DISEÑO TÉCNICO

SHIFT videogame

GAMELAB
UNIVERSIDAD DE OVIEDO

Alberto Carlos del Blanco Maraña
Madrid 28 de junio de 2007

Tabla de Contenidos

1. Introducción 4

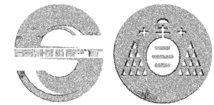
2. Esquema general 6

3. Detalle de los módulos 7
3.1. Motor Físico 7
3.2. Motor de Colisiones 7
3.3. Motor de Sonidos 7
3.4. Motor de Utilidades 8
3.5. Motor de Red 8
3.6. Motor Lógico 9

4. Bucle principal 10

Lista de Figuras

Imagen 1 : Esquema modular del motor gráfico 6
Imagen 2 : Diagrama de flujo del motor gráfico 11



1. Introducción

El presente documento detalla el diseño técnico para el videojuego SHIFT, como parte del proyecto fin de carrera realizado por Alberto Carlos del Blanco para la E.P.S.I.G. de la Universidad de Oviedo.

El videojuego se desarrolla a bajo nivel, utilizando únicamente las librerías gráficas DirectX, por lo que la mayor dificultad reside en el diseño y construcción de un motor gráfico en su totalidad.

Según el diseño propuesto para el videojuego, se requiere un planteamiento técnico que aporte la jugabilidad, pero sin perder en realismo, eficiencia o sencillez. En definitiva los criterios básicos que vamos a tener en cuenta para el diseño técnico son:

- **Eficiente**
  Hay que tener en cuenta que los videojuegos son quizás la rama de software que más exigencias tiene sobre el hardware. En este sentido uno de los aspectos mas complicados es la eficiencia en el diseño y programación. Debemos construir algoritmos computacionalmente sencillos, optimizando las zonas de código críticas, minimizando el uso de la memoria, etc.

- **Sencillo**
  Este aspecto choca con el anterior, ya que en muchas ocasiones es necesario sacrificar la sencillez del diseño en pro a la eficiencia del mismo. En todo caso se tratará de llegar a un compendio razonable entre ambos criterios, primando en cualquier caso la sencillez, ya que la orientación del proyecto es más académica que profesional.

- **Comprensible**
  Pero aparte de construir un motor gráfico sencillo, deberá ser comprensible. De esta forma, el proyecto podrá aprovechar de los beneficios que le aporta estar publicado bajo una licencia abierta. Es decir se debe garantizar, tanto a nivel de diseño, como de programación y documentación, que el proyecto pueda ser retomado por terceras personas.

- **Genérico**
  Pero además, los aspectos técnicos deben basarse en un diseño genérico, de forma que no haya dependencias o cualidades específicas para videojuego en cuestión.

- **Independiente**
  De forma similar, se debe garantizar un aislamiento entre los aspectos técnicos del proyecto y el diseño del videojuego, de forma que el motor gráfico sea independiente de las cualidades de SHIFT como videojuego.

- **Reutilizable**
  Acorde con los tres criterios anteriores, el motor gráfico debe diseñarse pensando en su posible reutilización o escalabilidad. Por lo la propuesta de este proyecto debe pensarse cómo un motor gráfico (parte técnica) y una aplicación para utilizarlo (videojuego SHIFT).

- **Modular**
  Por último, se debe respetar la modularidad en todo el diseño del videojuego. Por ello, una de las claves del diseño técnico va a ser la orientación a objetos.

Acorde a la modularidad del sistema, el motor gráfico se dividirá en diferentes módulos más sencillos para tratar cada uno se los aspectos principales del videojuego, que serán:

- **Motor Físico**
  Para gestionar la geometría, texturas y parámetros físicos de los objetos

- **Motor de Colisiones**
  Para gestionar las colisiones entre los objetos físicos que tengan esta cualidad.

- **Motor de Sonidos**
  Para manipular los buffers de sonido asociados a los objetos físicos.

- **Motor de Utilidades**
  Para mantener la cámara, la luz, el interfaz de usuario y un sistema de carga externa de modelos mediante XML, dotando al sistema de gran versatilidad en este aspecto.

- **Motor de Red**
  Para compartir los parámetros necesarios que habiliten el modo multijugador

- **Motor Lógico**
  Adicionalmente el sistema constará de un módulo que implementará la lógica propia del videojuego, siendo esté la única parte no genérica para el motor físico.

A continuación, en los siguientes apartados veremos la interacción y descripción en detalle de cada uno de estos módulos que componen el motor gráfico del videojuego.

2. Esquema general

En el siguiente diagrama se muestra una primera visión modular del motor gráfico, dónde se puede tener una primera impresión del ámbito de cada módulo y las interacciones que tienen entre ellos:

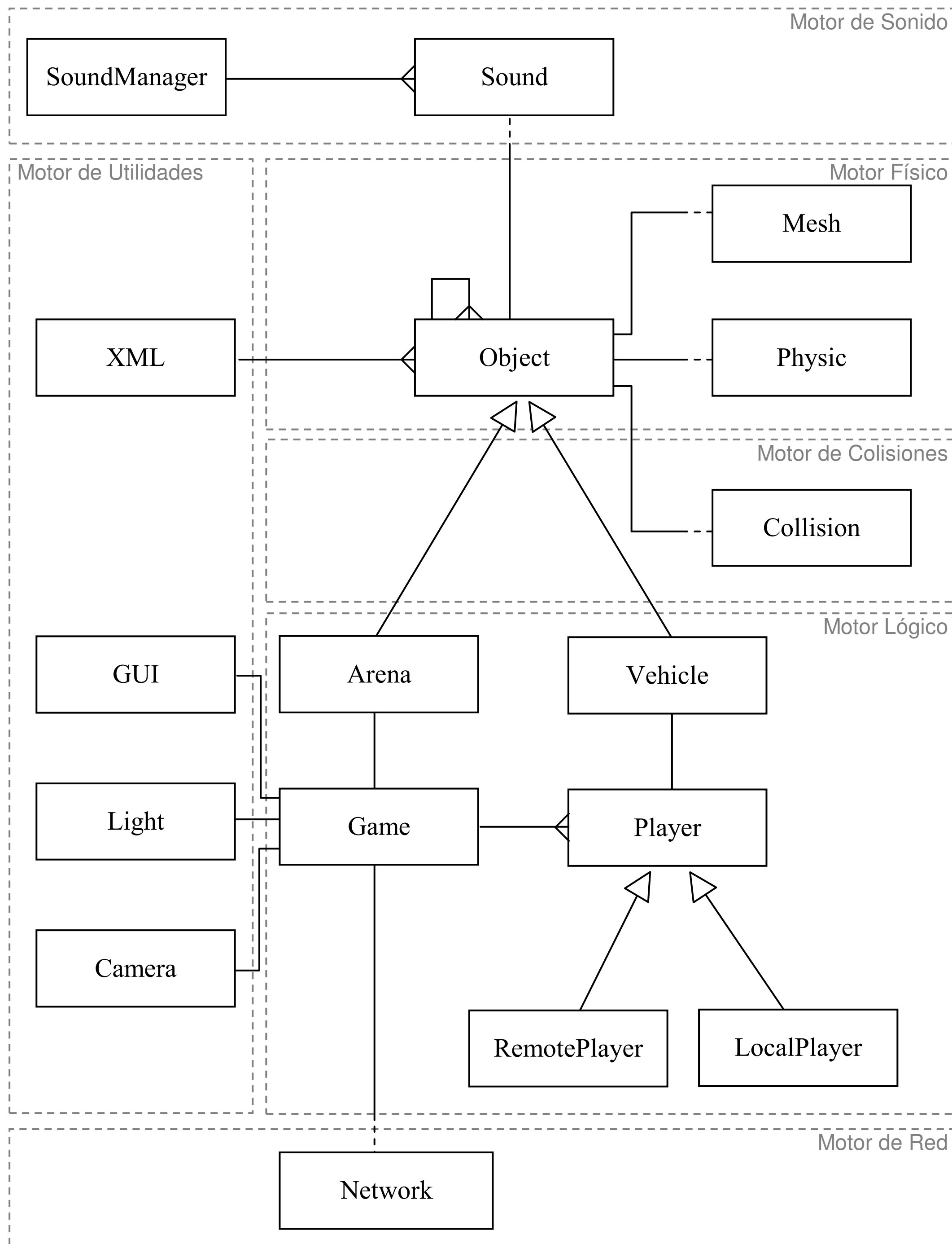
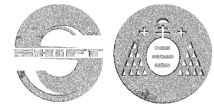


Imagen 1 : Esquema modular del motor gráfico



3. Detalle de los módulos

3.1. Motor Físico

Puede considerarse el módulo principal del motor gráfico. Permite gestionar los objetos en sí, manteniendo su geometría, texturas y parámetros físicos.

Se compone a su vez de las siguientes clases:

- **Object**
  Clase para abstraer el concepto de objeto dentro del motor gráfico. Es la clase que centraliza todas las propiedades técnicas y lógicas de cada elemento del videojuego

- **Mesh**
  Permite gestionar la geometría y texturas del objeto.

- **Physic**
  Permite mantener los datos físicos del objeto, como son posición, escalado, rotación velocidades lineales, angulares, aceleraciones lineales, angulares y fricciones.

3.2. Motor de Colisiones

Para gestionar las colisiones entre los objetos físicos que tengan esta cualidad, para ello se abstraen las colisiones con la clase:

- **Collision**
  Clase que permite implementar toda la lógica de colisiones entre objetos, como son la detección y el intercambio de parámetros físicos. Con el fin de simplificar el motor gráfico, todas las colisiones se reducirán al caso rectángulo-rectángulo en dos dimensiones.

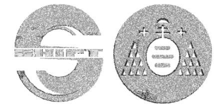
3.3. Motor de Sonidos

Para manipular los buffer de sonido, situándolos en el entorno 3D bajo la misma ubicación que los objetos a los que están asociados.

Se compone a su vez de las siguientes clases:

- **SoundManager**
  Gestor de todos los elementos de sonido de la aplicación.

- **Sound**
  Clase para abstraer el buffer de cada uno de los sonidos de la aplicación.



3.4. Motor de Utilidades

Para mantener la cámara, la luz, el interfaz de usuario y un sistema de carga externa de modelos mediante XML, dotando al sistema de gran versatilidad en este aspecto.

Se compone a su vez de las siguientes clases:

- **XML**
  Permite cargar o almacenar de forma persistente datos y estructuras de la aplicación con formato XML. Se usará este interfaz para el almacenamiento persistente, tanto para los modelos, como la configuración, etc.

- **GUI**
  Esta clase implementa el interfaz de usuario, incluyendo toda la lógica interna de navegación de menús y opciones de usuario.

- **Light**
  Clase para definir parámetros de luz ambiente, especular y difusa dentro de la aplicación. Para facilitar la programación no se implementan focos direccionales de luz, sino que sólo se configura la luz global de la escena.

- **Camera**
  Clase que abstrae el concepto de cámara para el dispositivo gráfico, con todos sus parámetros de foco, posición y rotación. Dentro de esta clase se implementan los algoritmos de seguimiento de objetos, cambios de vista, etc.

3.5. Motor de Red

Para compartir los parámetros necesarios que habiliten el modo multijugador, implementado por la clase:

- **Network**
  Implementa las funciones de red que nos permiten compartir los datos de la aplicación entre varios dispositivos de una forma P2P, es decir sin un servidor dedicado.

3.6. Motor Lógico

Adicionalmente el sistema constará de un módulo que implementará la lógica propia del videojuego, siendo esté la única parte no genérica para el motor físico.

Se compone a su vez de las siguientes clases:

- **Arena**
  Tipo de objeto con características especiales para contener un mapa de juego o arena concreta, incluyendo gran cantidad de parámetros de juego que implementan la mayor parte de la lógica.

- **Vehicle**
  Tipo de objeto con parámetros específicos para implementar los vehículos, incluyendo física y sonidos adicionales para ellos.

- **Game**
  Clase que gestiona toda la información referente a una partida en curso, desde el tipo de juego, hasta la arena seleccionada y los jugadores que participan.

- **Player**
  Clase que gestiona todos los datos de cada uno de los jugadores de la partida en curso.

- **RemotePlayer**
  Tipo de jugador remoto al equipo, aquí se mantienen parámetros específicos para gestionar este estado remoto.

- **LocalPlayer**
  Tipo de jugador local en el equipo

4. Bucle principal

Como hemos visto, el motor gráfico se compone de diferentes clases que implementan cada uno de los aspectos relevantes del mismo, tales como la física, las colisiones, los sonidos, la red, etc.

Cada una de estas clases compartirá un framework de cuatro operaciones básicas, que en cada clase concreta realizará las tareas adecuadas para la misma:

- **Initialize** Esta operación sólo se hace una vez antes de utilizar cada instancia. Las operaciones habituales dentro de esta función son las de inicialización, carga externa y registro de la instancia en el motor, tras haberlo hecho el objeto puede entrar a formar parte del motor gráfico.
- **Update** Esta operación se hace en cada iteración del bucle principal del motor gráfico. En ella se calcula el nuevo estado de cada instancia respecto la última iteración y el tiempo que pasó desde entonces. Es decir, si un objeto se encontraba en movimiento y pasaron 0,05 segundos respecto la anterior iteración, se deberá modificar su ubicación considerando sus parámetros físicos de velocidad, dirección, rozamiento, etc.

  La idea es que el entramado de clases anterior interactuará entre sí para ir conformando el nuevo estado de la escena respecto el estado anterior.
- **Render** Esta operación se hace tras la anterior, en cada iteración del bucle principal del motor gráfico. En ella, cada instancia comunicará al dispositivo de video (DirectX) las modificaciones necesarias para representar la instancia en la siguiente imagen renderizada.
- **Release** Función contraria a la de inicialización, para dar de baja una instancia del motor gráfico. Sólo se ejecutará una vez.

Gráficamente, el diagrama de flujo refleja el uso de las anteriores funciones para cada una de las instancias que haya en la escena del motor gráfico:

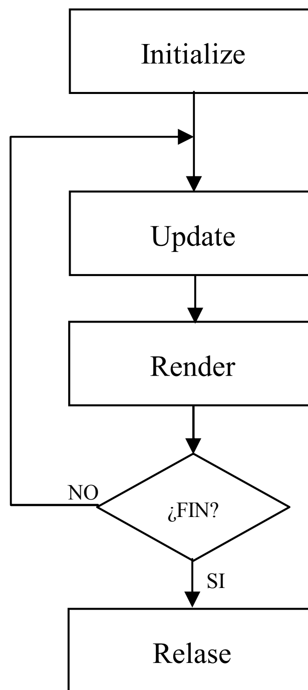


Imagen 2 : Diagrama de flujo del motor gráfico

Como vemos, una vez inicializadas las instancias, el motor entrará en un bucle dónde se actualizarán y dibujarán los objetos iterativamente.

Alberto Carlos del Blanco Maraña
Madrid 28 de junio de 2007