Desarrollo de un entorno de visualización 2D y 3D multiplataforma

Suárez Quirós, J.; Gayo Avello, D.; González Cobas, J.D.; García Díaz, R.P.; Álvarez Peñín, P.I.
Área de Expresión Gráfica en la Ingeniería – Grupo GIworks
Universidad de Oviedo
E.T.S.I.I.I.I.G., Campus de Viesques, s/n. Gijón – ASTURIAS
E-mail: quiros@etsiig.uniovi.es; Tfno.: 985 18 26 48; Fax: 985 18 22 40

1. Resumen

El vertiginoso desarrollo tecnológico acaecido durante las dos últimas décadas, especialmente en el campo de los gráficos por computador, ha traído como consecuencia la multiplicidad tanto de herramientas de programación empleadas en la creación de entornos gráficos de usuario (GUI: *Graphical User Interface*) como de plataformas informáticas sobre las que implementar las nuevas aplicaciones.

En estas circunstancias se hace palpable la necesidad de disponer de un entorno de visualización gráfico multiplataforma básico, dotado de una serie de especificaciones fundamentales que sea fácilmente adaptable al tipo de aplicación que se está desarrollando sin necesidad de partir de cero cada vez que se aborda el diseño de un proyecto de gráficos.

En el seno del grupo GIworks de la Universidad de Oviedo se ha llevado a cabo la implementación de un entorno como el descrito empleando la versatilidad que ofrecen hoy en día lenguajes interpretados como Tcl/Tk y librerías de visualización de alto nivel (VTK) desarrollando una estructura de *plug-in's* personalizables insertados sobre un *kernel* básico. Sus amplias prestaciones no están en modo alguno reñidas con la sencillez de programación, lo que proporciona una herramienta de desarrollo de elevada flexibilidad y gran eficacia.

2. Abstract

Fast technologic advances produced for last two decades, specially in computer graphics, have had as a result the growth in number of toolkits for graphical user interface development in addition to computer platforms on which new applications can be developed.

In this circumstances it is obvious the necessity for a multiplatform visualization environment that should provide basic facilities and be easily adaptable to the kind of application on development to avoid starting from scratch every time a computer graphics project begins.

GIworks workgroup at University of Oviedo has implemented such an environment thank to the versatility offered by script languages, like Tcl/Tk, and visualization libraries, like VTK, developing a structure based on customizable plug-in's embedded into a basic kernel. Its wide range of use does not mean complex programming so it provides a very flexible and high performance development tool.

3. Introducción

En la actualidad la demanda de gráficos por computador es enorme en áreas tan dispares como la ingeniería, la bioquímica, la medicina o la geografía, por citar tan sólo algunas. Las razones de tal demanda son muy variadas aunque se pueden citar las siguientes entre las más relevantes:

- Acceso a grandes volúmenes de datos de forma casi inmediata y con mínimo esfuerzo.
- Facilidad para extraer información inicialmente oculta o deteriorada.
- Oferta de nuevas formas de gestión de la información.

Tales circunstancias llevan a que sean muchos los proyectos relacionados con los gráficos por computador que se plantean; proyectos en los que, en gran medida, se repiten las circunstancias siguientes:

- Cada proyecto es distinto y requiere un enfoque diferente y, en general, totalmente nuevo.
- Todos los proyectos comparten una base común muy amplia e independiente del objetivo final.
- La mayor parte del esfuerzo se invierte en reimplementar dicha base común, desperdiciando el trabajo realizado anteriormente y afectando al desarrollo específico del proyecto.

Teniendo en cuenta todo lo anterior pueden establecerse dos supuestos:

1. El único esfuerzo justificable en un nuevo proyecto de gráficos es el necesario para el desarrollo de aquella parte del mismo que lo diferencia de otros proyectos haciéndolo único.
2. Aspectos tales como la carga de imágenes y/o geometría, manipulación y posterior visualización son tareas básicas, comunes a todas las aplicaciones gráficas y, por tanto, susceptibles de una amplia reutilización.

Partiendo de dichos supuestos, se plantea la necesidad de desarrollar un conjunto de herramientas que proporcionen todas las utilidades básicas de tratamiento gráfico y permitan, a su vez, construir nuevas aplicaciones de la forma más rápida posible, permitiendo al futuro desarrollador centrarse en su campo de acción sin necesidad de preocuparse por tareas rutinarias de gestión de información gráfica. El entorno de visualización 2D y 3D multiplataforma descrito en esta comunicación es una de dichas herramientas.

4. Objetivos

El entorno de visualización se planteó desde un primer momento como una única herramienta con dos espacios de trabajo bien diferenciados: uno para la manipulación de información bidimensional (imágenes) y otro para la información tridimensional (geometría).

El primero de dichos espacios, manipulación 2D, debería proporcionar métodos que facilitasen la carga, almacenamiento y visualización de imágenes en distintos formatos; así como la posibilidad de realizar conversiones entre formatos y unas mínimas operaciones de manipulación de imágenes.

El segundo, manipulación 3D, sería análogo al anterior; ofreciendo medios para cargar, almacenar y visualizar geometría, herramientas para manipular dicha geometría, así como las luces y cámaras de la escena, además de un elemental editor de materiales.

Por otra parte, la herramienta tenía que ser fácilmente extensible y personalizable; es decir, aplicaciones externas y totalmente ajenas se debían poder integrar de forma transparente, modificando el interfaz de la herramienta de tal manera que el usuario pudiese emplear tales utilidades de forma coherente con el resto de la aplicación.

5. Estructura de la herramienta

Para lograr tales objetivos de personalización y extensibilidad resultaba clara la necesidad de trascender el concepto de reutilización de código y proporcionar algo de un nivel superior: una estructura lo suficientemente sencilla como para que el interfaz entre la misma y las aplicaciones externas fuera mínimo y, a la vez, lo suficientemente flexible como para poder dar cabida a prácticamente cualquier tipo de posibilidad.

Esto sólo podría lograrse mediante el empleo de módulos dinámicos o *plug-in's*; consistiendo cada módulo en una porción de código independiente que pudiera ser cargada o no según la finalidad que pretendiera darle el usuario final a la aplicación. De esta manera, en función de las necesidades del usuario se emplearía un conjunto u otro, constituyendo dicha agrupación junto con el código que la liga un entorno coherente y cerrado.

Así pues, toda la herramienta consistiría en una colección de módulos destinados a realizar distintas tareas comunicándose a través de un canal común, dicho canal es lo que se vino en denominar el *kernel* de la aplicación. Por tanto, los elementos básicos que constituirían el entorno de visualización serían los siguientes:

- *Cargador:* desde el punto de vista del usuario, sería el programa principal puesto que se trataría del único código ejecutable, aquel que se invoca para iniciar la aplicación; sin embargo, es una de las partes menos importantes puesto que su única finalidad es la de ir cargando en memoria los módulos que constituirán la herramienta final.

- *Núcleo:* compuesto por el gestor 2D y el gestor 3D; el primero proporcionaría, fundamentalmente, métodos para obtener memoria para imágenes multicapa, liberar dicha memoria, administrar la pila de estados que constituye cada imagen y, además, visualizar las imágenes. El gestor 3D ofrecería mecanismos de gestión de geometría, luces y cámaras, así como para el *rendering* de estas escenas.

- *Plug-in's:* módulos que extienden la funcionalidad de la herramienta; aunque no serían estrictamente necesarios para la operación de la misma, la ausencia de alguno de ellos podría constituir una limitación importante. Parte de la funcionalidad que inicialmente se atribuyó al gestor 2D, la de tratamiento de distintos formatos gráficos, recaería posteriormente en extensiones ajenas al *kernel* con lo cual las facilidades para que la herramienta reconociera nuevos tipos de imágenes en el futuro serían aún mayores.

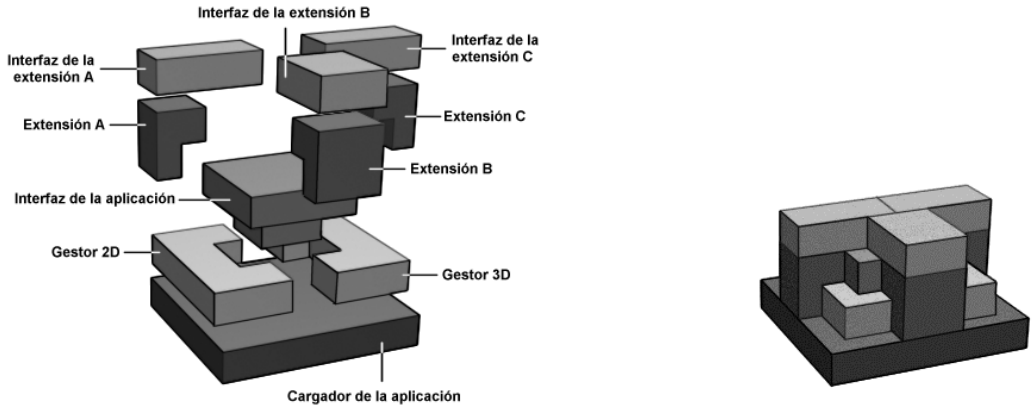


Fig. 1 – Estructura del visualizador 2D y 3D.

En la figura 1 se muestra la estructura de la herramienta. Como se puede observar, el interfaz de la aplicación permite el acceso a parte de la funcionalidad del núcleo de la aplicación (formado por los gestores 2D y 3D). Por su parte, las extensiones o *plug-in's* se componen de dos elementos distintos: el código compilado de la extensión, que interacciona y se apoya en el *kernel* de la aplicación, y el interfaz del módulo, que interacciona con el interfaz de la aplicación además de con el código de la extensión. Por último, núcleo, interfaz y extensiones se apoyan sobre el cargador de la aplicación.

6. Tecnologías empleadas

El planteamiento anterior es, sin lugar a dudas, elegante pero hace surgir un interrogante sobre la forma de llevarlo a cabo: ¿existe alguna tecnología que cumpla requisitos tales como...

- ser fácilmente extensible mediante un lenguaje estándar?
- permitir la implementación de módulos cargables dinámicamente?
- proporcionar mecanismos de desarrollo de interfaces sencillos, rápidos y fácilmente configurables por usuarios no expertos?
- ser portable entre distintas plataformas?

La respuesta es, afortunadamente, positiva: los actuales lenguajes de *script* [1]. Explicar en profundidad la naturaleza de dichos lenguajes no es el objetivo de esta comunicación, aunque sí se hace necesario comentar someramente algunas de las características fundamentales de los mismos, características que los hacen especialmente indicados para un desarrollo de la naturaleza del expuesto en el presente documento.

Entre los aspectos más atractivos de los modernos lenguajes de *script* se pueden destacar:

- Se distribuyen mediante un sistema *open-source*.
- Son lenguajes interpretados con lo cual se agiliza el proceso de desarrollo del *software*.
- Están diseñados con vistas a una amplia portabilidad entre plataformas.
- Son extensibles mediante código C o C++.
- Hay disponibles extensiones para desarrollo de interfaces gráficos de usuario.

Estos aspectos llevarían a enfocar el desarrollo de la herramienta de la siguiente manera: el *cargador de la aplicación* sería implementado como un intérprete de uno de estos lenguajes mientras que el *núcleo* y las *extensiones* lo harían como módulos dinámicos escritos en C o C++; por último, el interfaz de la herramienta se programaría mediante el propio lenguaje de *script* y sus extensiones para *GUI's*.

De los distintos lenguajes disponibles, se optó por la combinación formada por Tcl y su extensión Tk para desarrollo de interfaces [2][3][4]; esta elección se basó, fundamentalmente, en la madurez del lenguaje y en su reconocida fiabilidad para el desarrollo de complejos proyectos de *software*. En el apartado siguiente se tratará con algo más de detenimiento la forma en que se desarrollan proyectos mediante Tcl/Tk y en qué medida dicha filosofía permitió alcanzar los objetivos planteados para este proyecto.

6.1. Tcl/Tk

Tcl *(Tool Command Language)* surge de la mano de John Ousterhout en la Universidad de California en Berkeley; según palabras del propio Ousterhout [3]:

> *Tcl nació de la frustración. A principios de los años 80 mis estudiantes y yo desarrollamos algunas herramientas interactivas [...] pasándonos una gran cantidad de tiempo construyendo malos lenguajes de comandos. Cada herramienta necesitaba algún tipo de lenguaje de comandos, pero nuestro principal interés residía en la herramienta más que en su lenguaje. Dedicábamos tan poco tiempo como fuera posible al lenguaje que siempre terminaba siendo débil e incongruente. Además, el lenguaje de una herramienta nunca era suficientemente adecuado para la siguiente [...] Esto se hizo más y más frustrante.*
>
> *En 1987 [...] se me ocurrió que la solución residía en construir un lenguaje de comandos reusable. Si se pudiera implementar un lenguaje de "script" multipropósito como una librería en C, entonces quizás se pudiera reutilizar para distintos fines en diferentes aplicaciones. Por supuesto,* ***el lenguaje tendría que ser extensible de tal manera que cada aplicación pudiera añadir sus propias y específicas características al núcleo proporcionado por la librería.***

La idea era realmente brillante: una nueva aplicación no se enfocaría como un bloque de código aislado del mundo sino como un conjunto de nuevos comandos de un lenguaje multipropósito que podrían realizar tareas simples (generar un número pseudoaleatorio) o complejas (obtener los contornos de una imagen); el código compilado de ese conjunto de comandos constituiría una extensión que sería cargada por el intérprete del lenguaje cuando fuera necesario y podría ser utilizada de forma sistemática en el futuro.

Citando de nuevo a Ousterhout,

> *[...] Tcl es un* ***excelente "lenguaje de encolado".*** *Una aplicación Tcl puede incluir paquetes muy diferentes, cada uno de los cuales proporciona un conjunto de comandos Tcl. Los "scripts" Tcl para tal aplicación pueden incluir comandos de cualquiera de los paquetes.*

Esta forma de emplear código Tcl para integrar aplicaciones es fundamental; de hecho, es la principal razón del éxito en el desarrollo de la estructura anteriormente descrita, el lector ya comprenderá de qué manera el intérprete Tcl es la base del cargador de la aplicación mientras que el núcleo de la misma junto con los *plug-in's* son extensiones del lenguaje escritas en C/C++, unidas y configuradas mediante código Tcl; dichas extensiones son inmediatamente reusables gracias a su sencilla utilización como comandos Tcl.

A estas cualidades de Tcl hay que añadir la de su extensión Tk, un *toolkit* para desarrollo de interfaces gráficos de usuario; Tk permite obtener en un tiempo record interfaces de gran complejidad mediante la utilización de los denominados *widgets*. Tk, al igual que Tcl, es extensible con lo cual se pueden implementar nuevos componentes para un uso posterior. Uno de ellos es Togl[1], un componente que permite realizar renderizado mediante OpenGL; en una fase inicial del proyecto que se presenta se sopesó la posibilidad de emplear dicho *widget* aunque, finalmente, se descartó en favor de la librería de visualización VTK.

Así pues, las aplicaciones Tcl pueden estructurarse básicamente de dos formas: como un intérprete Tcl junto con unos pocos comandos específicos de la aplicación, tal y como se muestra en la figura 2, o bien añadiendo comandos proporcionados por Tk, además de por otras extensiones, como en la figura 3.

---

[1] http://togl.sourceforge.net

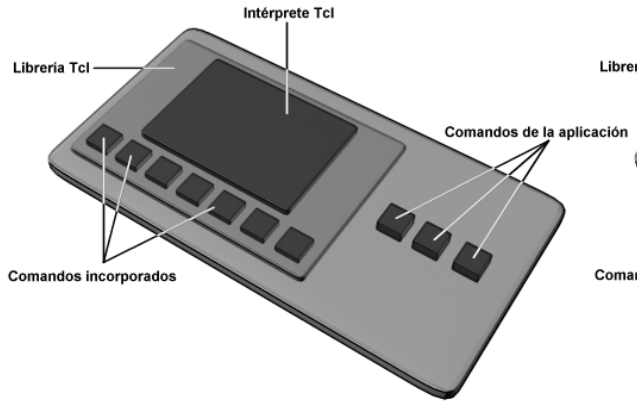


Fig. 2 – Aplicación Tcl.

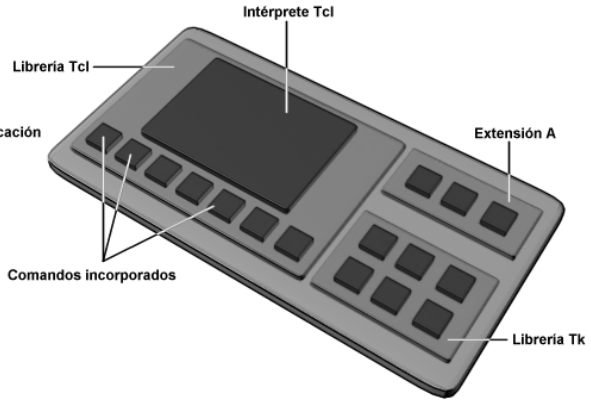


Fig. 3 – Aplicación Tcl/Tk.

La herramienta desarrollada se corresponde con el segundo caso: un programa basado en *wish*[2] carga y ejecuta una serie de *scripts* que, a su vez, proceden a cargar todas las extensiones disponibles para la herramienta así como el código Tk del interfaz y el código de personalización de dicho interfaz; este código emplea comandos Tcl definidos bien en el núcleo de la herramienta (es decir, en el gestor 2D o en el gestor 3D) o bien en las diferentes extensiones, comandos cuyo código habrá sido escrito en C o en C++.

Una vez se hubo determinado la forma en que se iba a desarrollar la herramienta mediante un planteamiento híbrido que mezclaría programación en Tcl/Tk (para el desarrollo de interfaces e integración) y C/C++ (para la implementación de los comandos que proporcionarían la funcionalidad de la herramienta) se pasó a una fase de estudio para determinar el siguiente paso a dar, la implementación del gestor 3D[3].

6.2. VTK *(Visualization ToolKit)*

En el punto anterior se menciona un *widget* para poder llevar a cabo renderizado mediante OpenGL en Tcl/Tk, también se afirma que dicho componente no llegó a ser utilizado puesto que se optó por emplear una librería de visualización. ¿Cuáles fueron las diferencias entre ambos planteamientos para desechar uno en favor de otro? Las razones fueron básicamente las mismas que llevaron a realizar una implementación basada en una combinación de lenguajes *script* y tradicionales en lugar de hacerlo íntegramente en C++: rapidez de desarrollo y reutilización de código.

Desarrollar el gestor 3D directamente en OpenGL conllevaría la implementación de toda una jerarquía de clases para la geometría, luces, cámaras, materiales, interacción con objetos, etc. Sin embargo, todo ese trabajo se hace innecesario si se emplea un marco de desarrollo que ya proporcione dichas clases; en la actualidad hay disponible una gran cantidad de entornos de visualización que llevan a cabo la mayor parte de las prestaciones necesarias para un proyecto de gráficos por computador, pudiendo, además, extenderse en caso necesario mediante clases y métodos propios.

---

[2] *wish* es una aplicación genérica que interpreta código Tcl/Tk, es decir, constituye la base para el desarrollo de aplicaciones con interfaz gráfico; de hecho, todas las aplicaciones basadas en Tcl/Tk emplean el código de *wish* o especializaciones del mismo.

[3] Para el caso del gestor 2D no fue necesaria tal fase puesto que la funcionalidad del mismo es, en comparación con el módulo 3D, reducida, centrándose en una gestión de imágenes multicapa y multiestado, así como en la visualización de las mismas.

Entre las herramientas de visualización más conocidas se encuentran IDL, IRIS Explorer, OpenDX y VTK[4], entre otras. ¿Qué cualidades exhibe VTK para haber fundamentado el núcleo 3D de la herramienta sobre dicha librería? Se podrían destacar las siguientes:

- Se proporciona de forma gratuita el código fuente de la librería.
- Es portable a diferentes variantes de Unix y Windows.
- Puede ser extendida por el usuario mediante sus propias clases.
- Proporciona interfaces de programación para C++ (está escrita en ese lenguaje), Java, Python... y Tcl!

De las otras tres herramientas, tan sólo OpenDX es *open-source*, requiriendo IDL e IRIS Explorer una licencia de uso; por otra parte, tan sólo VTK es una "simple" librería enlazable, el resto son entornos de trabajo que permiten el desarrollo de aplicaciones mediante un lenguaje propietario o de forma visual. Por supuesto, solamente VTK proporciona una capa de interfaz para otros lenguajes, en especial, Tcl.

Una de las mayores cualidades de VTK es precisamente ese interfaz entre la librería y lenguajes interpretados como Tcl [5]:

> *VTK consiste en dos componentes básicos: una librería de clases C++ compiladas y una capa "envolvente" que permite manipular las clases compiladas usando los lenguajes Java, Tcl y Python.*
>
> *La ventaja de esta arquitectura radica en la posibilidad de construir algoritmos eficientes (tanto en uso de CPU como en consumo de memoria) en un lenguaje compilado como C++ y mantener al mismo tiempo las facilidades de desarrollo rápido que proporcionan los lenguajes interpretados (inexistencia de ciclo de compilación y enlazado, herramientas simples pero a la vez potentes y acceso a facilidades para desarrollo de interfaces gráficos).*

Este aspecto es ciertamente interesante puesto que sigue una filosofía muy similar a la de Ousterhout: disponer de código compilado muy eficiente que es invocado mediante sencillos comandos desde un lenguaje de *script* de muy alto nivel. En este sentido VTK podía realizar el trabajo requerido por el gestor 3D, la cuestión era si la librería podría realmente afrontar proyectos gráficos de envergadura. La respuesta fue tremendamente positiva, VTK no sólo permite desarrollar rápidamente sino que proporciona características realmente avanzadas que la convierten en una base muy sólida para cualquier tipo de visualización científica:

- La forma de trabajo es un fiel reflejo del clásico *rendering pipeline*, por lo que cualquier persona familiarizada con los conceptos teóricos fundamentales de los gráficos 3D puede comprender la forma en que se usan las clases de la librería y obtener resultados desde el primer momento.
- Los objetos proporcionados por el núcleo de la librería resultan sencillos de comprender[5]; así, se dispone de vtkActor, vtkLight, vtkCamera, vtkProperty, vtkTransform, vtkRenderer y vtkRenderWindow entre otros. Sin saber nada de VTK, el lector ya puede imaginar qué hacen y cómo se "encadenarían" para obtener los resultados (recuérdese el *pipeline*).

---

[4] El lector interesado puede encontrar más información sobre estas utilidades en los enlaces siguientes: http://www.rsinc.com/idl/index.cfm (IDL), http://www.nag.com/Welcome_IEC.html (IRIS Explorer), http://www.opendx.org (OpenDX) y http://www.kitware.com/vtk.html (VTK).

[5] Esto es totalmente cierto, lo que también es cierto es que para obtener un rendimiento elevado en VTK se requiere una cierta práctica, aunque la curva de aprendizaje no es pendiente en exceso.

- Proporciona prestaciones de alto nivel tales como actores con distintos niveles de detalle, ensamblajes jerárquicos, modelado implícito, interacción en tiempo real, visualización estereográfica, etc.

- Al ser extensible existe una gran cantidad de clases aportadas por distintas personas[6], con lo que el paquete está en constante desarrollo, ganando más y más funcionalidad.

- Existen importadores y/o exportadores de imágenes, geometría y escenas para los formatos más extendidos.

7. Resultado final y conclusiones

En el momento de escribir esta comunicación el desarrollo de la herramienta descrita ya ha llegado a término; los objetivos planteados para la misma se han alcanzado de forma satisfactoria y se puede decir sin miedo a equivocarse que el enfoque adoptado para este trabajo no sólo ha sido un éxito sino que, con toda probabilidad, es uno de los pocos que pueden garantizar unos resultados análogos a los obtenidos en un lapso de tiempo razonablemente corto.

Como muestra del proyecto finalizado, en las figuras 4 a 6 aparecen algunas pantallas de la herramienta en funcionamiento. En la primera pueden apreciarse las capacidades de proceso de imágenes de la aplicación, así como los resultados obtenidos mediante el empleo del primer *plug-in* desarrollado para la misma, una herramienta de tratamiento digital[7]. En la figura 5 se aprecia el entorno de trabajo 3D, junto con una pequeña muestra de primitivas básicas, mientras que en las figuras 6 y 7 aparecen algunos resultados obtenidos a partir de una serie de *TAC's*.

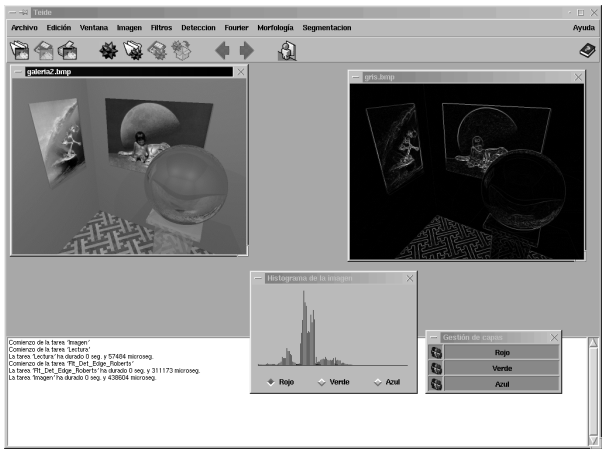


Fig. 4 – Aspecto de la herramienta en el entorno 2D.

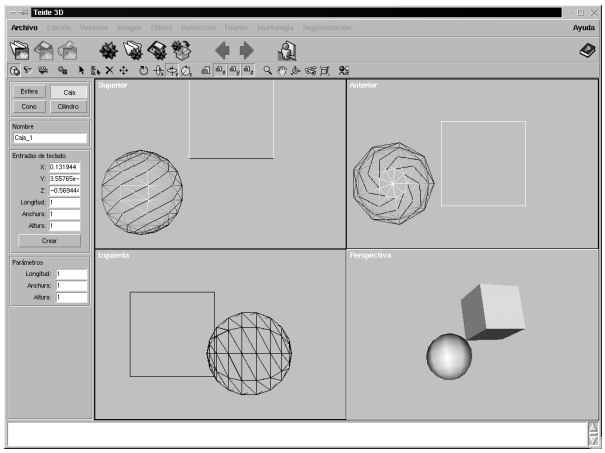


Fig. 5 – Aspecto de la herramienta en el entorno 3D.

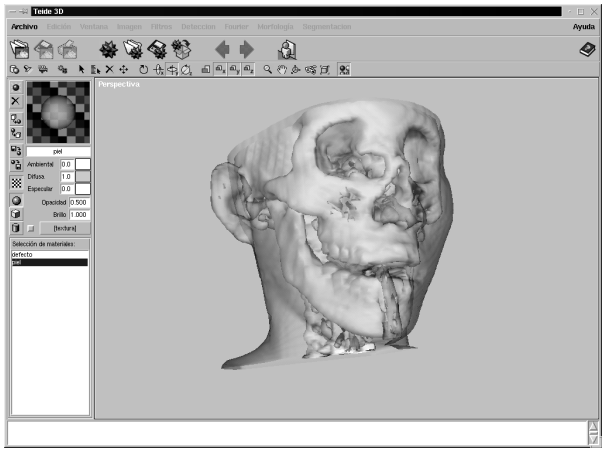
Para terminar y a modo de conclusión, la herramienta descrita no puede considerarse en modo alguno como un punto final sino como un primer paso para desarrollos futuros. Tal vez dichos desarrollos se implementen directamente sobre ella o tal vez no, lo que sí está claro es que se dan las circunstancias y está disponible la tecnología que permiten la construcción de herramientas gráficas multipropósito, potentes, fácilmente

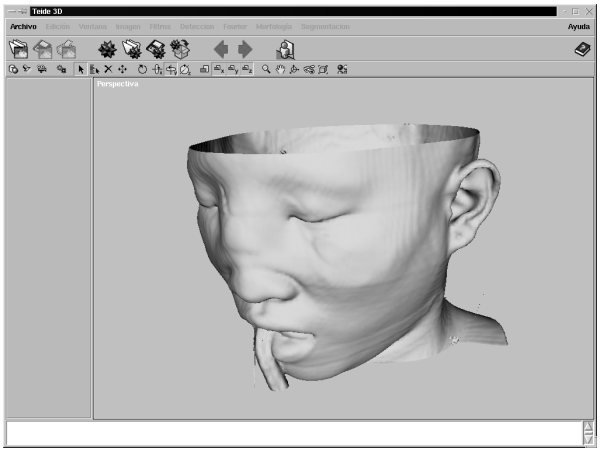
---

[6] O compañías, de hecho General Electric ha contribuido código a VTK, fundamentalmente algoritmos para facilitar el tratamiento de datos médicos; la única pega es que dicho código está patentado y para su empleo es precisa una licencia.

[7] Descrita en la comunicación *"Desarrollo de una librería de tratamiento digital de imágenes orientada a objetos"*.

configurables y extensibles; herramientas que permitirán un desarrollo rápido de aplicaciones gráficas que, a su vez y a modo de *feed-back*, contribuirán al progresivo enriquecimiento de tales utilidades.





Figs. 6 y 7 – Demostración del funcionamiento del entorno 3D para visualización de imagen médica.

8. Bibliografía

[1] Laird, C.; Soraiz, K.; “Choosing a scripting language”. http://www.sunworld.com/swol–10–1997/swol–10–scripting.html, 1999

[2] Harrison, M.; McLennan, M.; “Effective Tcl/Tk Programming”. Addison-Wesley, 1998.

[3] Ousterhout, J.K.; “Tcl and the Tk Toolkit”. Addison-Wesley, 1994.

[4] Welch, B.B.; “Practical Programming in Tcl & Tk”. Prentice Hall PTR, 1997.

[5] Schroeder, W.J.; Martin, K.M.; “The vtk User’s Guide”. Kitware, Inc., 1999.