Proyecto de Computación Gráfica: Juego de Tanques para 2 Jugadores

Autor: Asier Cabo

Índice

1. Introducción

En este documento se describe el diseño e implementación de un juego de tanques para dos jugadores, desarrollado en C++ utilizando OpenGL (GLFW y GLAD), con soporte de texturizado, iluminación, audio y detección de colisiones. El proyecto integra técnicas fundamentales de computación gráfica aprendidas en la asignatura para ofrecer una experiencia interactiva en entornos 3D. Se usa de base y de inspiración para el desarrollo de este juego la entrega de la grua en OpenGL.

2. Arquitectura del Proyecto

2.1 Estructura de Directorios

```
ProyectoFinal/
├── assets/
│   ├── audio/          # WAV para fondo y efectos (shoot, hit, tank-moving, etc.)
│   └── textures/       # PNG de suelo, tanques, torretas, balas, muros y arbustos
├── include/
│   ├── Engine/         # Cabeceras: Audio, Cámara, Shaders, Draw, Objetos, Input
│   └── Utils/          # Constantes, geometrías (Vertices, esfera), stb_image.h
├── src/
│   ├── Engine/         # Código fuente
│   └── Utils/          # glad.c, utilidades varias
├── shaders/            # GLSL: shader.vert y shader.frag
├── main.cpp            # Inicialización, bucle principal, llamadas a init y update
└── Makefile            # Compilación y limpieza
```

2.2 Flujo de ejecución (main.cpp)

1. Inicializar GLFW y crear ventana.
2. Cargar punteros de OpenGL con GLAD.
3. Configurar estados de OpenGL (depth test, cull face, clear color).
4. Compilar y linkar shaders (vertex & fragment).
5. Cargar texturas con stb_image.
6. Inicializar audio con SDL_mixer.
7. Crear escena: posicionar tanques, construir perímetro y muros aleatorios.
8. En el bucle principal:
   - Procesar entrada de usuario (tanques y cámara).
   - Actualizar física y colisiones.
   - Renderizar escena completa.
   - Reproducir sonidos según eventos.
9. Liberar recursos y finalizar.

3. Técnicas Gráficas y de Juego

3.1 Renderizado y Shaders

- **Vertex Shader** (`shader.vert`): aplica transformaciones modelo-vista-proyección con GLM, calcula posición del fragmento (`FragPos`), normal transformada y coordenadas UV.
- **Fragment Shader** (`shader.frag`): implementa Phong shading:
- Carga de textura con `texture(ourTexture, TexCoord)` y descarte de fragmentos transparentes (alpha < 0.1).
- Componente **ambient**: `ambientStrength * ambientColor`.
- Componente **difusa**: `max(dot(norm, lightDir), 0.0) * sunColor`.
- Componente **especular**: `pow(max(dot(viewDir, reflectDir), 0.0), 32) * specularStrength * sunColor`.
- Resultado final: `(ambient + diffuse + specular) * baseColor`.

3.2 Texturizado

- Carga de imágenes PNG y JPEG mediante **stb_image.h**.
- Texturas asignadas a tanques, torretas, balas, muros y suelo.
- Uso de unidades de textura en OpenGL y configuración de parámetros (WRAP, FILTER).

4.3 Iluminación

- Luz direccional “sun” con dirección y color uniformes.
- Combinación de tres componentes de iluminación: ambiente, difusa y especular.
- Parámetros configurables por uniformes (`ambientStrength`, `sunColor`, `sunDirection`).

3.4 Cámara y Transformaciones

- Cámara fija inicial en `(25, 25, 25)`, mirando al origen.
- Proyección perspectiva: fov 45º, aspecto `SCR_WIDTH/SCR_HEIGHT`, near=1, far=300.
- `glm::lookAt(cameraPos, cameraTarget, cameraUp)` para la vista.
- Controles con teclado (flechas, V/B) para desplazar la cámara en dirección, derecha y eje vertical.

3.5 Entorno y Árboles

- Árboles representados como planos (quads) con textura PNG que incluye canal alfa.
- Funciones `dibujarCubo()` construyen tanques a partir de cubos escalados.

3.6 Colisiones y Física Básica

- Detección **AABB** con consideración de rotación en múltiplos de 90º.
- Colisiones evaluadas entre balas y muros/tanques mediante método `intersects()`.
- Al colisionar: invocar `onHit()` en el objeto impactado y eliminar el proyectil.

4. Audio y Efectos Sonoros

- Empleo de **SDL_mixer** para reproducir WAV de fondo (`bgSound.wav`) y efectos (`shoot.wav`, `hit.wav`, `tank-moving.wav`, `wallHit.wav`).
- Gestión de canales para audio concurrente.
- Inicialización y carga en `AudioInit.cpp` y reproducción en eventos de `AudioManager`.

5. Diseño de Objetos y Administración de Recursos

5.2Tanques (Tank)

- **Atributos**: posición, rotación, IDs de textura (cuerpo y torreta). La salud está implementada, y al recibir disparos se disminuye, por lo que sería sencillo implementar funcionalidades como la destrucción de tanques o los puntos, pero por falta de tiempo no se ha desarrollado.
- **Métodos**: `draw()`, `onHit()`, entrada de disparo.

5.2 Proyectiles (Bullet)

- **Atributos**: posición, velocidad, propietario.
- **Métodos**: `update(deltaTime)`, `draw()`, `isAlive()`, `intersects()`.

5.2 Muros (Wall)

- **Generados aleatoriamente** con `initMuros()`: posición, largo, alto y rotación (0º o 90º).

5.4 Gestión de Texturas y VAOs

- Inicialización de VAOs/VBOs/EBOs en `Draw.cpp`.
- Carga de texturas y configuración con `initTextures()`.

6. Entrada de Usuario

Gestionado en el archivo ProcessInput.cpp.

- **Tanques (player1)**: WASD para movimiento, X para disparar, QE para rotar torreta.
- **Tanques (player1)**: IJKL para movimiento, M para disparar, UO para rotar torreta.
- **Cámara**: teclas de flecha + V/B para desplazarse.