Motores gráficos

Sergio Herrero
Sergio Ballesteros
Liviu Adrián Dodan

Índice

1- Tipos de motores gráficos

Hoy en día es imposible desarrollar un buen videojuego sin un motor gráfico, por lo tanto, la elección de este es muy importante. A la hora de elegir un motor u otro hay varios factores a tener en cuenta, ya seas una gran empresa o un estudio pequeño tendrás que elegir cuales son las características que mejor se amoldan al producto que quieres crear. Para encontrar el motor que necesitas debes pensar en los siguientes factores:

1.1- Licencias

Una licencia es un contrato entre dos partes, el licenciante, que crea un producto y tiene los derechos sobre este, y el licentario, que usa el producto bajo las condiciones que le establece el licenciante, como puede ser la libertad de estudio del código fuente o los plazos de duración y territorios en los que se aplican las licencias. Dependiendo de las condiciones que el licenciante ponga sobre su software podemos hablar de software libre o propietario.

Los motores gráficos son un software y como tal se tienen que regular bajo licencias. Las formas más comunes de licencia que tienen los motores gráficos en la actualidad son:

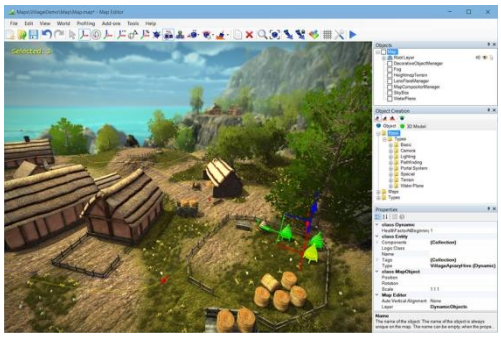
- **GNU y sus derivadas**: Se trata de una licencia de derechos de autor usada por el mundo del software libre que garantiza que el usuario final tiene la libertad de acceder al código fuente del software, estudiarlo, distribuirlo , poder modificarlo y distribuir copias de las modificaciones realizadas. Además se encarga de proteger el software mediante prácticas como el copyleft que impide que nadie pueda restringir la libertad del software ni de los productos que deriven de él. Algunos ejemplos de motores gráficos bajo esta licencia son:

ID TECH 4

SPRING ENGINE

- **MIT**: Es una licencia de software originaria del Instituto Tecnológico de Massachusetts, o también conocida como licencia X11. Es una licencia de software libre permisiva lo cual pone muy pocas limitaciones a la hora de usar el software que se encuentra bajo esta licencia. Algunos ejemplos de motores gráficos bajo esta licencia son:

OGRE ENGINE



STEPMANIA ENGINE

- Software propietario: Se trata del software del cual no se tiene acceso a su código fuente, por lo tanto no se puede estudiar su funcionamiento ni modificarlo. El creador tiene los derechos de distribución y los usuarios solo tienen derecho a usar el software bajo las condiciones del creador. Entre los motores de videojuegos comerciales más famosos se encuentran:

DECIMA ENGINE

4A ENGINE

1.2- Plataformas

Una plataforma es la combinación entre hardware y software en el cual una aplicación se ejecuta, es decir, la máquina que hace funcionar al programa (Hardware) y todos los programas que hacen que funcione(Software), no es lo mismo programar un videojuego para Windows que para Android y lo mismo pasa respecto a un ordenador y una Tablet.
Haremos una distinción entre plataformas de hardware, que comprenderán por así decirlo las diferencias "físicas" a la hora de programar, y las plataformas de software que tendrá su diferenciación en el apartado del software.

Plataformas de hardware

Las plataformas de Hardware establecen sus diferencias en base al tipo de arquitectura de la máquina o del procesador. En ordenadores, por ejemplo, una de las arquitecturas más utilizadas en la actualidad es la x86-x64. Las máquinas más comunes para trabajar con motores de videojuegos son:

- **Ordenadores**
- **Consolas**: como por ejemplo Play Station 4, Nintendo Switch o Xbox One.
- **Dispositivos móviles y Tablet**

Plataformas de software

Con plataformas de software nos referimos a sistemas operativos, entornos de programación o una combinación de ambas
Algunos ejemplos de plataformas software son:

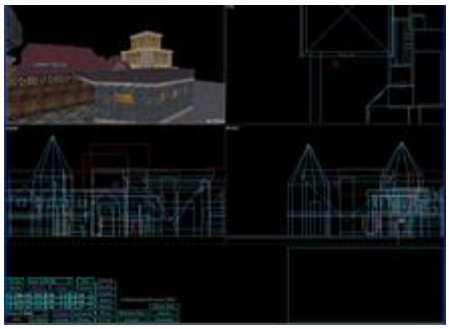
- **Android**: Uno de los sistemas operativos más comunes en dispositivos móviles y el que más juegos creados tiene en la PlayStore. Algunos de los motores de juegos que se usan para programar en esta plataforma son Allegro, Antyriad Gx y Defold.
- **iOS**: El sistema operativo para dispositivos móviles de Apple que funciona únicamente en máquinas de la misma compañía. Al ser formar parte de una de las empresas más importante de informática es sencillo encontrar motores de videojuegos compatibles con este sistema operativo, algunos ejemplos son Flare 3D, GamePlay 3D y Kivy.
- **Windows**: Una de las distribuciones de software más importantes, con aportaciones tanto en dispositivos móviles como en servidores y ordenadores. Algunos ejemplos de motores de videojuegos compatibles con este sistema operativo son RPG Maker, Unigine y Unity.

Una vez explicadas las plataformas software y hardware cabe destacar que un motor de videojuegos puede estar enfocado a una sola plataforma o a varias.

Motores monoplataforma

Un motor monoplataforma puede ser implementado en una única plataforma. Algunos ejemplos de motores de videojuegos que sólo pueden soportar una plataforma son:

DARK ENGINE(Windows)



Decima(Play Station 4)

Motores multiplataforma

Cuando decimos que un motor de videojuegos es multiplataforma nos referimos a que ese motor puede ser implementado como mínimo en dos plataformas y como máximo en todas las plataformas informáticas. Dentro del software multiplataforma podemos hacer dos diferenciaciones, existen los motores que necesitan una compilación individual para cada plataforma y por otro lado están los motores que se pueden ejecutar directamente en cualquier plataforma.

Existen varias formas de escribir una aplicación multiplataforma:

Una de ellas sería crear varias versiones del programa, cada una compatible con una plataforma diferente, por lo tanto un mismo programa dependiendo de la plataforma que lo vaya a hacer funcionar tendrá un código fuente u otro. Es la solución más sencilla pero a la vez la más cara, no es nada práctico escribir código por cada plataforma que queramos implementar.

Otra solución es llevar a cabo la llamada abstracción de la plataforma, que consiste en concentrar los detalles propios de la plataforma en una capa de abstracción. El objetivo es encapsular los detalles de las interfaces de programación de las distintas plataformas en una interfaz homogénea. Una capa de abstracción con estas características se denomina Middleware. El Middleware garantiza que la interfaz que provee se encuentra implementada en su totalidad en todas las plataformas con las que es compatible.

Los desafíos más comunes a los que se enfrenta la programación multiplataforma son:

- A la hora de realizar pruebas la cosa se complica debido a que cada versión está soportada por una plataforma diferente y esta puede estar programada en un lenguaje diferente lo cual implica comportamientos diferentes.
- Las diferentes plataformas tienen diferentes convenciones de interfaces de usuario, por lo tanto no se puede hacer una interfaz común para todas las plataformas.
- Cada plataforma puede tener sus formatos de paquetes nativos y a la hora de programar debe tenerse esto en cuenta.
- Al programar para varias plataformas pueden surgir fallos de seguridad.
- Algunos ejemplos de motores de videojuegos multiplataforma son:

CHROME ENGINE 6

CLICKTEAM FUSION

Como dato curioso cabe destacar que el primer cross-play entre plataformas se produjo en 1998 con la salida de la consola Sega Dreamcast que venía equipada con un modem el cual le permitía jugar con los usuarios de pc a juegos como 4x4 Evo, Maximum Pool, Quake 3 Arena y Phantasy Star Online.

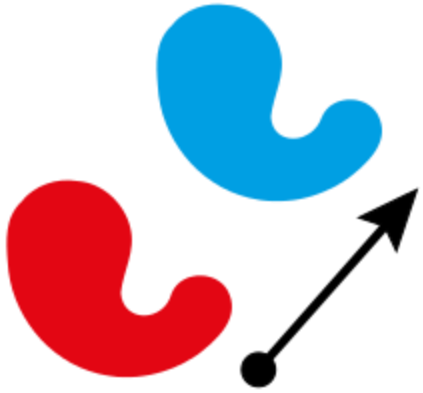
1.3- 2D

Los motores de videojuegos implementan un motor gráfico, el cual puede ser en 2D (dos dimensiones) o 3D (tres dimensiones).

Los gráficos 2D combinan modelos geométricos llamados vectores, imágenes digitales, texto, funciones matemáticas, ecuaciones, etc… Estos componentes pueden ser modificados mediante transformaciones geométricas en dos dimensiones como movimientos de translación, rotación y escalado.

Transformaciones geométricas

Traslación



La traslación consiste en mover cada punto de un objeto de un lugar del espacio a otro en una determinada dirección. La distancia y dirección de la traslación viene determinada por un vector.

La fórmula que que se aplica a la hora de realizar una traslación en el espacio es la siguiente:

Posición Final = Posición Inicial + vector de desplazamiento

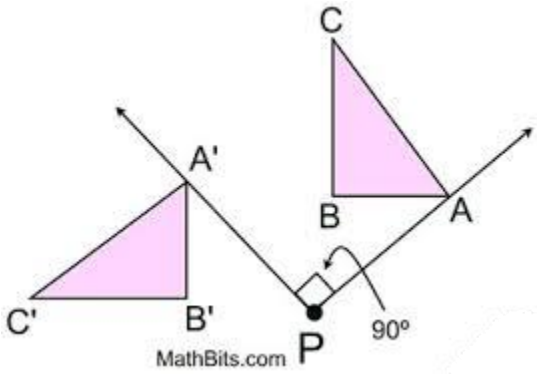
Posición Final = Posición Inicial($x_0+\Delta x$,$y_0+\Delta y$)

Rotación

La rotación de un objeto bidimensional consiste en el movimiento de todos los puntos de dicho objeto respecto a un punto un determinado ángulo.

En términos matemáticos una rotación se representa mediante una matriz R en la cual debemos especificar los puntos de los que se conforma nuestro objeto en el espacio en forma de vectores y el ángulo que va a interactuar con nuestro objeto.

Teniendo la matriz R:



$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}.$$

Se le aplica a nuestro vector de posición:

$$\begin{aligned} x' &= x\cos\theta - y\sin\theta, \\ y' &= x\sin\theta + y\cos\theta. \end{aligned} \qquad \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}.$$

Al aplicar esto a cada uno de los puntos de nuestro objeto habremos conseguido la rotación de este. Las rotaciones más comunes son:

$$R(90°) = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix}$$ (90° counterclockwise rotation)

$$R(180°) = \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix}$$ (180° rotation in either direction – a half-turn)

$$R(270°) = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$ (270° counterclockwise rotation, the same as a

Escalado

Escalar un objeto consiste en aumentar o disminuir el tamaño de un objeto respecto a un punto aplicando un factor de escalado. Esto se puede representar mediante una operación de matrices, donde S es la matriz de escalado y la matriz P son cada uno de los puntos del objeto.

$$S_v = \begin{bmatrix} v_x & 0 & 0 \\ 0 & v_y & 0 \\ 0 & 0 & v_z \end{bmatrix} \quad S_v p = \begin{bmatrix} v_x & 0 & 0 \\ 0 & v_y & 0 \\ 0 & 0 & v_z \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \begin{bmatrix} v_x p_x \\ v_y p_y \\ v_z p_z \end{bmatrix}$$

Al final se obtiene una matriz con la posición de los puntos una vez aplicado el escalado.

Dibujado de los gráficos

Una forma de crear imágenes es empezar con un lienzo de píxeles (Bitmap) de algún color y dibujar sobre él. Existen programas que se dedican a realizar este proceso como puede ser Gimp aunque la mayoría de motores de videojuegos cuentan con librerías de gráficos 2D que implementan multitud de herramientas para la realización de dibujos.

En este apartado se crean las formas, líneas y textos. Se les da un color especificado por el usuario. Muchas bibliotecas proporcionan gradientes de color, los cuales ofrecen una gran cantidad de variedad de color, útiles a la hora de realizar sombras o fondos. Los píxeles también pueden tomar sus colores de texturas reales para que los dibujos fuesen más realistas.

El hecho de darle color a un pixel significa que estás dibujando encima del píxel anterior, sin embargo, muchos sistemas tienen implementada la opción de pintar píxeles con colores transparentes o translúcidos para crear una mayor cantidad de oportunidades a la hora de colorear, esto se llama inversión de color.

Layers

Los modelos de gráficos en dos dimensiones no permiten la creación de figuras en tres dimensiones lo cual dificulta la creación de profundidad, sombras, etc… De la necesidad e crear estos fenómenos surge la idea de layer, que en español significa capa. Un Layer es una capa dibujada de forma que las partes que no están dibujadas son transparentes o translúcidas, esto hace que al crear varias capas y ponerlas unas sobre otras de la sensación de profundidad y mejore mucho la calidad de la imagen. Es un sistema muy bueno a la hora de dibujar ya que cada capa es independiente de las demás y puedes editar por separado sin que afecte a las demás. A la hora de juntar todas las capas se posicionan en orden decreciente de profundidad, es decir, las capas que representan objetos que están más lejos se ponen las primeras.

Algunos motores de videojuegos que trabajan con gráficos en 2D son Anura y Build Box.

1.4- 3D

El software con gráficos 3D se forman mediante la combinación del modelado 3D y el renderizado 3D.

Modelado

El modelado en 3D es el proceso de desarrollo de una imagen tridimensional a partir de una colección de puntos en el espacio unidos entre sí mediante entidades geométricas como triángulos, líneas, superficies, etc. Los Modelos 3D se pueden diferenciar en dos categorías:

- Sólidos: Definen el volumen de un objeto, es decir, la totalidad del cuerpo tanto en su exterior como en su interior. Son más realistas pero también son más difíciles de crear.
- Contornos: Sólo representan la superficie, como si el objeto a crear estuviese vacío. Son los más usados en videojuego ya que requieren de una menor capacidad de procesamiento.

Hay tres formas de representar un modelo en 3D:

- Modelo Poligonal: Consiste en unir cada uno de los puntos que forman el objeto. Hoy en día es el método más usado porque son fáciles de renderizar pero al unir los puntos se crean polígonos y para crear curvas hacen falta muchos polígonos.

- Modelado de curvas: La superficie se define a partir de curvas. Con este modelo no hay mucho problema a la hora de representar superficies curvas pero supone una mayor carga en el renderizado.
- Escultura digital: Se trata nada más y nada menos que la versión digital de la disciplina de la escultura. Es el más nuevo de todos y se compone de tres fases: Fase de desplazamiento, fase volumétrica y fase de teselación dinámica.

Renderizado

La renderización es un proceso gráfico mediante el cual generamos una imagen digital a partir de un modelo en tres dimensiones. El objetivo es crear una imagen realista desde cualquier perspectiva del modelo.
En videojuegos el proceso de renderización ocurre a tiempo real a velocidades de entre 20 y 120 fotogramas por segundo (fps), tener menos de 24 fps haría que el juego fuese incómodo de jugar ya que ese es el límite de fps que el ojo humano necesita ver para percibir la idea de movimiento en el juego.
Algunos motores de videojuegos que trabajan con gráficos 3D son BRender y Cafu Engine.
En el apartado siguiente se explica en detalle el funcionamiento del componente de renderización del motor gráfico.

2- Integración en entornos de desarrollo o IDE propios

Al principio los videojuegos de programaban en código máquina lo cual era un trabajo muy difícil. El programador trabaja con un editor de texto y un ensamblador, que traduce el lenguaje ensamblador en código máquina. Con el tiempo surgieron los sistemas de 16 bits y el lenguaje C pasa a usarse mayoritariamente a la hora de programar videojuegos. Con el tiempo los juegos creados empiezan a necesitar de una mayor velocidad a la hora de cargar gráficos queda casi desechada la idea de programar con algún lenguaje que no sea C.
Estos juegos que van necesitando de una mayor potencia centran su desarrollo en la creación de un motor propio que ejecute el núcleo central del juego. Este motor se ocupa de comprobar las acciones del usuario mediante su uso de elementos de entrada de la interfaz, de calcular los efectos sobre los elementos del juego, recalcular posiciones y estados, y generar los gráficos y sonidos que permitan visualizar las acciones y cambios del sistema.
Con el tiempo todas las empresas grandes que quiere realizar un videojuego, en especial los juegos en 3D, necesitan desarrollar un motor que les ofrezca ventaja sobre su competencia lo que desemboca en experiencias de juego mejores y gráficos más realistas.
A la hora de programar un videojuego ya no se puede escribir en una hoja de texto plano debido a la complejidad del proceso, por eso es necesario el uso de IDEs que faciliten el proceso.

¿Qué es un IDE?

Un IDE o entorno de desarrollo integrado es una aplicación software que sirve para simplificar el proceso de desarrollo, unificando en un mismo espacio el control y acceso a los diversos archivos y librerías implicados en un proyecto, y facilitando opciones de depuración e integración de funciones y librerías dentro del mismo entorno, estas aplicaciones surgen ante el aumento de la complejidad en la escritura de programación. Está formado como mínimo por un editor de código y un depurador, además otros IDEs como Eclipse pueden tener compiladores e intérpretes. Suelen tener una interfaz visual intuitiva y fácil de usar ya que el objetivo de estas aplicaciones es mejorar las condiciones a la hora de programar.

¿Se puede programar un motor gráfico desde un entorno de desarrollo como eclipse?

Lo que voy a explicar se puede aplicar a una gran cantidad de motores de videojuegos pero pondré el ejemplo de un motor conocido y con el cual se puede encontrar más información.
Uno de los motores gráficos más conocidos es Unity que cuenta con su propio IDE, el cual al principio puede abrumar un poco con tantas opciones pero tiene una curva de aprendizaje empinada. En su IDE se programa en C#, un lenguaje muy conocido. Pero el hecho de que tengan un IDE no te obliga a usarlo, ¿Es recomendable usarlo? Sí, por algo se inventaron los IDEs, para ayudar al programador, pero se puede programar un juego para Unity desde por ejemplo Eclipse, lo que necesitamos es configurar el entorno para escribir en C# y crear el de proyecto de eclipse dentro de una carpeta en los proyectos de Unity. Una vez estemos preparados para empezar a programar cabe destacar que están disponibles librerías externas para hacer más sencilla la tarea.

3- Componentes de los motores gráficos

Un motor es una pieza de software encargada de ejecutar un tipo de tarea para varias aplicaciones. Hay motores para bases de datos, transcripción de texto, gráficos, física, etc.



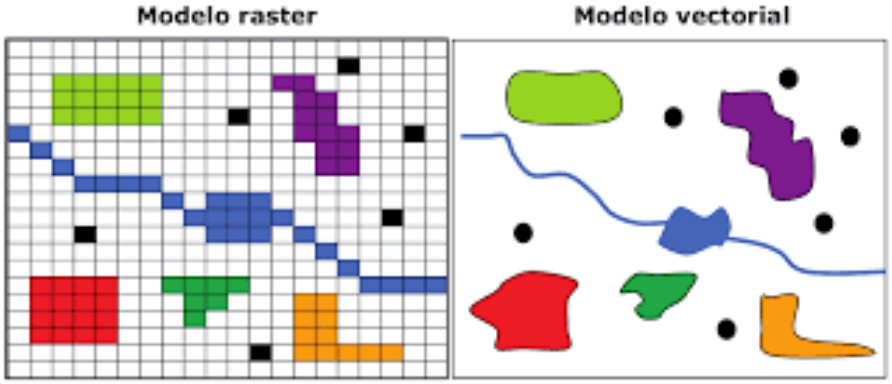
En este trabajo vamos a definir los motores gráficos y sus componentes. Un motor gráfico es un framework de software diseñado para desarrollar videojuegos, permitiendo al desarrollador dibujar gráficos en una pantalla a partir de modelos en 2D o 3D. Cada uno de los componentes se encarga de computar una tarea específica.

3.1- Componente de renderización

Renderizar consiste en generar una imagen a partir de un modelo de escena mediante un programa informático. El renderizado se suele basar en APIs como OpenGl, Direct3D o Vulkan, en otros casos se utilizan bibliotecas de bajo nivel como DirectX o Simple DirectMedia Layer. Existen varias tecnologías en cuanto a la renderización gráfica:

Rasterización gráfica

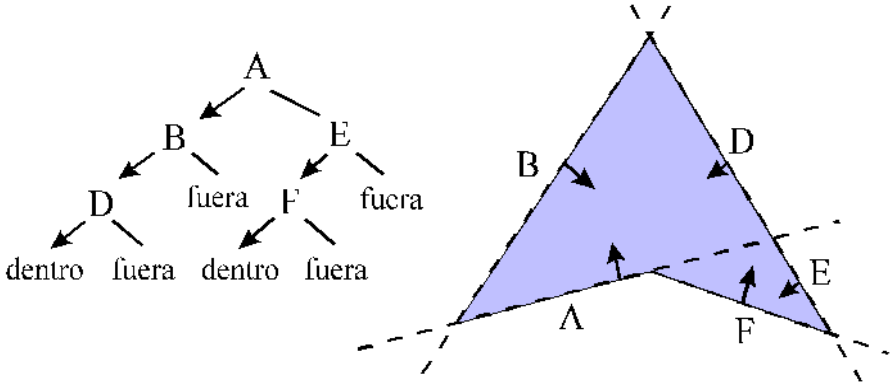
Es el método más común y antiguo, así como rápido comparado con otras tecnologías. Consiste en calcular la proyección de la escena en un punto en concreto que coincide con la cámara. Este método no genera una imagen en color, por lo que se usan métodos adicionales para colorear la imagen y dibujar las sombras.



Partición binaria del espacio

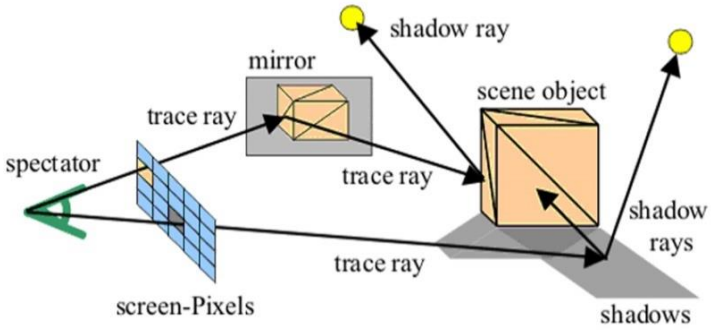
Este método consiste en dibujar la imagen subdividiendo el espacio y almacenándolo en un árbol de datos. En este árbol se guardarás en lo más alto el fondo de la imagen, que no cambiará y se dibujará antes, y en los nodos más bajos los elementos más cercanos, que serán los últimos en ser dibujados. Además los objetos en movimiento se almacenarán en un Z-Buffer que permitirá unirlos correctamente con el resto de la escena.

Una vez hecho este pre proceso se dibujarán los elementos con el algoritmo del pintor, es decir, primero el fondo y después los elementos más cercanos, superponiéndolos en la escena.

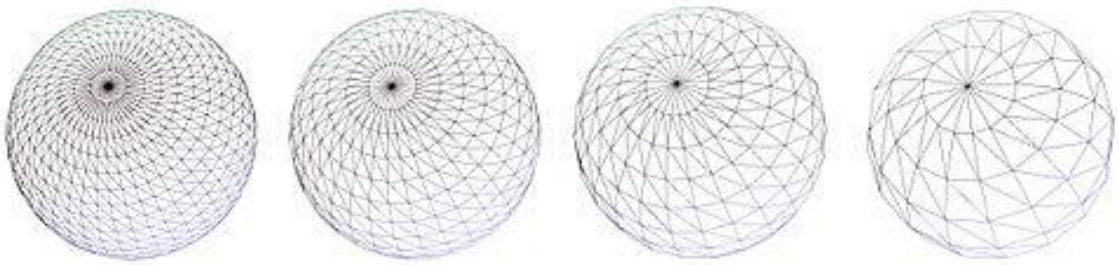


Ray tracing

Es un algoritmo basado en ray casting. Consiste en trazar rayos desde la cámara del jugador hasta la escena, teniendo en cuenta las propiedades de los diferentes materiales. La reflexión y refracción se calcula trazando rayos desde los puntos que se están sombreando hacia los puntos de luz. A partir de cada uno de los rayos se calcula el valor de cada pixel.



Al fin y al cabo todos estos algoritmos de renderización dependen de la resolución de la escena y los objetos, que vienen dados por modelos en 3D definidos por polígonos. Según la cantidad y el tamaño de polígonos de un modelo obtendremos mayor o menor definición. Tenemos que entender los modelos 3D como una malla cerrada de triángulos

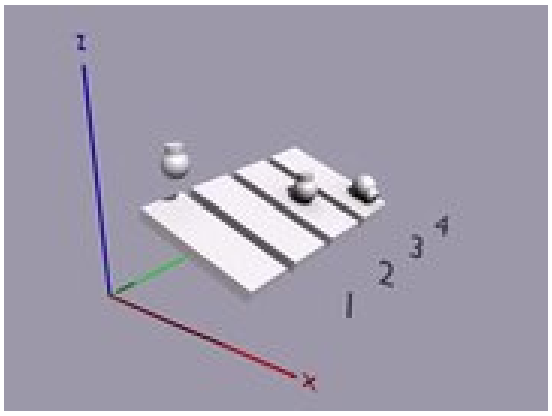


3.2- Motor de físicas

El motor de físicas es el encargado de implementar las mecánicas de sólidos rígidos como la detección de colisiones, dinámica de fluidos, dinámica de cuerpos deformables, etc. Debido a la complejidad de estos motores, las empresas prefieren implementar software de terceros en lugar de incorporar su propia versión. Uno de los más famosos y más utilizados es Havok Physics.

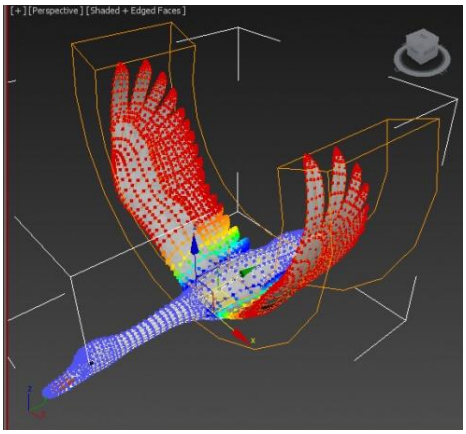
Leyes físicas

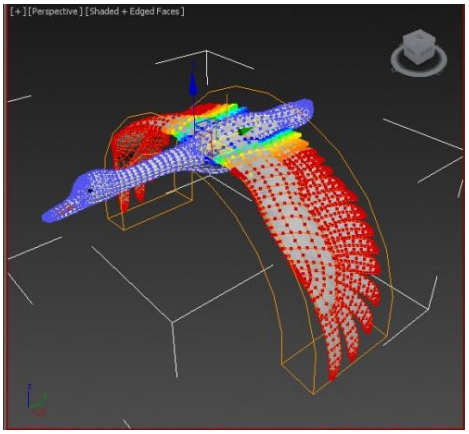
El motor de físicas añade movimiento realista al juego, simulando la gravedad. Una vez conseguido que el objeto se mueva deberemos de conseguir que no atraviese objetos. En la imagen se ve como reaccionaria el juego añadiendo cada vez una funcionalidad del motor de físicas. Como la gravedad solo actúa en el eje Z el objeto sólo se desplaza, por lo que habrá que desbloquear la rotación de los objetos. Además, se deben de establecer en la configuración de cada objeto constantes que definen el coeficiente de rozamiento. (Click en la imagen para ver el vídeo).



3.3- Animación

Las herramientas de animación toman un objeto con una o varias piezas y permite crear uniones “joints” de diferentes tipos que permiten crear una marioneta que permite el movimiento. Además de animar las uniones se pueden crear transformaciones y dobles del objeto como en el ejemplo, Una vez creados los ligamentos se deben de añadir animaciones. Esto se hará capturando diferentes posiciones separadas por un tiempo determinado, ya que la herramienta se encargará de animar la transición entre ellas. (Clic en cualquier imagen para ver animación).



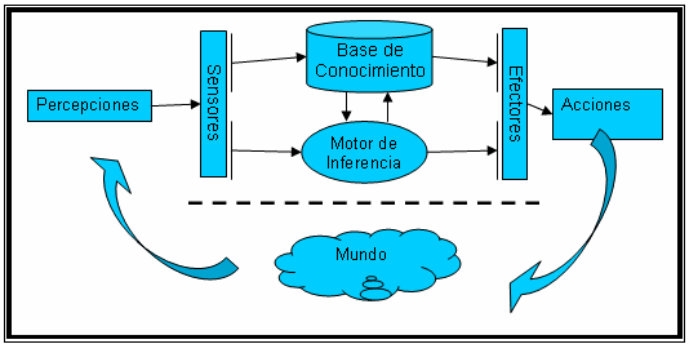


Sprites

En los videojuegos en 2D las animaciones son secuencias de 5-10 imágenes llamadas sprites, que superpuestas dan la sensación de animación, igual que los gifs. En la mayoría de los juegos, el personaje controlado por el jugador reproduce una animación cuando el jugador lo desplaza, salta o realiza una acción de ataque o defensa. (Clic en cualquier imagen para ver animación).

3.4- Inteligencia artificial

Una inteligencia artificial es una pieza de software diseñada para llevar a cabo acciones que simulan la inteligencia humana. En los videojuegos de hoy en dia se pueden observar aplicadas a PNJs, para desplazarlos por el mapa y en el caso de los shooters decidir cuándo disparar o cargar el arma. Se puede apreciar una buena IA cuando un PNJ responde a un evento y realiza una acción propia de una persona, por ejemplo cubrirse mientras carga el arma (el evento es una baja munición) o huir del jugador cuando tiene poca vida o coordinarse con otros PNJs.



4- Librerías (APIs)

Son un conjunto de subrutinas, funciones y procedimientos para ser utilizadas por otro software. Su principal propósito consiste en proporcionar un conjunto de funciones de uso general. También sirven para controlar el acceso a dispositivos de hardware y funciones de software que una aplicación puede no tener permiso para usar.

Al usar una API todo el desarrollo que se quiera realizar estará limitado por los métodos o funciones que esta incluya. Por eso se pueden crear APIs propias o usar APIs de terceros.

Las APIs de terceros pueden integrar servicios complejos en muy poco tiempo. Las APIs de grandes empresas realizan un mantenimiento continuo para asegurar el correcto funcionamiento de sus servicios, avisando de cambios y nuevas funcionalidades a sus integradores, pero generalmente las suelen ser muy costosas. Hacen que la prestación del servicio sea más flexible, es decir, se adaptan. También permiten que el contenido se incruste desde cualquier sitio o aplicación más fácilmente garantizando una entrega de información más fluida y una experiencia de usuario integrada.

5- Assets

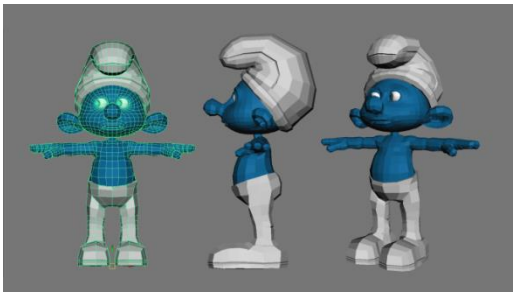
Los assets son todos y cada uno de los elementos que forman un videojuego, desde la escena hasta los modelos de los personajes pasando por las texturas y animaciones.

5.1- Modelo 3D

El modelado 3D es el proceso de desarrollo de una representación matemática de cualquier objeto tridimensional a través de un software especializado. Los modelos 3D representan un objeto tridimensional usando una colección de puntos en el espacio dentro de un espacio 3D, conectados por varias entidades geométricas tales como triángulos, líneas, superficies curvas, etc.

5.2- Animaciones

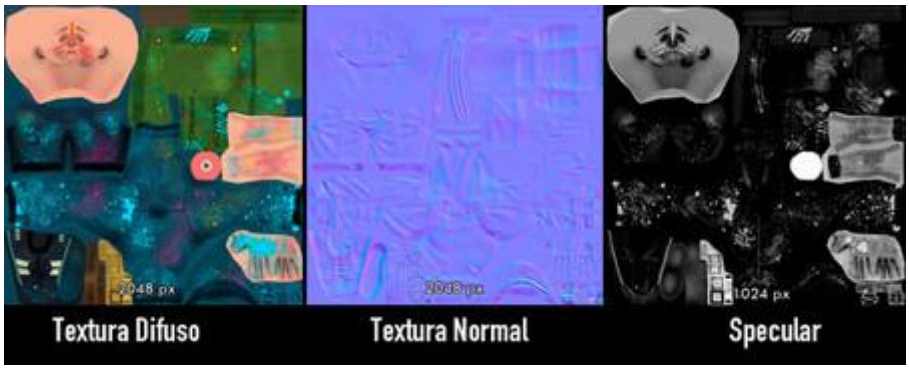
Las animaciones se guardan como archivos .fli, .cmf o .ani que el motor gráfico interpreta para representar movimiento. Van asociados a un modelo en 3d con articulaciones “joints” al que se denomina ragdoll (muñeco de trapo) que se usará para su visualización.



5.3- Texturas

En Unity se utiliza main maps o material parameters (parámetros de materiales) para dar texturas a un modelo. Esos parámetros son:

- Albedo: Controla el color base de la superficie.
- Metalic: Determina qué tan "metálica" es la superficie, es decir cuando una superficie es más metálica, refleja más el entorno y su color Albedo se vuelve menos visible.
- Normal Map: Son un tipo especial de textura que le permite agregar detalles de la superficie, como protuberancias, surcos y rasguños a un modelo que capta la luz como si estuvieran representados por una geometría real.
- Height Map: Concepto similar al Normal Map pero más costosa en rendimiento. Desplaza las áreas de la textura de la superficie visible, para lograr un tipo de efecto de oclusión a nivel de superficie.
- Occlusion: Se utiliza para proporcionar información sobre qué áreas del modelo deben recibir iluminación indirecta alta o baja.
- Emission: Controla el color y la intensidad de la luz emitida desde la superficie.
- Detail Mask o Máscara de Detalle: Permiten superponer un segundo conjunto de texturas encima de las texturas principales. También puede aplicar un segundo mapa de color Albedo y un segundo Normal Map.



6- Elementos del videojuego

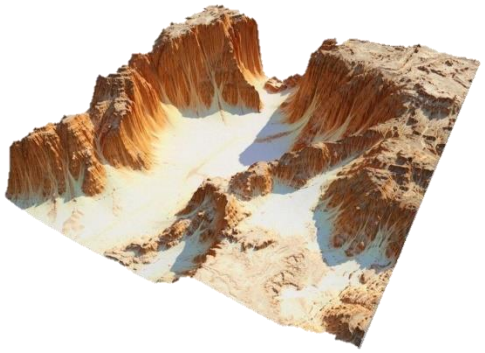
A parte de los Assets, cada motor gráfico incorpora ciertos elementos necesarios para el desarrollo del videojuego.

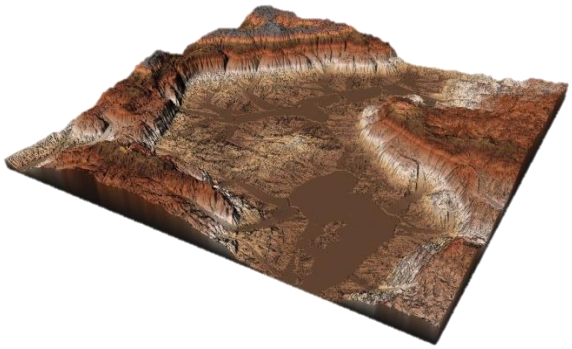
6.1- Escena

Una escena está formada por terrenos, texturas, modelos 3D con sus animaciones y todos los elementos del videojuego. Para liberar estrés al ordenador y facilitar el desarrollo se divide el juego en diferentes secciones que se cargan en memoria en el momento en el que son necesitados. En los videojuegos más recientes, que suelen ser multijugador, la escena es únicamente un mapa abierto. En el caso de los videojuegos con trama, se divide el espacio en el que se encuentra el jugador en mapas independientes en los que se cambia dinámicamente con la idea de que resulte imperceptible para el jugador. En otros casos se cambia de escena ante un evento como abrir una puerta o seleccionar un nivel en un menú, puesto que cada nivel se trata de una escena distinta, aunque comparta los assets con el resto del videojuego.

6.2- Terreno

Superficie sobre la que se colocan los objetos, personajes y se desarrolla toda la acción. Se le añade una o varias texturas para cambiar su apariencia y se modela con herramientas incluidas en el motor gráfico para añadir desnivel, vegetación, agua o pintar texturas en zonas concretas.

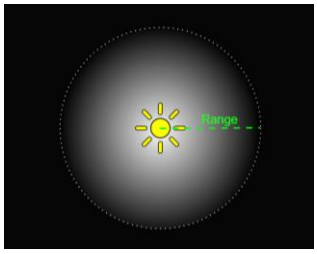




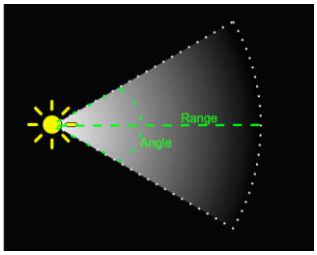
6.3- Luz

La escena no está iluminada hasta que introducimos un punto de luz, por lo que el motor gráfico no renderizará la imagen. Podemos destacar 3 tipos de iluminaciones diferentes:

- Direccional: Suele colocarse en el cielo para simular al sol, equivale a un foco lejano que ilumina todo el mapa por igual.
- Puntual: Se utiliza para simular lámparas, antorchas, etc. Es un punto de luz que ilumina un área determinada en todas las direcciones.
- Foco: Similar a la puntual, ilumina un área determinada pero solo en un ángulo determinado, como si se tratase de un foco.
- De área: Se define un rectángulo que se mantendrá iluminado, y una distancia en la que disminuirá la intensidad de la luz según se aleja del rectángulo
- De ambiente: No proviene de ningún punto en concreto, pero ilumina toda la escena por igual.
- Proveniente de elementos: Carteles luminosos, neones y otros gráficos que irradian luz con intención decorativa.



Puntual



Foco

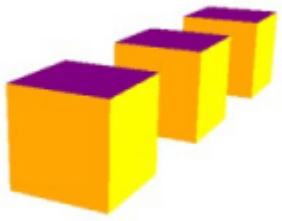
Área

6.4- Cámara

Es el punto desde el que se renderiza la imagen. Se puede modificar el ángulo de visión (fov) y el tipo de proyección que se muestra.

- Perspectiva: Se aprecia la diferencia de tamaño entre objetos dependiendo de la distancia entre ellos.
- Ortográfica: Se muestra su tamaño original independientemente de la distancia entre ellos.

Ortográfica



Perspectiva

7- Diferentes motores gráficos

Ejemplos	Lenguajes de programación	Ventajas	Inconvenientes
Unity	C#	Tiene una Asset Store muy completa que contiene principalmente extensiones, herramientas y paquetes de assets. Unity es multiplataforma de compilador cruzado, es decir que se puede compilar para cualquier plataforma con un solo script. A la hora de aprender a usar Unity, es muy sencillo por la estructura del editor y porque está escrito en C#. Comunidad muy amplia. No hay que pagar royalty.	No permite empezar con una base o plantilla sino que hay que empezar de cero con cada juego a desarrollar. Los proyectos pueden llegar a ocupar mucho espacio en disco si se usan assets complejos o modelos 3D de alta resolución. Suele exigir mucho a los ordenadores.
Unreal Engine 4	C++, Blueprints (Visual Scripting)	Es de código abierto. Ofrece una base donde empezar a la hora de desarrollar. Sistema de Blueprints, pero si se necesitan realizar tareas muy complejas será necesario programar en C++. Su rendimiento usa menos memoria y recursos que Unity. Mejor acceso a la depuración visual pudiendo controlar todo el proceso y el tiempo estimado.	A la hora de aprender a usar Unreal Engine es más complicado por su lenguaje de programación como por su editor. Cuando se gana dinero, si esa suma es mayor de 3000 dólares hay que pagar un royalty del 5% del beneficio por cada trimestre. Asset Store menos compleja que Unity.
Game Maker	Game Maker Language (GML)	Interfaz Drag & Drop Está más enfocado a la creación de juegos en 2D, pero también se puede hacer en 3D. Soporte para bibliotecas de enlace dinámico hechas en C++, Delphi y Pascal. Tiene integración con Git.	Solo soporta su propio lenguaje de programación. Licencia gratuita muy limitada. Falta de un editor interno y scripts específicos para animaciones.

8- Glosario

- **2D/3D**: Se refiere a las dimensiones que contiene el juego (X, Y, Z). Normalmente los juegos 2d usan una cámara cenital (vista desde arriba) o frontal (vista de frente), en algunas ocasiones podemos ver un "falso 3D" cuando el personaje se oculta tras un objeto o se añade una falsa dimensión en algunos niveles que percibimos en el tamaño del personaje.
- **Assets**: Activos necesarios a la hora de la creación de un videojuego.
- **Cámara en primera persona**: ángulo de cámara utilizado desde la perspectiva del personaje protagonista.
- **Cámara en tercera persona**: vista en la cual se ve el cuerpo entero del personaje principal.
- **Coeficiente de rozamiento**: Constante utilizada para calcular la fricción entre dos superficies
- **Compilador cruzado**: Compilador capaz de crear código ejecutable para otra plataforma distinta a aquella en la que el compilador se ejecuta.
- **Consola**: Es un aparato electrónico que se conecta a la televisión para poder jugar a videojuegos que previamente han sido introducidos en forma de disco , cartucho o en versión digital. Para jugar se usa un mando o controlador. Existe la versión portátil de las consolas las cuales destacan por no necesitar conectarse a una pantalla ya que la tienen incorporada, así como los mandos.
- **Cross-play**: Se produce cuando dos usuarios de diferentes plataformas pueden jugar juntos a un mismo videojuego mediante conexión a internet.
- **Librerías**: conjunto de implementaciones funcionales codificadas en un lenguaje de programación que ofrece una interfaz bien definida para la funcionalidad que se invoca.
- **Modelo de Escena**
- **Mono o MonoDevelop**: Entorno de desarrollo integrado libre y gratuito, diseñado para C# y otros lenguajes .NET como Nemerle, Boo o Java.
- **Motor gráfico**: Framework de software diseñado para desarrollar videojuegos. Todo motor gráfico debe de proporcionar al programador un motor de renderizado que computará gráficos 2D y 3D, un motor de colisiones de física de objetos, sonidos, animación, inteligencia artificial, comunicación con la red, etc.
- **Perspectiva top-down**: ángulo de cámara desde la perspectiva desde arriba. Comúnmente se utiliza en videojuegos 2D.
- **PNJ**: Personaje no jugador
- **Renderizar**: Generar una imagen a partir de un modelo de escena mediante un programa informático.
- **Royalty**: Pagos que una persona ha de realizar porque existe una patente ya creada por otra persona.
- **Voxel**: Es la representación de un cubo en el espacio tridimensional.
- **Z-Buffer**: Array de dos dimensiones (x,y) encargado de manejar la profundidad de los gráficos tridimensionales para decidir qué elementos de una escena son visibles o están ocultos. La calidad de la imagen generada dependerá de los bits del Z-Buffer entre otras cosas (8 bits muy baja calidad y 32 bits muy alta).

9- Webgrafía

https://es.wikipedia.org/wiki/Unity_(motor_de_videojuego)
https://en.wikipedia.org/wiki/Unity_(game_engine)
https://www.unrealengine.com/en-US/features
https://docs.unrealengine.com/en-US/Engine/index.html
https://es.wikipedia.org/wiki/GameMaker_Studio
http://docs2.yoyogames.com/
http://www.gamerdic.es/termino/motor-de-fisicas/
https://blogthinkbig.com/motores-graficos
https://giancr.com/animacion-de-movimiento-constante-con-loop-en-3d-studio-max/
http://www.pixelsmil.com/2012/04/crear-y-animar-un-sprite-tutorial-paso.html
http://ai-depot.com/GameAI/Design.htmlx
https://laleyendadedarwan.es/2017/12/21/cinco-cosas-buenas-y-cinco-no-tan-buenas-de-unity-3d/
https://docs.microsoft.com/es-es/dotnet/framework/wpf/graphics-multimedia/3-d-graphics-overview
https://docs.unity3d.com/Manual/index.html
https://www.eurogamer.es/articles/2013-02-04-destripando-los-motores-graficos
https://bbvaopen4u.com/es/actualidad/los-mejores-motores-graficos-de-videojuegos-i-soluciones-de-codigo-abierto
https://www.aragon.es/documents/20127/674325/Estado%20del%20arte%20GameEngines%20y%20su%20impacto%20en%20la%20industria.pdf/db827568-09ef-e931-01e8-d293b9fca834
https://hardzone.es/2018/05/06/motor-grafico-juegos/
http://diposit.ub.edu/dspace/bitstream/2445/65523/7/memoria.pdf
https://medium.com/@vancorso/motores-gráficos-y-motores-de-juego-e7d1d28f8157
https://slideplayer.es/slide/1841106/
https://blogthinkbig.com/motores-graficos
https://en.wikipedia.org/wiki/List_of_game_engines
https://en.wikipedia.org/wiki/2D_computer_graphics#2D_graphics_hardware
https://www.cocoschool.com/que-es-renderizado-3d/
http://ceur-ws.org/Vol-1196/cosecivi14_submission_14.pdf
https://docs.unity3d.com/Manual/index.html

10- Parte de cada uno

Tipos de motores gráficos _____ Sergio Ballesteros
Componentes de los motores gráficos _____ Sergio Herrero
Librerías (APIs) _____ Liviu Adrián Dodán
Assets _____ Liviu Adrián Dodán
Elementos del videojuego _____ Sergio Herrero
Integración en entornos de desarrollo o IDE propio _____ Sergio Ballesteros
Diferentes motores gráficos _____ Liviu Adrián Dodán
Portada, índice y maquetación _____ Sergio Herrero