UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DIVISIÓN DE INGENIERÍA ELÉCTRICA

INGENIERÍA EN COMPUTACIÓN

LABORATORIO DE COMPUTACIÓN GRÁFICA e INTERACCIÓN HUMANO COMPUTADORA

MANUAL TÉCNICO OVERMASK

NOMBRES COMPLETOS:

- Tapia Garcia Andrés

Nº de Cuenta:

- 320252367

GRUPO DE LABORATORIO: 13

GRUPO DE TEORÍA: 06

SEMESTRE 2026-1

FECHA DE ENTREGA LÍMITE: 12/11/2025

CALIFICACIÓN: _____

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DIVISIÓN DE INGENIERÍA ELÉCTRICA

INGENIERÍA EN COMPUTACIÓN

MANUAL TÉCNICO

DE DESARROLLO

Proyecto Final - Overmask

Computación Gráfica e Interacción Humano-Computadora

Nombre:	Tapia Garcia Andrés
No. de Cuenta:	320252367
Grupo de Laboratorio:	13
Grupo de Teoría:	06
Semestre:	2026-1

Noviembre 2025

Índice

1. Modulo Objects

El modulo `Objects` se encarga de definir los objetos Texture y Model, asi como de cargar la dirección de los modelos.

1.1. Archivo Objects.h

Crea los objetos de Textura y Modelos

```
#pragma once
#include <vector>
#include "Texture.h"
#include "Model.h"
class Objects
{
public:
    // ========== TEXTURAS ==========
    Texture brickTexture;
    Texture dirtTexture;
    Texture plainTexture;
    Texture pisoTexture;
    // ========== MODELOS ==========
    // Personaje principal
    Model DekuBase;
    Model DekuBrazo;
    Model DekuPierna;
    Model Nuez;

    // NPCs
    Model Navi;
    Model Salesman;
    Model SkullKid;
    // Elementos del escenario
    Model Luna;
    Model Piramide;
    Model Reloj;
    Model Manecillas;
    Model Ring;
    Model Choza;
    Model Lamp;

    // Limitadores de mapa
    Model Cerca;
    Model Pared;
    Model Puesto;

    // Cucko (NPC animado)
    Model CuckoBase;
    Model CuckoAlaL;
    Model CuckoAlaR;
    Model CuckoPata;

    Model Mesa;

    Objects();
    ~Objects();
    void LoadTextures();
    void LoadModels();
    void LoadAll();  // Carga todo en una sola llamada
};
```

Listing 1: Definición de la clase Objects

1.2. Implementación en Objects.cpp

aqui se define cada Model y se carga la dirección del objeto.

```
void Objects::LoadAll()
{
    LoadTextures();
    LoadModels();
}

void Objects::LoadModels()
{
    // Deku Link (Avatar jerarquico)
    DekuBase = Model();
    DekuBase.LoadModel("Models/DekuLink/DekuBase.obj");

    DekuBrazo = Model();
    DekuBrazo.LoadModel("Models/DekuLink/DekuBrazo.obj");

    DekuPierna = Model();
    DekuPierna.LoadModel("Models/DekuLink/DekuPierna.obj");

    // NPCs
    Navi = Model();
    Navi.LoadModel("Models/Navi.obj");

    Salesman = Model();
    Salesman.LoadModel("Models/Salesman.obj");

    // ... (mas modelos)
}
```

Listing 2: Método LoadAll() - Carga centralizada

2. Módulo Render

El modulo de Render me sirve para definir el renderizado de cada objeto fuera del main, para que no este tan lleno y sea un codigo mas legible

2.1. Archivo Render.h

```
#pragma once
#include <glew.h>
#include <glm.hpp>
#include <gtc/matrix_transform.hpp>
#include <gtc/type_ptr.hpp>
#include <vector>
#include "Objects.h"
#include "Material.h"
#include "Mesh.h"

// ========== PERSONAJE PRINCIPAL ==========
void RenderDekuLink(
    GLuint uniformModel,
    Objects& objects,
    glm::vec3 position = glm::vec3(-5.0f, 0.0f, 0.0f),
    float rotation = 0.0f,
    float armSwing = 0.0f,
    float legSwing = 0.0f
);

// ========== NPCs ==========
void RenderSalesMan(GLuint uniformModel, Objects& objects,
    glm::vec3 position = glm::vec3(8.0f, 3.5f, 110.0f));

void RenderSkullKid(GLuint uniformModel, Objects& objects,
    glm::vec3 position = glm::vec3(-10.0f, 3.5f, 90.0f));

void RenderNavi(GLuint uniformModel, Objects& objects,
    glm::vec3 position, glm::vec3 rotation);

void RenderCucko(GLuint uniformModel, Objects& objects,
    GLfloat time);

// ========== ENTORNO ==========
void RenderFloor(GLuint uniformModel, GLuint uniformColor,
    GLuint uniformSpecularIntensity, GLuint uniformShininess,
    Objects& objects, Material& material,
    std::vector<Mesh*>& meshList);

void RenderLuna(GLuint uniformModel, Objects& objects,
    GLfloat time,
    glm::vec3 position = glm::vec3(-0.0f, 70.0f, 100.0f));

void RenderReloj(GLuint uniformModel, Objects& objects,
    GLfloat time,
    glm::vec3 position = glm::vec3(0.0f, -1.0f, 300.0f));

void RenderRing(GLuint uniformModel, Objects& objects,
    glm::vec3 position = glm::vec3(0.0f, -1.0f, 100.0f));
```

Listing 3: Declaración de funciones de renderizado

2.2. Explicación de Implementaciones importantes en Render.cpp

2.2.1. Avatar Jerárquico - RenderDekuLink

Esta función implementa el **requisito de avatar jerárquico** con animación de caminata:

```
void RenderDekuLink(GLuint uniformModel, Objects& objects,
                    glm::vec3 position, float rotation,
                    float armSwing, float legSwing)
{
    glm::vec3 scale = glm::vec3(0.15f, 0.15f, 0.15f);

    // ===== CUERPO BASE (Raiz de jerarquia) =====
    glm::mat4 model = glm::mat4(1.0);
    model = glm::translate(model, position);
    model = glm::rotate(model, glm::radians(rotation),
                        glm::vec3(0.0f, 1.0f, 0.0f));
    model = glm::scale(model, scale);
    glm::mat4 modelBase = model;  // Guardar matriz base

    glUniformMatrix4fv(uniformModel, 1, GL_FALSE,
                       glm::value_ptr(model));
    objects.DekuBase.RenderModel();

    // ===== BRAZO DERECHO (Hijo del cuerpo) =====
    model = modelBase;
    model = glm::translate(model, glm::vec3(-2.0f, 12.0f, 0.0f));
    model = glm::rotate(model, glm::radians(armSwing),
                        glm::vec3(1.0f, 0.0f, 0.0f));
    glUniformMatrix4fv(uniformModel, 1, GL_FALSE,
                       glm::value_ptr(model));
    objects.DekuBrazo.RenderModel();

    // ===== BRAZO IZQUIERDO =====
    model = modelBase;
    model = glm::translate(model, glm::vec3(2.0f, 12.0f, 0.0f));
    model = glm::rotate(model, glm::radians(180.0f),
                        glm::vec3(0.0f, 1.0f, 0.0f));
    model = glm::rotate(model, glm::radians(armSwing),
                        glm::vec3(1.0f, 0.0f, 0.0f));
    glUniformMatrix4fv(uniformModel, 1, GL_FALSE,
                       glm::value_ptr(model));
    objects.DekuBrazo.RenderModel();

    // ===== PIERNAS (similar) =====
    // ... (c digo de piernas)
}
```

Listing 4: Renderizado de Deku Link con jerarquía

Explicación de la Jerarquía:

1. El **cuerpo base** es el nodo padre de la jerarquía
2. Las **extremidades** heredan las traslaciones y rotaciones del cuerpo
3. Cada extremidad tiene transformaciones propias como traslación para colocarlas donde deben de ir y rotación para la animacion de caminata
4. La variable `armSwing` y `legSwing` controlan la animación

2.2.2. Animación Compleja - RenderLuna

```
void RenderLuna(GLuint uniformModel, Objects& objects,
                GLfloat time, glm::vec3 position)
{
    glm::mat4 model = glm::mat4(1.0);

    // Posicion base
    model = glm::translate(model, position);

    // === MOVIMIENTO ORBITAL ===
    // Rotacion horizontal (circulo completo)
    model = glm::rotate(model, glm::radians(time * 30.0f),
                        glm::vec3(0.0f, 1.0f, 0.0f));

    // Desplazamiento con ondulacion vertical (seno)
    float verticalWave = 30.0f * std::sin(time);
    model = glm::translate(model,
                           glm::vec3(60.0f, verticalWave, 0.0f));

    // Rotacion sobre su propio eje
    model = glm::rotate(model, glm::radians(90.0f),
                        glm::vec3(0.0f, 1.0f, 0.0f));

    // Orientacion inicial del modelo
    model = glm::rotate(model, glm::radians(-90.0f),
                        glm::vec3(0.0f, 1.0f, 0.0f));
    model = glm::rotate(model, glm::radians(135.0f),
                        glm::vec3(0.0f, 0.0f, 1.0f));

    model = glm::scale(model, glm::vec3(0.01f, 0.01f, 0.01f));
    glUniformMatrix4fv(uniformModel, 1, GL_FALSE,
                       glm::value_ptr(model));
    objects.Luna.RenderModel();
}
```

Listing 5: Luna con movimiento orbital sinusoidal

Algoritmo de Animación:

- Usa `sin(time)` para crear oscilación vertical
- rotacion continua para orbita circular

2.2.3. Animación Compleja - RenderCucko

Es mi animación favorita jeje

```
void RenderCucko(GLuint uniformModel, Objects& objects,
                 GLfloat time)
{
    glm::mat4 model = glm::mat4(1.0);

    // === CICLO DE MOVIMIENTO TRASLACIONAL ===
    GLfloat cycle = fmod(time, 8.0f);
    bool forward = cycle < 4.0f;
    GLfloat progress = forward ? cycle / 4.0f
                               : (cycle - 4.0f) / 4.0f;
    GLfloat move = forward ? progress * 15.0f
                           : 15.0f - progress * 15.0f;
    GLfloat rot = forward ? 0.0f : 180.0f;

    model = glm::translate(model,
        glm::vec3(-100.0f, -1.0f, 57.5f - move));
    model = glm::rotate(model, glm::radians(180.0f + rot),
                        glm::vec3(0.0f, 1.0f, 0.0f));
    model = glm::scale(model, glm::vec3(1.0f, 1.0f, 1.0f));
    glm::mat4 modelaux = model;

    // === CUERPO CON OSCILACION ===
    modelaux = glm::translate(modelaux,
        glm::vec3(0.0f, sin(time * 5.0f) * 0.15f, 0.0f));
    modelaux = glm::rotate(modelaux,
        glm::radians(sin(time * 3.0f) * 2.0f),
        glm::vec3(0.0f, 0.0f, 1.0f));
    glUniformMatrix4fv(uniformModel, 1, GL_FALSE,
                       glm::value_ptr(modelaux));
    objects.CuckoBase.RenderModel();

    // === ANIMACION DE ALAS (ALETEO) ===
    GLfloat wing = sin(time * 8.0f) * 45.0f + 15.0f;

    // Ala izquierda
    model = modelaux;
    model = glm::translate(model,
                           glm::vec3(-1.0f, 2.2f, 0.0f));
    model = glm::rotate(model, glm::radians(-wing),
                        glm::vec3(0.0f, 0.0f, 1.0f));
    model = glm::rotate(model,
        glm::radians(sin(time * 8.0f) * 10.0f),
        glm::vec3(1.0f, 0.0f, 0.0f));
    glUniformMatrix4fv(uniformModel, 1, GL_FALSE,
                       glm::value_ptr(model));
    objects.CuckoAlaL.RenderModel();

    // Ala derecha (simetrica)
    // ... (codigo similar)

    // === PATAS CON MOVIMIENTO ALTERNADO ===
    // ... (codigo de patas)
}
```

Elementos de la Animación:

- Traslación con ida y vuelta en ciclo
- Rotación automática al cambiar dirección
- Aleteo de alas con función seno
- Movimiento alternado de patas

3. Sistema de Iluminación Día/Noche

3.1. Implementación del Ciclo

```
// Variables globales
float time = 0.0f;
float dayDuration = 3000.0f;  // Frames para ciclo completo
float dayNightCycle = 0.0f;   // 0.0 = dia, 1.0 = noche

// En el loop principal
while (!mainWindow.getShouldClose()) {
    // Normalizar tiempo al ciclo (0.0 a 1.0)
    dayNightCycle = time / dayDuration;

    // === TRANSICION DE COLORES ===
    glm::vec3 dayLightColor(1.0f, 0.95f, 0.8f);
    glm::vec3 sunsetColor(1.0f, 0.6f, 0.3f);
    glm::vec3 nightLightColor(0.15f, 0.15f, 0.3f);
    glm::vec3 currentLightColor;

    if (dayNightCycle < 0.4f) {
        // DIA COMPLETO
        currentLightColor = dayLightColor;
    }
    else if (dayNightCycle < 0.5f) {
        // ATARDECER
        float t = (dayNightCycle - 0.4f) / 0.1f;
        currentLightColor = glm::mix(dayLightColor,
                                     sunsetColor, t);
    }
    else if (dayNightCycle < 0.6f) {
        // ANOCHECER
        float t = (dayNightCycle - 0.5f) / 0.1f;
        currentLightColor = glm::mix(sunsetColor,
                                     nightLightColor, t);
    }
    else {
        // NOCHE COMPLETA
        currentLightColor = nightLightColor;
    }

    // === INTENSIDAD DE LUZ ===
    float currentAmbient, currentDiffuse;
    if (dayNightCycle < 0.5f) {
        currentAmbient = 0.8f;
        currentDiffuse = 0.6f;
    }
    else {
        float t = (dayNightCycle - 0.5f) / 0.5f;
        currentAmbient = glm::mix(0.8f, 0.15f, t);
        currentDiffuse = glm::mix(0.6f, 0.1f, t);
    }

    // === DIRECCION DEL SOL (ARCO) ===
    float sunAngle = dayNightCycle * 3.14159f;
    float sunDirX = sin(sunAngle) * 0.3f;
    float sunDirY = -cos(sunAngle);
    float sunDirZ = -0.3f;

    // Actualizar luz direccional
    mainLight = DirectionalLight(
        currentLightColor.r, currentLightColor.g,
        currentLightColor.b,
```

```
        currentAmbient, currentDiffuse,
        sunDirX, sunDirY, sunDirZ
    );

    time++;
}
```

Listing 6: Control del ciclo día/noche

3.2. Sistema de Skybox Dual

Se implementaron **dos skyboxes separados** que cambian según la hora:

```
// Cargar ambos skyboxes
std::vector<std::string> skyboxDayFaces;
skyboxDayFaces.push_back("Textures/Skybox/Dia/clouds1_east.png");
// ... (cargar 6 caras)

std::vector<std::string> skyboxNightFaces;
skyboxNightFaces.push_back("Textures/Skybox/Noche/galaxy+X.tga");
// ... (cargar 6 caras)

SkyboxDay = Skybox(skyboxDayFaces);
SkyboxNight = Skybox(skyboxNightFaces);

// En el loop: alternar skybox
if (time < dayDuration/2) {
    SkyboxDay.DrawSkybox(camera.calculateViewMatrix(),
                         projection);
}
else {
    SkyboxNight.DrawSkybox(camera.calculateViewMatrix(),
                           projection);
    if (time >= dayDuration) {
        time = 0.0f;  // Reiniciar ciclo
    }
}
```

Listing 7: Cambio de skybox

3.3. Control de Luces Puntuales (Faroles)

Las luces puntuales se **encienden automáticamente de noche**:

```
// Declaracion de faroles
pointLights[1] = PointLight(1.0f, 1.0f, 1.0f,
    50.0f, 15.0f,           // Intensidad
    80.0f, 35.0f, 70.0f,    // Posicion
    0.3f, 0.2f, 0.1f);      // Atenuacion

pointLights[2] = PointLight(1.0f, 1.0f, 1.0f,
    50.0f, 15.0f,
    -130.0f, 35.0f, 180.0f,
    0.3f, 0.2f, 0.1f);

// En el loop: control automatico
GLuint activeLightCount;
bool isNight = (dayNightCycle >= 0.575f);

if (isNight) {
    activeLightCount = pointLightCount;  // Todas activas
}
else {
    activeLightCount = 1;  // Solo luz de Navi
}

// Enviar solo las luces activas al shader
shaderList[0].SetPointLights(pointLights, activeLightCount);
```

Listing 8: Faroles automáticos

Explicación:

- Durante el día: `activeLightCount = 1` (solo Navi)
- Durante la noche: `activeLightCount = pointLightCount` (todos los faroles)
- El shader recibe solo las luces necesarias, optimizando rendimiento

4. Sistema de Cámara Tercera Persona

4.1. Implementación en Camera.cpp

El proyecto implementa una **cámara en tercera persona con movimiento esférico**:

```
void Camera::followTarget(glm::vec3 targetPosition,
                          GLfloat deltaTime)
{
    if (!thirdPersonMode) return;

    // Convertir angulos a radianes
    float radYaw = glm::radians(yaw);
    float radPitch = glm::radians(pitch);

    // === COORDENADAS ESFERICAS ===
    // X = radio * cos(pitch) * sin(yaw)
    // Y = radio * sin(pitch)
    // Z = radio * cos(pitch) * cos(yaw)

    glm::vec3 offset;
    offset.x = thirdPersonDistance * cos(radPitch)
               * sin(radYaw);
    offset.y = thirdPersonDistance * sin(radPitch);
    offset.z = thirdPersonDistance * cos(radPitch)
               * cos(radYaw);

    // Posicion deseada = target + offset esferico
    glm::vec3 desiredPosition = targetPosition - offset;

    // === SUAVIZADO (LERP) ===
    float lerpFactor = cameraSmoothing * deltaTime;
    if (lerpFactor > 1.0f) lerpFactor = 1.0f;

    position = glm::mix(position, desiredPosition,
                        lerpFactor);

    // Hacer que camara mire al target
    front = glm::normalize(targetPosition - position);

    // Actualizar vectores right y up
    right = glm::normalize(glm::cross(front, worldUp));
    up = glm::normalize(glm::cross(right, front));
}
```

4.2. Zoom con Rueda del Mouse

Se implementó **zoom dinámico** usando la rueda del mouse:

```
// En el loop principal
float scrollOffset = mainWindow.getScrollChange();
if (scrollOffset != 0.0f)
{
    float currentDistance = camera.getThirdPersonDistance();
    currentDistance -= scrollOffset * 2.0f;

    // Limites de zoom
    if (currentDistance < 5.0f) currentDistance = 5.0f;
    if (currentDistance > 300.0f) currentDistance = 300.0f;

    camera.setThirdPersonDistance(currentDistance);
}
```

Listing 9: Control de zoom

5. Sistema de Control del Personaje

El personaje se mueve **hacia donde mira la cámara**:

```
glm::vec3 movement(0.0f);
isWalking = false;

if (mainWindow.isMoving())
{
    isWalking = true;

    // Obtener direccion de camara (sin componente Y)
    glm::vec3 forward = camera.getCameraDirection();
    forward.y = 0.0f;
    forward = glm::normalize(forward);

    // Calcular direccion derecha
    glm::vec3 right = glm::normalize(
        glm::cross(forward, glm::vec3(0.0f, 1.0f, 0.0f))
    );

    // Acumular movimiento segun teclas
    if (mainWindow.isMovingForward())
        movement += forward;
    if (mainWindow.isMovingBackward())
        movement -= forward;
    if (mainWindow.isMovingRight())
        movement += right;
    if (mainWindow.isMovingLeft())
        movement -= right;

    // Normalizar para movimiento diagonal consistente
    if (glm::length(movement) > 0.0f)
    {
        movement = glm::normalize(movement);
        dekuPosition += movement * dekuSpeed * deltaTime;

        // Rotar personaje hacia direccion de movimiento
        dekuRotation = glm::degrees(atan2(movement.x,
                                          movement.z));
    }
}
```

Listing 10: Movimiento WASD relativo a cámara

Se hizo de tal manera que la animación solo se ejecuta **cuando el personaje se mueve**:

```
// Solo animar si esta caminando
if (isWalking)
{
    walkAnimationTime += deltaTime * 0.1f;
}

// Calcular swing de brazos y piernas
float armSwing = isWalking ? sin(walkAnimationTime) * 30.0f
                           : 0.0f;
float legSwing = isWalking ? sin(walkAnimationTime) * 15.0f
                           : 0.0f;

// Renderizar con animacion
RenderDekuLink(uniformModel, objects, dekuPosition,
               dekuRotation, armSwing, legSwing);
```

Listing 11: Animación condicional

6. Animaciones Por teclado

6.1. Proyectiles (Nueces)

Cuando se da click izquierdo el personaje principal lanza un proyectil que es una nuez, en el juego original tambien lo hace jeje

```
struct Nuez {
    glm::vec3 pos;
    glm::vec3 dir;
    bool activa;
};

std::vector<Nuez> nueces;
bool clickAnterior = false;

// En el loop
bool clickActual = mainWindow.getLeftMouseButton();
if (clickActual && !clickAnterior) {
    Nuez n;
    float rad = glm::radians(dekuRotation);
    n.pos = dekuPosition + glm::vec3(0.0f, 2.5f, 0.0f);
    n.dir = glm::vec3(sin(rad), 0.0f, cos(rad));
    n.activa = true;
    nueces.push_back(n);
}
clickAnterior = clickActual;

// Actualizar y renderizar nueces
for (auto& n : nueces) {
    if (n.activa) {
        n.pos += n.dir * 1.5f * deltaTime;
        if (glm::length(n.pos) > 1000.0f) {
            n.activa = false;
            continue;
        }

        glm::mat4 model = glm::mat4(1.0);
        model = glm::translate(model, n.pos);
        model = glm::scale(model, glm::vec3(0.5f));
        glUniformMatrix4fv(uniformModel, 1, GL_FALSE,
                           glm::value_ptr(model));
        objects.Nuez.RenderModel();
    }
}
```

6.1.1. Optimización para que desaparezcan las nueces

Las nueces fuera del rango de 1000.0 se **desactivan**:

```
for (auto& n : nueces) {
    if (n.activa) {
        n.pos += n.dir * 1.5f * deltaTime;
        // Desactivar si esta muy lejos
        if (glm::length(n.pos) > 1000.0f){
            n.activa = false;
            continue;  // No renderizar
        }
        // Solo renderizar si esta activa
        RenderNuez(n.pos);
    }
}
```

Listing 12: Desactivación de proyectiles lejanos

6.2. Animación Orbital de Navi

Navi realiza una **órbita alrededor de Deku** al presionar Q:

```
bool naviAnimating = false;
float naviAnimationAngle = 0.0f;
float naviAnimationSpeed = 5.0f;

// Detectar tecla Q
bool fKeyPressed = mainWindow.getEstadoNaviF();
if (fKeyPressed && !fKeyPressedBefore && !naviAnimating) {
    naviAnimating = true;
    naviAnimationAngle = 0.0f;
}

glm::vec3 naviPos;
if (naviAnimating) {
    // Orbita circular
    naviAnimationAngle += naviAnimationSpeed * deltaTime;
    float radius = 3.0f;
    float rad = glm::radians(naviAnimationAngle);

    naviPos = dekuPosition + glm::vec3(
        cos(rad) * radius,
        sin(glfwGetTime() * 3.0f) * 0.5f + 7.0f,
        sin(rad) * radius
    );

    // Terminar despues de vuelta completa
    if (naviAnimationAngle >= 360.0f) {
        naviAnimating = false;
    }
}
else {
    // Posicion normal (flotando sobre Deku)
    naviPos = glm::vec3(
        dekuPosition.x,
        sin(glfwGetTime() * 3.0f) * 0.5f + 7.0f,
        dekuPosition.z
    );
}
```

Listing 13: Órbita de Navi con tecla Q

7. Conclusiones Individuales

7.1. Conclusión Individual - 320237988

Este proyecto me represento un verdadero reto, dejando de lado que estuve solo, en algunas partes me la pase bien, y en otras estaba super frustrado, diria que esto no es para mi, pero, definitivamente me emocione mucho cuando logre hacer que navi diera una vuelta, la animación de cucko, cuando encontre como crear mis objetos en blender, mi mayor emoción fue cuando logre hacer que link lanzara sus proyectiles jaja, con ese aditamento lo senti como juego real, fue entretenido la parte de código, la logica que se sigue para hacer juegos en 3d es muy interesante y me llama la atención, pero, no lo suficiente, gracias por todas sus enseñanzas profesor.

8. Referencias

1. Khronos Group. (2023). *OpenGL 4.3 Core Profile Specification.* https://www.khronos.org/registry/OpenGL/specs/gl/glspec43.core.pdf
2. GLFW Documentation. (2023). *Window Guide - GLFW.* https://www.glfw.org/docs/latest/window_guide.html
3. Open Asset Import Library. (2023). *ASSIMP Documentation.* http://assimp.sourceforge.net/lib_html/index.html
4. Roque Roman Guadarrama, J. (2025).[Clase, Videos y códigos de curso] . Facultad de Ingeniería, UNAM.
5. Khronos Group. (2025). OpenGL API Documentation. The Khronos Group. https://www.opengl.org/
6. Nintendo. (1986–presente). The Legend of Zelda [franquicia de videojuegos]. Nintendo Co., Ltd. Nintendo. (2000). The Legend of Zelda: Majora's Mask [Videojuego]. Nintendo 64. Nintendo.

8.1. Referencias de Modelos 3D

A continuación se listan las fuentes de los modelos utilizados en el proyecto Overmask:

1. https://www.turbosquid.com/es/3d-models/free-boxing-ring-3d-model/1008778
2. https://cults3d.com/es/modelo-3d/arquitectura/sun-pyramid-teotihuacan-mexico-piramide-
3. https://models.spriters-resource.com/3ds/thelegendofzeldamajorasmask3d/asset/351753/
4. https://models.spriters-resource.com/3ds/thelegendofzeldamajorasmask3d/asset/303292/
5. https://sketchfab.com/3d-models/the-legend-of-zelda-majoras-mask-3d-moon-062f9773cb5b4
6. https://sketchfab.com/3d-models/bonky-the-cucco-49a44042c7af4675bf47eb2e77fd47f0
7. https://sketchfab.com/3d-models/legend-of-zelda-majoras-mask-happy-mask-salesman-c764b
8. https://models.spriters-resource.com/3ds/thelegendofzeldamajorasmask3d/asset/351756/
9. https://sketchfab.com/3d-models/ring-box-fight-pelea-9a3ba7df045d47f9b3d4106522612015#download

10. https://sketchfab.com/3d-models/cabeza-olmeca-3d-scan-108b5e9be0344aca84603d7ba992f12b
11. https://models.spriters-resource.com/nintendo_64/thelegendofzeldamajorasmask/asset/335444/
12. https://opengameart.org/content/clouds-skybox-1
13. https://opengameart.org/content/galaxy-skybox
14. https://www.turbosquid.com/es/3d-models/free-blend-model-rupees-rubees-legend/562363
15. https://www.cgtrader.com/items/2379971/download-page