//vector de dirección y longitud, a donde apunta y qu
double vectorBrazoDer[4] = {0.02, -0.16, 0, 1};
double vectorAntebrazoDer[4] = {0.06, -0.14, 0, 1};
double hombroIzq[2] = {-0.12, 0.2};
double vectorBrazoIzq[4] = {-0.02, -0.16, 0, 1};
double vectorAntebrazoIzq[4] = {-0.06, -0.14, 0, 1};
double inicioMusloDer[2] = {0.121, -0.44};
double vectorMusloDer[4] = {0.0, -0.16, 0, 1};
double vectorPantorrillaDer[4] = {0.0, -0.12, 0, 1};
double inicioMusloIzq[2] = {-0.080, -0.44};
double vectorMusloIzq[4] = {0.0, -0.16, 0, 1};
double vectorPantorrillaIzq[4] = {0.0, -0.12, 0, 1};
```

De ahí se declaran variables globales, prototipos de función y cargan las texturas.

```
// Limita un ángulo al rango permitido
double limitarAngulo(double angulo, double min, double max) {
    double angulo_grados = angulo * RAD_TO_DEG;
    if (angulo_grados < min) angulo_grados = min;
    if (angulo_grados > max) angulo_grados = max;
    return angulo_grados * DEG_TO_RAD;
}
```

Esta función hace un ciclo for calculando vértices para unirlos y formar un circulo.

```
void dibujarCirculo(float x, float y, float radio, int segmentos) {
    glBegin(GL_POLYGON);
    for(int i = 0; i < segmentos; i++) {
        float angulo = (2.0 * PI * i) / segmentos;//Se calcu el ang
        //se convierten los ang a coor normales
        float px = x + radio * cos(angulo);
        float py = y + radio * sin(angulo);
        glVertex2f(px, py);
    }
    glEnd();
}
```

Esta funcion declara los 2 puntos para trasar una linea uniendo dichos puntos. Es mas que nada para ahorrar lineas de codigo.

```
void dibujarLinea(float x1, float y1, float x2, float y2) {
    glBegin(GL_LINES);
        glVertex2f(x1, y1);
        glVertex2f(x2, y2);
    glEnd();
}
```

Dibuja el polígono basándose en el número de vértices que se llame en los argumentos de la función.

```
void dibujarPoligono(float vertices[][2], int numVertices) {
    glBegin(GL_POLYGON);
    for(int i = 0; i < numVertices; i++) {
        glVertex2f(vertices[i][0], vertices[i][1]);
    }
    glEnd();
}
```

Se crea el nodo, necesario para poder armar y unir nuestro árbol, este serán las articulaciones de mi personaje, pueden ser hijo, padre o hermano.

```
Nodo* crearNodo(char* nombre, double tx, double ty, double longitud, double angulo) {
    Nodo* nuevo = (Nodo*)malloc(sizeof(Nodo));
    strcpy(nuevo->nombre, nombre);// Copia la cadena 'nombre' dentro del campo nombre d
    nuevo->tx = tx;
    nuevo->ty = ty;
    nuevo->angulo = angulo;
    nuevo->longitud = longitud;
    nuevo->padre = NULL;
    nuevo->primerHijo = NULL;
    nuevo->siguienteHermano = NULL;
    return nuevo;
}
```

Se agrega el hijo al padre o al hijo, corroborando en que posición se encuentra, si es el primer nodo de todo el árbol, si ya hay hijos o si hay hermanos.

```
void agregarHijo(Nodo* padre, Nodo* hijo) {
    hijo->padre = padre;                //establece
    if (padre->primerHijo == NULL) {    //padre sin
        padre->primerHijo = hijo;
    } else {
        Nodo* hermano = padre->primerHijo;//recorre
        //
        while (hermano->siguienteHermano != NULL) {
            hermano = hermano->siguienteHermano;
        }
        hermano->siguienteHermano = hijo;//añade hi
    }
}
```

Se van liberando los nodos no necesarios.

```
void liberarArbol(Nodo* nodo) {
    if (nodo == NULL) return;
    liberarArbol(nodo->primerHijo);
    liberarArbol(nodo->siguienteHermano);
    free(nodo);
}
```

Se construyen los personajes, con las articulaciones, calculando la longitud de los segmentos con la distancia euclideana.

```
void construirArbolPersonaje(Personaje* p) {
    // calcular longitudes de cada segmento, distancia euclideana , gracias doc Manir
    double longBrazoDer = sqrt(vectorBrazoDer[0]*vectorBrazoDer[0] + vectorBrazoDer[1]*vectorBrazoDer[1]);
    double longAntebrazoDer = sqrt(vectorAntebrazoDer[0]*vectorAntebrazoDer[0] + vectorAntebrazoDer[1]*vectorAntebrazoDer[1]);
    double longBrazoIzq = sqrt(vectorBrazoIzq[0]*vectorBrazoIzq[0] + vectorBrazoIzq[1]*vectorBrazoIzq[1]);
    double longAntebrazoIzq = sqrt(vectorAntebrazoIzq[0]*vectorAntebrazoIzq[0] + vectorAntebrazoIzq[1]*vectorAntebrazoIzq[1]);
    double longMusloDer = sqrt(vectorMusloDer[0]*vectorMusloDer[0] + vectorMusloDer[1]*vectorMusloDer[1]);
    double longPantorrillaDer = sqrt(vectorPantorrillaDer[0]*vectorPantorrillaDer[0] + vectorPantorrillaDer[1]*vectorPantorrillaDer[1]);
    double longMusloIzq = sqrt(vectorMusloIzq[0]*vectorMusloIzq[0] + vectorMusloIzq[1]*vectorMusloIzq[1]);
    double longPantorrillaIzq = sqrt(vectorPantorrillaIzq[0]*vectorPantorrillaIzq[0] + vectorPantorrillaIzq[1]*vectorPantorrillaIzq[1]);

    // RAÍZ: Torso
    p->torso = crearNodo("Torso", centroTorso[0], centroTorso[1], 0, 0);

    // BRAZO DERECHO: Torso - Brazo - Antebrazo - Mano
    Nodo* brazoDer = crearNodo("Brazo Derecho", hombroDer[0], hombroDer[1], longBrazoDer, p->anguloBrazoDer);
    agregarHijo(p->torso, brazoDer);
    Nodo* antebrazoDer = crearNodo("Antebrazo Derecho", 0, -longBrazoDer, longAntebrazoDer, p->anguloAntebrazoDer);
    agregarHijo(brazoDer, antebrazoDer);
    Nodo* manoDer = crearNodo("Mano Derecha", 0, -longAntebrazoDer, 0, 0);
    agregarHijo(antebrazoDer, manoDer);
```

Funciona para sincronizar los ángulos del personaje con los del árbol.

```
void actualizarAngulosEnNodos(Nodo* nodo, Personaje* p) {
    if (nodo == NULL) return;

    // Actualizar según el nombre del nodo
    if (strcmp(nodo->nombre, "Brazo Derecho") == 0)
        nodo->angulo = limitarAngulo(p->anguloBrazoDer, BRAZO_MIN, BRAZO_MAX);
    else if (strcmp(nodo->nombre, "Antebrazo Derecho") == 0)
        nodo->angulo = limitarAngulo(p->anguloAntebrazoDer, ANTEBRAZO_MIN, ANTEBRAZO_MAX);
    else if (strcmp(nodo->nombre, "Brazo Izquierdo") == 0)
        nodo->angulo = limitarAngulo(p->anguloBrazoIzq, BRAZO_MIN, BRAZO_MAX);
    else if (strcmp(nodo->nombre, "Antebrazo Izquierdo") == 0)
        nodo->angulo = limitarAngulo(p->anguloAntebrazoIzq, ANTEBRAZO_MIN, ANTEBRAZO_MAX);
    else if (strcmp(nodo->nombre, "Pierna Derecha") == 0)
        nodo->angulo = limitarAngulo(p->anguloMusloDer, MUSLO_MIN, MUSLO_MAX);
    else if (strcmp(nodo->nombre, "Pantorrilla Derecha") == 0)
        nodo->angulo = limitarAngulo(p->anguloPantorrillaDer, RODILLA_MIN, RODILLA_MAX);
    else if (strcmp(nodo->nombre, "Pierna Izquierda") == 0)
        nodo->angulo = limitarAngulo(p->anguloMusloIzq, MUSLO_MIN, MUSLO_MAX);
    else if (strcmp(nodo->nombre, "Pantorrilla Izquierda") == 0)
        nodo->angulo = limitarAngulo(p->anguloPantorrillaIzq, RODILLA_MIN, RODILLA_MAX);

    // Continuar con hijos y hermanos
    actualizarAngulosEnNodos(nodo->primerHijo, p);
    actualizarAngulosEnNodos(nodo->siguienteHermano, p);
}
```

```
void actualizarAngulosPersonaje(Personaje* p) {
    actualizarAngulosEnNodos(p->torso, p);
}
```

Dibuja el torso y falda del vestido basandose en poligonos.

```
void dibujarVestidoPersonaje(Personaje* p) {
    glColor3f(p->colorVestido[0], p->colorVestido[1], p->colorVestido[2]);

    float torsoSuperior[][2] = {
        {-0.12 + centroTorso[0], 0.2 + centroTorso[1]},
        {0.12 + centroTorso[0], 0.2 + centroTorso[1]},
        {0.0808656432054 + centroTorso[0], -0.0498050953935 + centroTorso[1]},
        {-0.0791019006464 + centroTorso[0], -0.0504133370051 + centroTorso[1]}
    };
    dibujarPoligono(torsoSuperior, 4);

    float falda[][2] = {
        {-0.0791019006464 + centroTorso[0], -0.0504133370051 + centroTorso[1]},
        {0.0808656432054 + centroTorso[0], -0.0498050953935 + centroTorso[1]},
        {0.2472774325569 + centroTorso[0], -0.4431154961989 + centroTorso[1]},
        {-0.2458433796982 + centroTorso[0], -0.4450801209489 + centroTorso[1]}
    };
    dibujarPoligono(falda, 4);

    glColor3f(p->colorVestidoOscuro[0], p->colorVestidoOscuro[1], p->colorVestidoOscuro[2]);
    glLineWidth(2.0);
    dibujarLinea(-0.06 + centroTorso[0], 0.15 + centroTorso[1], -0.12 + centroTorso[0], -0.35 + centroTorso[1]);
    dibujarLinea(0.0 + centroTorso[0], 0.15 + centroTorso[1], 0.0 + centroTorso[0], -0.40 + centroTorso[1]);
    dibujarLinea(0.06 + centroTorso[0], 0.15 + centroTorso[1], 0.12 + centroTorso[0], -0.35 + centroTorso[1]);

    glColor3f(p->colorCuello[0], p->colorCuello[1], p->colorCuello[2]);
    float cuello[][2] = {
        {-0.02 + centroTorso[0], 0.22 + centroTorso[1]},
        {0.02 + centroTorso[0], 0.22 + centroTorso[1]},
        {0.02 + centroTorso[0], 0.2 + centroTorso[1]},
        {-0.02 + centroTorso[0], 0.2 + centroTorso[1]}
    };
    dibujarPoligono(cuello, 4);
}
```

Se dibuja la cabeza con cabelo, ojos, cara y boca.

```
void dibujarCabezaPersonaje(Personaje* p) {
    // Guardar la matriz actual
    glPushMatrix();

    // Mover al centro de la cabeza (posición más baja)
    glTranslatef(centroTorso[0], centroTorso[1] - 0.05, 0);

    // Dibujar cabello (parte de atrás)
    glColor3f(0.15, 0.10, 0.08); // Color café oscuro para el cabello
    dibujarCirculo(0, 0.36, 0.095, 30);

    // Dibujar cara
    glColor3f(0.95, 0.87, 0.78); // Color piel
    dibujarCirculo(0, 0.36, 0.08, 30);

    // Dibujar ojos ovalados
    glColor3f(1, 1, 1); // Blanco de los ojos
    // Ojo izquierdo (óvalo horizontal)
    glPushMatrix();
    glTranslatef(-0.03, 0.37, 0);
    glScalef(1.5, 1.0, 1.0); // Achatamos el círculo horizontalmente
    dibujarCirculo(0, 0, 0.02, 16);
    glPopMatrix();

    // Ojo derecho (óvalo horizontal)
    glPushMatrix();
    glTranslatef(0.03, 0.37, 0);
    glScalef(1.5, 1.0, 1.0); // Achatamos el círculo horizontalmente
    dibujarCirculo(0, 0, 0.02, 16);
    glPopMatrix();

    // Pupilas (también ovaladas)
    glColor3f(0, 0, 0); // Negro
    // Pupila izquierda
    glPushMatrix();
    glTranslatef(-0.03, 0.37, 0);
    glScalef(1.5, 1.0, 1.0);
    dibujarCirculo(0, 0, 0.01, 12);
    glPopMatrix();

    // Pupila derecha
    glPushMatrix();
    glTranslatef(0.03, 0.37, 0);
    glScalef(1.5, 1.0, 1.0);
    dibujarCirculo(0, 0, 0.01, 12);
    glPopMatrix();
```

```
// Dibuja un nodo del árbol jerárquico y sus hijos
void dibujarNodoPersonaje(Nodo* nodo) {
    if (nodo == NULL) return;

    glPushMatrix();  // Guardar matriz de transformación actual (PILA)
        // Aplicar transformaciones locales del nodo
        glTranslatef(nodo->tx, nodo->ty, 0);
        glRotatef(nodo->angulo * 180.0 / PI, 0, 0, 1);

        // Dibujar según el tipo de nodo
        if (strcmp(nodo->nombre, "Brazo Derecho") == 0 || strcmp(nodo->nombre, "Brazo Izquierdo") == 0) {
            glColor3f(0.95, 0.87, 0.78);
            glLineWidth(12.0);
            dibujarLinea(0, 0, 0, -nodo->longitud);
        }
        else if (strcmp(nodo->nombre, "Antebrazo Derecho") == 0 || strcmp(nodo->nombre, "Antebrazo Izquierdo") == 0) {
            glColor3f(0.95, 0.87, 0.78);
            glLineWidth(10.0);
            dibujarLinea(0, 0, 0, -nodo->longitud);
        }
        else if (strcmp(nodo->nombre, "Mano Derecha") == 0 || strcmp(nodo->nombre, "Mano Izquierda") == 0) {
            glColor3f(0.95, 0.87, 0.78);
            dibujarCirculo(0, 0, 0.025, 20);
        }
        else if (strcmp(nodo->nombre, "Pierna Derecha") == 0 || strcmp(nodo->nombre, "Pierna Izquierda") == 0) {
            glColor3f(0.95, 0.87, 0.78);
            glLineWidth(15.0);
            dibujarLinea(0, 0, 0, -nodo->longitud);
        }
        else if (strcmp(nodo->nombre, "Pantorrilla Derecha") == 0 || strcmp(nodo->nombre, "Pantorrilla Izquierda") == 0) {
            glColor3f(0.95, 0.87, 0.78);
            glLineWidth(12.0);
            dibujarLinea(0, 0, 0, -nodo->longitud);
        }
        else if (strcmp(nodo->nombre, "Pie Derecho") == 0 || strcmp(nodo->nombre, "Pie Izquierdo") == 0) {
            glColor3f(0.95, 0.87, 0.78);
            dibujarCirculo(0, 0, 0.030, 20);
        }

        // Dibujar hijos (recursión en profundidad)
        dibujarNodoPersonaje(nodo->primerHijo);
    glPopMatrix();  // Restaurar matriz anterior (PILA)

    // Dibujar hermanos (recursión en anchura)
    dibujarNodoPersonaje(nodo->siguienteHermano);
}
```

```
// Dibuja el personaje completo con todas sus transformaciones
void dibujarPersonaje(Personaje* p) {
    glPushMatrix();  // Guardar estado (PILA)
        glTranslatef(p->posX, p->posY, 0);  // Mover a posición en el mundo
        glScalef(p->escala, p->escala, 1.0); // Aplicar escala

        dibujarVestidoPersonaje(p);
        dibujarNodoPersonaje(p->torso);  // Dibujar árbol jerárquico
        dibujarCabezaPersonaje(p);
    glPopMatrix();  // Restaurar estado (PILA)
}
```

```
// Crea un nuevo personaje con sus características
Personaje* crearPersonaje(char* nombre, double x, double y, double escala, float r, float g, float b, int tipoAnimacion) {
    Personaje* p = (Personaje*)malloc(sizeof(Personaje));
    strcpy(p->nombre, nombre);
    p->posX = x;
    p->posY = y;
    p->escala = escala;

    // Configurar colores
    p->colorVestido[0] = r;
    p->colorVestido[1] = g;
    p->colorVestido[2] = b;
    p->colorVestidoOscuro[0] = r * 0.7;
    p->colorVestidoOscuro[1] = g * 0.7;
    p->colorVestidoOscuro[2] = b * 0.7;
    p->colorCuello[0] = r * 1.2;
    p->colorCuello[1] = g * 1.2;
    p->colorCuello[2] = b * 1.2;

    // Ángulos iniciales
    p->anguloBrazoDer = BRAZO_MIN * DEG_TO_RAD;
    p->anguloAntebrazoDer = ANTEBRAZO_MIN * DEG_TO_RAD;
    p->anguloBrazoIzq = BRAZO_MIN * DEG_TO_RAD;
    p->anguloAntebrazoIzq = ANTEBRAZO_MIN * DEG_TO_RAD;
    p->anguloMusloDer = MUSLO_MIN * DEG_TO_RAD;
    p->anguloPantorrillaDer = RODILLA_MAX * DEG_TO_RAD;
    p->anguloMusloIzq = MUSLO_MIN * DEG_TO_RAD;
    p->anguloPantorrillaIzq = RODILLA_MAX * DEG_TO_RAD;

    // Configurar animación
    p->tiempoAnimacion = 0.0;
    p->tipoAnimacion = tipoAnimacion;
    p->animando = (tipoAnimacion != 0);

    // Construir árbol jerárquico
    construirArbolPersonaje(p);

    return p;
}
```

```
void eliminarPersonaje(Personaje* p) {
    if (p == NULL) return;
    liberarArbol(p->torso);
    free(p);
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    // Renderizar la escena actual de la cola
    renderizarEscenaActual();

    // Indicador de información
    glColor3f(0.5, 0.5, 0.5);
    glRasterPos2f(-0.98, 0.95);
    char info[100];
    sprintf(info, "Escena %d | Tiempo: %.1fs", escena_actual, colaEscenas.tiempoTotal);
    for(int i = 0; info[i] != '\0'; i++) {
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, info[i]);
    }

    // Indicador de pausa
    if(animacion_pausada) {
        glColor3f(1.0, 1.0, 0.0);
        glRasterPos2f(-0.15, 0.95);
        char pausa[] = "|| PAUSADO ||";
        for(int i = 0; pausa[i] != '\0'; i++) {
            glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, pausa[i]);
        }
    }

    glutSwapBuffers();
}
```

```
void dibujarTexto(char *texto, float x, float y) {
    glRasterPos2f(x, y);
    for(int i = 0; texto[i] != '\0'; i++) {
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, texto[i]);
    }
}
```

Se dibuja por partes el fantasma para dar la perspectiva de la forma, con cuadros y círculos.

```
void dibujarFantasma(float x, float y, float escala) {
    glColor3f(0.95, 0.95, 1.0);
    dibujarCirculo(x, y + 0.05 * escala, 0.08 * escala, 30);

    glBegin(GL_QUADS);
        glVertex2f(x - 0.08 * escala, y + 0.05 * escala);
        glVertex2f(x + 0.08 * escala, y + 0.05 * escala);
        glVertex2f(x + 0.08 * escala, y - 0.08 * escala);
        glVertex2f(x - 0.08 * escala, y - 0.08 * escala);
    glEnd();

    float base_y = y - 0.08 * escala;
    for(int i = 0; i < 3; i++) {
        float pos_x = x - 0.05 * escala + (i * 0.05 * escala);
        glBegin(GL_POLYGON);
        for(int j = 0; j <= 10; j++) {
            float angulo = PI + (PI * j / 10.0);
            float px = pos_x + 0.025 * escala * cos(angulo);
            float py = base_y + 0.025 * escala * sin(angulo);
            glVertex2f(px, py);
        }
        glEnd();
    }

    glColor3f(0.1, 0.1, 0.1);
    dibujarCirculo(x - 0.03 * escala, y + 0.06 * escala, 0.015 * escala, 20);
    dibujarCirculo(x + 0.03 * escala, y + 0.06 * escala, 0.015 * escala, 20);

    glLineWidth(2.0);
    glBegin(GL_LINE_STRIP);
    for(int i = 0; i <= 10; i++) {
        float t = i / 10.0;
        float boca_x = x + (t - 0.5) * 0.04 * escala;
        float boca_y = y + 0.02 * escala - (t * (1 - t)) * 0.02 * escala;
        glVertex2f(boca_x, boca_y);
    }
    glEnd();
}
```

Mera decoración para una escena navideña del fantasma.

```
void dibujarGorroNavidad(float x, float y, float escala) {
    // Parte principal roja del gorro
    glColor3f(0.9, 0.15, 0.15);
    glBegin(GL_TRIANGLES);
        glVertex2f(x, y + 0.15 * escala);
        glVertex2f(x - 0.08 * escala, y);
        glVertex2f(x + 0.08 * escala, y);
    glEnd();

    // Borde blanco inferior
    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_QUADS);
        glVertex2f(x - 0.08 * escala, y);
        glVertex2f(x + 0.08 * escala, y);
        glVertex2f(x + 0.08 * escala, y + 0.02 * escala);
        glVertex2f(x - 0.08 * escala, y + 0.02 * escala);
    glEnd();

    // Pompón blanco en la punta
    dibujarCirculo(x, y + 0.15 * escala, 0.025 * escala, 15);
}
```

```
void redibujo() {
    SLEEP(30);

    if(!animaciones_inicializadas) {
        inicializarAnimaciones();
        inicializarEscenas();
    }

    if(!animacion_pausada) {
        colaEscenas.tiempoTotal += 0.03;
        tiempo_total += 0.03;

        for(int i = 0; i < numPersonajes; i++) {
            if(animaciones[i].numKeyframes > 0) {
                reproducirAnimacion(i);
            }
        }

        fantasma_oscilacion += 0.1;

        if(escena_actual == 1) {
        if(fantasma_escala_inicio < 2.5) {
            fantasma_escala_inicio += 0.015;
        }

        if(fantasma_escala_inicio > 0.8) {
            alpha_texto = (fantasma_escala_inicio - 0.8) / 1.7;
            if(alpha_texto > 1.0) alpha_texto = 1.0;
        } else {
            alpha_texto = 0.0;
        }
        } else {
            alpha_texto = 1.0;
        }
        if(escena_actual == 5 && fantasma_acercamiento < 0.0) {
            fantasma_acercamiento += 0.01;
        }
        else if(escena_actual == 16) {
            if(fantasma_alejamiento > 0.3) {  // Se va encogiendo
                fantasma_alejamiento -= 0.012;  // Velocidad de alejamiento
            }

            if(fantasma_alejamiento < 1.0) {
                alpha_texto = fantasma_alejamiento / 1.0;
            } else {
                alpha_texto = 1.0;
            }
        }
        else {
            alpha_texto = 1.0;
        }
    }
    glutPostRedisplay();
}
```

Es la configuración de el teclado para la ventana principal .

```
// Manejo de teclado en ventana principal
void teclado(unsigned char key, int x, int y) {
    if(key == 27) {
        for(int i = 0; i < numPersonajes; i++) {
            eliminarPersonaje(personajes[i]);
        }
        exit(0);
    }
    else if(key == 32) {
        animacion_pausada = !animacion_pausada;
        printf("Animacion %s\n", animacion_pausada ? "PAUSADA" : "REPRODUCIENDO");
        glutSetWindow(ventana_menu);
        glutPostRedisplay();
        glutSetWindow(ventana_principal);
    }
    else if(key == 'r' || key == 'R') {
    // Reiniciar animaciones
    for(int i = 0; i < numPersonajes; i++) {
        animaciones[i].tiempoAnimacion = 0.0;
    }

        // Reiniciar escenas y variables
        colaEscenas.tiempoTotal = 0.0;
        tiempo_total = 0.0;
        escena_actual = 0;  // CAMBIAR de 1 a 0
        alpha_texto = 0.0;
        fantasma_x = -0.8;
        fantasma_oscilacion = 0.0;
        fantasma_escala_inicio = 0.3;
        fantasma_acercamiento = -1.2;
        fantasma_alejamiento = 2.5;

        animacion_pausada = 1;
        printf("Animacion REINICIADA\n");

        glutSetWindow(ventana_menu);
        glutPostRedisplay();
        glutSetWindow(ventana_principal);
    }
    else if(key == 'm' || key == 'M') {
        menu_activo = !menu_activo;
        if(menu_activo) {
            glutSetWindow(ventana_menu);
            glutShowWindow();
        } else {
            glutSetWindow(ventana_menu);
            glutHideWindow();
        }
    }

    glutPostRedisplay();
}
```

Se configuran los botones de la ventana del menú.

```
// Dibuja un botón con texto
void dibujarBotonMenu(float x, float y, float ancho, float alto, char *texto, int seleccionado) {
    // Fondo del botón
    if(seleccionado) {
        glColor3f(0.3, 0.5, 0.8);
    } else {
        glColor3f(0.2, 0.2, 0.25);
    }

    glBegin(GL_QUADS);
        glVertex2f(x, y);
        glVertex2f(x + ancho, y);
        glVertex2f(x + ancho, y + alto);
        glVertex2f(x, y + alto);
    glEnd();

    // Borde
    glColor3f(0.5, 0.5, 0.5);
    glLineWidth(2.0);
    glBegin(GL_LINE_LOOP);
        glVertex2f(x, y);
        glVertex2f(x + ancho, y);
        glVertex2f(x + ancho, y + alto);
        glVertex2f(x, y + alto);
    glEnd();
    if(seleccionado) {
        glColor3f(1.0, 1.0, 1.0);
    } else {
        glColor3f(0.8, 0.8, 0.8);
    }

    glRasterPos2f(x + 0.05, y + alto * 0.6);
    for(int i = 0; texto[i] != '\0'; i++) {
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, texto[i]);
    }
}
```

Se hace toda la vista de la segunda ventana la cual es el menú.

```
void displayMenu(void) {
    glClear(GL_COLOR_BUFFER_BIT);

    // Título
    glColor3f(1.0, 0.6, 0.0);
    glRasterPos2f(-0.75, 0.85);
    char titulo[] = "CONTROL DE ANIMACION";
    for(int i = 0; titulo[i] != '\0'; i++) {
        glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, titulo[i]);
    }

    // Botones
    float y_base = 0.50;
    float espaciado = 0.22;

    dibujarBotonMenu(-0.85, y_base, 1.70, 0.18, "REPRODUCIR", opcion_seleccionada == 0);
    dibujarBotonMenu(-0.85, y_base - espaciado, 1.70, 0.18, "PAUSAR", opcion_seleccionada == 1);
    dibujarBotonMenu(-0.85, y_base - espaciado * 2, 1.70, 0.18, "REINICIAR", opcion_seleccionada == 2);
    dibujarBotonMenu(-0.85, y_base - espaciado * 3, 1.70, 0.18, "SALIR", opcion_seleccionada == 3);

    // Instrucciones
    glColor3f(0.6, 0.6, 0.6);
    glRasterPos2f(-0.80, -0.70);
    char inst1[] = "Flechas: Navegar";
    for(int i = 0; inst1[i] != '\0'; i++) {
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, inst1[i]);
    }

    glRasterPos2f(-0.80, -0.77);
    char inst2[] = "ENTER: Seleccionar";
    for(int i = 0; inst2[i] != '\0'; i++) {
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, inst2[i]);
    }

    glRasterPos2f(-0.80, -0.84);
    char inst3[] = "Click: Seleccion directa";
    for(int i = 0; inst3[i] != '\0'; i++) {
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_12, inst3[i]);
    }

    // Estado actual
    glColor3f(1.0, 1.0, 0.0);
    glRasterPos2f(-0.70, -0.55);
    char estado[50];
    if(animacion_pausada) {
        sprintf(estado, "Estado: PAUSADO");
    } else {
        sprintf(estado, "Estado: REPRODUCIENDO");
    }
    for(int i = 0; estado[i] != '\0'; i++) {
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18, estado[i]);
    }
```

Se configra lo necesario para poder pausar, reanudar y ocultar el menu.

```
void tecladoMenu(unsigned char key, int x, int y) {
    if(key == 13) {
        switch(opcion_seleccionada) {
            case 0:
                animacion_pausada = 0;
                printf("Animacion REPRODUCIENDO\n");
                break;
            case 1:
                animacion_pausada = 1;
                printf("Animacion PAUSADA\n");
                break;
            case 2:  // REINICIAR
                for(int i = 0; i < numPersonajes; i++) {
                    personajes[i]->tiempoAnimacion = 0.0;
                    animaciones[i].tiempoAnimacion = 0.0;
                }
                colaEscenas.tiempoTotal = 0.0;
                tiempo_total = 0.0;
                escena_actual = 0;  // CAMBIAR de 1 a 0
                alpha_texto = 0.0;
                fantasma_x = -0.8;  // CAMBIAR de -1.2 a -0.8
                fantasma_oscilacion = 0.0;
                fantasma_escala_inicio = 0.3;
                fantasma_acercamiento = -1.2;
                fantasma_alejamiento = 2.5;  // AGREGAR esta línea
                animacion_pausada = 1;
                printf("Animacion REINICIADA\n");

                break;
            case 3:  // SALIR
                for(int i = 0; i < numPersonajes; i++) {
                    eliminarPersonaje(personajes[i]);
                }
                exit(0);
                break;
        }
        glutSetWindow(ventana_menu);
        glutPostRedisplay();
        glutSetWindow(ventana_principal);
        glutPostRedisplay();
    }
    else if(key == 'm' || key == 'M') {  //ocultar menú
        menu_activo = !menu_activo;
        if(menu_activo) {
            glutSetWindow(ventana_menu);
            glutShowWindow();
        } else {
            glutSetWindow(ventana_menu);
            glutHideWindow();
        }
    }
}
```

se crean los frames para als escenas.

```
// Crear un keyframe copiando el estado actual del personaje
Keyframe* crearKeyframe(float tiempo, Personaje* p) {
    Keyframe* nuevo = (Keyframe*)malloc(sizeof(Keyframe));
    if(nuevo == NULL) {
        printf("Error: No se pudo crear keyframe\n");
        return NULL;
    }

    nuevo->tiempo = tiempo;

    // Copiar todos los ángulos
    nuevo->anguloBrazoDer = p->anguloBrazoDer;
    nuevo->anguloAntebrazoDer = p->anguloAntebrazoDer;
    nuevo->anguloBrazoIzq = p->anguloBrazoIzq;
    nuevo->anguloAntebrazoIzq = p->anguloAntebrazoIzq;
    nuevo->anguloMusloDer = p->anguloMusloDer;
    nuevo->anguloPantorrillaDer = p->anguloPantorrillaDer;
    nuevo->anguloMusloIzq = p->anguloMusloIzq;
    nuevo->anguloPantorrillaIzq = p->anguloPantorrillaIzq;

    // Copiar posición
    nuevo->posX = p->posX;
    nuevo->posY = p->posY;

    nuevo->siguiente = NULL;

    return nuevo;
}
```

Se configuran la flechas para navegar por el menu.

```
void tecladoEspecialMenu(int key, int x, int y) {
    if(key == GLUT_KEY_UP) {  // Flecha arriba
        opcion_seleccionada--;
        if(opcion_seleccionada < 0) opcion_seleccionada = 3;
    }
    else if(key == GLUT_KEY_DOWN) {  // Flecha abajo
        opcion_seleccionada++;
        if(opcion_seleccionada > 3) opcion_seleccionada = 0;
    }
    glutSetWindow(ventana_menu);
    glutPostRedisplay();
    glutSetWindow(ventana_principal);
    glutPostRedisplay();
}
```

se van uniendo los frames para las escenas.

```
void agregarKeyframe(AnimacionLista* anim, Keyframe* kf) {
    if(kf == NULL) return;

    if(anim->primerKeyframe == NULL) {
        // Lista vacía
        anim->primerKeyframe = kf;
        anim->keyframeActual = kf;
    } else {
        // Buscar el último
        Keyframe* actual = anim->primerKeyframe;
        while(actual->siguiente != NULL) {
            actual = actual->siguiente;
        }
        actual->siguiente = kf;
    }

    anim->numKeyframes++;
}
```

Se va siguiendo el orden de en que frame esta la escena.

```
Keyframe* buscarKeyframeActual(AnimacionLista* anim) {
    Keyframe* actual = anim->primerKeyframe;
    Keyframe* anterior = NULL;

    while(actual != NULL) {
        if(actual->tiempo > anim->tiempoAnimacion) {
            return anterior; // Retornar el keyframe anterior
        }
        anterior = actual;
        actual = actual->siguiente;
    }

    return anterior; // Último keyframe
}
```

Se crean las aniamciones con como se van a mover cada parte.

```
void inicializarAnimaciones() {
    if(animaciones_inicializadas){
        return;
    }

    for(int i = 0; i < 10; i++) {
        animaciones[i].primerKeyframe = NULL;
        animaciones[i].keyframeActual = NULL;
        animaciones[i].numKeyframes = 0;
        animaciones[i].tiempoAnimacion = 0.0;
    }

    // ANIMACIÓN 0: Rubi levanta brazos
    //se crean personajes temporales para diseñar, no son lo que s emuestran
    Personaje* temp0 = crearPersonaje("Temp", 0.0, 0.0, 1.0, 0.55, 0.25, 0.85, 0);
    temp0->anguloBrazoDer = BRAZO_MIN * DEG_TO_RAD;
    temp0->anguloBrazoIzq = BRAZO_MIN * DEG_TO_RAD;
    Keyframe* kf0_1 = crearKeyframe(0.0, temp0);
    agregarKeyframe(&animaciones[0], kf0_1);

    temp0->anguloBrazoDer = BRAZO_MAX * DEG_TO_RAD * 0.6;
    temp0->anguloAntebrazoDer = ANTEBRAZO_MAX * DEG_TO_RAD * 0.4;
    Keyframe* kf0_2 = crearKeyframe(2.0, temp0);
    agregarKeyframe(&animaciones[0], kf0_2);

    temp0->anguloBrazoIzq = BRAZO_MAX * DEG_TO_RAD * 0.6;
    temp0->anguloAntebrazoIzq = ANTEBRAZO_MAX * DEG_TO_RAD * 0.4;
    Keyframe* kf0_3 = crearKeyframe(4.0, temp0);
    agregarKeyframe(&animaciones[0], kf0_3);

    temp0->anguloBrazoDer = BRAZO_MIN * DEG_TO_RAD;
    temp0->anguloAntebrazoDer = ANTEBRAZO_MIN * DEG_TO_RAD;
    temp0->anguloBrazoIzq = BRAZO_MIN * DEG_TO_RAD;
    temp0->anguloAntebrazoIzq = ANTEBRAZO_MIN * DEG_TO_RAD;
    Keyframe* kf0_4 = crearKeyframe(6.0, temp0);
    agregarKeyframe(&animaciones[0], kf0_4);

    eliminarPersonaje(temp0);
```

Esto es para gestionar los caso de las animaciones dependiendo del personaje, por ejemplo personaje 1 con animación 2 , renata camina.

```
//Decide que animacion va a usar
void reproducirAnimacion(int indicePersonaje) {
    if(indicePersonaje < 0 || indicePersonaje >= numPersonajes){
        return;
    }

    int indiceAnimacion = indicePersonaje; // Por defecto usa su propia animación
    int debe_animar = 0; // Flag para saber si debe animar en esta escena

    // Decidir qué animación usar según la escena actual
    if(escena_actual == 2) {
        // ESCENA 2: Altar de muertos
        debe_animar = 1;
        if(indicePersonaje == 0) indiceAnimacion = 0;
        else if(indicePersonaje == 1) indiceAnimacion = 1;
        else if(indicePersonaje == 2) indiceAnimacion = 2;
        else if(indicePersonaje == 3) indiceAnimacion = 0;
    }
    else if(escena_actual == 4) {
        debe_animar = 1;
        if(indicePersonaje == 0) indiceAnimacion = 0;
        else if(indicePersonaje == 1) indiceAnimacion = 0;
        else if(indicePersonaje == 2) indiceAnimacion = 2;
        else if(indicePersonaje == 3) indiceAnimacion = 0;
    }
    else if(escena_actual == 7) {
        if(indicePersonaje == 3) {
            debe_animar = 1;
            indiceAnimacion = 3;
        }
    }
    else if(escena_actual == 8) {
        if(indicePersonaje == 0) {
            debe_animar = 1;
            indiceAnimacion = 2;
        }
    }
    else if(escena_actual == 9) {
        if(indicePersonaje == 2) {
            debe_animar = 1;
            indiceAnimacion = 1;
        }
    }
    else if(escena_actual == 10) {
        if(indicePersonaje == 0) {
            debe_animar = 1;
            indiceAnimacion = 4;
        }
    }
```

Esto se usa para la parte de los frames, como es una escena declaramos varios frames, con diferentes posiciones lo que hace es interpolar para que se vea un movimiento mas suave.

```
// Interpolar entre dos keyframes
void interpolarKeyframes(Keyframe* k1, Keyframe* k2, float t, Personaje* p) {
    if(k1 == NULL || k2 == NULL) return;

    if(t < 0.0) t = 0.0;
    if(t > 1.0) t = 1.0;

    // Interpolar BRAZOS
    p->anguloBrazoDer = k1->anguloBrazoDer + t * (k2->anguloBrazoDer - k1->anguloBrazoDer);
    p->anguloAntebrazoDer = k1->anguloAntebrazoDer + t * (k2->anguloAntebrazoDer - k1->anguloAntebrazoDer);
    p->anguloBrazoIzq = k1->anguloBrazoIzq + t * (k2->anguloBrazoIzq - k1->anguloBrazoIzq);
    p->anguloAntebrazoIzq = k1->anguloAntebrazoIzq + t * (k2->anguloAntebrazoIzq - k1->anguloAntebrazoIzq);

    // Interpolar PIERNAS
    p->anguloMusloDer = k1->anguloMusloDer + t * (k2->anguloMusloDer - k1->anguloMusloDer);
    p->anguloPantorrillaDer = k1->anguloPantorrillaDer + t * (k2->anguloPantorrillaDer - k1->anguloPantorrillaDer);
    p->anguloMusloIzq = k1->anguloMusloIzq + t * (k2->anguloMusloIzq - k1->anguloMusloIzq);
    p->anguloPantorrillaIzq = k1->anguloPantorrillaIzq + t * (k2->anguloPantorrillaIzq - k1->anguloPantorrillaIzq);

    // Interpolar POSICIÓN
    p->posX = k1->posX + t * (k2->posX - k1->posX);
    p->posY = k1->posY + t * (k2->posY - k1->posY);
}
```

Las escenas que se crean se van uniendo en la cola para poder llevar un orden.

```
void encolarEscena(int num, float inicio, float fin, char* nombre, void (*func)(void)) {
    Escena* nueva = (Escena*)malloc(sizeof(Escena));
    if(nueva == NULL) {
        return;
    }

    nueva->numero = num;
    nueva->duracionInicio = inicio;
    nueva->duracionFin = fin;
    strcpy(nueva->nombre, nombre);
    nueva->renderizar = func;
    nueva->siguiente = NULL;

    // Si la cola está vacía
    if(colaEscenas.frente == NULL) {
        colaEscenas.frente = nueva;
        colaEscenas.final = nueva;
    }
    // Si ya hay escenas
    else {
        colaEscenas.final->siguiente = nueva;
        colaEscenas.final = nueva;
    }

    colaEscenas.totalEscenas++;
}
```

Así como se declara esta, se declaran las 15 escenas mas , se dibuja el fondo, se dibuja la animación del fantasma, los personajes y los mensajes que desees agregar.

```
void escena1_presentacion() {
    if(texturas_cargadas > 0 && texInicioMictlan.data != NULL) {
        dibujarRectanguloTexturizado(-1.0, -1.0, 2.0, 2.0, texturaInicioMictlan);
    }

    float offset_y = sin(fantasma_oscilacion) * 0.06;
    dibujarFantasma(fantasma_x, fantasma_y + offset_y, fantasma_escala_inicio);


    if(alpha_texto > 0.3) {
        glColor3f(1.0, 1.0, 1.0);
        glRasterPos2f(-0.90, -0.80);//(X,Y)
        char texto1[] = "Juntos de nuevo,";
        for(int i = 0; texto1[i] != '\0'; i++) {
            glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, texto1[i]);
        }

        glRasterPos2f(-0.90, -0.88);
        char texto2[] = "aunque sea un ratito";
        for(int i = 0; texto2[i] != '\0'; i++) {
            glutBitmapCharacter(GLUT_BITMAP_TIMES_ROMAN_24, texto2[i]);
        }
    }
}
```

Se llaman todas las funciones de las escenas previamente declaradas.

```
void inicializarEscenas() {
    // encolarEscena(número, INICIO, FIN, "nombre", función);
    encolarEscena(1, 0.0, 10.0, "Fantasma llegando", &escena1_presentacion);
    encolarEscena(2, 10.0, 25.0, "Altar de muertos", &escena2_altar);
    encolarEscena(3, 25.0, 35.0, "Fantasma habla", &escena3_dialogo);
    encolarEscena(4, 35.0, 45.0, "Mujeres hablan", &escena4_despedida);
    encolarEscena(5, 45.0, 55.0, "Fantasma decision", &escena5_fantasma_decision);
    encolarEscena(6, 55.0, 60.0, "2 semanas despues...", &escena6_transicion_tiempo);

    encolarEscena(7, 60.0, 70.0, "Mujer verde cuarto", &escena7_mujer_verde_cuarto);
    encolarEscena(8, 70.0, 80.0, "Mujer rosa oficina", &escena8_mujer_rosa_oficina);
    encolarEscena(9, 80.0, 90.0, "Mujer azul graduacion", &escena9_mujer_azul_graduacion);
    encolarEscena(10, 90.0, 100.0, "Mujer morado computadora", &escena10_mujer_morado_computadora);

    encolarEscena(11, 100.0, 105.0, "1 semana despues...", &escena11_transicion_navidad);

    encolarEscena(12, 105.0, 125.0, "Navidad recuerdos", &escena12_navidad_recuerdos);
    encolarEscena(13, 125.0, 145.0, "Cena navidad", &escena13_cena_navidad);

    encolarEscena(14, 145.0, 157.0, "Fantasma despedida", &escena14_fantasma_despedida);
    encolarEscena(15, 157.0, 169.0, "Hijas despedida", &escena15_despedida_hijas);
    encolarEscena(16, 169.0, 180.0, "Camino Mictlan", &escena16_camino_mictlan);

}
```

```
void inicializarTexturas() {
    glEnable(GL_TEXTURE_2D);
    texInicioMictlan.data = cargarBMP("inicio_mictlan.bmp", &texInicioMictlan.width, &texInicioMictlan.height);
    if (texInicioMictlan.data != NULL) {
        glGenTextures(1, &texturaInicioMictlan);
        glBindTexture(GL_TEXTURE_2D, texturaInicioMictlan);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP);
        glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP);
        glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, texInicioMictlan.width, texInicioMictlan.height, 0, GL_BGR_EXT, GL_UNSIGNED_BYTE, texInicioMictlan.data);
        texturas_cargadas++;
    }
```

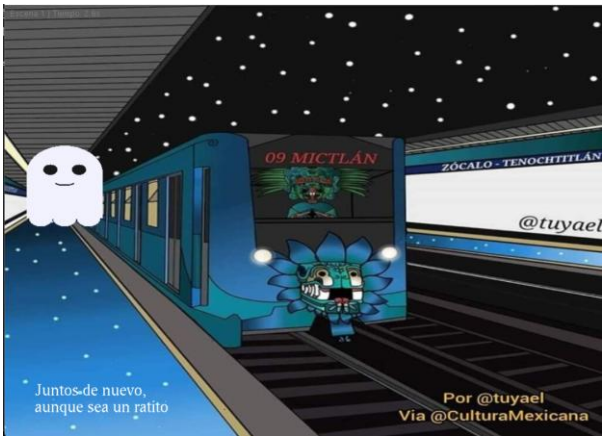
Esto se usa para el fondo con textura que se le agregue.

```
void dibujarRectanguloTexturizado(float x, float y, float ancho, float alto, GLuint textura) {
    glEnable(GL_TEXTURE_2D);
    glBindTexture(GL_TEXTURE_2D, textura);

    glColor3f(1.0, 1.0, 1.0);
    glBegin(GL_QUADS);
        glTexCoord2f(0.0, 0.0); glVertex2f(x, y);
        glTexCoord2f(1.0, 0.0); glVertex2f(x + ancho, y);
        glTexCoord2f(1.0, 1.0); glVertex2f(x + ancho, y + alto);
        glTexCoord2f(0.0, 1.0); glVertex2f(x, y + alto);
    glEnd();

    glDisable(GL_TEXTURE_2D);
}
```

Resultados

Escenas

Escena 1



Escena 2

Escena 3

Escena 4

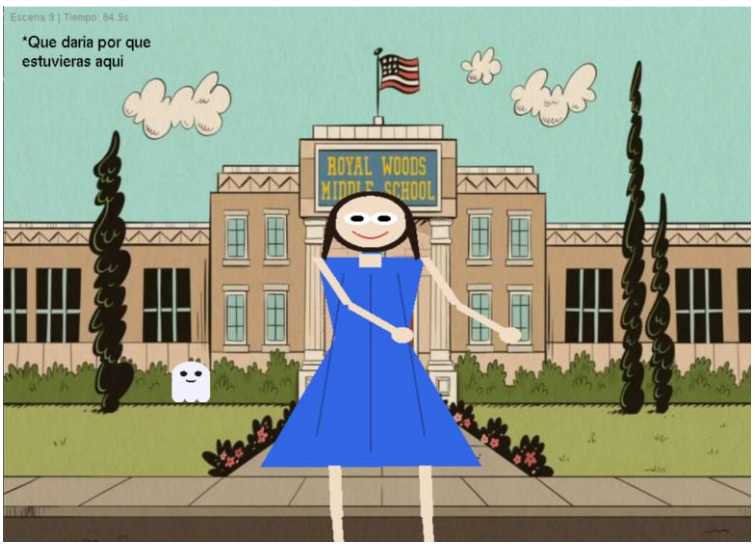
Escena 5

Escena 6

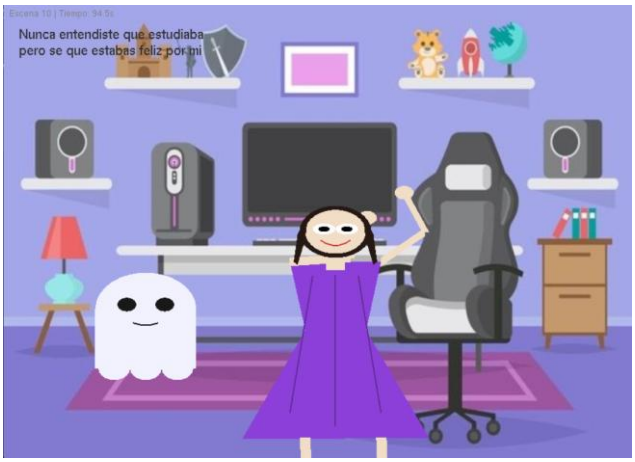
Escena 7

Escena 8

Escena 9



Escena 10



Escena 11

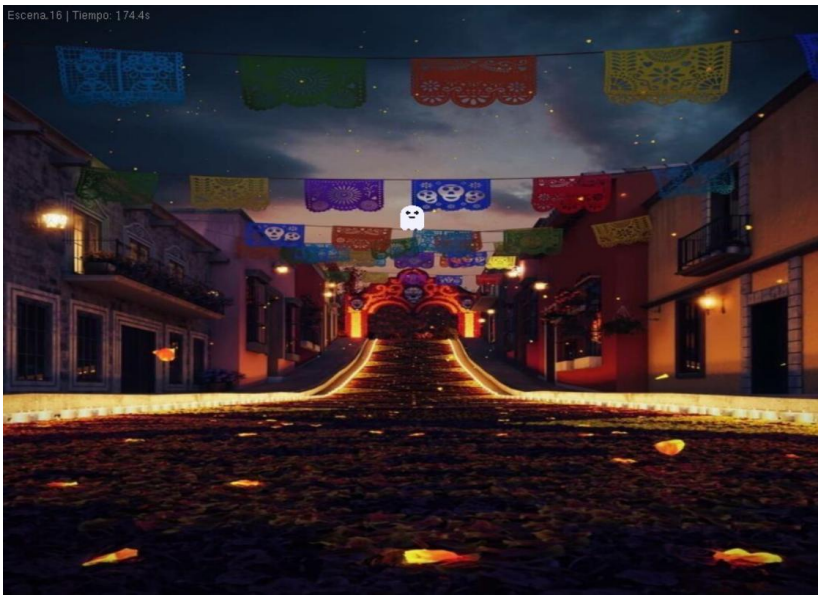
Escena 12

Escena 13

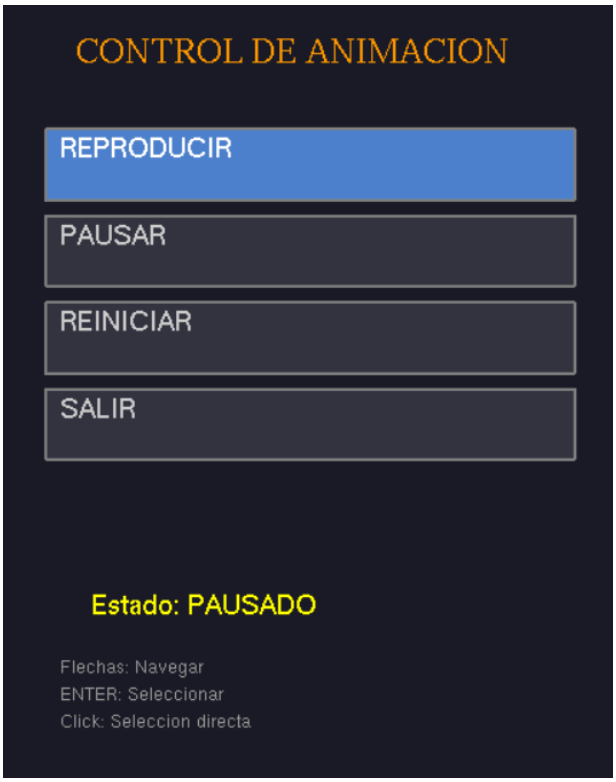
Escena 14

Escena 15

Escena 16



Menú



Manual de usuario.

Primero deberá de iniciar el programa con el nombre del archivo en consola, de esta manera.

```
COMPILACIÓN:
cd "C:\Users\renat\Desktop\AGO25-ENE26\Graficacion"
gcc Proyecto.c -o Proyecto.exe -lfreeglut -lopengl32 -lglu32 -mconsole
Proyecto.exe
```

Seguido de esto se desplegará un menú a la izquierda y una pantalla a la izquierda.

Para usar el menú se puede manejar con teclas como R-para reiniciar, M- para ocultar menú, las flechas para desplazarte por los botones y Enter para seleccionar la opción deseada.

Una vez le des enter en reproducir la animación empezará automáticamente y pasaran todas las escenas con animaciones.

En caso de querer pausar la animación o reiniciarla desplázate al botón deseado y haz Enter, en caso de querer salir , hacer lo mismo pero a la opción de salir o presionar la tecla esc.

Conclusiones

Logré implementar la mayoría de las cosas que se requerían en la rubrica, pero tuve dificultades en la animación, la falta de tiempo me dificultó con la lógica que tenia de las primeras