Teoría y Práctica

Informática Gráfica

https://ismael-sallami.github.io

https://elblogdeismael.github.io

Autor: **Ismael Sallami Moreno**

UNIVERSIDAD DE GRANADA

21 de septiembre de 2025

Licencia

Índice general

Parte I

Teoría

Capítulo 1

Introducción

Para acceder a los materiales debemos de entrar con la cuenta go.

La asignatura de Informática Gráfica tiene como objetivo principal proporcionar los fundamentos teóricos y prácticos necesarios para el desarrollo de aplicaciones gráficas interactivas. A lo largo del curso, se estudian conceptos clave como la representación y modelado de escenas, técnicas de visualización 2D y 3D, y el uso de APIs gráficas modernas. Además, se exploran algoritmos esenciales como la rasterización y el ray-tracing, así como su aplicación en contextos como videojuegos, simuladores y producción de efectos visuales.

Capítulo 2

Aplicaciones gráficas interactivas y visualización

2.1 Aplicaciones gráficas interactivas y proceso de visualización 2D y 3D

Un programa gráfico se define como un programa que constituye un sistema computacional. Pueden ser interactivos o no interactivos. Los elementos esenciales de una AG son los modelos digitales y las imágenes o vídeos digitales que se usan.
Destacamos los eventos de entrada que son las acciones del usuario mediante las cuales se envía información a la aplicación. Las aplicaciones gráficas siempre se estructuran como un bucle de gestión de eventos, podemos mencionar los siguientes pasos:

1. Esperar al evento y recuperar datos.

2. Procesar el evento actualizando el modelo y los parámetros de visualización.

3. Visualizar el modelo actualizado con los nuevos parámetros.

Las aplicaciones gráficas pueden dividirse en dos tipos:

- 2D: los objetos se definen en planos, pueden incluir algunos 3D (sombras). Ejemplos de ello puede ser un diagramas de barras, videojuegos 2D. En este proceso de visualización se produce una imagen a partir de un modelo y parámetros como entradas.

- 3D: se sitúan en un espacio tridimensional, incluyendo texturas, materiales, fuentes de luz,... A la vez, pueden incluir figuras 2D. Ejemplos: videojuegos, simuladores,... En este proceso de visualización se usa como entrada el modelo de escena y unos parámetros.
  - En el modelo de escena distinguimos dos partes:
    - Modelo geométrico: conjunto de primitivas (polígonos, planos) que definen los objetos a visualizar.

    - Modelo de aspecto: parámetros que definen el aspecto de los objetos.

  - En los parámetros de visualización encontramos:
    - Cámara virtual

    - Viewport

2.2 Rasterización versus ray-tracing

En este apartado vamos a ver algoritmos de rasterización.

```
1: Inicializar el color de todos los pixels al color de fondo.
2: for cada primitiva P del conjunto E do
3:     S ← conjunto de pixels de la imagen I cubiertos por P
4:     for cada pixel q de S do
5:         c ← color de la primitiva P en el pixel q
6:         Asignar el color c al pixel q en I
7:     end
8: end
```

Este pseudocódigo describe el proceso básico de rasterización, que es una técnica utilizada para convertir primitivas geométricas (como polígonos) en una imagen pixelada. El algoritmo recorre cada primitiva del conjunto de entrada, determina qué píxeles de la imagen están cubiertos por ella, calcula el color correspondiente para cada píxel y lo asigna a la imagen final. Es un enfoque eficiente para generar imágenes en aplicaciones gráficas interactivas.

```
1: Inicializar el color de todos los pixels
2: for cada pixel q de la imagen I a producir do
3:     T ← subconjunto de primitivas de E que cubren q
4:     for cada primitiva P del conjunto T do
5:         c ← color de la primitiva P en el pixel q
6:         Asignar color c al pixel q en I
7:     end
8: end
```

Este pseudocódigo describe el proceso básico del algoritmo de Ray-tracing. A diferencia de la rasterización, aquí se invierte el orden de los bucles: se recorre cada píxel de la imagen y se determina qué primitivas geométricas lo afectan. Luego, se calcula el color del píxel en función de las primitivas que lo cubren. Este enfoque permite generar imágenes con mayor realismo, aunque suele ser más costoso computacionalmente.

En el algoritmo de Ray-tracing podemos optimizarlo de manera que la eficiencia sea O(log n) mediante la indexación espacial.

2.2.1 Rasterización

Se lleva a cabo en GPUs. Es preferible para aplicaciones interactivas y para la simulación de videojuegos, realidad virtual.

2.2.2 Ray-tracing

Respecto a la técnica de Ray-tracing:

- Suele ser más lento, pero consigue resultados más realistas.
- Preferibles para elementos no interactivos. En la actualidad se usa para la producción de animaciones y efectos especiales.
- Se ha usado en algunos videojuegos, pero requiere elementos computacionales de alto rendimiento.

2.2.3 El cauce gráfico en rasterización

Cauce gráfico se define como el conjunto de etapas de cálculo para la generación de imágenes. Las entradas se definen como primitivas. Un vértice es un punto 2D o 3D. EL cauce escribe en el framebuffer.

Hay dos pasos importantes:

1. Transformación: partiendo de las coordenadas se calculan las coordenadas de proyección.
2. Sombreado: cálculo del color de un pixel.
3. Hay otras como recortado de polígonos.

Cada primitiva se sitúa en un plano imaginario (plano de visión) situado entre el observador y la escena. La proyección puede ser perspectiva o paralela. En la rasterización, para cada primitiva se calcula que píxeles tienen su centro cubierto por ella. En el sombreado se usan los atributos de la primitiva para asignar color al píxel.

Etapas del cauce gráfico

1. Procesado de vértices
    1. Transformación: los vértices de cada primitiva se transforman para encontrar su proyección en el plano.
    2. Teselación y nivel de detalle: transformaciones avanzadas.
2. Post-procesado de vértices y montaje de primitivas
3. Rasterización
4. Sombreado

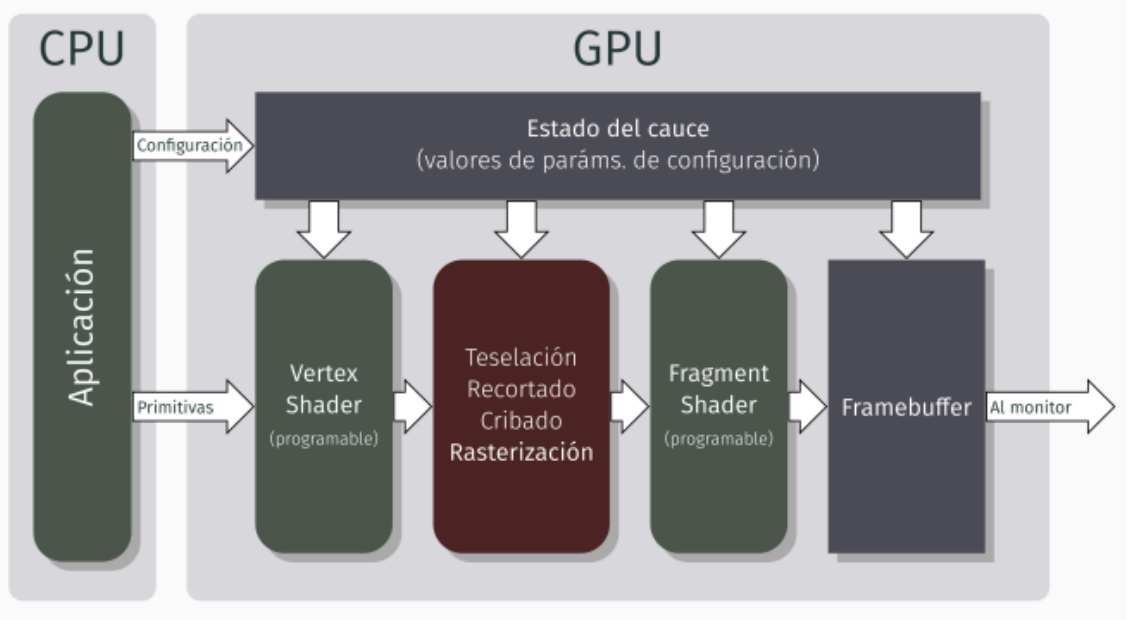

Figura 2.1: Esquema del cauce

2.3 APIs y motores gráficos

2.3.1 APIs para Rasterización, Ray-tracing y GPGPU

APIs de rasterización: conjunto de funciones para visualización 2D/3D, clases, interfaces. Son definidas sin ánimo de lucro, se puede dejar compilado en código intermedio.

APIs gráficas: estas proporcionan portabilidad y acceso simultáneo. La escritura en el framebuffer sigue siendo lenta, esto se soluciona usando GPU y enviando información de alto nivel a través del bus del sistema.

Como ejemplos podemos mencionar OpenGL, Vulkan, Metal, ...

Las APIs modernas son más eficientes aunque tienen ciertas desventajas como más complejidad y menos portabilidad.

Parte II

Prácticas

Capítulo 3

Introducción

Las prácticas de la asignatura se llevaran acabo con *Godot*.

Un motor de juego es una herramienta compleja y difícil de presentar en pocas palabras. Aquí hay una rápida sinopsis, que eres libre de reutilizar si necesitas una breve reseña sobre Godot Engine:

Godot Engine es un motor de videojuegos repleto de características, multiplataforma para crear juegos 2D y 3D por medio de una interfaz unificada. Provee un conjunto exhaustivo de herramientas comunes, para que los usuarios puedan enfocarse en crear juegos sin tener que reinventar la rueda. Los juegos pueden exportarse en un sólo clic a numerosas plataformas, incluyendo las principales plataformas de escritorio (Linux, macOS, Windows), plataformas móviles (Android, iOS), así como plataformas y consolas basadas en la web.

Godot es completamente gratis y de código abierto bajo la permisiva licencia MIT (Licencia del Instituto Tecnológico de Massachusetts). Sin condiciones, sin regalías, nada. Los juegos de los usuarios son suyos, hasta la última línea de código del motor. El desarrollo de Godot es totalmente independiente y dirigido por la comunidad, lo que permite a los usuarios ayudar a dar forma a su motor para que coincida con sus expectativas. Está respaldado por la Godot Foundation (Fundación Godot) sin fines de lucro.[1]

Para la *descarga* de Godot debemos de hacerlo mediante el enlace de la web oficial [2].

Nota

El equipo de Godot no puede proporcionar una exportación de consola de código abierto debido a los términos de licencia impuestos por los fabricantes de consolas. Independientemente del motor que use, lanzar juegos en consolas siempre es mucho trabajo. Puedes leer más sobre eso en el Soporte de consolas en Godot.

Se recomienda encarecidamente leer el *started* de Godot [3].

3.1 Aprendiendo a programar con Godot

En Godot, es posible escribir código utilizando los lenguajes de programación GDScript y C#. Para aprender los lenguajes relacionados, podemos realizar el curso de manera interactiva que nos ofrecen.[4]

Ejemplo de GDScript comparando con otros lenguajes que ya conocemos

[1] https://docs.godotengine.org/es/4.x/
[2] https://godotengine.org/es/
[3] https://docs.godotengine.org/es/4.x/getting_started
[4] https://gdquest.github.io/learn-gdscript

```gdscript
# GDScript
func take_damage(amount):
    health -= amount
    if health < 0:
        die()
```

```python
# Python
def take_damage(amount):
    health -= amount
    if health < 0:
        die()
```

```javascript
# Javascript
function takeDamage(amount) {
    health -= amount;
    if (health < 0) {
        die();
    }
}
```

Además, en el curso se nos ofrece la posibilidad de ir practicando como vemos en la figura 3.1.

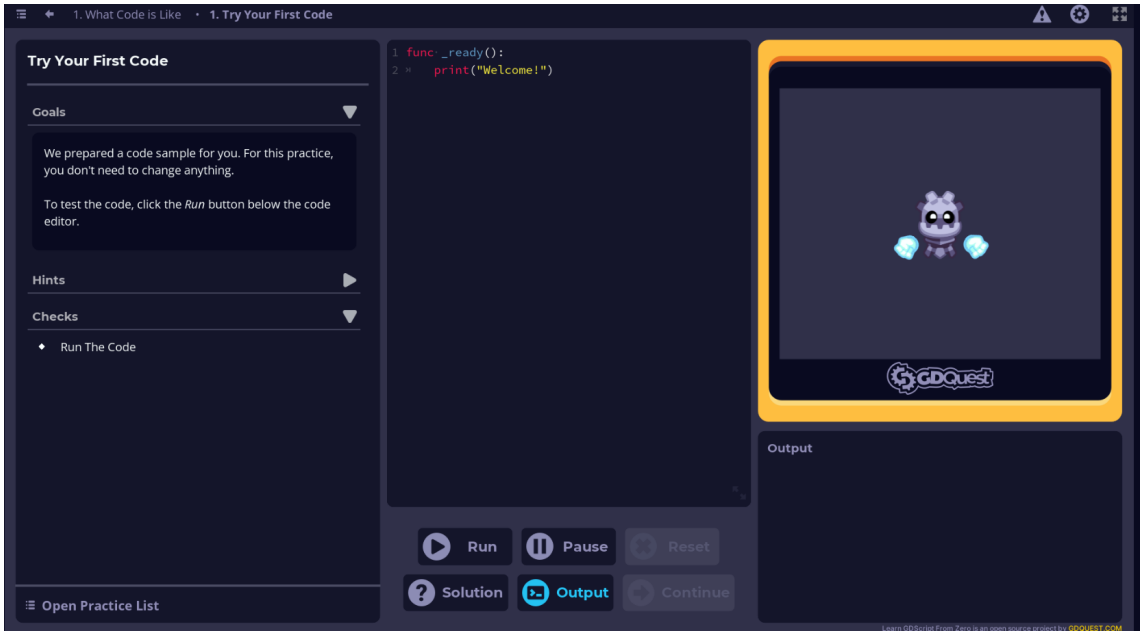

Figura 3.1: Imagen de práctica

3.2 Conceptos clave de Godot

Todos los motores de videojuegos giran alrededor de abstracciones que usas para crear tus aplicaciones. En Godot, un juego es un árbol de nodos agrupados en escenas. Los nodos pueden comunicarse entre sí mediante señales.

- **Escenas:** En Godot, puedes dividir tu juego en escenas reutilizables, como personajes, niveles o menús. Estas escenas pueden combinarse y anidarse para formar estructuras más complejas.
- **Nodos:** Las escenas están compuestas por nodos, que son los bloques de construcción básicos de tu juego. Godot ofrece una amplia biblioteca de nodos base que puedes combinar y extender.
- **El árbol de escenas:** El árbol de escenas organiza todas las escenas y nodos de tu juego. Representa la jerarquía y estructura general de tu proyecto.
- **Señales:** Los nodos emiten señales para comunicar eventos, como colisiones o interacciones. Esto permite una comunicación flexible entre nodos sin acoplamiento directo.
- **Sumario:** Los nodos, escenas, el árbol de escenas y las señales son los conceptos fundamentales de Godot. Estos elementos te permitirán estructurar y desarrollar tus juegos de manera eficiente.

Bibliografía

[1] Ismael Sallami Moreno, **Estudiante del Doble Grado en Ingeniería Informática + ADE**, Universidad de Granada, 2025.

[2] Universidad de Granada, *Diapositivas de la asignatura*, Curso 2025/2026.