Uso de aplicaciones de ejemplo en las clases de Informática Gráfica [*]

Marco A. Gómez Martín, Pedro P. Gómez Martín
Dpto. Sistemas Informáticos y Programación
Universidad Complutense de Madrid
28040 Madrid
e-mail: {marcoa, pedrop}@sip.ucm.es

Resumen

En este artículo se plantea el uso de aplicaciones de ejemplo como apoyo a la enseñanza de la asignatura Informática Gráfica. El desarrollo de dichas aplicaciones, al ser muy específicas, puede resultar laborioso para el profesor. Para evitarlo, proponemos el uso de un motor de código abierto pensado para el desarrollo de juegos denominado Nebula como alternativa para su implementación, al proporcionar un armazón que evita la repetición de código. Además se obtienen algunas ventajas adicionales, como la modificación dinámica de los ejemplos en tiempo de ejecución.

1. Introducción

El programa de la asignatura de Informática Gráfica comprende multitud de conceptos relativos a la creación, representación y dibujado de entornos tridimensionales. Por desgracia, muchos de ellos son difíciles de explicar en un aula haciendo uso tan sólo de pizarra y transparencias. La principal limitación se debe a su naturaleza de recursos didácticos bidimensionales y, especialmente, a su carencia de *dinamismo*, a pesar de lo cual son utilizados para explicar conceptos en las que intervienen tres dimensiones y bastante movimiento.

Esta limitación requiere a menudo que el profesor tenga que hacer uso de su imaginación para explicar determinados conceptos. Pongamos como ejemplo la exposición de los sistemas de referencia "anidados", en los que un sistema de coordenadas está subordinado a otro. Explicar esta idea a menudo termina con el profesor gesticulando y moviendo el brazo para indicar que el sistema de referencia de la muñeca viene expresado en el sistema de referencia del codo, que a su vez, depende del sistema del hombro.

[*] Financiado por el Ministerio de Educación y Ciencia (TIC2002-01961)

Existen otros conceptos para los que ni siquiera la gesticulación es suficiente. Por ejemplo, las transformaciones relacionadas con el plano cercano y lejano de la cámara se comprenden mucho mejor si se hace ver la idea intuitiva de sus efectos a partir de unas cuantas imágenes donde estos dos planos se hayan colocado a diferentes distancias. Aún mejor sería mostrar directamente un programa interactivo donde se puedan modificar las distancias a voluntad, para no atar la explicación a unas posiciones concretas.

En realidad todo esto tiene un problema de fondo más profundo que el mero desarrollo y uso de las aplicaciones de apoyo: los recursos didácticos disponibles en el aula. Tradicionalmente las clases de Informática Gráfica (y del resto de asignaturas) se han impartido utilizando la pizarra y las transparencias de acetato. Hoy en día se ha hecho popular el uso de "presentaciones" proyectadas con ordenadores portátiles con el añadido, en algunos casos, de animaciones muy simples disponibles en los programas de autoría de diapositivas.

Sin embargo, creemos que la disponibilidad de estos ordenadores debe aprovecharse más allá de hacer uso de los programas para presentaciones. En concreto, el uso de aplicaciones sencillas como la comentada previamente, desarrolladas específicamente para la explicación de conceptos concretos, suponen un gran apoyo a las explicaciones tradicionales de imágenes estáticas.

El problema de estas aplicaciones específicas es su coste de desarrollo, que supone una carga adicional para el profesor. En este artículo detallamos el uso de un motor gráfico de libre distribución que, si bien no está pensado explícitamente para este uso, facilita mucho el desarrollo de esas aplicaciones.

Las ideas aquí descritas surgieron a la hora de impartir el Curso de Programación de Videojuegos impartido en la Escuela Complutense de Verano en Madrid durante el mes de Julio de 2004. En él se utilizó Nebula [3] como motor gráfico.

En el siguiente apartado describiremos las carac-

terísticas aconsejables para que una aplicación tenga valor pedagógico. Los apartados tercero y cuarto hacen una descripción de Nebula. El quinto muestra algunos de los ejemplos implementados. La Sección 6 presenta algunas otras aplicaciones que pueden utilizarse en las clases de Informática Gráfica que ya están disponibles y no requieren un trabajo de desarrollo por parte del profesos. Por último, el artículo termina con algunas conclusiones.

2. Características de las demostraciones

La enseñanza de Informática Gráfica cuando el único recurso didáctico era la pizarra era complicada. Las explicaciones debían ser bastante teóricas, y el profesor tenía que enfrentarse al proceso de dibujar imagenes esquemáticas en el encerado con muy poca ayuda.

Las transparencias de acetato debieron mejorar un poco la situación, al permitir al profesor disponer de imagenes realizadas con más tiempo y cuidado. Por ejemplo, con un poco de dedicación, se podía disponer de imágenes que representaran de forma esquemática los efectos del cambio de parámetros en la proyección o en las transformaciones. Cuando las transparencias pudieron realizarse digitalmente, se podían incluso imprimir directamente las imagenes generadas por el ordenador con los parámetros que se estaban indicando.

Hoy en día, la disponibilidad de portátiles y cañones de proyección en las aulas es una realidad. Esto permite al profesorado eliminar las transparencias de acetato y hacer uso de programas informáticos para la exposición de presentaciones durante las clases. Este nuevo avance permite añadir pequeñas animaciones a las transparencias que hasta ese momento eran estáticas.

Sin embargo, la disponibilidad de portátiles abre una puerta adicional para facilitar aún más la enseñanza: la ejecución de programas. El profesor puede, por fin, mostrar *en el momento* la ejecución de los conceptos teóricos que está explicando. Por ejemplo, en lugar de ejecutar dos o tres imágenes estáticas sobre cómo afectan los parámetros de la proyección a la imagen final, es posible mostrar un programa que permita modificar dinámicamente esos parámetros y muestre en tiempo real la imagen resultado.

Esos programas deben cumplir varios requisitos para que su uso tenga sentido:

- **Fáciles de desarrollar:** a pesar de que puedan resultar muy útiles, el profesor tiene que dedicar un tiempo adicional a realizar los programas que considere apropiados para cada momento. Si el esfuerzo adicional es excesivo, seguramente descartará la idea de su desarrollo, manteniendo las clases habituales de pizarra y transparencias. La mejor de las opciones sería que las aplicaciones ya estuvieran desarrolladas, como las descritas en la Sección 6.

- **Interactivas:** la ventaja de estas aplicaciones frente a las transparencias (de acetato o digitales) es la posibilidad de modificar en el momento los parámetros que pretenden enseñar. Es decir deben ser interactivas para que el profesor pueda manipularlos fácilmente sobre la marcha y los resultados se muestren inmediatamente en tiempo real.

- **Fáciles de modificar:** al mostrar este tipo de aplicaciones, generalmente los alumnos prestan más atención y las ideas que se enseñan quedan más claras. Pero también aparecen nuevas preguntas relacionadas con el tema que se muestra. Desde el punto de vista pedagógico, es mejor contestar a esas preguntas en el momento, en lugar de posponerlas para la siguiente clase. Si el profesor tuviera la posibilidad de, no sólo variar los parámetros previstos, sino de modificar fácilmente la propia aplicación para adaptarla a las necesidades pedagógicas del momento, la experiencia educativa mejoraría aún más. Esta característica no suele estar disponible en las aplicaciones ya construidas.

- **Disponibilidad:** igual que las transparencias se suelen hacer llegar por uno u otro medio a los alumnos, estas aplicaciones se deberían también proporcionar como apoyo al estudio. Idealmente, los alumnos no deberían ver estas aplicaciones como meros "ejemplos ejecutables" de las transparencias, sino como "mesa de pruebas" que puedan modificar para *aprender haciendo* sus propias versiones de los programas. En realidad, esta característica engloba en cierta medida a las anteriores. Por ejemplo, para que el alumno pueda realizar sus propios experimentos es importante que las aplicaciones sean sencillas de entender y modificar.

3. Descripción de Nebula

Nebula es un motor 3D pensado para el desarrollo de juegos que se distribuye bajo licencia LGPL [1]. Su desarrollo lo mantiene RadonLabs [3], y sus creadores fueron el núcleo del equipo de Urban Assault.

Existen dos versiones distintas del motor, Nebula 1 y Nebula 2. Si bien Nebula 2 posee características avanzadas requeridas por cualquier juego de calidad hoy en día (programación de *shaders* por ejemplo), para cumplir nuestro objetivo es suficiente Nebula 1, cuya última liberación es de enero de 2004. Por lo tanto, en lo sucesivo, cuando hablemos de Nebula nos referiremos a la primera versión.

Nebula es en realidad un armazón (*framework*) escrito en C++ para el desarrollo de aplicaciones 3D de tiempo real. Bajo una fachada orientada a objetos, Nebula oculta el API gráfico subyacente, que puede ser OpenGL o DirectX según cómo se configure, permitiendo incluso saltar de uno a otro en tiempo de ejecución. Está disponible para diferentes plataformas y se distribuye con licencia de código abierto.

Hemos dicho que Nebula es un armazón (y no una librería) porque es Nebula quien posee el control de la ejecución de la aplicación. En realidad Nebula no es un simple motor gráfico, sino que se encarga de controlar la entrada del usuario, la reproducción del sonido, la creación de la ventana principal e incluso simplifica la detección de colisiones.

Una de las principales ventajas de Nebula es que la mayoría del API que proporciona es accesible tanto desde C++ como a través de guiones (*scripts*). En concreto, Nebula soporta acceso a la información del motor haciendo uso de Tcl/Tk o Python.

Como se ha dicho, Nebula está escrito en C++, y posee una arquitectura orientada a objetos bastante bien diseñada. Por ejemplo, la clase `nMeshNode` representa la información sobre geometría de una malla de triángulos, y `nShaderNode` contiene información sobre las características de sombreado que pueden aplicarse a algun objeto 3D.

Todos los objetos creados desde C++ son accesibles desde los guiones (*scripts*) y viceversa. Para ello, el núcleo de Nebula gestiona todos los objetos creados, y los organiza en lo que denomina un *sistema de ficheros virtual*, similar al creado sobre los dispositivos de almacenamiento por cualquier sistema operativo. La diferencia está en que los "directorios" representan en realidad *objetos de Nebula* y no tienen ningún tipo de relación con la persistencia en almacenamiento secundario. El sistema de ficheros virtual se utiliza por lo tanto como un modo de acceso a cualquier objeto existente en un determinado momento a partir de un nombre. Algunos objetos contienen a su vez a otros objetos como atributos. En ese caso, el directorio que representa al *objeto contenedor* posee *subdirectorios*, uno por cada objeto contenido. Por ejemplo, si existe el objeto cuyo nombre es `/usr/scene/lampara` y que representa el *objeto tridimensional* de una lámpara, existirán objetos del estilo de `/usr/scene/lampara/malla` que contendrá la malla (objeto de la clase `nMeshNode` comentada antes) y el objeto `/usr/scene/lampara/shader` con la información de sombreado (de la clase `nShaderNode`).

En general, las aplicaciones creadas con Nebula siguen un convenio en lo referente a los objetos creados y a sus nombres. Por ejemplo el directorio `/sys/servers` es un contenedor de objetos que simbolizan los diferentes servicios proporcionados por Nebula, como el gestor de tiempo o el detector de colisiones.

Esta estructura de directorios también se utiliza para almacenar en un árbol la escena que se dibuja. Normalmente, toda la escena se encuentra bajo `/usr/scene`, organizada de forma jerárquica. Muchos de los objetos (directorios) que posee se limitan a contener información sobre transformaciones (objetos de la clase `n3dnode`), creándose así una jerarquía de sistemas de referencia de forma que los cambios realizados sobre la posición y orientación de un directorio afectan a todos los objetos que cuelgan de él.

Cuando se crea un objeto Nebula desde el programa en C++, se le coloca en algún lugar de la jerarquía, para asignarle un nombre. Independientemente de que el objeto pueda manipularse a través de la variable de C++ recién declarada, el núcleo de Nebula permitirá a partir de ese momento recuperar el objeto a partir de su posición en el sistema de ficheros virtual. Aunque todo esto pueda resultar inicialmente un tanto esotérico, su razón de ser estriba en que el sistema de ficheros virtual sirve de puente entre C++ y los lenguajes de *script*.

Quizá la característica más destacable que hace de Nebula un motor aconsejable para el desarrollo de aplicaciones de apoyo a las clases de Informática

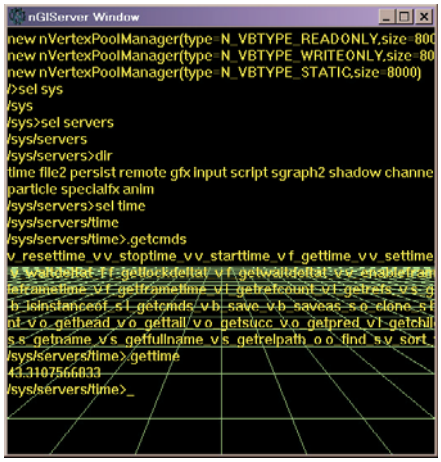

Figura 1: gfxserv con la consola activada

Gráfica es la disponibilidad de una consola (ver Figura 1). Durante la ejecución de una aplicación escrita sobre Nebula, se puede pulsar la tecla *escape*. En ese momento aparece en la ventana una consola que recuerda a los intérpretes de comandos de MS-DOS o Unix. En esta consola, existe la idea de "directorio actual", pero navegando sobre el sistema de ficheros virtual creado por Nebula. En este sentido, están disponibles las ordenes `dir` para listar su contenido y `sel` que se utiliza para cambiar el directorio. Además, dado que todos los directorios representan objetos de C++, desde la propia consola se le pueden enviar mensajes[1].

Los mensajes posibles dependerán del tipo de objeto que reciba el mensaje, y pueden listarse con el comando `.getcmds`. Por ejemplo, si nos encontramos sobre un objeto `n3dnode` (que representan sistemas de referencia) se puede escribir `.txyz 10 10 10`, para trasladar el sistema de referencia a esa posición, relativa al sistema del padre. Las ordenes enviadas se ejecutan inmediatamente y su resultado puede verse al instante. Las órdenes disponibles desde los lenguajes de *script* son casi siempre las mismas que los métodos accesibles a través de los objetos de C++, por lo que la decisión de si usar uno u otro se debe más a cuestiones de Ingeniería del Software que a posibles limitaciones existentes.

[1] De esta forma, el directorio actual puede verse como el puntero *this* de C++

A parte de "navegar" por los objetos, en todo momento se pueden introducir comandos del lenguaje de *script* que se esté utilizando. Nosotros hemos hecho uso de Tcl, por lo que podemos hacer uso desde simples `puts "Hola mundo"` hasta asignación de variables Tcl y su uso en los mensajes enviados a los objetos de Nebula.

4. Ejecutable principal

Escribir una aplicación desde cero en Nebula puede ser al principio un poco complicado. Sin embargo, en la distribución se incluyen dos programas, `nsh` y `gfxserv` que facilitan dicha tarea. El primero de ellos arranca una aplicación de consola (sin ventana gráfica) en el que únicamente se tiene accesible, desde una ventana de texto, una consola Tcl ampliada con los comandos que soporta Nebula (entre ellos crear una ventana donde se presenta la escena). El segundo, `gfxserv`, crea automáticamente una ventana gráfica en la que presenta una malla en el plano XY (desactivable a través de parámetros del ejecutable), y permite mover la cámara utilizando el ratón (ver Figura 1).

La principal ventaja de estos dos programas es su capacidad de ejecutar ficheros de *script* de inicialización que crean la escena que se quiere mostrar. De esta forma, el profesor no tiene que escribir código en C++; es suficiente con que cree objetos Nebula (como `nMeshNode` o `nShaderNode`) desde Tcl, organizándolos de manera adecuada para construir la escena. Únicamente con esto, tendrá disponible una aplicación que muestra esos objetos y permite desplazarse por el entorno para verlos desde cualquier perspectiva.

Por otro lado, dado que el lenguaje de script es Tcl, que dispone de la librería Tk, el profesor puede crear ventanas con botones o barras de desplazamiento para cambiar fácilmente en tiempo de ejecución los parámetros que quiera explicar.

La disponibilidad en tiempo de ejecución de la consola añade una ventaja adicional. Si el profesor es lo suficientemente diestro con los comandos que admiten los objetos Nebula, puede manipular el estado de la escena directamente en tiempo de ejecución para responder a preguntas de los alumnos no previstas con antelación y para las cuales no se han añadido controles en las ventanas creadas con Tk.

Nebula viene además con una serie de demostra-

ciones que crean escenas simples utilizando *scripts*, y que pueden utilizarse directamente como ejemplos en clase.

5. Ejemplos de uso

Para la implementación de los ejemplos, hemos utilizado el programa de Nebula `nsh`, con un subconjunto de los *scripts*, mallas y texturas que componen las demostraciones que se distribuyen con el motor.

De esta forma, hemos aprovechado las escenas o geometría mostradas en los ejemplos, acondicionándolas a las características que queremos mostrar en cada ejemplo. En todos ellos, los conceptos mostrados se explican cambiando ciertos parámetros, que son fácilmente modificados desde la consola, pero que, por mayor comodidad, han sido asociados a controles que aparecen en una ventana adicional creada con Tk.

Todos los ejemplos pueden descargarse de http://gaia.sip.ucm.es/people/marcoa/publications/jenui05.

5.1. Jerarquía de la escena

En la introducción hacíamos referencia a la explicación de los sistemas de referencia "anidados" en los que un sistema de coordenadas está subordinado a otro. Como complemento a su exposición, y basándonos en un ejemplo distribuido con Nebula, la aplicación muestra una escena con un sistema planetario, donde el sistema de referencia de los planetas depende del sistema de referencia de la estrella, y los sistemas de referencia de los satélites dependen a su vez del sistema de cada uno de sus planetas.

Una vez creada esa estructura, Nebula permite modificar, en tiempo de ejecución, la rotación de cualquiera de los sistemas de referencia desde la consola con un simple `.ry <angulo>`, lo que repercute en todos los sistemas de referencia subordinados. Para que la aplicación de ejemplo sea más cómoda de usar, se han añadido mediante Tk controles que permiten modificar la rotación de algunos de los sistemas de referencia. La Figura 2 muestra la aplicación en ejecución.

Para que sirva como guía, la aplicación completa ha requerido escribir 274 líneas de código Tcl/Tk. De ellas, 62 son en realidad un "esqueleto" de inicialización de Nebula y Tk que se reaprovecha en todos

Figura 2: Ejemplo de jerarquía de sistemas de referencia

los demás ejemplos; 96 se encargan de la creación de la escena y están inspiradas (prácticamente copiadas) de un ejemplo proporcionado con Nebula; 53 son de creación de la ventana mediante Tk; y por último 40 son las que realmente se encargan de la gestión de eventos de los controles (de las cuales únicamente 7 son llamadas a Nebula para cambio de parámetros, una por cada deslizador en la ventana de controles). Por tanto, de las 274 líneas sólo 40 son específicas de la aplicación; las demás pueden en muchos casos servir como guía para la creación de más ejemplos. En la página web antes referida se pueden ver tablas más detalladas sobre el número de líneas de código requeridos por este ejemplo y los dos siguientes.

5.2. Planos cercano y lejano

Es difícil explicar que existe un límite de cercanía y lejanía en lo que se puede mostrar en una escena, y sobre todo los resultados de modificar esos parámetros. Una vez explicado el concepto de campo de visión y la pirámide truncada (*frustum*) que simboliza el espacio visible, resulta ilustrativo mostrar la ejecución de una aplicación donde esos planos cercano y lejano puedan modificarse.

En Nebula, el cambio puede hacerse llamando al método `setperspective` del servidor gráfico localizado en `/sys/servers/gfx`. Hemos desarrollado un ejemplo sobre Nebula también usando Tcl/Tk con el que se pueden modificar los parámetros importantes de los planos mediante controles deslizantes y que permite comprobar los efectos de sus modificaciones.

5.3. Diferencia entre "zoom" y acercar la cámara

También al enseñar el campo de visión se trata con el ángulo de apertura de los planos laterales y superior e inferior. En muchos casos a los alumnos les cuesta ver que esos parámetros son los que equivalen al "zoom" de la cámara. En el transcurso del curso de verano, provocó la pregunta de un alumno de la diferencia entre cambiar ese ángulo de apertura y mover la cámara. Aunque en principio esa pregunta es más bien sobre cámaras reales y no sobre Informática Gráfica, la duda la resolvimos utilizando la consola de Nebula para mover la cámara.

Como consideramos que es una aclaración importante, a posteriori hemos desarrollado también una aplicación en Nebula que sirve para mostrar las diferencias utilizando controles de Tk para modificar los parámetros importantes.

6. Otros recursos

Si bien el uso de aplicaciones específicas como complemento a las explicaciones con pizarra y transparencias es útil, requiere un esfuerzo adicional por parte del profesor, que tiene que dedicar tiempo a su implementación. En este artículo proponemos el uso de Nebula para su desarrollo, pues facilita su creación mediante lenguajes de *script*, a parte de permitir la modificación en tiempo de ejecución. No obstante, para ciertos conceptos existen ya algunos programas disponibles que pueden servir como apoyo para las clases y que pueden utilizarse directamente en lugar de tener que desarrollar ejemplos específicos usando Nebula.

Existen por ejemplo, un conjunto de aplicaciones desarrolladas por Nate Robins [2] útiles para explicar los conceptos de proyección, niebla, materiales y proyección de texturas, entre otros. En la mayoría de estos programas, la ventana aparece dividida en tres partes, la primera de ellas con la imagen generada por el motor, la segunda con la imagen desde un punto de vista externo a la escena desde el que se alcanza a ver la cámara y en el que se muestra con líneas el campo de visión, y una tercera parte en la que se pueden cambiar diversos parámetros. La Figura 3 es una captura del tutorial que muestra el efecto de una luz direccional sobre un objeto, y que permite el cambio de la posición tanto de la luz como de la cámara. Estos tutoriales tienen un cometido similar a las aplicaciones que hemos propuesto creadas con Nebula, aunque difieren en un punto importante. Están pensados para enseñar el uso de llamadas a OpenGL, como `glRotate` o `glLight`, por lo que la aplicación muestra en la ventana explícitamente estas llamadas, y los parámetros que se cambian durante la ejecución son los parámetros de las invocaciones. Por esta razón, si el programa de la asignatura de Informática Gráfica está dividido en una primera parte de conceptos, y una segunda parte de programación con un API gráfico como OpenGL, se deberán utilizar con cuidado en la primera parte del curso, sin poder entrar en los detalles de las llamadas que la ventana muestra.

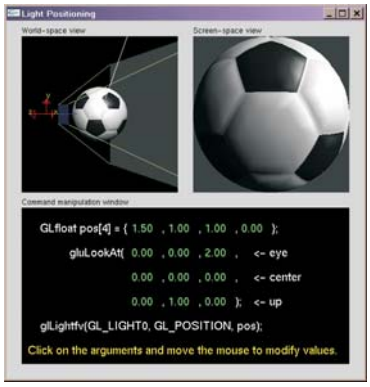

Figura 3: Tutorial de Nate Robins sobre luces [2]

Tanto las aplicaciones desarrolladas con Nebula como las de Nate Robins son sin embargo aplicaciones "de juguete", desarrolladas expresamente para una explicación. Como contrapartida, el uso en ciertas ocasiones de aplicaciones reales puede ser altamente motivamente. En el transcurso del Curso de Verano que impartimos, mostramos a los alumnos en diversas ocasiones juegos de ordenador en los que se hace uso de técnicas concretas, para que apreciaran el resultado final de éstas en el contexto de una aplicación real. Se pueden aprovechar los juegos como ejemplo para explicar cosas como:

- Uso de *sprites*: para mostrar el nivel de vida, munición o puntuación en casi cualquier videojuego.
- Billboards o modelos alineados con la cámara: como por ejemplo en *Rayman 3: Hoodlum Havoc* donde aparecen diamantes que el jugador

Figura 4: Juego *Terminator 3* donde se aprecian los distintos niveles de detalle de la valla.

debe recoger, y que se muestran siempre igual, independientemente de la posición de la cámara.

- Tipos de cámara: por ejemplo en *El Príncipe de Persia: las arenas del tiempo* donde dependiendo del nivel, la cámara sigue al usuario, o está fija en un punto y gira para manternelo dentro de su campo de visión.

- Niveles de detalle: un buen ejemplo es el juego *Terminator 3*, ya que en algunos momentos se aprecia que según se va moviendo el jugador va apareciendo nueva geometría. En la Figura 4 se ve que la valla está construida utilizando distintas representaciones dependiendo de la distancia.

- Mapas de luces o suma de texturas: como por ejemplo en el Half-Life 1, donde las sombras que proyectan los edificios están hechas con texturas que se suman en tiempo de ejecución a las del suelo.

No obstante, hacer uso de juegos en clase (ejecutarlos delante de los alumnos) es una decisión que no se puede tomar a la ligera. Hoy en día los juegos son aplicaciones muy complejas que requieren ordenadores bastante más sofisticados que los necesarios para mostrar presentaciones, por lo que su ejecución podría ser, directamente, inviable. Un problema adicional es que desde que el programa se ejecuta hasta que se llega al punto que se deseaba enseñar, a menudo hay que atravesar varios menús y recorrer una pequeña parte del juego. Por último, comparándolas con las aplicaciones "de juguete" comentadas antes no pueden dejarse disponibles a los alumnos por tratarse de software comercial. No obstante, hacer uso de capturas de pantalla de los juegos en transparencias puede ser una solución intermedia bastante adecuada, que a menudo motiva enormemente a los alumnos.

7. Conclusiones

En este artículo hemos planteado el uso de aplicaciones de ejemplo para facilitar las explicaciones de Informática Gráfica que, tradicionalmente, se realizaban únicamente a base de pizarra y transparencias.

Si bien el uso de este tipo de aplicaciones de apoyo parece bastante prometedor, la carga adicional que impone su desarrollo sobre el a menudo sobrecargado trabajo del profesor hace que se queden más como buen propósito que como realidad.

Como posible solución, hemos planteado el uso de un armazón de desarrollo de juegos llamado Nebula como punto de partida para la creación de esas aplicaciones. En concreto, sus puntos fuertes para este uso son la disponibilidad de acceso a los objetos a través de un lenguaje de *script* y la existencia de una "consola" en tiempo de ejecución para poder modificar el estado de la aplicación en cualquier momento. No obstante, al ser el lenguaje de *script* Tcl/Tk, con un mínimo esfuerzo es posible mostrar ventanas con controles que permitan modificar los parámetros de la aplicación en tiempo de ejecución sin tener que recurrir a los, a veces un tanto crípticos, comandos de la consola.

Por último, se han descrito algunas aplicaciones de ejemplo (varias desarrolladas con Nebula) que pueden utilizarse como apoyo a las clases de Informática Gráfica.

Aunque no hemos llevado a cabo ningún tipo de investigación experimental comparando el aprendizaje de algún grupo de control frente a un grupo experimental, sí hemos probado a utilizar algunos de los tutoriales y juegos aquí descritos en el Curso de Programación de Videojuegos impartido en la Escuela Complutense de Verano en Madrid en Julio de 2004, para explicar conceptos como los planos cercanos y lejanos o la gestión de la cámara.

Referencias

[1] GNU Lesser General Public License, *http://www.gnu.org/copyleft/lesser.html*

[2] Nate Robins, OpenGL Tutors, *http://www.xmission.com/~nate/tutors.html*

[3] Nebula Device, de Radon Labs GmbH Game Development (2004), *http://www.radonlabs.de/*