Universidad de las Ciencias Informáticas

Facultad 5

Manipulación de vídeos digitales en Sistemas de Realidad Virtual para la herramienta gráfica SceneToolKit

Trabajo final presentado en opción al título de
Máster en Informática Aplicada

Autor: Ing. Orlay García Ducongé

Tutor: MSc. Yanoski Rogelio Camacho Román

La Habana

octubre del 2011

DECLARACIÓN JURADA DE AUTORÍA

Declaro por este medio que yo Orlay García Ducongé, con carné de identidad 84090404587, soy el autor principal del trabajo final de maestría Manipulación de vídeos digitales en Sistemas de Realidad Virtual, destinado a la herramienta SceneToolKit, desarrollada como parte de la Maestría en Informática Aplicada y que autorizo a la Universidad de las Ciencias Informáticas a hacer uso de la misma en su beneficio, así como los derechos patrimoniales con carácter exclusivo.

Y para que así conste, firmo la presente declaración jurada de autoría en la Habana a los ____ días del mes de _____ del año _____.

_____

Orlay García Ducongé.

AGRADECIMIENTOS

Un especial agradecimiento:

A quienes contribuyeron en mi formación como ingeniero y me han apoyado en mi superación.

A la dirección de la Facultad 5 por confiar en mí.

A los que han compartido conmigo en el proyecto y han dado su aporte a la SceneToolKit.

A los que siempre han apoyado el desarrollo de la STK.

A mi familia y amigos.

Muchas gracias.

RESUMEN

Este trabajo propone el desarrollo de funcionalidades para la representación de vídeos digitales en sistemas de realidad virtual. Surge a partir de la necesidad de incorporarle a la herramienta SceneToolKit, que se desarrolla en la Universidad de las Ciencias Informáticas (UCI), funcionalidades para la manipulación de vídeos digitales.

El trabajo se centra en los sistemas para manejo de vídeos digitales en sistemas de realidad virtual. Se propone como variante de solución la técnica de utilizar "*pixel buffer object*", para aumentar el rendimiento y permitir la visualización de vídeos de mayores dimensiones, a través de la generación de múltiples texturas dependiendo de un factor tiempo. El origen de la información, para la generación de estas texturas, son ficheros de vídeo digital de extensión ".avi". Se propone para leer estos ficheros de vídeo y reproducir el audio, el empleo de la biblioteca "vfw32" (*Video for Windows*), para el trabajo sobre la plataforma Windows, y de la biblioteca "*avifile*" en su versión 7.0, sobre la plataforma Linux. Se propone el uso de OpenGL para la visualización en un entorno virtual.

El resultado obtenido incorporó a la herramienta SceneToolKit funcionalidades para la manipulación de vídeos digitales. La arquitectura que se utilizó para la implementación permitió la creación de un módulo que puede ser utilizado de forma independiente de la SceneToolKit.

PALABRAS CLAVE

Vídeo
Vídeo Digital
Manipulación de Vídeos
OpenGL
Texturizar
Frame buffer object
Realidad Virtual

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS Y TABLAS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

INTRODUCCIÓN

A pesar de haber surgido hace ya algunos años, el fenómeno denominado Realidad Virtual (RV) continúa revolucionando el mundo, no sólo de la informática sino también de diversas áreas como la medicina, la arquitectura y la educación.

A finales de los 80 surgió la necesidad de crear un espacio totalmente interactivo, creado computacionalmente, por lo que los gráficos por computadoras entraron en una nueva época. Se comenzaron a remplazar los enfoques bidimensionales y de dibujo de líneas (2D) por las soluciones tridimensionales (3D), las cuales se han visto enriquecidas con sensaciones del mundo real a través de estímulos visuales, auditivos y de otros tipos que afectan al usuario de manera interactiva.

Con el surgimiento de los entornos virtuales, los mundos que se representan pueden no existir (ser producto de la imaginación humana), o una copia exacta del mundo real. Un entorno virtual es una forma de representar la realidad o la imaginación, pero de forma digital.

En el desarrollo de productos relacionados con la Realidad Virtual se destacan los videojuegos y los simuladores. En el caso de los videojuegos lo importante es la modalidad (cómo se tiene que jugar), pero las historias, que por lo general narran los juegos, gustan mucho y son mostradas a través de vídeos. En los simuladores, los vídeos se emplean sobre todo en las presentaciones, y en ocasiones se utilizan para dar instrucciones en un momento determinado. La utilización, cada vez más frecuente, de vídeos en entornos virtuales ha implicado que los usuarios de estas aplicaciones exijan su presencia y calidad.

En la facultad cinco de la Universidad de las Ciencias Informáticas, se comenzó a desarrollar videojuegos y simuladores. Para el desarrollo de estos se utilizaban motores de visualización comprados en el extranjero, teniéndose que pagar licencias y versiones actualizadas aún cuando no respondieran a todas las necesidades que se tenían, lo cual creaba una alta dependencia tecnológica. Luego se comenzaron a utilizar herramientas libres y de código abierto, pero estas brindaban poca flexibilidad lo que provocó un trabajo engorroso de las aplicaciones ya que había que profundizar en el código fuente de estas herramientas para

detectar y corregir algunos problemas que no se solucionaban a alto nivel, por lo que el trabajo se hacía más lento.

Entonces se decidió desarrollar un motor propio, así como herramientas de trabajo que permitieran desarrollar productos con calidad y rapidez. Esto permitiría tener un código propio que ofreciera seguridad y capacidad de actualización. [19]

La herramienta SceneToolKit (STK) surge a partir de la necesidad mencionada anteriormente. Esta herramienta inicialmente constituía un simple visualizador de entornos virtuales. Posteriormente se le fue añadiendo el trabajo con matrices y vectores, cámaras y luces, las geometrías, las animaciones, un grafo de escena y otras muchas funcionalidades [20], que permitieron el desarrollo de aplicaciones como el simulador Quirúrgico (ver Figura 15, Anexo 7), Paseo Virtual de la UCI versión 1.0 (ver Figura 16, Anexo 8), Meteorix (ver Figura 17, Anexo 9), el videojuego Energía para Aprender (ver Figura 18, Anexo 10), entre otros productos.

Durante el desarrollo de estos productos en varios momentos se pensó utilizar vídeos digitales en sus presentaciones, pero la STK no contaba con el soporte para ello. En el caso de Energía para Aprender, uno de los requerimientos de los usuarios era que apareciera un vídeo con una animación 3d al inicio de cada nivel, este requisito no fue pactado por la deficiencia de la herramienta en este sentido, ya que se contaba con poco tiempo para el desarrollo de este producto.

Partiendo de una búsqueda bibliográfica, se detectó que una gran cantidad de empresas productoras de videojuegos utilizan vídeos digitales, ejemplo la Compañía Electronic Arts [21], en toda su gama de videojuegos deportivos como Fifa [22], Need for Speed [23], NBA [24]. También en videojuegos como Proyecto IGI [25], se utilizan vídeos, para introducir las misiones y dar a conocer sus objetivos, de igual forma lo hace el Call of Duty [26] y Medalla de Honor [27]. También la compañía Blizzard Entertainment, productora del Word of Warcraft [28], ha encontrado en la utilización de vídeos, con gran número de efectos visuales y producidos en tercera dimensión, una forma de atraer a nuevos usuarios.

En el caso de los simuladores también se encontró que existen varias compañías que los producen, y que utilizan vídeos, ejemplo el simulador de conducción Virtual

GT [29], que los utiliza en la presentación del producto y otros como el simulador de tiro KD [30], los utiliza para proyectar escenas reales y hacer disparos sobre estas.

Si se pretendía desarrollar videojuegos y simuladores que compitieran con semejantes a nivel internacional se hacía necesario dotar a la STK de funcionalidades para la manipulación de vídeos digitales. Partiendo de los trabajos consultados se creó una solución que brindaba las funcionalidades requeridas, pero cuando se utilizaban vídeos de dimensiones mayores a 512 píxeles de ancho y de altura, el rendimiento de la aplicación, medida en *frames* por segundos, decaía notablemente, llegando a ser inferior a los 30 *frames* por segundos para vídeos de 800 píxeles de ancho y 600 píxeles de alto, en *hardware* de pocas prestaciones gráficas.

Analizando detenidamente la problemática anterior, se propuso como problema científico a resolver: ¿Cómo lograr la visualización de vídeos digitales, de mayores dimensiones, utilizando la herramienta SceneToolKit?

De esta forma se planteó como objeto de investigación, la manipulación de vídeos digitales.

Por lo que se tuvo como Objetivo General de Investigación: Incorporar a la herramienta SceneToolKit funcionalidades para la manipulación de vídeos digitales de dimensiones mayores a los 512 píxeles de ancho y alto.

A partir del objetivo seleccionado se define como campo de acción: la manipulación de vídeos digitales en sistemas de Realidad Virtual.

Se proponen las siguientes Tareas de Investigación:

- Identificación de las técnicas existentes para la manipulación de vídeos digitales en sistemas de realidad virtual.
- Identificación de los diferentes formatos de vídeos digitales que se utilizan en sistemas de realidad virtual, para seleccionar uno de ellos.
- Análisis de la arquitectura e interfaz de la herramienta SceneToolKit, para diseñar una forma conveniente de integración del módulo a realizar.

- Diseño de un módulo para la manipulación de vídeos digitales con una arquitectura compatible con la herramienta SceneToolKit.
- Implementación de un módulo para la manipulación de vídeos digitales con una arquitectura compatible con la herramienta SceneToolKit.

Con el desarrollo de este trabajo se pretende que la optimización del proceso de generación de texturas conlleve a un aumento del rendimiento, en *frames* por segundos, de la aplicación; lo que permitirá poder utilizar vídeos de dimensiones mayores a 512 píxeles, de ancho y alto, en las aplicaciones gráficas que se desarrollen utilizando la herramienta SceneToolKit.

CAPÍTULO 1: FUNDAMENTACIÓN TEÓRICA

En el presente capítulo se presentan los principales conceptos con los que se trabajó.

1.1 El vídeo

Etimológicamente la palabra vídeo proviene del verbo latino vidêre y significa "ver". [1]

El vídeo es la reproducción en forma secuencial de imágenes, que al verse con una determinada velocidad y continuidad dan la sensación al ojo humano de apreciar el movimiento natural. Además de la imagen, el otro componente esencial es el sonido. [2]

Se puede definir un vídeo como la mezcla en un único fichero de un conjunto de sonidos e imágenes que conjuntamente transmiten un mensaje al usuario.

En el aspecto gráfico, un vídeo se compone de una secuencia de imágenes denominadas fotogramas (también *frames* o cuadros), cada una de las cuales aparece en pantalla un determinado espacio de tiempo, suficiente para crear en el espectador la sensación de continuidad entre fotogramas, generando así la visión global de una única escena en movimiento.

Las imágenes pueden ser sintetizadas (creadas manualmente) o captadas a partir del entorno (vídeo).

Al igual que en el caso de las imágenes estáticas, los ficheros pueden ser muy voluminosos, y tienen unas capacidades de modificación limitadas. [3]

1.1.1 Digitalización de vídeos

Cuando se comienza a digitalizar un vídeo debe tenerse presente que es necesario digitalizar la información referente a la imagen y la información referente al audio.

Cuando la información a digitalizar es la de las imágenes, cada cuadro de la imagen es muestreado en unidades de píxeles, con lo que los datos a almacenar serán los correspondientes al color de cada píxel.

Para digitalizar una señal de vídeo analógico es necesario muestrear todas las líneas de vídeo activo.

Las imágenes de vídeo están compuestas de información en el dominio del espacio y el tiempo. La información en el dominio del espacio es provista por los píxeles, y la información en el dominio del tiempo es provista por imágenes que cambian en el tiempo. Puesto que los cambios entre cuadros colindantes son diminutos, los objetos aparentan moverse suavemente.

El valor de cada píxel es cuantificado con ocho *bits* para el caso de imágenes blanco y negro. En el caso de imágenes de color, cada píxel mantiene la información de color asociada; una imagen completa es una composición de tres fotogramas, uno para cada componente de color, así los tres elementos de la información de luminancia son cuantificados a ocho *bits*. [2]

Por otra parte, cuando la información a digitalizar es la del audio, consiste básicamente en realizar de forma periódica muestras de la señal. Según la periodicidad de muestras que se realice la conversión de analógico a digital será de mayor o menor calidad.

En la conversión analógica-digital del audio el muestreo es uno de los procesos que permite la digitalización de las señales. Estas muestras no se toman de forma aleatoria (al azar), sino que se toman intervalos fijos de tiempo. Cada muestra debe durar el mismo tiempo y efectuarse en el mismo intervalo. La velocidad a la que se hace este muestreo, es decir, el número de muestras que se toman por segundo, es lo que se conoce como frecuencia de muestreo.

Por muy eficaz que sea el muestreo realizado y por muy alta que sea la frecuencia de este, debe tenerse presente que siempre que haya un muestreo va a haber una cierta pérdida de calidad de la señal. Siempre habrá matices de la señal que no van a ser tenidos en cuenta, dado que no han sido muestreados, pero si se realiza un

muestreo de buena calidad, luego se podrá reconstruir la señal de manera prácticamente idéntica a la original.

Luego es necesario realizar un proceso de filtrado que consiste en la realización interna de un procesamiento de datos de entrada. El valor de la muestra de la entrada y algunas muestras anteriores (que previamente habían sido almacenadas) son multiplicados por unos coeficientes definidos. Finalmente los resultados de todas estas multiplicaciones son sumados, dando una salida para el instante actual.

Los filtros digitales se usan frecuentemente para atenuar las componentes de baja frecuencia, las componentes de alta frecuencia y dejar pasar un determinado rango de frecuencias de una señal y atenuar el paso del resto. [4]

1.2 Los vídeos en la realidad virtual

1.2.1 Tendencia a utilizar vídeos

En la actualidad una tendencia para usar vídeos, es como presentación en las aplicaciones gráficas, sobre todo en videojuegos y simuladores. Como es el caso de los juegos Need for Speed (ver Figura 9, Anexo 1), MVP (ver Figura 10, Anexo 2) y simuladores de auto como VirtualGT (ver Figura 11, Anexo 3) y de tiro como KD (ver Figura 12, Anexo 4). En otros se utilizan como transición entre misiones por ejemplo en juegos como Proyecto IGI (ver Figura 13, Anexo 5) y en ocasiones se pueden generar de acuerdo a situaciones específicas como es el caso del videojuego de fútbol Fifa (ver Figura 14, Anexo 6).

En el caso de los videojuegos, los vídeos constituyen una forma agradable y entretenida de conocer la historia del juego, es una forma más de cautivar a los jugadores introduciéndolos cada vez más en un mundo real o imaginario. Mientras que en los simuladores los vídeos son empleados sobre todo en las presentaciones y en ocasiones se utilizan para dar algún tipo de instrucción. En ambos casos los vídeos constituyen una manera de introducir a los usuarios en distintos tipos de misiones o situaciones específicas.

1.2.2 Vídeos en ambientes gráficos tridimensionales

En los últimos años, se han logrado avances realmente increíbles en los gráficos 3D. Los avances en vídeo digital no han sido menos impresionantes. Hace poco tiempo que vídeos bruscos e imágenes de pequeños tamaños representaban el marco teórico en vídeos por ordenador. En la actualidad se ha logrado captura de movimientos en tiempo real y vídeos de alta resolución.

En muchos aspectos, los gráficos 3D y el vídeo son tecnologías que se complementan. Con gráficos en 3D, se pueden crear mundos imaginarios que nunca existieron, y explorar e interactuar con ellos. La parte difícil es hacer que parezca realista, y es ahí donde aparece el ingenio de los programadores y artistas. El vídeo, aunque es de dos dimensiones y no es intrínsecamente interactivo, captura el mundo real en todos los detalles.

Sin embargo, dentro de la computadora, estas dos tecnologías tienen más en común de lo que parece.

Considerando el modo en que un objeto 3D se representa, el objeto empieza como un conjunto de puntos (vértices), que son transformados matemáticamente en coordenadas de pantalla 2D. Los detalles de la superficie del objeto, como el patrón de ladrillos en un edificio, se almacenan en una imagen de mapa de *bits* (textura), que en efecto es "pegado" a los lados del objeto.

Considerando el modo en que un vídeo se representa en una pantalla de ordenador, un área determinada de la pantalla es designada para mostrar el vídeo. Esto podría ser un rectángulo dentro de una ventana, o toda la pantalla. En cualquier caso, el área está definida por un conjunto de coordenadas de la pantalla que forman un rectángulo de destino. Cada fotograma de vídeo es un mapa de *bits* que se extiende para adaptarse al rectángulo de destino.

En ambos casos los pasos son similares, aunque hay diferencias importantes. Por ejemplo, el rectángulo de destino para el vídeo se define generalmente en coordenadas de pantalla 2D, en lugar de transformar los vértices 3D. Además, en

el vídeo, el mapa de *bits* se actualiza varias veces por segundo, mientras que la mayoría de los modelos 3D utilizan texturas estáticas.

En los juegos, un uso común para el vídeo es el de crear escenas de corte o vídeo a pantalla completa. Se puede usar una escena de corte como presentación, como unión entre los niveles, o en los créditos. Escenas de corte puede dar a su juego dramatismo, sensación cinematográfica. [5]

1.2.3 Vídeo 3D

El vídeo 3D es un nuevo tipo de recurso media, que captura apariencia tridimensional y la dinámica de escenas del mundo real mediante la combinación de múltiples datos de entrada de vídeo a un modelo coherente. Durante la reproducción, el espectador puede elegir libremente el punto de vista de una cámara virtual en el espacio y el tiempo.

Como una aplicación más allá de la reproducción virtual se presenta un sistema versátil para la edición no lineal de imágenes de vídeo 3D. Combina las ventajas de la edición de vídeo convencional en 2D con el poder más avanzado de la geometría 3D.

Utilizando este tipo de recurso media, el usuario es capaz de generar efectos visuales complejos. Por lo tanto, vídeo 3D ofrece una novedosa herramienta para post-producción de vídeo, la obtención de las posibilidades de efectos especiales que sería difícil o imposible de lograr con las tecnologías convencionales de imágenes bidimensionales. [6]

1.3 Técnicas para la reproducción de vídeos

1.3.1 Textura Animada

Si al espacio de textura, se le añade la dimensión tiempo, se obtiene lo que se conoce como una textura en el tiempo o textura animada. El contenido del espacio de textura puede interpretarse como una serie de imágenes (o incluso de volúmenes) cada una de las cuales se emplea como espacio de textura según el

instante de tiempo en el que nos encontremos. En algunos ordenadores incluso puede capturarse en tiempo real una secuencia de vídeo y aplicarla como textura. [16]

La creación de una textura puede ser más costosa computacionalmente que modificar una ya existente, por lo que si se desea utilizar una textura animada, tal vez texturas de vídeo en vivo, se debe reemplazar toda o parte de una imagen de una textura ya existente.

Esto puede ser útil para ciertas aplicaciones, tales como el uso en tiempo real, captura de vídeo, imágenes como textura. Para estas aplicaciones, tiene sentido crear una textura única y reemplazar los datos repetidamente de la textura con las nuevas imágenes del vídeo. Existen restricciones de tamaño que obligan al ancho o altura de la textura a ser de una potencia de dos, algo a tener en cuenta para procesar imágenes de vídeo, que generalmente no tienen tamaños que son potencia de dos. [17]

1.3.2 Reproducción del formato FLIC

FLI es un formato de animación desarrollada por Autodesk [18], para crear y reproducir animaciones generadas por ordenador a altas resoluciones con Autodesk Animator, mientras que el formato FLC era el formato estándar utilizado en Autodesk Animator Pro. Estos dos formatos (FLI y FLC) son mencionados como el formato FLIC.

Las funciones para leer y reproducir las películas FLIC, son especialmente útiles entre las escenas dentro de un juego o como vídeo en la apertura que frecuentemente es como comienzan los juegos. [13]

1.3.3 Biblioteca de vídeo para Windows "vfw32"

Las API de Windows son bibliotecas de vínculos dinámicos (DLL) que forman parte del sistema operativo Windows. Se utilizan para realizar tareas cuando resulta difícil escribir procedimientos equivalentes.

La ventaja de utilizar las API de Windows en el código es que pueden ahorrar tiempo porque contienen numerosas funciones útiles ya escritas y listas para utilizar. La desventaja es que puede resultar difícil trabajar con ellas y pueden ser implacables cuando las cosas van mal.

Las API de Windows representan una categoría especial de interoperabilidad. No utilizan código administrado, no tienen otras bibliotecas integradas y utilizan tipos de datos especiales.

La biblioteca "vfw32" (*video for windows*) pertenece a las API de Windows, propiedad de la empresa Microsoft. Esta biblioteca provee un grupo de funciones que permiten la reproducción de ficheros de vídeo de extensión (.avi). Sus titulares permiten la utilización gratis de ella mientras no se desensamble. [50]

Para utilizarla es necesario incluir el fichero de cabecera "vfw.h" y enlazar la biblioteca vfw32.lib. Antes de esta operación es importante incluir la cabecera "windows.h" que especifica que se van a utilizar funciones de las APIs de Windows.[47]

```
#include <windows.h>
#include <vfw.h>
#pragma comment( lib, "vfw32.lib" )
```

Es necesario inicializar la biblioteca por medio de la función "*AVIFileInit()"*

Existen varias formas de abrir un fichero de extensión ".avi", la primera abre un único *stream* de datos. Como primer parámetro recibe un puntero al buffer que contiene el manejador del *stream*, luego el nombre del fichero que se pretende abrir; el tercer parámetro es el tipo de *stream* que se extraerá, que puede ser de vídeo (*streamtypevideo*) o de audio (*streamtypeAUDIO*); el cuarto parámetro es el número de *stream* (ya que puede haber múltiples *streams* de vídeo en un fichero ".avi"), siendo en este caso 0, lo que significa que se extraerá el primer *stream*; *OF_READ* es una constante que especifica abrir solo para lectura.[47]

```
PAVIFILE pAviFile;
```

AVIStreamOpenFromFile(&pAviFile, "filename.avi", streamtype, 0, OF_READ, NULL);

En otra de las variantes se abren todos los *stream* contenidos en el vídeo. Al igual que la anterior como primer parámetro recibe un puntero al *buffer* que contiene el manejador del *stream*, luego el nombre del fichero que queremos abrir; el tercer parámetro significa abrir solo para lectura, el último parámetro se especifica como nulo.[48]

AVIFileOpen(&pAviFile, "filename.avi", OF_READ, NULL);

Hay que obtener información referente al *stream* de audio y al de vídeo, esto se logra con la función que se muestra a continuación. El segundo parámetro es la estructura que contendrá la información sobre el fichero ".avi". El tercer parámetro es el tamaño de la estructura.

AVIFILEINFO info;
AVIFileInfo(pAviFile, &info, sizeof(info));

Accediendo a la información contenida en la estructura anterior se puede calcular el ancho en píxeles de un *frame*, restando el borde izquierdo del derecho. Para la altura es equivalente.[47]

Ancho = info.rcFrame.right - info.rcFrame.left;
Alto = info.rcFrame.bottom - info.rcFrame.top;

En el caso del audio se hace necesario conocer además el formato que tienen los datos almacenados en el *stream* de audio. La función recibe como primer parámetro un manipulador que apunta al *stream*, el segundo parámetro es la posición en el *stream* usada para obtener el formato de los datos de audio, tiene valor 0 en este caso porque se lee el audio desde el principio; el tercer parámetro es un puntero a un *buffer* para almacenar el formato de los datos; el cuarto parámetro es un puntero a una ubicación que indica el tamaño del bloque de memoria referenciado por el *buffer*, el valor se cambia para indicar la cantidad de datos leídos. La función devuelve cero si todo ocurrió bien o un error en caso contrario.

PAVISTREAM pavi,
AVIStreamReadFormat (pavi, 0, LpFormat, &lFormat);

El formato de los datos de audio en el *stream* está basado en la estructura WAVEFORMAT, por lo que se puede utilizar la siguiente función; la cual abre el dispositivo de sonido para reproducir el formato especificado. Como primer parámetro recibe un puntero a un *buffer* donde se almacena el manipulador del dispositivo de salida de audio. Este manipulador se utiliza para identificar el dispositivo que se está utilizando; el segundo parámetro es un identificador del dispositivo de salida de audio que será abierto, en este caso se utiliza (WAVE_MAPPER) ya que esta función selecciona el dispositivo capaz de reproducir el formato especificado; el tercer parámetro es un puntero al *buffer* donde se almacenan los datos que serán enviados al dispositivo; el cuarto parámetro se utiliza para procesar mensajes relacionados con el proceso de reproducción, por último recibe una bandera para abrir el dispositivo, en este caso es un parámetro del manipulador de *Windows*. [48]

waveOutOpen(&WaveOut, WAVE_MAPPER, (LPWAVEFORMATEX) LpFormat,0, 0L, CALLBACK_WINDOW);

Luego se utilizan un grupo de funciones para controlar la reproducción del audio. Para detener la reproducción en el dispositivo especificado y reiniciar a cero la posición de reproducción, se utiliza la siguiente función:

waveOutReset(WaveOut);

Para continuar la reproducción partiendo de donde se haya pausado o detenido el audio del vídeo se utiliza:

waveOutRestart(WaveOut);

Si se pretende pausar la reproducción del audio y guardar la posición, para continuar desde ese punto antes de una llamada a continuar, se invoca la función:[50]

waveOutPause(WaveOut);

Es necesario conocer la información del número de *frames* lo cual es posible mediante la siguiente función:

AVIStreamLength(pAviFile);

Para mostrar la animación se calcula cuánto tiempo debe ser mostrado cada *frame*, o sea el *framerate* (*frames* por segundo). La función siguiente, dada una posición en la animación, devuelve cuántos milisegundos lleva llegar a esa posición. Recibe el número del último *frame* (obtenido por la función anterior) y el valor devuelto se divide por el número de *frames*, con lo que se obtiene: milisegundos por *frame*. [14]

Fps = AVIStreamSampleToTime (pAviFile, Lastframe) / Lastframe ;

Generalmente los datos en el fichero de vídeo están comprimidos por lo que hay que preparar el programa para descomprimir los *frames* de vídeo. Como primer parámetro recibe el puntero que apunta al *stream* de vídeo, como segundo parámetro un puntero a una estructura que define el formato de vídeo específico que se requiere, en este caso se utiliza el formato por defecto, aunque se podría utilizar AVIGETFRAMEF_BESTDISPLAYFMT para decodificar los *frames* al mejor formato posible para ser mostrado. Si todo es correcto, un objeto GETFRAME es devuelto. Desafortunadamente lo único que se puede cambiar es el ancho y alto de la imagen devuelta. [47]

AVIStreamGetFrameOpen (pAviFile, NULL);

Se selecciona un *frame* del fichero de vídeo, el cual se convierte a un tamaño y profundidad de color que pueda ser utilizado para su correcta visualización, esto es en dependencia del sistema que se utilice para mostrar las imágenes.

La función siguiente devuelve la dirección del *frame* de vídeo descomprimido. Recibe como primer parámetro un puntero a un objeto GetFrame (se obtiene mediante la función anterior). Como segundo parámetro recibe la posición del *frame* que se desea.

```
AVIStreamGetFrame ( PGETFRAME , Frame);
```

Al cerrar se llaman a las funciones que cierran correctamente el fichero de vídeo y se liberan los recursos. [50]

```
AVIStreamGetFrameClose(m_PIPgf);
AVIStreamRelease(m_PIPavi);
waveOutClose(WaveOut);
AVIFileExit();
```

1.3.4 Biblioteca de vídeo para Linux "avifile"

La biblioteca "avifile" intenta proporcionar utilidades relacionadas con las multimedias para Linux. La idea principal del proyecto estaba en la utilización de bibliotecas dinámicas Win32 en el ambiente Linux y su objetivo primario era la captura de vídeo y la recomprensión. Actualmente la mayor parte del trabajo se realiza para poder reproducir archivos codificados con una amplia gama de códec, con buen funcionamiento y estabilidad. [51]

Facilita abrir y extraer los *streams* del fichero de extensión ".avi" sobre la plataforma Linux. Para utilizarla, lo primero es incluirla, este caso muestra la forma de hacerlo con la versión 0.7.

```
#include <avifile-0.7/avifile.h>
#include <avifile-0.7/version.h>
#include <avifile-0.7/image.h>
#include <avifile-0.7/StreamInfo.h>
```

No es necesario inicializar la biblioteca directamente.

Para abrir un fichero ".avi", se utiliza la siguiente función. Es necesario especificar la dirección donde se encuentra ubicado el fichero de vídeo.

```
IReadFile *m_pkAvifile;
m_pkAvifile = CreateReadFile(arg_FileName);
```

Para acceder al *stream* de vídeo es necesario preguntar primero si se abrió correctamente el fichero, para ello se utiliza:

```
m_pkAvifile->IsOpened();
```

En caso positivo se obtiene el *stream* de vídeo.

La función siguiente recibe como primer parámetro el número de *stream* (ya que puede haber múltiples *streams* de video en un fichero ".avi"), siendo en este caso 0, lo que significa que se requiere el primer *stream*, como segundo parámetro recibe el tipo de *stream* que se requiere seleccionar, en este caso se extrae el *stream* de video, de querer el *stream* de audio sería IReadStream::audio, como parámetro.

```
m_pkAvifile->GetStream(0, IReadStream::video );
```

Es posible obtener información referente al *stream* de video, esto se logra con la función que se muestra a continuación. La función devuelve una estructura que contendrá dicha información.

```
StreamInfo *m_pkStreaminfo;
m_pkStreaminfo = m_pkAvistream->GetStreamInfo();
```

Accediendo a la información contenida en la estructura anterior se puede obtener el ancho y la altura en píxeles de los *frames* que conforman el vídeo.

```
Ancho  =  m_pkStreaminfo->GetvídeoWidth();
Alto  =  m_pkStreaminfo->GetvídeoHeight();
```

Para mostrar la animación se obtiene cuánto tiempo debe ser mostrado cada *frame*, o sea el *framerate* (*frames* por segundo). La función siguiente, devuelve la cantidad de *frames* por segundos.

```
Fps = m_pkStreaminfo->GetFps();
```

Hay que preparar el programa para descomprimir los *frames* de video, para ello es necesario llamar a la función que se muestra a continuación, la cual inicializa el decodificador.

m_pkAvistream->StartStreaming();

Se selecciona un *frame*, el cual se convierte a un tamaño y profundidad de color que pueda ser utilizado para su correcta visualización, esto es en dependencia del sistema que se utilice para mostrar las imágenes.

Para el proceso de decodificación, la biblioteca utiliza un contador interno, al cual no se tiene acceso. La función siguiente sitúa el puntero en el *frame* que se desea extraer.

m_pkAvistream->Seek(0);

La siguiente función lee un *frame* y lo descomprime en un *buffer* interno e incrementa la posición del contador.

m_pkAvistream -> ReadFrame();

La siguiente función devuelve un puntero al *buffer* donde se almacena el *stream* de vídeo descomprimido, específicamente en el *frame* deseado.

m_pkImage = m_pkAvistream->GetFrame();

Se convierte una imagen que provee la biblioteca a un formato especificado como segundo parámetro de la siguiente función:

*CImage *m_pkImage;*
CImage kConverted(m_pkImage, IMG_FMT_BGR24);

Luego, se tiene acceso a los datos del *stream* de vídeo, utilizando:

kConverted.Data()

Para trabajar con el sonido se puede extraer la información referente al audio leyendo el *stream* de audio y luego preguntando por la información de este, así como por el formato, pero la biblioteca avifile permite construir directamente un objeto de reproducción el cual se maneja internamente. La función a continuación recibe como primer parámetro un puntero a la interfaz que manipula los eventos de mouse y teclado; como segundo parámetro recibe una ventana donde se desea mostrar el vídeo, en ambos casos utilizamos NULL porque solo queremos obtener el audio del vídeo; el tercer parámetro es para especificar la ubicación del fichero de vídeo.

```
IAviPlayer* m_pkPlayer;
m_pkPlayer = CreateAviPlayer2 (NULL, NULL, FileName);
```

Para detener la reproducción del audio se utiliza la siguiente función.

```
m_pkPlayer->Stop();
```

La próxima función continúa la reproducción del audio a partir de la posición en la que se encuentre.

```
m_pkPlayer->Start();
```

La función siguiente pausa la reproducción del audio, recibe como parámetro "*true"* o *"false"* (verdadero o falso) dependiendo si se quiere detener o seguir la reproducción.

```
m_pkPlayer->Pause(true);
```

Al cerrar es necesario detener el proceso de descompresión. [51]

```
m_pkAvistream->StopStreaming();
```

1.3.5 Visualización de texturas 2D utilizando OpenGL

Primeramente es necesario inicializar OpenGL. Utilizando la función a continuación activamos el mapeado de texturas. Las texturas en OpenGL pueden ser 1D, 2D o 3D. Las 1D tienen ancho pero no altura; las 2D son imágenes que tienen un ancho y una altura de más de 1 píxel; en el caso de las texturas 3D tienen volumen. [37]

glEnable(GL_TEXTURE_2D)

La textura tiene 2 coordenadas, una para el eje horizontal y la otra para el eje vertical, y toma valores de 0.0 a 1.0, aunque el valor puede ser mayor que 1.0 en cuyo caso se repite la textura en la forma que sea especificada. [36]

Es necesario definir la manera de cómo será aplicada la textura a cada píxel de la superficie que queremos texturizar. Cuando la imagen de la textura no se adecua con el tamaño de la superficie que se quiere cubrir, OpenGL utiliza unos filtros de textura para interpolar entre los píxeles de la imagen y adecuarla a la superficie. Cuando la superficie es menor que la imagen, OpenGL utiliza un filtro para minimizar la imagen y cuando la superficie es mayor que la imagen, utiliza un filtro para maximizarla. OpenGL pone a disposición 6 tipos de filtros. En este caso se ha optado por una interpolación lineal. [37]

La función que se utiliza se muestra a continuación, recibe como primer parámetro el tipo de textura, el segundo parámetro es para especificar si el filtro se utilizará para achicar la imagen o para expandirla y el tercero es el tipo de filtro.

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

OpenGl, requiere que el tamaño de las texturas sea potencia de dos, por lo que se necesita una forma rápida de cambiar el tamaño del vídeo a un formato que se pueda usar como textura. [38]

Para generar la textura se usa la función siguiente, cuyos parámetros son: el objetivo, que es una textura 2D; el nivel de detalle usado; los desplazamientos, a partir de donde OpenGL empezará a copiar (0,0 es la esquina inferior izquierda);

luego está el ancho y alto de la imagen en píxeles que se quiere copiar; GL_RGB es el formato de los datos; finalmente el puntero a los datos.

glTexSubImage2D (GL_TEXTURE_2D, 0, 0, 0, Ancho, Alto, GL_RGB, GL_UNSIGNED_BYTE, data);

Si se desea utilizar texturas que no sean potencia de 2, se puede utilizar la función siguiente, aunque es más costosa en términos de memoria y más lenta que la anterior. Recibe como primer parámetro el tipo de textura, luego el formato interno que puede ser de 1, 2, 3, 4 o constantes definidas por OpenGL, como tercer y cuarto parámetro son las dimensiones de la textura, el siguiente parámetro es el formato de los píxeles, luego se especifica el tipo de dato y por último un puntero a los datos de la imagen en memoria. [36]

gluBuild2DMipmaps(GL_TEXTURE_2D, 3, Ancho, Alto, GL_RGB,GL_UNSIGNED_BYTE, data());

Windows guarda los colores RGB (rojo verde azul) como BGR (azul verde rojo) [42] por lo que es necesario corregir el problema, por el que los colores se muestran intercambiados, rojo con azul, (ver Figura 21, Anexo 13).

Para intercambiar los colores hay que recorrer todos los píxeles de cada imagen e intercambiar los colores. [14] [42] En cada píxel están almacenados los valores de rojo, verde y azul que le da su color final. [38]

Para dibujar, se limpian el buffer de colores y de profundidad.

glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

Se crea un objeto que puede ser un polígono. A partir de que se van creando los vértices del polígono se van leyendo las coordenadas de la textura que ha sido generada. [34][35]

```
glBegin(GL_POLYGON);
        glTexCoord2f(1.0f, 1.0f);  glVertex3f( XLeftBottom,   YLeftBottom,   ZLeftBottom);
        glTexCoord2f(0.0f, 1.0f); glVertex3f( XRightBottom, YRightBottom, ZRightBottom);
```

```
        glTexCoord2f(0.0f, 0.0f); glVertex3f( XRightTop,    YRightTop,    ZRightTop);
        glTexCoord2f(1.0f, 0.0f); glVertex3f( XLeftTop,     YLeftTop,     ZLeftTop);
glEnd();
```

El resultado es un polígono texturizado con la textura generada.

1.4 Formatos de vídeo digital

La reproducción de vídeo es ya un elemento común en los ordenadores modernos, siendo esta una de las tareas que más recursos consume. La sensación de movimiento se consigue con secuencias de imágenes con una velocidad de unos treinta fotogramas por segundo. Las posibilidades se reducen al empleo de sistemas de compresión y a la reducción del tamaño de las ventanas de vídeo y del número de fotogramas por segundo.

Una fotografía sin comprimir de 800x600 píxeles de resolución ocupa aproximadamente 1,3 Mb. Así pues, secuencias de vídeo de este tamaño y con 30 fotogramas por segundo generarían ficheros de vídeo con un tamaño de 390 Mb para 10 segundos o 11,4 Gigabytes para 5 minutos. Por tanto, son imprescindibles sistemas de compresión eficientes para el manejo de vídeo. [7]

1.4.1 Formato AVI

Los archivos AVI son un caso especial de archivos RIFF (*Resource Interchange File Format* o Formato de Archivos para el Intercambio de Recursos) un formato de propósito general para el intercambio de datos multimedia. [8]

Han existido dos versiones de formatos AVI. El primero que tenía algunas limitantes y la segunda versión que eliminó dichas limitantes, aunque ocasionó archivos gigantescos de vídeo. Sólo existen dos tipos generales de AVI: los basados en "*video for windows*" y los basados en "*directshow*".

Para que todos los flujos de datos de vídeo y de audio puedan ser reproducidos simultáneamente es necesario que se almacenen de manera entrelazada. De esta

manera, cada fragmento de archivo tiene suficiente información como para reproducir unos pocos fotogramas junto con el sonido correspondiente.

El formato AVI admite varios flujos de datos de audio, lo que en la práctica significa que puede contener varias bandas sonoras en varios idiomas. Es el reproductor multimedia quien decide cuál de estos flujos debe ser reproducido, según las preferencias del usuario.

Los archivos AVI se dividen en fragmentos bien diferenciados denominados *chunks*. Cada *chunk* tiene asociado un identificador denominado etiqueta *FourCC*. El primer fragmento se denomina cabecera y su papel es describir meta información respecto al archivo, por ejemplo, las dimensiones de la imagen y la velocidad en fotogramas por segundo. El segundo *chunk* contiene los flujos entrelazados de audio y vídeo. Opcionalmente, puede existir un tercer *chunk* que actúa a modo de índice para el resto de *chunks*. [9]

Uno de los problemas más comunes es la falta de un estándar claro. AVI no es un formato claro de almacenamiento de vídeo, es más un formato de propósito general. La especificación del AVI lo único que dice es que si se almacena un vídeo con ese formato, simplemente se le pone una cabecera para identificar el tipo de fichero que es y que se almacena un segmento de video y uno de audio a continuación, de forma intercalada.

El problema viene en que el trozo de imagen y de audio puede ser de cualquier tipo. Es decir, que no se especifica el formato que tiene la imagen y el audio.

Un AVI puede ser creado con un formato de video y otro de sonido. Esto trae como inconveniente que si no se posee el códec adecuado en el sistema, simplemente no funcionará. Si por ejemplo la película está en MPG4-OGG y no se tiene el códec OGG, pues se verá la película pero se oirá. Si la película está en Xvid-MP3, pues se oirá la película pero no se verá. Si la película está en Xvid-OGG ni se verá ni se oirá (siempre que no se tengan los códec Xvid y OGG). [10]

1.4.2 QuickTime

Es un estándar extensible para multimedia digital, desarrollado por Apple, que establece las bases para la representación de objetos y formatos de archivo, manteniendo las técnicas de compresión de imagen y vídeo, las interfaces con los humanos y las interfaces de programación de aplicaciones. Incluye soporte directo para la sincronización del audio y vídeo así como algoritmos de compresión de imágenes. QuickTime trae medios dinámicos para una amplia gama de aplicaciones, incluyendo herramientas de creación de multimedia como editores de vídeo y sistemas de animación. [58]

Cuenta con un reproductor, con un diseño simple y controles fáciles de usar, que ofrece funcionalidades de avanzar por una película o bajar el ritmo de reproducción, así como encontrar un determinado fotograma. Reproduce el formato de vídeo estándar de Apple (MOV), su uso se hace muy aconsejable, ya que aprovecha las mejores tecnologías de codificación de vídeo. [63]

QuickTime no es sólo un reproductor, sino un sistema multimedia completo capaz de reproducir y de transmitir contenidos de alta calidad en internet y otros dispositivos.

Su versión siete es compatible con el estándar MPEG-4, utiliza una avanzada tecnología de compresión de vídeo llamada H.264 para ofrecer imágenes en alta definición usando menos ancho de banda y espacio de almacenamiento. Esta tecnología es conocida como AVC (Advanced Video Coding) o Codificación de Vídeo Avanzada que permite contenidos muy nítidos superiores al estándar de DVD, y otros códec de alta calidad. [59]

En la medida en la que el Software Apple contenga funcionalidad de codificación y/o descodificación AVC, el uso comercial de la tecnología H.264/AVC requerirá la concesión de una licencia adicional y se aplicará la disposición de que se concede a los consumidores una licencia de uso, de esta funcionalidad, únicamente con fines personales y no comerciales. [62]

Se encuentra disponible para los sistemas operativos Windows y Macintosh [60]. Los sistemas GNU/Linux pueden usar QuickTime mediante programas como Mplayer [61].

1.4.3 MPEG

MPEG (Grupo de Expertos en Imágenes en Movimiento) es el nombre de la familia de un estándar, para la representación codificada y comprimida de información visual y auditiva, en medios de almacenamiento digital.

Los ficheros de video MPEG tienen una extensión (.mpg) y los ficheros de audio generalmente tienen una extensión (.mp2) o (.mp3)

La mayor ventaja comparada con otros formatos de codificación de vídeo y audio es que los ficheros son mucho más pequeños manteniendo una misma calidad. Esto es producto a que se utilizan sofisticadas técnicas de compresión. [64]

El Grupo de Expertos en Imágenes en Movimiento está encargado de la elaboración de normas para la representación codificada de audio, video digital y datos relacionados. Desde 1988, cuando se ha establecido, el grupo ha elaborado normas que ayudan a los usuarios a ofrecer una experiencia digital cada vez más agradable de los medios. El número de normas independiente es más de 125, sin contar las modificaciones. [65]

Algunos de estos estándares lo constituyen:

MPEG-1
Es la norma en la que se basa los productos como Video CD y MP3. Guarda una imagen, la compara con la siguiente y almacena sólo las diferencias. Se alcanzan así grados de compresión muy elevados. Consigue el mayor grado de compresión a costa de un mayor tiempo de cálculo. [65]

MPEG-2
Es la norma en la que se basan los productos como televisión digital y DVD. Con una calidad superior al MPEG-1, fue universalmente aceptado para transmitir video

digital comprimido con velocidades mayores de 1Mb/s aproximadamente. Pueden conseguirse elevados ratios de hasta 100:1, dependiendo de las características del propio video.[65]

MPEG4

Es el estándar para multimedia para la web fija y móvil. Es un estándar orientado inicialmente a las videoconferencias y para Internet. El objetivo es crear un contexto audiovisual en el cual existen unas primitivas llamadas AVO (objetos audiovisuales). Se definen métodos para codificar estas primitivas que podrían clasificarse en texto y gráficos.

Ha sido especialmente diseñado para distribuir videos con elevados ratios de compresión, sobre redes con bajo ancho de banda manteniendo una excelente calidad para usuarios con buen ancho de banda. Es rápido codificando el video de alta calidad para contenidos en tiempo real y bajo demanda. [65]

1.4.4 Formato Avanzado de Secuencias

Advanced Systems Format (ASF) es un formato de archivo extensible, diseñado para almacenar datos digitales de audio y video de forma coordinada. Es compatible con la entrega de datos sobre una amplia variedad de redes y también es adecuado para su reproducción local.

ASF fue diseñado con los siguientes objetivos:

•Soportar la reproducción eficiente desde los servidores de medios digitales, servidores HTTP, y dispositivos de almacenamiento local.
• Para soportar escalables archivos digitales de audio y video.
•Para permitir que una única composición, en el archivo digital, esté presente sobre una amplia gama de anchos de banda.
•Para permitir el control de autoría sobre las relaciones de transmisión de medios digitales, especialmente en situaciones con restricciones de ancho de banda.
•Para ser independiente de cualquier sistema de composición digital, los medios de comunicación, sistemas operativos, o el protocolo de comunicaciones de datos.

Cada archivo de ASF se compone de uno o más *streams*. El encabezado del archivo especifica las propiedades de la totalidad del fichero, junto con las propiedades específicas de los *streams*. Los datos, almacenados después de la cabecera del archivo, hacen referencia a un determinado número de *stream* para indicar su tipo y la finalidad. La entrega y presentación, de todos los datos de flujo, están alineados con una línea de tiempo común.

La definición del archivo ASF incluye la especificación de los tipos de recursos media utilizados. Si una implementación soporta recursos media compatibles con los tipos de recursos media estándar especificados en [68], entonces esos tipos de recursos media son soportados de la manera descrita en la especificación del fichero ASF.

ASF soporta la transmisión de "contenido en vivo" en una red. Contenido en vivo se refiere al contenido de medios digitales, que se transmite directamente y que puede o no ser grabado en un medio persistente (por ejemplo, un disco, CD, DVD, etc). Este uso de manera explícita y únicamente significa que la información que describe el contenido de los ficheros media tiene que ser recibida antes que los datos a reproducir de los ficheros digitales (para poder ser interpretados los datos de los ficheros media), y que esta información debe contener la semántica de los objetos de encabezado ASF. Del mismo modo, los datos recibidos deben ajustarse al formato de los paquetes de datos ASF. Para su transmisión no es necesario el envío de información sobre los protocolos de control de red o protocolos de red de transmisión, únicamente hay que respetar el orden de llegada de la información (semántica de cabecera antes de los datos) y el formato de los datos.

Una descarga parcial de archivos ASF puede ser enteramente funcional. Mientras estén disponibles los objetos de cabeceras necesarios y un conjunto completo de los objetos de datos, es posible buscar en cualquier posición (hacia atrás y hacia adelante) en el archivo descargado parcialmente. Para buscar en un archivo ASF no se requiere el uso de un índice, sin embargo, muchas implementaciones utilizan un índice para tener un acceso eficiente a los datos.

ASF es una presentación de un formato de archivo de media digital. Es compatible con el contenido digital en vivo y bajo demanda. Los archivos ASF pueden ser

editados, pero ASF está diseñado específicamente para la transmisión y la reproducción local. [68]

Este formato está patentado por Microsoft quien no permite que se desarrollen herramientas para su decodificación, así como herramientas que conviertan de este formato a otros. Avery Lee autor de VirtualDub, herramienta para la captura y procesamiento de vídeo, se vio forzado a remover la funcionalidad de conversión del formato ASF a formato AVI, por temor a una demanda legal por parte del grupo Windows Media. [69]

1.4.5 Windows Media Video

El formato .wmv (Windows Media Video) es una extensión que no tiene diferencia con los archivos .asf. Estos usan el formato de fichero estándar de Windows Media.

Los ficheros .wma (Windows Media Audio) y .wmv se introdujeron como un convenio para posibilitar a los usuarios la fácil diferenciación entre los ficheros de audio .wma y los de video .wmv. Sin embargo, no hay ninguna limitación en el formato, y las extensiones pueden usarse intercambiablemente.

Algunas herramientas y servicios requieren la extensión de .asf para aceptar el contenido. Se puede simplemente renombrar cualquier archivo .wma o .wmv para usar la extensión de .asf y usarlos con esas herramientas.

1.4.6 Real Media

El formato Real Media (.rm) es contenedor de los formatos Real Audio y Real Video. El formato Real Video es propietario, desarrollado por RealNetworks. Fue liberado en 1997 y a partir del 2006 se encuentra en la versión 10. Real Video es compatible con muchas plataformas, incluyendo Windows, Mac, Linux, Solaris y varios teléfonos móviles.

Real Media es adecuado para su uso como formato *streaming*, es uno de los formatos que se tiene en cuenta para enviar video a través de la red. Se puede

utilizar para ver televisión en directo, ya que no requiere descargar el video de antemano. [66]

Cuenta con un reproductor que permite descargar música y videos desde la web, puede grabar CD de música o transferir la música y videos a dispositivos móviles. Se distribuye de forma gratuita pero cuenta con una versión añadida con licencia comercial que permite crear los CD ajustar el sonido con ecualizadores avanzados y transferir videos de alta calidad a DVD y a teléfonos móviles. [67]

1.5 Características de la SceneToolKit

El desarrollo de la herramienta persigue desarrollar un motor de juego (*game engine*) para sistemas de realidad virtual con las siguientes características:

- Independencia entre el gráfico y el resto de los módulos, para poder ser utilizado sin necesidad de usar todos los módulos que componen la herramienta.

- Arquitectura modularizada de forma tal que se puedan usar las partes deseadas, por ejemplo, usar solamente el módulo de sonido, o el de red, sin tener que usar necesariamente el módulo gráfico de la SceneToolKit.

- Núcleo gráfico minimizado, y el resto de las prestaciones se consideran módulos extras que se brindan en bibliotecas independientes para no obligar su utilización.

- Ser multiplataforma, de manera que puedan montarse sobre varias arquitecturas de *hardware* (incluyendo o no tarjetas gráficas), sobre varios sistemas operativos (inicialmente GNU/ Linux y Windows).

- Brindar interfaces para el uso de funcionalidades, y a la vez dar acceso al uso de las funcionalidades a bajo nivel.

- Emplear técnicas de la optimización de la visualización de las escenas.

- Ser extensible a actualizaciones futuras.

- Estar orientada a aplicaciones gráficas tridimensionales genéricas, esencialmente simuladores y juegos 3D, por lo que su versión principal no considera aplicaciones 2D o aplicaciones de propósito muy específico.

Se utiliza, como patrón arquitectónico, uno de los patrones de uso recurrente en los videojuegos, el Modelo-Vista-Controlador [70] (MVC, Model-View-Controller), el cual pretende separar la lógica de la aplicación de su representación ante el usuario y la comunicación al modelo, como se representa en la Figura 1.

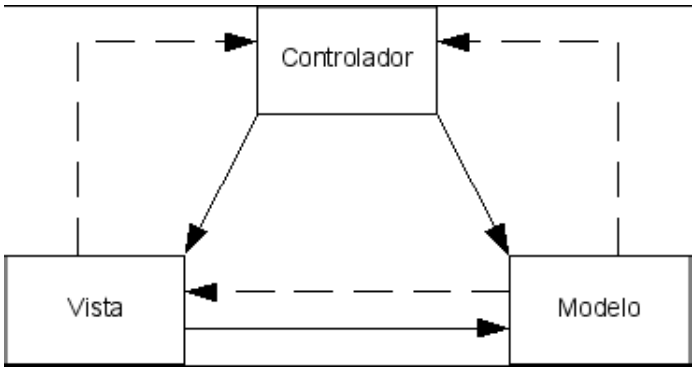

Figura 1. Patrón de arquitectura Modelo-Vista-Controlador (MVC).

Para este patrón la Vista es la representación de todos los elementos de interacción con los usuarios, como los gráficos, sonidos, eventos de teclado, joystick u otros dispositivos de entrada/salida (E/S), de manera que se pueda cambiar de vista, ejemplo usar OpenGL o Direct3D, y el resto del sistema no sufra cambios.

El Modelo encierra los datos propios de las entidades como la posición, la velocidad, la orientación y la lógica básica alrededor de éstos. La información que se almacena es la mínima necesaria, tal como una máquina de estados.

El Controlador por su parte, maneja las entradas que modifican al modelo, abstrayendo su procesamiento para finalmente aplicar los cambios en el modelo. Esto permite hacer cambios en el control sin afectar los elementos que conforman la Vista y el Modelo.

Para garantizar que la STK sea multiplataforma, se trabaja con:

- Lenguaje de desarrollo: C++.
- IDE (Integrated Development Environment, Entorno Integrado de Desarrollo):
  . Sobre GNU/Linux: CodeBlocks 8.02 (lic. GNU GPL versión 3).
  . Sobre Windows: VisualStudio 7.0 (2003) (propietario).

- Sistemas Operativos de desarrollo:
  . GNU/Linux Debian 4.0 “Etch”.
  . Windows XP.

Además se usan las siguientes interfaces de programación de aplicaciones (API, Application Programming Interface):

- API Gráfica para GNU/Linux:
  . OpenGL-FreeGLUT.
- API Gráfica para Windows:
  . OpenGL-GLUT.
  . OpenGL-WGL.
  . Direct3D (de DirectX SDK dx9c).

- API Manejo de datos: STL (C++).
- API Física: ODE 0.9 (Open Dynamics Engine, lic. BSD y LGPL, multiplataforma).
- API Sonido: OpenAL 1.0.
- API Red: HawkNL.
- API IA y scripting: LUA.

La arquitectura de la Herramienta queda basada en los conceptos de “motores gráficos” y “motores de videojuego", separándose claramente las funcionalidades gráficas del resto, como se puede observar en el esquema de la Figura 2, donde se muestra la representación a alto nivel de los módulos que componen a la Herramienta y las relaciones entre éstos.

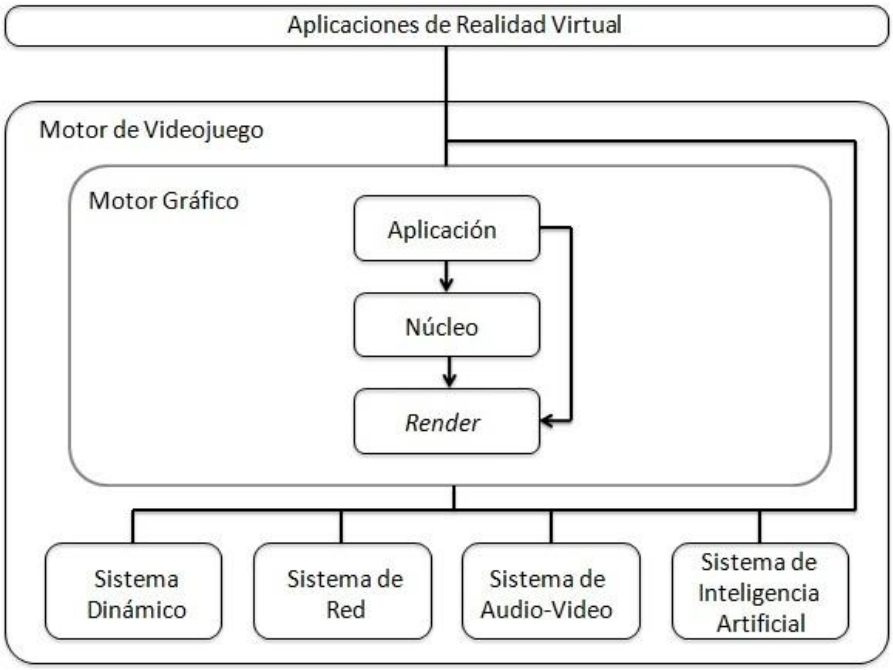

Figura 2: Representación de la arquitectura modular de la Herramienta.

Se puede observar tres niveles de abstracción, uno más bajo que representa el motor gráfico, sobre éste el motor de videojuego y finalmente en un nivel más alto, las aplicaciones de realidad virtual que harán uso de este sistema.

En el motor gráfico se tienen tres capas definidas: la capa Núcleo, que contiene todo el proceso de simulación: manejo de las estructuras de datos, actualización de objetos como geometrías, cámaras, luces, etc.; la capa Render, que regula el proceso de dibujado; y la capa Aplicación, que actúa como interfaz con el programador de aplicaciones finales, dándole acceso a las funcionalidades.

Incluye además un sistema de comunicación con periféricos (Input/Output System). Brinda las distintas implementaciones de la interfaz de dibujado de la STK, según la API gráfica definida (WGL, GLUT o Direct3D). También contiene las distintas implementaciones de la interfaz de aplicación de la Herramienta según el sistema operativo y la manera en que se manejan los eventos.

En el motor gráfico se brindan además un grupo de módulos extras, que no necesariamente tienen que formar parte del núcleo gráfico, por lo que su uso es opcional por parte de los usuarios. En este paquete se ubican módulos para

oclusión estática, optimización por niveles de detalles, un sistema de interfaz de usuarios (UI), sistemas de partículas, efectos visuales, estructuras de datos específicas para la visualización eficiente, una solución para la manipulación de objetos dinámicos, y módulos para la simulación de horarios y estaciones.

El motor de videojuego por su parte, además del motor gráfico, contiene los diversos sistemas que de forma general constituyen interfaces con API externas a la STK, siendo éstos:

Sistema Dinámico: maneja toda la física de la escena y los objetos, las animaciones de autos, personajes, entre otros. Actúa como interfaz con la biblioteca ODE.

Sistema Audio-Video: Con soporte sólo para sonido, maneja los sonidos de la escena. Actúa como interfaz con la biblioteca de sonido OpenAL.

Sistema de Red: maneja las conexiones y transmisión de datos por red. Actúa como interfaz con la biblioteca HawkNL.

Sistema de Inteligencia Artificial: aunque está concebido que la lógica de los juegos la implemente el programador de la aplicación final, este sistema reúne funcionalidades básicas de inteligencia artificial y brinda las facilidades para la programación con scripting. Actúa como interfaz con LUA.

Algunos patrones usados (y en propuesta de usar) en la STK, son los siguientes:

Patrón *singleton*: se utiliza en la clase *CApplitacion* ubicada en el motor gráfico, la cual contiene funcionalidades para la creación de la ventana de la aplicación, así como el control de eventos de periféricos. Para hacer una aplicación final, se debe heredar de *CApplication*, y con el uso de este patrón se garantiza una única instancia del juego, así como que el manejo de eventos de entrada/salida sea único.

Patrón *abstract factory*: se emplea en el módulo de carga de ficheros, donde existe un creador de grafo con funcionalidades para crear diferentes tipos de nodos.

Patrón *composite*: se utiliza en algunas estructuras jerárquicas como el grafo de escena y los componentes visuales.

Patrón *bridge*: se utiliza en la implementación del *render*, donde se define una interfaz que se redefine para cada API gráfica (OpenGL o Direct3D).

Patrón *facade*: se propone su uso esencialmente para interacción con los diferentes componentes del motor de videojuego (física, sonido, red, IA…), con el objetivo de propiciar en los módulos, interfaces para diferentes implementaciones, por ejemplo, el módulo físico con versiones sobre varias bibliotecas existentes con similares prestaciones.

Patrón *strategy*: se utiliza para darle comportamientos a los objetos de la escena representados a través de los nodos: cada nodo brinda la posibilidad de atachársele un controlador, y según el tipo de controlador será el comportamiento de dicho objeto.

Patrón *command*: se utiliza en el módulo de interfaz de usuarios (UI), donde al crear cualquier componente visual se le puede indicar como parámetro las funciones a ejecutar, ejemplo, en el *OnkeyDown* ejecutar un método *MyOnKeyDownExecFunction*. Se propone además usar este patrón para hacer una cola de acciones en el ciclo principal de la escena, ya que existen operaciones que se ejecutan cíclicamente de forma secuencial, incluso en varios hilos diferentes de ejecución, por lo que se pueden tener varios intérpretes (uno por cada hilo), y para cada uno de ellos tener una secuencia de acciones en forma de comandos, permitiendo al usuario final redefinir el orden de ejecución de estas acciones e incluso insertar comandos propios.

Patrón *template method*: se utiliza esencialmente en la clase CApplication central, la cual posee métodos virtuales que regulan el proceso de creación, inicialización, ciclo de escena y terminación del juego, los cuales pueden ser redefinidos por el usuario final; igualmente se pueden redefinir métodos que se invocan ante eventos (*OnResize, OnKeyDown, OnDraw*. . .).

1.6 Motores gráficos

1.6.1. Allegro

Allegro es una biblioteca multiplataforma, principalmente dirigida al desarrollo de videojuegos y programación multimedia. Maneja tareas de bajo nivel, como la creación de ventanas, acepta la entrada de datos por parte de los usuarios, la carga de datos, dibujado de imágenes, reproducción de sonidos y en general para lograr estas funcionalidades se abstrae de la plataforma. Sin embargo, Allegro no es un motor de juego permite diseñar y estructurar el programa como se desee.

Allegro en italiano significa rápido, vivo, brillante. Proviene del acrónimo (Allegro Low LEvel Game ROutines) que significa rutinas de bajo nivel para juego. Fue fundado a mediados de los años 90, pero desde entonces ha recibido contribuciones de cientos de personas en la red. [12]

Está escrito en C y C++ y está diseñado para ser utilizado con estos lenguajes. Es posible usarlo desde otros lenguajes de programación ejemplo Python , Pascal, Perl, C#, entre otros.

Su más reciente versión, la número 5, está diseñada para aprovechar las ventajas del *hardware* moderno (aceleración usando tarjetas gráficas). Esta versión soporta las plataformas Unix/Linux, Windows, MacOS X y iPhone. Solo soporta nativamente primitivas gráficas 2D, pero permite la integración de OpenGL, Direct3D y otras bibliotecas de alto nivel. Está diseñada para ser modular, permitiendo la sustitución de las bibliotecas que la integran.

Se encuentra disponible de forma gratuita y sin garantías de buen funcionamiento, por lo que sus autores no se hacen responsables en caso de daños que provoque su utilización. Los productos que se realicen utilizándolo se pueden comercializar o distribuir libremente. Los agradecimientos, por utilizar este software, no son obligatorios. Solo se establece que el software original, así como modificaciones a este no se puede presentar como propiedad de otras personas. [12]

Con Allegro se puede utilizar una animación para cualquier propósito dentro de un juego, utilizando el soporte integrado para la carga y reproducción del formato FLI (tanto de la memoria y como de un archivo de disco). La única limitación es que sólo se puede reproducir un fichero FLI a la vez. Si se necesitan varias animaciones para ejecutar al mismo tiempo, hay que convertir el archivo FLI a una o más imágenes de mapas de bits y realizar el tratamiento de la película como secuencia de imágenes animadas. Primero sería necesario convertir el fichero FLI a las imágenes de mapa de bits individuales. [13]

Reproducción de un fichero FLIC en Allegro

Para reproducir una animación FLIC utilizando Allegro es necesario abrir el fichero de extensión (.flc) mediante la función “open_fli” que recibe como parámetro el nombre del fichero.

Luego es necesario comenzar su reproducción mediante la función “play_fli” que recibe como primer parámetro el nombre del fichero de animación, el segundo parámetro el identificador de la pantalla donde va a ser mostrado, luego un valor booleano que indica si se va a reproducir de forma cíclica, y el cuarto parámetro se establece como nulo.

Luego se realiza un ciclo en función del tiempo y se va iterando por cada uno de los *frames* que componen la animación. En el momento que se detecta que la animación ha concluido se rompe el ciclo y se prosigue a cerrar la animación mediante la función “close_fli”. [71]

El seudocódigo a continuación muestra la forma básica para utilizar esta funcionalidad.

```
void PlayIntro()
{
        int FliFrames = 0;
        int finishedflag = 0;
        open_fli("introex.flc");
        play_fli("introex.flc", screen, 1, NULL);

while (!finishedflag)
{
                if (fli_timer)
```

```
                {
                        next_fli_frame(1);
                        if (fli_bitmap)
                        {
                                set_palette(fli_palette);
                 blit(fli_bitmap, screen, 0, 0, 0, 30, fli_bitmap->w, fli_bitmap->h);
                        }
                else    {
                        finishedflag = 1;
                        }
                }
}
        close_fli();
}
```

Reproducción de un fichero de vídeo en Allegro

Oscar Giner, miembro de la comunidad de Allegro, a partir de la necesidad publicaba en varias ocasiones en los foros de este motor, de poder reproducir archivos de vídeo, escribió una simple clase en lenguaje de programación C++ para este propósito. Esta nueva funcionalidad la encapsuló en forma de *plugin* nombrándolo AllegAVI y permite reproducir una película en el centro de la pantalla y reproducir el audio utilizando las rutinas de salida de sonido que Allegro ya tenía integradas. El código fuente disponible muestra que para su confección se basó en las API de Windows, específicamente en la biblioteca vfw32.lib. [72]

AllegAVI se encuentra disponible de forma gratuita, sus autores no garantizan su correcto funcionamiento, ni asumen responsabilidad por daños que pueda provocar su utilización. Se concede permiso para usar, copiar, modificar y distribuir el código fuente, para cualquier propósito, solo debe tenerse presente que el origen del código debe ser respetado, las versiones alternas deben ser claramente marcadas como tal y no ser malinterpretadas como la fuente original y estas reglas no pueden ser alteradas o suprimidas en las versiones alternas. El reconocimiento a su autor no es obligatorio.

Para la utilización de este *plugin* se incluye, en el programa principal, la clase “allegavi.h”, luego se crea un objeto del tipo de dato AllegAVI a través del cual se tiene acceso a los métodos que permiten abrir el fichero de vídeo y comenzar su reproducción. [73]

```
#include <allegro.h>
#include "allegavi.h"

int main(int argc, char* argv[])
{
	AllegAVI avi;

	allegro_init();

	avi.Open(argv[i]);

	avi.Play(screen, true, wait_vsync);

	avi.Close();
}
```

1.6.2. Nehe SDK

Nehe SDK, combina un número de tutoriales dentro de un *framework* para la construcción de juegos. Su principal autor refiere que estas lecciones pueden contener errores y que no se deben considerar como el mejor recurso para aprender OpenGL, con estos tutoriales lo que se pretende es mejorar el proceso de aprendizaje para las personas nuevas en el trabajo con OpenGL.

Este *framework* brinda soporte para la representación de vídeos digitales y la explicación de esta funcionalidad se brinda a través del tutorial número 35. [14]

El código de los tutoriales, así como el de la SDK ha sufrido varios cambios desde su nacimiento, en aras de ir perfeccionándolo, pero que este código no es tomado de otros sitios web. En los casos en los que se ha recibido colaboración para el desarrollo de alguna funcionalidad se les han dado los créditos a sus autores. [15]

Para la reproducción de vídeo se utiliza la biblioteca “vfw32” (*video for windows*) perteneciente a las APIs de Windows y que limita el código a ser funcional solo para esta plataforma. La solución que propone no incluye la reproducción del sonido y la visualización de la imagen se realiza utilizando OpenGL.

La técnica que utiliza, consiste en convertir la imagen leída del vídeo en una textura y mapear con ella los objetos. Para ello utiliza un grupo de cabeceras definidas para la creación de la ventana y el control general de la aplicación.

Primeramente se inicializa OpenGL y se habilita el trabajo con texturas. Luego de aplicar los filtros necesarios se invoca la función que abre el fichero de vídeo, la cual recibe la dirección donde se encuentra almacenado. Los datos del vídeo son extraídos a través de la biblioteca "vfw32" (*video for windows*) y son almacenados en un *buffer*.

En la función *Draw* (pintar) se crean los objetos que van a ser mapeados con la textura que se crea a partir de los *frames* del vídeo. Esta función se invoca cíclicamente y antes de pintar los objetos se invoca la función "GrabAVIFrame(…)" que recibe como parámetro un valor entero y que corresponde al número de fotograma dentro del vídeo, es en este método donde se extrae del *buffer,* que almacena la información leída del vídeo, la correspondiente al *frame* especificado por parámetro. También se invoca la función "flipIt(…)" encargada del intercambio de colores RGB por BGR y luego se genera la textura de OpenGL.

Mediante la función "Update(…)" se controla el tiempo de reproducción en milisegundos para que el tiempo entre fotograma y fotograma sea en correspondencia con la información de *frames* por segundos leída del fichero de vídeo. Esta función también es llamada cíclicamente, en ella se realiza además el control de teclas presionadas.

```
#include <vfw.h>
#include "NeHeGL.h"

bool Initialize ()
{
  OpenAVI("fichero.avi");
}
void Draw()
{
  GrabAVIFrame(frame);
  CrearObjetos();
}

int WINAPI WinMain (HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow)
```

```
{
        while(true)
        {
                GetTickCount();
                Update ();
                Draw ();
        }
}
```

1.6.3 Ogre 3D

Ogre es un motor gráfico de código abierto y gratis. Su nombre proviene de las iniciales (*Object-Oriented Graphics Rendering Engine*), es un motor flexible para la visualización gráfica orientado a escena y escrito en lenguaje C++. Está diseñado para hacer de una forma fácil y más intuitiva la producción de aplicaciones gráficas 3D, utilizando aceleración de *hardware*. Su mantenimiento y desarrollo se lleva a cabo por un pequeño grupo y existe una amplia comunidad en la que se realizan las contribuciones.

Ogre no es un motor de juegos, aunque se puede utilizar para hacer juegos, pero está diseñado para proveer solamente la visualización gráfica. Para el sonido, el trabajo con redes, inteligencia artificial, física y otras funcionalidades muy utilizadas en el desarrollo de videos-juegos, es necesario integrarlo con otras bibliotecas.

No admite de forma nativa la representación de vídeos en las definiciones de unidad de textura, pero tiene un mecanismo más genérico que permite a las unidades de textura obtener datos de textura de una fuente exterior como un vídeo.[11]

Extrayendo la información del vídeo y escribiendo texturas en tiempo real, es el mecanismo que se pude utilizar para la representación de vídeos. Entre los aportes que brinda la comunidad de Ogre se encuentra un código para este propósito basado en Direct Show [31], un *framework* multimedia producido por Microsoft. [33]

Para su utilización se incluye primeramente la clase “DirectShowMovieTexture” de la cual se crea una instancia que permite tener acceso a las funcionalidades que brinda. En el constructor del objeto es necesario especificarle por parámetro las dimensiones, alto y ancho, con las que se va a reproducir el vídeo. Luego a través de la función loadMovie(…) se especifica la dirección en la que se encuentra el fichero de vídeo. La clase permite pausar la reproducción y poder detenerla cuando se requiera.

```
#include "UtilsOgreDshow.h"

OgreUtils::DirectShowMovieTexture* dshowMovieTextureSystem =
new OgreUtils::DirectShowMovieTexture(640, 480);

Ogre::String movieName="C:/videofiles/video.avi";
dshowMovieTextureSystem->loadMovie( movieName );

dshowMovieTextureSystem->playMovie();
dshowMovieTextureSystem->pauseMovie();
dshowMovieTextureSystem->stopMovie();
```

Para lograr la visualización es necesario llamar constantemente la función “updateMovieTexture()”, donde se actualiza la información de cada *frame* del vídeo, utilizando un objeto de tipo “DirectShowData” que provee las funciones para abrir el fichero de vídeo pasándole como parámetro la dirección del fichero y controlar su reproducción.

```
dsdata->pGraph->RenderFile( “nombre_fichero.avi”, NULL);
dsdata->pControl->Run();
dsdata->pControl->Pause();
dsdata->pControl->Stop();
```

Para el trabajo con texturas Ogre define una estructura de materiales, que se pueden crear dinámicamente o leer desde un recurso externo. En este caso las texturas están contenidas todas en un fichero y a medidas que se van extrayendo es necesario crear el material, por lo que se realiza de forma dinámica. Después de crear un material es necesario crear una unidad de textura y se le será asignado el

nuevo material. Un material puede estar compuesto por varias técnicas y estas por varias pasadas de *render* que a su vez tienen varios estados de textura, en este caso se utiliza sólo uno de cada tipo por eso siempre se invocan en la posición (0). A través del método "setTextureName(…)" se cambia la textura por lo que el método recibe como parámetro el nombre de la nueva textura creada. [31]

```
Ogre::MaterialPtr mat;
Ogre::TextureUnitState* tex;

  Ogre::String materialName = "MyMaterial";
  mat = Ogre::MaterialManager::getSingleton().getByName(materialName);
  tex = mat->getTechnique(0)->getPass(0)->getTextureUnitState(0);
  tex->setTextureName( dshowMovieTextureSystem->getMovieTexture()->getName() );
```

CAPÍTULO 2: SOLUCIONES TÉCNICAS

En el presente capítulo se mencionarán las herramientas que se utilizaron, y se describirá el proceso de desarrollo y los problemas que se enfrentaron.

2.1 Descripción general

Se propone el modelo de dominio correspondiente a la Figura 3. En el que se representan los conceptos fundamentales asociados al trabajo con vídeos digitales. Se representa el reproductor (que reproduce el vídeo) y este tiene un formato determinado y ocupa una posición. Además está compuesto por dos tipos de información: audio e imagen, que pueden estar codificados indistintamente.

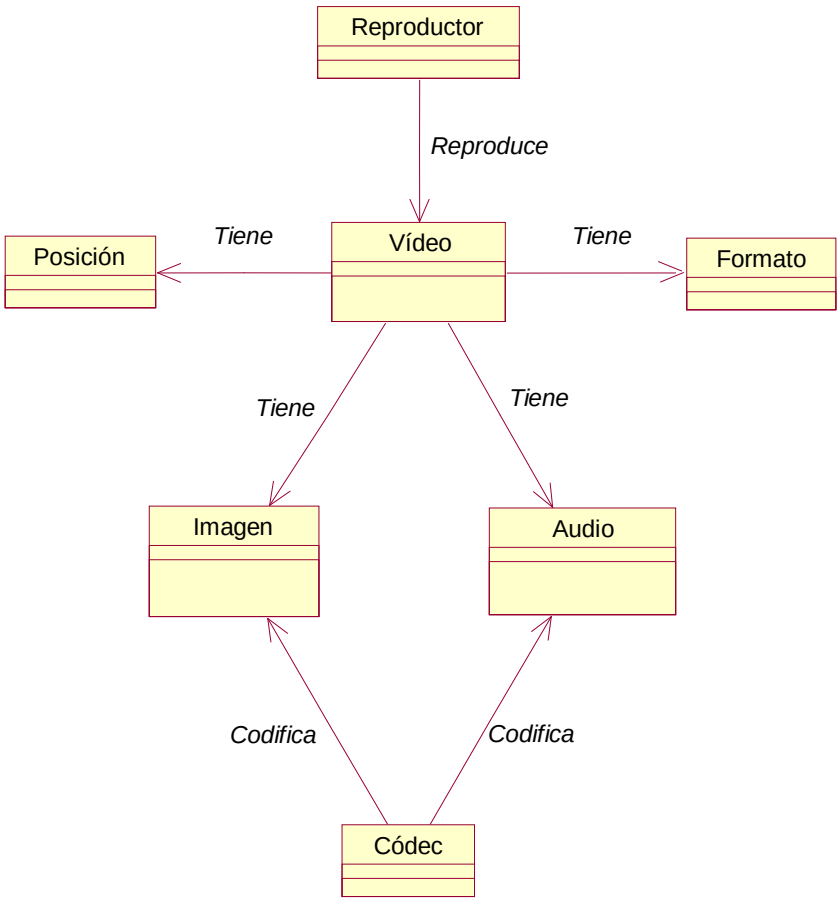

Figura 3: Modelo del Dominio.

Durante la búsqueda se constató que las empresas que utilizan vídeos en las aplicaciones que producen, en su mayoría son privativas y no publican las experiencias en el desarrollo de estas funcionalidades. Producto de esto se decidió realizar una búsqueda por los motores gráficos que se comercializan de forma gratuita y de código abierto, en busca de mecanismos que permitiesen incorporarle a la herramienta estas funcionalidades para el trabajo con vídeos digitales.

Inicialmente se analizó la forma en la que se basaba Allegro para hacer las presentaciones. Se detectó que permitía utilizar una animación para cualquier propósito dentro de un juego, utilizando el soporte para la carga y reproducción del formato FLI. Esta opción no era factible para ser incluida en el desarrollo de la STK ya que el software que se utiliza para su creación, Autodesk Animator, es privativo y no cumplía con las políticas de desarrollo de la herramienta.

En la documentación sobre Allegro se planteaba la limitante de que sólo se podía reproducir un fichero FLI a la vez y si se necesitan varias animaciones para ejecutar al mismo tiempo, había que convertir el archivo a una o más imágenes de mapas de *bits* y realizar el tratamiento de la película como secuencia de imágenes animadas.

Se analizó el motor gráfico Ogre en busca de los mecanismos para la reproducción de vídeos digitales. Se analizó el *plugin* Dshow[31], basado en DirectShow[32], este *plugin* no es compatible con la herramienta STK ya que este motor gráfico maneja las texturas a través de una estructura de materiales que responde a su arquitectura y no cumple con ser multiplataforma. Pero está basado en la generación de texturas a partir de la información almacenada en un fichero de vídeo.

También se analizó el *framework*, NeHe SDK. El código fuente para la reproducción de vídeos que brinda es funcional, está basado en la generación de texturas a partir de la información almacenada en un fichero de vídeo, pero no incluye el sonido del vídeo, además estaba destinado para productos desarrollados en plataforma Windows.

En todos los casos analizados, era común la utilización de texturas a partir de la información almacenada en un fichero, seleccionándose esta técnica para ser desarrollada e incluida en la STK.

La técnica de utilizar texturas a partir de la información almacenada en un fichero de vídeo, plantea añadir la dimensión tiempo al espacio de textura. Si un vídeo está compuesto por una secuencia de imágenes denominadas fotogramas, y cada una de ellas aparece un determinado tiempo, crean en el espectador la sensación de continuidad, y se genera la visión global de una única escena en movimiento. Esta técnica era factible para desarrollar e incluir en la STK ya que se contaba con el soporte para generar texturas y mapearlas sobre los polígonos.

Se escogió el formato AVI como el medio de almacenamiento de la información origen de las texturas que se generarían, y del sonido que se reproduciría, ya que era el formato con el que se trabajaba por parte del grupo de edición y producción de estos. Además su utilización es libre y gratis, y es soportado por gran número de arquitecturas de *hardware*.

El módulo implementado se rige por las mismas políticas definidas para el estándar de código de la herramienta SceneToolKit. Se programó en inglés, utilizando como interfaz de desarrollo CodeBlock[49] y C++ como lenguaje de programación. Estas tecnologías son multiplataforma pero el trabajo con los ficheros de vídeos se realiza dependiendo del sistema operativo (Windows o Linux), ya que para el caso de Windows se utiliza la biblioteca “vfw32” (*Video for Windows*) y en el caso de Linux “avifile” en la versión 0.7.

Para el desarrollo de las funcionalidades, de reproducción de vídeos digitales, a ser integradas en la STK, además de la documentación, pseudocódigos y códigos fuentes que utilizaban los motores gráficos analizados, se consultaron las técnicas para el trabajo con texturas, publicadas en [34], [35], [36], [37] y [38], así como las técnicas para la reproducción del sonido en [39], [40] y [41].

Se decidió, para la reproducción del sonido, hacerlo mediante las bibliotecas que se utilizaron para la lectura de los datos almacenados en los ficheros de vídeo, ya que

estas proveían las funcionalidades necesarias para esta tarea. Se realiza una reproducción 2D del sonido.

2.2 Adaptaciones técnicas

Partiendo de la bibliografía analizada, se implementó una técnica similar basada en OpenGL. Para generar las texturas, correspondientes a cada *frame* del vídeo, se utilizan las funciones básicas que provee OpenGL. Inicialmente se genera un identificador para la textura, OpenGL busca cuantas texturas tiene ya almacenadas y genera un nuevo valor para el identificador. Los vídeos están compuestos por más de un *frame*, a pesar de esto se genera un único identificador, ya que de un determinado vídeo, se va a representar un solo *frame,* en un instante de tiempo, por lo que se va generando una única textura pero con diferentes datos, en correspondencia a la información de cada uno de los *frames*.

Después se asigna el valor del identificador, a una textura de destino, para garantizar que todas las propiedades que modifiquemos, serán modificaciones de esa textura solamente y no de las demás que se pudiesen tener en la aplicación.

Luego se genera la textura con OpenGL, especificándole el tipo de textura, el número de componentes, las dimensiones (ancho y altura) de la imagen, el formato en que está almacenada la textura en memoria, y la información de los píxeles de las imágenes que corresponden a cada *frame*.

OpenGL no provee funciones que permiten cargar ficheros de imágenes en memoria, por lo que no es posible extraer la información de los píxeles de las imágenes directamente, y tampoco provee funciones para cargar vídeos digitales en memoria. Para leer la información de los píxeles de las imágenes que corresponden a cada *frame* es necesario utilizar bibliotecas adicionales, sobre la plataforma Windows, se decidió utilizar la biblioteca “vfw32” (*Video for Windows*), y en el caso de Linux “avifile” en la versión 0.7.

En la documentación, se brindaban pseudocódigos que mostraban la lógica a seguir para la lectura de la información contenida en los ficheros AVI, pero estos

eran muy generales y estaban enfocados al trabajo con vídeos sobre plataforma Windows. Los ejemplos de código que se brindaban no eran reutilizables ya que estaban enfocados a propósitos personales de sus autores más allá de que se pudieran utilizar de forma genérica en cualquier ambiente y con distintos propósitos. Por lo que se decidió implementar una nueva solución basada en los trabajos consultados pero enfocada específicamente a ser utilizada en la herramienta STK.

Teniendo en cuenta que en Windows se guardan los colores RGB (rojo verde azul) como BGR (azul verde rojo), era necesario corregir el problema, por el que los colores se muestran intercambiados, rojo con azul. En los materiales consultados esta operación se realizaba por cada *frame* antes de generar la textura, para lo cual se recorrían todos los píxeles de cada imagen del vídeo y se intercambiaba el color rojo con el azul. Esta operación es muy costosa en cuanto al tiempo que tarda y se incrementaba en correspondencia con las dimensiones de las imágenes. En la nueva solución se modifica el código de generación de texturas para usar GL_BGR en vez de GL_RGB.

El control del tiempo para la transición de las imágenes, se maneja a nivel de milisegundos, se controla para generar sólo la textura perteneciente al fotograma que se corresponde visualizar, evitando generaciones innecesarias de un mismo *frame*, así como para mantener una reproducción en correspondencia con la información de *frame* por segundos que tenga el vídeo, lo que permite que las transiciones no ocurran aceleradamente o muy lentas.

Al terminar de desarrollar esta técnica, muy similar a las analizadas en la bibliografía, se realizaron pruebas en una PC con procesador Pentium 4, a 3.0 Ghz, 1 Gb de memoria RAM y 128 Mb de video *onboard ATI RADEON XPRESS*. Los resultados de las pruebas se muestran en la Tabla 1, en la columna “Técnica inicial”. Al aumentar la resolución del vídeo, disminuía drásticamente el rendimiento de la aplicación.

Se pretendía aumentar el rendimiento de la solución implementada, por lo que se analizó nuevamente la lógica de la técnica utilizada. Se detectó que el paso que más influía en la pérdida de rendimiento lo constituía la generación de las texturas.

Se intentó generar todas las texturas desde un inicio con diferentes identificadores, para que cuando fuese necesario su utilización ya estuviesen generadas y que el proceso sólo consistiese en seleccionar el identificador adecuado. Esta solución no fue factible ya que un vídeo de cuatro minutos y de 30 *frames* por segundos tiene 7200 *frames* lo que significa 7200 texturas generadas y almacenadas en memoria, lo que provocaba que el programa colapsase.

Se consideró hacer una generación paulatina, a medida que se fuesen consumiendo algunos *frames* del vídeo se iban generando las texturas de los próximos *frames*. Para la implementación de esta solución se utilizó la biblioteca de multihilos "boost" [43]. En el programa principal se realizaba la carga y visualización del vídeo, mientras que en un hilo en paralelo se realizaba la generación de las nuevas texturas. Para que las funciones de OpenGL funcionasen en un hilo en paralelo a la aplicación principal, era necesario generar un nuevo contexto de OpenGL y enlazarlo con el contexto ya existente del programa principal [44]. Al generar un nuevo contexto el rendimiento de la aplicación disminuyó a la mitad por lo que se decidió no utilizar esta técnica.

Basados en que la pérdida de rendimiento estaba principalmente en la generación de las texturas se intentó optimizar este paso. En OpenGL las texturas son internas al driver por lo que no se puede obtener un puntero a la textura. Uno de los parámetros necesarios para su generación son los datos correspondientes a los píxeles de cada *frame* del vídeo. El driver copia internamente los datos almacenados en memoria RAM a través de la función de generación de textura, y se actualiza internamente. Cuando se carga la textura una vez y se utiliza durante el transcurso de toda la aplicación no conlleva muchos problemas, pero como se requiere realizar copias continuamente, provoca una caída de rendimiento. Si se lograba que cuando fuese invocada la función de generar textura, la información de los píxeles correspondiente a cada *frame* ya estuviese en memoria, significaría una mejoraba en el rendimiento. Se plantea como técnica de solución al problema, la utilización de la técnica basada en "*pixel buffer object*".

La técnica que se implementó consiste en generar un *buffer* en la memoria gráfica. En este *buffer* se guardan los datos de cada píxel que compone el *frame* del vídeo y se le especifica al *driver* que los remplace según se vayan generando nuevos

frames, de esta forma se elimina la copia intermedia que se producía en el mismo momento de generar la textura. Antes de generar la textura es necesario activar el *buffer* donde está almacenada la información, para ello se utiliza un mecanismo basado en identificadores, y además es necesario especificar un identificador de textura para cada vídeo, para evitar que las modificaciones se apliquen sobre otras texturas que pueda contener la aplicación. Cuando se activa el *buffer,* que contiene la información que se requiere, todas las operaciones que se especifiquen se realizan con sus datos almacenados por lo que se hace necesaria su desactivación al finalizar su utilización.

Con esta técnica se logra aumentar el rendimiento de la aplicación, como se muestra en la Tabla 1. A pesar de que también decae el rendimiento en dependencia de las dimensiones del vídeo, se obtiene una mayor prestación en comparación con la técnica inicial.

Dimensiones	Técnica inicial (fps)	Utilizando *Pixel Buffer Object* (fps)
256 X 256	61	82
256 X 512	38	54
512 X 512	30	40
600 X 800	19	32

Tabla 1: Frame por segundos (fps) en función de la técnica utilizada.

CAPÍTULO 3: RESULTADOS Y VALIDACIONES

En el presente capítulo se comentarán los resultados obtenidos y se describirán las funcionalidades desarrolladas.

3.1 Funcionalidades

El módulo obtenido permite el control de la reproducción en todo momento, es posible pausar la reproducción, detenerla o reiniciarla nuevamente. Se pueden hacer reproducciones cíclicas o reproducir un vídeo un determinado número de veces, esto tiene gran utilidad si se pretende utilizar en las presentaciones o al inicio de los diferentes niveles en un videojuego.

El diagrama de secuencia que se muestra en la Figura 4, muestra al actor Aplicación interactuando con la clase CVideo, la cual envía, cuando el actor desee reproducir el vídeo, el mensaje de comenzar la reproducción a las clases encargadas de manipular la información referente a la imagen y al audio respectivamente. Como resultado se comienza la reproducción del vídeo.

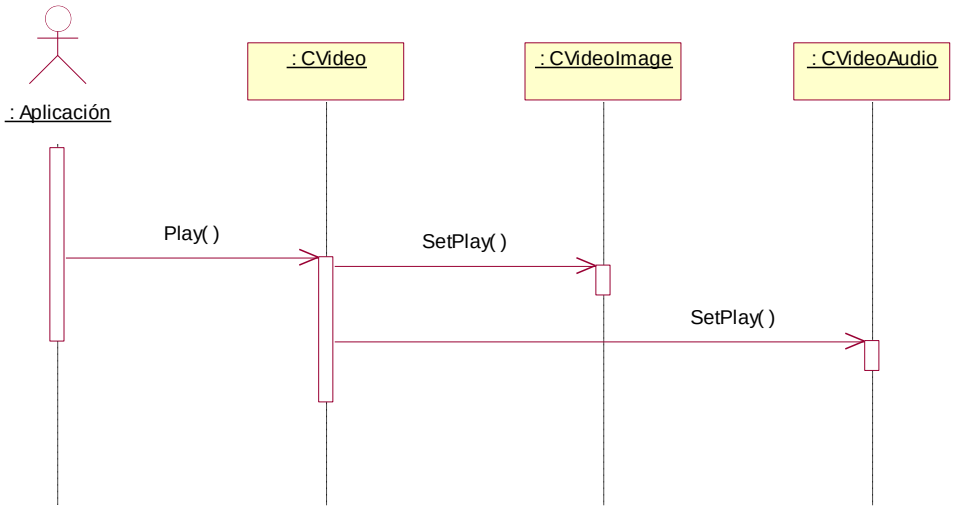

Figura 4: Diagrama de Secuencia Controlar Vídeo Sección: Play.

De igual forma ocurre cuando se desea pausar la reproducción. La Figura 5, muestra la clase CVideo, que envía a petición del actor, el mensaje de pausar la reproducción a las clases encargadas de manipular la información referente a la imagen y al audio respectivamente. Como resultado la reproducción del vídeo es pausada.

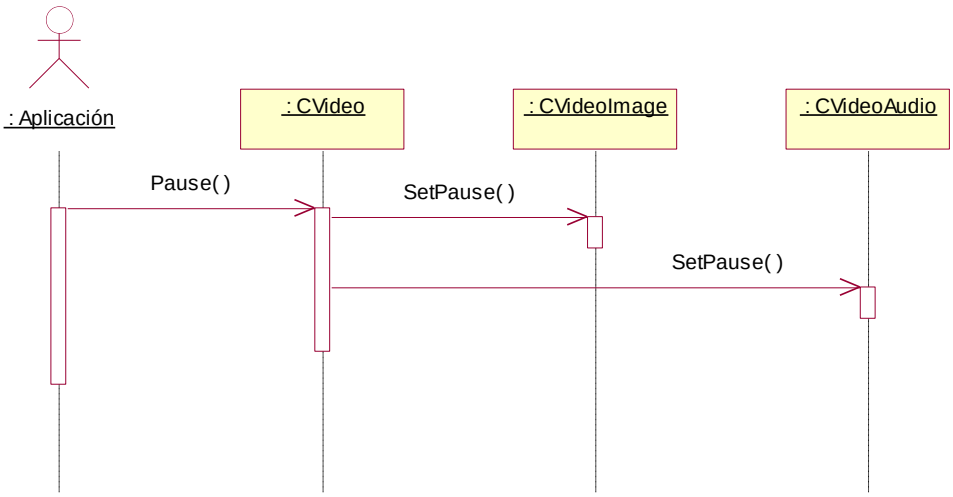

Figura 5: Diagrama de Secuencia Controlar Vídeo Sección: Pause.

El diagrama de secuencia que se muestra en la Figura 6, muestra el flujo de información desde el actor del sistema hasta las clases encargadas de manipular la información referente a la imagen y al audio del vídeo, cuando se requiere detener la reproducción. Como resultado se detiene la reproducción del vídeo.

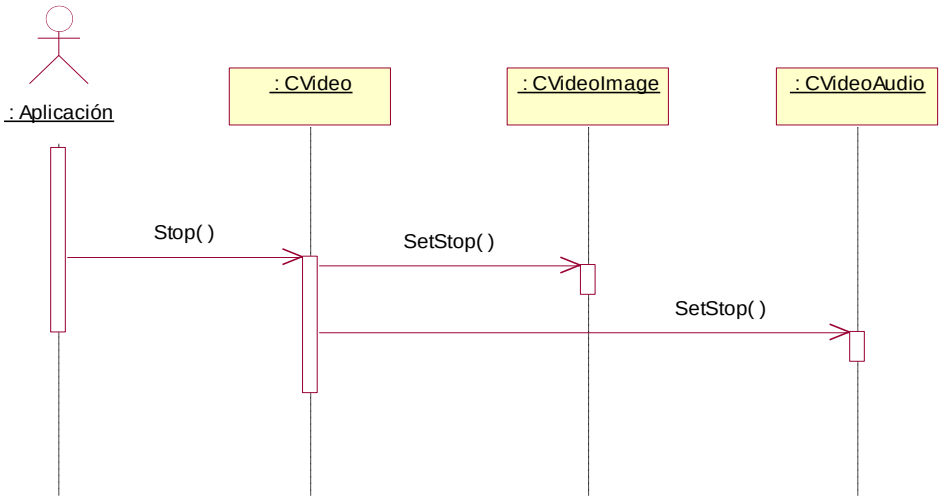

Figura 6: Diagrama de Secuencia Controlar Vídeo Sección: Stop.

A diferencia de otras implementaciones analizadas, se brinda un reporte de errores, que apoya a los programadores ya que se informa cuando el fichero de vídeo está corrupto o se detecta que el audio o las imágenes no se pueden descomprimir. Además se detecta si el fichero existe en la dirección desde la cual se está tratando de cargar en memoria.

La posición en el mundo virtual, del polígono donde se visualiza el vídeo, es editable por lo se puede representar en cualquier posición, no sólo en un primer plano. Pudiéndose utilizar para la simulación de objetos 3D como televisores y pantallas, así como para recrear escenas filmadas en el mundo real.

Al permitir darle profundidad a la superficie donde se visualizan los vídeos es posible la visualización de varios al mismo tiempo. El módulo permite esta opción para lo cual se pueden crear varias instancias de la clase CVideo e internamente se crea un identificador diferente para cada uno. El número de vídeos que de forma concurrente se pueden reproducir está en dependencia del *hardware*.

Se implementó un sistema multiplataforma, como se muestra en el diagrama de componentes (ver Figura 7). La arquitectura que se utilizó para la implementación permitió agrupar las funcionalidad desarrolladas en un módulo que puede ser utilizado por todo aquel que requiera incluir un vídeo en las aplicaciones gráfica que desarrolle utilizando OpenGL (ver Figura 17, Anexo 11) o un motor gráfico como Ogre al cual también se pudo integral (ver Figura 21, Anexo 15). El programador no tiene que especificar la plataforma que está utilizando, este proceso es transparente a él. Para ello se especificaron directivas de preprocesador en cada uno de los ficheros.

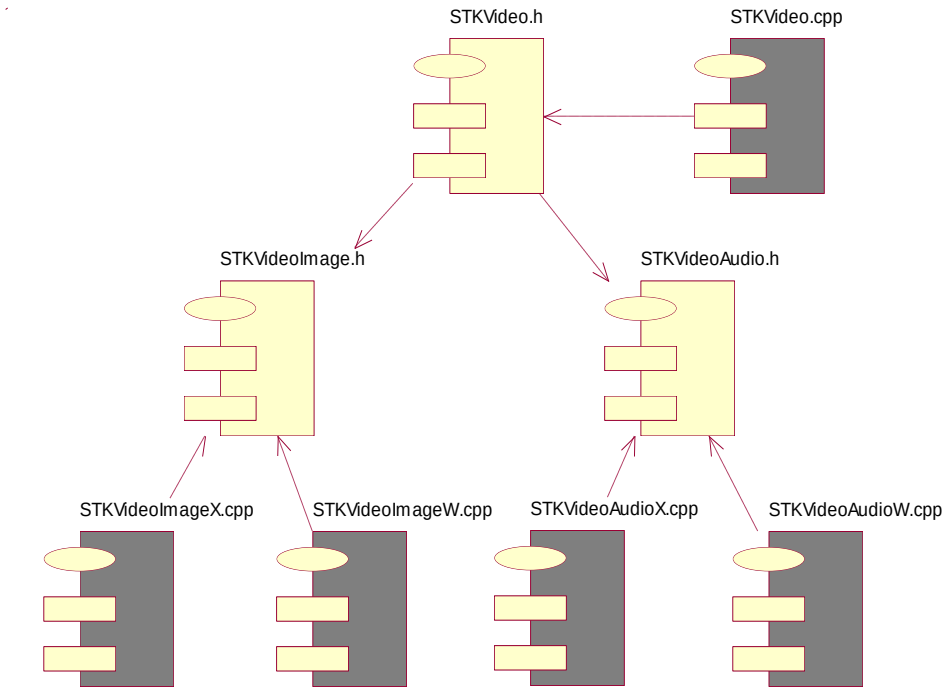

Figura 7: Diagrama de componentes.

Se implementó una clase “STKVideo”, compuesta por las clases “STKVideoAudio” y “STKVideoImage” las cuales se especializan en el manejo del audio o de la imagen respectivamente. La implementación, para cada plataforma, se realizó bajo el mismo estándar de código, se utilizaron nombres iguales, para los métodos así como para los atributos comunes, pero para tener agrupadas todas las funcionalidades en un mismo fichero (.cpp) se hacía necesario identificar en todo momento la plataforma que se estaba utilizando, por lo que se decidió separar las implementaciones por plataforma y el compilador compilaba la implementación correspondiente a la plataforma en la que se estuviera trabajando, por lo que se cuenta con dos archivos (.cpp) por cada (.h), correspondientes a las clases que manejan la información de la imagen y el sonido del vídeo.

Esta estructura permite que en trabajos posteriores, para brindar soporte para otras plataformas, sólo se necesite incluir una nueva implementación en dependencia de sus características y no se tengan que modificar las implementaciones ya existentes.

3.2 Pruebas realizadas

Luego de aplicarle distintos códec de vídeo a un mismo fichero, de cuatro minutos de duración, y con una dimensión de 256 píxeles de ancho por 256 píxeles de altura, con un *framerate* de 30 *frames* por segundo, se obtuvo salidas de diferentes tamaños, (ver Tabla 1).

Códec	**Tamaño (Mb)**
Cinepak Códec por Radius	105
Códec Intel IYUV	687
DivX® 5.2.1 Códec	32,1
Indeo® video 5.10	49,7
Intel Indeo(R) Video R3.2	42,7
Intel Indeo® Video 4.5	60,5
Microsoft Video 1	114
Uncompressed	1362

Tabla 2: Tamaño de los ficheros dependiendo del códec.

Partiendo de que se contaba con varios ficheros, a los cuales se les había aplicado distintos códec, se prosiguió a su visualización, la Figura 8 muestra una comparación entre tamaño y *frames* por segundos dependiendo del códec utilizado.

Al analizar esta comparación entre estos distintos ficheros y los resultados en cuanto a *frames* por segundos, eran prácticamente constantes, no existían diferencias ya que la descompresión del vídeo se realiza antes de comenzar la reproducción.

Las diferencias se ven representadas cuando se compara el tamaño de los ficheros, siendo el de menor tamaño el fichero codificado con DivX y el de mayor tamaño el fichero sin codificar.

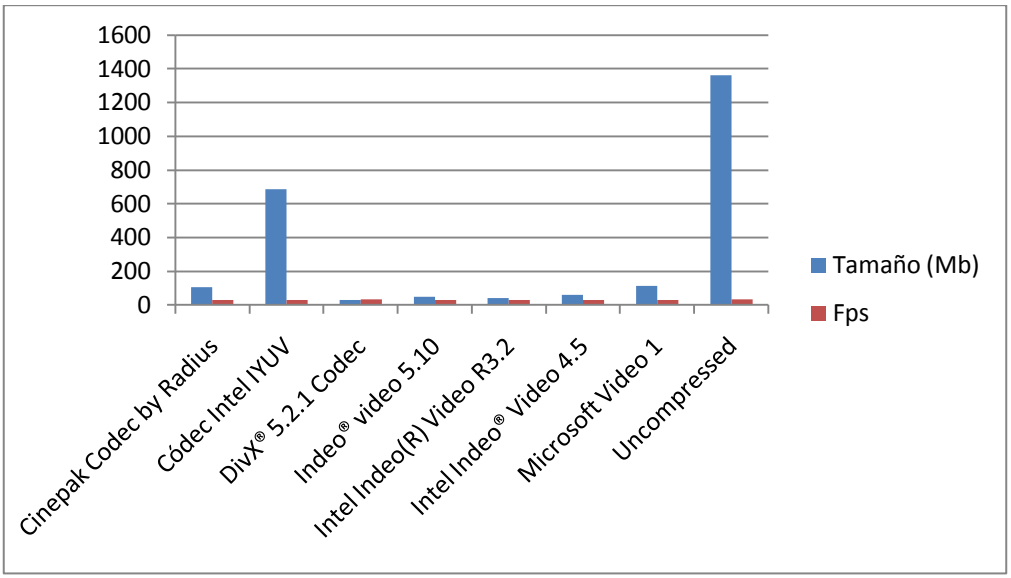

Figura 8: Tamaño y Fps por códec de vídeo.

Luego de realizar esta comparación se recomienda utilizar los vídeos codificados con DivX, ya que además de ser los de menor tamaño, durante las pruebas realizadas a simple vista no se detectan anomalías en las imágenes, no se recomienda el empleo de ficheros sin codificar ya que el tamaño de los ficheros es muy grande.

Partiendo de que los mejores resultados en cuanto a tamaño, respecto a *frames* por segundo, se obtuvo con ficheros codificados con DivX, se modificó varias veces las dimensiones de tamaño de un mismo vídeo manteniendo constante este códec, (ver Tabla 3).

Dimensiones	Códec
256 X 256	DivX® 5.2.1 Códec
340 X 320	DivX® 5.2.1 Códec
340 X 320	DivX® 5.2.1 Códec
400 X 400	DivX® 5.2.1 Códec
640 X 480	DivX® 5.2.1 Códec
800 X 600	DivX® 5.2.1 Códec

Tabla 3: Dimensiones bajo un mismo códec.

Al realizar la representación de estos ficheros de diferentes dimensiones, se obtuvo que los *frames* disminuyeran dependiendo del aumento de las dimensiones del vídeo, (ver Figura 9), ya que las texturas de los *frames* que se van generando son de mayor tamaño.

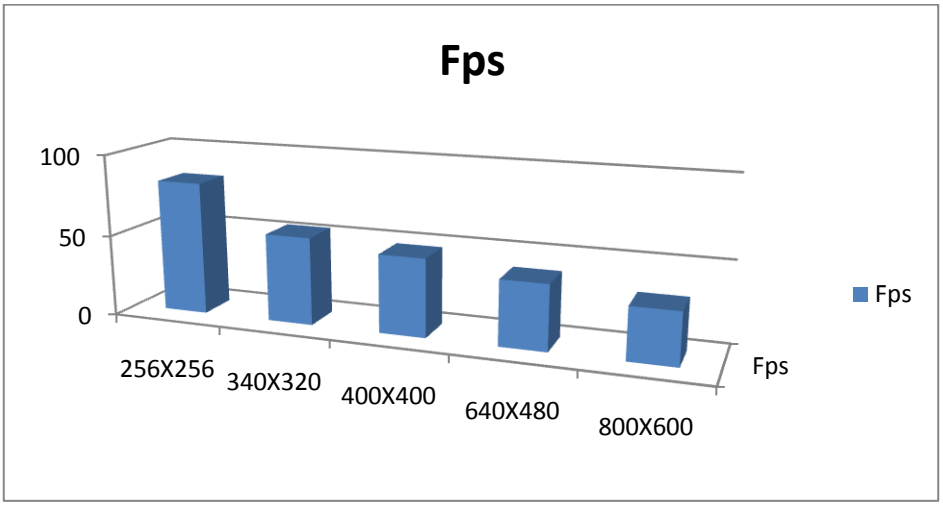

Figura 9: Frames por segundos en función de las dimensiones.

Leer ficheros de vídeo no implica leer audio e imagen al mismo tiempo, puede ser que sólo uno de ellos sea el que se requiera reproducir. Se comprobó con dos ficheros, ambo de extensión (.avi), pero cada uno contenedor de un sólo tipo de información.

Se tomó como muestra para realizar la prueba, un vídeo de dimensiones 256 píxeles de ancho por 256 píxeles de altura, con un solo canal de audio y codificado con DivX® 5.2.1.

En el caso de la reproducción del fichero que contenía solo la información del sonido, se logró que la aplicación funcionara a una mayor velocidad, que cuando la imagen del vídeo era procesada, lo que demuestra que la generación de texturas es la operación del módulo, que más tiempo consume (ver Figura 10).

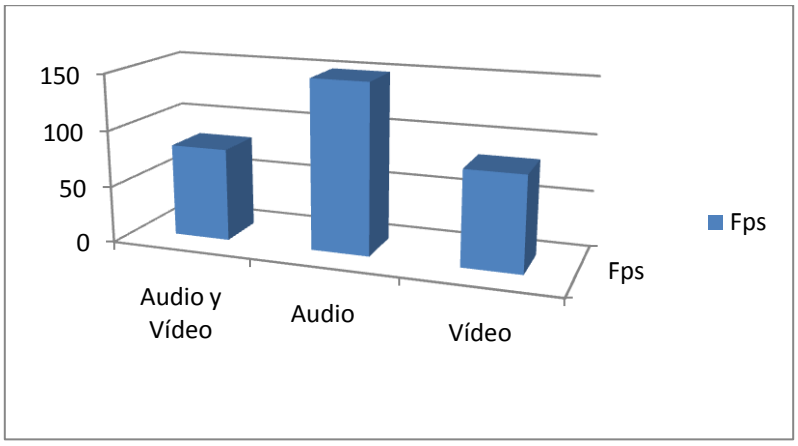

Figura 10: Frame por segundos en función del contenido.

Se logró utilizar los vídeos digitales de extensión (.avi), como presentación en las aplicaciones que se desarrollan con la herramienta SceneToolKit (ver Figura 20, Anexo 14), además se permite colocar múltiples vídeos en distintas posiciones del mundo virtual, (ver Figura 18, Anexo 12).

CONCLUSIONES

Con el trabajo se obtuvo un módulo multiplataforma capaz de representar la imagen de un vídeo en un mundo 3D, para la herramienta SceneToolKit. Con la utilización de la técnica de *pixel buffer object* se logró la visualización de vídeos de mayores dimensiones, ya que se aumentó el rendimiento en el proceso de generación de texturas.

RECOMENDACIONES

Se recomienda utilizar una única biblioteca para la lectura de los datos contenidos en los ficheros de vídeo digital.

Se recomienda reproducir el sonido de una forma 3D, ya que se logró poder visualizar un vídeo en un polígono localizado en cualquier posición dentro de un mundo virtual, se hace necesario poder percibir el sonido con profundidad 3D.

BIBLIOGRAFÍA

[1]. Palladium. [Online] Área de cultura y lenguas clásicas. Ministerio de Educación. Gobierno de España. [Cited: septiembre 10, 2010] http://recursos.cnice.mec.es/latingriego/Palladium/5_aps/esplap03.htm

[2]. Amanatides, John. Mitchell, Don P.“Antialiasing of Interlaced Video Animation,” Computer Graphics 4. Television Color Encoding and “Hot” Broadcast Colors) 1990

[3]. Schreer, Oliver. Kauff, Peter. Sikora Thomas. 3D Videocomunication, Algorithms, concepts and real-time systems in human centred communication. England. 2005

[4]. Ramos Raul, Universidad Pompeu Fabra de Barcelona. [Online] Universidad Pompeu Fabra de Barcelona. [Cited: septiembre 22, 2010] http://www.iua.upf.es/~berenguer/recursos/ima_dig/_8_/estampas/3_1.htm

[5]. Turcan Peter, Wasson Mike. Fundamentals of Audio and vídeo Programming for Games. [Documento .chm] ISBN:073561945x [2004].

[6]. Waschbüsch Michael. 3D video Acquisition, Representation & Editing. [Documento .pdf]. ETH No. 17558. For the Degree of Doctor of Sciences.[2007].

[7]. Alfaro Ferreres, Luis and Roca Estellés, María José. Universidad de Jaén. [Documento .doc] Universidad de Jaén. [Cited: enero 29, 2010.] http://wwwdi.ujaen.es/asignaturas/informatica/Teoria/Software%20Grafico/Grafico-doc/Graficos%20audio%20y%20video.doc

[8]. Megaservice A.S.D.A S.R.L. [Cited: febrero 11, 2010.] http://www.megaservice.com.ar/Info_espe/Los%20archivos%20AVI.htm

[9]. E.U.I.T. de Informática y Telemática. [Documento .pdf] [Cited: marzo 2, 2010.] http://lear.inforg.uniovi.es/sm/descargas/teoria/Formatos%20de%20Video.pdf

[10]. Erino, José Ignacio. La taberna del Grumete. [Online] [Cited: enero 29, 2010.] http://www.eumed.net/grumetes/2004/formatoavi.htm

[11]. Junker Gregory. Pro OGRE 3D Programming .[Documento .pdf] [2006]. ISBN-13: 978-1-59059-710-1 and ISBN-10: 1-59059-710-9. Library of Congress Cataloging 9 8 7 6 5 4 3 2 1.

[12]. Allegro. A game programming library. http://alleg.sourceforge.net/

[13]. Thomson Course Technology 2nd Edition.[Documento .pdf] Game Programming, chapter 17, Playing FLIC Movies. ISBN: 1-59200-383-4. Library of Congress Catalog Card Number: 2004091915, 2004.

[14]. Molofee, Jeff (NeHe): Playing AVI Files. Gamedev.net. [Online] NeHe Production. [Cited: febrero 19, 2010.]

[15]. Molofee, Jeff (NeHe), NeHe SDK http://nehe.gamedev.net/counter.asp?file=files/resources/nehe_sdk.zip

[16]. E.U.I.T. de Informática y Telemática. [Online] Formatos de Vídeos Digitales. [Cited: marzo 2, 2010.] http://lear.inforg.uniovi.es/sm/descargas/teoria/Formatos%20de%20Video.pdf

[17]. Instituto Corinaldesi. [Online] Imágenes Digitales [Cited: febrero 11, 2010.] www.corinaldesi.it/Progetti/pro-storici/gioconda/ricerca/IMAGENES%20DIGITALES.doc

[18]. Copyright 2010 Autodesk, Inc. All rights reserved. http://usa.autodesk.com/

[19]. Camacho Roman, Yanoski, Jimenez Lopez, Fernando: Informe Final del proyecto Herramientas de Desarrollo para Sistemas de Realidad Virtual. Informe técnico, Facultad 5, UCI [Documento .pdf] InformeFinalProyecto.pdf. 2009.

[20]. Camacho Román, Yanoski. STK 2.3.1. [Documento .pdf] La habana: UCI, 2008.

[21]. Electronic Arts 2010. http://www.ea.com/uk/page/about-ea. 2010

[22]. Electronic Arts Inc 2010. http://www.ea.com/intl/football/fifa. 2010

[23]. Electronic Arts 2010. Need for Speed World. http://www.ea.com/uk/game/need-for-speed-world.

[24]. Electronic Arts 2010 .http://www.ea.com/es/juegos/nba-live.

[25]. Donald Marshall,Copyright 2001-2004. http://www.projectigi2.co.uk/site/story.php

[26]. Activision Publishing, Inc. Activision and Call of Duty are registered trademarks & Black Ops is a trademark of Activision Publishing, Inc. 2010 http://www.callofduty.com/hub.

[27]. Electronic Arts Inc. Trademarks belong to their respective owners. All rights reserved. http://www.medalofhonor.com/

[28]. Blizzard Entertainment. http://www.worldofwarcraft.com/splash-sc2date.htm

[29]. Virtual-E Corporation. 2010. http://www.virtualgt.com/

[30]. KdCompany. http://www.kdcompany.ru/es/pneumo.shtml

[31]. Horacio Hernan Moraldo, DirectShow video in ogre texture http://www.ogre3d.org/tikiwiki/DirectShow+video+in+ogre+texture October, 2010

[32]. Abi George Ullattil. A small code snippet for playing videos on Ogre materials / textures using DirectShow http://www.ogre3d.org/tikiwiki/DirectShow+video+in+ogre+texture&structure=Cookbook

[33]. msdn. Windows Embedded Developer Center [Online] Microsoft. [Cited: abril 3, 2010.] http://msdn.microsoft.com/en-us/library/aa909811.aspx ,2010.

[34]. Edited by James Arvo. Graphics Gems II. Program of Computer Graphics Cornell University, Ithaca, New York. ISBN 0-12-059756-X.

[35]. Edited by David Kirk. Graphics Gems III. California Institute of Technology Computer Graphics Laboratory Pasadena, California. ISBN 0-12-059756-X.

[36]. Guibas, Leo. Stolfi, Jorge . A Language for Bitmap Manipulation, ACM Transactions on Graphics.

[37]. HECKBERT, P. S. Fundamentals of texture mapping and image warping. Master's thesis, CS Division, U.C. Berkeley. 1989

[38]. Van der Byl, Leigh. LightWave 3D. 8 Texturing. ISBN 1-55622-285-8. Texas. 2004

[39]. Singer Marq. Real-Time Sound Generation from Deformable Meshs. Gem VI Section 6 Audio.

[40]. Boer, James. Dynamic Variables and Audio Programming. Game Programming Gem IV, Charles River Media. 2004.

[41]. Bencina, Ross. Phil, Burk. PortAudio, http:// www.portaudio.com. 2005

[42]. Andreas Hartl, 2010. Programming Tips & Tricks. Bitmap Tutorial: Loading and Saving Bitmaps. ttp://tipsandtricks.runicsoft.com/Cpp/BitmapTutorial.html

[43]. Beman Dawes, David Abrahams, 1998-2005. Copyright Rene Rivera 2004-2007. http://www.boost.org/

[44]. OpenGL. The Industry's Foundation for High Performance Graphics. Creating an OpenGL Context. 25 June 2011 http://www.opengl.org/wiki/Creating_an_OpenGL_Context

[45]. OpenGL. The Industry's Foundation for High Performance Graphics. Pixel Buffer Object. 25 March 2011 http://www.opengl.org/wiki/Pixel_Buffer_Object

[46]. Ikits, Milan and Magallon, Marcelo. GLEW http://glew.sourceforge.net/

[47]. Nix, Jonathan: Working whith AVI Files. Gamedev.net. [Online] gamedev.net. [Cited: febrero 21, 2010] http://www.gamedev.net/reference/programming/features/avifile/, 1999.

[48]. McGowan, John F. [Online] AVI Overview. [Cited: marzo 4, 2010.] http://www.jmcgowan.com/avi.html. , 2004.

[49] CodeBlocks. The open source, cross platform, free C++ IDE http://www.codeblocks.org/

[50]. msdn. Windows Embedded Developer Center [Online] Microsoft Corporation. [Cited: abril 3, 2010.]http://msdn.microsoft.com/en-us/library/aa909811.aspx ,2010.

[51]. Kabelac, Zdenek. Linux AVI file library [Online] [Cited: mayo 16, 2010.] http://avifile.sourceforge.net/, 2008.

[52]. Textura Animada. [Documento .pdf] Dpto. Informática, Universidad de València. Ampliación de Informática Gráfica Tema1, capitulo 7.1.6. http://informatica.uv.es/iiguia/AIG/teoria.htm, 2008.

[53]. Iznaga Benites, Arsenio. Perez Mallea, Ivan. Fundamentos de la gráfica por computadora. Nov. 2006.

[54]. Neider, J., Davis, T., Woo, M.; The OpenGL Programming Guide, 1993.

[55]. Foley, James D. Dam, Andries, Feiner, Steven K., and Hughes, John F. Computer Graphics Principles and Practice. Addison-Wesley, Reading, Massachusetts. General Filtered Image Rescaling

[56]. LI, Y., SUN, J., and SHUM, H.-Y., 2005. Video object cut and paste. Proc. of SIGGRAPH '05

[57]. GROSS, M. Visual Computing: Integration of Computer Graphics, Visual Perception and Imaging. Springer-Verlag.1994.

[58] Hoffert, E. ; Krueger, M. ; Mighdoll, L. ; Mills, M. Apple Comput. Inc., Cupertino, CA . Compcon Spring '92. Thirty-Seventh IEEE Computer Society International Conference, Digest of Papers. ISBN: 0-8186-2655-0. INSPEC Accession Number: 4315110 Digital Object Identifier : 10.1109/CMPCON.1992.186680. August 2002

[59] http://www.apple.com/es/quicktime/ 2011 Apple Inc.

[60] http://www.apple.com/es/quicktime/download/

[61] http://www.mplayerhq.hu/design7/info.html 2000-2011 The MPlayer Project

[62] http://licencias-os.blogspot.com/2009/07/quicktime.html Publicado por Fredy Rojas en 1/1/2009 APPLE INC. CONTRATO DE LICENCIA DE SOFTWARE PARA QUICKTIME

[63] http://quicktime.programas-gratis.net/ Programas-gratis.net © Mediaprogramas 2011 QuickTime Gratis

[64] http://www.mpeg.org/ Mpeg*TV* and edited by Tristan Savatier who has been an active member of the MPEG Committee from 1988 to 1995 and has participated to the making of the MPEG-1 and MPEG-2 Video standards.

[65] http://mpeg.chiariglione.org/ The MPEG Home Page Leonardo Chiariglione

[66]. *Real.* [Online] RealNetworks. [Cited: febrero 23, 2008] http://www.realnetworks.com/products/codecs/realaudio.html

[67] *Real* Player http://www.real.com/realplayer/ 2011 RealNetworks, Inc. RealPlayer

[68] Advanced Systems Format (ASF) Specification Revision 01.20.03 Microsoft Corporation December 2004

[69] Advogato. Microsoft patents ASF media file format, stops reverse engineering. 2000

[70] Bahsoon, R. y W Emmerich: Evaluating software architectures: development, stability and evolution. Julio 14{18 2003.

[71] Member #10,454 . Allegro. http://www.allegro.cc/forums/thread/599762. 2009

[72] Oscar Giner [http://oginer.webcindario.com/index.php?page=allegavi] 2002

[73] Oscar Giner. AllegAVI Documentation. http://oginer.webcindario.com/allegavionlinedocs/. 2004

GLOSARIO

API: Interfaz de programación de aplicaciones (Applications Programming Interface): una serie de funciones que están disponibles para realizar programas para un cierto entorno.

Archivo: Elemento en el que se guarda información: textos, datos, gráficos, fotos, películas, música. Se identifica mediante un nombre, por ejemplo, "misdatos.txt", "foto.jpg", "cancion.mp3" o “video.avi”.

AVI: Es un tipo de archivo contenedor, que puede contener una pista de video codificada y una o más pistas de audio. Su nombre proviene de que el video y audio están "entrelazados" dentro del contenedor.

Bitrate: Hace referencia a la velocidad con que se transmiten los datos de un contenido multimedia (video o audio). Se suele medir en Kbps (Kilobits por segundo), y por tanto indica el valor medio de Kilobits que ocupa cada segundo de contenido reproducido. Cuanto más alto sea el bitrate, más espacio ocuparán los archivos, pero también se conseguirá mejor calidad ya que hay mayor cantidad de datos disponibles y los contenidos se pueden reproducir con mayor detalle.

Buffers: Memoria intermedia, una porción reservada de la memoria, que se utiliza para almacenar datos mientras son procesados.

C++: Lenguaje de programación orientado a objetos.

Campo: Cada una de las partes que forman un video entrelazado. Los campos no contienen imágenes completas, sino las líneas que forman la imagen (par o impar según el campo).

Códec: Acrónimo de codificador/decodificador. Es un software que consta de una serie de algoritmos que permiten codificar y decodificar contenidos multimedia.

Entrelazado: Es un tipo de almacenamiento de video donde los fotogramas no se guardan como imágenes completas, sino en dos campos donde cada uno contiene la mitad de las líneas que forman el fotograma (un campo las líneas pares y el otro las impares).

Digital: Representación de la información mediante combinaciones de unidades binarias o '*bits*'.

Formato: En el mundo de la informática, es el conjunto de reglas o especificaciones mediante las cuales se pueden organizar datos de diversa naturaleza, para poder acceder posteriormente a estos a través de los intérpretes adecuados.

Frame: Es cada imagen de una película, es decir, un fotograma. La sucesión de frames a gran velocidad produce el efecto de movimiento.

Framerate: Es la velocidad con que pasan los frames (fotogramas) en una película. Su velocidad se mide en fps (frames por segundo). En el estándar PAL (Europa) esta velocidad es de 25 fps, mientras que en el estándar NTSC (América) la velocidad es de 29.97 fps, aunque existe una variante NTSC Film a 23.976 fps.

Interleaving: Son los puntos de conexión entre audio y video que aseguran la sincronización cuando nos desplazamos a través de nuestra película. Suponen puntos de unión a lo largo de la película entre audio y video, que el ordenador tomará como referencia.

Linux: Sistema operativo desarrollado inicialmente por Linux Torvalds, un estudiante finlandés de Informática. Actualmente son miles de personas las que colaboran en todo el mundo, en su desarrollo. Se desarrolla como software libre, por lo que puede distribuirse y copiarse libremente. Existen muchas distribuciones que lo distribuyen.

Multiplataforma: Que la aplicación corra en varios sistemas operativos.

Mundos virtuales: Simulación de mundos o entornos, denominados virtuales, en los que el hombre interacciona con la máquina en entornos artificiales semejantes a la vida real.

Ogg: Formato contenedor que incluye un número de códecs separados e independientes de vídeo y audio, ambos desarrollados en código abierto.

OpenAL: significa Open Audio Library, lo que en castellano significa: Biblioteca Abierta de Audio. Pretende ser una extensión de OpenGl que provea herramientas para el manejo de audio.

OpenGl: es una biblioteca gráfica desarrollada originalmente por Silicon Graphics Incorporated (SGI). OpenGl significa Open Graphics Library, cuya traducción es biblioteca abierta de gráficos.

Píxel: Es cada uno de los puntos de luz que forman una imagen en un monitor, es decir, la unidad básica que forma una imagen.

Progresivo: Es lo opuesto a entrelazado. Los fotogramas se guardan como imágenes completas y no en campos entrelazados.

Realidad: cualidad o estado de ser real o verdadero.

Realidad Virtual: La Realidad Virtual es simulación por computadora, dinámica y tridimensional, con alto contenido gráfico, acústico y táctil, orientada a la visualización de situaciones y variables complejas, durante la cual el usuario ingresa, a través del uso de sofisticados dispositivos de entrada, a "mundos" que aparentan ser reales, resultando inmerso en ambientes altamente participativos, de origen artificial.

Resolución: Es el número de píxeles que se muestran en una pantalla. Al ser ésta una matriz de filas y columnas de píxeles, primero se nombra la cantidad de columnas (resolución horizontal) y luego la cantidad de filas (resolución vertical).

RGB: Es el acrónimo inglés Red, Green, Blue (Rojo, Verde, Azul). Es un modelo de color en el cual es posible representar un color mediante la mezcla de tres colores primarios: rojo, verde y azul.

Video digital: Una señal de video representado por computadora y puede leer números binarios que describen un conjunto finito de colores y niveles de luminosidad.

Virtual: existe o resulta en esencia o efecto pero no como forma, nombre o hecho real.

ANEXOS

Anexo 1:

Figura 11: Need for Speed.

Anexo 2:

Figura 12: MVP.

Anexo 3:

Figura 13: VirtualGT (Simulador personal de conducción).

Anexo 4:

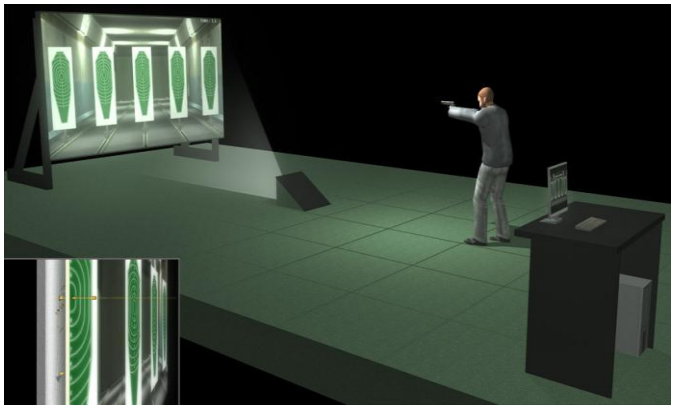

Figura 14: KD (Simulador de Tiro).

Anexo 5:

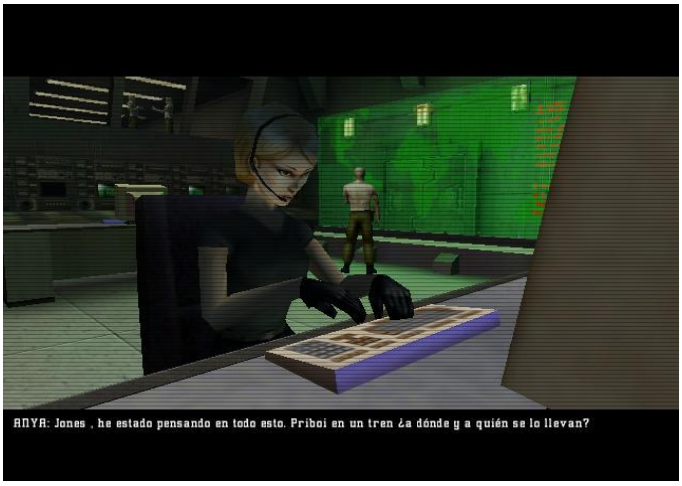

Figura 15: Proyecto IGI.

Anexo 6:

Figura 16: Fifa 2005.

Anexo 7:

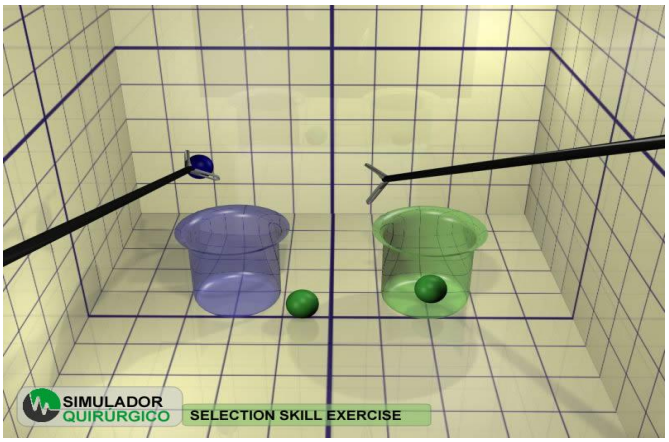

Figura 17: Simulador Quirúrgico.

Anexo 8:

Figura 18: Paseo Virtual UCI.

Anexo 9:

Figura 19: Meteorix.

Anexo 10:

Figura 20: Energía para Aprender.

Anexo 11:

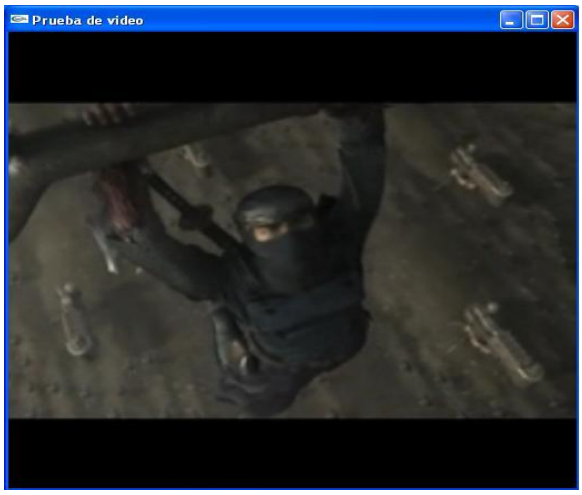

Figura 21: Utilización de un vídeo como presentación.

Anexo 12:

Figura 22: Cubo creado a partir de 6 planos texturizados con videos diferentes.

Anexo 13:

Figura 23: Sistema RGB en Linux y Windows respectivamente.

Anexo 14:

Figura 24: Reproducción de vídeos utilizando la herramienta SceneToolKit.

Anexo 15:

Figura 25: Reproducción de vídeos utilizando el motor gráfico Ogre.