ANÁLISIS COMPARATIVO DE LOS APIS GRÁFICOS

*César Julio Bustacara Medina**

*David Alejandro Uribe Pardo***

Resumen: las aplicaciones computacionales han mostrado una tendencia incremental hacia una interfaz gráfica y amigable. El problema de las aplicaciones gráficas es la exigencia de memoria y procesamiento, que se debe a la cantidad de cálculos que deben realizar. Los *APIs* son librerías que ofrecen a las aplicaciones un contacto directo con los drivers de dispositivos al mismo nivel del sistema operativo. En el ámbito gráfico, los *APIs* más importantes son *DirectX y OpenGL*; este artículo los analiza en cuanto a su potencialidad, rendimiento, y, en general, características importantes que influyen al tomar decisiones respecto a cada uno. Estas conclusiones fueron tomadas de una investigación en la Pontificia Universidad Javeriana, cuyo fin es ofrecer en detalle cómo y cuándo usar cada uno de los *APIs*.

Abstract: computer applications have shown an increasing tendency towards a graphical and friendly interface. The problem with graphical applications is the demand for memory and processing given the amount of calculations that it must do. The *APIs* are libraries that offer the applications direct bonding with device drivers at the same level as the operating system. In the graphical environment, the most important *APIs* are *DirectX* and *OpenGL*. This article analyzes each one with regards to its potentiality, performance, and, in general, its important characteristics which influence decision making concerning either one. These conclusions were taken from a research project carried out at the Pontificia Universidad Javeriana, whose aim was to offer a detailed view of how and when to use each *API*.

1. INTRODUCCIÓN

Gracias a la permanente evolución del *hardware* se han diseñado librerías de programación especiales para el desarrollo de aplicacio-

* *Ingeniero de Sistemas, Universidad Nacional de Colombia, Magíster en Ingeniería de Sistemas, Universidad de los Andes, Profesor Instructor del Departamento de Ingeniería de Sistemas de la Pontificia Universidad Javeriana.*

** *Ingeniero de Sistemas, Pontificia Universidad Javeriana.*

nes multimedia y de computación gráfica, que permiten al programador acceder al *hardware* a bajo nivel. Estas se conocen como *APIs* gráficos y en este artículo se mostrarán los resultados de un análisis comparativo de los dos más importantes: *DirectX y OpenGL.*

Un *API (Application Programming Interfaces)* es una librería compuesta de muchas funciones relacionadas con un tema específico, para que el desarrollador de *software* las utilice y así conformar un programa de computador completo.

En el caso específico de la computación gráfica, los *APIs* proveen funciones para dibujar objetos, simular efectos de luces, determinar qué objetos son visibles desde un punto de vista dado, etc. Un buen *API* gráfico tridimensional, debe ofrecer facilidad y alto desempeño.

Los *APIs* se pueden entender como una interfase entre el *hardware* y la aplicación. Para el desarrollador son transparentes los cálculos y el dibujado que el *hardware* realiza. Por ejemplo, si se tiene instalada una tarjeta aceleradora 3D, y se va a dibujar un objeto tridimensional complejo, el propio *API* se encarga de ejecutar el dibujado de la manera más óptima sobre la tarjeta; en caso de no estar disponible la función, se ejecuta por medio de *software* de emulación (a un nivel más alto y con menos desempeño). Por esta razón, no es necesario programar específicamente para cada una de las tarjetas 3D o de video, ya que el propio *API* se encarga de la compatibilidad.

Un *driver de dispositivo* es un elemento de *software* que implementa un *API* en un *hardware* de dispositivo dado; por ejemplo, para cada tarjeta de video se necesita un *driver* teóricamente diferente. Las funciones del *API* gráfico se implementan de diferente manera por cada tarjeta de video, por ello, los fabricantes de estas tarjetas deben ofrecer sus propios *drivers*, para que los *APIs* tengan independencia del dispositivo, es decir, que sirvan con cualquier tarjeta de video. No puede haber un *API* de alto desempeño y confiabilidad, si no se tiene un buen dispositivo asociado; por ello, para tener una buena aplicación gráfica se necesita de la calidad de ambos.

2. Antecedentes de los APIs gráficos

El desarrollo de *OpenGL* determinó un gran avance de los *APIs*, teniendo una gran gama de plataformas sobre las cuales implementarse eficientemente. *Microsoft* y *Silicon Graphics* se unieron para desarrollar una implementación de *OpenGL* para *Windows NT*. Un aspecto importante de esta implementación fue el diseño de un nuevo driver de dispositivo llamado *Installable Client-side Driver (ICD)*, el cual ofrecía alto desempeño gráfico y permitía que cualquier fabricante de *hardware* gráfico se extendiera al *API* de *OpenGL* para soportar su manejo tridimensional.

Más tarde, el *hardware* para gráficos tridimensionales se vuelve más accesible; *Microsoft* desarrolla otro *driver* de dispositivo llamado *Mini Client Driver (MCD)*, el cual redujo notablemente el tiempo requerido

para producir implementaciones de calidad en *OpenGL*; por otro lado, soportaba una gran gama de tarjetas de video; así, la accesibilidad de *OpenGL* creció exponencialmente.

Entre 1995 y 1996, *Microsoft* decide no utilizar la tecnología de *OpenGL* ya empleada en *Windows NT*; en vez de ello compra a *Rendermorphics Ltd*, y, por ende, adquiere su *API* gráfico tridimensional llamado *RealityLab*. *Microsoft* replantea el diseño del *driver* de dispositivo para *RealityLab* y lo anuncia como *Direct3D Immediate-Mode de DirectX.*[1]

En 1996 *Silicon Graphics* decide montar una demostración en la conferencia *Special Interest Group on Computer Graphics (SIGGRAPH)* en New Orleans. Esta demostración evidenció que *OpenGL* era por lo menos tan rápido como *DirectX*. Así mismo, desencadenó un fuerte debate en la computación gráfica y el desarrollo de videojuegos.

Microsoft continúa actualizando a *Direct3D*, y con cada nueva versión incorpora más aspectos de *OpenGL*. No obstante, *Direct3D* es mucho más eficiente de lo que era hace unos años y su evolución apenas comienza.

3. Arquitectura de los APIs gráficos

Los sistemas operativos *Windows 95/98* poseen un subsistema encargado de representar gráficos, tanto en el sistema de video como en la impresora; este subsistema se conoce como *Graphics Device Interface (GDI)*. Al igual que todos los *APIs de Windows*, se utilizan en ella *Dynamic Link Libraries (DLL)*, de las que se importan las funciones que se requieran en tiempo de ejecución; en particular, en *Windows 95/98* se utiliza la librería GDI32.DLL. Después de haber definido la librería se requiere especificar el dispositivo en que se desea dibujar (tarjeta de video, impresora, etc.); para ello se utiliza una interfase o un contexto de dispositivo, en el cual se efectúan todas las gráficas primitivas que el *GDI* posee.

Con el *GDI* se puede utilizar la filosofía de transferencia de bloques de *bits*, o transferencia de *buffer* de despliegue, donde se puede tener un contexto de dispositivo de un bitmap, en memoria de video, para trabajar sobre él y luego transferirlo al contexto de dispositivo del *hardware*, con el fin de optimizar el desempeño, ya que es más óptimo trabajar sobre memoria de video que sobre el mismo *hardware*.

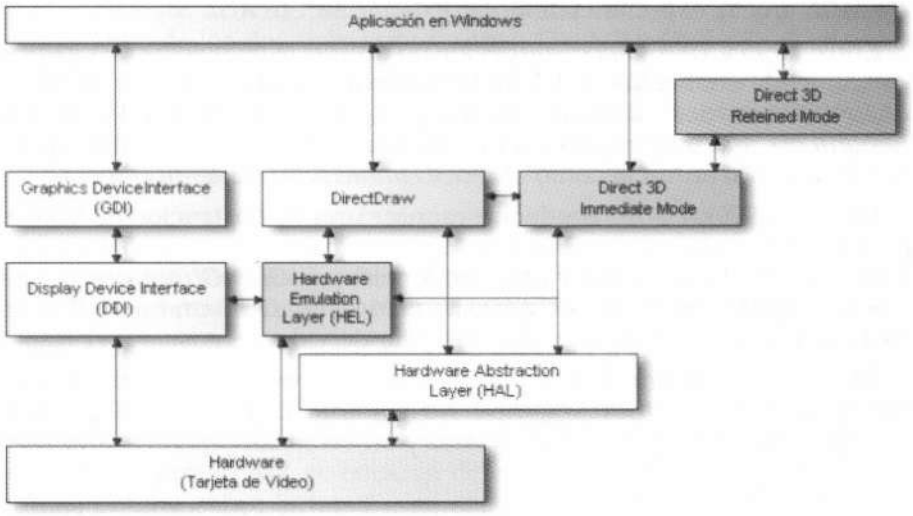
3.1. La arquitectura de *DirectX*

En el ámbito de la computación gráfica, *DirectX* utiliza dos componentes básicos, el *DirectDraw y el Direct3D* (véase figura 1).

---

1 Cabe aclarar que la verdadera competencia se da entre OpenGL y Direct3D, ya que DirectX es un API mucho más amplio que adicionalmente ofrece DirectDraw (gráficas en 2D), DirectInput (dispositivos de entrada), DirectPlay (soporte para redes), DirectSound (sonido) y DirectMusic (música).

FIGURA 1. Arquitectura de *DirectX*.



El *DirectDraw* se encarga de acelerar las técnicas de animación en *hardware* y *software*, ofreciendo acceso directo a *bitmaps* y memorias de despliegue, dando gran rapidez al intercambiar el *buffer*. El *Direct3D* provee una interfaz de alto nivel llamada *Retained Mode*, la cual ofrece un manejo fácil de gráficas tridimensionales. Además, provee una interfase a bajo nivel llamada *Immediate Mode*, en la que se puede tener un control total del *renderizado*.

DirectDraw ofrece una independencia del dispositivo a través del *hardware abstraction layer (HAL)*. El *HAL* es una interfase específica del dispositivo, ofrecido por sus fabricantes para que *DirectDraw* trabaje directamente con el *hardware* del dispositivo. El *HAL* puede ser parte de un *driver* de dispositivo o de una *DLL* que se comunica con el *driver* del dispositivo a través de una interfase privada que los creadores del *driver* definen, ya que las aplicaciones nunca trabajan directamente con el *HAL*.

Cuando el *hardware* no soporta una función, *DirectDraw* intenta emularlo. Esta funcionalidad de emulación se hace a través del *hardware emulation layer (HEL)*. El *HEL* presenta sus capacidades a *DirectDraw* así como el *HAL* y las aplicaciones nunca trabajan directamente con el *HEL*. Por razones obvias, con el *HEL* no se tendrá el mismo desempeño que con el *HAL*.

Direct3D y *DirectDraw* son complementarios, y además existe una interfase entre ellos, permitiendo que *Direct3D* pueda aprovechar las superficies de *DirectDraw* (*buffer*) para dibujar sobre ellos.

DirectX está compuesto de objetos e interfases basadas en el *Component Object Model (COM)*. *COM* es una especificación orientada a objetos que ofrece una robusta interoperabilidad entre objetos a través de rutinas predefinidas llamadas interfases (estas interfases se pue-

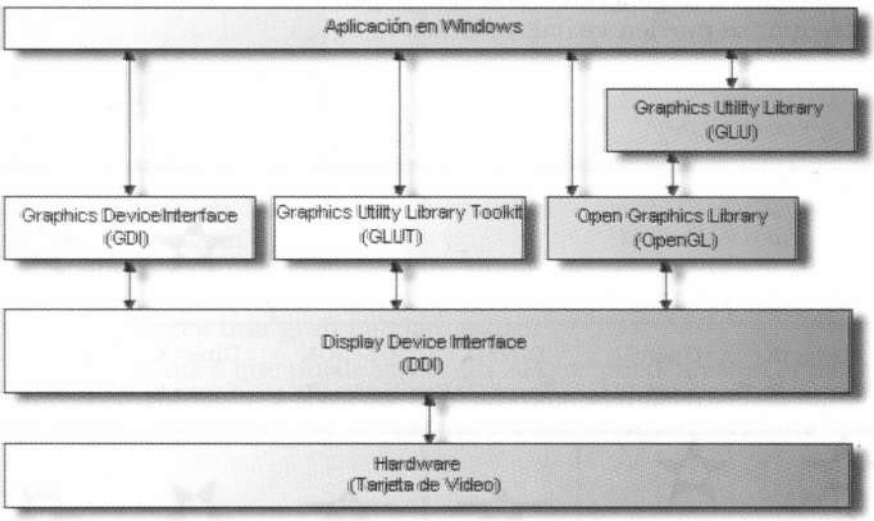
den entender como un contrato de servicios proporcionados por un componente; por esta razón, una interfase consta de los prototipos de los métodos de dicho componente).

En *DirectX* la jerarquía es muy sencilla, se basa en tres objetos que representan sus respectivos dispositivos: *IDirectDraw, IDirectSound, e IDirectPlay*. Estos objetos se derivan directamente de *Iunknown*, que es la interfase que tienen los objetos *COM* por defecto.

3.2 La arquitectura de *OPENGL*

En *OpenGL* existen oficialmente tres librerías básicas donde se encuentra todo el manejo gráfico necesario para producir aplicaciones de gran calidad (véase figura 2). Estas librerías son en primer lugar *Graphics Library* (GL), que constituye el corazón de *OpenGL*, ya que aquí está toda la geometría básica con la que se definen los objetos, sus características, transformaciones, proyecciones y *renderizado*. Está también la *Graphics Library Utility (GLU)* que es un conjunto de rutinas que sólo contienen funciones de *GL*, pero que facilitan mucho el manejo de *OpenGL*. Entre sus funciones más destacadas está el tratamiento de curvas, dibujo de primitivas y proyecciones. Finalmente se encuentra la *Graphics Library Utility Toolkit (GLUT)*, que es la librería enfocada al manejo de la interactividad y a la administración de la aplicación. Entre sus funcionalidades está el manejo de ventanas, menús, entradas de dispositivo, eventos, etc.

Figura 2. Arquitectura de *OpenGL*.



A diferencia de *DirectX*, que sólo funciona en sistemas operativos bajo *Windows*, *OpenGL* soporta plataformas como *Mac OS, OS/2, UNIX, Windows 95/98, Windows NT, Linux, OPENStep, Python, y BeOS*. Ade-

más, trabaja con los grandes sistemas de ventanas tales como *Presentation Manager, Win32 y X/Window System. OpenGL* puede ser invocado por *Ada, C, C++, Fortran, Java y Pascal* (aunque sólo en Windows), adicionalmente ofrece completa independencia de protocolos de red y topologías. A pesar de esta inmensa independencia, la única forma de comparar *OpenGL* con *DirectX* es a través de *Windows*.

4. Pruebas, medidas y parámetros de comparación

La herramienta de desarrollo que se escogió para la realización de las pruebas de comparación fue *Borland Delphi 3.0* por las grandes ventajas que ofrece, entre las que se encuentran la facilidad en el manejo de ventanas, y la más importante, que utiliza el lenguaje *Pascal*. Por su rigidez, este lenguaje permite comparar de una manera más objetiva diferentes programas fuentes; así mismo, su compilación es más ágil y menos compleja.

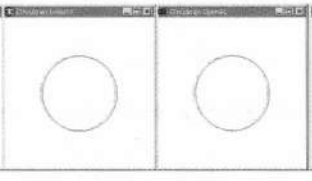
Las librerías del software *Development Kit (SDK)*, tanto de *DirectX 6.1* como de *OpenGL 1.2*, están hechas en lenguaje *C++;* por ello fue necesario buscar las librerías de los *SDK* para *Delphi*; una dirección en *Internet* que ofrece estos y muchos otros ejemplos de muestra es http://www.delphi-jedi.org.

En cuanto al equipo en el que se realizaron las mediciones, fue un microcomputador con procesador *Pentium* de 100 Mhz, con 32 *Mb* de memoria *RAM* y 2 *Mb* de video. Las pruebas de rendimiento se hicieron de una manera evolutiva, es decir, comenzando desde lo básico, por ejemplo, el trazado de una línea, hasta una animación interactiva. Los despliegues que generaron *DirectX* y *OpenGL* respectivamente en cada categoría, se pueden visualizar desde la figura 3 hasta la figura 18.

Figura 3. Línea

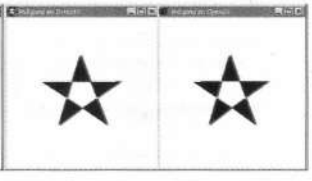
DirectX OpenGL

Figura 4. Círculo



DirectX OpenGL

Figura 5. Polígono 2D

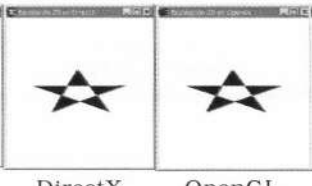


DirectX OpenGL

Figura 6. Traslación 2D

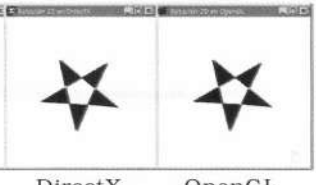
DirectX OpenGL

Figura 7. Escalación 2D



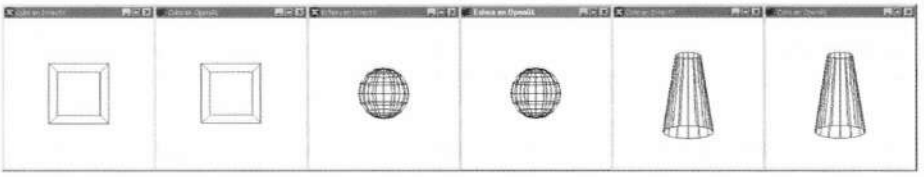
DirectX OpenGL

Figura 8. Rotación 2D



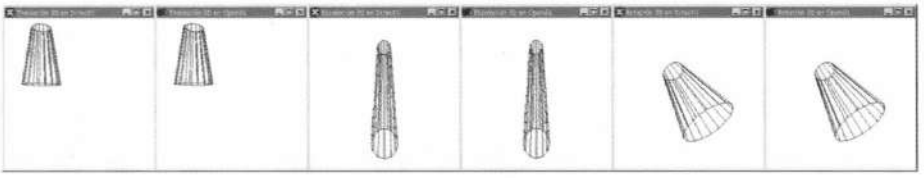
DirectX OpenGL

FIGURA 9. Cubo FIGURA 10. Esfera FIGURA 11. Cono



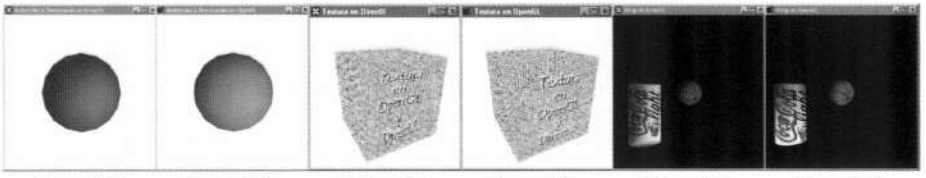
DirectX OpenGL DirectX OpenGL DirectX OpenGL

FIGURA 12. Traslación 3D FIGURA 13. Escalación 3D FIGURA 14. Rotación 3D



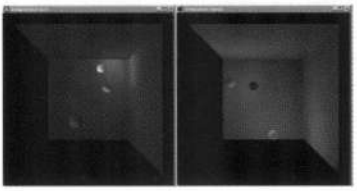
DirectX OpenGL DirectX OpenGL DirectX OpenGL

FIGURA 15. Materiales e Iluminación FIGURA 16. Textura FIGURA 17. Envolventes (Wrap)



DirectX OpenGL DirectX OpenGL DirectX OpenGL

FIGURA 18. Animación



DirectX OpenGL

Los amantes de la programación incremental y por componentes, pueden percibir en *DirectX* una gran ventaja sobre *OpenGL*, ya que ofrece un paradigma de orientación por objetos, el cual hace que tanto el *API* como sus aplicaciones sean altamente sostenibles y de fácil abstracción. Sin embargo, la calidad de *OpenGL*, como se aprecia en las figuras, ofrece una gran calidad gráfica.

Las necesidades que puede suplir un *API*, se pueden estereotipar en cuanto al programador y el cliente final tal como se muestra en la tabla 1.

TABLA 1. Necesidades que puede suplir un API

Necesidad	Explicación
Plataforma	Para que una aplicación sea aceptada por mercados amplios, se debe tener en cuenta la portabilidad y disponibilidad en las diferentes plataformas o sistemas operativos que dichas aplicaciones ofrezcan.
Drivers de dispositivo	Como programador se debe utilizar un *API* que pueda ser soportado por un número considerable de dispositivos, para así ampliar el mercado.
Aprendizaje y facilidad	Si como programador no se conoce ninguno de los *APIs*, es necesario aprender de una manera ágil su funcionamiento, para ofrecer más efectividad en el desarrollo y un mayor cubrimiento en el soporte de las aplicaciones.
Líneas de código	Un programa que menos líneas de código requiera, implica una mayor agilidad en el desarrollo de los proyectos. Este factor es muy útil para medir la duración de los proyectos.
Archivos generados en compilación sin ejecutable	El espacio que ocupen en disco las fuentes (incluyendo los archivos generados en compilación sin ejecutable) determinan su portabilidad y facilidad para efectuar copias de seguridad.
Ejecutables	Los ejecutables representan el producto final del desarrollo de un proyecto gráfico y ello incluye el *entregable* al usuario final. En este orden de ideas un ejecutable de corta extensión ofrece comodidad a los clientes.
Velocidad	Las aplicaciones en general deben ser muy rápidas, más si se trata de aplicaciones gráficas que realizan innumerables cálculos.
Memoria	Las aplicaciones gráficas (en especial las vectoriales) manejan mucha información para representar los objetos que hacen parte de una escena a dibujar. Toda esta información se maneja en memoria principal y de los *APIs* depende en gran medida su utilización.
Procesador	Cuando se va a desplegar una escena en una aplicación gráfica, los *APIs* deben hacer ciertos cálculos para determinar aspectos tales como transformaciones, proyecciones, iluminación, etc. La utilización del procesador depende del modo en que los *APIs* desempeñan estos cálculos.
Animación (cuadros por segundo)	La calidad de una aplicación gráfica animada depende casi en su totalidad del número de cuadros por segundo que se emiten para dar el efecto de animación; con muy pocos la animación perderá realismo. Este requerimiento es imprescindible en los juegos de video.
Calidad visual	Las aplicaciones gráficas deben ofrecer, además de buen desempeño, una gran calidad visual, sobre todo en los casos en los que se utilice la aplicación para diseño.

5. ANÁLISIS DE LOS RESULTADOS

Los parámetros de comparación anteriormente explicados fueron medidos a través de categorías evolutivas que engloban las necesidades básicas de las aplicaciones gráficas; las categorías empleadas fueron las siguientes:

- Aplicaciones en dos dimensiones: línea, círculo, polígono, traslación, escalación, rotación.
- Aplicaciones en tres dimensiones: cubo, esfera, cono, translación, escalación, rotación, materiales e iluminación, textura, envolventes de textura (*wrap*).

En cuanto a la animación, se realizaron diez pruebas por cada una de estas categorías y se midieron los parámetros de comparación que pueden variar de una prueba a otra. Estas pruebas se sintetizan porcentualmente en las figuras 19, 20 y 21.

FIGURA 19. Rendimiento porcentual en aplicaciones 2D

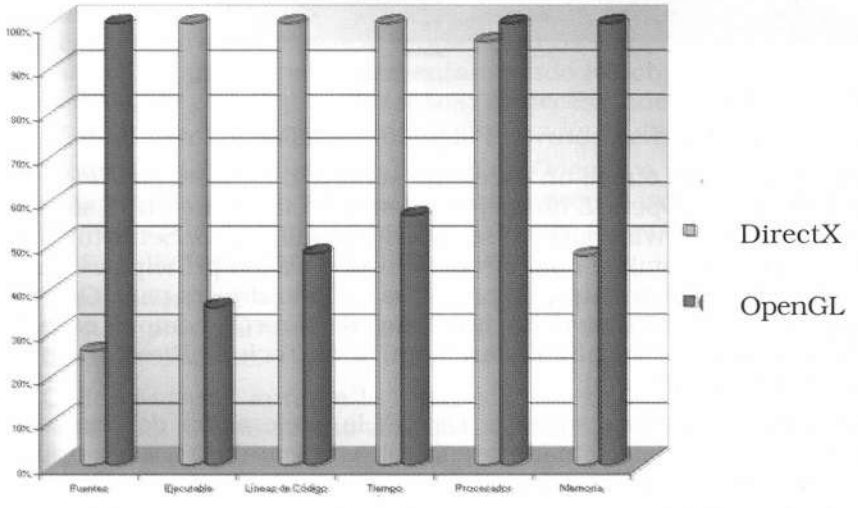


FIGURA 20. Rendimiento porcentual en aplicaciones 3D

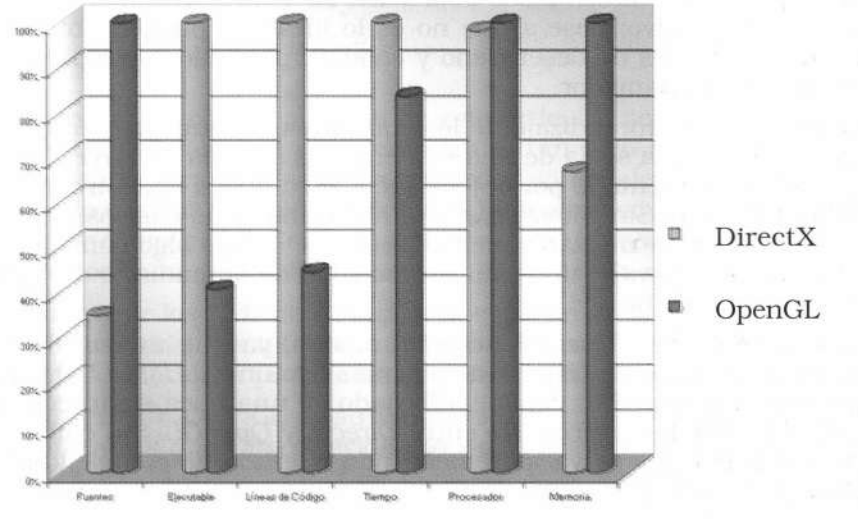
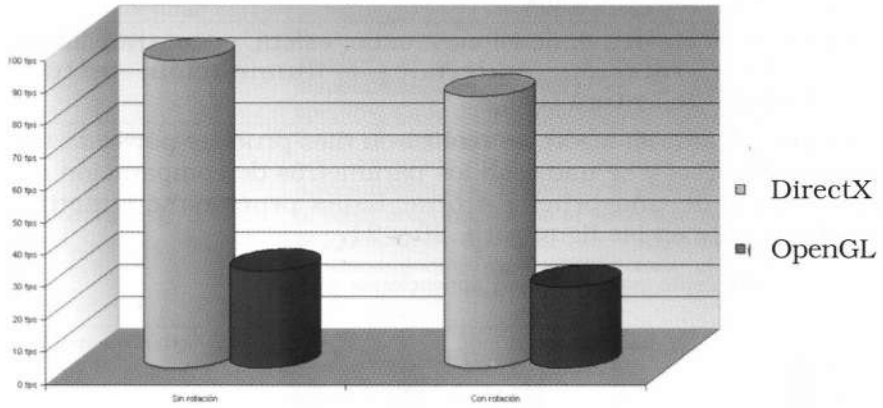


FIGURA 21. Frames por segundo en la categoría de animación



De estas pruebas se obtuvo el siguiente análisis de resultados:

- *Drivers: DirectX* ofrece un excelente soporte de *drivers en Windows*. Por su parte, *OpenGL* ha mejorado sustancialmente en el soporte de *drivers* para *Windows 9X*; se está desarrollando *GLSetup* (http://www.glsetup.com), el cual ofrece *drivers* para las principales tarjetas gráficas 3D. Incluso, se han desarrollado *drivers* para *OpenGL* que utilizan a bajo nivel las funciones de *DirectX*, como el caso de *Altogl* (http://www.altsoftware.com) y *SciTech GLDirect* (http://www.scitechsoft.com).
- Aceleración de *hardware*: en *DirectX* la aceleración de *hardware* para una tarea determinada se controla verificando la lista de *drivers* de dispositivo que posea el sistema para así poder seleccionar el que más se ajuste a las necesidades de la aplicación, mientras que con *OpenGL* se obtiene siempre el *driver* de dispositivo con mejores capacidades de manera automática. Con *DirectX*, tanto en *modo retenido* como en *modo inmediato*, se debe seleccionar el *driver* del dispositivo; pese a que no es lo ideal, se pueden obtener diferentes niveles de desempeño y calidad, pero ello es responsabilidad del programador.
- Aprendizaje: el aprendizaje de los conceptos involucrados en una aplicación gráfica se da de una manera más sencilla en *DirectX*, ya que, al estar orientado por objetos, se acerca más a la abstracción del mundo real. Sin embargo, en los algoritmos más básicos, como el de la línea, se utilizan muchas líneas de código algo complicadas. Por ello *OpenGL* ofrece mayor facilidad de aprendizaje del *API* propiamente dicho.
- Líneas de código: *OpenGL* necesita menos, ya que utiliza librerías que agrupan sentencias de uso más común (*GLU y GLUT*). En procesos tan sencillos como el dibujado de una línea se aprecian notablemente las diferencias entre *DirectX* y *OpenGL*. Sin embargo, a medida que se va incrementando el grado de complejidad, *DirectX* comienza a ofrecer nuevas funciones macro a un nivel

más alto en la arquitectura (*Retained Mode*). *OpenGL* utiliza aproximadamente un 64% menos de líneas de código de las que utiliza *DirectX*.

- Ejecutables: el tamaño de los archivos ejecutables en *OpenGL* es considerablemente menor que en *DirectX*, ocupando un 61% menos de espacio. Esta medida no tiene en cuenta las librerías que deben incluirse en los archivos que se entregan al usuario final.
- Archivos generados en compilación sin ejecutable: en este aspecto, la superioridad de *DirectX* es definitiva; *OpenGL* utiliza un 137% más de espacio en disco duro al crear sus archivos generados en compilación.
- Velocidad: la velocidad, representada en los milisegundos que requieren los *APIs* para inicializar sus variables y dibujar las imágenes, muestra que *OpenGL* requiere un 32% menos del tiempo que *DirectX*. El problema se presenta cuando los objetos a dibujar son demasiado complejos y extensos: se necesita de un gran apoyo de *drivers* que en muchas tarjetas aceleradoras gráficas no se da.
- Memoria: debido a la portabilidad de *OpenGL* a cualquier plataforma, se deben usar muchos más recursos de máquina para adaptarse a cada sistema operativo. Por ello *Open GL* utiliza un 63% más de memoria principal que *DirectX*.
- Procesador: por la misma razón anterior, *OpenGL* requiere un 3% más de procesamiento que *DirectX*. Esta diferencia no es, sin embargo, relevante.
- Calidad visual: aunque la diferencia no es significativa, se puede notar una mayor calidad visual en *OpenGL*, tal como se puede apreciar en las figuras 3 a 18.
- Animación: se presentan muchos inconvenientes en *OpenGL*, debido a su manejo de memoria y el poco soporte de *drivers*. Se requiere una sólida estructura de programación para hacerla lo más efectiva posible.

6. Conclusiones

Como se mencionó al inicio de este artículo, no se pretende definir cual *API* debe usarse unilateralmente, tan sólo se establecen unos parámetros para ayudar al programador en la selección de un *API* de acuerdo con sus necesidades (diseño gráfico industrial y mecánico asistido por computador o entretenimiento), ya que cada *API* tiene un campo de acción muy marcado dentro del ámbito de la computación gráfica.

Comparar los *APIs* de *DirectX* y *OpenGL* no es tarea fácil, ya que poseen características intrínsecas diferentes pese a que los dos están dirigidos a solucionar problemas en el desarrollo de aplicaciones gráficas.

En general, *OpenGL* mostró más tendencia hacia el campo del diseño, mientras que *DirectX* evidenció una inclinación hacia el entretenimiento. Esto se debe a que *DirectX* mostró más desempeño en animación que calidad gráfica, semejándose a los requerimientos de un juego; por

su parte, *OpenGL* mostró más calidad gráfica que desempeño en animación, asemejándose a los requerimientos de una herramienta de diseño de modelos.

En aplicaciones de gran escala, como un paseo virtual, se detecta la superioridad de *DirectX* sobre *OpenGL* en cuanto a la velocidad de renderizado; la animación de *OpenGL* se muestra mucho más pesada que *DirectX*, debido principalmente a su soporte de *drivers* y a la amplia cantidad de recursos de máquina que consume.

DirectX brinda muchas facilidades en el manejo de dispositivos, animaciones y detección de colisiones, las cuales son de vital importancia en una aplicación de paseo virtual. Sin embargo, con la implementación que se hizo de la librería ArchivosX.pas, el manejo en *OpenGL* se hace incluso más sencillo que en *DirectX*.

Referencias

Akin, Microsoft and 3D Graphics: "A Case Study in Suppressing Innovation and Competition", http://www.vcnet.com/, 1998.

Angel, E. *Interactive Computer Graphics: A top-down approach with OpenGL*. Reading, MA: Addison Wesley, 1997.

DirectX 6.1 Programmer's Reference, Microsoft Corporation, 1999.

Foley, J. D., *Computer Graphics: Principles and Practice*, Addison-Wesley, 1991.

Hearn, D. and Baker, M. P. *Computer Graphics*, Prentice-Hall, 1995.

Petzold, C., *Programming Windows 98*, Microsoft Press, 1998.

Silicon Graphics. *Guide to OpenGL on Windows*. 1996.

Woo, M., Neider, J. y Davis, T. *OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 1.1*. Reading, MA: Addison Wesley, 1997.