INSTITUTO POLITÉCNICO NACIONAL

ESCUELA SUPERIOR DE CÓMPUTO

PROPUESTA DE PROYECTO:

Escena Interactiva 3D con Estética de Videojuego

MATERIA:

Computer Graphics

ALUMNO:

Toral Alvarez Yael Adair N.L 34

PROFESOR:

Palma Orozco Rosaura

FECHA DE ENTREGA:

05/01/2026

Índice

Descripción de la problemática a resolver

El desarrollo de cualquier videojuego o aplicación 3D interactiva moderna se fundamenta en la capacidad de generar y manipular gráficos por computadora en tiempo real. Esto implica no solo crear y posicionar objetos en un espacio tridimensional, sino también aplicarles una apariencia visual coherente y animarlos de manera fluida en respuesta a las acciones del usuario.

La problemática central de este proyecto es **integrar los tres pilares fundamentales de la graficación por computadora en una sola aplicación funcional**:

1. **Representación y transformación:** ¿Cómo definimos objetos 3D y los manipulamos en un espacio virtual?
2. **Visualización y realismo:** ¿Cómo hacemos que esos objetos de alambre se vean sólidos, coloridos e iluminados de una manera estilizada?
3. **Dinamismo y animación:** ¿Cómo dotamos de movimiento y vida a los objetos a lo largo del tiempo?

Resolver esta problemática requiere sintetizar los conocimientos teóricos y prácticos de todo el semestre, desde el manejo de vectores y matrices hasta la implementación de modelos de sombreado y secuencias de animación.

Problema: Pregunta

¿Cómo se pueden integrar los fundamentos de modelado geométrico, las técnicas de renderizado e iluminación y los principios de animación por computadora para construir una escena 3D interactiva en tiempo real con una estética *Cel Shading* inspirada en los videojuegos de anime?

Solución al Problema: Respuesta a la pregunta

La solución se construirá de manera incremental, desarrollando un prototipo que abarque las competencias de las tres unidades temáticas del curso. El resultado será una aplicación donde un usuario controla un personaje 3D en un escenario simple y puede ejecutar una animación de "ataque especial".

El desarrollo se desglosa de la siguiente manera:

- **Fase 1: Creación del mundo y controles (Basado en unidad I)**

- Se definirá la geometría del personaje y el escenario usando vértices para construir **mallas poligonales**.
- Se implementarán las **transformaciones geométricas** (traslación, rotación, escalamiento) mediante **representaciones matriciales y coordenadas homogéneas** para posicionar al personaje en el mundo.
- Se creará una cámara virtual utilizando **proyecciones de perspectiva** para renderizar la escena 3D en la pantalla 2D.
- Se capturará la entrada del teclado para modificar la matriz de transformación del personaje, permitiendo su movimiento en tiempo real (adelante, atrás, rotar).

- **Fase 2: Estilo Visual y renderizado (Basado en unidad II)**
  - Para que los objetos se vean sólidos, se implementará un algoritmo de **eliminación de partes ocultas** como el Z-Buffer.
  - Se definirá una fuente de luz y se aplicarán **modelos básicos de iluminación** (ambiental, difusa, especular) para calcular cómo la luz interactúa con las superficies del modelo.
  - Para lograr la estética de videojuego/anime, se implementará un **modelo de sombreado** no fotorrealista conocido como **Cel Shading (o Toon Shading)**. Esto cuantiza la luz en bandas de color sólido, creando el aspecto de "caricatura".
  - Se añadirá un contorno negro a los modelos para acentuar el estilo, una técnica avanzada de renderizado.
- **Fase 3: Animación y efectos (Basado en unidad III)**
  - Se diseñará una animación de "ataque de energía" utilizando los principios de **movimiento y tiempo** y **cinemática.**
  - La trayectoria del proyectil de energía se definirá mediante una **curva paramétrica de Bézier** para lograr un movimiento más dinámico y estilizado que una simple línea recta.
  - Esta animación se activará con una tecla, demostrando la integración de la interacción del usuario con los sistemas de animación.

La aplicación final ejecutará un bucle principal que, en cada fotograma, procesará la entrada, actualizará la lógica del juego (posición del personaje, estado de la animación) y volverá a dibujar toda la escena con las técnicas mencionadas.

Herramientas y técnicas: SW, HW y Tecnología

- **Software (SW):**
    - **Lenguaje de programación: C++**, por su alto rendimiento, que es crucial para gráficos en tiempo real.
    - **API Gráfica: OpenGL** (Open Graphics Library), ya que es un estándar de la industria y se menciona en la bibliografía del curso. Proporciona la interfaz para comunicarse con el hardware gráfico.
    - **Librerías adicionales:**
        - **GLFW:** Para la creación de la ventana, el contexto de OpenGL y la gestión de entradas de teclado y ratón.
        - **GLM (OpenGL Mathematics):** Para las operaciones de álgebra lineal (vectores, matrices) necesarias para las transformaciones.
        - **Assimp (Open Asset Import Library):** Para cargar modelos 3D (formato .obj o .fbx) sin tener que definir su geometría manualmente en el código.
    - **Entorno de Desarrollo (IDE):** Visual Studio Code o Visual Studio 2022.
- **Hardware (HW):**
    - Se cuenta con una computadora de escritorio y una laptop.

      Las especificaciones relevantes de la laptop son:
        - Procesador Intel Core i5 11400H
        - Tarjeta gráfica NVIDIA RTX 3050 de 4 GB
        - 16 GB de RAM
        - Disco duro SSD de 500 GB

      Las especificaciones relevantes de la PC de escritorio:
        - Procesador Intel Core i7 12700K
        - Tarjeta gráfica NVIDIA RTX 3060 12 GB
        - 32GB RAM DDR4
        - Disco duro SSD de 1 TB M.2 NVME

        - Disco Duro HDD de 3 TB

- **Tecnología y Técnicas Clave a Utilizar:**
    - **Pipeline Gráfico en Tiempo Real:** El concepto central que engloba todo el proceso, desde los datos del modelo hasta el píxel final en la pantalla.
    - **GLSL (OpenGL Shading Language):** Se escribirán *shaders* (pequeños programas que corren en la GPU) para implementar de forma personalizada el modelo de iluminación y el efecto de *Cel Shading*.
    - **Modelado poligonal:** Técnica para representar la geometría de los objetos 3D.
    - **Animación por Keyframes:** Para definir los momentos clave de la animación del ataque.

Desarrollo del proyecto

Fase 1: Creación del Mundo y Sistema de Control

Para la construcción del entorno virtual, se utilizó la librería **Assimp** para la gestión de assets, permitiendo la carga de modelos complejos en formato .fbx.

Geometría y Transformaciones

Se implementó un sistema de matrices de transformación (Modelo, Vista y Proyección) utilizando la librería **GLM**. Esto permitió ubicar al personaje principal ("Goku") en el espacio de mundo y aplicar transformaciones de escala y rotación en tiempo real.

- **Matriz de Modelo:** Gestiona la posición (x, y, z) y la orientación del personaje.
- **Matriz de Vista:** Define la posición de la cámara virtual.
- **Matriz de Proyección:** Establece la perspectiva y el campo de visión (FOV) de 45 grados.

```cpp
glm::mat4 projection = glm::perspective(glm::radians(45.0f), (float)SCR_WIDTH / (float)SCR_HEIGHT, 0.1f, 100.0f);
glm::mat4 view = glm::lookAt(cameraPos, gokuPos + glm::vec3(0.0f, 1.5f, 0.0f), cameraUp);

// --- RENDERIZADO DEL CIELO (SKYBOX/DOME) ---

// CAMBIO: En lugar de glCullFace(GL_FRONT), usamos Disable.
// Esto obliga a dibujar la esfera por ambos lados.
glDisable(GL_CULL_FACE);

ourShader.use();
ourShader.setMat4("projection", projection);
ourShader.setMat4("view", view);

glm::mat4 modelSky = glm::mat4(1.0f);
modelSky = glm::translate(modelSky, gokuPos);
ourShader.setMat4("model", modelSky);

// Poner la luz muy alta
ourShader.setVec3("lightPos", glm::vec3(0.0f, 200.0f, 0.0f));

glActiveTexture(GL_TEXTURE0);
// Asegúrate que skyTexture se cargó bien (si no, prueba con floorTexture aquí para testear)
glBindTexture(GL_TEXTURE_2D, skyTexture);
ourShader.setInt("texture_diffuse1", 0);

skyDome.Draw();

// CAMBIO: Volver a activar el Culling normal para Goku y el resto
glEnable(GL_CULL_FACE);
glCullFace(GL_BACK); // Aseguramos que vuelva al estándar
```

Implementación de las matrices de transformación en C++.

```cpp
// Matriz de Goku
glm::mat4 modelBase = glm::mat4(1.0f);
modelBase = glm::translate(modelBase, gokuPos);
modelBase = glm::rotate(modelBase, glm::radians(gokuAngle), glm::vec3(0.0f, 1.0f, 0.0f));
modelBase = glm::rotate(modelBase, glm::radians(-90.0f), glm::vec3(1.0f, 0.0f, 0.0f));
modelBase = glm::translate(modelBase, glm::vec3(0.0f, -1.0f, 0.0f));

// Outline
glCullFace(GL_FRONT);
outlineShader.use();
outlineShader.setMat4("projection", projection);
outlineShader.setMat4("view", view);
glm::mat4 modelOutline = glm::scale(modelBase, glm::vec3(1.02f, 1.02f, 1.02f));
outlineShader.setMat4("model", modelOutline);
currentModel->Draw(outlineShader);

// Normal
glCullFace(GL_BACK);
ourShader.use();
ourShader.setMat4("projection", projection);
ourShader.setMat4("view", view);
// Luz tipo SOL (Dirección fija desde arriba a la derecha)
ourShader.setVec3("lightPos", glm::vec3(50.0f, 100.0f, 50.0f));
ourShader.setVec3("viewPos", cameraPos);

glm::mat4 modelNormal = glm::scale(modelBase, glm::vec3(1.0f, 1.0f, 1.0f));
ourShader.setMat4("model", modelNormal);
currentModel->Draw(ourShader);
```

Implementación de las matrices de transformación en C++.

Cámara Orbital e Interactividad

Se desarrolló una cámara en tercera persona con comportamiento orbital. A diferencia de una cámara estática, esta permite al usuario rotar alrededor del personaje utilizando el mouse, calculando las coordenadas cartesianas a partir de coordenadas esféricas (radio, yaw, pitch). El movimiento del personaje se gestiona mediante el teclado (WASD), actualizando su vector de posición en cada frame.

Fase 2: Estilo Visual y Renderizado (Cel Shading)

Uno de los pilares del proyecto fue lograr que los modelos 3D no se vieran realistas, sino como dibujos animados planos, técnica conocida como *Cel Shading* o *Toon Shading*.

Algoritmo de Cel Shading

Se programó un **Fragment Shader** personalizado (basic.frag). En lugar de utilizar una iluminación de Phong suave, se implementó una discretización de la intensidad de la luz.

El algoritmo calcula el producto punto entre la normal de la superficie y la dirección de la luz. El resultado se "escalona" en niveles discretos (sombra, tono medio, luz) utilizando la función floor, eliminando los degradados suaves típicos del 3D convencional.

Resultado del efecto Cel Shading aplicado al modelo.

Técnica de Contorno (Outline)

Para reforzar la estética de anime, se implementó la técnica de **"Inverted Hull" (Casco Invertido)**. Esto requirió un renderizado en dos pases por cada frame:

1. **Pase 1:** Se dibuja el modelo ligeramente escalado (más grande) y de color negro sólido, invirtiendo el *Face Culling* (GL_FRONT) para mostrar solo las caras internas traseras.

2. **Pase 2:** Se dibuja el modelo normal con sus colores y texturas sobre el anterior.

El resultado es que el modelo negro del fondo sobresale únicamente por los bordes, creando un contorno perfecto en tiempo real.

Efecto de Outline generado mediante la técnica de Inverted Hull.

Fase 3: Animación y Mecánicas Avanzadas

Para dotar de dinamismo a la escena, se implementaron dos sistemas de animación distintos: por estados y procedimental.

Animación de Estados (Idle vs Run)

Dado que el motor fue construido desde cero, se implementó un sistema de intercambio de mallas (*Model Swapping*). El programa carga en memoria dos versiones del modelo: una en postura de reposo (*Idle*) y otra corriendo (*Run*). Dependiendo de la entrada del usuario (si la velocidad es mayor a 0), el motor renderiza la malla correspondiente, simulando la animación de movimiento.

Estados de animación del personaje controlados por input.

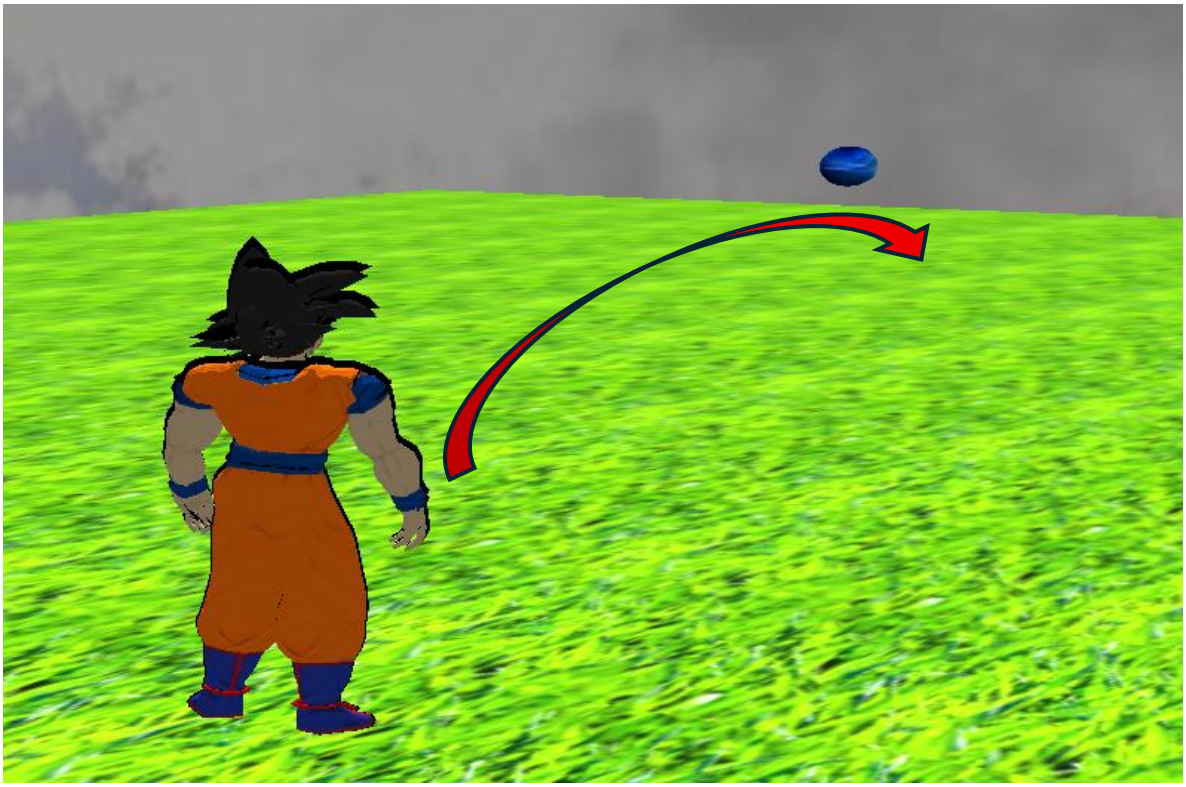
Ataque con Curvas de Bézier

Cumpliendo con la propuesta de implementar curvas paramétricas, se desarrolló la mecánica de "Ataque de Energía". La trayectoria del proyectil no es lineal, sino que sigue una **Curva de Bézier Cúbica**.

Se definieron cuatro puntos de control vectoriales (P_0, P_1, P_2, P_3):

- P_0: Origen (Mano del personaje).
- P_3: Destino (Objetivo en el suelo).
- $P_1\ y\ P_2$: Puntos de control elevados en el aire.

Mediante una función matemática que interpola estos puntos basada en el tiempo T (de 0 a 1), se logra que el poder describa un arco fluido y estético.



Trayectoria del ataque calculada mediante Curva de Bézier Cúbica.

```
// Bezier
glm::vec3 calculateBezier(float t, glm::vec3 p0, glm::vec3 p1, glm::vec3 p2, glm::vec3 p3) {
    float u = 1.0f - t;
    float tt = t * t;
    float uu = u * u;
    float uuu = uu * u;
    float ttt = tt * t;
    glm::vec3 p = uuu * p0;
    p += 3 * uu * t * p1;
    p += 3 * u * tt * p2;
    p += ttt * p3;
    return p;
}
```

Cálculo de la Curva de Bézier Cúbica.

Entorno y Acabado Final

Para finalizar la escena, se integraron elementos ambientales que dan contexto al modelo:

- **Skybox Esférico:** Se generó una esfera de gran escala con coordenadas de textura UV invertidas y una textura de cielo panorámico, simulando un horizonte infinito.
- **Iluminación Global:** Se configuró una luz direccional ("Sol") para iluminar uniformemente el terreno y el personaje, evitando oscurecimientos irreales al alejarse del centro.

Conclusiones

Se ha logrado integrar exitosamente los componentes teóricos de la graficación por computadora en una aplicación funcional. Se cumplieron todos los requisitos de la propuesta original, destacando la correcta implementación de los shaders para la estilización visual y el uso de matemáticas avanzadas (Bézier y Matrices) para la lógica de juego. El resultado es una escena que emula decentemente la estética de los videojuegos de animación japonesa.

Referencias

[1] D. Hearn and M. P. Baker, Computer Graphics with OpenGL, 4th ed. Upper Saddle River, NJ, USA: Pearson Prentice Hall, 2011.

[2] J. de Vries, "LearnOpenGL," 2020. [Online]. Available: https://learnopengl.com/

[3] The Khronos Group, "OpenGL API," 2025. [Online]. Available: https://www.opengl.org/

[4] G. Sellers, R. S. Wright, and N. Haemel, OpenGL SuperBible: Comprehensive Tutorial and Reference, 7th ed. Indianapolis, IN, USA: Addison-Wesley Professional, 2015.

[5] Lake, A., & Jones, M. (2000). "Stylized Rendering Techniques for Games". In Game Developers Conference. San Jose, CA.