ESCOLA TÈCNICA SUPERIOR ENGINYERIA
UNIVERSITAT ROVIRA I VIRGILI
Departament d'Enginyeria Electrònica Elèctrica i Automàtica

Representación Tridimensional de Imágenes Termográficas.

Uso de la Librería Gráfica Coin3D.

Titulación: Ingeniería Técnica Industrial especialidad Electrónica Industrial

AUTOR: Abel Morón Barroso
DIRECTOR: Esteban del Castillo Pérez

Junio del 2009

Índice

Estructura de la Memoria

La estructura de la memoria es la siguiente:

- *Introducción y Objetivos.*

  Ámbito de uso de las librerías gráficas de alto nivel y descripción de los objetivos principales del proyecto.

- *Capítulo 1. Las librerías Gráficas.*

  Se describen los conceptos de librería gráfica y se resumen las principales características y aplicaciones de los dos grupos existentes, en especial las de alto nivel.

- *Capítulo 2. La librería Gráfica Coin3D.*

  Se describe el origen de esta librería, su estructura y cuáles son sus principales características y ámbitos de aplicación. Seguidamente, se detallan los conceptos más importantes, a tener presentes, para su uso en el desarrollo de aplicaciones.

- *Capítulo 3. Introducción a la Termometría.*

  Se pretende realizar una breve introducción de los conceptos más importantes en los que se basa ésta técnica. Se explican, de forma general, los tipos de cámaras termográficas, sus aplicaciones y sus principales características.

- *Capítulo 4. Diseño y Desarrollo de una Aplicación de Representación Tridimensional de Imágenes Termográficas.*

  Se realiza el estudio previo de los datos de entrada que utilizará el programa a desarrollar. Después se divide el diseño de la misma en bloques funcionales. En los apartados de cada uno de los bloques, se describen las principales funciones y clases utilizadas. También se justifica la creación de un programa adicional a modo de interfaz gráfica.

- *Capítulo 5. Manual de Usuario.*

  Manual de uso del programa desarrollado en el proyecto. También se incluye el manual de uso del programa adicional.

- *Capítulo 6. Juego de Pruebas del Programa.*

  Se realiza la representación de una imagen termográfica. Los resultados, obtenidos con el programa desarrollado en el proyecto, se comparan con los obtenidos por un software comercial.

- *Capítulo 7. Conclusiones*

Se valoran los resultados obtenidos y se comparan con los objetivos que se pretendían conseguir. También se tratan los principales problemas encontrados durante la realización del proyecto, qué limitaciones finales posee el programa, cuáles podrían ser los principales puntos para la mejora de los mismos y cómo se podría optimizar el código final.

- *Bibliografía.*

  Libros, manuales, artículos y páginas *web* consultadas durante la realización del proyecto.

- *Anexos*

  Formato de los archivos utilizados por la aplicación, listado de comandos y el código fuente del programa desarrollado.

Introducción y Objetivos

En los últimos años, la rápida evolución de los sistemas de computación ha facilitado en gran medida el tratamiento de datos. Diferentes disciplinas se benefician de éstos avances, que les permiten no solo poder manejar más cantidad de datos, sino hacerlo de forma más rápida.

Una parte muy importante de este tratamiento de datos es la posibilidad de representarlos de forma gráfica, y de manera muy especial, en tres dimensiones. La representación tridimensional no solo tiene aplicación en los procesos de diseño estáticos (visualización de cotas en objetos, volumen, etc.), sino que es usada, cada vez más, en ensayos de simulación posteriores (respuestas a esfuerzos, a cambios térmicos, etc.). Éste tratamiento visual de datos requiere el uso de hardware y de software apropiado.

Respecto al software, las conocidas como librerías gráficas de alto nivel, son las herramientas más utilizadas para llevar a cabo el desarrollo de aplicaciones. Su uso acelera el proceso de implementación de código al liberar al programador de detalles de funcionamiento interno, tanto del hardware como de los de la propia librería.

Éste proyecto tiene como objetivo el estudio de algunas de las librerías gráficas existentes, con especial atención a las librerías gráficas de alto nivel con capacidades de representación tridimensional, y a sus características y aplicaciones. A modo de ilustración, se plantea el desarrollo de una programa que haga uso de éste tipo de librerías.

Se pretende desarrollar un programa para el tratamiento de imágenes, en escala de grises, captadas por cámaras termográficas. El programa se enfocará, especialmente, en la representación y visualización tridimensional de las termografias, utilizando para ello la librería gráfica *Coin3D* de *Silicon Graphics* .

Éstas imágenes son producidas a partir de la radiación térmica captada por unos sensores que la transforman en señal eléctrica. Después de diferentes procesos de adecuación y tratamiento de la señal, se representa una imagen en la pantalla de la cámara. Ésta representación se basa en la asignación de un color, habitualmente una tonalidad de gris, a cada uno de los pixeles de la pantalla, conformando de esta manera una imagen térmica. Posteriormente, las imágenes pueden ser almacenadas o exportadas, en diferentes formatos, a un ordenador.

El programa utiliza éstas imágenes, junto con la información suministrada por el usuario, para generar una representación gráfica de los datos de temperatura y ofrecer diversas herramientas para su visualización y/o manipulación, como por ejemplo: obtención de temperatura máxima y mínima, temperatura media, rotaciones, escalado, cambios de paletas, inversión de colores, filtro de temperaturas, isotermas, histogramas y volcados de datos a archivos.

Capítulo 1. Las Librerías Gráficas. Generalidades.

Las librerías gráficas se pueden definir como el conjunto de funciones y clases que generan representaciones de imágenes. Utilizan para ello modelos matemáticos junto con patrones de texturas, iluminación, colores, etc.

1.1 Objetivos de las Librerías Gráficas.

Los principales objetivos de las librerías gráficas son:

- Eliminar la dependencia de la librería con el tipo de hardware donde se usará. Se busca que la librería pueda funcionar en diferentes dispositivos tanto de entrada como de salida.

- Proporcionar al usuario una interface única de acceso a sus recursos.

1.2 Tipos de Librerías Gráficas

Las librerías gráficas para representación de datos tridimensionales se pueden clasificar, principalmente, en dos grupos:

A) Librerías de rendering directo.

Este tipo de librerías requiere tener un buen conocimiento del hardware de representación de datos 2D/3D, para conocer las posibilidades y limitaciones que tenemos.
Por tanto, implica un proceso de desarrollo más largo. Se debe realizar un estudio cuidadoso de todos los procesos que se deben de llevar a cabo antes de poder llegar a ver en pantalla el objeto 3D.

El programador que utilice este tipo de librerías necesita conocimientos sobre aspectos como: la gestión de la carga/descarga de memoria, uso del buffer gráfico, etc.
Dentro de este grupo se englobarían las siguientes las siguientes librerías:
OpenGL, Direct3D, GKS, PHIGS, PEX, etc.

B) Librerías de alto nivel o descriptivas.

Estas librerías permiten desarrollos de aplicaciones 3D más rápidos que las anteriores. En este caso, la programación se simplifica más debido a que se focaliza cada representación de las imágenes mediante un árbol de escena. Además la librería gestiona la carga/descarga de memoria, gestiona el uso del buffer gráfico, gestiona los elementos no visibles de la escena (evitando renderizarlos, para ahorrar recursos del sistema y tiempo de procesado), etc.

En esta categoría también podrían englobarse los *toolkits*, ya que facilitan la tarea de programación del usuario, ofreciéndole un conjunto de funciones y de clases, generalmente en C/C++ como paradigma de POO. Además ofrecen la

posibilidad de usar varias capas de interface (*bindings*) por si se requiere su uso con otros lenguajes de programación.

El uso de *toolkits*, o paquetes de librerías de alto nivel, ofrecen al usuario un conjunto de funciones y clases con el objetivo de que el usuario reduzca el tiempo invertido en el desarrollo de la aplicación.

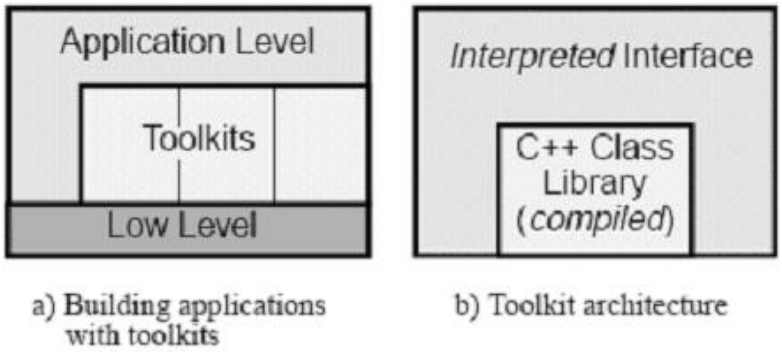


figura 1. Arquitectura de los toolkits y uso en el desarrollo de aplicaciones

Dentro de las librerías gráficas de alto nivel encontramos: Open Inventor, Performer, Open Scenegraph, OpenGL Performer, OpenGL Optimizer, PHIGS+, etc.

1.3 Librerías de Bajo Nivel

Las dos librerías principales de bajo nivel para representación gráfica y tridimensional que se han convertido en estándar son: Direct3D y OpenGL.

1.3.1 Direct3D

Forma parte del conjunto DirectX. Se trata de una API para facilitar la programación y procesado de aplicaciones de gráficos 3D. Facilita la realización, manipulación y transformación de entidades elementales como: líneas, polígonos, texturas, etc.

Aunque ésta API provee un cierto nivel de abstracción del hardware, su uso no se basa en escenas gráficas y jerarquías de nodos, por lo que se la puede considerar como una capa intermedia entre la aplicación y los drivers de la tarjeta gráfica.

La API es propiedad de Microsoft, y su uso es exclusivo para plataformas Windows de 32 y 64 bits: Windows 95/98/ME, 2000/NT, XP, Vista.
Es una API de altas prestaciones que provee total soporte para aceleraciones por hardware.

Su licencia es del tipo EULA (*End User License Agreement*), por la cual el uso de un producto sólo está permitido al comprador de la misma.

1.3.2 OpenGL (Open Graphics Libray).

Es una API multiplataforma, introducida en 1992 por *Silicon Graphics Inc* (SGI), basada en IRIS GL, una API para workstation SGI.

Al tratarse de una API de bajo nivel, el desarrollador necesita tener un buen conocimiento del proceso que se vaya a llevar a cabo antes de realizar el renderizado de la imagen en pantalla.

En ésta API, el proceso de renderizar las entidades gráficas lo realiza la llamada máquina de estados de OpenGL, por tanto, el programador es el encargado de describir explícitamente todos los pasos necesarios para la construcción, manejo y renderización de la imagen 3D.

Aunque esto, al principio, puede suponer una dificultad añadida para su uso, como contrapartida se obtienen todas las ventajas de poder tener un acceso al hardware a más bajo nivel, lo que proporciona, prácticamente, total libertad para realizar los efectos o acciones en 3D deseados.

Existen diferentes implementaciones de OpenGL. Estas implementaciones deben de ajustarse a las especificaciones marcadas por el OpenGL *Architecture Review Board* (ARB), en los orígenes de la API. Actualmente esta tarea la lleva a cabo el OpenGL ARB *Working Group*, dentro del grupo *Khrono*s.

Una característica muy apreciada de esta librería, es que se pueden crear extensiones que dan la posibilidad de introducir nuevas capacidades. Básicamente, éstas extensiones suponen la creación o eliminación/asimilación de nuevas funciones para la librería.
Para diferenciar los contenidos extendidos de la API base original, los fabricantes incluyen una serie de sufijos a modo de abreviatura que las identifica como extensiones propias del fabricante en cuestión.

OpenGL se ha convertido en un estándar en gráficos 3D porque ofrece altas prestaciones, y se dedica, casi exclusivamente, al renderizado de las imágenes. No realiza la gestión de las ventanas, sonido, etc. Deja estas tareas para el sistema operativo. Por tanto, se libera de poseer módulos API adicionales, tal y como posee su principal competidor DirectX.

Otro de los aspectos que ha hecho a OpenGL un estándar es la gran documentación existente sobre la misma.

Existen versiones de OpenGL para una gran variedad de lenguajes de programación. Hay versiones para: Java, Fortran, Delphi, C#, Perl, Phyton, VB, Ada, etc.

Basándose en OpenGL se han creado muchos de los *toolkits* o librerías de alto nivel existentes: como Performer, Quesa3D, Open Inventor, OpenGL VizServer, OpenGL Volumizer, Hoops3D, Gizmo3D, OpenSceneGraph, entre otras.

1.4 Librerías de Alto Nivel o Descriptivas

1.4.1 VTK The Visualization Toolkit.

Se trata de un *toolkit* multiplataforma, con soporte para sistemas basados en UNIX, Windows y MacOS, implementado en C++, para aprovechar todos los beneficios de la POO, y compuesto por más de 700 clases. Es usado para el modelado, procesado, y manejo de datos.

Es muy usado, especialmente en la comunidad científica y de investigación, para la representación de datos en 3D. También en la comunidad universitaria y en empresas en todo el mundo.
Provee soporte auxiliar para dispositivos de interacción 3D y para la computación en paralelo. Como exponente de la potencia y versatilidad de VTK se encuentra *ParaView*, una aplicación escrita en C++ y Tcl/Tk, usada para el análisis y tratamiento de cantidades masivas de datos. *ParaView* utiliza las sobresalientes capacidades de computación en paralelo que posee VTK.

VTK ofrece soporte para el uso con lenguajes de intérprete como Java, Tcl/Tk y Pitón.

Los proyectos realizados en VTK se basan en dos partes diferenciadas.

- Un modelo de gráficos 3D abstracto compuesto de nueve objetos básicos:

  Render Master, Renderwindow, Renderer, Light, Camera, Actor, Property, Mapper y Transform.

- Visualization Model.

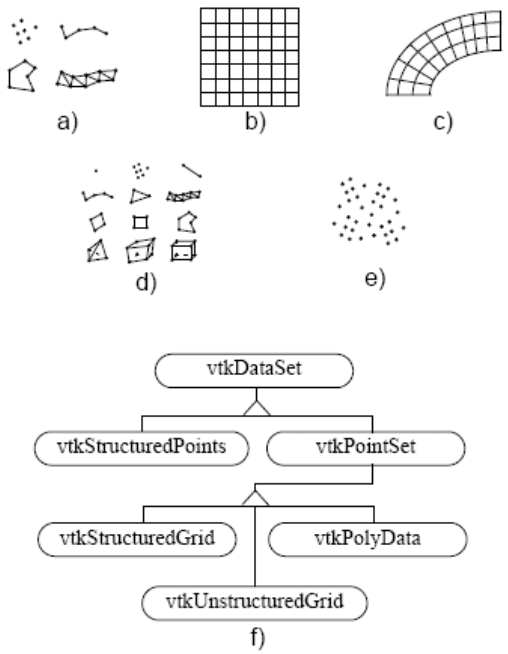


Figure 4 Dataset types. a) polygonal data, b) structured points, c) structured grid, d) unstructured grid, e) unstructured points, f) object diagram using OMT [2] notation.

Figura 2. Diferentes tipos de objetos de la librería VTK

El modelo de visualización de VTK se basa en el paradigma de flujo de datos.

Los módulos están conectados juntos en una red (*network*) de visualización.

Los diversos módulos realizan las operaciones algorítmicas sobre los flujos de datos.

La ejecución de la visualización de la red, se controla en respuesta a las demandas de datos (ejecución bajo demanda de datos) o en respuesta a un evento del usuario (ejecución por evento).

El modelo de visualización consta de dos tipos básicos de objetos: *process objects* y *dataobjects o datasets.*

Los *processObjects* son porciones de código de la red de visualización y los *dataobjects* representan y posibilitan las operaciones con los datos.

El *toolkit* VTK se distribuye como software libre bajo licencia BSD.

1.4.2 Matpack.

Matpack es una librería numérica y de representación de graficos3D, basada en C++, que implementa métodos de utilidad para diversos tipos de necesidades.
La librería cuenta con usuarios como científicos, ingenieros, universidades e institutos y también es usada en productos comerciales.

Posee un variado surtido de soluciones algorítmicas como: generador de números aleatorios, mediante el uso diferentes distribuciones (uniforme, exponencial, *gamma*, *Erlang*, *Poisson*, binominal, etc.), integración, interpolación *(Splines: cubic, Bezier, cardinal, rational, exponential, Akima, Fritsch-Carlson*, etc.), álgebra para matrices/vectores, criptografia, FFT, etc.

En cuanto a sus capacidades 3D, las principales son:

Generación de gráficos de superficies 3D, gráficos de imagen, librería de manipulaciones de imagen, histogramas, gráficos 3D de secciones de objetos, Isosuperficies 3D, superficies paramétricas, etc.

También posee herramientas adicionales para complementar y facilitar la tarea del programador. Algunas de ellas son: *Xmatrix* (herramienta de visualización 2D/3D que utiliza la clase *MpView*), generador de fractales, *MpWindows* (para facilitar la creación de interfaces gráficas de usuario), etc.

MatPack es multiplataforma. Funciona en sistemas basados en Unix, LINUX y en Windows. Se distribuye bajo licencia GPL.

1.4.3 OpenInventor

Es un API 3D de alto nivel, basada en OpenGL, orientada a objetos, multiplataforma, desarrollada por la empresa *Silicon graphics Inc* (SGI).

Su principal característica es que permite simplificar la programación en 3D mediante el uso de *scenegraphs.*
Incluye un conjunto predefinido para creación de objetos 3D como: cubos, texto, cámaras, luces, manipuladores varios, visores 3D, etc.

Además posee las siguientes características:

- Independencia del sistema de ventanas.
- Modelo simplificado de eventos para la interacción en 3D.
- Capacidades de animación de objetos mediante elementos especiales llamados *engines.*
- Capacidades de selección de objetos en pantalla.

- Permite extender sus capacidades con la programación de nuevos objetos personalizados.

Existen dos versiones de Open Inventor:

- Una *open source*. Se distribuye con licencia GNU *LGPL (Lesser General Public License*) por la empresa SGI.
- Versión comercial de la empresa Mercury Computer Systems.

1.4.4 OpenSceneGraph

OpenSceneGraph es un *toolkit*, orientado a objetos, desarrollado en C++ y basado en OpenGL, es multiplataforma y completamente independiente del sistema de ventanas. Ofrece soporte nativo para los sistemas de ventanas Win32, Unix y OSX (*Carbon*).

Está enfocado al desarrollo de aplicaciones gráficas de alto rendimiento como: simuladores de vuelo, deportes, realidad virtual y visualización científica.
Como la mayoría de *toolkits* basados en OpenGL, facilita el desarrollo de aplicaciones mediante el uso de *scenegraphs.*

Sus principales características son:

- Uso de nodos de tipo (LOD). Gestionan la carga de polígonos dependiendo de la distancia de visualización de los objetos.

- Soporte para múltiples formatos de archivos gracias al uso de *plug-ins* (módulos o programas anexados que aumentan las funcionalidades del programa principal) específicos. Existen *plug-ins* para archivos de tipo:

  LightWave (.lwo), VRML 1.0 (.wrl), Alias Wavefront (.obj), Carbon Graphics GEO (.geo), 3D Studio MAX (.3ds), Quake Character Models (.md2), Direct X (.x), Inventor Ascii 2.0 (.iv), Designer Workshop (.dw), AC3D (.ac).

- Soporte para los formatos de imagen:
  SGI *Silicon Graphics* (.rgb), Tagged Image File Format (.tiff), Graphics Interchange format (.gif), Independent JPEG Group (.jpg), Portable Network Graphics (.png), pic, bmp, DirectDraw Surface (.dds), Truevision Targa Files (.tga).

- Uso de *nodekits*. Éstos contienen los elementos necesarios para facilitar el desarrollo de aplicaciones. Los principales tipos son los siguientes:

  OSGPArticle. Para sistemas de partículas,
  OSGShadow. Para el manejo de sombras
  OSGMAnipulator. Proporciona controla 3D interactivos
  OSGTerrain. Facilita el rendering de terrenos.
  OSGVolume. Rendering de volumen de alta calidad para aplicaciones médicas.
  OSGAnimation. Para la creación y simulación de elementos del cuerpo humano.

Se trata de una librería *opensource* con licencia OpenSceneGraph *Public License* (OSGPL) que está basada en el tipo de licencia *GNU Public License* (LGPL).

Capítulo 2. La librería Gráfica Coin3D.

2.1 Orígenes y Trayectoria de la Librería.

Los orígenes de la librería se remontan al año 1995. Los desarrolladores de la empresa *System in Motion*, deseaban realizar una librería de rendering de gráficos 3D para dar soporte al formato de archivos VRML.

Pronto la librería que se había desarrollado necesitó de mayor flexibilidad y funcionalidad para poder soportar los crecientes requisitos en el desarrollo de aplicaciones 3D. Se replanteó el diseño de la librería, considerándose como objetivos principales el conseguir una librería para el renderizado de gráficos 3D con soporte para múltiples plataformas y formatos de archivo.

Basándose en el diseño de la API Open Inventor, se creó una nueva librería compatible con Open Inventor, que pasaron a llamar Coin3D. El principal esfuerzo de los desarrolladores fue hacer la librería lo más compatible posible con Open Inventor.

La librería gráfica Coin3D se puede considerar como una alternativa a la API Open Inventor de *Silicon Graphics Industries* (SGI), y a la versión propietaria de la misma (originalmente denominada IRIS Inventor) perteneciente a la empresa *Tamplate Graphics Softwar*e(TGS).

Aproximadamente al mismo tiempo que se lanzó la versión *open source* de la API Open Inventor, en el año 2000, por parte de SGI, la empresa *System in Motion* (SIM) lanzó la librería Coin3D, que se distribuyó con un modelo de licencia dual y desde entonces está disponible en dos versiones:
Una versión como software libre bajo licencia *Lesser General Public License* (LGP) (*Coin Free Edition*) y otra versión comercial para el desarrollo de software propietario (*Coin Professional Edition*), ambas con funcionalidad completa para las plataformas oficialmente soportadas, acceso al código fuente y a paquetes binarios.

Poco después la empresa SIM fue adquirida por el grupo Kongsberg, un grupo formado por las compañías Kongsberg Defence & Aerospace (KDA) y Kongsberg Maritime (KM), pasándose a llamar Kongsberg SIM.

El objetivo principal de los desarrolladores de Coin3D es conseguir una librería que combine una codificación eficiente y una alta funcionalidad, para cualquier usuario o desarrollador que necesite realizar programas que requieran capacidades 3D.

La librería Coin3D sigue desarrollándose para dotarla de mejoras y soporte para más formatos de archivo. Actualmente, la última versión estable de la librería Coin3D es la 3.0.0, que fué lanzada el 11-09-2008.

2.2 Estructura y Características

Coin3D es un *toolkit* de gráficos 3D de alto nivel, desarrollado en C++ y multiplataforma, ideado para ser usado en programas de visualización 3D en tiempo real. Se puede considerar como una implementación independiente o clon de la API

Open Inventor, API considerada como un estándar en entornos científicos y aplicaciones de ingeniería.

Al igual que Open Inventor, Coin3D se basa en OpenGL. Se considera una capa de alto nivel para la programación gráfica y renderizado 3D. Así, código escrito mediante Coin3D (alto nivel) puede coexistir con código de OpenGL (bajo nivel), de manera que los desarrolladores que usen OpenGL pueden realizar una migración a Coin 3D de manera gradual.

De éste modo, se consigue reducir el tiempo de desarrollo de aplicaciones que necesiten implementar capacidades de creación o manipulación de objetos en 3D, ya que la creación/importación, el renderizado y la interacción con los objetos, se consigue mediante pocas líneas de código, lo que supone una mayor eficiencia respecto al uso exclusivo de OpenGL.

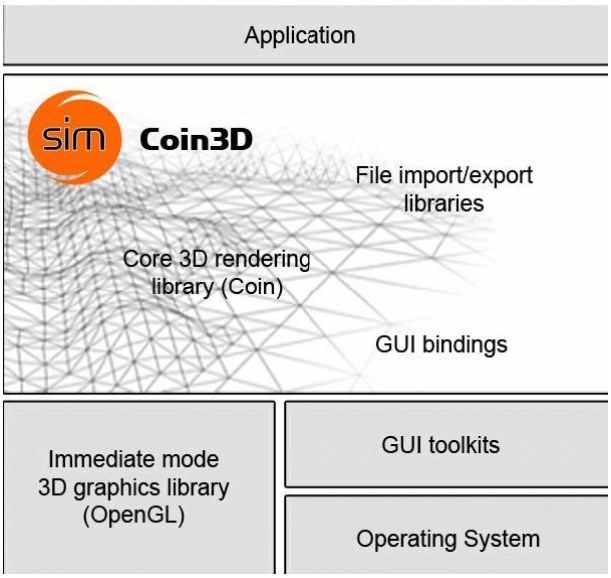


Figura 3. Diagrama librería Coin3D

El núcleo de la librería COIN3D usa la estructura de escenas gráficas para almacenar y realizar el renderizado de la imagen en tiempo real. Gracias a su diseño basado en datos, solo vuelve a dibujar la escena gráfica si se producen cambios en los datos de la misma, lo que la hace útil en aplicaciones basadas en interfaces, por ejemplo, en aplicaciones científicas o de ingeniería que no reclamen recursos excesivos de la CPU.

El núcleo de COIN3D es independiente y portable entre distintas plataformas. Actualmente la lista oficial de plataformas soportadas es:

- Windows 95/98/NT/2000/XP.
- GNU/LINUX.
- Mac OSX.
- SGI IRIX.
- Sun Solaris.

Para facilitar la programación en los diversos sistemas operativos, Coin3D ofrece una interfaz de usuario básica, que permite crear ventanas para el renderizado. También, proporciona facilidades en la gestión de los eventos del teclado y ratón mediante una serie de *bindings* disponibles. Actualmente existen los *bindings*:

- *SoWin*. Para GUI nativo de Windows.
- *SoQt*. Para GUI nativo de Trolltech's QT
- *SoXt*. Para GUI nativo de Xt/Motif en X Windows
- *Sc21*. Para GUI en Mac OSX .

Las características más importantes de la librería son:

- Soporte de fuentes tipo *TrueType* para anotaciones en 2D/3D en la ventana de renderizado.
- Texturas 3D, multitexturas, "*bump mapping*".
- Librerías de importación/exportación para diversos formatos de archivo.
- Soporte de datos geográficos.
- Conjunto de primitivas (box, cone, cilinder, etc.).
- Renderizado en tiempo real.
- *Generic GLSL Shader Support.*
- Renderizado de sombras.
- Soporte para *multithreading*.
- Soporte completo para el formato VRML97.
- Renderizado paralelo en múltiples procesadores.

Coin3D integra, adicionalmente, una serie de librerías que facilitan la tarea de importación/exportación de archivos de uso común en aplicaciones 3D.

- ***Simage:*** permite cargar, manipular y guardar imágenes en diferentes formatos. Posibilita la creación y grabación de archivos de vídeo (AVI y MPEG). También soporta reproducción de archivos de sonido. Los archivos soportados son los siguientes:

  Archivos de imagen.
  - JPEG (escritura/lectura mediante el uso de la librería libjpeg)
  - PNG (escritura/lectura usando las librerías libpng y zlib)
  - GIF, TIFF
  - RGB (solo lectura)
  - PIC (solo lectura).
  - Targa TGA (solo lectura).
  - *encapsulated postscript* EPS (solo escritura).

  Archivos de vídeo.
  - AVI (creación y guardado), MPEG (creación y escritura)

  Archivos de sonido.
  - OGG (reproducción)

- ***Dime:*** Permite cargar, manipular y salvar archivos en formato DXF.

- ***Profit:*** Permite cargar, manipular y salvar ficheros de modelado *3D Multigen Open Flight.*

Adicionalmente, y a modo de complemento, existen las librerías:

- ***SIM Voleon*** *(volume rendering library*). Especializada en el renderizado volumétrico de objetos, característica muy usada en programas de aplicación en el campo de la medicina.

- ***SIM Scenery.*** Su objetivo principal es la visualización optimizada de terrenos 3D.

2.3 Conceptos de Interés para la Programación con Coin3D.

2.3.1 Convenciones de Nombres en Coin3D.

La librería usa una nomenclatura propia aplicable a diferentes elementos. Los principales son:

- ***Tipos básicos de datos.***
  Al igual que *long*, *int*, *char* en C/C++, la librería posee sus propios tipos de datos. Los tipos básicos empiezan siempre con el prefijo "Sb".
  Ejemplo: *SbColor*, *SbViewVolume*.

- ***Clases.***
  Todos las clases en Coin3D llevan como prefijo las letras "So".
  Ejemplo: *SoCone*, *SoMaterial*, *SoTransform*, etc.

- ***Métodos y variables.***
  Los métodos y las variables miembro de las clases comienzan siempre con letra minúscula. Cada palabra que no sea una clase, método o variable, comienza con letra mayúscula.
  Ejemplo : *getNormal()*, *SetScaneGraph ()*, *myCube()*.

- ***Enumeraciones.***
  Los valores de tipo enumeración se escriben todos en mayúsculas.
  Ejemplo: *FILLED*, *PER_PART*.

2.3.2 Database y Scenegraph

El desarrollo de aplicaciones 3D se ve facilitado al usar librerías de alto nivel. Coin3D, como el resto de librerías de alto nivel, usa una estructura de datos denominada *scenegraph*. El concepto de *scenegraph* viene a representar la manera en la que el programador construye, agrupa y manipula los objetos 3D. Una *scenegraph* no es más que una colección de nodos ordenados en una estructura de tipo árbol. Una vez construida la *scenegraph* se pueden realizar diversas operaciones, aplicar acciones, seleccionar objetos, guardar objetos en un archivo, etc.

Cada vez que se crea una *scenegraph*, ésta se almacena en la base de datos de escenas, llamada *scenedatabase*, que puede contener más de una *scenegraph*. Los elementos más importantes en la construcción de *scenegraphs* son los nodos.

2.3.3 Tipos de Nodos

Los nodos contienen los datos que definen algunas figuras 3D, propiedades o grupos, en los llamados campos (*fields*). Cuando un nodo se crea es automáticamente insertado en la *scene database* como un nodo raíz. Después, cuando el programador lo desea, conecta el nodo a otros nodos de la *database* para construir una jerarquía.

En Coin3D los nodos se pueden dividir en 3 categorías básicas:

- ***Shape nodes***.
  Representan objetos geométricos en 3D.
- ***Property nodes*** .
  Representan apariencia y otras cualidades características de las escenas.
- ***Group nodes***.
  Se pueden considerar como contenedores de otros nodos del gráfico. Son los únicos, salvo algunas excepciones, que pueden tener otros nodos como hijos. De ahí que se conozcan también con el nombre de nodos padres. Sin embargo sí pueden añadirse como nodos hijos de otros nodos de grupo.

Todas las *scenegraph* están hechas de uno o más nodos. Los nodos son clases con sus propios métodos y campos. Por tanto, cada uno de los nodos, como objeto de una clase, hereda las propiedades del nodo de la clase de donde deriva.
El conjunto de métodos y variables miembro de cada nodo representa una geometría, propiedad o grupos de objetos.

Las jerarquías, en las *scenegraph*, son creadas añadiendo nodos hijos a un nodo ya existente, creando, de esta manera, una estructura de nodos similar a un árbol. Cada nodo puede tener más de un nodo hijo, y también, aunque no es lo habitual, más de un nodo padre.

Un ejemplo de *scene database* simple seria el de la figura siguiente. En ella se aprecian 3 *scenegraphs* A, B y C, que representan, respectivamente, las 3 configuraciones más habituales que pueden adquirir los nodos dentro de las *scenegraph.*

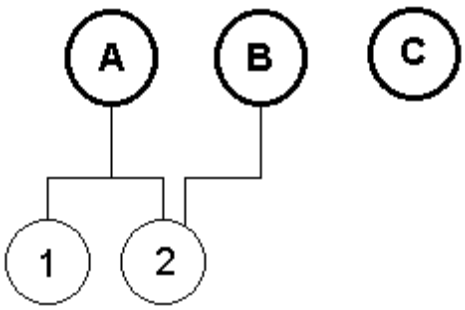


Figura 4. Ejemplo de *database*

Los nodos A, B y C son denominados, comúnmente, nodos raíz. Reciben este nombre porque se encuentran a la cabeza de la *scenegraph* que representan.

El nodo 2 representa un caso de instancia compartida, llamado *shared instancing*, ya que se encuentra compartido por dos nodos padres.
No es una práctica muy habitual. Puede resultar útil en ciertos casos, como por ejemplo, en el caso de que se quieran reutilizar objetos sin tenerlos que volver a definir.

El nodo raíz C no está conectado a ningún otro nodo de la *scene database*. Usualmente, éste es un estado temporal y posteriormente, cuando se necesite, el nodo se conectará a otros nodos para construir una escena gráfica.

Una parte fundamental en el desarrollo de aplicaciones 3D, que usen librerías de alto nivel, es el de empezar diseñando el árbol de nodos de la escena.

Esta práctica permite identificar errores que de otra manera pueden pasar desapercibidos, o consumir mucho tiempo en el depurado de los mismos.
Otro factor a tener presente, que aconseja adoptar esta práctica, es el concepto de *traversal state* y su relación con el orden de los nodos dentro del árbol. El conocimiento de esta relación evita que se originen situaciones en las que, aparentemente, el código no contiene errores pero en cambio los resultados obtenidos no se corresponden a los esperados.

2.3.4 Traversal state y Orden de los Nodos

El *traversal state* hace referencia a la manera en cómo se recorren los árboles de nodos y en cómo se aplican las acciones a las *scenegraph.*

Algunos de los datos más importantes a los que hace referencia el *traversal state* son los siguientes:

- Valores actuales de material para el renderizado de objetos. Incluye: color de superficie, color ambiental, brillo, transparencia, etc.
- Tipo de luz actual y sus valores.
- Tipo de estilo de dibujo.
  Relleno(*filled*), líneas, puntos, invisible, anchura de puntos, *lineWidth*, *linePattern*, etc.
- Tipo de fuente actual.
- Coordenadas 3D actuales.
- Componentes normales actuales.
- Estado de la vista actual. Tipo de cámara (*camera position*, ángulo, rotación), etc.

El recorrido habitual del *traversal state* es el que se realiza al renderizar la imagen.
El proceso comienza en el nodo raíz, ubicado en la parte superior del árbol. Después pasa por el nodo hijo, que se encuentra más a la izquierda, recorriendo progresivamente el resto de nodos hijos.

En todos los nodos por donde se pasa se recorren todos sus nodos hijos, antes de seguir avanzando hacia la derecha.

De esta manera, al final del proceso, se ha recorrido toda el árbol de nodos, desde el nodo raíz hasta el último nodo hijo en la zona baja derecha del árbol.

2.3.5 Importancia del orden de los nodos.

Cada nodo tiene su propia manera de responder a las acciones que se realicen en la *database*.

- Si el nodo que se va a renderizar es del tipo grupal, se invoca un *render* en orden. Primero se pasa por cada uno de sus hijos, de izquierda a derecha, del árbol de nodos que compone la *scenegraph.* A su vez, cada nodo hijo ejecuta sus propios *renders* a sus hijos. Y así sucesivamente.

- Si se trata de un nodo de propiedades, éste modificará los valores actuales (de color de superficie, textura, posición, etc.) del *traversal state.*

- Si el nodo es del tipo *Shape node* (figura), la figura se dibujará usando los valores actuales (de color de superficie, textura, posición, etc.) del *traversal state.*

El siguiente ejemplo (figura 5) pretende ilustrar la estrecha relación que guarda el *traversal state* y el orden de los nodos dentro de la *scenegraph*.

Supongamos que la propiedad que establece el color de la superficie de los objetos renderizados está establecida, por defecto, con el color blanco.

El recorrido comienza en el nodo raíz. Se baja después al nodo *grupo1* donde se visitan sus dos nodos hijos. Al pasar por el nodo *PropRojo*, la propiedad actual del *traversal state* se establece en rojo. Seguidamente, al pasar por el nodo *ShapeEsfera*, se realiza el renderizado de la esfera en ese color.

Al llegar al nodo *grupo2*, el color cambia a verde, y al pasar, seguidamente, por el nodo *ShapeCubo*, se renderiza un objeto tipo cubo con dicho color. Alcanzado el nodo *grupo3*, se visita su único nodo hijo, pero como la propiedad actual de color del *traversal state* está establecida en verde, el cilindro se mostrará en dicho color.

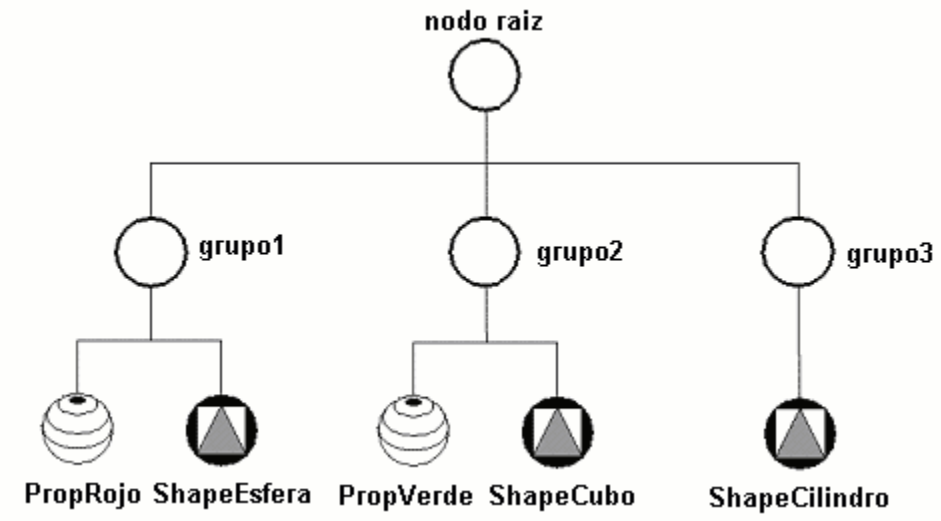


Figura 5. Arbol de nodos con nodos de grupo

Es habitual que el programador requiera establecer una propiedad del *traversal state* que solo afecte a un nodo especifico. Para poder lograrlo se utilizan unos nodos de grupo denominados separadores (*SoSeparator*). Éstos guardan automáticamente el estado del *traversal state* y, cuando se han recorrido todos los nodos hijos del nodo separador, vuelven a restaurarlo.

En el siguiente ejemplo (figura 6) se muestra una estructura de nodos similar al caso anterior. En éste caso, también se usan nodos de grupo, pero del tipo separadores, para los nodos *grupo1*, *grupo2* y *grupo3*.

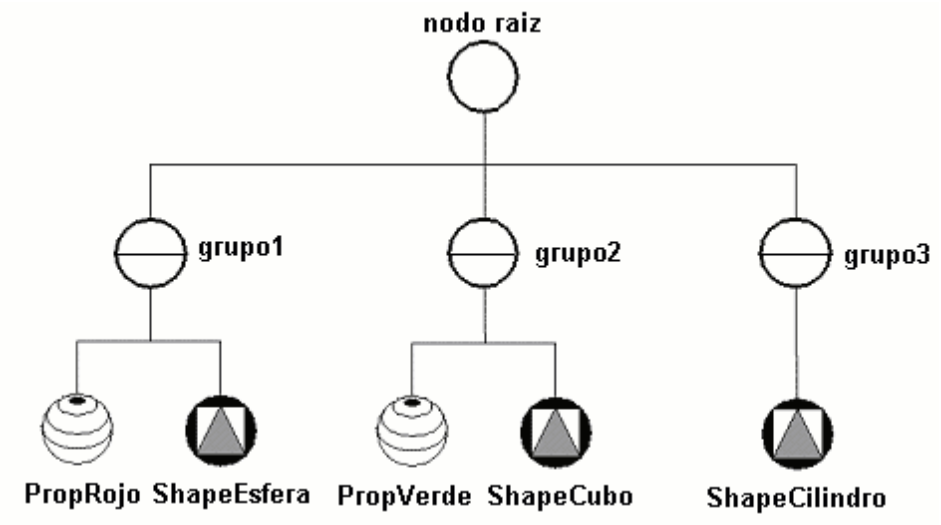


Figura 6. Árbol de nodos usando nodos de grupo del tipo separadores.

En este caso, como en el anterior, la propiedad de color inicial del *traversal state* está establecida en blanco.

El recorrido de los nodos es el mismo, pero el resultado obtenido no. Se empieza por el nodo raíz, el cual visita a su primer nodo hijo *grupo1*. Éste, al pasar por el nodo de propiedad (*PropRojo*), establece el color en rojo y, al visitar al nodo esfera, da como resultado el renderizado de la esfera en dicho color.

Al salir del nodo *grupo1*, el *traversal state* restaura su estado anterior (color = blanco). Después se dirige al nodo *grupo2*. Aquí el nodo *PropVerde* establece la propiedad de color a verde, lo que resulta, al pasar por el nodo *ShapeCubo*, en el renderizado de un cubo en dicho color.

Al salir del nodo *grupo2*, se vuelve a restaurar el estado inicial del *traversal state* (color = blanco), dando como resultado el renderizado de un cilindro de éste color cuando se recorre el único nodo hijo (*ShapeCilindro*) de *grupo3*.

Resumiendo, el *traversal state* es una clase interna, usada por Coin3D para almacenar el estado transitorio de los elementos (parámetros), durante la ejecución de una acción o del *Render*[1] .

2.3.6 Tipos de datos básicos.

Coin3D utiliza sus tipos de datos propios, derivados de los tipos de datos básicos usados en C++. Éstos, no solo sirven para conocer o establecer el valor de los campos que conforman las propiedades de un nodo, sino que facilitan la tarea del programador, brindándole métodos para lectura/escritura, manipulación y conversión.
La lista de los principales tipos de datos de Coin3D es la siguiente:

- **SbBool:** *Boolean value* (TRUE o FALSE)
- **SbBox*nx*:** Utilizado para definir un *box* 2D/3D mediante planos paralelos a los ejes. Se definen por 2 puntos de una diagonal. *"n"* indica el número de datos (2 o 3) y "*x*" el tipo de dato ("*f* "para *float* y "*s*" para *short int*). Incluye SbBox3f, SbBox2f, SbBox2s.
- **SbColor:** Facilita el manejo de las componentes de color RGB (*red/green/blue*). Internamente cada componente es representado por un dato tipo *float.*
- **SbCylinder:** Tipo de dato específico para la construcción de cilindros mediante un eje y un radio.
- **SbLine:** Para representar una línea 3D en el espacio, mediante un punto de origen y un vector de dirección.
- **SbMatrix:** Almacenamiento de valores, en coma flotante de simple precisión (*float*), en una matriz de 4x4.
- **SbName:** Cadena de caracteres utilizada, habitualmente, para tareas de lectura y comparación. Típicamente usada como identificador de objetos o nodos.
- **SbPList:** Es una clase contenedor para punteros de tipo *void* *.
- **SbPlane:** Representa un plano 3D.
- **SbRotation:** Representación de una rotación 3D alrededor de un eje. Internamente constituido por un conjunto de 4 *floats*.
- **SbSphere:** Tipo de dato específico para la construcción de esferas mediante un punto 3D y un radio.
- **SbString:** Permite el almacenamiento de cadenas y ofrece métodos para su fácil

[1] En las librerías de alto nivel el renderizado se suele considerar como una acción.

manipulación (operadores sobrecargados, de asignación, comparación, concatenación, etc.).

- **SbTime:** Representación de valores de tiempo en alta resolución. Los valores son independientes del sistema. Representación de segundos, μs o usando estructuras propias denominadas *timeval*. Ofrece operadores sobrecargados, de comparación, incremento/decremento, asignación, etc.

- **SbViewportRegion:** Contiene información que representa una zona de la ventana. Ofrece métodos para consultar y manipular coordenadas normalizadas o coordenadas a nivel de pixel.

- **SbViewVolume:** Almacena información de vista de volumen 3D.

- **SbVec*nx*:** Vector para representación de puntos o direcciones 2D/3D, donde "*n*" indica el número de valores contenidos (2, 3 o 4) y "*x*" el tipo de dato ("*f* " para *float* y "*s*" para *short int*).

En Coin3D todos los tipos de datos básicos, propios de la librería, comienzan con el prefijo "*Sb"*, tal y como se aprecia en la lista anterior.

Aunque la librería incluya este tipo de datos básicos, no es necesario, aunque sí muy recomendable, su uso. Un ejemplo sencillo, si deseamos crear dos puntos, que representen cada uno una coordenada en el espacio 3D, podríamos utilizar el tipo de dato básico propio de Coin3D:

```
punto3d_1 SbVec3f(1.0, 2.0, 3.0);//se crea un dato
                                  //compuesto internamente por 3
                                  //floats con los valores 1.0, 2.0
                                  //y 3.0
punto3d_2 SbVec3f ( 4.0, 5.0, 6.0 );
```

Pero nada impide definirlo usando un arreglo de *floats*. Por ejemplo:

```
float punto3d_1[3]={1.0, 2.0, 3.0};
float punto3d_2[3]={4.0, 5.0, 6.0};
```

Suponiendo que se necesite realizar la suma de ambas coordenadas y almacenar el dato en la primera de ellas, en la primera alternativa bastaría con la siguiente sentencia:

```
punto3d_1+=punto3d_2;
```

en el segundo caso se necesitaría acceder directamente a cada uno de los 3 valores de cada arreglo y sumar sus componentes una a una, ya sea mediante un acceso directo, mediante un bucle o punteros:

```
Punto3d_1[0]+= Punto3d_2[0];
Punto3d_1[1]+= Punto3d_2[1];
Punto3d_1[2]+= Punto3d_2[2];
```

En la primera alternativa no se crea una variable, sino que se crea un objeto de la clase SbVec3f, que permite los beneficios que representa la programación orientada a objetos

(encapsulamiento, herencia, sobrecarga de operadores, etc.), conteniendo métodos que permiten realizar operaciones de una manera sencilla y eficaz. De ésta manera, se obtiene un código más compacto y se reduce el tiempo de desarrollo de las aplicaciones.

2.3.7 Fields vs Tipos de Datos

Los campos (*fields*) son estructuras que almacenan valores en los nodos. Los campos están siempre contenidos dentro de los nodos y contienen valores derivados de los tipos básicos con los que trabaja la librería.

Según el tipo de dato que contienen los campos, se pueden diferenciar entre:

- Un campo de valor simple que contiene un solo dato básico "*Sb"*. Para identificar el tipo de dato, como de valor simple, se incluyen las letras "*SF"* de *single field*. Por ejemplo:

SoSFBool.	Contiene un solo dato de tipo *SbBool*.
SoSFFloat.	Contiene un solo dato de tipo *float*.
SoSFVec3f.	Contiene un solo dato de tipo *SbVec3f*
SoSFRotation.	Contiene un solo dato de tipo *SbRotation.*
SoSFName.	Contiene un solo dato de tipo *SbName.*
SoSFColor.	Contiene un solo dato de tipo *SbColor.*

La gran mayoría de campos de valor simple poseen una función o método, *setValue( )* y *getValue( )*, para establecer y obtener su valor. En los campos, dónde esté definido, también se puede usar el operador de asignación "= ".

Un ejemplo de asignación de valores, en un campo de valor de simple, seria el del siguiente ejemplo. En él se establece a 1 el valor del campo simple *height*. Éste campo pertenece al nodo de tipo cámara (*SoCamera*).

```
camara->height.setValue(1.0);
camara->height = 1.0; //siempre que esté definido el
                      //operador =
```

Asimismo, para obtener el valor de un campo, se usa el método *getValue( )*:

```
float resultado = camara->height.getValue();
```

Otros métodos permiten hacer espacio dentro de un *array* existente. Por ejemplo, se supone que el campo *transparency* del nodo *SoMaterial* contiene un arreglo de 14 elementos de tipo *float*. Un ejemplo de inserción de valores dentro del *array* seria el siguiente:

```
float newValues[2];
newValues[0] = 0.1;
newValues[1] = 0.2;
            //Primero hacemos espacio; después de ésto
            //myMtl->transparency[10]
            //y myMtl->transparency[11] tendrán
            //valores arbitrarios.
```

```
myMtl->transparency.insertSpace(10, 2);

// Se establecen los nuevos valores.
myMtl->transparency.setValues(10, 2, newValues);
```

Para borrar elementos existen métodos como *deleteValues( )*, que además vuelven a organizar los valores dentro del array. Siguiendo con el ejemplo anterior:

```
MyMtl->transparency.deleteValues(8, 2);
```

Después de ejecutar la sentencia anterior los valores contenidos en la posición 8 y 9 del array son eliminados. El array se ordena automáticamente moviendo todos los datos por encima del índice 9, hacia abajo, 2 posiciones. De esta manera rellena el espacio sobrante y da como resultado un array de 12 elementos.

- Un campo de valores múltiples contiene un array de datos de valor simple, de tipo "*Sb"*. Para identificarlos se incluyen las letras "*MF"* de múltiple *field*. Por ejemplo:

  SoMFBool, *SoMFFloat*, *SoMFVec3f*, *SoMFColor*.

La manera más habitual de establecer valores en campos de valor múltiple es usando el método *setValues( )*. La estructura general del método adquiere la forma siguiente:

```
NodeName->fieldName.setValues(starting index, number of
                        values, pointer to array of values);
```

El método *setValues( )* facilita el uso de arrays reorganizando el tamaño y los valores del mismo cuando sea necesario.

En el siguiente ejemplo se muestra como establecer valores que contienen más de un dato de tipo *float*. El campo *transparency*, del nodo *SoMaterial*, es del tipo *SoMFFloat*, el cual contiene más de un valor de tipo *float*.

```
SoMaterial *mtl=new SoMaterial;
float vals[3];
vals[0] = 0.2;
vals[1] = 0.5;
vals[2] = 0.9;
mtl->transparency.setValues(0, 3, vals);
```

El siguiente ejemplo muestra como establecer valores para campos que requieren más de un dato de tipo *Sb*. Éste es el caso del campo *diffuseColor*, del nodo *SoMaterial*, que requiere valores del tipo *SoMFColor*:

```
SoMaterial *mtl=new SoMaterial;
SbVec3f vals[3];    //tipo de dato propio de la librería
                    //que contiene 3 datos de tipo
                    //float.
vals[0].setValue(1.0, 0.0, 0.0);
vals[1].setValue(0.0, 1.0, 0.0);
vals[2].setValue(0.0, 0.0, 1.0);
mtl->diffuseColor.setValues(0, 3, vals);
```

Si interesa modificar solo un elemento del array existen métodos, como por ejemplo *Set1Value( ),* que permiten modificar el valor de un solo elemento del array, sin perder el resto de la información, tan solo indicando la posición mediante un índice y el valor a establecer.

En el archivo *help* que se distribuye con la librería se incluye una extensa referencia sobre éstos y otros métodos que resultan útiles para diversas situaciones.

Capítulo 3. Introducción a la Termometría

3.1 Fundamentos Físicos.

La termografia es una técnica que posibilita conocer la temperatura de un objeto o conjunto de objetos mediante la captación del flujo de radiación que emiten, sin necesidad de contacto físico y en tiempo real. Cualquier objeto que se encuentra a una temperatura por encima del cero absoluto (-273 ºC) emite energía electromagnética.

Para que un cuerpo emita energía por radiación no es necesario que se caliente. Ésto posibilita que se pueda aproximar su temperatura mediante la captación de la energía que emite, ya que ésta guarda relación con su temperatura.

La energía electromagnética se divide en diferentes bandas o regiones de longitud de onda, tal y como se aprecia en la figura 7.

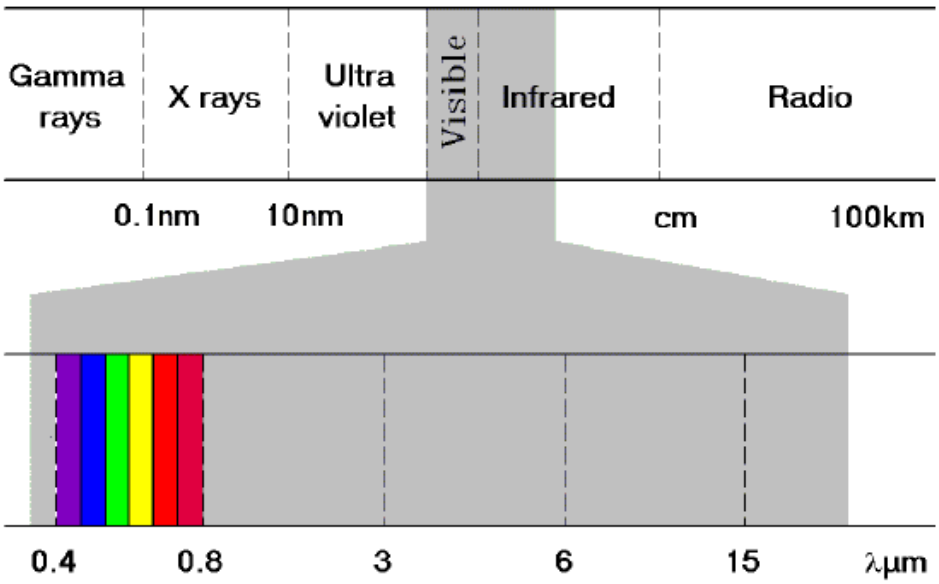


Figura 7. Espectro electromagnético

El ojo humano solo es capaz de captar radiaciones electromagnéticas en el rango de 0.4 a 0.8 μm. El rango de las radiaciones térmicas se sitúa entre 0,1μm y 100 μm (este rango incluye tanto los UV como el de la luz visible).

Conforme un objeto se vuelve mas caliente, la radiación emitida por el mismo alcanza su pico a una frecuencia electromagnética de longitud de onda cada vez más corta, llegando incluso a verse un cambio de color del objeto. Si el objeto supera, aproximadamente, los 600 ºC, la energía irradiada por el objeto se extiende y comienza a emitir en el espectro visible. Se podría apreciar entonces un cambio en el color de su superficie, comenzando por las tonalidades de rojo, luego amarillas, verdes y por último azules. Éste efecto se puede apreciar en los métales cuando se encuentran incandescentes.

La radiación infrarroja, que un objeto irradia al ambiente, es generada por el movimiento de los átomos y moléculas de su material, cuando éste se encuentra a

temperaturas superiores a las del cero absoluto (0 K). Así, ante un aumento de temperatura del material, este movimiento se hace mayor, lo que se traduce a su vez en un aumento de radiación emitida.

Los materiales aparte de emitir energía infrarroja pueden reflejarla, absorberla y en algunos casos transmitirla. La cantidad de energía radiada por un objeto depende, aparte de su temperatura, de su emisividad y de su entorno, ya que a la vez que emiten energía, también la absorben del medio. La ley de *Kirchhoff* establece que: un objeto se encuentra en equilibrio térmico con su entorno cuando la cantidad de energía absorbida coincide con la emitida.

3.1.1 Cuerpo Negro y Emisividad

La energía radiante que incide sobre un objeto, se puede disipar de tres formas distintas: absorción (a), reflexión (r) y transmisión (t).
De acuerdo con la teoría de conservación de la energía, la capacidad de los objetos de transmitir, reflejar y absorber la energía infrarroja se conoce como emisividad del material.

Para definir la emisividad se necesita atender primero al concepto de cuerpo negro. Se entiende como un cuerpo negro a aquél que tiene la capacidad de absorber toda la radiación térmica que incida sobre él. El cuerpo negro no existe en la realidad, se trata de un objeto ideal en el cual las capacidades de absorción y emisión de radiación no tienen pérdidas.

La emisividad de un objeto esta definida por la expresión:

$$e_o = \frac{W_o}{W_{cn}} \qquad (1)$$

donde

W_o= Energía radiante emitida por un cuerpo que se encuentra a una cierta temperatura T.

W_{cn}= Energia radiante emitida por un cuerpo negro que se encuentra a la misma temperatura T.

Si toda la energía que incide sobre el objeto es absorbida, no hay transmitida ni reflejada. Cuando el objeto se encuentra en equilibrio térmico toda la energía absorbida se vuelve a emitir. Por tanto, en un cuerpo negro :

$$a = e = 1 \qquad (2)$$

En la realidad ningún objeto es un emisor ideal, por tanto se necesita definir la totalidad de energía incidente sobre él con la expresión que relaciona la absorbancia, reflectancia y transmitancia:

$$a + r + t = 1 \qquad (3)$$

La energía radiada por un cuerpo negro esta descrita por la ley de *Planck*:

$$W_\lambda = \frac{C_1}{\lambda^5\left(e^{C_2/\lambda T}-1\right)} \qquad (4)$$

donde

$c_1 = 3{,}7418 \cdot 10^4 \ / cm^2 \cdot \mu m^4$

$c_2 = 1{,}4388 \cdot 10^4$ μmK

K=constante de Boltzmann

λ=longitud de onda (μm)

T=temperatura absoluta (K)

W_λ= en W/cm^2·μm

Integrando la ecuación anterior entre 0 e infinito se obtiene la tasa a la cual irradia energía un cuerpo negro. La expresión recibe el nombre de ecuación de *Stefan-Boltzmann*:

$$W = e\sigma T^4 \qquad (5)$$

donde

e = emisividad del cuerpo (0<*e*<1).

σ = constante de Stefan-Boltzmann.

T= temperatura absoluta (en K)

W= en Watt/m^2

Para un cuerpo negro *e*=1. En el caso real, *e* será siempre inferior a 1.

La temperatura de un cuerpo negro se puede obtener directamente de la energía radiada por el mismo. Para encontrar la longitud de onda a la cual acontece el máximo de radiación emitida, se deriva e iguala a 0 la ecuación de *Planck*.

$\lambda_m T = 2897.8$ μm·K Ley de desplazamiento de Wien.

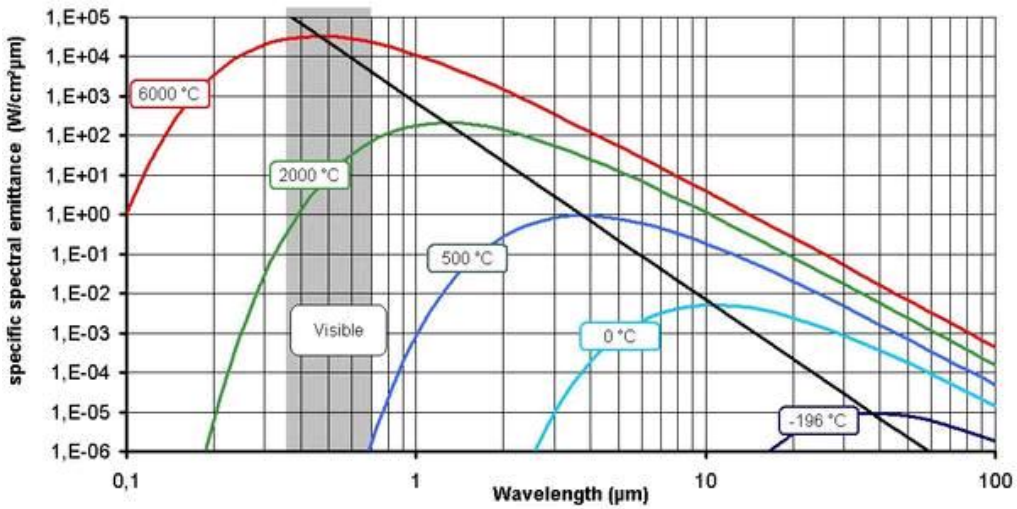


Figura 8. Energía radiante emitida por un cuerpo negro a diferentes longitudes de onda.

En la figura 8 se aprecia como conforme se incrementa la temperatura, la radiación emitida se desplaza hacia longitudes de onda más cortas, hasta alcanzar, como se ha dicho anteriormente, la zona de luz visible en torno a los 600 ºC.

3.2 Importancia de la Emisividad en las Medidas de Temperatura

Resumiendo lo anterior. Aunque un cuerpo negro es un objeto ideal, cualquier objeto puede aproximarse a su comportamiento considerando las relaciones descritas en las ecuaciones 2 y 3.

Para una correcta lectura de la temperatura de un objeto sería necesario considerar a las diversas variables que intervienen en el proceso, es decir: conocer la radiación emitida por el objeto, no solo la cantidad, sino también la longitud de onda a la cual emite, la temperatura ambiental, la emisividad del material que constituye el objeto (en especial la superficie) y la posición relativa de otros objetos, de diferente temperatura, que se puedan encontrar entorno al objeto o zona de interés.

En la práctica este proceso se suele simplificar y se consideran, como las variables más importantes, la emisividad y la temperatura ambiente.

Material	Emisividad	Condiciones (ºC)
Acero brillante	0.18	20
Acero galvanizado	0.28	20
Bronce pulido	0.1	50
Latón brillante	0.03	200
Cobre	0.07	20
Aluminio brillante	0.04-0.06	50-100
Acero Oxidado	0.85	200-600
Aluminio Oxidado	0.20-0.30	50-100
Latón Oxidado	0.61	200-600
Hierro pulido	0.21	200
Titanio pulido	0.40	200
Tungsteno	0.05	200
Zinc pulido	0.04-0.05	400
Zinc oxidado	0.50-0.60	1000-2000
Arcilla	0.91	70
Cemento	0.90	25
Barniz blanco	0.8-0.95	40-100
Barniz térmico	0.92	100
Asfalto	0.90	50
Papel	0.70-0.90	20
Caucho	0.95	20
Hielo	0.98	0
Arena	0.60	25
Marmol pulido	0.93	20

Figura 9. Listado de valores de emisividad de diferentes materiales.

Tal y como se aprecia en los valores de la tabla anterior, cuanto más reflectante sea el material del objeto del cual se quiera medir su temperatura, menor será su capacidad emisiva y viceversa. La emisividad de los materiales se obtiene utilizando fuentes cuyas características se acerquen, lo máximo posible, a las de un cuerpo negro y aplicando la expresión 1.

Por tanto, en la práctica, para una correcta lectura de la temperatura de un objeto, no basta con captar el flujo de radiación que emite, sino que es necesario también conocer la emisividad de dicho objeto.
Este hecho lleva a situaciones prácticas en las que, por ejemplo, dos objetos de diferente material, calentados a la misma temperatura y situados en el mismo medio con la misma temperatura ambiente, proporcionen medidas de temperatura diferentes.

La termografía obtenida en una situación como ésta, representaría a un objeto (baja emisividad) con un color mucho más tenue que el otro (alta emisividad), dando la impresión de que se encuentran a temperaturas diferentes, cuando en realidad se encuentran a la misma temperatura y bajo las mismas condiciones ambientales.

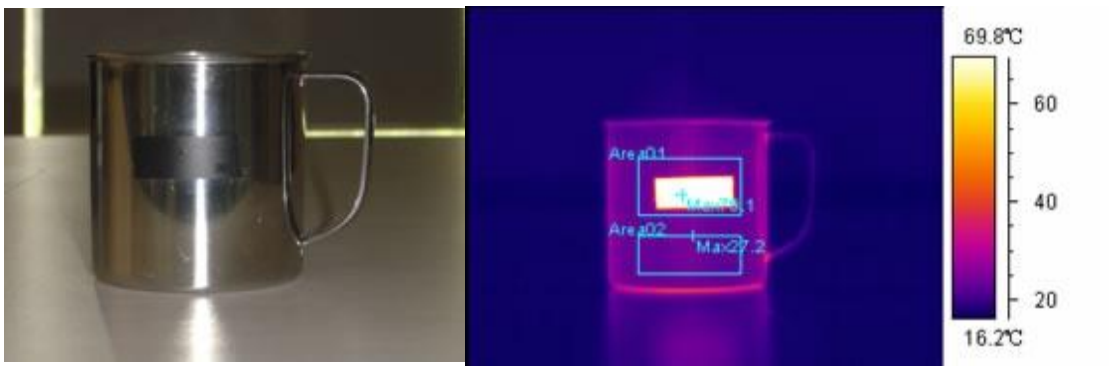


Figura 10. Efecto de la emisividad de diferentes materiales en la imagen termográfica.

3.2.1 La Emisividad en las Medidas Prácticas .

La emisividad se puede aproximar, en la práctica, mediante:

1. Utilizar las tablas de emisividades que proporcionan los valores a aplicar para distintos materiales. Éstas deberían de usarse solo si se conoce exactamente el material del objeto en cuestión. Como norma general solo se aconseja el uso de tablas como valor de referencia inicial.

2. Mediante el uso de un termómetro de contacto, se constata que el objeto se encuentra en equilibrio térmico y que la fuente de tipo cuerpo negro se encuentra a la misma temperatura. Se procede, entonces, a medir la radiación emitida por cada uno y se obtiene la emisividad del objeto mediante la definición teórica de la misma:

$$e_o = \frac{W_o}{W_{cn}} \qquad (6)$$

3. Se calienta la fuente de tipo cuerpo negro y el objeto a la misma temperatura. Después se procede igual que en el caso anterior.

Desgraciadamente los métodos anteriores son considerados métodos de laboratorio, ya que en la práctica no se suele tener acceso a fuentes del tipo cuerpo negro.

En la práctica las alternativas para poder conocer la emisividad, del objeto del cual se quiere conocer la temperatura, se dividen básicamente en:

- **Pintar la superficie del objeto con pintura térmica o recubrirla con cinta de alta emisividad.**
  Esta pintura suele ser de color negro y su uso es aconsejado solo para medir un cierto margen de longitudes de onda de radiación IR.

Elemento	Emisividad	Margen longitud de onda medida
Pintura térmica de color negro	0,96	8-13 μm
Pintura térmica de color negro	0,95	3-5,3 μm
Cintas térmicas	0,94	8-13 μm

Figura 11. Pinturas y recubrimientos industriales.

Este método es más efectivo aplicado a objetos con baja o muy baja emisividad. El inconveniente principal que presenta es que no siempre es posible pintar/cubrir la zona de interés.

- **Corregir el valor medido de temperatura**.
  La mayoría de cámaras termográficas actuales poseen un ajuste de compensación de la emisividad que puede ser regulado a voluntad.

Los procedimientos de obtención de la emisividad requerirán de mayor atención según el tipo de uso que se vaya a hacer de la termografia obtenida con el instrumento de medida.

Por ejemplo, en usos de mantenimiento y localización de fallas el valor de emisividad muchas veces pierde importancia, debido a que las detecciones de fallas se hacen por simple comparación entre componentes similares que se encuentran en condiciones ambientales también similares.

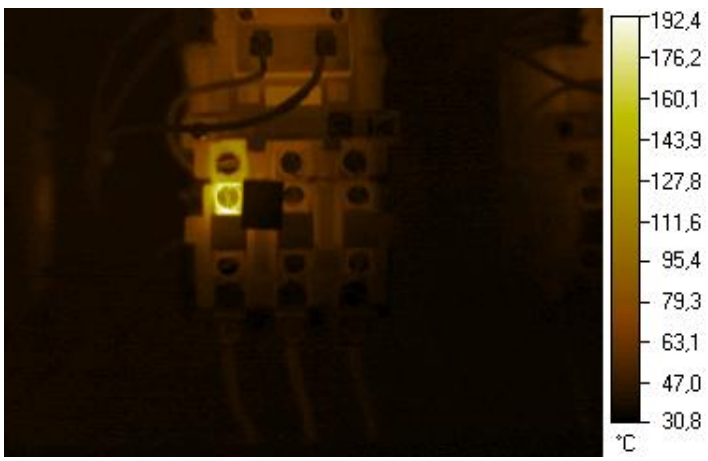


Figura 12. Detección de fallas en un proceso de mantenimiento

Tal y como se aprecia en la figura anterior el componente conflictivo destaca sobremanera del resto de componentes que le rodean. En estos casos, como se ha dicho, conocer el valor de emisividad no tiene tanta importancia.

Un caso contrario seria el del uso de la termografia en ciertas aplicaciones, por ejemplo, en el campo de la medicina o en procesos de desarrollo I+D, dónde la temperatura juegue un papel principal (desarrollo de discos para frenos). En este tipo de aplicaciones no sólo es necesario detectar diferencias de temperatura sino que es igual o más importante que el valor de ésta sea el más cercano posible a la realidad.

3.2.2 Ruido de Fondo o Ambiental

No sólo los rayos IR del objeto de interés entran por el sensor de los termógrafos, sino también aquellos radiados por todos los objetos de su entorno. En situaciones donde sea importante evitar este tipo de perturbaciones, la influencia de este ruido de fondo puede ser reducida adoptando una serie de medidas:

- Reducir la distancia entre el objeto medido y el instrumento de medida, sin que ello signifique sacrificar la distancia de seguridad mínima entre el operario y el objeto o zona de interés.

- Evitar, en la medida de lo posible, la incidencia de luz solar sobre el objeto o zona de interés.

- Evitar las medidas en ambientes con partículas en suspensión, tales como polvo o vapor, que puedan provocar mediciones incorrectas.

- Aislar el objeto o zona de interés de posibles fuentes térmicas que se encuentren en su entorno más cercano.

3.3 Aplicaciones de la Termografia.

La obtención y análisis de la temperatura de objetos o entornos de interés es útil en multitud de situaciones. Actualmente, la termografia se usa en campos tan diversos como:

- Mantenimiento preventivo en entornos industriales.
- Seguridad y vigilancia.
- Detección de incendios.
- En el campo militar (detección y reconocimiento de objetos y personas).
- En la industria alimentaria.
- Inspecciones mecánicas.
- En medicina y veterinaria.
- Detección de fallos en aislamientos.
- En imágenes geoespaciales tomadas por satélites.
- etc.

3.4 Instrumentos de Medición.

Los instrumentos de medición de radiación IR más usados, son las llamadas cámaras termográficas. Éstas cámaras se asemejan en su apariencia externa a las cámaras digitales normales. Poseen diferencias que las hacen especiales para la detección y medición de las radiaciones IR.

Las cámaras termográficas se clasifican habitualmente según las bandas de longitud de onda a las que son más sensibles. En general se distinguen entre:

- Onda corta o infrarrojo cercano (*Near InfraRed*, NIR o *ShortWave InfraRed*, SWIR).
  Se refieren a la banda de radiación IR situada entre 0.8 μm y 2.5 μm.

- Onda media (*Mid Wave InfraRed*, MWIR).
  Para radiaciones entre 2.5 μm y 5.5 μm.

- Onda Larga (*LongWave InfraRed*, LWIR).
  Entre 8 μm y 12 μm.

Las cámaras termográficas consiguen captar las radiaciones infrarrojas gracias al uso de unos sensores especiales que se suelen clasificar en dos categorías: térmicos y de detección de fotones.

Ambos tipos de sensor se basan en diferentes mecanismos para transformar las radiaciones. Mientras los sensores térmicos generan una señal cuando la temperatura a la que se encuentran varía, los detectores de fotones se basan en generar una señal en proporción al número de fotones que inciden sobre él.

En la actualidad la tendencia habitual es el uso de FPA (*Focal Plane Array*), que se compone de una matriz de detectores individuales.

El proceso que origina la lectura de la temperatura en pantalla, se puede resumir, de manera simplificada, como el siguiente:

A. La radiación emitida por un objeto pasa por unas lentes que eliminan las longitudes de onda de la luz visible y dejan pasar solo las correspondientes a la región IR del espectro electromagnético.

B. Seguidamente, para evitar las perturbaciones, se utilizan filtros pasa banda que dejan pasar solo la radiación situada en la longitud de onda de interés. Una vez que la radiación ha sido filtrada, ésta incide sobre cada uno de los sensores individuales que conforman la matriz.

C. Los sensores transforman la radiación en una señal eléctrica que es procesada, amplificándola y comparándola con sensores internos de la cámara, para obtener el valor de temperatura. Basándose en los valores de temperatura, se genera una

imagen térmica mediante la asignación de un color (generalmente un tono de gris) proporcional al valor leído.

La imagen de la figura 13 muestra de forma gráfica el proceso explicado.

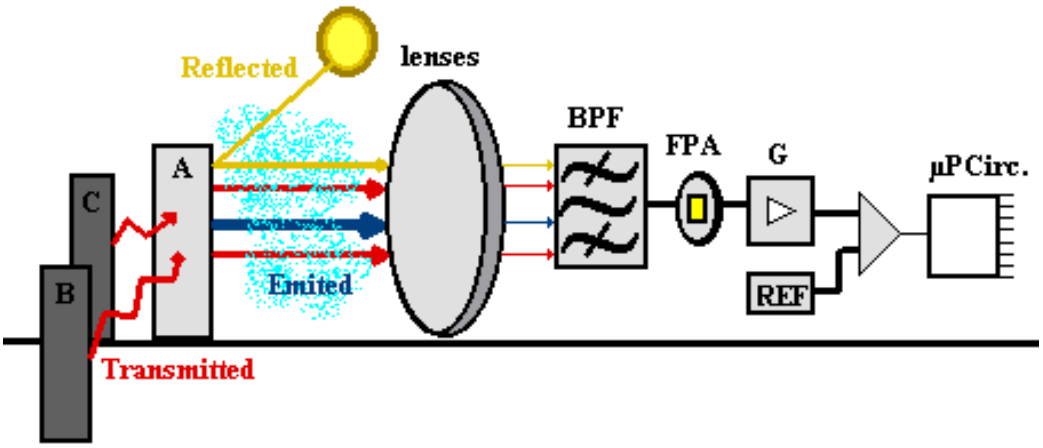


Figura 13. Esquema de bloques de una cámara termográfica.

Los sensores de la cámara necesitan de una refrigeración criogénica siendo al principio proporcionada mediante el uso de nitrógeno líquido.

El esfuerzo en los modelos posteriores se centró en eliminar esta dependencia del nitrógeno y se usaron las alternativas ofrecidas por el efecto *Peltier* (únicamente en cámaras para onda corta) y el ciclo *Stirling*.
A finales de la década de los 90 es cuando comenzaron a surgir modelos, de cámaras termográficas basadas en sensores, que funcionaban a temperatura ambiente.

Otra de las características importantes es la que indica la máxima distancia a la cual se pueden obtener medidas de temperatura. Ésta viene determinada por la llamada resolución espacial, que se define como la relación entre el tamaño del sensor y la separación entre el propio sensor y las lentes de la cámara.
La relación entre ambas es inversa, es decir, a menor separación mayor será la distancia a la cual es posible realizar medidas, y viceversa.

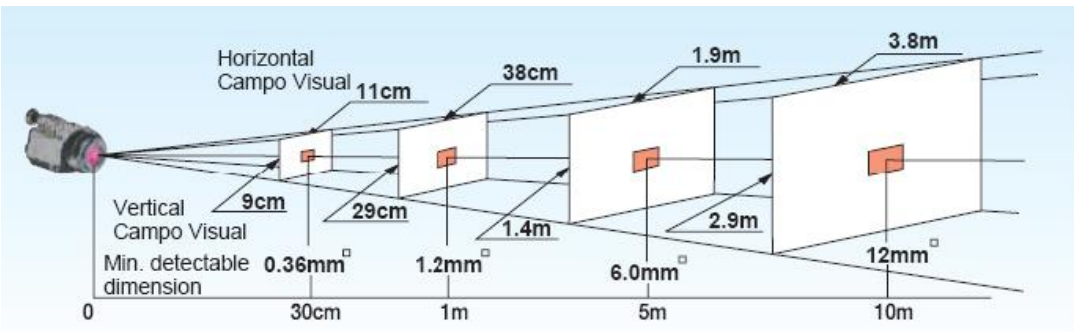


Figura 14. Resolución espacial de una camara termográfica

3.4.1 Características de las Cámaras Termográficas.

A modo ilustrativo, se presentan las principales características de una cámara termográfica[2].

Rango de temperatura:	3 rangos: -20 a 100 ºC, 0 a 250 ºC, 100 a 800 ºC
Precisión:	2 ºC o 2% del valor leído (el mayor de ambos)
Velocidad de adquisición:	60 imágenes/s.
Resolución:	0.06 ºC en condiciones de 30 ºC a 60Hz.
Tipo de detector:	FPA (microbolómetro) de 320x240 pixeles.
Enfoque:	de 30cm a infinito.
Resolución A/D:	14 bits.
Corrección de emisividad:	Variable de 0.1 a 1 (incrementos 0.01)
Compensación ambiental:	incluida.
Visualización:	Display color: color/monocromo. Gradación: 16, 32, 64, 128 y 256 colores. Paletas de color: Rainbow, hot-iron, medical, fine, shine, brightness.
Isotermas:	máximo 4 bandas.
Interface de comunicación:	RS-232C, IEEE1394.
Almacenamiento de datos:	imagen térmica: formato SIG o BMP. Imagen visual: formato SIG o BMP. Composición entre imagen visual y térmica: formato BMP.

Una de las principales características, común en todas las cámaras, es la capacidad de almacenar la información en distintos formatos. Los formatos de almacenaje de información se pueden dividir en 3 categorías:

- **Datos en formato binarios.**
  Se trata de un formato de almacenaje compacto que incluye los valores de temperatura de cada uno de los sensores que conforman la matriz FPA de la cámara.
  Éstos datos se pueden transferir a un ordenador mediante la comunicación por interface RS-232, IEEE1594, FireWire (six pins), RJ-45 Ethernet.

- **Datos en formato bitmap.**
  En esencia no se transfieren los datos de temperatura capturados por la cámara, sino que se trata de un fichero de imagen en alguno de los formatos gráficos más habituales. La imagen se puede considerar como una representación térmica del objeto. Las tonalidades de colores varían según las paletas de color elegidas por el usuario. Las imágenes también son transferibles mediante los mismos sistemas de comunicación que en el caso anterior.

- **Datos en formatos propietarios.**
  Las empresas más importantes utilizan su propio formato de archivos. Aunque el uso de estos formatos pueden provocar que programas de pos-procesamiento no puedan leer alguno de los mismos, ofrecen ciertas ventajas respecto a los dos

[2] modelo NEC TH9100PMV

sistemas de almacenaje anteriores, por ejemplo: capacidades para incluir una pequeña grabación (la cual se asocia al mismo archivo de la termografia), capacidades de fusionar la imagen de luz visible con la imagen termográfica, etc.

Otro aspecto a destacar es la resolución en bits del conversor de la cámara. Este dato nos informa de la capacidad de la cámara para distinguir los incrementos de temperatura. Para el valor de 14 bits, visto en las características anteriores y considerando la resolución de 0.06 ºC, se obtiene una capacidad de digitalización de temperaturas de 2^{14}, lo cual nos proporciona una escala de temperaturas con un rango absoluto de 983,04 ºC.

A cada valor de temperatura detectado por la cámara, en cada una de las escalas disponibles, se le asigna un color con una tonalidad de gris, o el color correspondiente, de la paleta de colores. En los datos anteriores se observa que la máxima profundidad de colores de la cámara es de 256 tonos, por tanto se obtiene una imagen térmica de 8 bpp. Así, partiendo de una resolución inicial de 14 bits, se ha pasado a una representación de los datos usando una resolución de 8 bits (en color). Aunque puede parecer una pérdida de datos, ésta, en verdad, no existe debido a que internamente la cámara sigue manejando datos de temperatura de 14 bits y es solo la representación visual la que se realiza a 8 bits.

Capítulo 4. Diseño y Desarrollo de una Aplicación de Representación Tridimensional de Imágenes Termográficas.

4.1 Estudio Previo.

Tal y como se ha comentado en el capitulo anterior, las cámaras termográficas tienen la capacidad de almacenar las imágenes térmicas en diferentes formatos. Los más comunes son: JPEG, BMP, GIF, PNG, WMF, EMF. También existen los formatos específicos como el IS2 (con capacidades de fusión entre imagen térmica y normal), IRI, SIT, etc.

Comúnmente, éstos formatos almacenan la información internamente como números decimales con precisión simple (4 bytes), junto con archivos de audio (de duración limitada) y información referente a fecha y condiciones de toma de la imagen termográfica. Todo empaquetado en un único archivo, con las extensiones de archivo especificas, antes mencionadas.

Los datos de precisión simple, que en C tendrían su equivalente inmediato en datos de tipo *float*, son luego cargados por el programa de pos-procesamiento, asignándole a cada incremento de temperatura un índice a un color de una escala o paleta de colores de 8 bits.

Esto supone que, aunque los puntos que componen la imagen en pantalla devuelvan un valor de temperatura igual al almacenado por la cámara, la imagen generada y presentada es en realidad una imagen de 8 bits. De ésta manera se facilita la manipulación de la imagen, puesto que el direccionamiento de 1 byte por píxel, en los archivos de imagen, es el más fácil de manejar. Éstas imágenes generadas pueden ser luego exportadas en el formato elegido por el usuario.

4.1.1 Estudio ficheros de Imagen (8 bpp).

El inconveniente que se presenta de manera más habitual en este tipo de imágenes es el contenido inapropiado de la paleta de colores. Para poder obtener los datos de temperatura, es necesario que la paleta guarde relación directa con los datos que representa. Por tanto, es necesario que la paleta responda a algún tipo de orden lógico de colores.

Existen aplicaciones, de tratamiento de imágenes, que guardan las paletas dentro del archivo, respondiendo a sus propias necesidades. Un ejemplo de ello sería el de guardar los colores en la paleta según su uso en la imagen, es decir, el color más usado se encuentra en la primera posición o la última, etc.

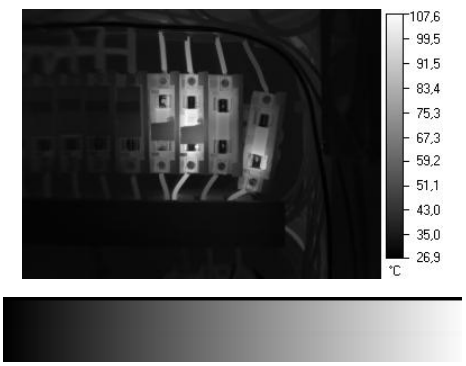


Figura 15. Imagen 8bpp grises (paleta progresiva).

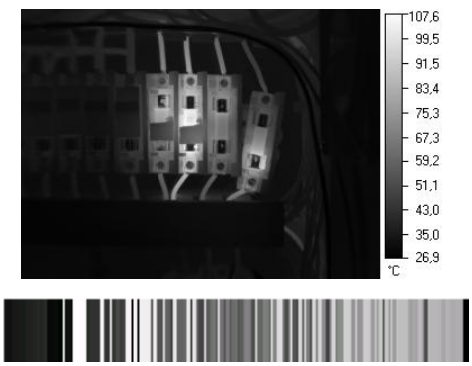


Figura 16. Imagen 8bpp grises (paleta no progresiva)

En las imágenes anteriores se muestran dos imágenes idénticas, pero la paleta de la imagen de la figura 16 no puede ser utilizada de manera directa para convertir los pixeles de la imagen en valores de temperatura, ya que como se aprecia los grises no se encuentran en orden progresivo.

De manera indirecta, y especialmente para los casos de escala de grises, si se podría utilizar la paleta de la figura 16, utilizando para la conversión entre el color y la temperatura, las componentes de color en vez del índice a la paleta.

La diferencia entre el formato en que se guardan las paletas puede ser provocado por conversiones aplicadas a la imagen original. Por ejemplo, al tener una imagen en tonos grises de 16 bpp o más y reducirla a 256 colores en vez de a 256 grises.

Para el caso de conversiones, entre índices a paletas de colores progresivas y valores de temperatura se usan los datos ofrecidos por los pixeles que componen la imagen. Ésta conversión es inmediata. Utilizando como ejemplo las dos imágenes anteriores, un posible proceso a seguir seria el siguiente:

1º Se obtiene el valor del índice de color de cada pixel a la paleta de grises.

2º Conociendo que la temperatura máxima es de 107.6 y la mínima 26.9, para cada índice de la paleta se puede aproximar la temperatura correspondiente al pixel en cuestión mediante la expresión:

$$Tpixel = \frac{(T\,\text{máx} - Tmin.)}{255} \cdot indice + Tmin \qquad (1)$$

Para poner de manifiesto el error que generaría el uso de la imagen2 frente a la imagen1, se supone, por ejemplo, que se lee el mismo pixel en ambas. El pixel leído posee un índice de 255, por tanto en ambas imágenes le correspondería el valor de temperatura máxima, tal y como se obtiene de la expresión anterior.

Pero el resultado en la imagen 2 seria incorrecto, ya que un índice de 255 indica el color negro y tal y como se aprecia en la escala de grados ºC, incluida a la derecha de cada imagen, el índice 255 hace referencia al color blanco.

Es necesario que el usuario conozca este tipo de limitaciones para que no obtenga resultados incongruentes. Una manera de solucionar este tipo de problemas es realizar siempre las conversiones a escala de grises de la manera propuesta.

Una vez que la imagen es correctamente cargada en la aplicación, se puede aplicar diferentes tipos de escalas de color para, por ejemplo, facilitar el visionado de zonas de interés, mejorar el contraste entre franjas de temperaturas, etc.

4.1.2 Estudio Ficheros Binarios.

La interpretación del contenido de este tipo de archivos no posee mayores complicaciones.

Los primeros 4 bytes del archivo indican las dimensiones de la imagen: los 2 primeros el ancho y los 2 siguientes el alto.
Cada conjunto de 4 bytes del resto de datos hacen referencia a la temperatura que se asociaría a cada pixel de la imagen, codificados como *float* y notación *little Endian*.

El procesado de este tipo de ficheros es el inverso al del uso de imágenes. Partimos de un conjunto de valores de temperatura y se necesita generar una imagen representativa de los mismos.

En este caso se tiene que para cada valor de temperatura leído se ha de generar un índice a una paleta de colores. Para simplificar al máximo el proceso se podría utilizar una paleta en escala de grises.

Siguiendo con el ejemplo de la imagen de la figura 15, el proceso a seguir seria:

1º Lectura del valor de temperatura.

2º Conociendo que la temperatura máxima es de 107.6 y la mínima 26.9, el índice correspondiente a cada valor de temperatura se obtiene aplicando:

$$indice = \frac{T_L - T_{min}}{T_{max} - T_{min}} \cdot 255 + 0.5 \qquad (2)$$

donde el subíndice L indica temperatura leída.

Con esta expresión se pone de manifiesto la pérdida de información que se obtiene al obtener la imagen de 8bpp. No sólo por el hecho de que el índice deba ser un entero sino porque 256 valores diferentes pueden llegar a ser insuficientes.

La significación de esta pérdida de datos, en la representación de la imagen obtenida, depende de la aplicación que se quiera hacer de la misma. Es perfectamente válida, por ejemplo, para ciertas aplicaciones de mantenimiento industrial, donde se requiere detección de zonas calientes antes que una gran resolución y resultar insuficiente en casos donde se requiere la medición de temperaturas con una gran precisión, como podría ser el caso de ciertas aplicaciones en el campo de la medicina.

La ventaja de usar este tipo de ficheros es evidente, se trabaja con los mismos datos obtenidos de la cámara termográfica y, al mismo tiempo, se obtiene una representación de la termografia mediante una aproximación utilizando una imagen de 8bpp. Se gana en resolución para los valores de temperatura y en sencillez y eficiencia en la manipulación de pixeles de la imagen.

Como en el caso anterior, una vez generada la imagen se pueden usar diferentes escalas de color.

4.1.3 Estudio Ficheros formato .txi

Este tipo de ficheros son en esencia como un fichero bmp, incluso los caracteres de la firma de la cabecera son los mismos. Por tanto se puede abrir y tratar como un archivo

bitmap (bmp). Las únicas diferencias son las incorporaciones de los datos de temperatura mínima y máxima, los cuales se encuentran, respectivamente, en los *offsets* 6 y 8 de la cabecera.

4.1.4 Consideraciones sobre las Paletas de Color.

Se podría pensar que la aplicación de diferentes paletas de color a la imagen termográfica tiene un efecto puramente estético en la distinción de los datos que la conforman. En verdad el papel que juega no es solo estético.

Cuando se realiza una asignación de color a los datos de temperatura que conforman la termografia (*colormapping*), la imagen obtenida puede pasar a percibirse de manera completamente diferente.

Por ejemplo, el uso de paletas que no ofrezcan una secuencia con sentido para el sistema visual puede llevar a engaño si no se acompaña a la paleta con la escala de temperaturas graduadas. En la figura 17 se aprecian dos imágenes. La de la izquierda utiliza una paleta llamada espectral o arco iris (*rainbow*) que es muy usada en diversos campos (medicina, cartografía, simulación en dinámica de fluidos, astronomía, etc.). Su uso se ha difundido mucho porque los colores siguen su orden físico, es decir, el orden que les corresponde según su longitud de onda. A la derecha se muestra la misma imagen pero con una paleta de las llamadas perceptivas.

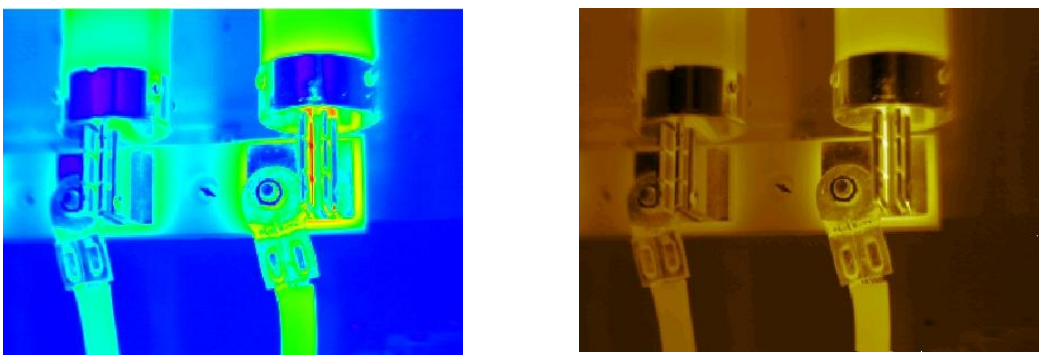


Figura 17. Aplicación de diferentes paletas a una imagen termográfica

Se puede observar que la imagen de la izquierda de la figura 17 puede inducir a error debido a que la transición entre colores no es inequívoca, mientras que en la de la derecha, se intuye de manera directa qué zonas poseen mayor temperatura.
Ésto es debido a que la percepción de luminancia es evidente, lo cual no ocurre con los colores (aunque estén representados en su ordenación física, como es el caso de la paleta *rainbow*).

El programa a desarrollar debe de ofrecer al usuario, en la medida de lo posible, una correcta visualización de la imagen termográfica, esto es, que no le induzca a interpretar erróneamente la imagen mostrada.

Atendiendo a la taxonomía de Stevens, la representación de la temperatura, mediante una secuencia de pseudocolores, correspondería a una escala de medida ordinal. En este

tipo de escala se diferencia, perceptivamente, la ordenación de los valores de los datos asociados a los colores.

Considerando lo anterior, junto a otras reglas[3] de selección para las paletas de color, el programa a desarrollar debe ser capaz de ofrecer al usuario la posibilidad de generar y asignar degradados de color a la imagen termográfica, para facilitar la percepción del usuario en cuanto a variaciones de temperatura se refiere. Igualmente, debe de permitir el uso de cualquier tipo de paleta, como la *rainbow*, para mejorar la distinción entre diferentes zonas de la imagen.

Es el usuario el encargado de seleccionar la paleta que mejor corresponda, según el estudio que desee realizar sobre la imagen termográfica.

En general, se considera apropiado el uso de degradados de color simple con variaciones de luminancia uniforme[4] para la representación de grupos de datos que guarden un orden particular (como es el caso de las termografias). Se aconseja dejar el uso de las paletas no perceptivas para simple diferenciación de zonas específicas de la imagen.

4.1.5 Consideraciones sobre los datos

Durante el desarrollo de la aplicación será necesario considerar ciertos puntos importantes:

- En los archivos de imágenes se considerará que éstos se han obtenido con los valores correctos de emisividad y que la imagen es representativa de los valores de temperatura de los cuales se obtuvo.

- Para facilitar el proceso de desarrollo de la aplicación, se considera que la imagen procesada se ha obtenido usando los mismos valores de emisividad para todos los puntos que la componen.

- Durante el desarrollo de la aplicación se utilizará la notación de color RGB.
  Para las funciones donde sea necesario que el usuario introduzca valores de color se considerará, salvo en casos donde se indique lo contrario, codificación RGB en bytes (valores 0 a 255). Internamente la aplicación hace uso de la notación RGB normalizada.
  Se ha decidido así ya que se ha considerado más común la identificación de los tonos de cada componente como enteros que no mediante el uso de decimales.

4.2 Definición de Objetivos de la Aplicación.

La aplicación que se plantea realizar en el presente proyecto (en adelante Visor3D) debe ser capaz de soportar dos tipos de información diferentes. Por un lado, admitir como ficheros de entrada bitmaps en los formatos siguientes:

[3] Bergman, Rogowitz y Treinish. *A Rule-based Tool for Assisting Colormap Selection*
[4] Todos los degradados de color simple con variaciones de luminancia uniforme se consideran como una sucesión de colores perceptiva.

- Archivo Bitmap de Windows o OS/2 (*.BMP)
- ZSoft Paintbrush PCX bitmap format (*.PCX)
- Independent JPEG Group (*.JPG,*.JIF,*.JPEG,*.JPE)
- "Portable Network Graphics" (*.PNG)
- "Truevision Targa Files (*.TGA,*.TARGA)
- "Tagged Image File Format (*.TIF,*.TIFF)
- "Graphics Interchange Format (*.GIF)
- "JPEG-2000 Codestream (*.J2K,*.J2C)
- "JPEG-2000 File Format (*.JP2)

Por otro lado también debería de admitir archivos binarios de temperatura, en formato 16 bits. En éste caso el proceso es inverso ya que se parte de los datos de temperatura para crear una imagen termográfica de 8bpp.

Usando estos dos tipos de archivos el usuario puede cargar en pantalla una malla en 3D representando la imagen tridimensional de la termografia. Después puede realizar diferentes acciones sobre ella mediante el uso de una serie de comandos, como realizar ciertas transformaciones para facilitar su visualización (rotación, escalado, translación, etc.), operaciones sobre la presentación de la misma (filtros de temperatura, cambios de paleta, inversión de colores, extracción/saturación de canales RGB), permitir la obtención de información de cada punto de la figura, obtención del histograma, entre otras.

La aplicación debe permitir al usuario realizar un volcado de los datos manejados a un fichero en formato texto o binario. De esta manera pueden ser utilizados por programas especializados en tratamientos de datos.

En resumen, la aplicación a desarrollar usará dos tipos de ficheros de entrada de datos:

- Imágenes de mapas de bits.
- Binarios.

Si el usuario utiliza mapas de bits, debe de asumir una pérdida de información, ya que los datos que se pueden extraer de ella poseerán una resolución de 8 bits.

El error en la obtención del valor de la temperatura, en el caso de utilizar imágenes de 8 bpp como fichero de entrada de datos, será más grande cuanto más grande sea el rango de temperaturas a medir.
Si se usan archivos binarios, no habrá pérdida de información en las manipulaciones internas de los datos de temperatura y la imagen generada será una aproximación de 8bpp.

4.3 Diseño de la aplicación Visor3D.

El diseño de la aplicación se ha dividido en los bloques que se muestran en la figura 18. En cada uno de estos bloques se explican las principales clases de objeto y funciones que lo conforman y se muestran las soluciones de configuración de nodos que se han tomado en cada caso.

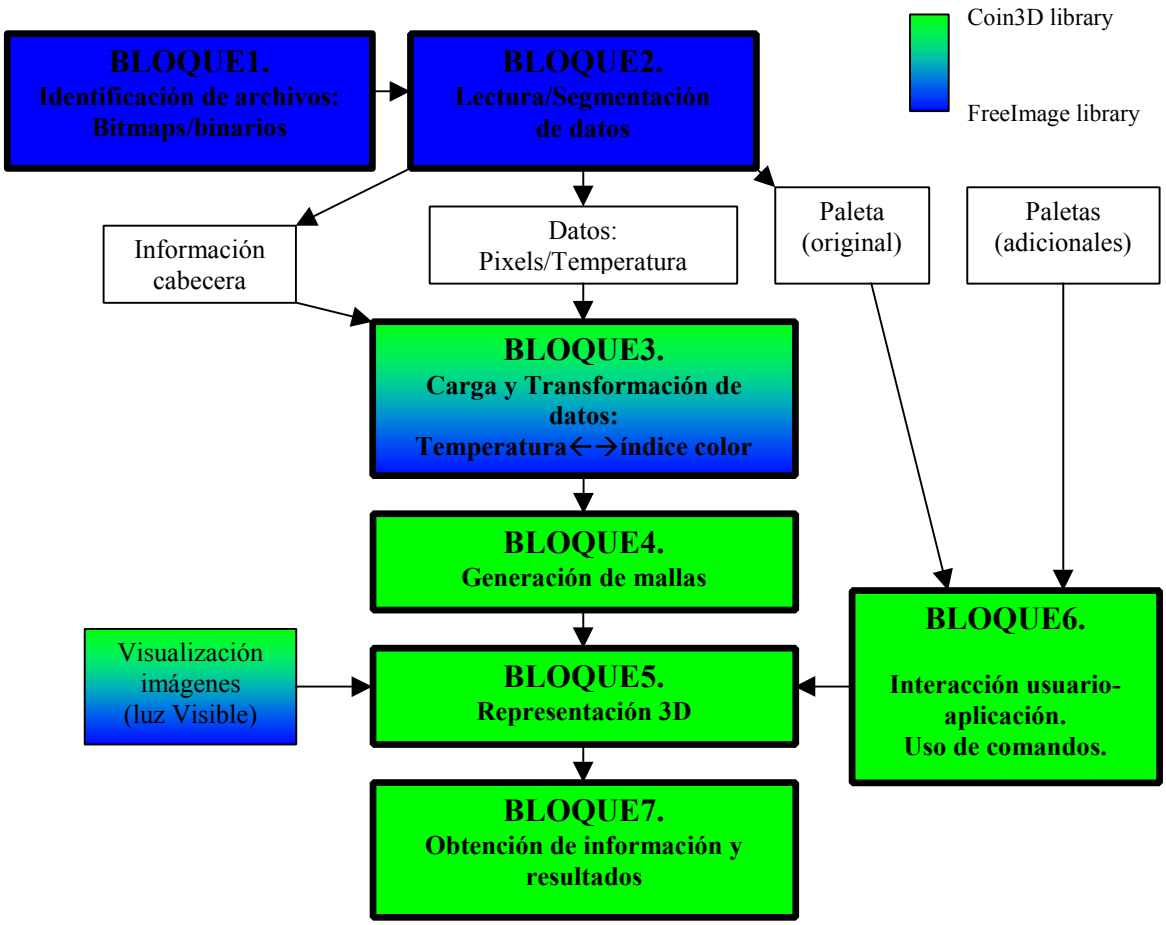


Figura 18. Bloques de división de la aplicación.

- **Bloque 1. Archivos de Entrada**
  Se realiza la recopilación de la estructura e información de los archivos de entrada de la aplicación.

- **Bloque 2. Lectura/Segmentación de los Datos.**

  Separación de los datos en diferentes archivos para facilitar el acceso selectivo a la información.
  Todos los ficheros de entrada se dividen en 3 archivos de texto: información sobre la imagen (bpp, dimensiones, etc.), datos y paleta.

- **Bloque 3. Carga y Transformación de datos**

  En este bloque se desarrollan todas las funciones necesarias para la conversión de los datos de color a temperatura, en caso de archivos de entrada bitmaps o conversión de temperatura a índice en caso de archivo de entrada binario.

- **Bloque 4. Generación de Mallas Basadas en los Datos.**

  Se crean los objetos para la representación de los datos mediante el uso de mallas poligonales.

- **Bloque 5. Representación 3D.**

  Se desarrolla la parte más visible de la aplicación. Se muestran los objetos generados.

- **Bloque 6. Interacción Usuario-Aplicación**

  Incluye todos los comandos de manipulación directa e indirecta de los datos y/o de la malla 3D, por ejemplo:
  Cambios de color asociados a valores de temperatura, inversiones, filtros, selección de zonas de interés, etc.

- **Bloque 7. Obtención de Información y Resultados.**

  Funciones de obtención de información y representación de la misma en el mismo entorno de visualización. Generación de archivos de datos de salida.

4.3.1 Bloque 1. Lectura de Archivos:

Como ya se ha explicado, existen dos tipos de archivos de entrada para la aplicación: archivos de mapa de bits o archivos binarios. A continuación se explica la forma de realizar la lectura de los mismos.

Proceso de lectura de las imágenes termográficas y archivos binarios

Para poder dar soporte a la mayor cantidad posible de formatos gráficos, la aplicación utiliza la librería *open source* FreeImage. Ésta librería proporciona un conjunto de funciones y clases en C++ mediante las cuales es posible la lectura y manejo de una amplia variedad de formatos gráficos.

La versión de FreeImage utilizada en la aplicación desarrollada es la 3.11.00. Ofrece soporte para más de 20 formatos de imágenes diferentes. Gracias a su utilización se ha dotado a la aplicación de soporte para los formatos siguientes: bmp, jpg, jpeg, jif, gif, tif, tiff, pcx, tga, pic y png.

En principio se aceptan imágenes de más 8 bits de profundidad de color. La aplicación permite al usuario abrir imágenes de 1, 4, 8 y 24 bits, pero solo permite utilizar las imágenes de 1 (sin utilidad), 4 y 8 bpp para obtener los datos de temperatura.

Con esto se da la posibilidad de representar los datos de temperatura de la termografía de manera simultánea con una imagen en alta resolución (24 bpp) de la imagen en luz visible, lo cual puede resultar útil en caso de querer identificar zonas específicas en la termografia.

Opcionalmente, se ofrece al usuario la posibilidad de “forzar” a la aplicación a usar imágenes de 24 bpp para extracción de datos. En éstos casos el usuario debe de ser consciente de que la imagen será tratada como una imagen en tonos de gris. Básicamente se considerará que las tres componentes de cada pixel son iguales.

Descripción de las funciones de lectura de archivos:

Función *abrir_bitmap( ):*

Es la función encargada de la apertura de todos los archivos bitmaps introducidos por la línea de comandos. Realiza una comprobación de la signatura y extensión del archivo, si la lectura del archivo esta soportada por la librería FreeImage, abre un archivo y devuelve un puntero a la primera posición del mismo, en caso de error o formato de bitmap no soportado devuelve un puntero nulo.

La implementación de las funciones necesarias para la lectura de los archivos se han incluido en los archivos *leerfichero.h* y *leerfichero.cpp* (ver código fuente adjuntado al final de la memoria).

El desarrollo de las funciones de este bloque es muy limitado, básicamente hace la comprobación en el formato de archivo, identificándolo para su posterior lectura.

4.3.2 Bloque 2. Segmentación de los Datos.

Este bloque está muy ligado al anterior ya que la generación de los archivos temporales se hace mientras se va leyendo el archivo de entrada. Por tanto, en el desarrollo de éste bloque se incluye la gran mayoría de código destinado a la lectura de los archivos.
Las funciones más importantes que lo componen son:

Función *descomponer_fichero( ):*

Es la encargada de segmentar el archivo de entrada y de generar 3 ficheros de datos. Éstos ficheros contienen información referente a la cabecera, datos y paleta de la imagen.

Estos 3 archivos se crean en el mismo directorio desde el cual se ejecuta la aplicación. Se ha decidido utilizar el formato texto para facilitar consultas y modificaciones, por si el usuario lo desea, aunque esto ha significado un aumento considerable en el tamaño de los archivos y un mayor tiempo para la lectura/escritura de los mismos, sobretodo en el archivo que contiene los valores de los pixeles de la imagen (*datosimag.txt*).

El archivo de cabecera llamado *datoscab.txt* contiene la siguiente información:

- Clasificación de tipo imagen utilizada por la librería FreeImage (constantes mostradas en la figura 19).
- Bpp.
- Anchura y altura de la imagen en pixeles.
- Anchura y altura de la imagen en bytes.
- El directorio y el nombre del archivo imagen de la termografia.

Estos datos se almacenan en un objeto de la clase *infFichero* declarada en el fichero *leerfichero.h*

Los ficheros de tipo binario requieren algunas consideraciones ya que solo incluyen información sobre el ancho y el alto de la matriz de datos.

En estos ficheros las informaciones sobre el ancho y alto en bytes se obtienen multiplicando el ancho y alto de la matriz por 4 bytes, que es el tamaño de cada dato *float* almacenado.
Para la información de bpp se asigna un valor de 8, ya que la imagen que se generará será considerada siempre como una imagen de 8 bpp.

El fichero, llamado *datosimag.txt*, contiene la información de los índices a la paleta de colores en el caso de las imágenes con paleta (4 y 8 bpp).
Para imágenes de mayor profundidad de color el archivo contiene las componentes RGB de cada pixel, guardadas en notación *Big Endian* .

El fichero de paleta, *datospal.txt* contiene los colores usados en la imagen en caso de 4 y 8 bpp. En el resto de imágenes este fichero solo contiene una línea con el texto MONOCROMO (imágenes 1 bit) o 24 en el caso de 24 bpp.

Función *DescripcionTipoFich()*

Devuelve una cadena conteniendo la descripción del formato de bitmap. Para ello utiliza las constantes de la librería FreeImage mostradas en la figura 19.

Tipo de fichero	Constante en FreeImage library
Formato no soportado	-1
.bmp	0
.ico	1
.jpg, .jpeg	2
.pcx	10
.png	13
.tga	17
.tif	18
.gif	25
.j2k, .j2c	30
.jp2	31

Figura 19. Constantes usadas de la librería FreeImage

4.3.3 Bloque 3. Carga y Transformación de Datos

En este bloque se engloban las funciones necesarias para obtener la información que posibilitará la posterior creación de las mallas poligonales. Para que la posterior representación 3D sea la correcta es necesario controlar en todo momento las conversiones entre diferentes formatos, sobre todo al realizar *cast* entre diferentes datos.

Clase infFichero.

Los objetos de esta clase son utilizados para almacenar la información de los archivos bitmap o binarios.

Función *CargarInfo().*

La función devuelve un puntero a un objeto de la clase *infFichero*. El origen de los datos se especifica mediante el parámetro *nombreFichero*, realizando las acciones necesarias en caso de ficheros bitmap o binarios.

Función *Cargar_datos().*

Puede considerarse como la función más importante en el caso de archivos bitmap. Se encarga de tanto del almacenamiento en memoria de la paleta de colores, como de los pixeles de la imagen.

En este último caso los pixeles son almacenados de forma lineal en un arreglo de floats que representan las coordenadas que posteriormente serán usadas para la creación de mallas poligonales.

Las coordenadas se generan de manera predeterminada con una separación unitaria, aunque es posible modificar este parámetro según las necesidades del usuario.

Función *Cargar_paleta().*

La función recibe en su primer parámetro el nombre del fichero que contiene la paleta que se cargará en memoria. Para decidir la longitud de la misma se utiliza el parámetro *bpp* (bits por pixel). El valor de *bpp* pasado debe de coincidir con el número de colores almacenados en el archivo. Los únicos valores válidos para *bpp* son 1,4 y 8, para paletas de 2, 16 y 256 colores respectivamente.

Los valores devueltos se almacenan en el tipo de dato *SbColor*, propio de la librería COIN, el cual internamente esta formado por 3 *floats*, el primero corresponde a la componente R del color y los siguientes a la G y B, respectivamente.
Los valores se normalizan antes de guardarlos en el array, por lo que se dividen por el valor de índice más grande de la paleta o *bpp*-1.

4.3.4 Bloque 4. Generación de mallas basadas en los datos.

Tipos de representación de datos en pantalla

Para facilitar el proceso de representación de datos en 3D, se ha considerado siempre el trabajo a nivel de vértices. Cada imagen o conjunto de datos es considerado como una malla de mxn vértices, donde a cada vértice le corresponde un valor de temperatura (valor coordenada Z). Cada 4 vértices forman un grid. La aplicación desarrollada utiliza los siguientes tipos básicos de visualización:

- Representación de la nube de puntos.

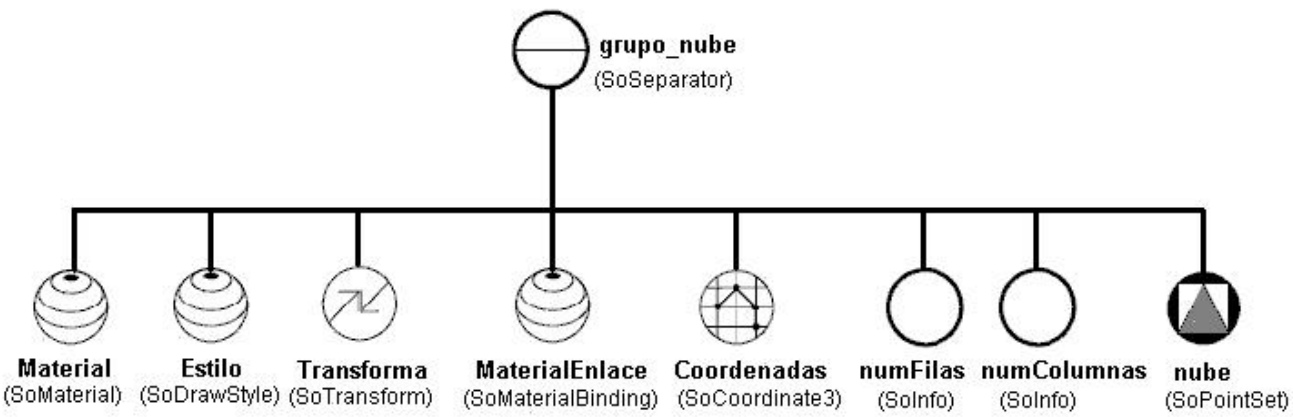


Figura 20. Árbol de nodos para la representación de la nube de datos.

Es el caso más sencillo de representación de los datos. Sencillamente se representan los puntos en las zonas de coordenada correspondiente según la posición del pixel o del valor de temperatura leído, respectivamente en caso de archivos bitmap o de ficheros binarios.

Debido a que el nodo de la clase *SoPointSet* no contiene ningún campo que posibilite el conocimiento del número de filas y de columnas que conforman la imagen, se ha hecho necesario añadir dos nodos de información para contener dicha información.

Esta información es utilizada con posterioridad en situaciones donde se requiere conocer necesariamente estos datos, por ejemplo, en los comandos *cortarxfila*, *cortarxcolumna* y *cortarxarea*, entre otros.

Función *generar_nube()*.

Es la encargada de crear la estructura de nodos de la figura 20 cada vez que es llamada. Los datos necesarios para el nodo Coordenadas son obtenidos del archivo *datosimag.txt*, en el caso de ficheros de entrada de tipo bitmap, o directamente de la lectura del archivo binario, mediante la carga previa en memoria producida por la función *cargar_datos()*.

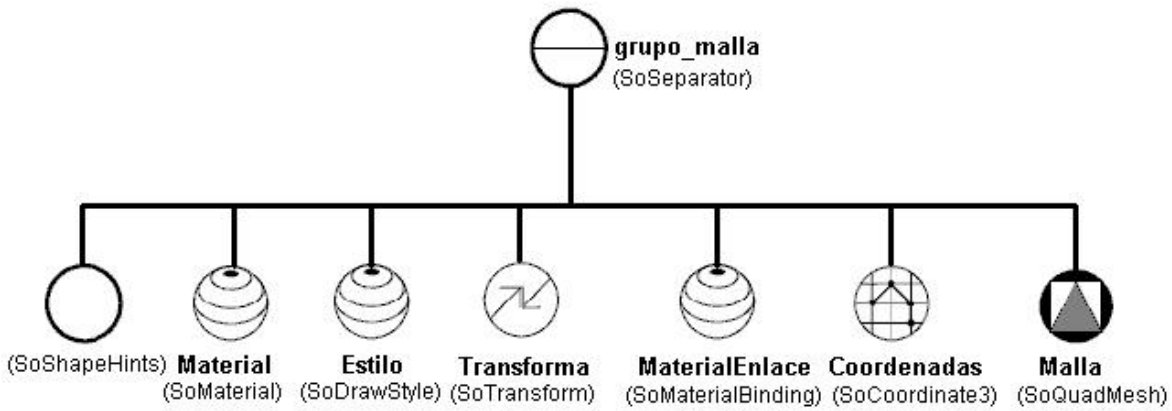


Figura 21. Diagrama o árbol de nodos para la representación de la malla de datos (*simple grid*)

- Uso de mallas (*mesh*) de grid simple.

  Se crea una estructura de datos conteniendo la posición de cada vértice de la malla. Por tanto la malla contiene exactamente la misma información que la imagen.

En la figura 21 se muestra el diagrama de nodos utilizado para este tipo de representación.

El proceso se ve facilitado por el uso del nodo específico *SoQuadMesh* ofrecido por la librería Coin3D para la creación de este tipo de mallas.

En este caso no se hace necesario el uso de nodos de información para conocer el numero de filas y de columnas de la imagen que representa la malla. La información se puede obtener directamente de los campos *verticesPerColumn* y *verticesPerRow* del nodo malla de la clase *SoQuadMesh.*

El nodo de la clase *SoShapeHints* se hace necesario para poder modificar ciertos aspectos en la representación de la malla, sobretodo el valor de los campos *vertexOrdering* y *shapeType*, los cuales permiten el renderizado de la malla por las dos caras.
El resto de nodos representan la misma funcionalidad que en el caso de la nube de puntos

Función *cargar_malla().*

Es la encargada de generar la estructura de nodos del árbol de la figura 21. Al igual que en el caso anterior, los datos de coordenadas necesarios son los obtenidos por el uso de la función *cargar_datos()*.

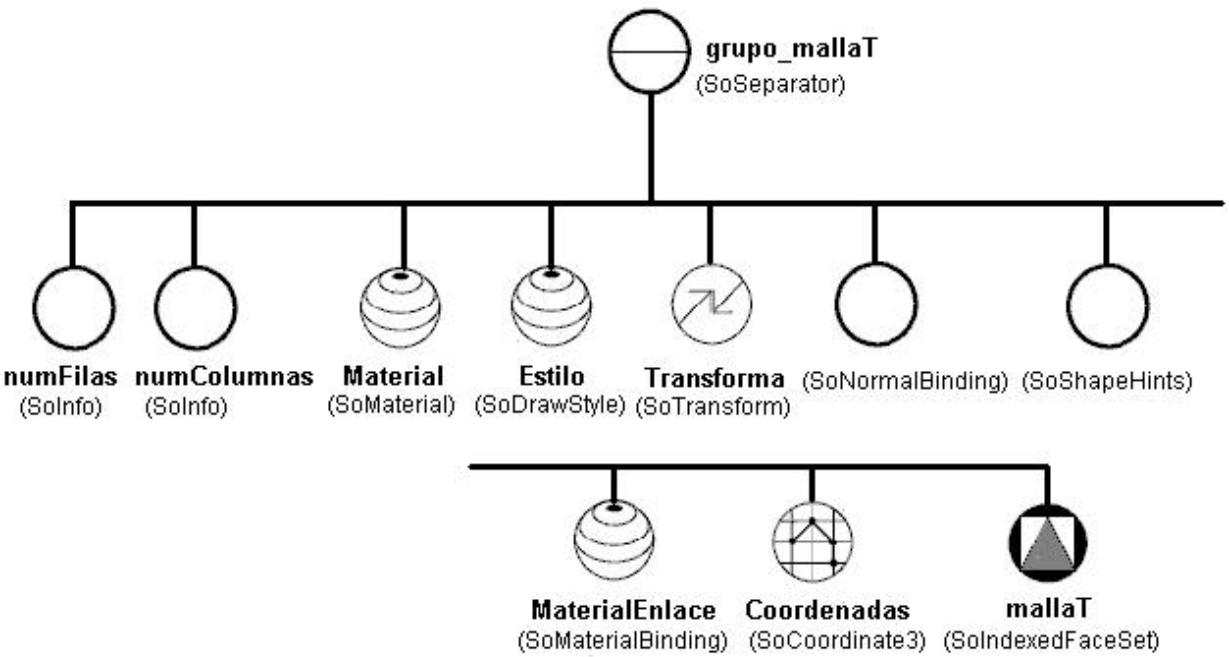


Figura 22. Diagrama o árbol de nodos para la representación de la malla de datos (*Triang. grid*).

- Uso de mallas de grids formados por unión de triángulos. Se crea una malla formada por grids mediante unión de 2 triángulos.

En este caso la matriz contiene la misma información que la anterior (a nivel de vértices) pero en este caso la visualización de la figura es sensiblemente diferente a la anterior.

En el diagrama de la figura 22 se muestra cuál es la estructura de nodos utilizada. La generación de la malla es más elaborada que en los dos casos anteriores. En este caso la generación de cada uno de los grids se debe de hacer "manualmente".

En los casos anteriores no había que preocuparse por la conexión de los vértices mediante las aristas correspondientes. Este trabajo es realizado internamente por la librería, la única responsabilidad del programador era indicar los vértices que componían la imagen de manera lineal.

Función *cargar_mallat()*.

Esta función crea la estructura de nodos mostrada en el diagrama de la figura 22.
Al igual que en el caso de la representación mediante la nube de puntos, se hace necesario el uso de los dos nodos de información (clase *SoInfo*) para poder conocer el número de filas y columnas.

Todas las funciones anteriores se encuentran declaradas y definidas en los ficheros *figuras.h* y *figuras.cpp* (ver código fuente).

Datos adicionales para la generación de las mallas.

Los procesos de creación de las mallas y de la nube de puntos, difieren según el fichero de entrada de datos.

Para el caso de *bitmaps*, se requieren, al menos, dos datos que deben ser facilitados por el usuario: temperatura máxima y temperatura mínima.
Estos dos datos, junto con el valor del índice a la paleta de colores, son los que proporcionan la cota Z en los vértices que conforman las mallas.

Para el caso de ficheros de entrada binarios, la cota de la coordenada Z es asignada de forma directa mediante la lectura del valor de temperatura que almacena el fichero.

4.3.5 Bloque 5. Representación 3D.

Cuando el usuario crea una malla, mediante el comando correspondiente, ésta es añadida a la *scenegraph* de manera automática. Todas estas mallas tienen sus respectivas propiedades sin tener porque ser iguales a las de otras mallas presentes en la misma *scenegraph*.

Aunque se podría haber adoptado como solución el crear para cada malla diferente una nueva instancia de la ventana de render, se ha decidido implementar una estructura de nodos que pueda contener más de una malla simultáneamente. Se ha considerado que puede ser útil en ciertos casos, como por ejemplo: añadir otros objetos, ejes de coordenadas, visualizar la foto de luz visible correspondiente a la zona donde se ha capturado la termografia, ampliar secciones diferentes de la misma malla, etc.

Para ello se ha utilizado el árbol de nodos de la figura 23.

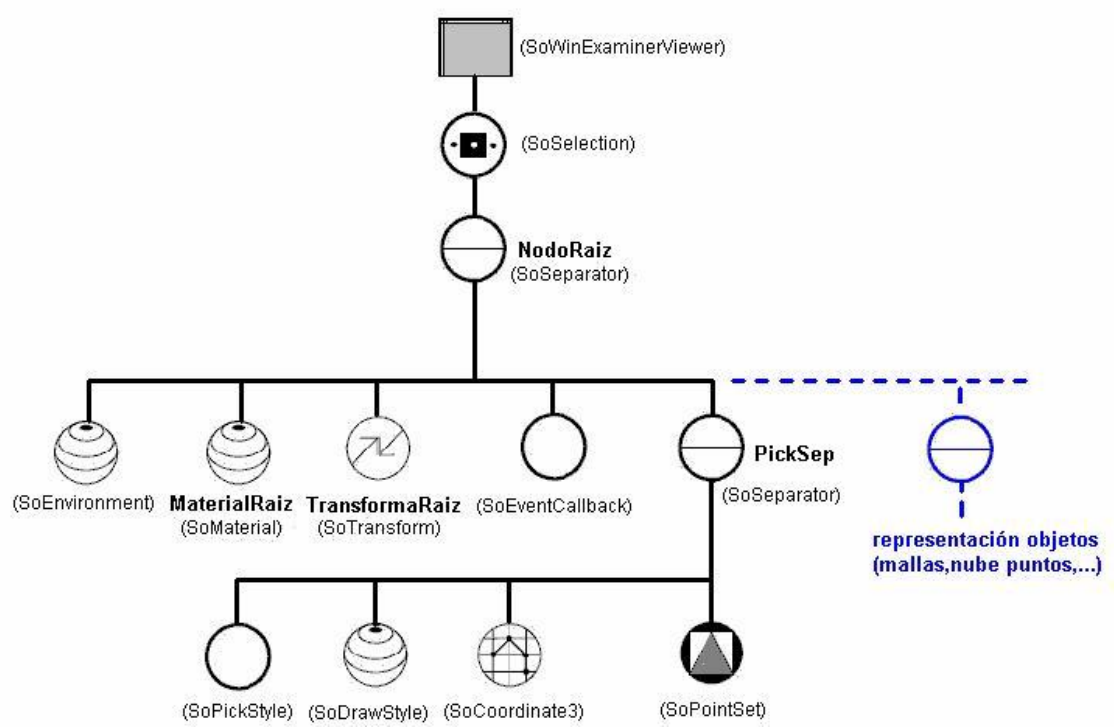


Figura 23. Árbol de nodos para Representación de objetos.

Para la visualización se utiliza el nodo de la clase *SoWinExaminerViewer*, que forma parte del *toolkit SoWin* de la librería Coin3D (*SoXt* o *SoQt* para sistemas LINUX). Éste nodo proporciona la pantalla donde se realiza el renderizado de la escena gráfica, así como algunos botones para realizar diferentes acciones sobre la imagen (restitución de la posición inicial de la imagen "home", rotación de cada uno de los ejes, posicionamiento de cámara, tipo de render de la imagen en estado estático y en el de animación, tipo de buffer del render, tipo de transparencia, etc.).

El nodo de selección es el encargado de almacenar y gestionar los objetos que son seleccionados en la escena gráfica. Su ubicación encima del nodo raíz garantiza que cualquier objeto generado en la ventana pueda ser seleccionado. La gestión de los eventos de selección y deselección de objetos se explica en el siguiente bloque.

Por su parte el nodo *transformaRaiz* posibilita el realizar las principales acciones de transformación de manera global a todas las figuras que contenga el nodo raíz.

El nodo de la clase *SoEnvironment* y el nodo *MaterialRaiz* permiten establecer las propiedades del entorno de visualización y de los objetos incluidos de manera directa (sin nodos separadores) en la *scenegraph*.

El conjunto formado por los 4 nodos hijo del nodo *PickSep* permiten obtener información sobre las zonas del objeto seleccionado donde el usuario presione el botón del ratón.
El nodo *SoEventCallback* es el encargado de llamar a las funciones que atienden los eventos del teclado y del ratón.

La línea discontinua, de diferente color, que se aprecia en la figura 23, representa la posición donde se añaden el resto de nodos que se pueden crear y visualizar en tiempo de ejecución.

Las declaraciones de cada uno de los nodos que conforman la estructura de la figura 23 y el establecimiento de los valores por defecto de sus campos, incluyendo las funciones de manejo de eventos (*callbacks*) del teclado y del ratón y las funciones de respuesta a la selección de objetos, se realiza en la función *main()*.

4.3.6 Bloque 6. Interacción Usuario-Aplicación (eventos teclado, ratón)

La principal forma de comunicación entre el usuario y la aplicación es mediante el uso de comandos. Se han definido 53 comandos diferentes que se explicarán en el capítulo 5.

A cada ejecución de la aplicación Visor3D se abren dos ventanas (ver figura 30), una representa la interfaz del usuario proporcionada por la librería *Coin* y es la explicada en el bloque anterior. La otra es una ventana del tipo consola. Se usa para mostrar los caracteres de teclado introducidos por el usuario y los mensajes devueltos por la aplicación.

Para poder introducir los comandos es necesario que no haya ninguna función de visionado activa (función *view( )*) en la ventana de interfaz de usuario. El usuario puede ver el estado pulsando el botón derecho del ratón dentro de la interfaz gráfica o seleccionando el botón que representa el cursor.

En esta situación, cada pulsación del teclado genera un evento que es detectado por la propia librería Coin3D.
Ésta es la función del nodo de la clase *SoEventCallback*. A éste nodo se le indican las funciones que se deben de ejecutar según el tipo de evento que ocurre, entre ellos el del teclado.

Para responder a los eventos de teclado se usa la función *eventoteclado()*, la cual se encuentra definida dentro del mismo fichero donde se ubica la función *main()*. El funcionamiento de la función *eventoteclado()* se explica a continuación.

función *eventoteclado()*

Recoge todas las pulsaciones de teclas y las va enviando a un objeto del tipo *bufferteclas*, el cual va contando el número de teclas y las va almacenando en un buffer interno.

Cuando se detecta la pulsación de la tecla *enter*, la cadena contenida en el buffer se le pasa a un objeto de la clase *ccomando*, el cuál, mediante el método *ordenarcomando(),* se encarga de separar la cadena en conjuntos de caracteres llamados parámetros.
La aplicación espera que los comandos se introduzcan mediante el siguiente formato:

Nombre_comando parametro_1 parametro_2 ... parametro_n

donde el nombre del comando representa el parametro_0.

Una vez seccionada la cadena, se detecta si el primer conjunto de caracteres (parametro[0]) corresponde o no a un comando válido.
En caso afirmativo el resto de la cadena se considera como parámetros válidos del comando.

Tipos de comandos				
Obtención datos	**creación**	**Manipulación/modificación**	**Propiedades**	**Información**
Analizarimagen	Cargarnube	Cortarxfila	Invertirc	Eje
	Cargarmalla	Cortarxcolumna	Asignarpaleta	Autoeje
	Cargarmallat	Cortarxarea	Colorfondo	Info
	Crearmalla	Seleccionararea	Luz-on/-off	Info-off
	script	Translacion	Posicionluz	Referencia
		Rotación	Colorluz	V2bin
		Escalar	Luces-off	V2txt
		Centro	Texturizar	H2txt
		Trackball-on/-off	Texturizar-off	promedio
		Box-on/-off	poscamara	histo
		Manipuladores-off	transcamara	etiqueta
		Cls	orcamara	
		Suprimir		
		Filtro		
		Filtrobanda		
		posicionar		
		selobjeto		
		isoterma		
		Reflejarh/v		

Figura 24. Listado de comandos de la aplicación.

Una vez detectada la introducción de un comando válido se llama al método *gestionar_comando()*, que se encarga de evaluar si el número de parámetros pasados corresponde tanto en número como en formato, a los esperados, ejecutando en caso afirmativo la función asociada a dicho comando.
Se ha implementado un set de 53 comandos que se pueden clasificar en las categorías mostradas en la figura 24.

Todas las funciones que se llaman al introducir los comandos se encuentran declaradas y definidas en los ficheros *gestion_comandos.h* y *gestion_comandos.cpp* (ver código fuente).

Por su parte la declaración y definición de la clase *bufferteclas* se encuentra en los ficheros *gestionteclas.h* y *gestionteclas.cpp* respectivamente(ver código fuente).

De manera similar a los eventos de teclado, los eventos de ratón son detectados por el nodo de la clase *SoEventCallback*, el cual es el encargado de llamar a la función *eventoraton()*.

Aunque esta función es llamada a cada evento de ratón, ya sea pulsación/liberación del botón izquierdo/derecho/central, el bloque principal de código solo se ejecuta al detectarse pulsaciones del botón izquierdo.

Los eventos detectados son usados para obtener la coordenada, en el espacio del objeto, del objeto que se encuentre seleccionado en el momento en que se generó el evento. La función encargada de realizar este trabajo es *BusvarverticeCercano()*.

Una vez obtenidas las coordenadas, éstas se utilizan para:

1º Obtener el valor de la temperatura (coordenada Z) del punto del objeto seleccionado.

2º Obtener las coordenadas necesarias para el comando *seleccionararea*. La manera de conocer cuando se capturan los datos se consigue mediante la utilización de la variable global *capturaPuntos*, que es usada a modo de *flag*.

Aparte de las herramientas de manipulación de los objetos 3D, propias del nodo *SoWinExaminerViewer*, se han implementado una serie de comandos que amplían las posibilidades de visualización y manipulación.

4.3.7 Bloque 7. Obtención de Información y Resultados.

Se han implementado diversos métodos de obtención de información. El programa dispone de 2 vías para la obtención de los resultados. Una mediante la propia ventana del render y la otra generando algún tipo de fichero de salida.

El proceso de obtención de información sobre cada uno de los objetos en pantalla es el siguiente:

- El usuario selecciona un objeto mediante el ratón. Para indicar que el objeto se encuentra seleccionado se utiliza el nodo *SoBoxHighLightRenderAction* que genera un rectángulo que rodea el objeto.

- Después se llama a alguno de los comandos de información. Éstos son los encargados de llamar a las diferentes funciones de recopilación de datos.

Función *infoObjeto()*.

Esta función recopila información sobre el objeto que se le pase como parámetro. La almacena en las variables miembro de un objeto de la clase *infObjeto*. La información recopilada es la siguiente:

- Nombre del archivo origen de los datos.
- Posición del objeto en el espacio.
- Número de columnas.
- Número de filas.
- Número de vértices del objeto.
- Tamaño del objeto.

- Centro del objeto.
- Coordenadas absolutas del objeto.
- Factor escala del objeto.
- Eje y ángulo de rotación del objeto.
- Máximo y mínimo valor de las coordenadas X, Y, Z.
- Valor medio de coordenada Z (temperatura).

Mediante el parámetro *mostrar*, se decide donde mostrar la información. Para un valor de 0, solo se muestra la información en la línea de comandos devolviendo NULL.

Con valor de 1, se devuelve la información del objeto en un objeto de la clase *infObjeto*. Con un valor de 2, se muestra la información en la línea de comandos y se devuelve un objeto de la clase *infObjeto.* De ésta manera se puede utilizar la función tanto para obtener información y mostrarla por pantalla, como para otras funciones donde se necesite conocer alguno de los datos anteriores.

Clase *infObjeto*.

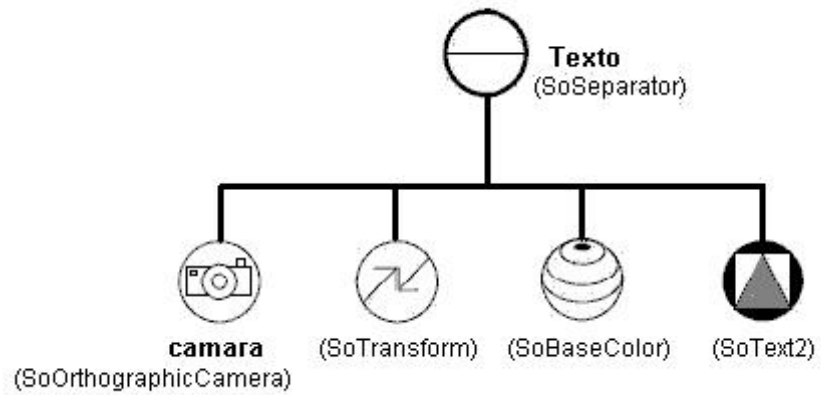


Figura 25. Árbol nodos del objeto *infObjeto*.

Aparte de contener las variables donde se almacenan los datos de los objetos, los objetos de esta clase contienen un conjunto de nodos que se utilizan cuando se requiere representar la información en la ventana de renderizado. La información se muestra como texto 2D, y se mantiene en todo momento paralela a la vista de la ventana del Visor3D.

En la figura 25 se muestra el árbol de nodos que contienen los objetos de la clase *infObjeto* y en la figura 26 un ejemplo de la posición donde se insertarían en el caso de representación de una malla.

El grupo de nodos siempre se inserta en la última posición del árbol de nodos que representa el objeto (nube de puntos/mallas).

Tanto la clase *infObjeto* como la función *infoObjeto()* se encuentran declaradas y definidas en los archivos *figuras.h* y *figuras.cpp* (ver código fuente).

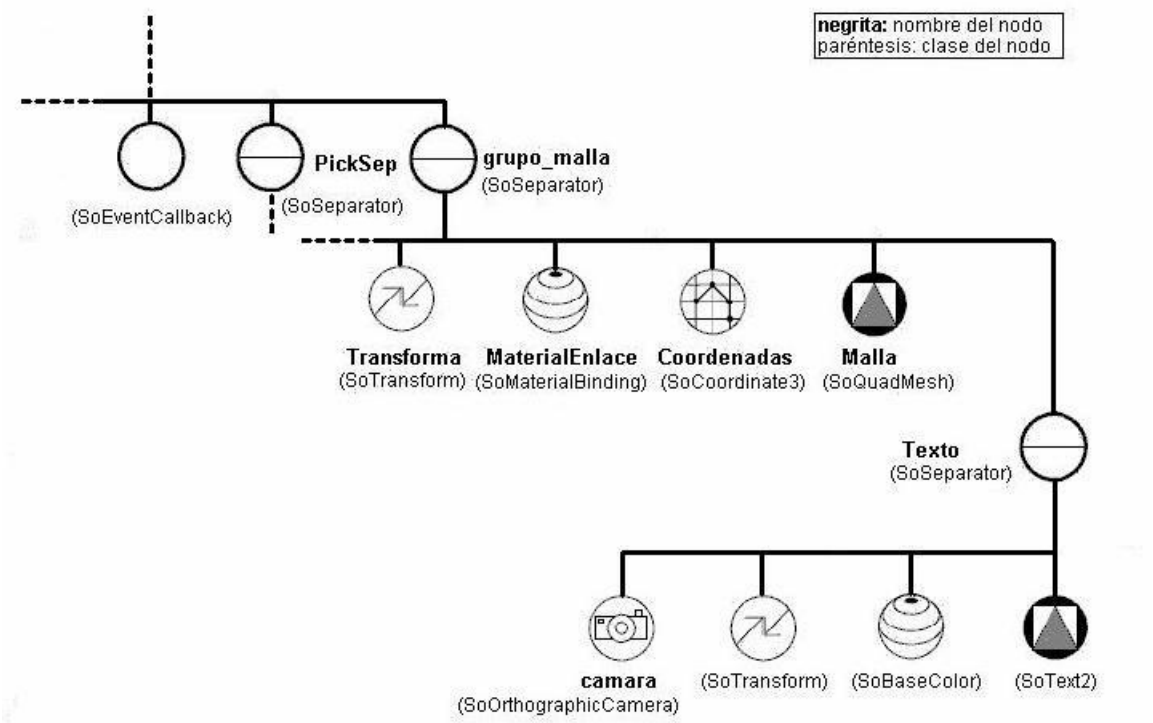


Figura 26. Integración de información en un objeto tipo malla (*simple grid*).

El otro método de obtención de información, del objeto seleccionado en pantalla, es mediante la generación de archivos, con el uso de las funciones *v2bin()* y *v2ascii()*.

Función *v2bin().*

Genera un archivo, en el directorio de ejecución de la aplicación, que contiene la información recopilada por la función *infoObjeto()*.

Para la creación del fichero se ha utilizado el mismo formato que los archivos binarios de 16 bits que se utilizan en la entrada de datos. De esta manera, el usuario puede generar el archivo y utilizarlo posteriormente como archivo de entrada de datos.

Por tanto, solo se incluye el número de columnas, de filas y seguidamente los datos de temperatura de cada uno de los vértices que componen la imagen termográfica.

Función *v2txt().*

Su funcionamiento es muy similar a la anterior. Genera un fichero en el directorio de trabajo actual, pero en este caso se incluye más información. Éste archivo se debe considerar como solo informativo y no puede ser leído de nuevo por la aplicación, es decir, no puedo ser usado como archivo para la entrada de datos de la aplicación.

Tanto la función *v2bin()* como *v2txt()* se encuentran en los archivos *ficheros.h* y *ficheros.cpp* (ver código fuente).

Función *generarEjeCoord()*.

Crea un eje de coordenadas que se puede incluir a cada uno de los objetos representados en pantalla. A cada llamada a la función se crea la estructura de nodos que se muestra en la figura 27.

Cada uno de los nodos separadores, *EjeY* y *EjeZ*, contienen la misma estructura que la mostrada en el *EjeX*. La función requiere un total de 20 argumentos para definir la creación del eje de coordenadas.

Los argumentos se le pasan mediante un objeto de la clase *ejecoordenadas*, y son los siguientes:

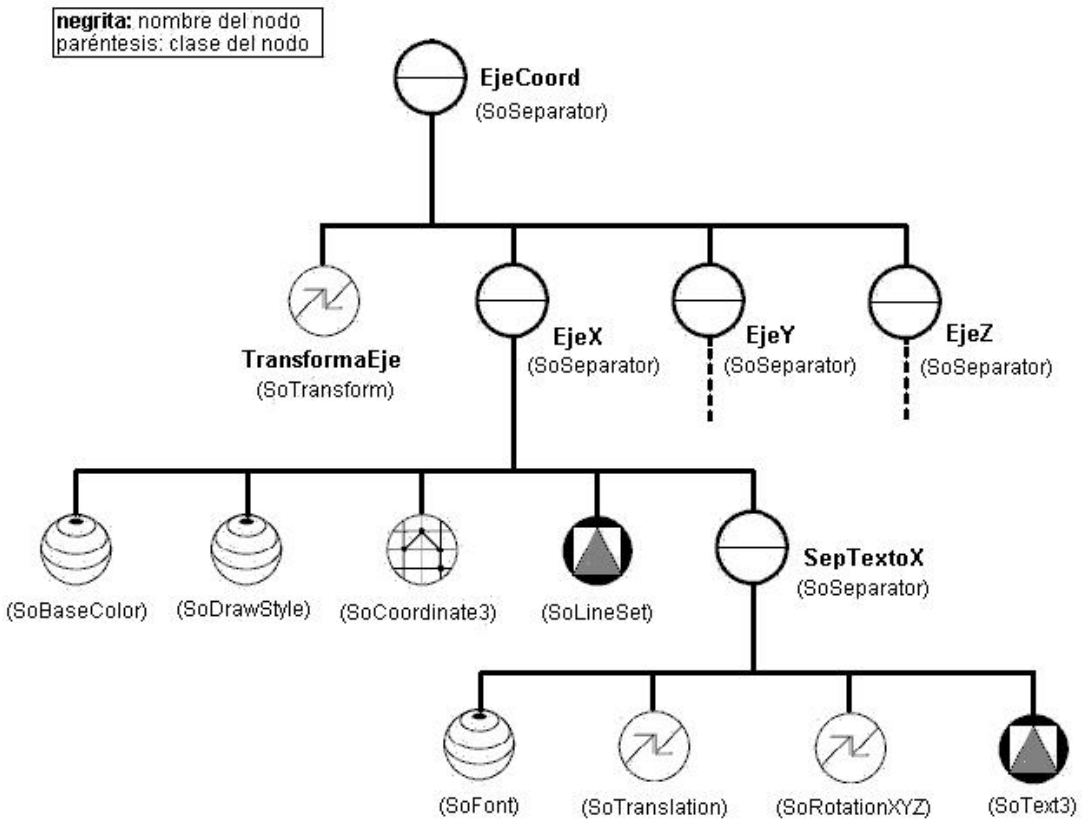


Figura 27. Árbol de nodos de los ejes de coordenadas.

- Máximo y mínimo valor para cada uno de los 3 ejes X, Y, Z.
- Número de marcas para cada uno de los 3 ejes.
- La longitud de las marcas de cada uno de los 3 ejes.
- Tamaño de fuente utilizado en las etiquetas.
- Separación de la etiqueta respecto al eje correspondiente.
- Coordenadas X, Y, Z para el posicionamiento del eje.

El nodo de la clase *SoTransform*, que cuelga del nodo *EjeCoord*, sirve para ubicar el eje en la coordenada necesaria.

Los nodos de la clase *BaseColor* y *SoDrawStyle* establecen el color y el tipo de representación, respectivamente, que será usado para cada uno de los componentes que conforman el eje. Las coordenadas de los puntos necesarios para trazar las líneas que conforman cada eje se indican en el nodo *SoCoordinate3*.

El nodo de la clase *SoLineSet* es el encargado de dibujar en pantalla las líneas.

El nodo *SepTexto*, formado por sus 4 nodos hijos, se encarga de generar cada una de las etiquetas que se colocarán en las posiciones correspondientes a las marcas realizadas en el eje.

4.4 Visor2D: GUI para la aplicación Visor3D.

El proceso a seguir por el usuario para obtener la representación y los datos de la termografia por medio de la introducción de comandos por teclado, puede resultar pesado, lento y poco intuitivo. Además cada comando posee sus respectivos argumentos. Por esta razón se ha desarrollado un programa que ofrezca, de una manera más rápida y sencilla, el acceso a todos los comandos del Visor3D.

Para el desarrollo de este Visor2D, se ha optado por el lenguaje de programación VBasic, el cual ofrece una manera rápida de diseñar la interfaz de usuario sin escribir apenas código. También posibilita implementar, de una manera rápida, los procedimientos y funciones necesarios para atender a los eventos de teclado y ratón generados por el usuario. También ha jugado un papel importante en la elección de VBasic como entorno de desarrollo, el hecho de que la librería FreeImage ofrezca un buen soporte para él, lo que ha facilitado en gran medida el proceso de apertura y manipulación de los archivos de mapas de bits.

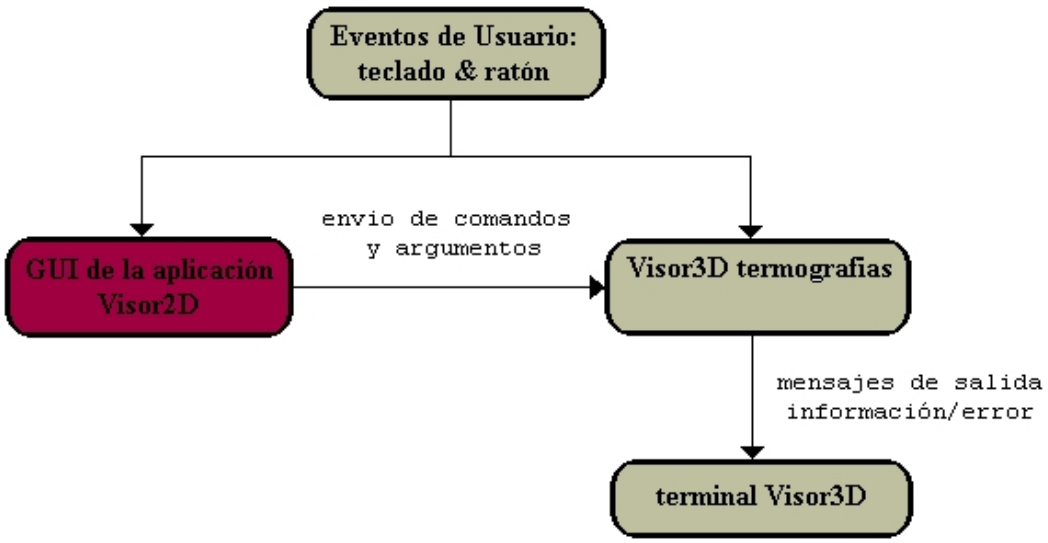


Figura 28. Integración del Visor2D

No todo son ventajas con el uso de VBasic. Su uso ha impuesto grandes limitaciones. La más importante es la que hace referencia a la comunicación con la aplicación Visor3D. El proceso de conexión entre ambas aplicaciones se ha tenido que realizar mediante llamadas a funciones de la API Win32.

En la figura 28 se muestra la integración de la aplicación Visor2D en la aplicación central Visor3D.

Tal y como se aprecia en la figura 28, el uso de la aplicación Visor2D es totalmente prescindible. Es el usuario quien decide si usar el Visor3D de manera individual o junto con el Visor2D.

El funcionamiento es sencillo. El usuario por medio de menús y botones configura y envía la secuencia de comandos, con los argumentos necesarios correspondientes, al Visor3D. Éste los ejecuta y muestra los posibles mensajes de error y/o información en la ventana de la consola.

Aprovechando la facilidad de uso que brinda VBasic, se han implementado algunas opciones que solo están disponibles en el Visor2D, como son: la creación de degradados de color simple y la posibilidad adicional de ajustar los valores de emisividad y de temperatura ambiental. De esta manera, se pueden corregir los valores de lectura de la temperatura.

4.5 Método para automatizar la ejecución de comandos.

Gracias a las facilidades ofrecidas por la librería Coin3D, se ha incorporado al programa Visor3D, la posibilidad de ejecutar archivos de texto escritos por el usuario y de ésta manera automatizar tareas.

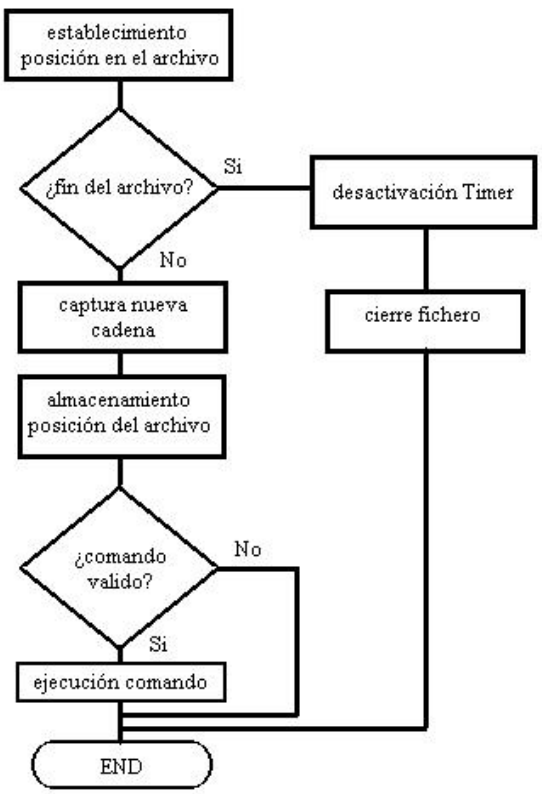


Figura 29. Diagrama de flujo de la función *callbackTimer()*

Para ello se utiliza el nodo *SoTimerSensor*, el cual posee un campo donde se indica el intervalo en segundos entre ejecuciones de la función de atención al *Timer* (*callbackTimer()*).

El proceso a seguir para usar el nodo es muy sencillo. Solo requiere el establecimiento de 3 campos:

- Intervalo de tiempo del *timer* (en segundos).
- Indicación de la función a ejecutar. Función *callbacktimer()*.
- Activación/desactivación del *timer*.

Las funciones encargadas de leer y ejecutar los comandos contenidos en el archivo de texto (*script*) son: *leer_script()* y la propia *callbackTimer()*.

Función *leer_script()*.

Esta función abre el fichero *script*, y si no se han producido errores de lectura, establece en el *timer* el intervalo de tiempo indicado por el usuario. Acto seguido lo activa.
A partir de ese momento, cuando transcurren los segundos indicados, se ejecuta la función *callbackTimer()*.

Función *callbackTimer()*.

El diagrama de flujo de la función se muestra en la figura 29.

A cada ejecución de la función, se muestra en la pantalla de la consola (y se captura) un carácter del archivo. El carácter se almacena en una instancia de la clase *ccomando*, ya explicada anteriormente.

Al detectar el carácter de fin de comando (carácter ';'), se ordenan los argumentos y se comprueba que se trata de un comando válido. En tal caso, se ejecuta el comando mediante la función *gestionar_comando()*, y se termina la ejecución de la función.
Este proceso se repite periódicamente hasta que se alcanza el final del fichero.

El formato del archivo de texto usado como archivo *script*, se muestra en el anexo 3.

Capítulo 5. Manual de Usuario

El usuario puede utilizar el software de visualización 3D de termografías de dos maneras diferentes:

- Ejecutando la aplicación Visor3D y introduciendo los comandos deseados desde el teclado.

- Ejecutando la aplicación Visor2D y, mediante la interfaz de usuario, realizar las acciones deseadas.

A continuación se muestra el uso de la aplicación Visor3D ejecutada de manera individual. Seguidamente se muestra el uso de la aplicación Visor2D y Visor3D trabajando de manera conjunta.

5.1 Uso del software Visor3D.

Al ejecutar el archivo *Visor3d.exe* se abren dos ventanas (figura 30). En la ventana de la derecha se mostrarán todos los objetos generados, mientras que en la ventana de la izquierda (ventana de consola) se irán mostrando los comandos introducidos por el usuario y los mensajes de respuesta de la aplicación.

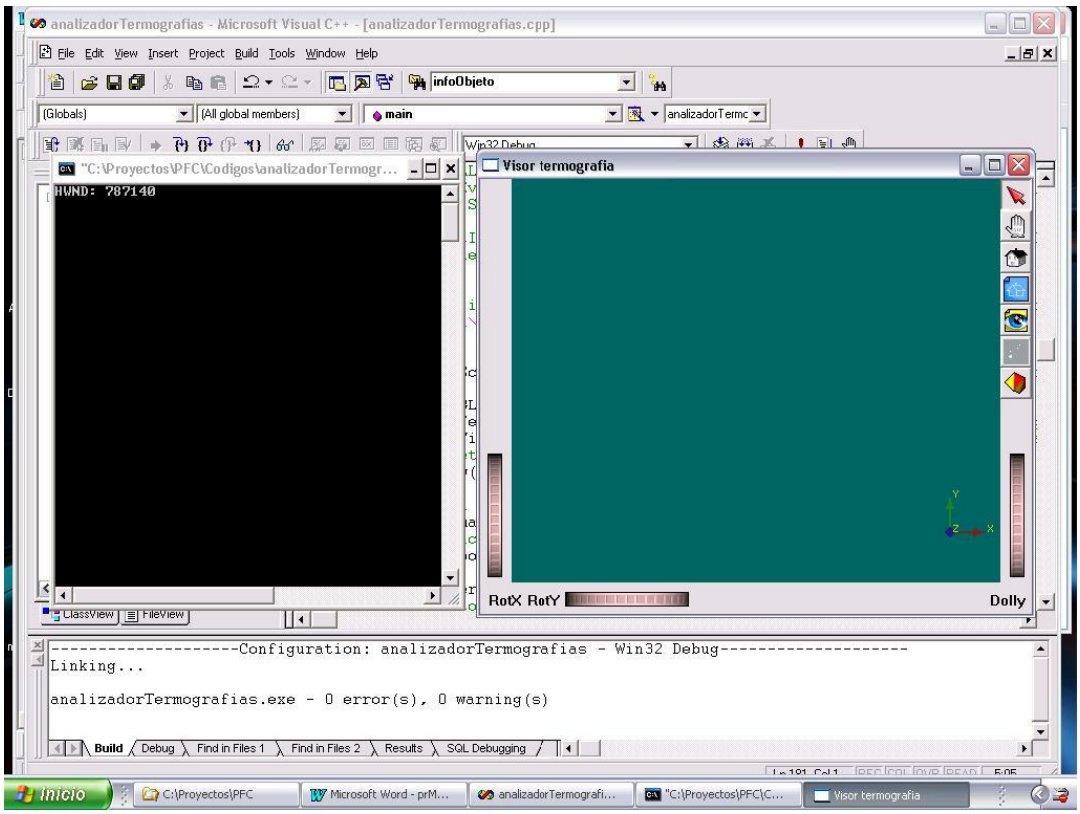


Figura 30. Ventanas de la aplicación Visor3D de termografías.

Para poder introducir los comandos es necesario que la función *view()*, de la ventana de renderizado, esté desactivada. Ésto se consigue activando el botón situado en la zona superior derecha de la ventana (botón de selección).

La lista de comandos existentes, su descripción, sintaxis, número y formato de los argumentos, es la siguiente:

Comando *cls*.

Descripción: Borra todos los objetos que haya en la ventana.
Acción previa: -
Nº argumentos: -
sintaxis: `cls`

Comando *suprimir.*

Descripción: Elimina el/los objeto(s) seleccionado(s)
Acción previa: debe de haber seleccionado como mínimo un objeto.
Nº argumentos: -
sintaxis: `suprimir`

Comando *analizarimagen.*

Descripción: Identifica el tipo de archivo bitmap que se le pasa y en caso de ser un formato soportado, crea 3 archivos temporales (*datoscab.txt*, *datosimag.txt* y *datospal.txt*) en el directorio de ejecución de la aplicación y devuelve los principales datos del bitmap.
Acción previa: -
Nº argumentos: 1
sintaxis: `analizarimagen archivo_bitmap`

Comando *colorfondo*.

Descripción: Establece el color de fondo para la ventana donde se representan los objetos 3D. Los valores de las componentes deben de estar comprendidas entre 255 y 0.
Acción previa: -
Nº argumentos: 3
sintaxis: `colorfondo R G B`

Comando *cargarnube*.

Descripción: muestra en pantalla una nube de puntos que representa la imagen 3D de una termografia.
Acción previa: deben de existir los ficheros *datoscab.txt*, *datosimag.txt* y *datospal.txt* en el directorio de ejecución de la aplicación en caso de usar 8 argumentos.
Nº argumentos: Variable:
7 en caso de entrada de datos mediante archivo binario.
8 en caso de entrada de datos con archivos bitmap.
sintaxis:
```
cargarnube     usarpaleta R G B Tmax
               Tmin nombre forzar_datos
```

```
cargarnube      fichero_binario
                usarpaleta R G B nombre
                forzar_datos
```

usarpaleta	1 se usa la paleta original de la imagen, 0 se utiliza el mismo color para todos los puntos de la nube. Argumento ignorado en caso de 7 argumentos.
RGB	deben de ser valores entre 0 y 255.
Tmax	Establece la temperatura máxima que se asociará al último color de la paleta.
Tmin	Establece la temeperatura asociada al primer color de la paleta.
Nombre	Establece el nombre indicado a la nube de puntos generada.
forzar_datos	Con un valor de 1, se fuerza a considerar a las imágenes de 24bpp como imágenes termográficas válidas para la extracción de datos de temperatura.

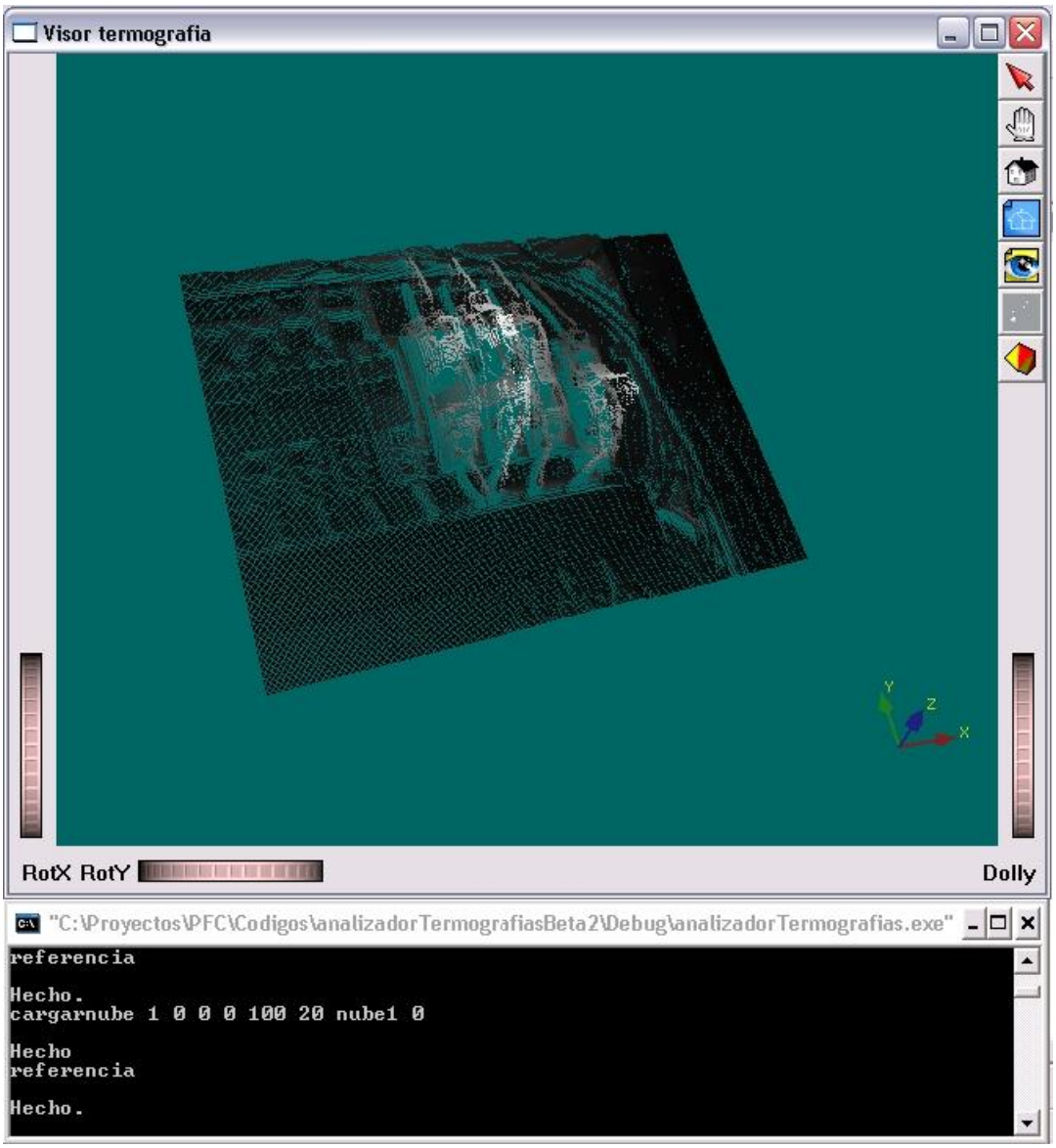


Figura 31. Ejemplo de ejecución del comando *cargarnube.*

Comando *cargarmalla*.

Descripción: muestra en pantalla una malla tipo *simple grid* que representa la imagen 3D de una termografía.

Acción previa: Deben de existir los ficheros *datoscab.txt*, *datosimag.txt* y *datospal.txt* en el directorio de ejecución de la aplicación en caso de usar 8 argumentos.

Nº argumentos: Variable:
7 en caso de entrada de datos mediante archivo binario.
8 en caso de entrada de datos con archivos bitmap.

sintaxis:

```
cargarmalla    usarpaleta R G B Tmax
               Tmin nombre forzar_datos
cargarmalla    fichero_binario
               usarpaleta R G B nombre
               forzar_datos
```

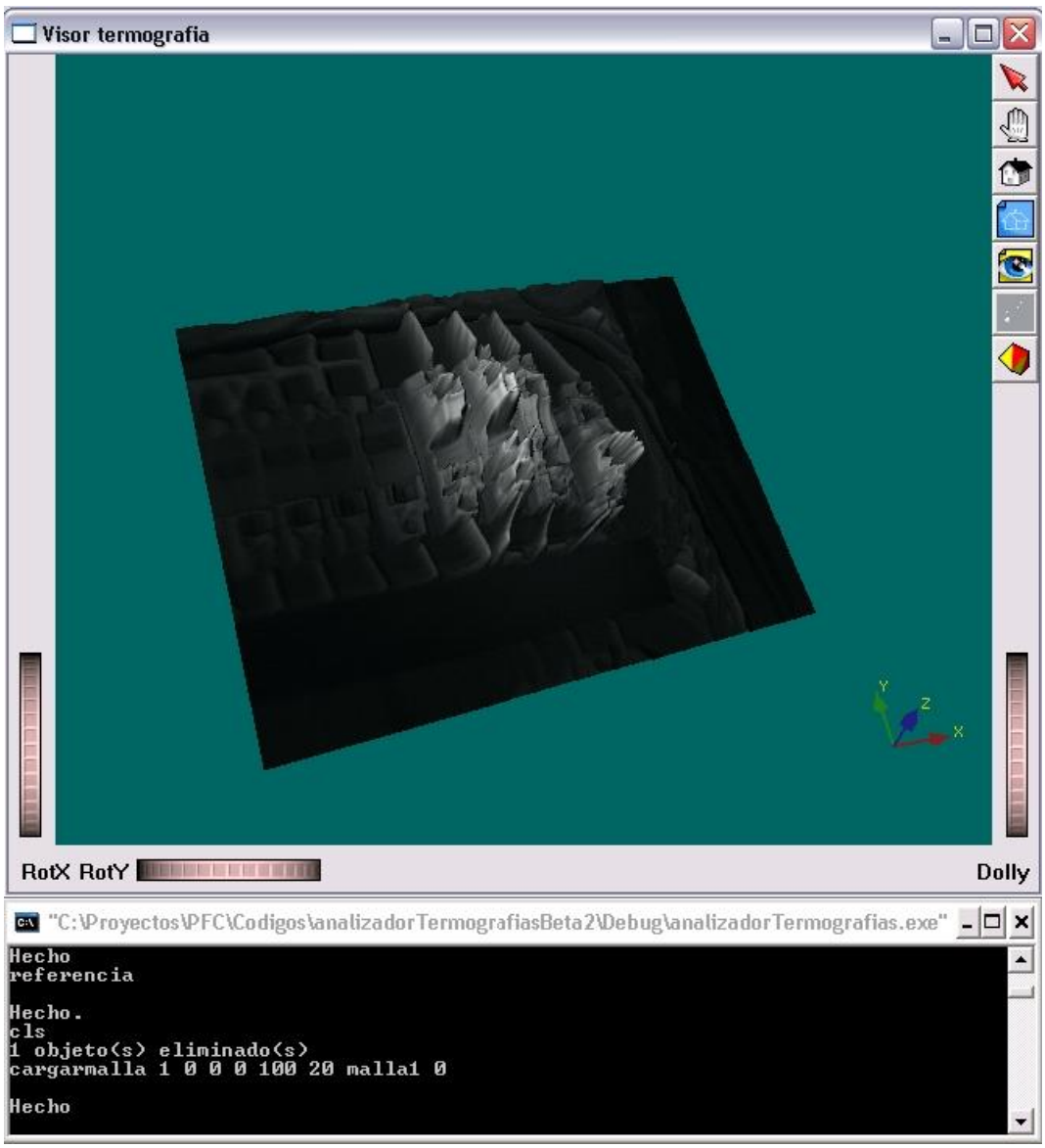


Figura 32. Ejemplo de ejecución del comando *cargarmalla* (*simple grid*).

usarpaleta	1 se usa la paleta original de la imagen, 0 se utiliza el mismo color para todos los puntos de la nube. Este argumento es ignorado en caso de usar la sintaxis 7 argumentos.
RGB	Deben de ser valores entre 0 y 255.
Tmax	Establece la temperatura máxima que se asociará al último color de la paleta.
Tmin	Establece la temperatura asociada al primer color de la paleta.
nombre	*E*stablece el nombre indicado a la malla generada.
forzar_datos	Con un valor de 1, se fuerza a considerar a las imágenes de 24bpp como imágenes termográficas válidas para la extracción de datos de temperatura.

Comando *cargarmallat.*

Descripción: muestra en pantalla una malla de tipo *triangulated grid* que representa la imagen 3D de una termografia.

Acción previa: Deben de existir los ficheros *datoscab.txt*, *datosimag.txt* y *datospal.txt* en el directorio de ejecución de la aplicación en caso de usar 8 argumentos.

Nº argumentos: Variable:
7 en caso de entrada de datos mediante archivo binario
8 en caso de entrada de datos con archivos bitmap.

sintaxis:

```
cargarmallat   usarpaleta R G B Tmax
               Tmin nombre forzar_datos

Cargarmallat   fichero_binario
               usarpaleta R G B nombre
               forzar_datos
```

Usarpaleta	Para un valor de1 se usa la paleta original de la imagen, 0 se utiliza el mismo color para todos los puntos de la nube. Argumento ignorado en caso del uso de la forma con 7 argumentos.
R G B	deben de ser valores entre 0 y 255.
Tmax	Establece la temperatura máxima que se asociará al último color de la paleta.
Tmin	Establece la temperatura asociada al primer color de la paleta.
nombre	*E*stablece el nombre indicado a la malla generada.
forzar_datos	Con un valor de 1, se fuerza a considerar a las imágenes de 24bpp como imágenes termográficas válidas para la extracción de datos de temperatura.

Comando *eje.*

Descripción: Muestra un eje de coordenadas en las coordenadas especificadas y con el número de marcas, longitud y colores especificados.

Acción previa: -

Nº argumentos: 20

sintaxis:

```
eje  coordX coordY coordZ marcasX
     marcasY marcasZ maxX maxY maxZ
     minX minY minZ longmarcaX
     longmarcaY longmarcaZ
     tamaño_fuente  separacion_etiquetas
     tituloX       tituloY       tituloZ
```

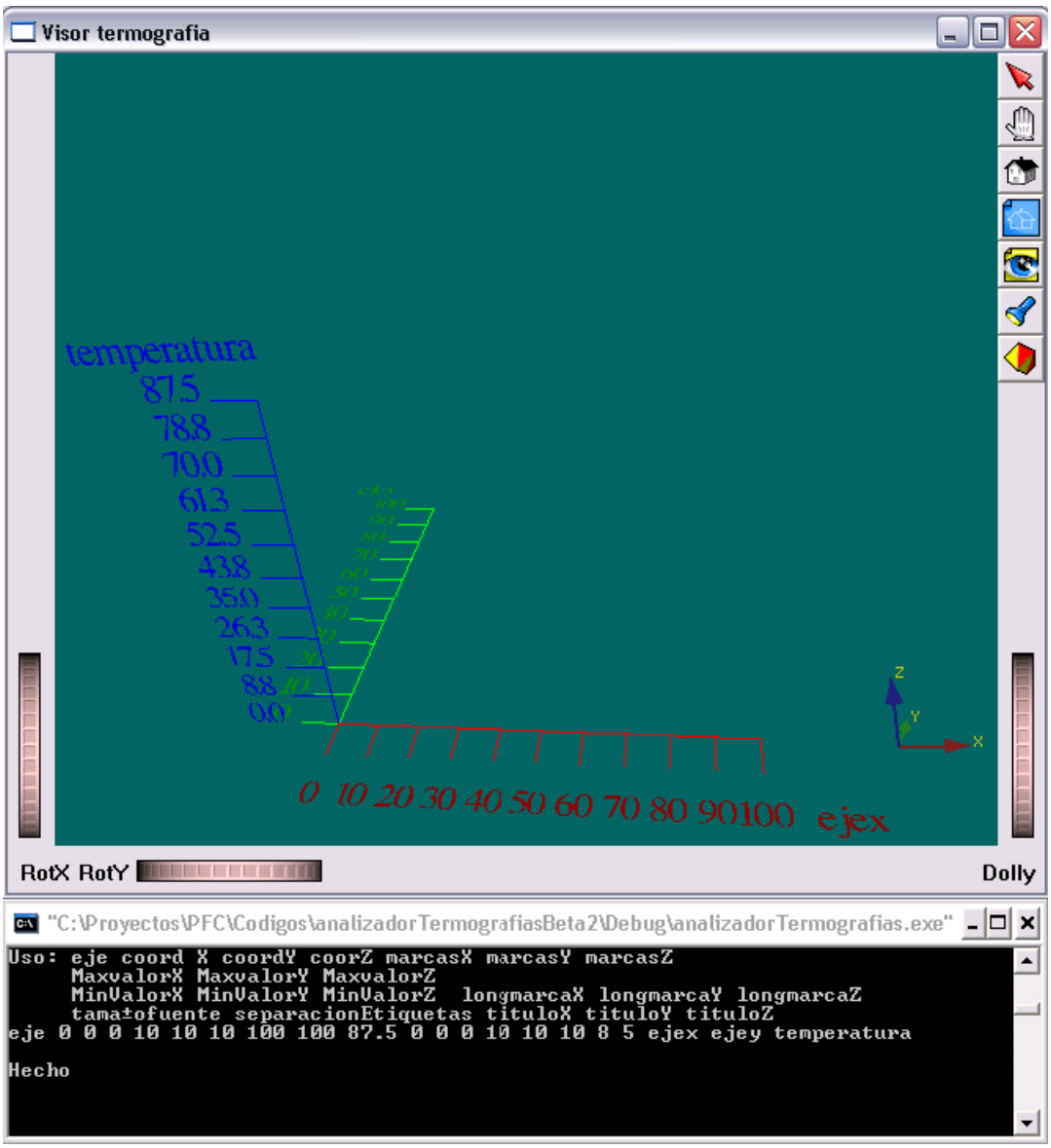


Figura 33. Ejemplo de ejecución del comando *eje*.

coordX/Y/Z indican la posición espacial del eje de coordenadas.
marcasX/Y/Z indican el número de marcas que habrá en los ejes.
maxX/Y/Z indican el máximo valor para cada uno de los ejes de coordenadas.
minX/Y/Z indican el valor mínimo para cada uno de los ejes de coordenadas.
longmarcaX/Y/Z indican la longitud de las marcas de separación de cada uno de los ejes.
tamaño_fuente tamaño de la fuente usada para las etiquetas de los ejes.
separacion_etiquetas separación entre las etiquetas y las marcas de los ejes.
tituloX/Y/Z titulo para cada uno de los tres ejes.

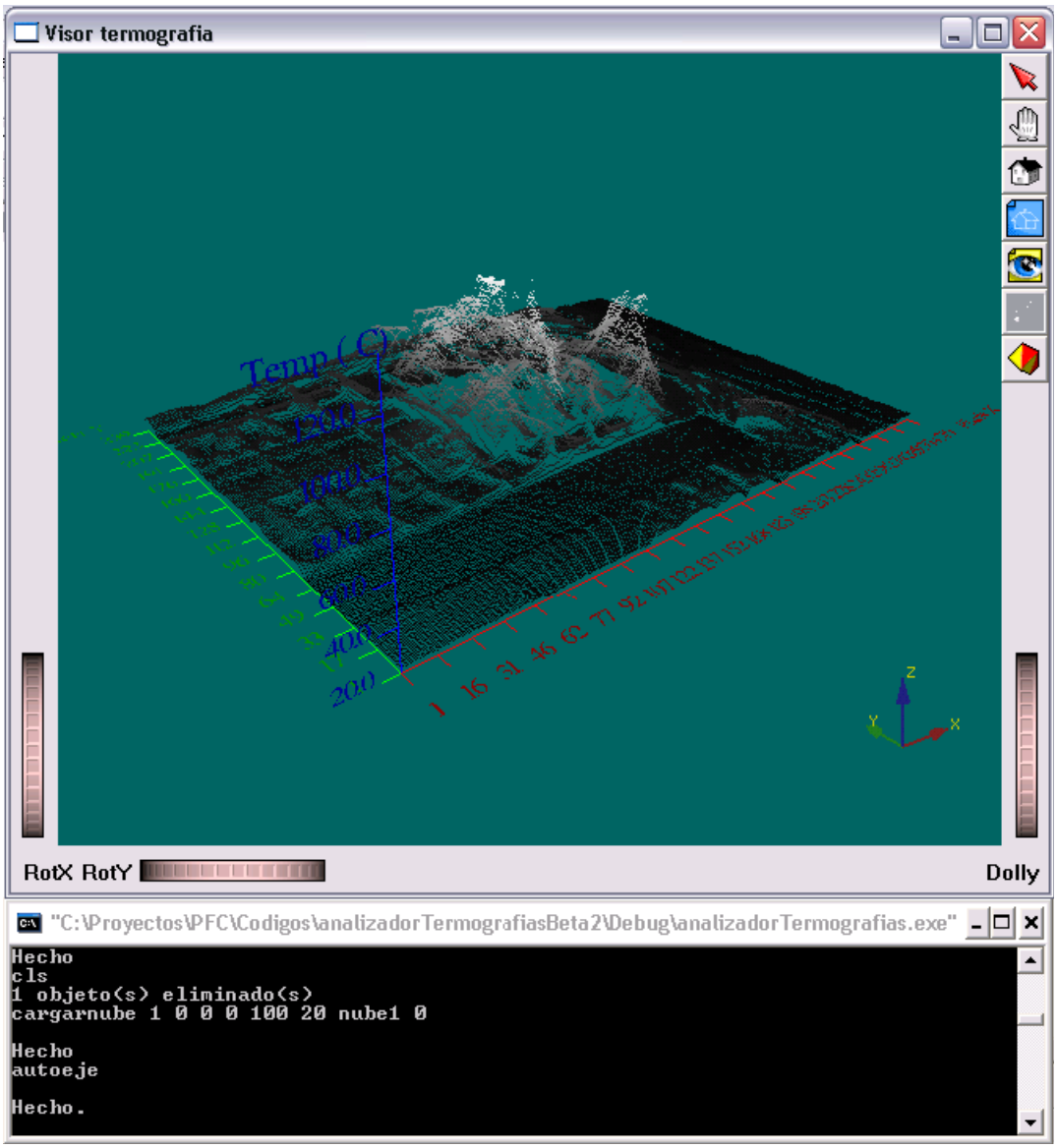


Figura 34. Ejemplo de ejecución del comando *autoeje*.

Comando *autoeje*.

Descripción: Asocia un eje de coordenadas, cuyas propiedades se intentan ajustar de manera automática a las dimensiones del objeto seleccionado.
Acción previa: Debe de haber un objeto seleccionado.
Nº argumentos: -
sintaxis: `autoeje`

Comando *translacion.*

Descripción: Realiza una translación del objeto seleccionado en la magnitud indicada.
Acción previa: Debe de haber seleccionado un objeto.
Nº argumentos: 3
sintaxis: `translacion posX posY posZ`

posX/Y/Z indican la magnitud de la translación para cada uno de los tres ejes.

Comando *rotacion.*

Descripción: Realiza la rotación del objeto selecionado en el ángulo especificado.
Acción previa: Debe de haber seleccionado un objeto.
Nº argumentos: 3
sintaxis: `rotacion X Y Z angulo`

X/Y/Z Deben de tomar el valor 0 (no se aplica rotación) o 1 (aplica rotación).
angulo Valor en grados sexagesimales.

Comando *escalar.*

Descripción: Realiza el escalado del objeto seleccionado.
Acción previa: Debe de haber seleccionado un objeto.
Nº argumentos: 3
sintaxis: `escalar factorX factorY factorZ`

factorX/Y/Z establecen el factor de escala en cada uno de los 3 ejes.

Comando *centro*.

Descripción: Modifica las coordenadas del centro del objeto. Equivale a establecer el nuevo origen de coordenadas del objeto en las coordenadas absolutas (en el espacio del mundo 3D) especificadas.
Acción previa: Debe de haber seleccionado un objeto.
Nº argumentos: 3
sintaxis: `centro coordX coordY coordZ`

Comando *trackball-on*.

Descripción: Asigna al objeto seleccionado un manipulador del tipo trackball.
Acción previa: Debe de haber un objeto seleccionado.
Nº argumentos: -
sintaxis: `trackball-on`

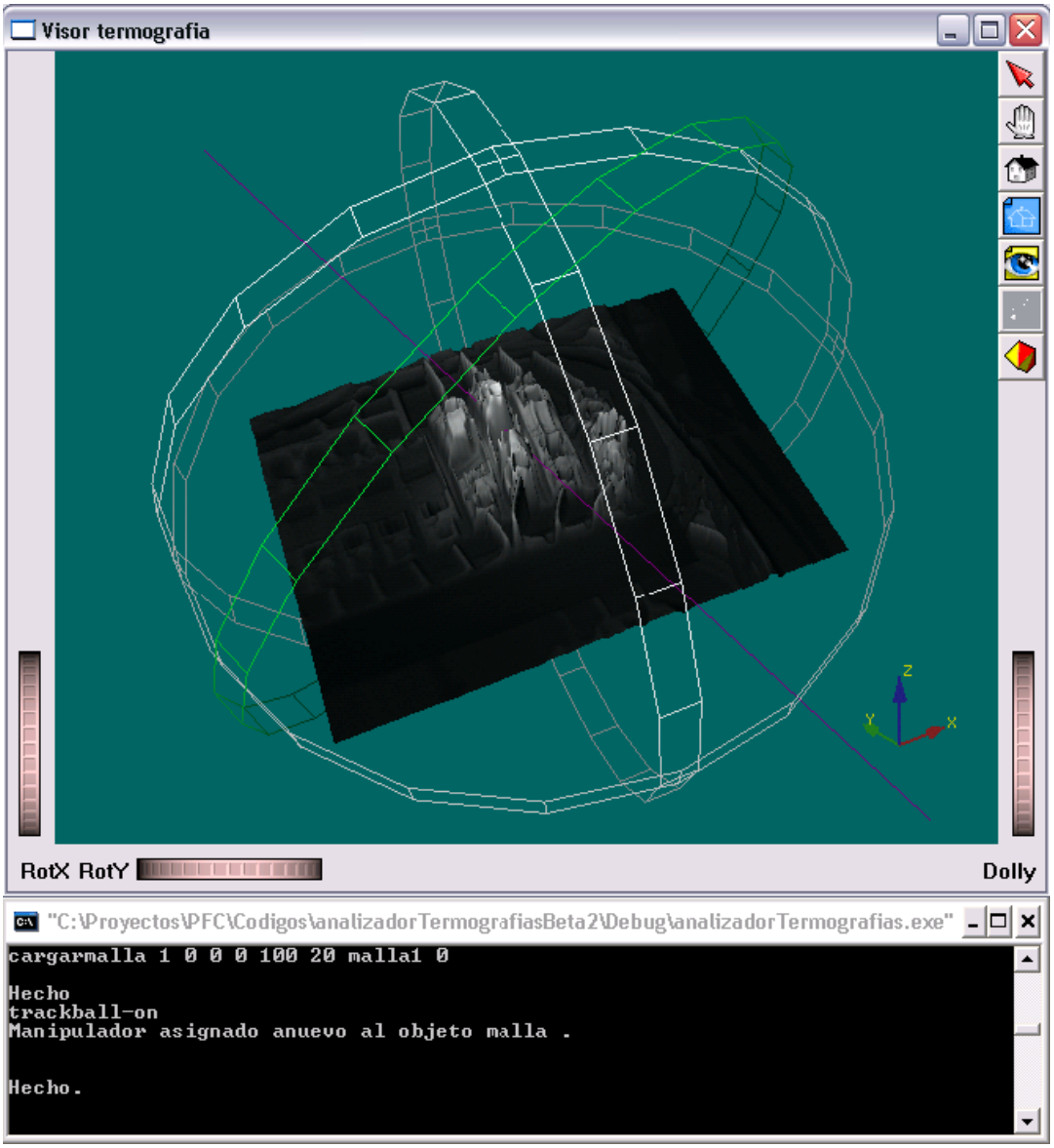


Figura 35. Ejemplo de uso del comando *trackball-on*.

Comando *trackball-off*.

Descripción: Desactiva el manipulador de tipo trackball asociado al objeto.
Acción previa: Debe de haber un objeto seleccionado.
Nº argumentos: -
sintaxis: `trackball-off`

Comando *box-on*.

Descripción: Asigna al objeto seleccionado un manipulador del tipo *box*.
Acción previa: Debe de haber un objeto seleccionado.
Nº argumentos: -
sintaxis: `box-on`

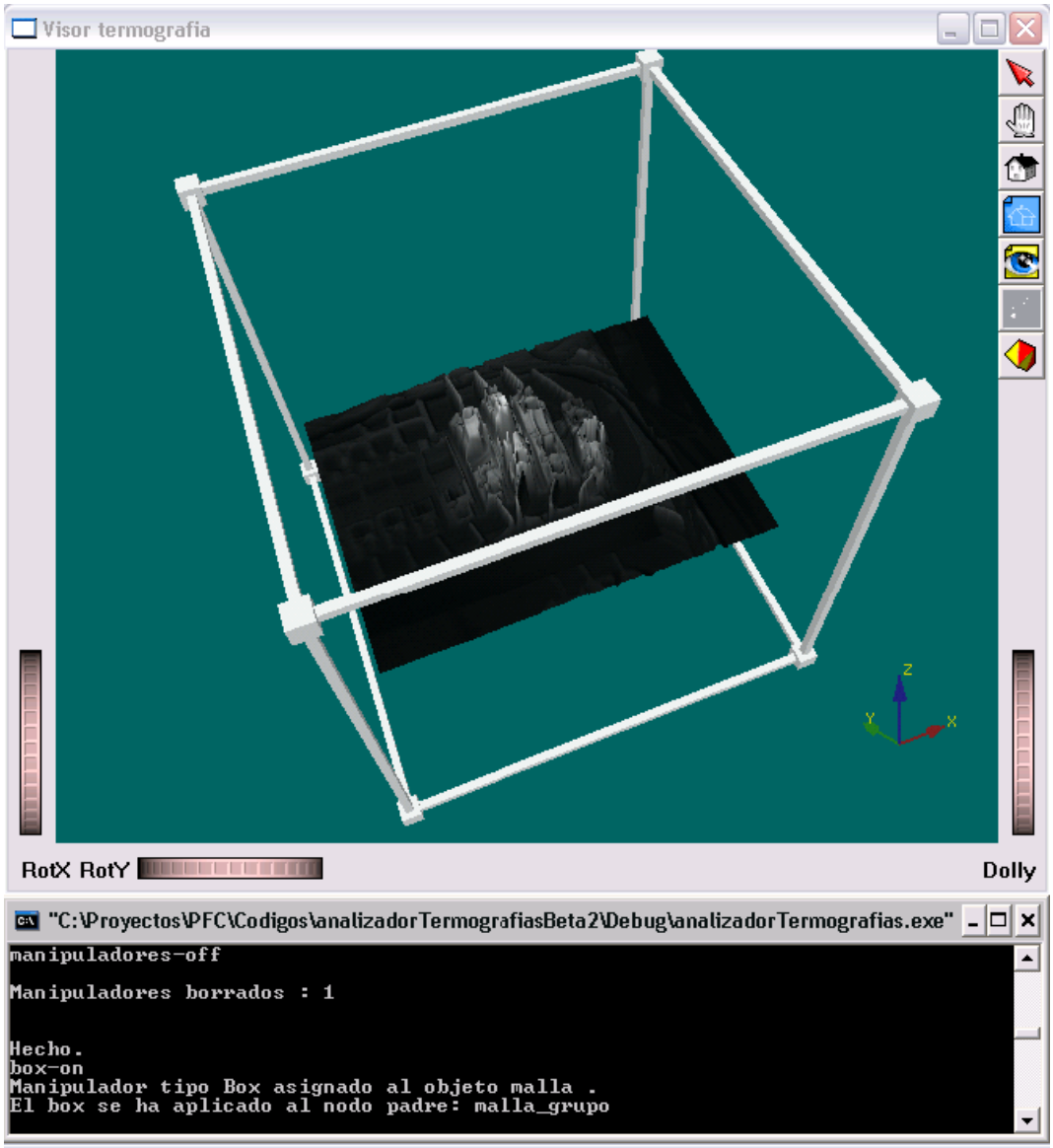


Figura 36. Ejemplo de uso del comando *box-on*.

Comando *box-off*.

Descripción: Desactiva el manipulador de tipo box asociado al objeto.
Acción previa: Debe de haber un objeto seleccionado.
Nº argumentos: -
sintaxis: `box-off`

Comando *manipuladores-off.*

Descripción: Desactiva todos los manipuladores de todos los objetos que tengan asociado algún tipo de manipulador.
Acción previa: -
Nº argumentos: -
sintaxis: `manipuladores-off`

Comando *luz-on.*

Descripción: Asigna una luz posicional al objeto seleccionado en las coordenadas (en el espacio del objeto) y el color indicado.
Acción previa: Debe de haber un objeto seleccionado.
Nº argumentos: 6
sintaxis: `luz-on fila columna altura R G B`

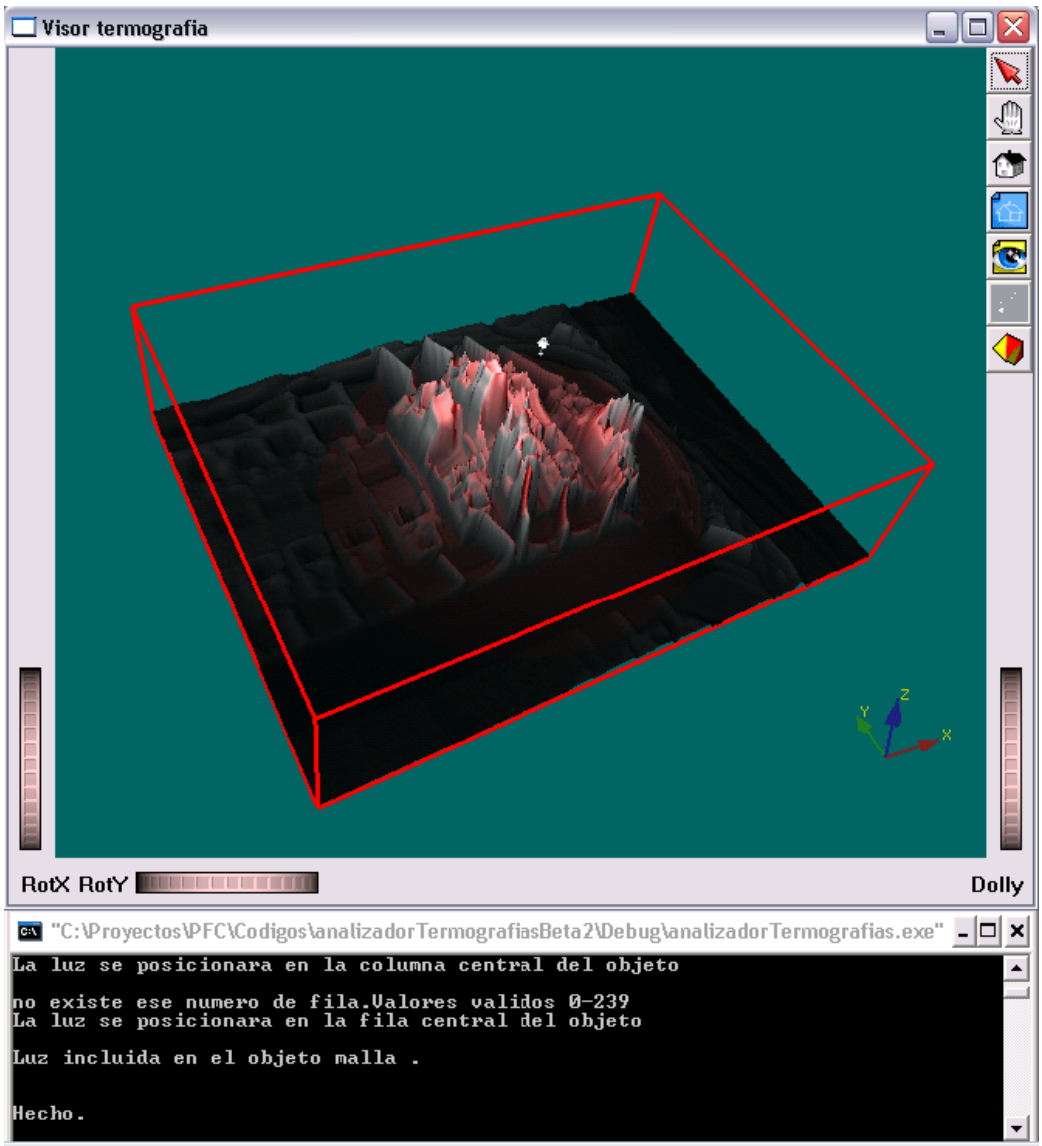


Figura 37. Ejemplo de uso del comando *luz-on*.

fila indica la fila del conjunto de vértices que conforman la malla/nube donde se ubicará la luz. En caso de un valor –1, la luz se posicionará en la fila central.
columna indica la fila del conjunto de vértices que conforman la malla/nube donde se ubicará la luz. En caso de un valor –1, la luz se posicionará en la fila central.
altura se refiere al valor de la coordenada Z donde se ubicará la luz.
RGB indican las componentes del color de la luz.

Comando *luz-off.*

Descripción: desactiva la luz posicional del objeto seleccionado.
Acción previa: Debe de haber un objeto seleccionado.
Nº argumentos: -
sintaxis: `luz-off`

Comando *posicionluz.*

Descripción: posiciona, en las coordenadas indicadas, una luz asociada a un objeto.
Acción previa: Debe de haber un objeto seleccionado y que éste tenga una luz asociada.
Nº argumentos: 3
sintaxis: `posicionluz coordX coordY coordZ`

Comando *colorluz.*

Descripción: establece el color de la luz.
Acción previa: Debe de haber un objeto seleccionado.
Nº argumentos: -
sintaxis: `colorluz R G B`

RGB deben de ser valores comprendidos entre 0 y 255.

Comando *luces-off.*

Descripción: desactiva todas las luces de todos los objetos que tengan asociado algún tipo de luz.
Acción previa: -
Nº argumentos: -
sintaxis: `luces-off`

Comando *crearmalla.*

Descripción: crea una malla (*simple grid*) sin utilizar ningún tipo de dato de entrada. Los vértices de la misma están separados el valor especificado. Valor de la coordenada Z a 0.
Acción previa: -

Nº argumentos: 7

sintaxis:

```
crearmalla numFilas numColumnas valorZ
           R G B separacion nombre
```

numFilas indica el número de filas para la nueva malla.

numColumnas indica el número de columnas para la nueva malla.

RGB definen el color aplicado a la malla.

separacion indica el valor de separación entre los vértices que componen la malla.

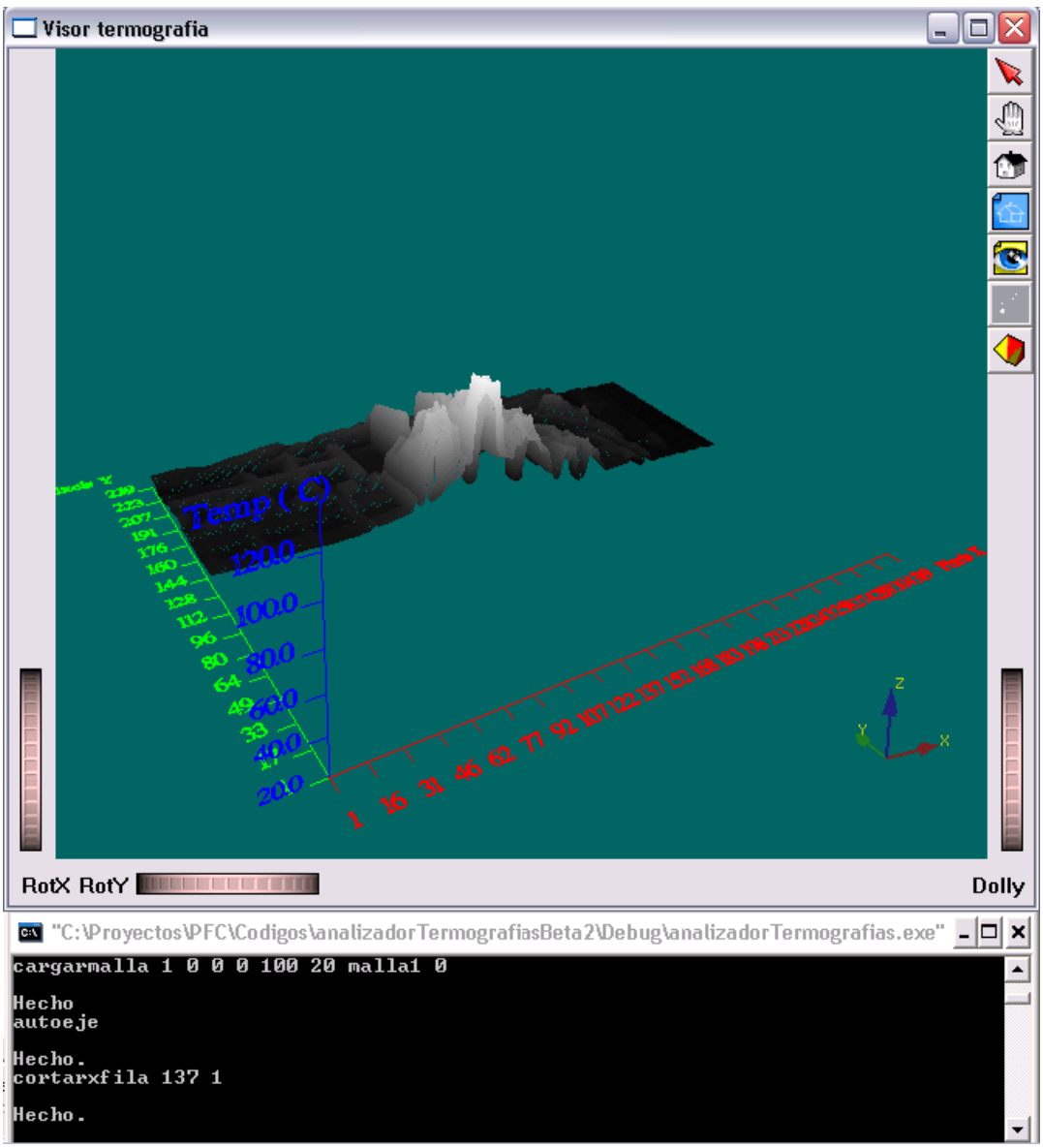


Figura 38. Ejemplo de uso del comando *cortarxfila*.

Comando *cortarxfila.*

Descripción: Corta el objeto seleccionado por la fila indicada y eliminando la zona especificada.

Acción previa: Debe de haber un objeto seleccionado.

Nº argumentos: 2

sintaxis: `cortarxfila numfila  zona_a_eliminar`

numfila Representa el número de la fila por donde se cortará el objeto.

zona_a_eliminar. Para un valor de 1 se elimina la zona inferior del objeto(valores de fila<numfilas). 0 se elimina la zonasuperior.

Limitaciones: No aplicable a objetos tipo *mallat* (*triang. grid*).

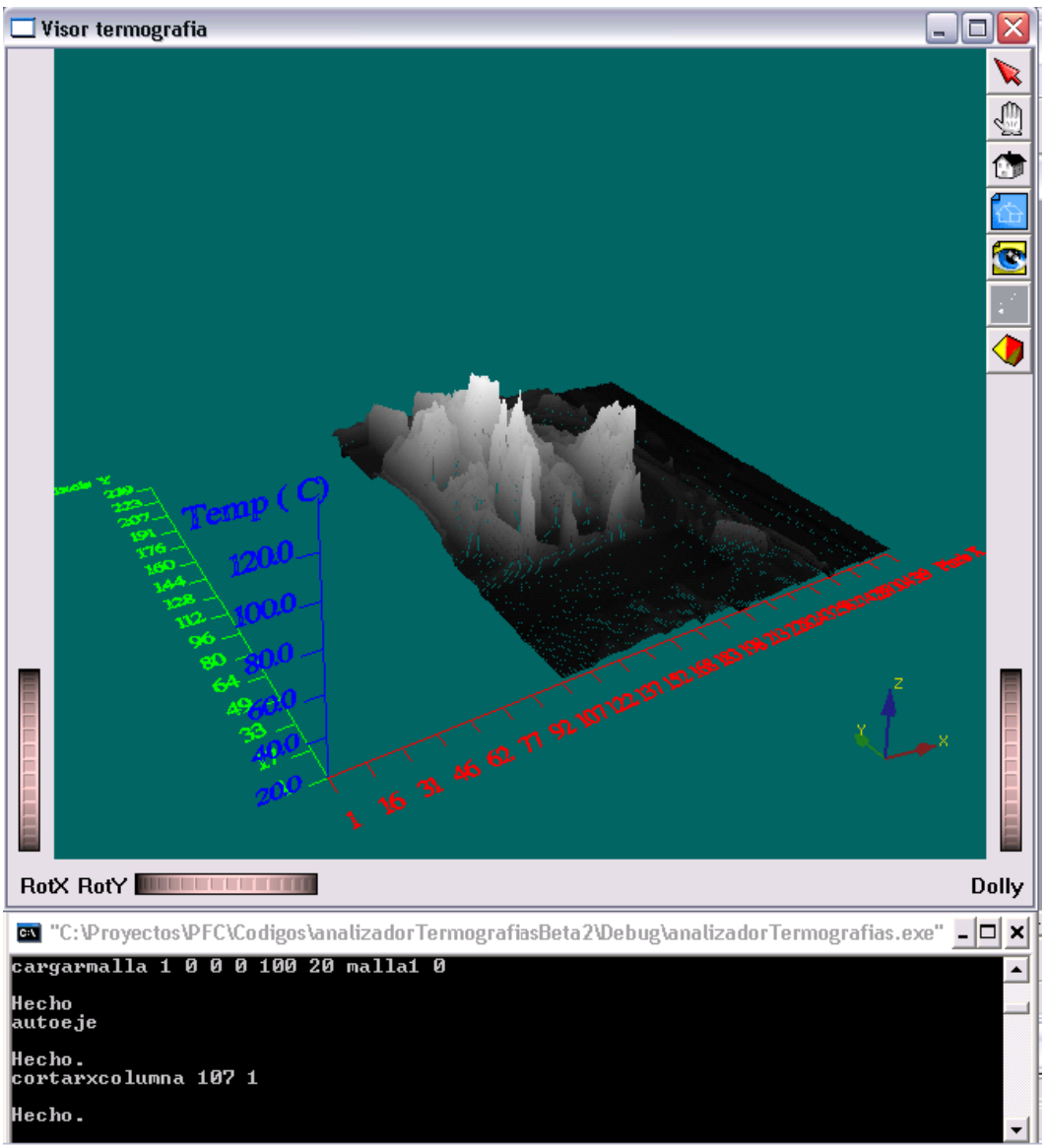


Figura 39. Ejemplo de uso del comando *cortarxcolumna*

Comando *cortarxcolumna.*

Descripción: Corta el objeto seleccionado por la columna indicada y eliminando la zona especificada.

Acción previa: Debe de haber un objeto seleccionado.

Nº argumentos: 2

sintaxis: `cortarxcolumna numcolumna zona_eliminar`

numcolumna Representa el número de la columna por donde se cortará el objeto.

zona_eliminar Valor 1. Se elimina la zona izquierda (valores de columna<numcolumnas) del objeto. 0 se elimina la zona superior.

Limitaciones: No aplicable a objetos tipo *mallat* (*triang. grid*).

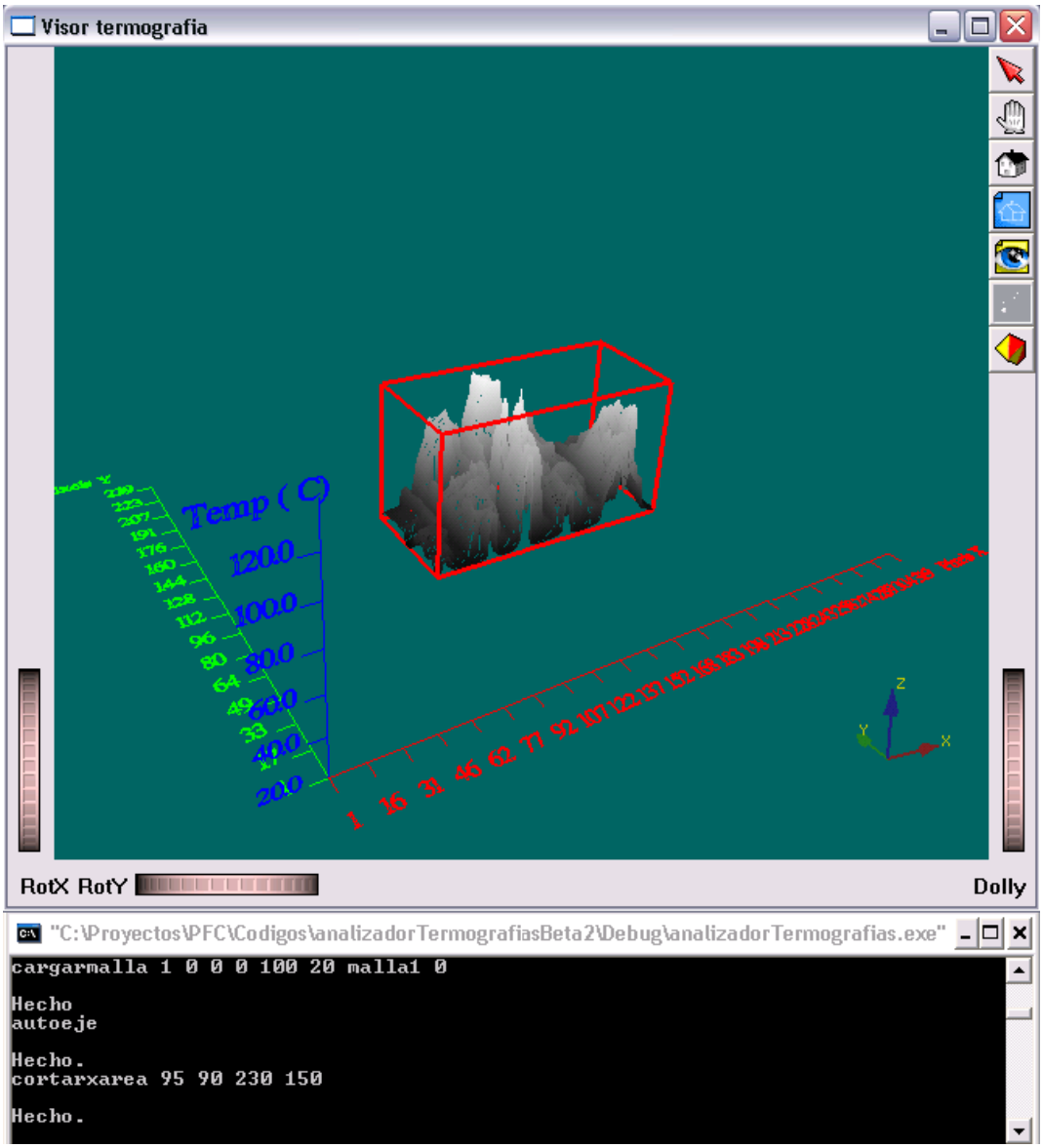


Figura 40. Ejemplo de uso del comando *cortarxarea*.

Comando *cortarxarea.*

Descripción: Corta una área especifica del objeto seleccionado definida por un rectángulo.
Acción previa: Debe de haber un objeto seleccionado.
Nº argumentos: 4
sintaxis: `cortarxarea X1 Y1 X2 Y2`

X1 Y1 primer vértice del rectángulo.
X2 Y2 segundo vértice para el rectángulo.

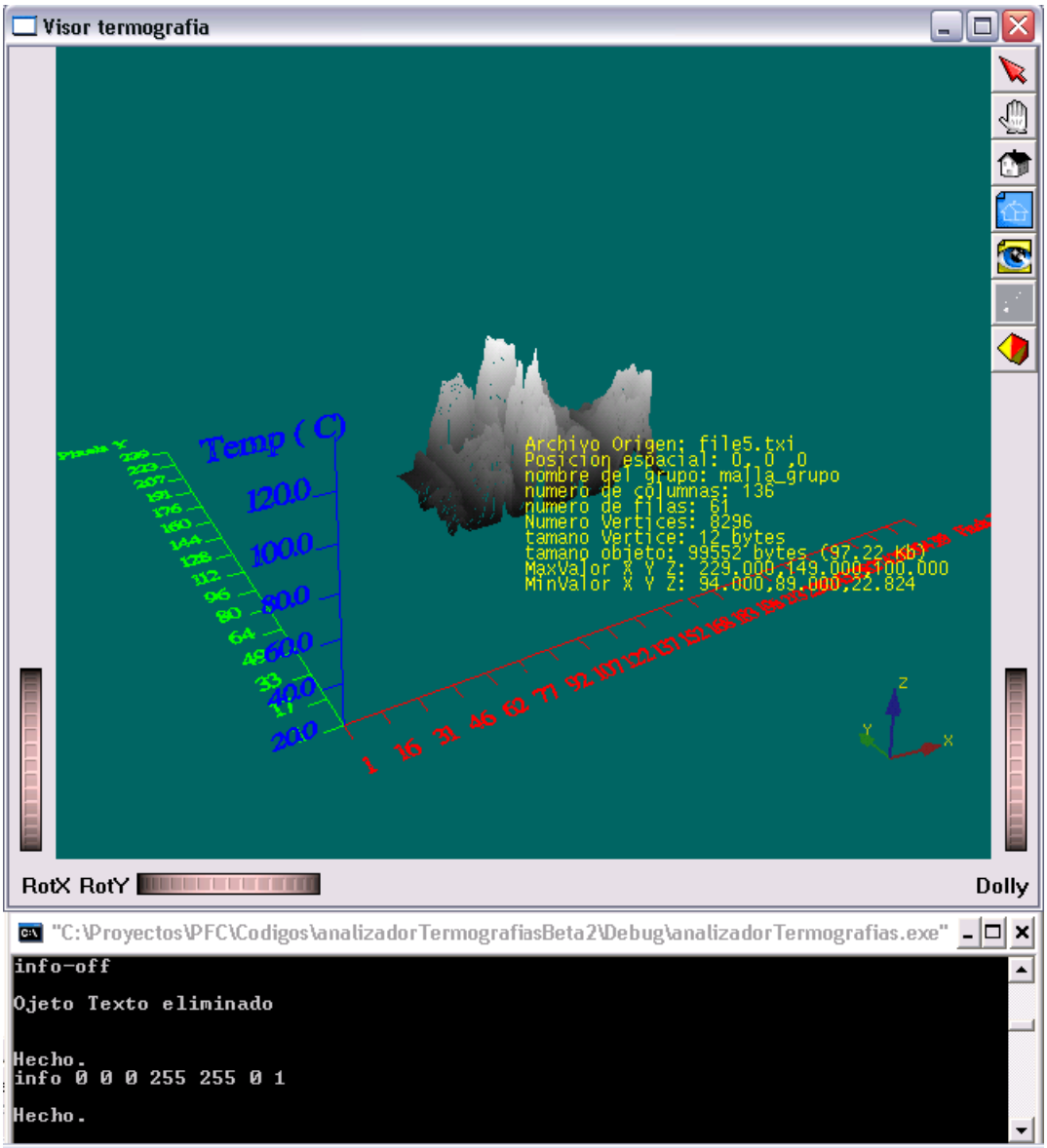


Figura 41. Ejemplo de uso del comando *info*.

Comando *info.*

Descripción: Muestra en la línea de comandos y/o en el visor información sobre el objeto seleccionado.

Acción previa: Debe de haber un objeto seleccionado.
Nº argumentos: 7
sintaxis: `info coordZ R G B mostrar`

mostrar Un valor de 1 incluye la información en la ventana del visor3d. 0 solo se muestra en la línea de comandos. 2 muestra la información tanto en la ventana de consola como en la del visor.

RGB Definen el color para el texto mostrado en la ventana del visor. Se requieren valores comprendidos entre 0 y 255. Los valores se ignoran en caso de que mostrar tome el valor 0.

Comando *info-off*.

Descripción: Elimina el texto2D de información del objeto seleccionado
Acción previa: Debe de haber un objeto seleccionado.
Nº argumentos: -
sintaxis: `info-off`

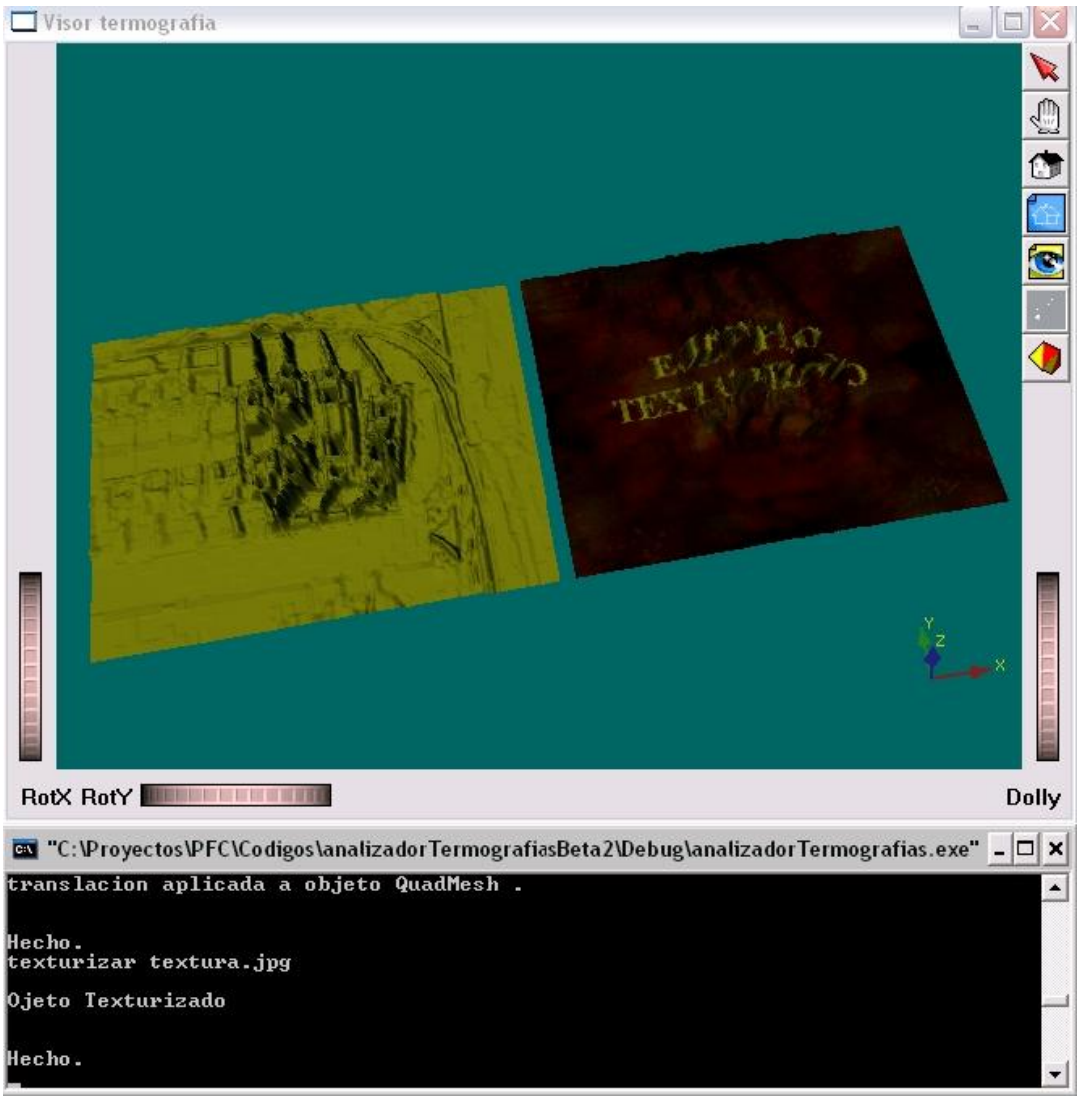


Figura 42. Ejemplo de uso del comando *texturizar*.

Comando *texturizar.*

Descripción: Texturiza el objeto seleccionado con el archivo de imagen indicado.
Acción previa: Debe de haber un objeto seleccionado.
Nº argumentos: 1
sintaxis: `texturizar archivo_bitmap`

archivo_bitmap Archivo de textura a utilizar. Solo están soportados los formatos: .jpeg, .jpg, .png y .rgb

Comando *texturizar-off.*

Descripción: Elimina la textura del objeto seleccionado.
Acción previa: Debe de haber un objeto seleccionado.
Nº argumentos: 1
sintaxis: `texturizar-off`

Comando *filtro.*

Descripción: Filtra los valores de temperatura según el tipo y el valor de temperaturas de corte especificados.
Acción previa: Debe de haber un objeto seleccionado.
Nº argumentos: Variable:
7
3
sintaxis:

```
filtro          Tempcorte Rsup Gsup
                Bsup Rinf Ginf Binf tipo

filtro          TempCorte tipo
```

tipo 0=paso bajo, 1= paso alto.
R/G/Binf indican el color para los vértices de las zonas con temperatura<TempCorte. Valores entre 0-255.
R/G/Bsup indican el color para los vértices de las zonas con temperatura>TempCorte. Valores entre 0-255.
Para valores negativos se utilizara el mismo valor que el contenido en el color original del objeto. Para valores superiores a 255 se usará el mismo valor que el contenido en el color del fondo de la ventana.

Se permiten combinaciones de las posibilidades anteriores. La siguiente expresión:

```
filtro 68.5 -1 255 300 -1 300 -1 1
```

Realizaría un filtro de temperaturas paso alto con temperatura de corte=68.5 ºC, estableciendo las componentes de color para los puntos de temperatura>Tempcorte como *Rsup*=sin modificar,*Gsup*=255 y *Bsup*= componente B del color de fondo de la pantalla. Las componentes de color para los puntos con temperatura<Tempcorte se establecen a *Rinf*=sin modificar, *Ginf*= componente G del color de fondo de la pantalla, *Binf*=sin modificar.

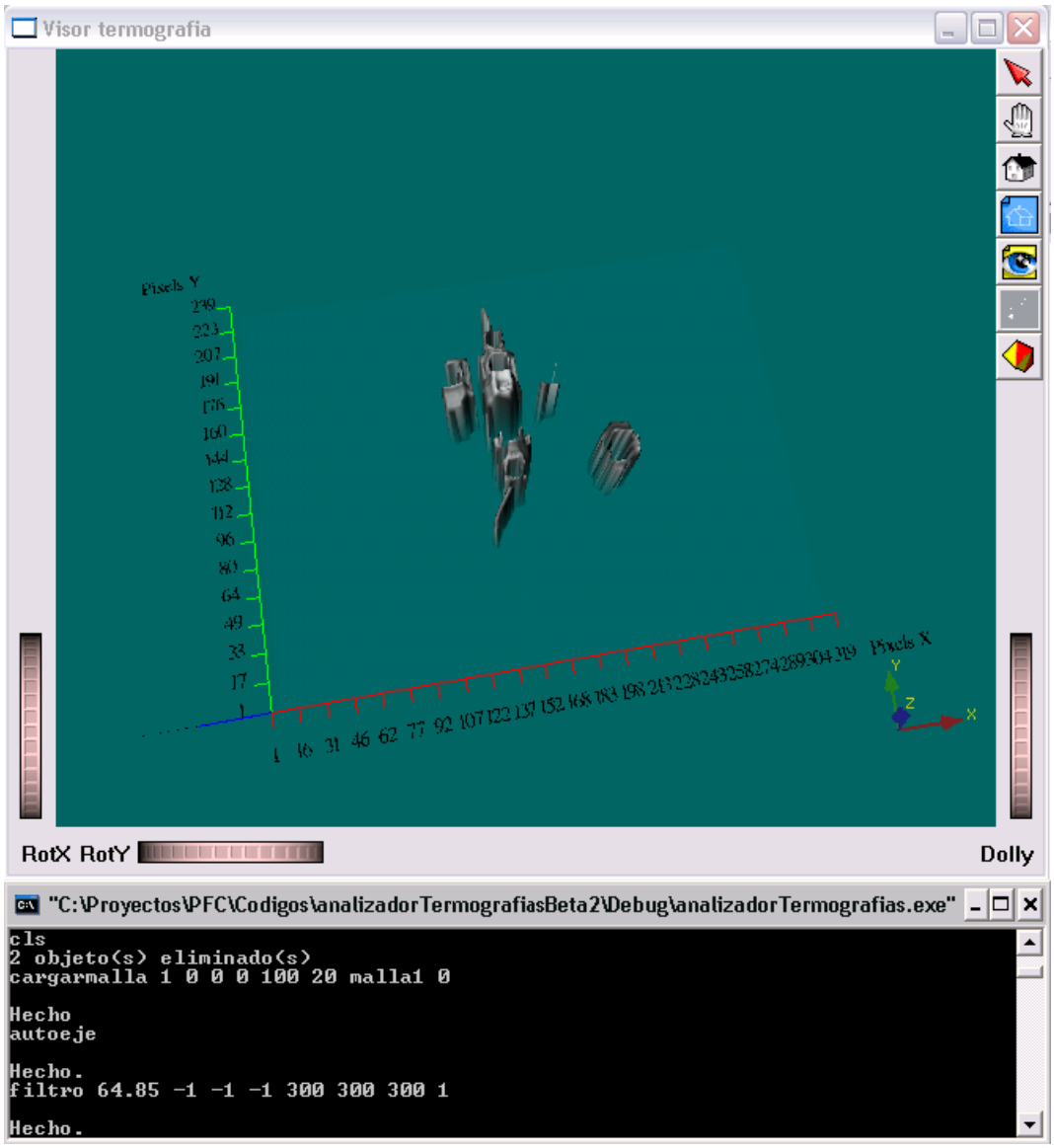


Figura 43. Ejemplo de uso del comando *filtro*.

Comando *filtrobanda.*

Descripción:	Filtra (fuerza a valor 0) los valores de temperatura según el tipo especificado y temperaturas de corte especificadas
Acción previa:	Debe de haber un objeto seleccionado.
Nº argumentos:	Variable: 12, 3
sintaxis:	`filtrobanda TempInf TempSup Rsup Gsup Bsup Rbanda Gbanda`

```
                    Bbanda  Rinf  Ginf  Binf
                    tipo

filtrobanda         TempInf TempSup tipo
```

tipo	0=rechaza banda, 1=pasa banda.
TempSup	Establece la temperatura de corte superior.
TempInf	Establece la temperatura de corte inferior.
R/G/Bsup	Definen las componentes de color para las temperaturas superiores a las de corte.
R/G/Bbanda	Definen las componentes de color para las temperaturas de la zona de banda.
R/G/Binf	Definen las componentes de color para las temperaturas por debajo de la de corte.

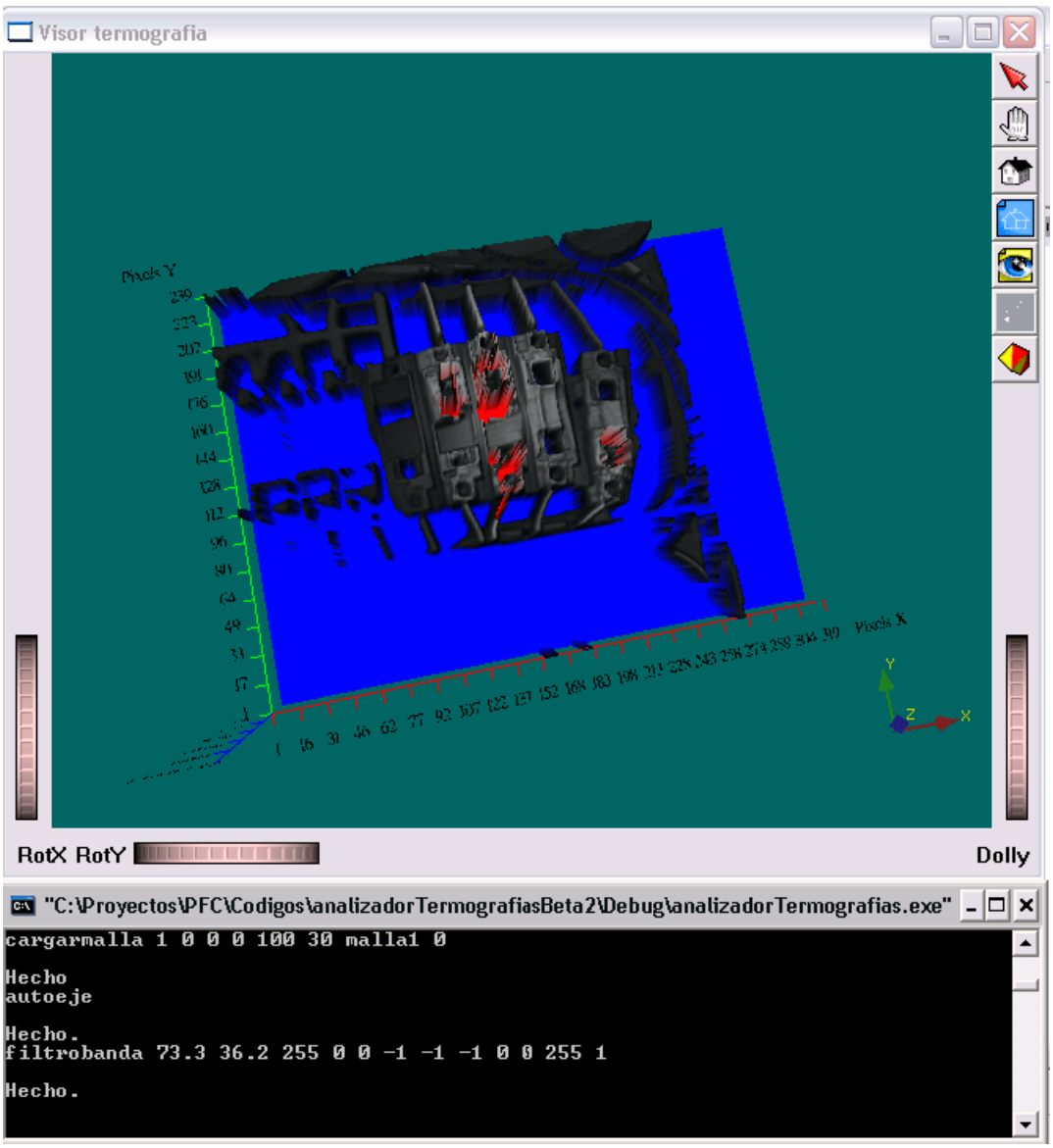


Figura 44. Ejemplo de uso del comando *filtrobanda* (en este caso un pasa banda).

Para las componentes de color tendremos que para:

- Valores de 0-255 la componente cogerá dicho valor.
- Para valores negativos se utilizara el mismo valor que el contenido en el color original del objeto.
- Para valores superiores a 255 se usará el mismo valor que el contenido en el color de fondo de la ventana.

Comando *referencia*.

Descripción: muestra/oculta los ejes de referencia del espacio 3D.
Acción previa: -
Nº argumentos: -
sintaxis: `referencia`

Comando *seleccionararea*.

Descripción: Permite el uso del comando *cortarxarea* de una manera gráfica usando el raton. Al ejecutar el comando el programa permanece a la espera de que el usuario seleccione los dos puntos que definen el área de interés mediante la pulsación del botón izquierdo del ratón.
Acción previa: Debe de haber un objeto seleccionado.
Nº argumentos: -
sintaxis: `seleccionararea`
Limitaciones: No aplicable a objetos tipo *mallat* (*triang. grid*) ni nube de puntos.

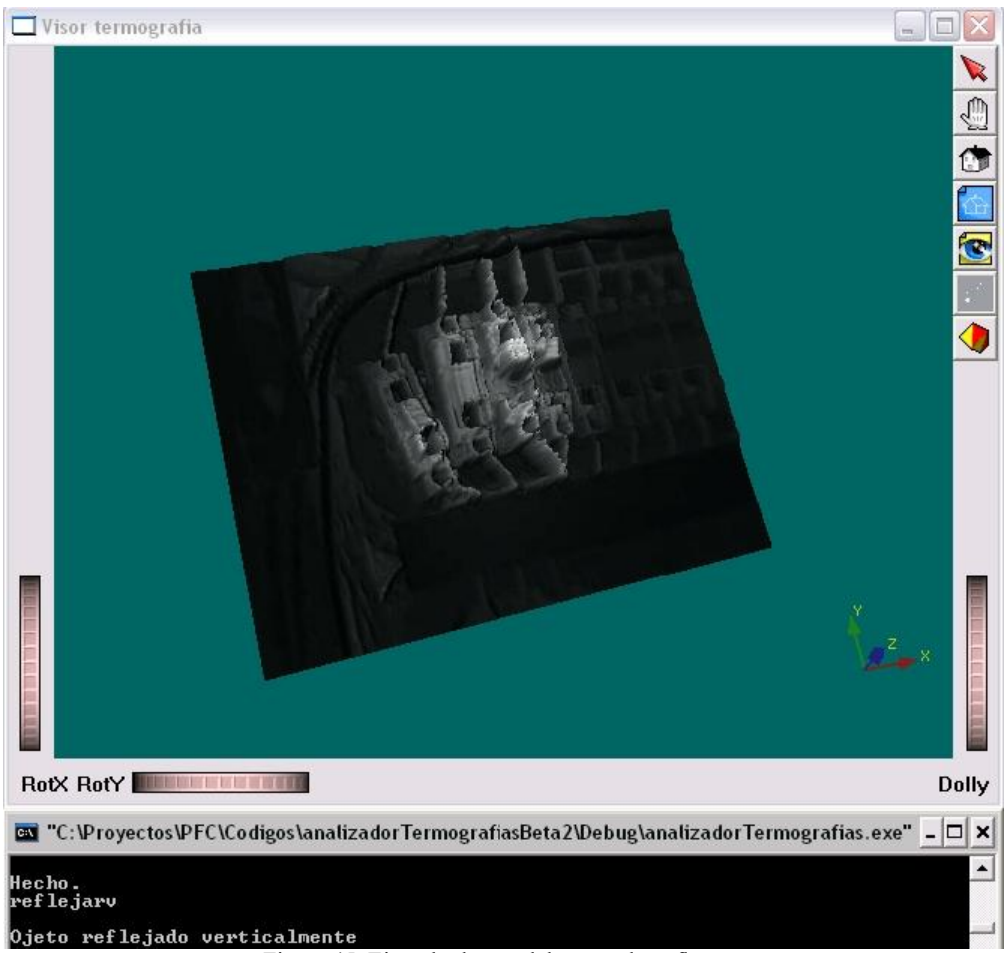


Figura 45. Ejemplo de uso del comando *reflejarv*.

Comando *reflejarv*.

Descripción: Refleja verticalmente el objeto seleccionado.
Acción previa: Debe de haber un objeto seleccionado.
Nº argumentos: -
sintaxis: `reflejarv`
Limitaciones: No aplicable a objetos tipo *mallat* (*triang. grid*)

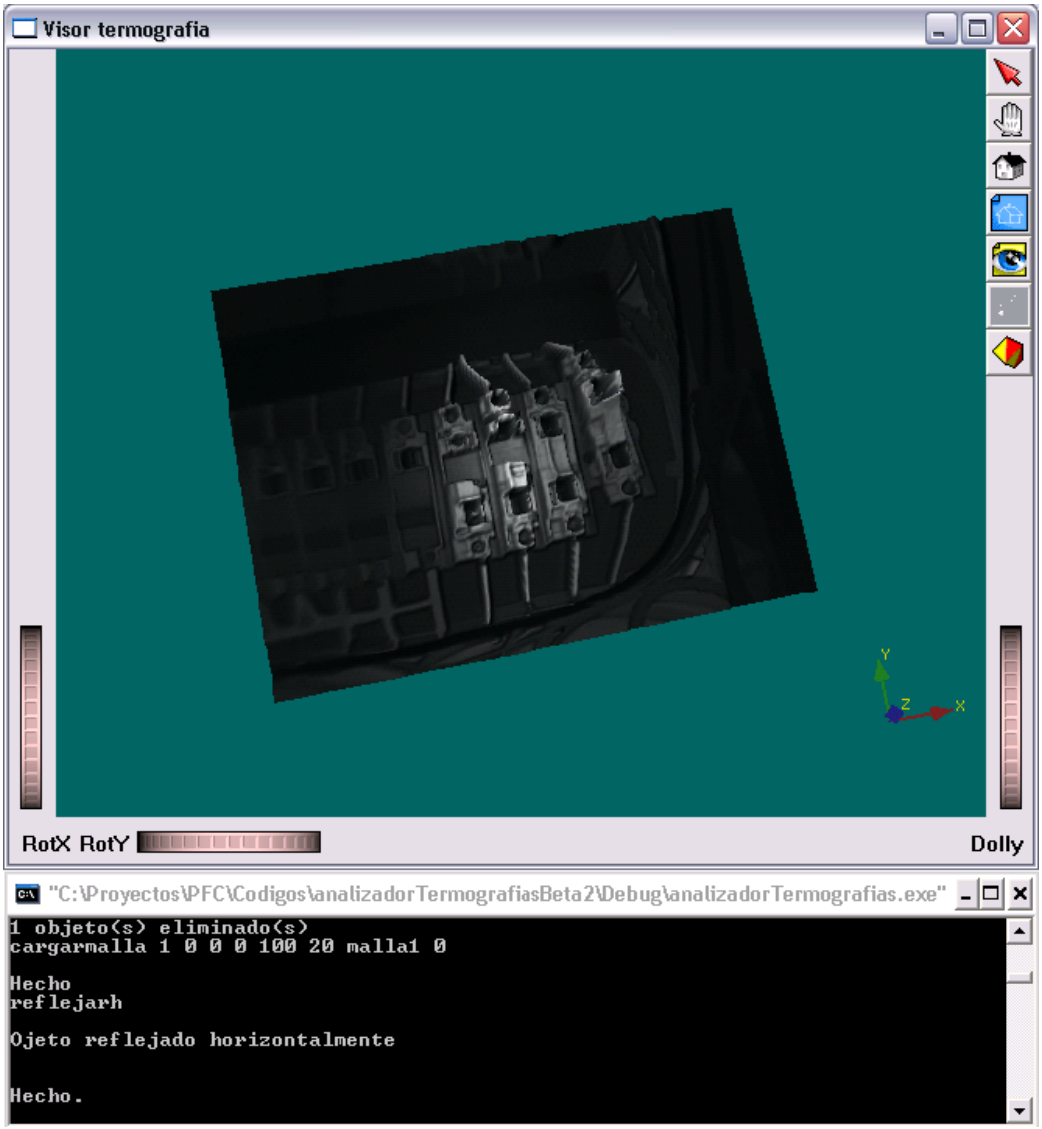


Figura 46. Ejemplo de uso del comando *reflejarh*.

Comando *reflejarh*.

Descripción: Refleja horizontalmente el objeto seleccionado.
Acción previa: Debe de haber un objeto seleccionado.
Nº argumentos: -
sintaxis: `reflejarh`
Limitaciones: No aplicable a objetos tipo *mallat* (*triang. grid*).

Comando *asignarpaleta.*

Descripción: Asigna al objeto seleccionado la paleta de color del archivo especificado.

Acción previa: Debe de haber un objeto seleccionado. Debe de existir el archivo *datospal.txt* en el directorio de ejecución en caso de no indicar el argumento *archivo_paleta*.

Nº argumentos: Variable:

1. Se indica el archivo de texto conteniendo la paleta que se aplicará.
- Se aplica la paleta contenida en el archivo temporal *datospal.txt*.

sintaxis:
```
asignarpaleta archivo_paleta
asignarpaleta
```

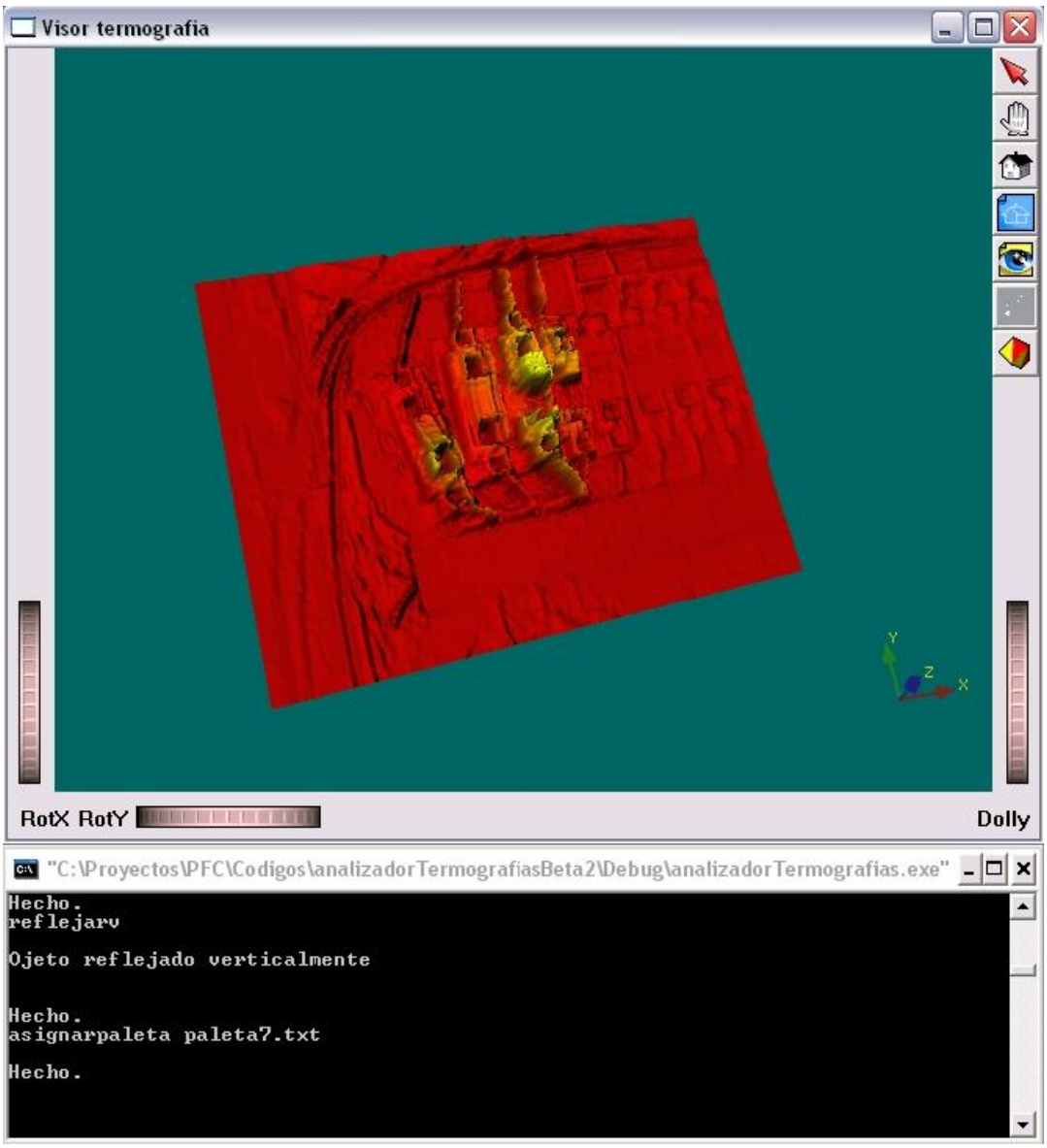


Figura 47. Ejemplo uso del comando *asignarpaleta* (paleta *hotmetal*).

Comando *invertirc.*

Descripción: Invierte los colores de cada uno de los vértices del objeto seleccionado
Acción previa: Debe de haber un objeto seleccionado.
Nº argumentos: -
sintaxis: `invertirc`

Comando *v2bin.*

Descripción: Crea en el directorio de ejecución un archivo binario, con el nombre especificado, que contiene las temperaturas de cada uno de los vértices del objeto seleccionado.
Acción previa: Debe de haber un objeto seleccionado.
Nº argumentos: 1
sintaxis: `v2bin fichero_binario`

fichero_binario indica el nombre con el que se guardará el archivo.

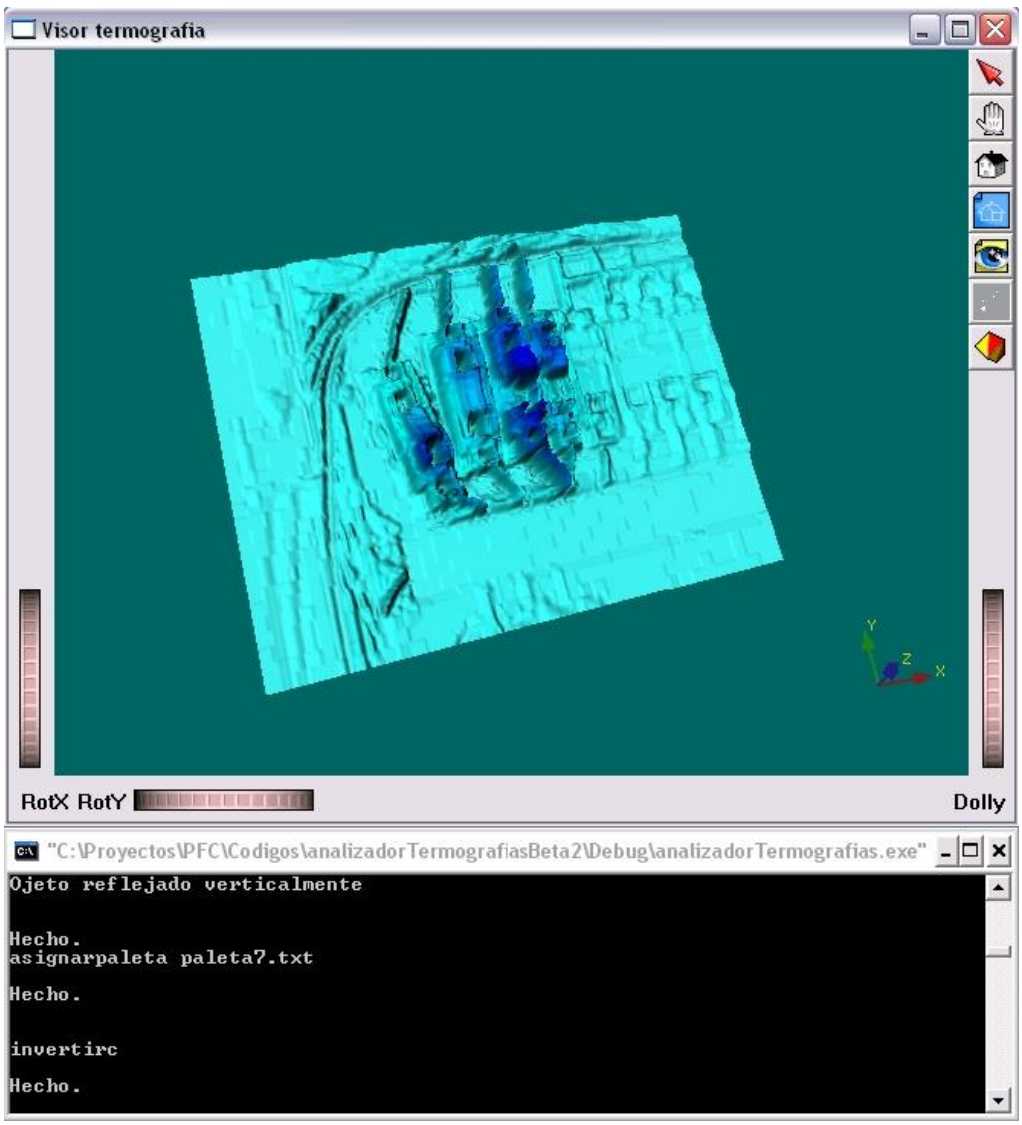


Figura 48. Ejemplo de uso del comando *invertirc*. (Inversión realizada a la figura 47)

Comando *isoterma.*

Descripción:	Asigna el color especificado a la franja de temperaturas indicada.
Acción previa:	Debe de haber un objeto seleccionado.
Nº argumentos:	1
sintaxis:	`isoterma  Tinicial Tfinal R G B`

Tinicial	Valor de temperatura que establece el inicio de la franja a la que se aplicará el color especificado mediante *colorFranja*.
Tfinal	Valor de temperatura que establece el final de la franja a la que se aplicará el color especificado mediante *colorFranja*.
R/G/B	Valor de las componentes del color que se aplicará a las temperaturas que se encuentren entre *Tinicial* y *Tfinal*. Valores de 0 a 255. Para valores superiores a 255 se usará el mismo valor que el contenido en el color del fondo de la ventana.

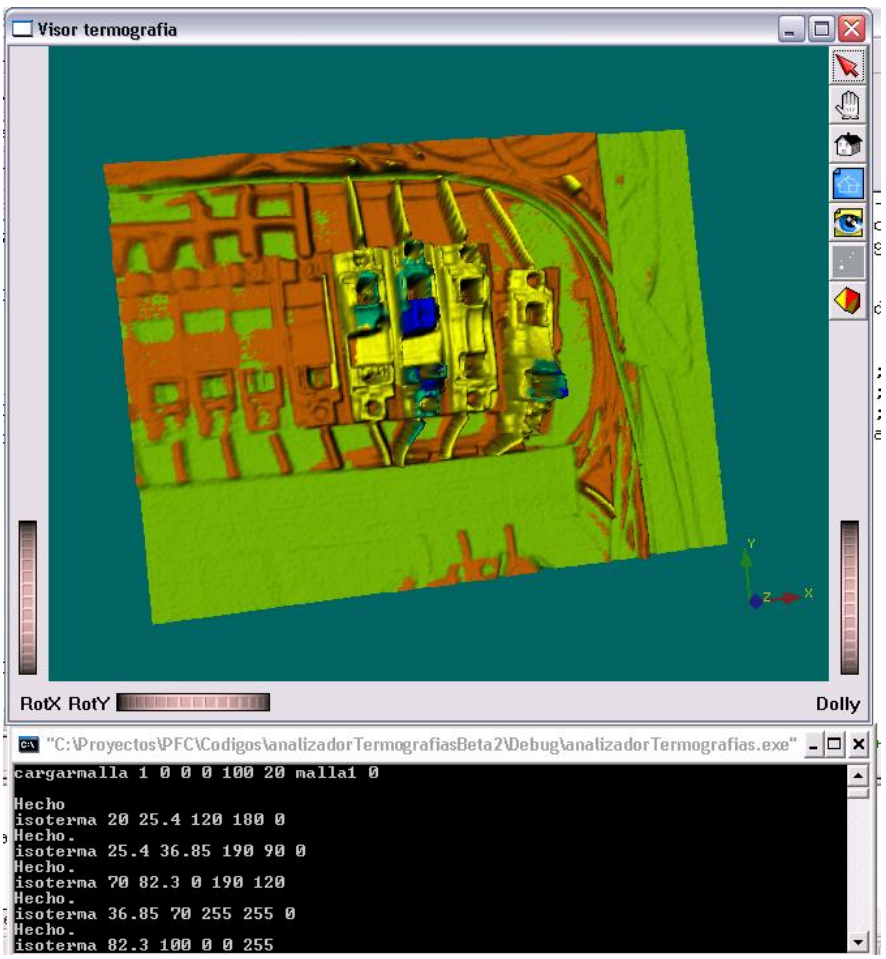


Figura 49. Ejemplo de uso del comando *isoterma*.

Comando *v2txt.*

Descripción: Crea en el directorio de ejecución , un archivo de texto, con el nombre especificado, que contiene información del objeto y las temperaturas de cada uno de los vértices que lo componen.
Acción previa: Debe de haber un objeto seleccionado.
Nº argumentos: 1
sintaxis:
```
v2txt archivo_texto
```

archivo_texto indica el nombre con el que se guardará el archivo.

Comando *h2txt.*

Descripción: Crea en el directorio de ejecución un archivo de texto, con el nombre especificado, que contiene información de las ocurrencias de los valores de temperatura y porcentaje que representan sobre el total de puntos de temperatura que componen la termografia.

Acción previa: Debe de haber un objeto seleccionado.
Nº argumentos: 1
sintaxis:
```
h2txt archivo_texto
```

archivo_texto Indica el nombre con el que se guardará el archivo.

Comando *promedio.*

Descripción: Calcula el valor medio de todos los datos de temperatura del objeto seleccionado mostrando el resultado por la consola.
Acción previa: Debe de haber un objeto seleccionado.
Nº argumentos: 1
sintaxis:
```
promedio
```

Comando *posicionar.*

Descripción: Traslada al objeto seleccionado a la posición indicada (en el espacio del mundo3D) por los argumentos *coordX coordY* y *coordZ*.
Acción previa: Debe de haber un objeto seleccionado.
Nº argumentos: 3
sintaxis:
```
posicionar coordX coordY coordZ
```

Comando *histo.*

Descripción: Crea un histograma de barras del objeto seleccionado y lo muestra como una gráfica independiente del objeto.

Acción previa: Debe de haber un objeto seleccionado.

Nº argumentos: Variable:
0. Se usan los títulos por defecto para los ejes X e Y.
2. Se indican los títulos de los ejes X e Y.

sintaxis:
```
histo
histo    tituloX      tituloY
```

tituloX Valor predeterminado para la versión sin argumentos es "indicePal".

tituloY Valor predeterminado para la versión sin argumentos es "Ocurrencias (porcentaje sobremaxOcurrencias)"

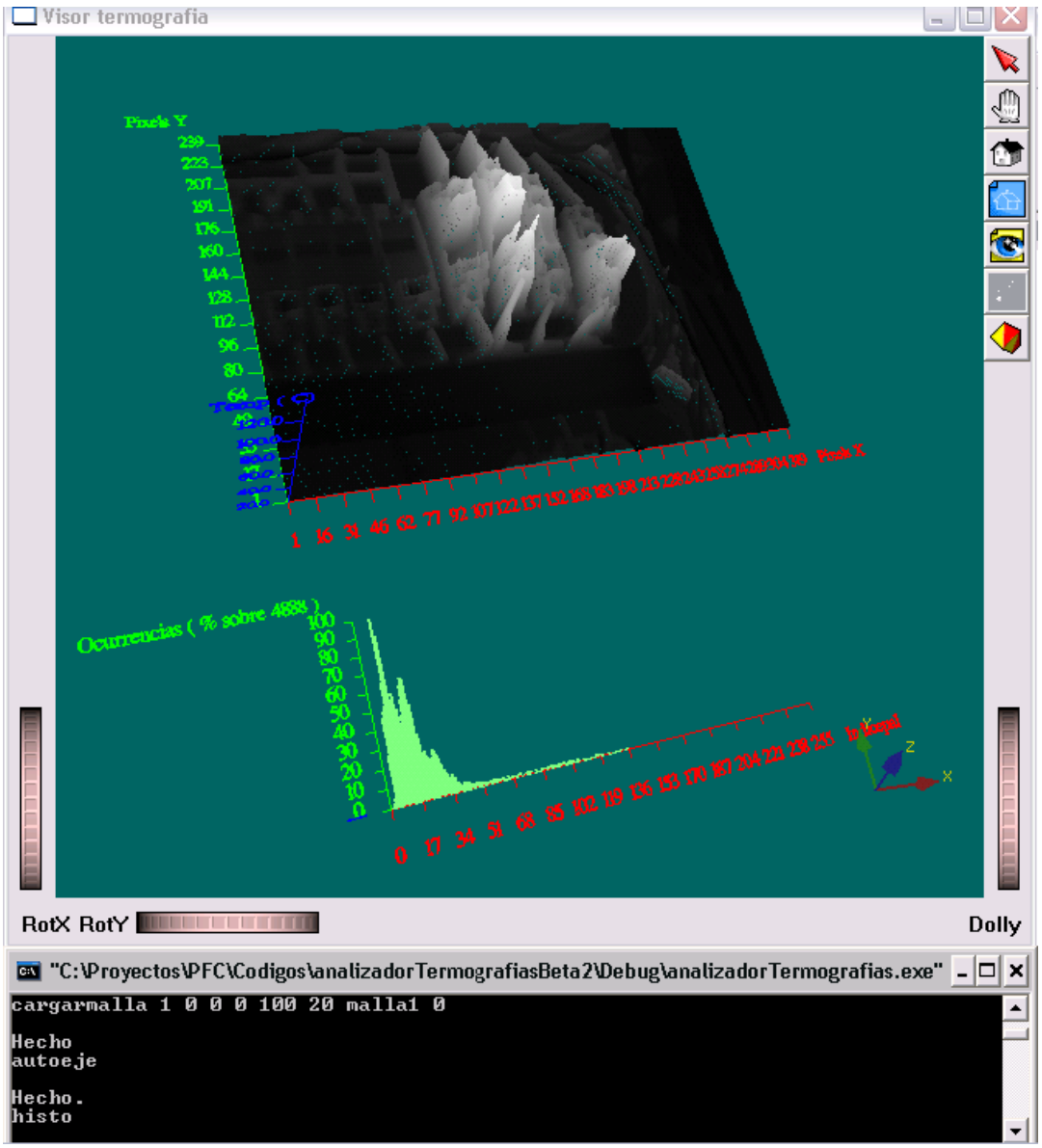


Figura 50. Ejemplo de uso del comando *histo*.

Comando *script.*

Descripción: Abre un archivo de *script* y ejecuta los comandos contenidos de forma secuencial y periódica según el tiempo indicado en el argumento *intervalo*.

Acción previa: -

Nº argumentos: 2

sintaxis:

```
script nombre_archivo intervalo
```

nombre_archivo	Indica el nombre del archivo *script* a abrir, con el *path* completo e incluida la extensión.
Intervalo	Especifica el tiempo en segundos entre la ejecución de dos comandos.

Comando *selobjeto.*

Descripción: Selecciona el objeto en pantalla con el nombre especificado.

Acción previa: -

Nº argumentos: 1

sintaxis:

```
selobjeto      nombre
```

nombre	Indica el nombre del objeto que se desea seleccionar.

Comando *etiqueta.*

Descripción: Crea una etiqueta de texto3D con el texto, las coordenadas y el color especificados.

Acción previa: -

Nº argumentos: 9

sintaxis:

```
etiqueta  texto_etiqueta size_letra X Y
          Z R G B idEtiqueta
```

Texto_etiqueta	Texto a mostrar en pantalla. La cadena de texto no puede contener espacios.
Size_letra	Tamaño de la letra fuente usada.
X/Y/Z	Coordenadas donde posicionar la etiqueta.
R/G/B	Componentes de color para el texto de la etiqueta. Valores válidos 0 a 255.
idEtiqueta	Cadena de texto que identifica a la etiqueta. No puede contener espacios.

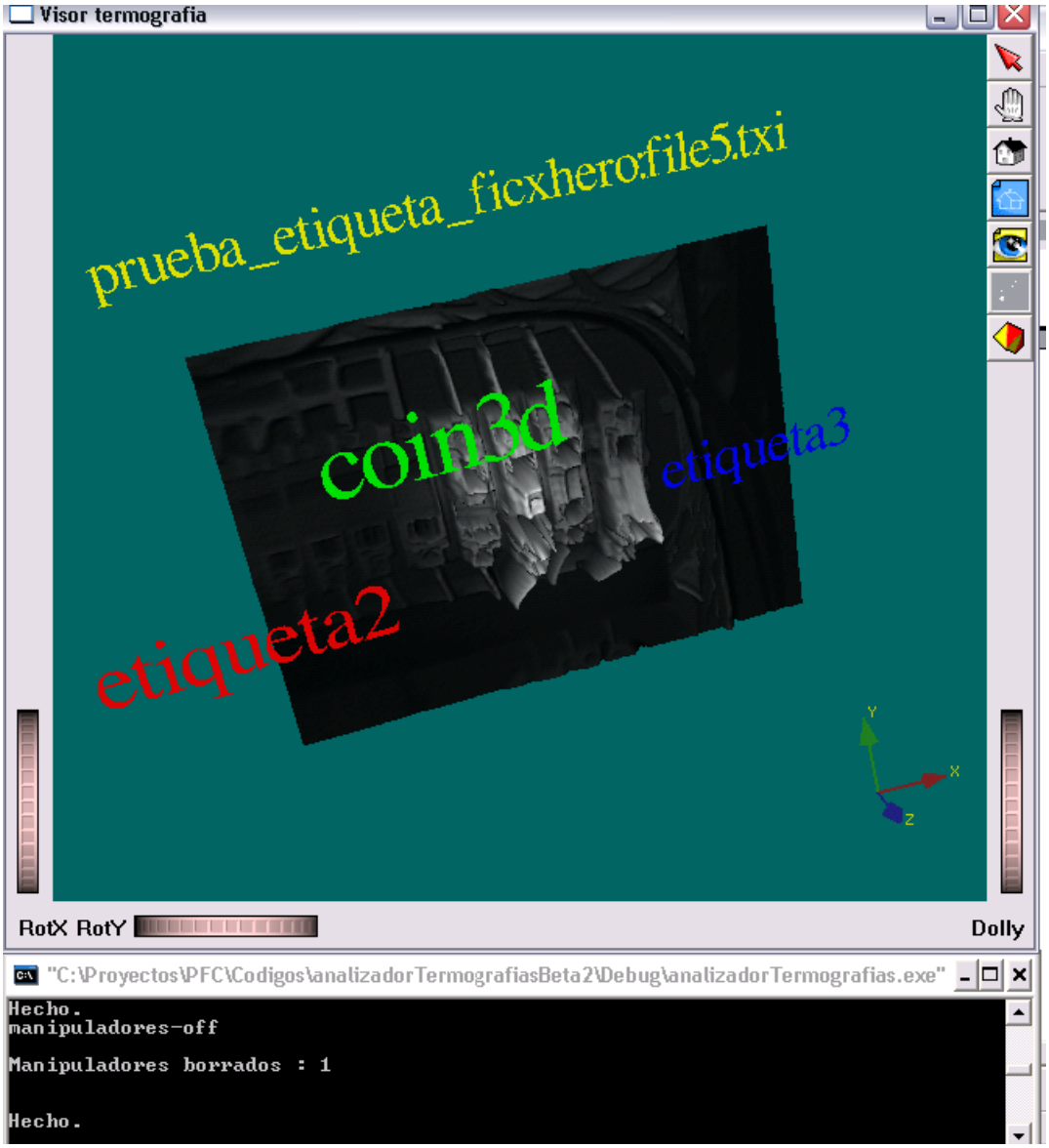


Figura 51. Ejemplo de uso del comando *etiqueta*.

Comando *autoenfoque.*

Descripción:	Establece automáticamente los valores necesarios para que la cámara enfoque a la totalidad de objetos contenidos en la ventana del visor.
Acción previa:	-
Nº argumentos:	-
sintaxis:	`autoenfoque`

Comando *poscamara.*

Descripción:	Posiciona la cámara en las coordenadas especificadas.
Acción previa:	-
Nº argumentos:	3
sintaxis:	`poscamara X Y Z`

X/Y/Z	Coordenadas absolutas para el posicionamiento de la cámara.

Comando *transcamara.*

Descripción: Translación de la cámara en los ejes indicados.
Acción previa: -
Nº argumentos: 3
sintaxis:

```
transcamara    X Y Z
```

X/Y/Z	Magnitud para la translación de la cámara. Valores diferentes de 0 producen una translación en el eje correspondiente.

Comando *orcamara.*

Descripción: Rotación de la cámara en el eje y con el ángulo especificado.
Acción previa: -
Nº argumentos: 4
sintaxis:

```
orcamara  X Y Z angulo
```

X/Y/Z	Para un valor igual a 0 en el respectivo eje, no se produce ninguna rotación. Valores diferentes de 0 producen la rotación del respectivo eje los grados indicados en *angulo.*
angulo	Especifica el ángulo a rotar en grados sexagesimales.

5.2 Uso del software Visor2D de Termografias.

El software Visor2D de termografias, intenta facilitar al usuario el manejo del software 3D de visualización de termografias. Ofrece acceso a todas las funciones del Visor3D y además añade algunas herramientas adicionales que pueden resultar útiles. Su ventana principal se muestra en la figura 52.

Las principales características del Visor2D de termografias son:

- Acceso a la mayoría comandos del Visor 3D de termografias mediante menús y botones.
- Facilita el manejo de múltiples ventanas de visores3D.
- Información en pantalla sobre la temperatura de cada uno de los pixeles que conforman la imagen termográfica.

- Generación de degradados personalizables (color simple y luminancia uniforme) que se pueden guardar o aplicar a la imagen 3D de la termografía a modo de paletas perceptivas.
- Establecimiento de etiquetas informativas de temperatura.
- Permite la gestión completa de las paletas de color (apertura, edición y guardar).
- Cálculo de las principales figuras de interés de la termografía (temperatura máxima, mínima y media)sin necesidad del uso del visor3D.
- Ajuste de los valores de emisividad para corrección de las medidas de temperatura. Ajuste de la temperatura ambiental.

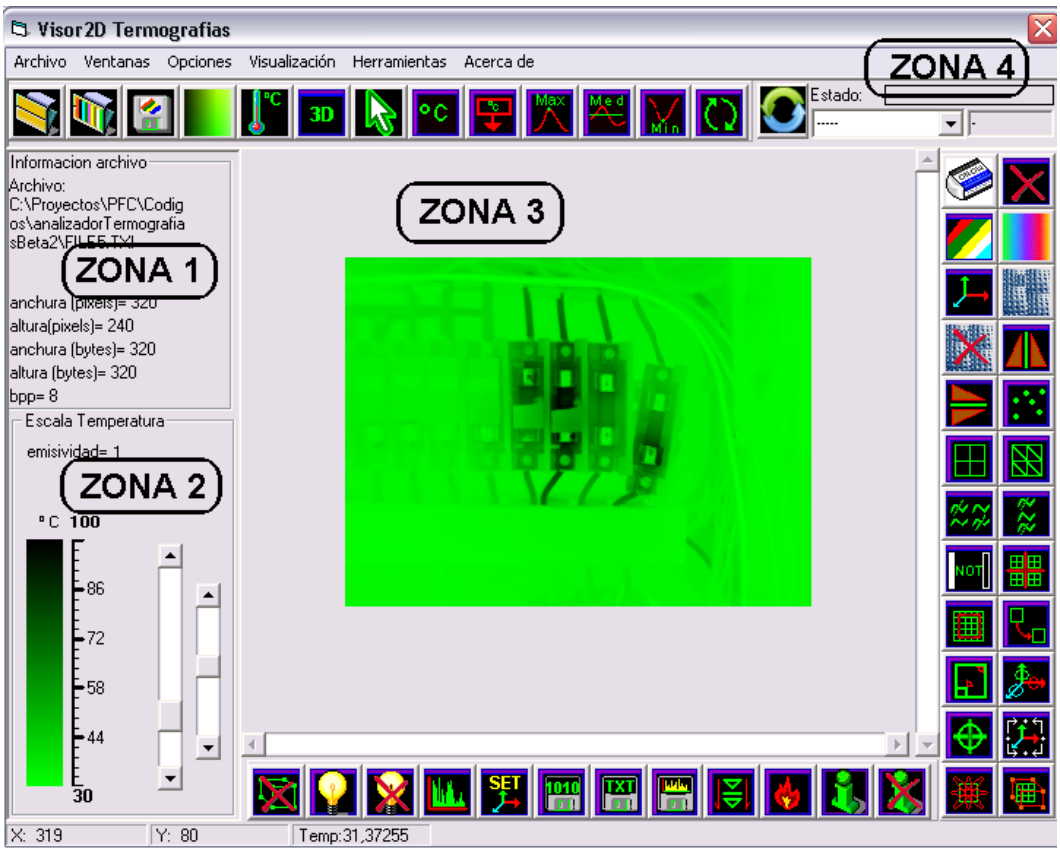


Figura 52. Vista general de la aplicación Visor2D desarrollada en el presente proyecto

Se aprecian cuatro zonas: área de información del archivo abierto (zona 1), información de la paleta de trabajo actual (zona 2) y zona de despliegue de la imagen termográfica (zona 3) y el área de información de estado (zona 4).

Descripción de los controles del programa.

5.2.1 Barra principal del programa.

Figura 53. Barra principal de la aplicación Visor2D.

1. Abre un archivo de tipo bitmap/binario en la zona de despliegue. Al abrir archivos bitmap se detecta si la imagen contiene paleta. Se pueden abrir archivos de diferentes *bpp* pero solo en el caso de que la imagen contenga una paleta de colores y estableciendo los valores de temperatura necesarios, la imagen podrá considerarse como una imagen representativa de una termografia.

2. Abre archivos de paletas de color. Cuando se abre una paleta se sustituye la paleta de trabajo actual por la abierta.

3. Guarda la paleta de trabajo actual en un archivo de texto para su posterior edición/uso.

4. Abre la ventana de generación de degradados, que permite generar, asignar y guardar los degradados personalizados por el usuario.

5. Abre el diálogo de configuración de los datos de temperatura que se utilizarán en el Visor 3D para realizar la representación de la termografia.

6. Abre una nueva ventana del visor de termografias 3D.

7. Cursor de selección por defecto.

8. Activa la información emergente en la imagen desplegada en la zona 3, ofreciendo información sobre la temperatura del punto donde se encuentra el cursor del ratón.

9. Establece etiquetas informativas sobre la temperatura del punto seleccionado en la imagen desplegada en zona 3.

10. Recorre todos los puntos que conforman la imagen termográfica devolviendo el valor máximo de temperatura encontrado y la última posición, en coordenadas X e Y, donde se encuentra.

11. Devuelve el valor medio calculado sobre todo los puntos que conforman la imagen.

12. Recorre todos los puntos que conforman la imagen termográfica devolviendo el valor mínimo de temperatura encontrado y la última posición, en coordenadas X e Y, donde se encuentra.

13. Fuerza la recarga de los datos de la imagen desplegada en la zona 3.

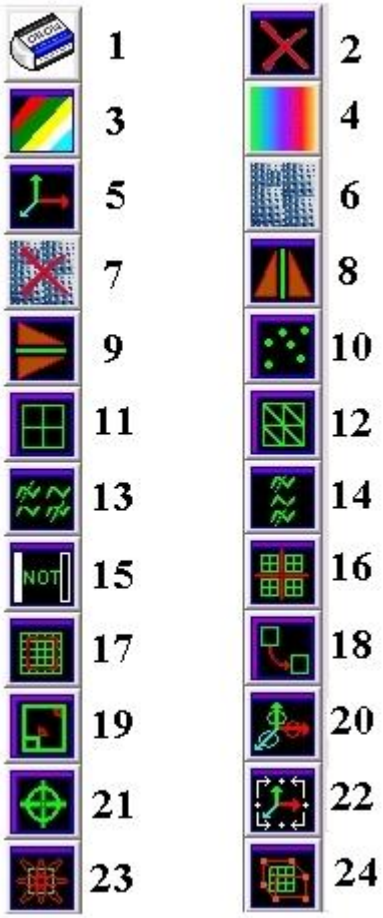
5.2.2 Barra1. Comandos para el visor 3D.

Los comandos de esta barra solo se activan al cargar un archivo válido, establecer los valores de configuración de temperatura adecuados y abrir como mínimo una ventana de visualización 3D.

1. Acceso al comando *cls* (*clear screen*) del visor 3D. Borra todo el contenido de la ventana de visor3D activa.

2. Acceso al comando *suprimir*. Elimina el objeto seleccionado en la ventana de visor3D activa.

3. Acceso al comando *colorfondo*. Establece el color de fondo de la ventana de visor3D activa.



4. Acceso al comando *asignarpaleta*. Asigna la paleta de trabajo actual al objeto seleccionado en la ventana de Visor3D activa.

5. Acceso al comando *referencia*. Habilita/Deshabilita los ejes de referencia de la ventana de Visor3D activa.

6. Acceso al comando *textura-on*. Texturiza el objeto seleccionado en la ventana de Visor3D activa con un fichero de imagen. (formatos soportados: .jpg, .jpeg, .rgb y .png)

7. Acceso al comando *textura-off*. Elimina la textura del objeto seleccionado en la ventana de Visor 3D activa.

8. Acceso al comando *reflejarv*. Refleja horizontalmente el objeto seleccionado de la ventana del Visor3D activa.

Figura 54. Barra1. Comandos para el visor3D.

9. Acceso al comando *reflejarh*. Refleja horizontalmente el objeto seleccionado de la ventana de Visor3D activa.

10. Acceso al comando *cargarnube*. Crea y carga un objeto de tipo nube de puntos que representan los datos de temperatura de la imagen termográfica desplegada en zona3.

11. Acceso al comando *cargarmalla*. Crea y carga un objeto de tipo malla (*simple grid*) que representa los datos de temperatura de la imagen termográfica desplegada en zona3.

12. Acceso al comando *cargarmallat*. Crea y carga un objeto de tipo malla (*triang. grid*) que representa los datos de temperatura de la imagen termográfica desplegada en zona3.

13. Acceso al comando *filtro*. Abre una ventana de configuración de las opciones de filtrado de temperaturas, pudiendo establecer la temperatura de corte y los colores para las temperaturas superiores e inferiores a la de corte, del objeto seleccionado en la ventana de Visor3D activa.

14. Acceso al comando *filtrobanda*. Abre una ventana de configuración de las opciones de filtrado pasa banda/rechaza banda de temperaturas, pudiendo establecer la temperatura de corte y los colores para las temperaturas superiores, inferiores y de banda, del objeto seleccionado en la ventana de Visor3D activa.

15. Acceso al comando *invertirc*. Realiza la negación lógica de cada uno de los colores de los puntos que conforman el objeto representativo de la imagen termográfica, en la ventana de Visor3D activa.

16. Acceso al comando *cortarxfila/cortarxcolumna*. Abre una ventana de configuración de las opciones de corte (fila/columna) del objeto seleccionado en la ventana de Visor3D activa.

17. Acceso al comando *cortarxarea/seleccionararea*. Da la posibilidad de cortar el objeto seleccionado en la ventana3D activa, mediante el uso del ratón sobre la imagen desplegada en zona3 (en este caso internamente se usa el comando *cortarxarea*) o mediante el uso del ratón sobre la el objeto seleccionado representado en el Visor3d (en este caso internamente se usa el comando *cortarxarea*).

18. Acceso al comando *translacion*. Abre una ventana donde se establecen la magnitud y la coordenada donde se aplicará la translación del objeto seleccionado en la ventana del Visor 3D activa.

19. Acceso al comando *escalar*. Abre una ventana donde se establece el eje y el factor de escala a aplicar, sobre el objeto seleccionado en la ventana de Visor3D activa.

20. Acceso al comando *rotación*. Abre una ventana donde se establece el eje y el ángulo de giro a aplicar, sobre el objeto seleccionado en la ventana de Visor3D activa.

21. Acceso al comando *centro*. Establece las coordenada, de manera absoluta (en el espacio del mundo 3D) que usará el objeto seleccionado como origen de sus coordenadas (espacio del objeto).

22. Acceso al comando *autoeje*. Crea, carga y asigna al objeto seleccionado en la ventana de Visor 3D activa, un eje de coordenadas que intenta ajustarse de la mejor manera posible a las dimensiones del objeto.

23. Acceso al comando *trackball-on*. Carga y asigna un manipulador de tipo *trackball* al objeto seleccionado de la ventana 3D activa.

24. Acceso al comando *box-on*. Carga y asigna un manipulador de tipo *box* al objeto seleccionado de la ventana 3D activa.

5.2.3 Barra2. Comandos para el visor 3D.

Al igual que la Barra1, esta barra solo será activada al cargar un archivo válido, establecer los valores de configuración de temperatura adecuados y establecer una ventana activa del visor3D.

Figura 55. Barra2. Comandos para el Visor3D.

1. Acceso al comando *manipuladores-off*.
   Elimina todos los manipuladores en los objetos seleccionados de la ventana de Visor3D activa.

2. Acceso al comando *luz-on*. Abre la ventana asignación de un foco de luz al objeto seleccionado en la ventana de Visor3D activa, pudiendo establecer las coordenadas (relativas al sistema de coordenadas del objeto, es decir fila, columna) donde se situará el foco de luz, así como la altura de dicho foco y el color de la luz generada por el mismo.

3. Acceso al comando *luz-off*. Elimina un foco de luz del objeto seleccionado en la ventana de Visor 3D activa.

4. Acceso al comando *histo*. Crea y carga en la ventana de Visor 3D activa un histograma del objeto seleccionado.
5. Acceso al comando *eje*. Crea y carga en la ventana de visor 3D un eje de coordenadas con las propiedades establecidas por el usuario mediante la ventana de configuración que se abre mediante el menú *Visualización→ eje de coordenadas*.

6. Acceso al comando *v2bin*. Permite guardar un archivo binario, en la ruta especificada por el usuario, que contiene la información de temperatura de cada uno de los puntos que componen el objeto seleccionado en la ventana de visor3D activa.

7. Acceso al comando *v2txt*. Crea un archivo de texto, en la ruta especificada por el usuario, que contiene una recopilación de información sobre el objeto seleccionado en la ventana de visor 3D activa.

8. Acceso al comando *h2txt*. Permite guardar un archivo de texto, en la ruta especificada por el usuario, que contiene información sobre las ocurrencias de cada uno de los valores de temperatura de la imagen y su porcentaje sobre el total.

9. Acceso al comando *script*. Abre un archivo de texto para ejecutar los comandos contenidos en él. Mediante el menú *Opciones→script...* se establece el valor de intervalo de tiempo, que define cada cuanto tiempo (en segundos) se ejecuta un comando.

10. Acceso al comando *isoterma*. Permite establecer filtros a los colores de la paleta para aislar o resaltar zonas específicas de la termografía.

11. Acceso al comando *info*. Añade, a la ventana del Visor3D activa, información general sobre el objeto seleccionado.

12. Acceso al comando *info-off*. Desactiva la información del objeto seleccionado, mostrada en la pantalla del Visor3D.

Capítulo 6. Juego de Pruebas del Programa

Para comprobar el funcionamiento de la aplicación desarrollada se han comparado los resultados obtenidos, con los de la distribución de evaluación del programa comercial mikroSpec 2.9 de mikronInfrared, Inc.
El visor3D se ha usado junto con el Visor2D para acelerar y facilitar la obtención de resultados.

Para mostrar el uso del programa se han realizado capturas del uso del mismo durante todo el proceso de pruebas.
Para la prueba se ha utilizado una imagen termográfica de ejemplo que se incluye en la instalación del software MikroSpec. La imagen se muestra en la figura 50.

Figura 56. Imagen termografica de prueba .

Se parte de la imagen de la termografia en formato TXI. Una vez cargada la aplicación Visor2D de termografias, se procede a abrir la imagen mediante el menú o con el botón correspondiente (figura 57).

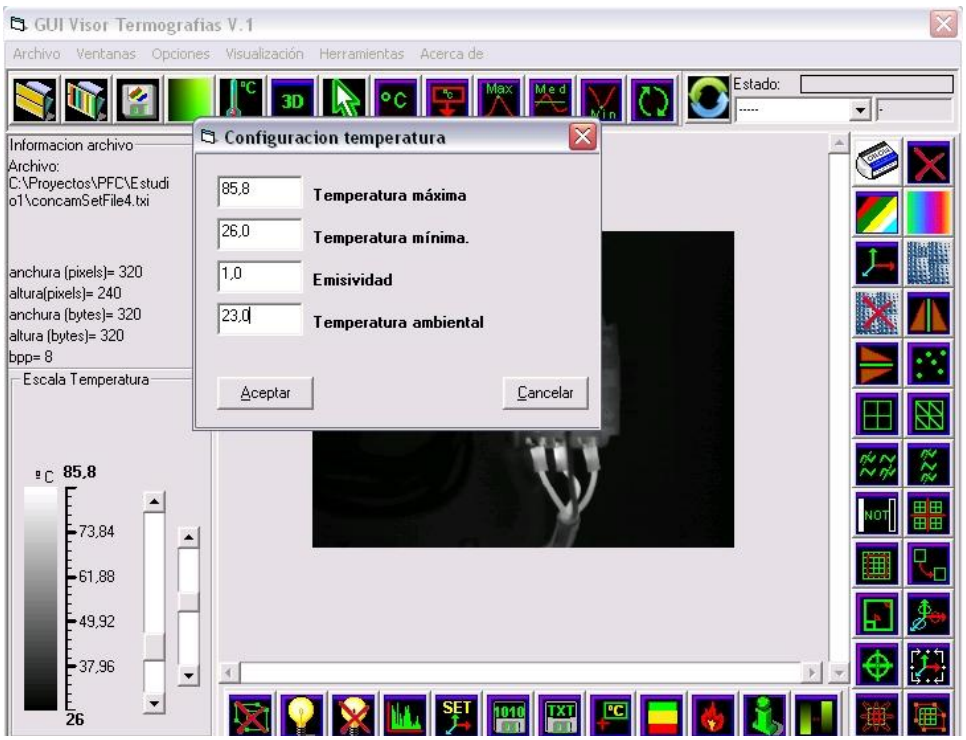


Figura 57. Apertura y establecimiento de los valores de temperatura.

Al tratarse de un formato que contiene la información de la temperatura máxima y mínima integrada, sólo es necesario modificar los valores de emisividad y de temperatura ambiental para seguir el proceso de análisis de la termografia. Para ello se utiliza el botón correspondiente de la barra principal de la aplicación o mediante el menú *Opciones* → *establecer datos de temperatura*.

Mediante los botones correspondientes de la barra de tareas se obtienen los datos de temperatura máxima, temperatura mínima y temperatura media, para compararlos con los obtenidos por la aplicación mikroSpec2.9.

Figura 58. Apertura del mismo archivo en la aplicación comercial mikroSpec 2.9.

La siguiente tabla muestra los resultados obtenidos con cada una de las dos aplicaciones:

	Temp. Máx.		**Temp.mínima**		**Temp. media (ºC)**
	Valor (ºC)	ocurrencias	Valor (ºC)	ocurrencias	
Visor2D&Visor3D	85,8	1	26,0	10	30,658
MikroSpec2.9	85,8	1	26,0	4	30,7

Los resultados obtenidos son muy similares, las diferencias se pueden atribuir al nivel de precisión utilizado para el cálculo de la temperatura y para mostrar los resultados. Si consideramos solo la representación con una cifra decimal el resultado se puede considerar igual.

Seguidamente se obtienen las representaciones en 3D de la termografia.
Gracias al uso de la librería gráfica Coin3D la representación 3D posee mayor resolución que la generada por la aplicación mikroSpec.

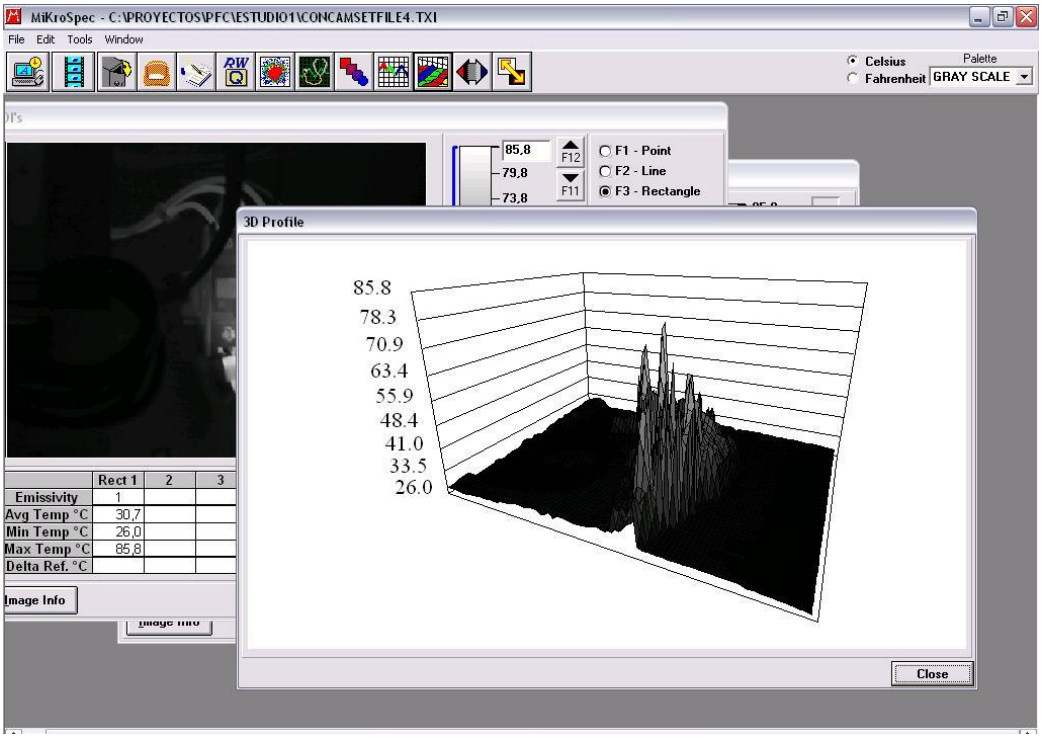


Figura 59. Representación 3D de la termografía en la aplicación comercial mikroSpec2.9

Además de la resolución, existen más diferencias entre las imágenes generadas por el Visor3D y la aplicación mikroSpec. En ésta última las imágenes generadas solo pueden utilizar degradados de colores (128 máximo), no siendo posible el uso de la paleta de color aplicada a la imagen en la representación 3D.

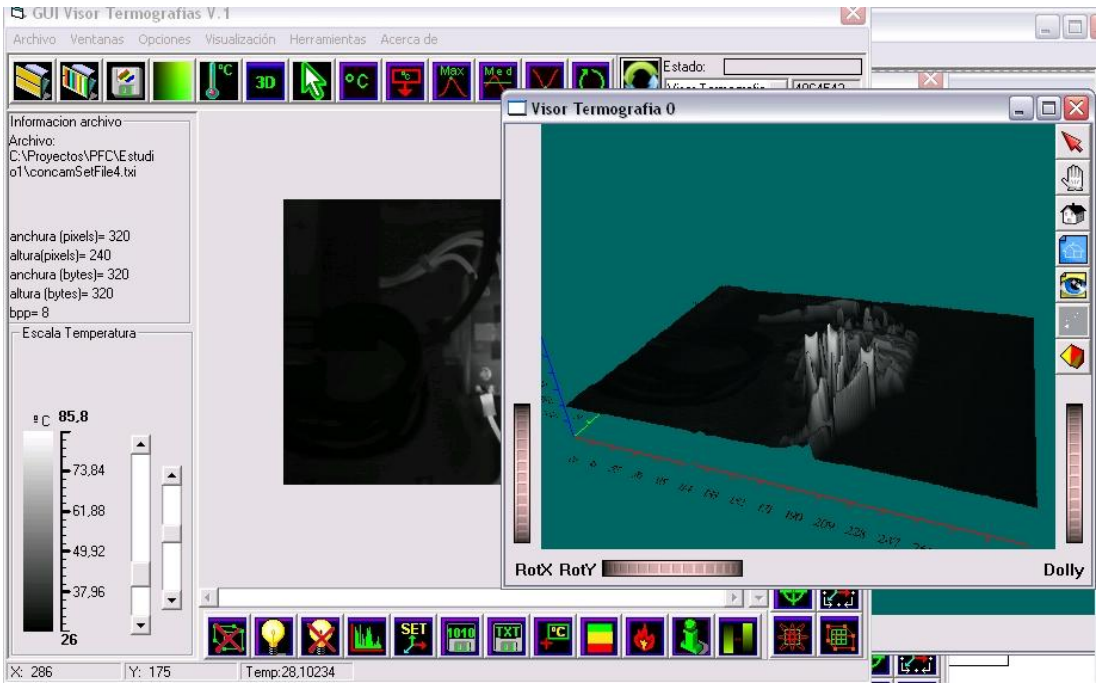


Figura 60. Representación 3D de la termografía con el uso de la aplicación Visor2D&Visor3D desarrollada.

Para realizar la comparación en las condiciones más similares posibles se ha obtenido la representación 3D en la aplicación Visor3D usando un degradado de color simple basado en color negro (temperatura mínima) a blanco (temperatura máxima). Para ello

se procede abrir el diálogo correspondiente de generación de degradados (menú *opciones→ asignarpaleta→generar degradados*) y una vez generado se aplica la figura. Seguidamente se abre una ventana del Visor 3D mediante el botón correspondiente y se carga el objeto. En la imagen de la figura 60 se ha usado la representación mediante malla de tipo *simple grid*.

Siguiendo con el análisis de la imagen, se selecciona como zona de interés la definida por 150, 135 - 240, 226 ya que se aprecia un incremento de temperatura respecto al resto de la imagen. Antes se realiza la aplicación de una paleta de colores (*rainbow*) para mejorar el contraste entre las diferentes zonas.

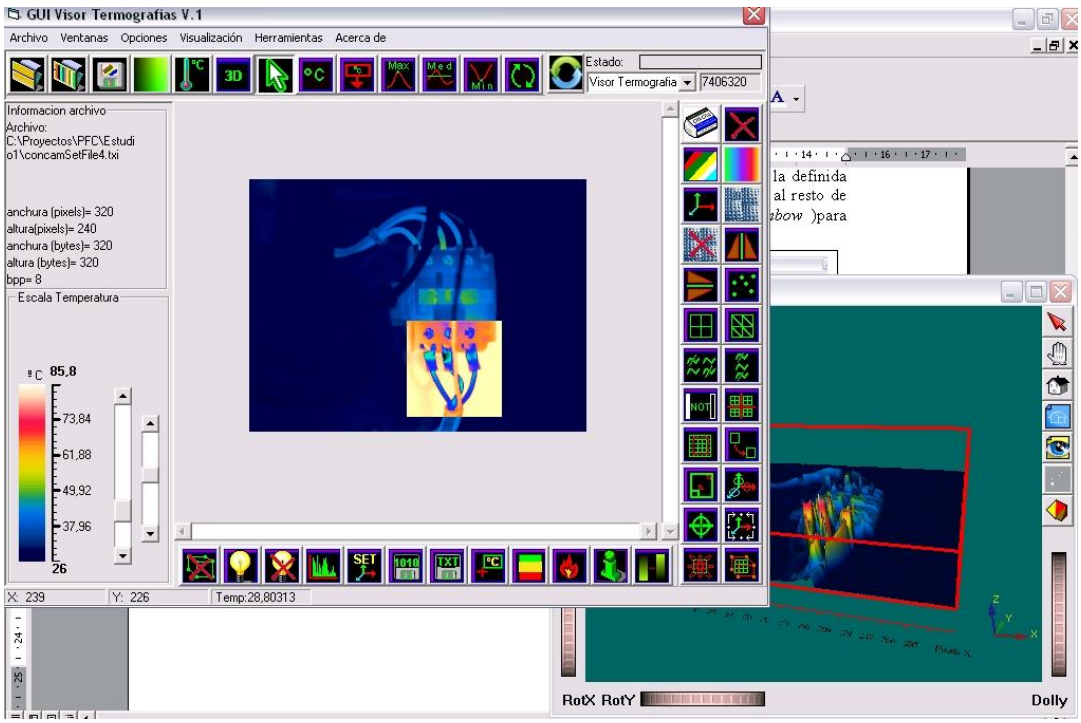


Fig.61 selección del área de interés usando la aplicación Visor2D&Visor3D desarrollada.

Al resultado de la selección se le aplica el comando *autoeje*, y se procede a obtener el histograma mediante el botón correspondiente de la barra 2.
Tras esto, se obtienen los resultados de las imágenes mostradas en la tabla de la figura 62.

	Temp. Máxima (ºC)	**Temp.Mínima (ºC)**	**Temp.media (ºC)**
Visor2D&Visor3D	85,8	27,64	38,34
MikroSpec	85,8	27,6	38,5

Figura 62. Valores obtenidos para la zona seleccionada.

En el histograma se observa que el número máximo de ocurrencias se concentra en la zona de índices inferiores al 25 (31,8 ºC) aproximadamente.

El programa desarrollado ofrece la posibilidad de mostrar el histograma y la representación 3D del área a estudiar, característica que no posee el software comercial, más enfocado al tratamiento de los datos y a la representación 2D.

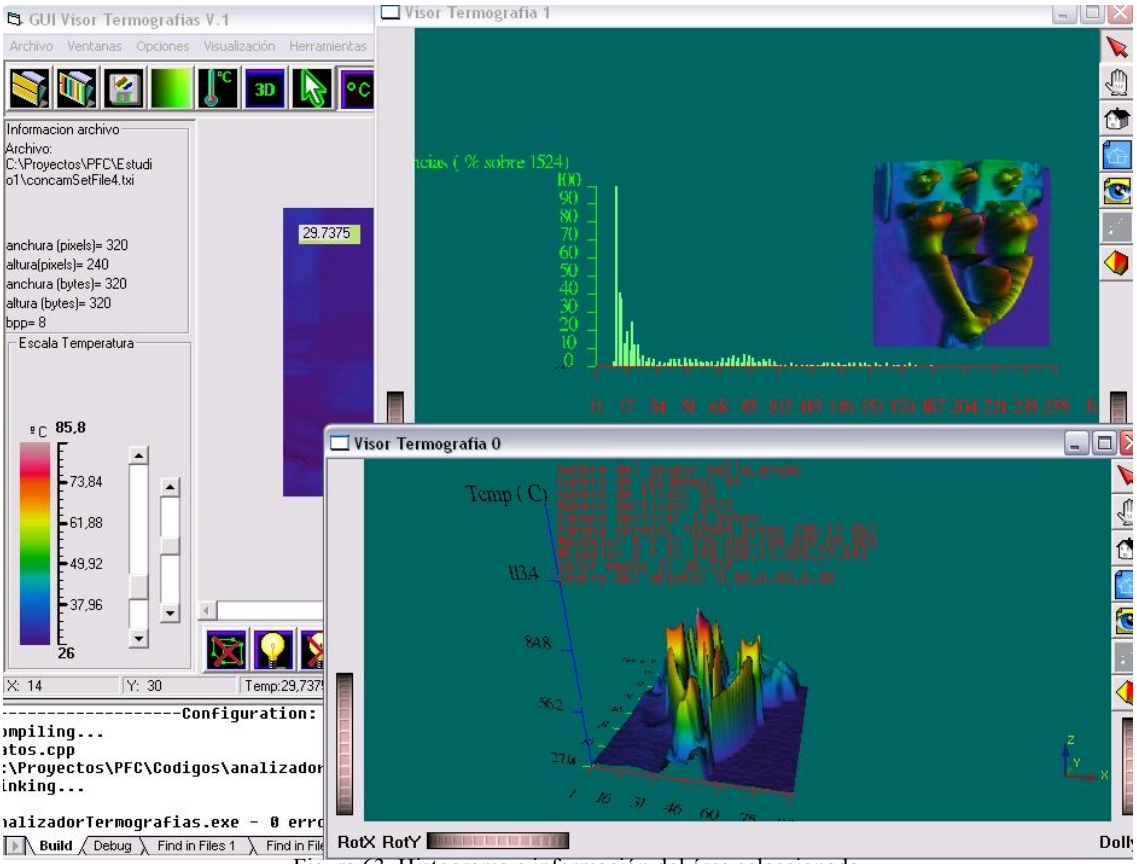


Figura 63. Histograma e información del área seleccionada.

A continuación se realiza un filtro de paso alto sobre la sección de área estableciéndose la temperatura de corte en 72.5 ºC, dejando los colores para cada banda de temperaturas sin modificaciones. La imagen de la figura 64 (imagen inferior) muestra el resultado.

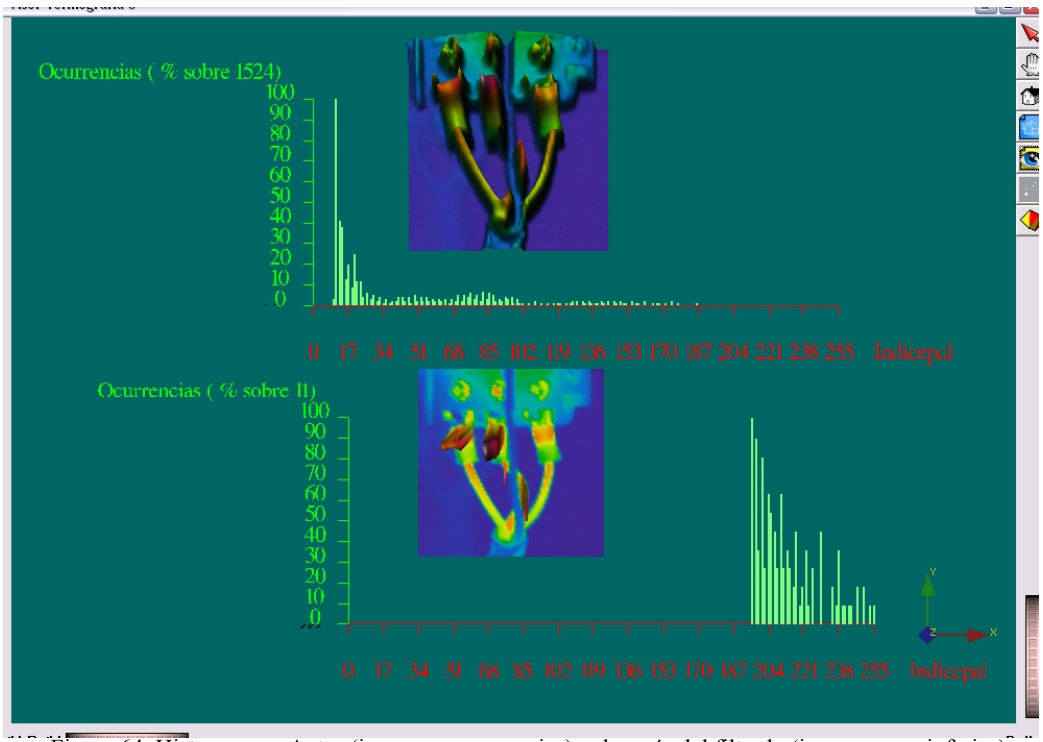


Figura 64. Histogramas. Antes (imagen zona superior) y después del filtrado (imagen zona inferior).

Se aprecia como la función del filtro actúa a modo de "amplificador" de las componentes de índice más elevado (temperaturas más elevadas).
En este caso, los resultados se obtienen en porcentaje respecto al número de ocurrencias máximo, que para este caso es de 11, siendo de 1524 en el caso de la figura sin filtrar.

Para finalizar el proceso se realiza un volcado de la información, tanto para los datos de temperatura que componen la imagen, como del histograma. Los datos obtenidos en un fichero de texto pueden usarse, posteriormente, para realizar análisis con otro tipo de programas.
Por si es de interés, el contenido de ambos ficheros se encuentran en el anexo2.

Capítulo 7. Conclusiones

Mediante el uso de la librería gráfica Coin3D se ha podido representar de manera eficaz las imágenes termográficas.

Los resultados obtenidos han resultado correctos para las pruebas realizadas, considerando los datos de entrada que utiliza la aplicación.

Muchas veces las condiciones para la obtención de resultados no son las idóneas. En estos casos, es necesario obtener información partiendo de muy pocos datos, lo que puede provocar que los valores obtenidos no se aproximen lo suficiente a los reales. El ejemplo más significativo de una situación como la comentada, es el caso de imágenes termográficas en formatos bitmap (bmp, etc.), donde solo es posible extraer información de los pixeles que la componen. Esta limitación (resolución de 8bpp) se acentúa si no se ha hecho un tratamiento correcto de la imagen. Como ejemplo de tratamiento incorrecto se puede citar a las reducciones de *bpp*.

A los factores de pérdida de datos comentados, habría que sumarle el del uso de ciertos formatos gráficos. Existen formatos gráficos que aplican algoritmos de compresión que causan pérdida adicional de datos.

En este sentido, algunas aplicaciones comerciales de análisis de termografías y representación de datos, trabajan, ante estos casos, con índices de la paleta de colores, pudiendo resultar para el usuario una tarea pesada y propensa a errores la correcta interpretación de dichos índices. Aquí es donde la aplicación desarrollada en el presente proyecto puede resultar útil, ya que ofrece la posibilidad de trabajar siempre con datos de temperatura sólo con el requisito de introducir unos pocos datos adicionales como son temperatura máxima y mínima, teniendo en cuenta las lógicas limitaciones de los formatos de imagen usados.

La representación en la mayoría de las veces posee mayor resolución que la generada por algunos de los programas comerciales, que muchas veces usan representaciones mediante nubes de puntos y sin dar la posibilidad al usuario de interactuar directamente con ella, limitándose en muchos casos a rotaciones y *zooms* no dinámicos y a usar representaciones de color para cada zona de temperatura con posibilidades muy reducidas.

La eficiencia en las representaciones tridimensionales es la suficiente para que se pueda interactuar de una manera fluida con la malla. Considerando que las imágenes termográficas suelen tener resoluciones estándar entre 160x120 y 640x480, el programa puede ser usado en equipos con recursos bastante ajustados (probado en un AMD XP 1.7 @ 1.1 Ghz con 192 Mb y gráfica AGP S3 Trio3D 8Mb).

Además la posibilidad de poder guardar los datos de temperatura, tanto en formato binario como en texto, resultan útiles para que el usuario pueda utilizarlos en otro tipo de aplicaciones de tratamientos de datos. Se puede usar el programa del presente proyecto como una aplicación para la conversión de los datos de los pixeles en datos de temperatura.

En cuanto a los problemas encontrados durante el desarrollo de la aplicación estos han sido variados, pero el más importante ha sido el que hace referencia a la gestión de la entrada/salida. En este caso la captura de caracteres de teclado se ha complicado bastante.

La librería Coin3D no permite un acceso directo a la ventana del intérprete de comandos si no que, como es propio en este tipo de librerías, permanece en un estado de bucle continuo a la espera de cualquier cambio en los datos de la *scenegraph* (campos de los nodos, acciones...), que le hagan recorrer de nuevo el árbol de nodos para generar la imagen, por lo que la captura y detección de los comandos han sido los aspectos que más dificultades han causado.

Este hecho ha resultado en importantes limitaciones del software en cuanto a reconocimiento de caracteres se refiere. Aunque el juego de caracteres soportado ha resultado suficiente para los objetivos planteados, ha imposibilitado el realizar ciertas acciones como, por ejemplo, el paso de parámetros que contengan espacios.

Los puntos de una posible futura mejora del programa son muchos y variados. Los que se pueden considerar más importantes son:

- Tratamiento de errores:

  El programa funciona correctamente en el uso normal, entendiendo como "normal" la introducción de formatos de archivos correctos, el uso de argumentos dentro de los límites y con los formatos especificados en el capítulo 5, etc. Si no se siguen las normas anteriores es posible que, en ciertas ocasiones, el funcionamiento no sea el esperado.

  En este sentido cabe decir que, considerando la limitación temporal para el desarrollo del presente proyecto, el esfuerzo en el desarrollo del programa se ha centrado en el de proporcionar facilidad de uso (factor que evita sobremanera errores o problemas), funcionalidad y diversidad de herramientas al usuario, dejando en segundo lugar el tratamiento de errores no críticos y la información de los mismos al usuario.

- Optimización del código.

  Aspectos como redundancia en el código y una estructuración alternativa pueden suponer una mejora importante.

- Comunicación entre la interfaz de usuario y el programa principal de representación 3D.

  La comunicación entre las dos aplicaciones desarrolladas (Visor2D y Visor3D) es prácticamente inexistente, no existe ningún mecanismo que permita el *feedback* entre el visor3D y el Visor2D.
  En el desarrollo de este proyecto la aplicación Visor2D funciona como una simple interfaz. Se limita al envío de la correspondiente secuencia de comandos a ejecutar por el Visor3D.

Por tanto se podría plantear realizar la aplicación del Visor3D y el Visor2D como una única aplicación, desarrollada íntegramente en C++. Se podría evitar el crear una interfaz de usuario desde cero, aprovechando las capacidades de la librería para añadir botones adicionales a la ventana del visor ofrecida por el nodo *SoWinExaminerViewer*. También se podría plantear la realización de la interfaz gráfica mediante el uso de las MFC de Microsoft, o el entorno de desarrollo de Borland o, mejor aún, bajo sistemas LINUX mediante el uso del *Binding SoXt* o *SoQt*. En este último caso, una gran parte del código seria portable con mínimas modificaciones (ficheros de encabezado y los nombres de algunas clases).

Seguramente con la adopción de cualquiera de estas alternativas el problema de comunicación se hubiera reducido mucho, o eliminado completamente, y los esfuerzos en el desarrollo se podrían haber centrado en otros aspectos, como por ejemplo, el ya comentado punto de mejora en la gestión de errores.

- Gestión de la entrada/salida.

  Como se comentó antes, una mejora en la entrada/salida que no impusiera las limitaciones que posee el programa actual facilitaría e incluso podría reducir considerablemente ciertas partes del código. Un mejor uso del *binding* SoWin facilitaría esta tarea.

- Un mayor aprovechamiento de las capacidades ofrecidas por la librería FreeImage en conjunción con Coin3D. Por ejemplo, se podrían mejorar o añadir las siguientes capacidades:
  Posibilidades de tratamiento de imágenes en 16/32bpp, transformaciones en escala de grises, saturación/extracción de canales RGB para aplicarlos en funciones del tipo filtro o isotermas, etc.

- Aprovechar las facilidades ofrecidas por Coin3D para guardar/cargar archivos de escenas gráficas, posibilitando al usuario guardar una escena para su posterior restauración, sin necesidad de realizar los mismos pasos otra vez.

En cuanto a la experiencia personal son diversos los aspectos en los que he podido introducirme.
Para el desarrollo de este proyecto me ha sido necesario aprender los lenguajes VBasic y C++, lo cual ha consumido gran parte del tiempo invertido. En especial C++, por el cambio de mentalidad que supone asimilar y programar bajo los conceptos de la POO (clases, herencia, encapsulamiento, sobrecarga de funciones y operadores, etc).

La realización del proyecto me ha servido para adquirir experiencia en el uso de la librería Coin3D, que ha supuesto mi primer contacto con una librería gráfica de alto nivel y su uso para la representación de datos.

Además, considero que la experiencia adquirida no se ciñe únicamente al uso de Coin3D, sino que puede resultarme útil para el uso de otras librerías gráficas de alto nivel. En especial, puede resultar útil para aquellas que basen su estructura de datos en los árboles de nodos y los conceptos de *scene database*, *scenegraph* y de *traversal state*.

Por ejemplo y sin ir más lejos, Coin3D se considera un clon de la librería OpenInventor, así que parte de los conocimientos adquiridos podrían ser aplicados a esta última.

Bibliografía

Libros, Manuales y Artículos Consultados.

"Open Inventor C++ Reference Manual, Release 2"
Addison Wesley (1994)

"FreeImage: a Free OpenSource Grpahics Library"
Documentation. Library version 3.11.00

Josie Wernecke, Open Inventor Architecture Group.
"The Inventor Mentor : Programming Object Oriented 3D Graphics with OpenInventor, Release2"
Addison -Wesley Professional (1994)

Michael F.Modest, The Pennsylvania State University
"Radiative Heat Transfer", 2nd. edition
Academic Press (2003)

Donald A.Burns, Emil W. Ciurczak
"Handbook of Near-Infrared Analysis"
Marcel Dekker, Inc. (2001)

Constantine Pozrikidis.
"Introduction to C++ Programming and Graphics"
Springer+Business Media,LLC (2007)

"Guia Básica a la Termografia"
Lands Instruments International (2004)

"MiKroSpec TM 2.9 Thermal Imaging Software Manual"
Mikron Infrared, Inc.

"ATS Introduction. Thermal image analysis software"
Nippon Avionics Co.Ltd

J. K. Berry.
"Visualización de datos en estructuras de Malla."

Bergman, Rogowitz, Treinish.
"A Rule-based Tool for Assisting Colormap Selection"

Rogowitz, Treinish.
"Why Should Engineers and Scientists be Worried about Color"

Bruce Eckel.
"Pensar en C++ ", 2nd.ed volume1 (2000)

B.W Kernighan, D.M Ritchie
"El lenguaje de Programación C". Pearson Educación, segunda edición (1991)

Programación en C/C++, Visual C++
Version 1.0.0 Grupo EIDOS (2000)

J.M Mejía Vilet ."Apuntes de procesamiento digital de imágenes"
Área de computación e Informática. Facultad de Ingenieria UASLP

Programación en VBasic.
Universidad de Navarra. 1999

Páginas web consultadas:

Coin3D main page
http://www.coin3d.org

VTK Home page.
http://www.vtk.org

Matpack C++ Numerics and Graphics Library - Home Page
http://www.matpack.de

Open Inventor
http://oss.sgi.com

OpenSceneGraph
http://www.openscenegraph.org

IrFileConverter User's Manual. NEC Avio Infrared Technologies Co.,Ltd
http://www.nec-avio.co.jp/en/index.html

FILExt – The File Extension Source
http://www.filext.com

Wikipedia
http://es.wikipedia.org

"Características cámaras termográficas"
http://www.fluke.com

"Cámaras Termográficas"
http://www.amperis.com

http://msdn.microsoft.com/es-es/vbrun/default.aspx

"Articulo de revista nº 192"
http://www.infovis.net

Software de pos-procesamiento de imágenes termográficas
http://irimaging.com

Anexo 1. Formato de los ficheros temporales.

Los siguientes archivos son los generados por la función *analizar_imagen()* al aplicarse a la imagen utilizada en el de juego de pruebas.

Contenido del fichero *datoscab.txt*:

```
0
8
320
240
320
320
"C:\Proyectos\PFC\Estudio1\concamSetFile4.txi"
```

Contenido del fichero *datosimag.txt*:

Debido a su extensión y su simplicidad solo se incluye un extracto del mismo. El resto del archivo contiene el resto de índices a la paleta de colores de cada uno de los pixeles que conforman la imagen termográfica.

```
8 9 11 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 9 9 8 7 7 7 8 9 10 12
11 11 11 11 11 11 11 12 12 13 13 13 14 15 15 13 14 14 14 14 14 14 14 14 15 15
15 14 14 13 12 11 13 14 15 16 17 16 16 16 17 17 16 16 16 17 18 17 17 17 16 17
18 17 17 16 17 17 18 18 18 18 19 18 18 18 18 18 18 18 18 18 19 20 20 19 20 20
18 17 17 17 15 14 15 16 19 22 24 26 24 22 22 15 14 14 16 19 21 21 20 20 20 20
19 19 19 19 19 19 19 18 17 18 18 18 19 19 19 19 19 19 20 20 20 20 21 22 22 21
22 23 23 22 22 23 22 22 21 22 22 22 23 27 33 39 48 56 65 69 72 75 51 37 43 45
47 47 46 45 44 43 60 73 71 62 42 20 14 13 13 13 13 12 12 12 12 13 13 13 13 13
11 11 12 12 12 12 12 12 12 13 13 13 12 12 12 12 12 12 12 13 11 11 11 13 12 12
```

Contenido del fichero *datospal.txt*:

El contenido de este fichero almacena la paleta que utiliza la representación de la termografia tanto en el Visor2D (color del pixel) como en el visor3D (color del vértice). En este caso no es mas que una progresión de valores correspondientes a una paleta progresiva de escala de grises de 0 0 0 a 255 255 255 (componentes RGB).

```
0 0 0
1 1 1
2 2 2
3 3 3
4 4 4
5 5 5
6 6 6
```

Anexo 2. Archivos de datos de salida.

Archivo de datos de temperatura.

Debido a la longitud del fichero solo se incluye un extracto del mismo (la cabecera y tres filas de datos de temperatura), suficiente para mostrar el formato usado para éste tipo de ficheros.

```
Archivo Origen de datos: "C:\Proyectos\PFC\Estudio1\concamSetFile4.txi"
Tipo de objeto: malla_grupo
N° filas del objeto: 92
N° columnas del objeto: 91
Máx X: 239.000
    Y: 104.000
Temperatura máxima: 85.800(°C)
Min X: 149.000
    Y: 13.000
Temperatura mínima: 27.642(°C)
Temperatura media: 38.337(°C)

29.752    29.752    29.518    29.987    29.987    29.987    29.987    29.987
29.987    29.987    29.752    29.987    29.987    30.221    30.456    30.925
30.690    30.925    30.925    30.925    30.925    31.159    30.925    30.690
30.456    30.456    29.987    29.752    29.518    29.049    28.814    29.049
29.283    30.690    37.256    40.305    37.491    37.022    36.553    36.318
36.084    36.084    36.084    36.084    36.084    35.849    40.071    43.119
43.354    43.354    42.181    37.491    29.987    29.049    28.814    28.814
28.814    28.814    28.814    28.814    28.814    28.814    28.814    28.814
28.814    28.814    28.814    28.814    28.814    28.580    29.049    28.814
29.049    29.049    29.049    28.814    28.814    28.814    28.814    28.814
29.049    28.814    28.814    28.814    28.345    28.580    28.580    28.580
28.580    28.580    28.580
29.752    29.752    29.752    29.987    29.987    30.221    29.987    29.987
29.987    29.987    29.987    29.987    29.987    30.221    30.456    30.690
30.925    31.159    31.159    31.159    31.159    31.159    30.925    30.690
30.456    30.221    29.987    29.752    29.752    29.518    29.049    29.049
29.049    29.987    34.677    39.602    38.664    37.960    37.491    36.787
36.553    36.553    36.318    36.318    36.084    36.084    38.664    42.416
43.119    44.057    44.526    42.181    33.035    29.049    28.814    28.814
29.049    29.049    28.814    28.814    28.814    28.814    28.814    29.049
29.049    28.814    28.814    29.049    29.049    28.814    28.814    28.814
28.814    29.049    28.580    28.814    28.814    28.814    28.814    28.814
28.814    28.580    28.580    28.580    28.580    28.580    28.580    28.580
28.814    28.814    28.814
29.518    29.518    29.518    29.752    29.752    29.987    29.987    29.987
30.221    29.752    29.752    29.987    29.987    29.987    30.221    30.456
30.925    30.925    30.925    30.925    30.925    30.925    30.690    30.456
30.456    30.221    29.987    29.518    29.283    29.283    29.049    29.049
29.049    29.283    32.566    38.898    39.367    38.664    37.725    37.256
36.553    36.553    36.553    36.318    36.318    36.553    36.787    39.836
41.478    43.119    44.526    44.057    36.553    29.049    28.814    28.580
28.814    28.814    28.580    28.580    28.814    28.814    28.814    28.814
28.814    28.814    28.814    28.814    28.580    28.580    28.580    28.580
28.580    28.580    28.580    28.580    28.580    28.580    28.580    28.580
28.580    28.580    28.580    28.580    28.580    28.580    28.814    29.049
28.345    28.580    28.580
```

Archivo de datos del histograma.

Debido a la extensión del archivo solo se incluyen las primeras líneas del archivo de salida de datos.

```
Archivo Origen de datos: "C:\Proyectos\PFC\Estudio1\concamSetFile4.txi"
N° filas del objeto: 92
N° columnas del objeto: 91

Numero de elementos: 8372
Temperatura Maxima (°C): 85.800
Temperatura Mínima: 26.000
Número máximo de ocurrencias: 1040
```

indice pal	Temp(°C)	Ocurrencias	%total
	27.642		0.000
1	27.870		0.000
2	28.098		0.000
3	28.326		0.000
4	28.554		0.000
5	28.782		0.000
6	29.010		0.000
7	29.238	2	0.024
8	29.466	4	0.048
9	29.694	5	0.060
10	29.922	59	0.705
11	30.150	484	5.781
12	30.378	1040	12.422
13	30.607	629	7.513
14	30.835	297	3.548
15	31.063	283	3.380
16	31.291	200	2.389
17	31.519	171	2.043
18	31.747	145	1.732
19	31.975	142	1.696
20	32.203	172	2.054
21	32.431	221	2.640
22	32.659	183	2.186
23	32.887	88	1.051
24	33.115	97	1.159
25	33.343	73	0.872
26	33.571	54	0.645
27	33.800	46	0.549
28	34.028	51	0.609
29	34.256	36	0.430
30	34.484	45	0.538
31	34.712	41	0.490
32	34.940	45	0.538
33	35.168	23	0.275
34	35.396	26	0.311
35	35.624	22	0.263
36	35.852	29	0.346
37	36.080	30	0.358
38	36.308	19	0.227
39	36.536	20	0.239
40	36.764	33	0.394
41	36.993	27	0.323
42	37.221	34	0.406
43	37.449	34	0.406
44	37.677	32	0.382
45	37.905	37	0.442
46	38.133	39	0.466
47	38.361	29	0.346
48	38.589	22	0.263
49	38.817	40	0.478

Anexo 3. Formato de los archivos *script.*

Los ficheros están compuestos simplemente de llamadas a los comandos de la aplicación. Cada comando con sus respectivos argumentos se escribe en una nueva línea y el carácter ';' es el encargado de indicar el final del comando.
A modo de ejemplo se muestra el siguiente fichero script.

```
cls;
analizarimagen concamSetFile4.txi;
cargarmalla 1 0 0 0 90.5 23.8 malla1;
selobjeto malla1;
cortarxarea 10 10 200 200;
selobjeto malla1;
autoeje;
selobjeto malla1;
h2txt histoArea.txt;
analizarimagen concamSetFile5.txi;
cargarnube 1 0 0 0 78.4 45.6 malla2;
selobjeto malla2;
promedio;
selobjeto malla1;
posicionar 350 0 0;
selobjeto malla2;
cortarxfila 150 1;
cls;
```

hay que resaltar que todos y cada uno de los comandos que componen el fichero se ejecutan transcurridos los segundos indicados por el usuario al llamar al comando *script*. En éste ejemplo desde que se corta el área indicada hasta que se genera el eje de coordenadas habrán transcurrido 2*intervalo segundos, ya que *selobjeto* también consume el mismo intervalo. Es necesario tenerlo en cuenta ya que el comando *selobjeto* no suele mostrar ningún cambio perceptible en la ventana del Visor3D, de manera que el usuario puede pensar que los comandos ejecutados después de una llamada a *selobjeto* se ejecutan más tarde de lo esperado.

Anexo 4. Listado de comandos de la aplicación Visor3D.

Presupuesto.

El presupuesto se ha realizado considerando un periodo de 4 meses de trabajo a 40 h/semanales.

Descripción	Cantidad	Precio unidad(€)	Total (€)
Equipo PC	1	700	700
Visual Studio6	1	260	260
Desarrollo de la aplicación	640 h	35	22400
		Total	**23360**

Veintitrés mil trescientos sesenta euros.

Código Fuente de la Aplicación Visor3D

A continuación se incluyen los archivos que conforman la aplicación desarrollada en el proyecto. En total se divide en 8 archivos. Cada uno incluye el número de línea y de página de forma independiente del resto de archivos.

```
 1 #include "stdafx.h"
 2 #include <iostream>
 3 #include "stdio.h"
 4 #include "stdlib.h"
 5 #include "string.h"
 6 #include <Inventor/Win/SoWin.h>
 7 #include <Inventor/Win/viewers/SoWinExaminerViewer.h>
 8 #include <Inventor/nodes/SoMaterial.h>
 9 #include <Inventor/details/SoFaceDetail.h>
10 #include <Inventor/nodes/SoSeparator.h>
11 #include <Inventor/nodes/SoEnvironment.h>
12 #include <Inventor/nodes/SoTransform.h>
13 #include <Inventor/nodes/SoDrawStyle.h>
14 #include <Inventor/sensors/SoTimerSensor.h>
15 #include <Inventor/nodes/SoCoordinate3.h>
16 #include <Inventor/SbColor.h>
17 #include <Inventor/SbVec3f.h>
18 #include <Inventor/events/SoKeyboardEvent.h>
19 #include <Inventor/SoPickedPoint.h>
20 #include <Inventor/nodes/SoPointSet.h>
21 #include <Inventor/actions/SoRayPickAction.h>
22 #include <Inventor/actions/SoSearchAction.h>
23 #include <Inventor/events/SoMouseButtonEvent.h>
24 #include <Inventor/actions/SoCallbackAction.h>
25 #include <Inventor/nodes/SoEventCallback.h>
26 #include <Inventor/nodes/SoRotationXYZ.h>
27 #include <Inventor/nodes/SoSelection.h>
28 #include <Inventor/lists/SoNodeList.h>
29 #include <Inventor/SoPickedPoint.h>
30 #include <Inventor/actions/SoBoxHighlightRenderAction.h>
31
32 #include "gestionteclas.h"  //gestión de teclas
33 #include "figuras.h"       //creación objetos
34 #include "leerfichero.h"   //lectura de cabeceras&punteros a datos
35 #include "colores.h"
36 #include "nodos.h"
37 #include "gestioncomandos.h"  //manejo general de comandos
38
39 using namespace std;
40
41 SoWinExaminerViewer *eviewer=NULL;
42 unsigned int contador=0;
43 SoSelection * seleccion=NULL;
44 SoTimerSensor *Timer=NULL;
45 buffferteclas nbuffer;         //buffer de gestión de teclas
46 unsigned char capturaPuntos=0;  //flag para captura de puntos
47
48 #define  PI 3.141592653
49
50 //funciones de gestión de eventos
51 void eventoteclado(void *userData, SoEventCallback *mievento);
52 void eventoraton(void *userData, SoEventCallback *mievento);
53
54 static SoSeparator *pickedPointsSubGraph(void);
55 //refresco, explícito
56 void seleccion_objeto(void * userdata, SoPath * path )
57 {
58   seleccion->touch();
59 }
60
61 void deseleccion_objeto(void * userdata, SoPath * path )
```

```
 62 {
 63   seleccion->touch();
 64 }
 65
 66 /*************************************************************
    ********/
 67 /*    MAIN
    */
 68 /*************************************************************
    ********/
 69 int main(int argc,char **argv)
 70 {
 71
 72   HWND mainwin = SoWin::init("Visor termografia");
 73
 74   eviewer=new SoWinExaminerViewer(mainwin) ;
 75   seleccion=new SoSelection;
 76   //nodo "grupal", raiz
 77   SoSeparator *raiz=new SoSeparator;
 78
 79   //propiedades por defecto de color
 80   SbColor colorFondo;
 81   colorFondo.setValue( 0,0.5,0.5);
 82   eviewer->setBackgroundColor( colorFondo );
 83   SoEnvironment *entorno=new SoEnvironment;
 84   entorno->ambientColor.setValue(0,0.5,0.5);
 85
 86   raiz->addChild(entorno);
 87
 88   raiz->ref();  //referenciación necesaria
 89   raiz->setName("NodoRaiz");
 90   SoMaterial *material1=new SoMaterial;
 91   material1->setName("MaterialRaiz");
 92   SoTransform *transforma=new SoTransform;
 93   transforma->setName("TransformaRaiz");
 94   material1->diffuseColor.setValue(0.5,1.0,0.5); //establecemos el
    color de la superficie del objeto1
 95
 96   seleccion->policy=SoSelection::SINGLE;  //solo puede haber un
    objeto seleccionado
 97   seleccion->ref();
 98   seleccion->addChild(raiz);
 99
100   //callbakcs para (de)selección
101   seleccion->addSelectionCallback(seleccion_objeto, (void *)1L);
102   seleccion->addDeselectionCallback(deseleccion_objeto, (void
    *)0L);
103
104   raiz->addChild(material1);
105   raiz->addChild(transforma);
106
107 SoEventCallback *mievento = new SoEventCallback;
108 raiz->addChild(pickedPointsSubGraph());
109 //callbacks para eventos
110 mievento->addEventCallback(SoKeyboardEvent::getClassTypeId(),evento
    teclado,raiz);
111 mievento->addEventCallback(SoMouseButtonEvent::getClassTypeId(),eve
    ntoraton,raiz);
112
113 raiz->addChild(mievento);
114
```

```
115 Timer=new SoTimerSensor(callbackTimer, raiz );
116
117 //presentamos el identificador de la ventana para posible uso en la
    integracion con VBasic
118 printf("HWND: %ld\n\n",mainwin);
119
120   eviewer->setSceneGraph(seleccion);  //añadimos a la escena el
    nodo nprincipal
121   eviewer->setGLRenderAction (new SoBoxHighlightRenderAction );
122   eviewer->setFeedbackVisibility(TRUE);
123   eviewer->setViewing(FALSE); //opcion de rotacion con cursor del
    raton desactivada(por defecto el boton pulsado es el cursor)
124   eviewer->show();
125
126   SoWin::show(mainwin);
127   // Bucle principal.
128   SoWin::mainLoop();
129
130   delete eviewer;
131   seleccion->unref();
132   return 0;
133 }
134
135 void eventoteclado(void *userData, SoEventCallback *mievento)
136 {
137
138 int contador=0,flag_alt=0,flag_shift=0;
139 char caracter;
140
141   //cast para acceso al nodo raiz
142   SoSeparator *figura=(SoSeparator *) userData;
143   SoKeyboardEvent *evento = (SoKeyboardEvent
    *)mievento->getEvent();
144
145   //evaluación de las pulsaciones de teclas y de la captura de
    datos en tiempo de ejecución
146   if ( (evento->isKeyPressEvent(evento,SoKeyboardEvent::ANY)==TRUE)
    && (capturaPuntos==0) )//si se trata de pulsación de intro y no se
    estan capturando puntos
147   {
148     //gestion de caracteres especiales SHIFT+caracter
149     if ((evento->wasShiftDown()) == 1) //true
150       {
151       flag_shift=1;
152       caracter=evento->getKey(); //capturamos caracter
153       //llamamos a la funcion de conversion
154       caracter=conv_shift(caracter);
155       if (caracter!=-1)  //en caso de conversión
156         {
157         evento->setPrintableCharacter(caracter);
158         nbuffer.add_caracter(caracter); //gestionamos pulsacion de
    la tecla+ALT pulsado
159         flag_shift=0; //reset flag
160         }
161       }
162     else //caracter
163       {
164       flag_shift=0;
165       caracter=evento->getKey(); //lo capturamos
166       nbuffer.add_caracter(caracter); //gestinamos
167       }
```

```
168     if (evento->isKeyPressEvent(evento,SoKeyboardEvent::RETURN)==TRUE) //pulsación de intro
169     {
170       //nuevo objeto ccomando
171       ccomando *nuevo_comando=ordenar_comando( nbuffer.get_cadena() );  //se evalua cadena y se separan argumentos
172 
173       if(nuevo_comando!=NULL)   //en caso de comando válido
174       gestionar_comando(nuevo_comando,figura);  //se evaluan argumentos y si son correctos
175                                                 //se ejecuta la función necesaria
176     }
177 
178    //gestión de caracteres no imprimibles
179     if(caracter=='\b') //caso de backspace
180     {
181     printf("%c",'\b');
182     printf(" "); //se borra tecla anterior en la consola
183     }
184 
185     if (flag_alt==0)
186     printf("%c",caracter);  //presentación del caracter en la consola
187   }// Cierre de si se ha pulsado cualquier tecla
188   mievento->setHandled(); //evento atendido
189 } //Cierre funcion
190 
191 /*************************************************************/
192 /*  función para la creación de nodos necesarios para la      */
193 /*  obtención de información de los vértices de un objeto     */
194 /*************************************************************/
195 static SoSeparator *pickedPointsSubGraph(void)
196 {
197   SoSeparator * picksep = new SoSeparator;
198   SoPickStyle * pickstyle = new SoPickStyle;
199   pickstyle->style = SoPickStyle::SHAPE;
200   picksep->addChild(pickstyle);
201 
202   //punto de seleccion
203   SoDrawStyle * drawstyle = new SoDrawStyle;
204   drawstyle->style=SoDrawStyle::INVISIBLE;
205   drawstyle->pointSize = 1; //anchura del punto del punto de seleccion
206   picksep->addChild(drawstyle);
207 
208   // The SoCoordinate3 pointer is a global variable.
209   SoCoordinate3 *pickpoints = new SoCoordinate3;
210   picksep->addChild(pickpoints);
211 
212   picksep->addChild(new SoPointSet);
213   picksep->setName("PickSep");
214   return picksep;
215 }
216 /*************************************************************************************/
217 /*  Encuentra el vértice más cercano para un punto de la intersección          */
218 /*  en una figura compuesta por diversas caras.
    */
```

```
219 /*****************************************************************
    **************/
220 bool BuscarVerticeCercano(const SoPickedPoint *pickedPoint,
221                                     SoCoordinate3 *&coordNode,
222                                     long &closestIndex)
223 {
224    const SoDetail *pickDetail = pickedPoint->getDetail();
225 
226    if ( (pickDetail != NULL) &&
    pickDetail->getTypeId()==SoFaceDetail::getClassTypeId() )
227    {
228 
229       SoFaceDetail *faceDetail = (SoFaceDetail *) pickDetail;
230       SoSearchAction  mySearchAction;
231       mySearchAction.setType(SoCoordinate3::getClassTypeId());
232       mySearchAction.setInterest(SoSearchAction::LAST);
233       mySearchAction.apply(pickedPoint->getPath());
234 
235       if (mySearchAction.getPath() != NULL) {
236          coordNode = (SoCoordinate3 *)
237                      mySearchAction.getPath()->getTail();
238          SbVec3f objIntersect = pickedPoint->getObjectPoint();
239 
240          double minDistance = 1e12;
241          closestIndex = -1;
242          for (long i = 0; i < faceDetail->getNumPoints(); i++) {
243             long pointIndex =
244                   faceDetail->getPoint(i)->getCoordinateIndex();
245             float curDistance = (coordNode->point[pointIndex] -
246                   objIntersect).length();
247             if (curDistance < minDistance) {
248                closestIndex = pointIndex;
249                minDistance = curDistance;
250             }
251          }
252          if (closestIndex >= 0)
253            return TRUE;
254       }
255    }
256    return FALSE;
257 }
258 
259 
260 /*****************************************************************
    *********/
261 /*    Función para la gestión de eventos generados por el ratón
    (callback)*/
262 /*****************************************************************
    *********/
263 void eventoraton(void *userData, SoEventCallback *mievento)
264 {
265   if ( seleccion->getNumSelected()!=0 )
266   {
267   const SoMouseButtonEvent *EventoMouse = (SoMouseButtonEvent
    *)mievento->getEvent();
268 
269   if (EventoMouse->getButton() == SoMouseButtonEvent::BUTTON1 &&
270       EventoMouse->getState() == SoButtonEvent::DOWN)
271   {
272 
273     SoNode *nraiz = (SoNode *)userData; //acceso al nodo raiz
```

```
274
275     SoPath *pathseleccionado=(SoPath *)  (seleccion->getPath(0) );
276     SoNode *nodoSeleccionado=(SoNode *)  (
    pathseleccionado->getTail());
277     char *nombretipo= (char *)
    nodoSeleccionado->getName().getString() ;
278     //obtencion del nodo padre
279     SoNode *nodoPadre= obtenerPadre( nodoSeleccionado,nraiz);
280
281     //creamos un objeto para detectar puntos("rayo selector" :-s )
282     SoRayPickAction rayo(eviewer->getViewportRegion());
283     rayo.setPoint(EventoMouse->getPosition());
284     rayo.apply(eviewer->getSceneManager()->getSceneGraph());
285
286     //creamos el nodo necesario para almacenar la diversa
    informacion
287     //del punto de la superficie de un objeto que ha sido alcanzado
288       SoPickedPoint *point = rayo.getPickedPoint();
289
290     //se evita el intentar obtener informacion
291     if(point!=NULL) //aun existiendo un punto clicado perteneciente
    a un objeto
292     {
293       SoNode * nodoClic=(SoNode *)(point->getPath())->getTail();
    //si dicho objeto
294       char *nombreNodoClic=(char *)
    nodoClic->getName().getString(); //clicado no es
295       if( ( strcmp(nombreNodoClic,"malla")!=0 )&&(
    strcmp(nombreNodoClic,"mallaT")!=0 ) ) //del tipo malla
296         return; //salimos
297     }
298     else //y si no se obtiene ninguna informacion...
299     return;  //también salimos
300
301   //obtenemos informacion del punto interceptado por el rayo
302   SbVec3f v = point->getPoint(); //obtenemos coordenadas del
    punto(espacio mundo3D)
303   SbVec3f vo= point->getObjectPoint(); //coordenadas (espacio
    objeto)
304   int Imaterial=point->getMaterialIndex();
305
306   //busqueda del vértice más cercano
307   SoCoordinate3 *coordenadas=(SoCoordinate3
    *)seleccionarHijo(nodoPadre,"Coordenadas");
308   long indiceP;
309   BuscarVerticeCercano(point,coordenadas,indiceP);
310   //Presentación de los valores obtenidos
311   printf("-------Valores devueltos:--------\nIndice:
    %ld\n",indiceP);
312   printf("Valor del pixel X=%ld, Y=%ld -> %.2f
    Celsius\n",(long)(coordenadas->point[indiceP][0]),(long)(coordenada
    s->point[indiceP][1]),coordenadas->point[indiceP][2]);
313
314   //Captura de datos para uso en diferentes comandos
315     static int numdatos=0;
316     static int X[2];
317     static int Y[2];
318     if ((capturaPuntos==1) && numdatos<2) //captura de datos
319      {
320
    X[numdatos]=(float)(coordenadas->point[indiceP][0])-(float)(coorden
```

```
     adas->point[0][0]);
 321
     Y[numdatos]=(float)(coordenadas->point[indiceP][1])-(float)(coorden
     adas->point[0][1]);
 322
 323          printf("\nCapturado dato %d\n",numdatos);
 324          numdatos++;
 325       }
 326      if(numdatos==2) //terminada la captura de datos...
 327      {
 328        seleccionarArea(X[0],X[1],Y[0],Y[1],(SoSeparator
     *)nodoPadre,1);
 329      capturaPuntos=0; //reset flag
 330      numdatos=0;
 331      }
 332    }
 333    }
 334    return;
 335  }
```

```
 1 #include "StdAfx.h"
 2 #include <Inventor/SbVec3f.h>
 3 #include <Inventor/Win/SoWin.h>
 4 #include <Inventor/Win/viewers/SoWinExaminerViewer.h>
 5 #include <Inventor/nodes/SoMaterial.h>
 6 #include <Inventor/nodes/SoCoordinate3.h>
 7 #include <Inventor/nodes/SoSeparator.h>
 8 #include <Inventor/nodes/SoSelection.h>
 9 #include <Inventor/nodes/SoInfo.h>
10 #include <Inventor/SbColor.h>
11 #include "colores.h"
12 #include "leerfichero.h"
13 #include "math.h"
14 #include "nodos.h"
15 #include "figuras.h"
16
17 extern SoWinExaminerViewer *eviewer;
18 extern SoSelection *seleccion;
19
20 SbColor *generar_paleta( float Lcolor,float Scolor, unsigned long
   escala, char *canales)
21 {
22
23   unsigned long i=0;
24   double incremento=0;
25   int codigo=0;
26
27   if(strlen(canales)>3 )  //cadena incorrecta
28     return(NULL); //
29   else
30   {
31     SbColor *ppaleta=new SbColor[escala];  //creación de paleta del
   tamaño indicado.
32     incremento=(Scolor-Lcolor)/(escala-1);
33     for(i=0; i< (int)(strlen(canales)); i++)
34     {
35         codigo=codigo+toupper(canales[i]);  //suma de ASCII
36     }
37     //creación de paletas progresivas
38     switch(codigo)
39     {
40     case 219:  //RGB
41       for(i=0;i<escala;i++)
42       {
43         ppaleta[i][0]=(float)((Lcolor+i*incremento)/255.0);
44         ppaleta[i][1]=(float)((Lcolor+i*incremento)/255.0);
45         ppaleta[i][2]=(float)((Lcolor+i*incremento)/255.0);
46       }
47     break;
48     case 153:   //R+G
49       for(i=0;i<escala;i++)
50       {
51         ppaleta[i][0]=(float)((Lcolor+i*incremento)/255.0);
52         ppaleta[i][1]=(float)((Lcolor+i*incremento)/255.0);
53         ppaleta[i][2]=0;
54       }
55     break;
56     case 148:  //R+B
57       for(i=0;i<escala;i++)
58       {
59         ppaleta[i][0]=(float)((Lcolor+i*incremento)/255.0);
```

```
 60          ppaleta[i][1]=0.0;
 61          ppaleta[i][2]=(float)((Lcolor+i*incremento)/255.0);
 62        }
 63      break;
 64      case 137: //B+G
 65        for(i=0;i<escala;i++)
 66        {
 67          ppaleta[i][0]=0;
 68          ppaleta[i][1]=(float)((Lcolor+i*incremento)/255.0);
 69          ppaleta[i][2]=(float)((Lcolor+i*incremento)/255.0);
 70        }
 71      break;
 72      case 82: //R
 73        for(i=0;i<escala;i++)
 74        {
 75          ppaleta[i][0]=(float)((Lcolor+i*incremento)/255.0);
 76          ppaleta[i][1]=0;
 77          ppaleta[i][2]=0;
 78        }
 79      break;
 80      case 71: //G
 81        for(i=0;i<escala;i++)
 82        {
 83          ppaleta[i][0]=0;
 84          ppaleta[i][1]=(float)((Lcolor+i*incremento)/255.0);
 85          ppaleta[i][2]=0;
 86        }
 87      break;
 88      case 66: //B
 89        for(i=0;i<escala;i++)
 90        {
 91          ppaleta[i][0]=0.0;
 92          ppaleta[i][1]=0.0;
 93          ppaleta[i][2]=(float)((Lcolor+i*incremento)/255.0);
 94        }
 95      break;
 96      } //cierre de evaluacion del codigo de las letras
 97
 98      return(ppaleta);
 99    } //cierre del else
100  };
101
102  /*****************************************************************
     ********/
103  /*  Aplica a una franja de temperaturas especificada el color
     pasado      */
104  /*****************************************************************
     ********/
105  int isoterma( float Tinic,float Tfinal, struct colorRGB
     colorFranja, SoSeparator *grupo_figura )
106  {
107    //detección del tipo de objeto pasado
108    char *nombreGrupo=(char *) (grupo_figura->getName().getString());
109    if( (nombreGrupo==NULL) || (
     (strcmp(nombreGrupo,"malla_grupo")!=0) &&
     (strcmp(nombreGrupo,"mallaT_grupo")!=0) &&
     (strcmp(nombreGrupo,"nube_grupo")!=0)) )
110      return(-1);  //salimos con errores
111
112    //obtenemos el nodo de Coordenadas
113    SoCoordinate3 *coordenadas=(SoCoordinate3 *)seleccionarHijo(
```

```
    (SoNode *)grupo_figura, "Coordenadas");
114   if(coordenadas==NULL)
115     return(-2);  //error
116   SoMaterial *material=(SoMaterial *)seleccionarHijo( (SoNode
    *)grupo_figura,"material");
117   if(coordenadas==NULL)
118     return(-3);  //error
119
120   unsigned long numElementos=(unsigned
    long)coordenadas->point.getNum();
121   unsigned long numColores=(unsigned
    long)material->diffuseColor.getNum();
122   float valorTemp=0;
123   SbColor colorFondo=(SbColor) eviewer->getBackgroundColor();
124   SbColor colorAplicar;
125
126   //adecuación de los valores para los componentes de color
127   if(colorFranja.componenteR>255)
128     colorFranja.componenteR=colorFondo[0];
129   else
130     colorFranja.componenteR=colorFranja.componenteR/255.0;
131
132   if(colorFranja.componenteG>255)
133     colorFranja.componenteG=colorFondo[1];
134   else
135     colorFranja.componenteG=colorFranja.componenteG/255.0;
136
137   if(colorFranja.componenteB>255)
138     colorFranja.componenteB=colorFondo[2];
139   else
140     colorFranja.componenteB=colorFranja.componenteB/255.0;
141
142   colorAplicar.setValue(
    colorFranja.componenteR,colorFranja.componenteG,colorFranja.compone
    nteB);
143   //Se establecen los nuevo valores
144   if(numElementos=numColores)
145   {
146   for (unsigned long i=0;i<numElementos;i++)
147     {
148       valorTemp=coordenadas->point[i][2];  //temperatura (coord. Z)
149       if ( ((valorTemp<Tinic) && (valorTemp>Tfinal)) ||
    (valorTemp>Tinic) && (valorTemp<Tfinal) ) //excluidos los limites
150         material->diffuseColor.set1Value(i,colorAplicar);
151     }
152   }
153 }
154
155 /*****************************************************************
    ****/
156 /* Invierte cada uno de los colores aplicados a la malla
    */
157 /*****************************************************************
    ****/
158 void invertirColores( SoSeparator *grupo_figura )
159 {
160   //obtencion informacion del objeto seleccionado
161   infObjeto *nuevaInfo=infoObjeto (0,0,0,0,0,0,1,(SoNode *)
    grupo_figura); //mostrar=1 para obtener solo los resultados
162
163   //obtencion del nodo de material
```

```
164   SoMaterial *material=(SoMaterial *)seleccionarHijo( (SoNode
    *)grupo_figura,"material");
165   unsigned char compR,compG,compB;
166   SbVec3f color;
167   if (material!=NULL)  //en caso de encontrar el nodo
168   {
169     unsigned long numElem=(material->diffuseColor.getNum());
170     for (unsigned long i=0;i<numElem ;i++) //recorremos todos los
    colores
171     {
172       compR=unsigned char(material->diffuseColor[i][0]*255);
173       compG=unsigned char(material->diffuseColor[i][1]*255);
174       compB=unsigned char(material->diffuseColor[i][2]*255);
175       color.setValue((unsigned char)(~compR)/255.0,(unsigned
    char)(~compG)/255.0,(unsigned char)(~compB)/255.0);
176       material->diffuseColor.set1Value(i,color );
177       seleccion->touch();
178     }
179   }
180 }
181
```

```cpp
#include "StdAfx.h"
#include <Inventor/SbVec3f.h>
#include <Inventor/Win/SoWin.h>
#include <Inventor/Win/viewers/SoWinExaminerViewer.h>
#include <Inventor/nodes/SoMaterial.h>
#include <Inventor/nodes/SoSeparator.h>
#include <Inventor/nodes/SoCoordinate3.h>
#include <Inventor/nodes/SoInfo.h>
#include <Inventor/nodes/SoTransform.h>
#include <Inventor/nodes/SoDrawStyle.h>
#include <Inventor/nodes/SoLineSet.h>
#include "datos.h"
#include "nodos.h"
#include "figuras.h"
#include "math.h"

/***********************************************/
/* Constructor clase Shistograma               */
/***********************************************/
datosHistograma::datosHistograma()
{
  histograma=NULL;
  Tmax=0;
  Tmin=0;
  maxOcurrencias=0;
}


/*****************************************************************
**********/
/* Calcula el valor medio de la coordenadaZ de todos los vértices
que compo */
/* nen la malla/nube de puntos
*/
/*****************************************************************
**********/
double promedio(SoSeparator *grupo_figura)
{
  double valorPromedio=0;
  unsigned long numElementos=0;

  SoCoordinate3 *coordenadas=(SoCoordinate3*)seleccionarHijo(
(SoNode *)grupo_figura,"Coordenadas");
     if(coordenadas==NULL)
   {
     printf("\nError. no se ha podido acceder al nodo de
coordenadas\n");
     return(NULL);
   }
   numElementos=(unsigned long)(coordenadas->point.getNum());

  for (unsigned long i=0; i < numElementos ;i++) //se recorren
todos los puntos del objeto
  {
    valorPromedio=valorPromedio+(double)(coordenadas->point[i][2]);
//sumando las coordenadas Z

  }
  valorPromedio=valorPromedio/numElementos;
  return(valorPromedio);
}
```

```
54
55 /*****************************************************************
   **********/
56 /*  Obtiene el valor de temperatura de la coordenada indicada del
   objeto    */
57 /*  seleccionado
   */
58 /*****************************************************************
   **********/
59 double tempPixel(unsigned int X, unsigned int Y,SoSeparator
   *grupo_figura)
60 {
61   double tempPixel=0;
62   //obtencion de la informacion del objeto
63   class infObjeto *nuevaInfo=infoObjeto(0,0,0,0,0,0,1,(SoNode
   *)grupo_figura);
64   //acceso a las coordenadas
65   SoCoordinate3 *coordenadas=(SoCoordinate3*)seleccionarHijo(
   (SoNode *)grupo_figura,"Coordenadas");
66      if(coordenadas==NULL)
67    {
68      printf("\nError. no se ha podido acceder al nodo de
   coordenadas\n");
69      return(NULL);
70    }
71
72    if( (X>nuevaInfo->numColumnas)||(X<0) )
73    {
74      printf("\ncoordenada X fuera de rango.Max valor de X:0 -
   %d\n",nuevaInfo->numColumnas-1);
75        return(NULL);
76    }
77
78    if( (Y>nuevaInfo->numFilas)||(Y<0) )
79    {
80      printf("\ncoordenada Y fuera de rango.rango: 0 -
   %d\n",nuevaInfo->numFilas-1);
81        return(NULL);
82    }
83    tempPixel=coordenadas->point[nuevaInfo->numColumnas*Y+X][2];
84    return(tempPixel);
85 }
86
87 /*****************************************************************
   ********/
88 /* Vuelca el contenido del histograma del objeto seleccionado
   */
89 /* en un archivo de tipo texto.Esta funcion se puede llamar tanto
   para    */
90 /* obtener los datos de de una figura sin necesitar realizar la
   represen- */
91 /* tación del hostograma en pantalla como para únicamente obtener
   los     */
92 /* los datos de un histograma seleccionado
   */
93 /*****************************************************************
   ********/
94
95 int h2txt ( char *nombreArchivo,SoSeparator *grupo_figura)
96 {
97   //evaluación delñ tipo de objeto
```

```
 98    char *nombre=(char *)(grupo_figura->getName().getString());
 99    if ( (strcmp(nombre,"nube_grupo")!=0) &&
    (strcmp(nombre,"malla_grupo")!=0) &&
    (strcmp(nombre,"mallaT_grupo")!=0) )
100      return(-1); //error
101
102    //obtencion de la informacion del objeto seleccionado
103    class infObjeto *nuevaInfo=infoObjeto(0,0,0,0,0,0,1, (SoNode *)
    grupo_figura);
104
105    //nuevo objeto para contener los datos
106    datosHistograma *nuevoHisto=cargar_histograma(
    grupo_figura,NULL,NULL,NULL);
107
108    //creamos un archivo de texto para almacenar los datos
109    FILE *parchivo=fopen(nombreArchivo,"w");
110
111    if (parchivo==NULL) {printf("\n\nError al abrir el archivo para
    escritura\n"); return(-2); }
112
113    //Información de cabecera
114    fprintf(parchivo,"Archivo Origen de datos:
    %s\n",nuevaInfo->origen_datos);
115    fprintf(parchivo,"Nº filas del objeto:
    %ld\n",nuevaInfo->numFilas);
116    fprintf(parchivo,"Nº columnas del objeto:
    %ld\n\n",nuevaInfo->numColumnas);
117
118    fprintf(parchivo,"Numero de elementos:
    %ld\n",nuevaInfo->numColumnas*nuevaInfo->numFilas);
119    fprintf(parchivo,"Temperatura Maxima (ºC): %.3f\nTemperatura
    Mínima: %.3f\n",nuevoHisto->Tmax,nuevoHisto->Tmin);
120    fprintf(parchivo,"Número máximo de ocurrencias:
    %ld\n\n",nuevoHisto->maxOcurrencias);
121
122    fprintf(parchivo,"indice pal          Temp(ºC)
    Ocurrencias          % total\n");
123    fprintf(parchivo,"-----------------------------------------------
    ------------------------\n");
124    //se escribe el numero de valores y ocurrencias de cada uno
125    float temperatura=0,tpc=0; //variables para calculo temporal
126
127    for (int i=0; i<256;i++)
128    {
129
    temperatura=float(((nuevaInfo->maxValorZ-nuevaInfo->minValorZ)/255.
    0)*i+nuevaInfo->minValorZ);
130      tpc=float((nuevoHisto->ocurrencias[i]*100.0)/
    (nuevaInfo->numColumnas*nuevaInfo->numFilas));
131      fprintf(parchivo,"  %3.d             %7.3f           %10.ld
    %7.3f\n",i,temperatura,nuevoHisto->ocurrencias[i],tpc);
132    }
133    fclose(parchivo);
134
135  return(0);
136  }
137
138  /*****************************************************************
    **/
139  /*    Crea el grupo de nodos necesarios para la representación
    */
```

```
140 /*    del histograma de temperaturas del objeto seleccionado
    */
141 /*****************************************************************
    **/
142 class datosHistograma *cargar_histograma ( SoSeparator
    *grupo_figura, char *TituloEjeX,char *TituloEjeY, char
    *idHistograma)
143 {
144   //se obtiene el nodo de coordenadas del objeto
145   SoInfo *NombreHistograma=new SoInfo;
146   SoCoordinate3 *pcoord=(SoCoordinate3 *) seleccionarHijo( (SoNode
    *)grupo_figura,"Coordenadas");
147     SoInfo *temperaturas;
148
149   //creacion de nuevo objeto de tipo para contener los datos del
    histograma
150   datosHistograma *ndhisto=new datosHistograma;
151
152   double Tmin,Tmax;
153   //obtencion de Tmin y Tmax del objeto seleccionado
154   temperaturas=(SoInfo *) seleccionarHijo( (SoNode
    *)grupo_figura,"Tmax");
155   Tmax=atof( (temperaturas->string.getValue()).getString() );
156
157   temperaturas=(SoInfo *) seleccionarHijo( (SoNode
    *)grupo_figura,"Tmin");
158   Tmin=atof( (temperaturas->string.getValue()).getString() );
159
160   ndhisto->Tmin=Tmin;
161   ndhisto->Tmax=Tmax;
162
163   int indice=0;
164   unsigned int numElementos=pcoord->point.getNum();
165   double Z=0;
166
167   for (unsigned long i=0;i<256;i++)
168   {
169     ndhisto->ocurrencias[i]=0;  //inicialización array
170   }
171
172   for ( i=0;i< numElementos;i++)
173   {
174     Z= pcoord->point[i][2];  //obtención de la temperatura
175
176     indice=(int)( ( (Z-Tmin)/(Tmax-Tmin) )*255+0.5); //se obtiene
    el indice a la paleta de color mediante la temperatura (se asume el
    truncamiento)
177
    ndhisto->ocurrencias[indice]=ndhisto->ocurrencias[indice]+1;//se
    incrementan las cuentas el indice correspondiente
178   }
179   //obtención máximas ocurrencias
180   for(i=0;i<256;i++)
181   {
182     if(ndhisto->maxOcurrencias < ndhisto->ocurrencias[i])
183       ndhisto->maxOcurrencias = ndhisto->ocurrencias[i];
184   }
185
186   //factor de escala para proporciones
187   float FactorEscala=ndhisto->maxOcurrencias/255;
188
```

```
189   //nuevas instancias de nodos necesarios
190   ndhisto->histograma=new SoSeparator;
191   ndhisto->histograma->setName("Histograma");
192   SoTransform *transforma=new SoTransform;
193   SoDrawStyle *estilo=new SoDrawStyle;
194   estilo->setName("estilo");
195   transforma->setName("Transforma");
196   SoCoordinate3 *coordenadas=new SoCoordinate3;
197   coordenadas->setName("Coordenadas");
198   SoLineSet *lineas=new SoLineSet;
199
200   estilo->lineWidth=2; //ancho de la linea;
201
202   SbVec3f *vertices=new SbVec3f[256*2];//+4]; //para contener las
    coordenadas vértices de las lineas de los ejes
203
204   SoTransform *transformaObjeto=(SoTransform *)seleccionarHijo(
    (SoNode *)grupo_figura,"Transforma");
205
206      //copiamos los valores de cada campo en el nodo SoTransform del
    histograma
207      transforma->translation.setValue(
    transformaObjeto->translation.getValue() );
208      transforma->rotation.setValue(
    transformaObjeto->rotation.getValue() );   //valor de rotación
209      transforma->scaleFactor.setValue(
    transformaObjeto->scaleFactor.getValue() );  // factor de escala
210      transforma->center.setValue(
    transformaObjeto->center.getValue() );  // centro del objeto
211      transforma->scaleOrientation.setValue(
    transformaObjeto->scaleOrientation.getValue() );  //esc orientación
212
213      //ubicación predeterminada del histograma
214      SbVec3f posicion=transforma->translation.getValue();
    //obtenemos los datos de translacion
215
216      transforma->translation.setValue(posicion);
217
218      //barras histograma
219      for( i=0 ;i <256;i++)
220      {
221        vertices[2*i][0]=i;
222        vertices[2*i][1]=0;
223        vertices[2*i][2]=0;
224
225        vertices[2*i+1][0]=i;
226        if(ndhisto->ocurrencias[i]==0)
227          vertices[2*i+1][1]=0;
228        else
229
    vertices[2*i+1][1]=ndhisto->ocurrencias[i]*100/ndhisto->maxOcurrenc
    ias;
230        vertices[2*i+1][2]=0;
231      }
232
233   int *verticeslinea=new int[256];
234   //indicamos los vertices que forman cada linea
235   for (i=0;i<256;i++)
236   {
237     verticeslinea[i]=2;  //.. ;-)
238   }
```

```
239
240   //introducción de valores en los nodos correspondientes
241   lineas->numVertices.setValues(0,256,verticeslinea);
242   coordenadas->point.setValues(0,256*2,vertices);
243
244     //creación nuevo eje de coordenadas
245     ejecoordenadas *nuevoEje=new ejecoordenadas;
246     //y de un objetyo de información
247     infObjeto *nuevaInfo=infoObjeto (0,0,0,0,0,0,1,(SoNode *)
    grupo_figura);
248
249
250     nuevoEje->maxvalueX=255;
251     nuevoEje->maxvalueY=100;
252     nuevoEje->maxvalueZ=1;  //valor mínimo debe ser 1
253
254     nuevoEje->minvalueX=0;
255     nuevoEje->minvalueY=0;
256     nuevoEje->minvalueZ=1;
257
258     nuevoEje->colorejeX->componenteR=1;
259     nuevoEje->colorejeX->componenteG=0;
260     nuevoEje->colorejeX->componenteB=0;
261
262     nuevoEje->colorejeY->componenteR=0;
263     nuevoEje->colorejeY->componenteG=1;
264     nuevoEje->colorejeY->componenteB=0;
265
266     nuevoEje->colorejeZ->componenteR=0;
267     nuevoEje->colorejeZ->componenteG=0;
268     nuevoEje->colorejeZ->componenteB=1;
269
270     //Establecimiento de los titulos X e Y
271     if (TituloEjeX==NULL)
272     nuevoEje->tituloX->sprintf("%s","Indicepal");
273     else
274     nuevoEje->tituloX->sprintf("%s",TituloEjeX);
275
276     if(TituloEjeY==NULL)
277     nuevoEje->tituloY->sprintf("%s %d %s","Ocurrencias ( \%
    sobre",ndhisto->maxOcurrencias,")");
278     else
279     nuevoEje->tituloX->sprintf("%s",TituloEjeY);
280
281     nuevoEje->tituloZ->sprintf("%s","");  //sin uso del ejej Z
282
283     //numero de marcas para cada eje
284     nuevoEje->marcasX=15;
285     nuevoEje->marcasY=10;
286     nuevoEje->marcasZ=1;
287     //y resto de propiedades
288     nuevoEje->tamanoFuente=10;
289     nuevoEje->longmarcaX=5;
290     nuevoEje->longmarcaY=5;
291     nuevoEje->longmarcaZ=5;
292     nuevoEje->separacionEtiqueta=10;
293
294     SoSeparator *nodoEje=generarEjeCoord(nuevoEje);
295     nodoEje->setName("EjeCoord");
296
297     nodoEje->ref();
```

```
298       //FIN GENERACIÓN DEL EJE
299
300     if(idHistograma!=NULL)  //nombre identificativo
301     NombreHistograma->string=idHistograma;
302     else
303     NombreHistograma->string=""; //cadena vacia
304
305     NombreHistograma->setName("Nombre");
306
307       ndhisto->histograma->addChild(NombreHistograma);
308       ndhisto->histograma->addChild(transforma);
309       ndhisto->histograma->addChild(nodoEje);
310       ndhisto->histograma->addChild(coordenadas);
311       ndhisto->histograma->addChild(estilo);
312       ndhisto->histograma->addChild(lineas);
313       ndhisto->histograma->unrefNoDelete();
314
315     return(ndhisto);
316   }
317
318
319   /*****************************************************************
      /
320   /* Recopila informacion del objeto seleccionado y la vuelca un
      */
321   /* archivo de texto
      */
322   /*****************************************************************
      /
323   int v2txt(char *nombreFichero,SoSeparator *grupo_figura)
324   {
325     //obtención información sobre el tipo de objeto
326     class infObjeto *nuevaInfo=infoObjeto(0,0,0,0,0,0,1, (SoNode *)
      grupo_figura);
327     //y delas coordenadas de sus vértices
328     SoCoordinate3 *pcoordenadas=(SoCoordinate3 *) seleccionarHijo(
      (SoNode *)grupo_figura,"Coordenadas");
329
330     if(pcoordenadas==NULL)
331     { printf("\nImposible generar el archivo de datos ASCII.\nNo se
      ha podido acceder al nodo de coordenadas\n");
332       return(-1); //fin de la función
333     }
334
335     unsigned long numElementos=(unsigned
      long)pcoordenadas->point.getNum();
336     //apertura de archivo de texto para escritura
337     FILE *parchivo=fopen(nombreFichero,"w");
338     if(parchivo==NULL)
339     {
340       printf("\nError al abrir el fichero para escritura de
      datos\n");
341       return(-1);
342     }
343
344     //escritura de la informacion de cabecera del archivo:
345     fprintf(parchivo, "Archivo Origen de datos:
      %s\n",nuevaInfo->origen_datos);
346     fprintf(parchivo, "Tipo de objeto:
      %s\n",nuevaInfo->nombre_grupo);
347     fprintf(parchivo, "Nº filas del objeto:
```

```
     %ld\n",nuevaInfo->numFilas);
348    fprintf(parchivo, "Nº columnas del objeto:
     %ld\n",nuevaInfo->numColumnas);
349      fprintf(parchivo, "Máx X: %.3f\n    Y: %.3f\nTemperatura
     máxima:
     %.3f(ºC)\n",nuevaInfo->maxValorX,nuevaInfo->maxValorY,nuevaInfo->ma
     xValorZ);
350    fprintf(parchivo,"Min X: %.3f\n    Y:%.3f\nTemperatura mínima:
     %.3f(ºC)\n",nuevaInfo->minValorX,nuevaInfo->minValorY,nuevaInfo->mi
     nValorZ);
351    fprintf(parchivo,"Temperatura media:
     %.3f(ºC)\n\n",nuevaInfo->vmedio);
352
353
354    //y a continuación todos los datos de temperatura
355    for(unsigned int i=0; i< nuevaInfo->numFilas;i++)
356    {
357      for(unsigned int j=0; j< nuevaInfo->numColumnas;j++)
358      {
359        if(j<nuevaInfo->numColumnas-1)
360        fprintf(parchivo,"%.3f
     ",pcoordenadas->point[i*nuevaInfo->numColumnas+j][2]);
361        else  //para el último valor
362
     fprintf(parchivo,"%.3f\n",pcoordenadas->point[i*nuevaInfo->numColum
     nas+j][2]); //escribimos y comenzamos nueva fila
363      }
364    }
365    //cierre fichero
366    fclose(parchivo);
367    return(0);
368  }
369
370  /*****************************************************************
     /
371  /* Almacena en un fichero binario los datos de temperatura que
     */
372  /* componen la termografia
     */
373  /*****************************************************************
     /
374  int V2Bin (char *nombreFichero,SoSeparator *grupo_figura)
375  {
376    //obtención de la información sobre el tipo de objeto
377    class infObjeto *nuevaInfo=infoObjeto(0,0,0,0,0,0,1, (SoNode *)
     grupo_figura);
378    //y acceso a coordenadas de los vértices
379    SoCoordinate3 *pcoordenadas=(SoCoordinate3 *) seleccionarHijo(
     (SoNode *)grupo_figura,"Coordenadas");
380
381    if(pcoordenadas==NULL)
382    { printf("\nImposible generar el archivo de datos Binarios.\nNo
     se ha podido acceder al nodo de coordenadas\n");
383      return(-1); //fin de la función
384    }
385
386    unsigned long numElementos=(unsigned
     long)pcoordenadas->point.getNum();
387    FILE *parchivo=fopen(nombreFichero,"wb");
388    if(parchivo==NULL)
389    {
```

```
390       printf("\nError al abrir el fichero para escritura de
      datos\n");
391       return(-1);
392     }
393     //ancho y alto de la imagen
394     fwrite(&nuevaInfo->numColumnas,2,1,parchivo);
395     fwrite(&nuevaInfo->numFilas,2,1,parchivo);
396
397     //y los datos de temperatura
398     for(unsigned int i=0; i< numElementos;i++)
399       fwrite(&pcoordenadas->point[i][2],4,1,parchivo);
400     //cierre del fichero
401     fclose(parchivo);
402
403     return(0);
404   }
```

```cpp
#include "stdAfx.h"
#include <Inventor/nodes/SoSeparator.h>
#include <Inventor/nodes/SoSelection.h>
#include <Inventor/nodes/SoInfo.h>
#include <Inventor/nodes/SoSelection.h>
#include <Inventor/Win/viewers/SoWinExaminerViewer.h>
#include <Inventor/nodes/SoMaterialBinding.h>
#include <Inventor/nodes/SoTransform.h>
#include <Inventor/actions/SoSearchAction.h>

#include "nodos.h"
#include "gestioncomandos.h"

extern SoSelection *seleccion;

/*************************************************************************/
/*  Devuelve el nodo padre del nodo especificado.                        */
/*************************************************************************/
 SoNode * obtenerPadre(SoNode *nodo, SoNode *raiz)
  {
    SoSearchAction Nbusqueda;
    Nbusqueda.setNode(nodo);
    Nbusqueda.setInterest(SoSearchAction::FIRST);  //primero encontrado
    Nbusqueda.apply(raiz);
    SoPath * ppath = Nbusqueda.getPath();
    assert(ppath && "not found");
    if (ppath->getLength() < 2) { return NULL; } //no existe padre. ¿ADN? ;-)
    return (SoNode *)ppath->getNodeFromTail(1);
  }

/*************************************************************************/
/*  Devuelve el primer nodo encontrado con  el nombre especificado       */
/*************************************************************************/
SoNode * getNodeByName(SoNode * root, const char * name)
{
  static SoSearchAction * action;
  if ( !action ) action = new SoSearchAction;
  action->reset();
  // action->setFind(SoSearchAction::NAME);
  action->setName(SbName(name));
  action->setInterest(SoSearchAction::FIRST);
  action->apply(root);
  if ( !action->getPath() ) return NULL;
  return action->getPath()->getTail();
}


/*************************************************************************/
/*   Devuelve un puntero al primer nodo hijo con el nombre indicado.   */
/*   A la funcion se le pasa un puntero al nodo padre (en el que se      */
```

```
51 /*   buscara al nodo hijo) y el nombre del nodo hijo a buscar
   */
52 /*   (previamente se debe haber dado nombre al nodo.)
   */
53 /*****************************************************************
   *****/
54 SoNode *seleccionarHijo(SoNode *nodoPadre,char *nombreNodoHijo)
55 {
56   SoSearchAction Nbusqueda;
57   Nbusqueda.setName(nombreNodoHijo);
58   Nbusqueda.setInterest(SoSearchAction::FIRST);
59     Nbusqueda.apply(nodoPadre);
60     SoPath * ppath = Nbusqueda.getPath();
61     //Obtención del nodo buscado
62   if (ppath!=NULL)  //se ha encontrado el nodo
63   {
64     return(ppath->getNodeFromTail(0));
65   }
66   else
67     return (NULL);
68 }
69
70 /*****************************************************************
   *****/
71 /*   Recorre el nodo raíz y todos sus nodos hijos en busca del nodo
   */
72 /*   especificado. En el caso de encontrarlos los elimina.
   */
73 /*****************************************************************
   *****/
74
75 int borrar_nodos(SoNode *nraiz, char *nombreNodo)
76 {
77   int objetos_borrados=0;
78   SoSearchAction Nbusqueda;
79   Nbusqueda.setName(nombreNodo);
80   Nbusqueda.setInterest(SoSearchAction::ALL);
81   Nbusqueda.apply(nraiz);
82   SoPathList ppaths = Nbusqueda.getPaths();
83
84   //evaluación de paths
85   if (ppaths!=NULL)
86   {
87     for(int i=0;i<(ppaths.getLength() );i++) //para cada path
88     {
89       SoPath *ppath=ppaths[i];  //seleccionamos cada camino
90       SoNode *nodoEncontrado=ppath->getNodeFromTail(0); //obtenemos
   el nodo
91       SoNode *NodoPadre=obtenerPadre(nodoEncontrado,nraiz);
   //buscamos nodo padre
92       ( (SoSeparator *) NodoPadre )->removeChild(nodoEncontrado);
93       seleccion->touch();
94       objetos_borrados++;
95     }
96   }
97   return(objetos_borrados);
98 }
99
100 /*****************************************************************
   *****/
101 /* Devuelve un puntero al objeto que tenga la propiedad string del
```

```
    nodo*/
102 /* SoInfo: nombreMalla nombreMallat o nombreNube, segun el objeto.
    */
103 /*****************************************************************
    *****/
104
105 SoNode *buscarObjeto( SoNode *nraiz , char *nombreObjeto)
106 {
107  int objetos_borrados=0;
108   SoSearchAction Nbusqueda;
109   Nbusqueda.setName("Nombre");
110   Nbusqueda.setInterest(SoSearchAction::ALL);
111   Nbusqueda.apply(nraiz);
112   SoPathList ppaths = Nbusqueda.getPaths();
113
114   if (ppaths!=NULL)
115   {
116     for(int i=0;i<(ppaths.getLength() );i++)
117     {
118       SoPath *ppath=ppaths[i];
119       SoInfo *InfoNombre=(SoInfo *) ppath->getNodeFromTail(0);
    //obtención del nodo de información
120       char *cadena= (char
    *)(InfoNombre->string.getValue()).getString() ;
121       if ( strcmp(nombreObjeto,cadena )==0 )
122       {
123       SoNode *NodoPadre=obtenerPadre(InfoNombre,nraiz);//búsqueda
    nodo padre
124       return( InfoNombre);
125       }
126     }
127   }
128 return(NULL);
129 }
```

```
 1 #include "StdAfx.h"
 2 #include <Inventor/nodes/SoSelection.h>
 3 #include <Inventor/nodes/SoMaterial.h>
 4 #include <Inventor/nodes/SoInfo.h>
 5 #include <Inventor/nodes/SoQuadMesh.h>
 6 #include <Inventor/nodes/SoPointSet.h>
 7 #include <Inventor/actions/SoSearchAction.h>
 8 #include <Inventor/nodes/SoCoordinate3.h>
 9 #include <Inventor/Win/viewers/SoWinExaminerViewer.h>
10 #include <Inventor/nodes/SoTexture2.h>
11 #include <Inventor/nodes/SoCamera.h>
12 #include <Inventor/manips/SoSpotLightManip.h>
13 #include <Inventor/manips/SoTransformBoxManip.h>
14 #include <Inventor/manips/SoTrackballManip.h>
15 #include <Inventor/fields/SoSFRotation.h>
16
17 #include "gestioncomandos.h"
18 #include "string.h"
19 #include "datos.h"
20 #include "stdio.h"
21 #include "nodos.h"
22 #include "leerfichero.h"
23 #include "figuras.h"
24 #include "colores.h"
25
26 extern unsigned char capturaPuntos;
27 extern SoWinExaminerViewer *eviewer;
28
29 #define NM_COMANDOS 53
30 const char *comando_valido[]={
31   "analizarimagen",
32   "cargarnube",
33   "cargarmalla",
34   "cls",
35   "suprimir",
36   "eje",
37   "colorfondo",
38   "translacion",
39   "rotacion",
40   "escalar",
41   "centro",
42   "trackball-on",
43   "trackball-off",
44   "box-on",
45   "box-off",
46   "manipuladores-off",
47   "luz-on",
48   "luz-off",
49   "posicionluz",
50   "colorluz",
51   "luces-off",
52   "crearmalla",
53   "cortarxfila",
54   "cortarxcolumna",
55   "cortarxarea",
56   "info",
57   "info-off",
58   "texturizar",
59   "filtro",
60   "filtrobanda",
61   "referencia",
```

```
62    "autoeje",
63    "seleccionararea",
64    "reflejarv",
65    "reflejarh",
66    "invertirc",
67    "cargarmallat",
68    "asignarpaleta",
69    "v2bin",
70    "promedio",
71    "histo",
72    "posicionar",
73    "script",
74    "selobjeto",
75    "texturizar-off",
76    "v2txt",
77    "h2txt",
78    "etiqueta",
79    "autoenfoque",
80    "isoterma",
81    "poscamara",
82    "transcamara",
83    "orcamara"};
84
85  extern SoSelection *seleccion;
86  extern SoWinExaminerViewer *eviewer;
87  extern unsigned char capturaPuntos;
88      static int X1;
89      static int X2;
90      static int Y1;
91      static int Y2;
92
93
94  #define  PI 3.141592653
95
96  /* constructor por defecto */
97  ccomando::ccomando()
98  {
99    indice_comando=0;
100   num_argumentos=0;
101   long_max_argumento=0;
102 }
103
104 ccomando::set_argumento (int indice,char *cadargumento)
105 {
106   //primero creamos un array para contener la cadena
107
108   char *temporal=new char [strlen(cadargumento)+1];
109   strcpy(temporal,cadargumento);  //la copiamos
110
111   strcpy(argumento[indice],cadargumento);
112   //strcpy(argumento[indice],cadargumento);
113
114 }
115 char * ccomando::get_argumento (int indice)
116 {
117   return(this->argumento[indice]);
118 }
119
120 /* destructor */
121 ccomando::~ccomando()
122 {
```

```
123   delete [] argumento;
124 }
125
126
127 /*****************************************************************
    *****/
128 /*  Evalúa la cadena pasada y la divide en palabras ,guardándolas
    en   */
129 /*  el mismo orden en el se encuentran en la cadena
    */
130 /*****************************************************************
    *****/
131 class ccomando *ordenar_comando (char *cadena)
132 {
133   int i=0,j=0,temp=0;
134   int flag_valido=0;
135   int cont_arg=0,cont_car=0;  //contador de argumentos y de
    caracteres
136   long  max_caracter=0;  //almacena el numero de caracteres del
    argumento mas largo
137
138   //nueva instancia
139   ccomando *ncomando=new ccomando();
140   char *copia=new char [strlen(cadena)];
141   strcpy(copia,cadena);
142   for(i=0;i<(int)(strlen(cadena)+1);i++)   //recorremos cada uno de
    los caracteres
143   {
144     if ( ( ( cadena[i]==' ') || (i==strlen(cadena) ) )  &&
    (cadena[i-1]!=' ') )
145     {
146       //creamos cadena para copiar la palabra
147       char *palabra=new char [cont_car+1];
148       for(j=0;j<cont_car;j++)
149       {
150         palabra[j]=cadena[i-cont_car+j];
151       }
152       palabra=palabra+j;
153       *palabra='\0';  //finalización de la palabra
154       palabra=palabra-j;  //volvemos al comienzo
155       palabra=eliminar_espacios(palabra);
156       ncomando->set_argumento(cont_arg, palabra );
157       cont_arg++;
158       cont_car=0;
159       if(max_caracter< (int)strlen(palabra))  //guardamos la
    longitud de la cadena más larga
160         max_caracter=strlen(palabra);
161       delete [] palabra;
162       //Identificación  del parametro[0]=comando para saber si se
    trata o no de un comando válido
163        if(cont_arg==1)
164        {
165          for(temp=0;temp<NM_COMANDOS;temp++)
166          {
167            if(
    strcmp(comando_valido[temp],ncomando->argumento[0])==0 ) //comando
    NO válido
168            {
169              flag_valido=1;  //flag indicativo
170              ncomando->indice_comando=temp;  //guardamos índice del
    comando
```

```
171             }
172           }
173           if (flag_valido==0)
174           {
175             printf("\n\n  \"%s\"  no es un comando
    valido\n",ncomando->argumento[0]);
176            //ncomando->~ccomando(); //liberamos memoria
177             return NULL;  //exit
178           }
179         }
180       }
181       else
182       {
183         cont_car++;
184       }
185     }
186     //guardamos resto de las varibles miembro
187     ncomando->long_max_argumento=max_caracter;
188     ncomando->num_argumentos=cont_arg;
189     return ncomando;
190   }
191
192    void translacion_objeto (float X,float Y,float Z, SoTransform
    *pobjeto);
193    void rotacion_objeto (float X,float Y,float Z,float
    angulo,SoTransform *pobjeto);
194    void escalar_objeto (float factorX,float factorY,float
    factorZ,SoTransform *pobjeto);
195    void centro_objeto (float coordX,float coordY,float
    coordZ,SoTransform *pobjeto);
196
197   /*****************************************************************
    *****/
198   /*   Evalúa las palabras guardadas en un objeto ccomando
    identificando */
199   /*   el comando y el numero y tipo de argumentos del mismo
    */
200   /*****************************************************************
    *****/
201
202   int gestionar_comando (class ccomando *ncomando,SoSeparator *nraiz)
203   {
204     switch (ncomando->indice_comando)
205     {
206
207       //llamadas a funciones en respuesta a comandos válidos
208       case 0:  //comando "analizarimagen"
209        if(ncomando->num_argumentos==2)
210        {
211          //primero leemos el tipo de fichero
212          FIBITMAP *imagen=abrir_bitmap(ncomando->argumento[1]);
213          //generación de 3 archivos temporales
214         descomponer_fichero(imagen,ncomando->argumento[1]);
215         printf("\n\nHecho.\n");
216        }
217        else
218        {
219          printf("\n\nNumero de argumentos incorrecto\n.Se espera 1
    arguemnto");
220          printf("Uso: analizarimagen nombre_bitmap.\n");
221          return -1;
```

```
222       }
223      break;
224      case 1:  //comando "cargarnube"
225      if(ncomando->num_argumentos==9) //creacion de malla en caso de
    bitmaps
226      {
227
228         SoSeparator *pnube= generar_nube(
    "",atoi(ncomando->argumento[1]),
229          atoi(ncomando->argumento[2]),atoi(ncomando->argumento[3]),
230          atoi(ncomando->argumento[4]),atof(ncomando->argumento[5]),
231
    atof(ncomando->argumento[6]),ncomando->argumento[7],atoi(ncomando->
    argumento[8]));
232
233          pnube->ref();
234          nraiz->addChild(pnube);//se añade la nube al nodo raiz
235          printf("\n\nHecho\n");
236          seleccion->touch();
237      }
238      else if(ncomando->num_argumentos==8)//creación en caso de
    archivos binarios
239      {
240         SoSeparator *pnube= generar_nube( ncomando->argumento[1] ,
241          atoi(ncomando->argumento[2]),atoi(ncomando->argumento[3]),
242          atoi(ncomando->argumento[4]),atoi(ncomando->argumento[5]),
243          0,0,ncomando->argumento[6],atoi(ncomando->argumento[7]));
244        nraiz->addChild(pnube);
245        pnube->ref();
246        printf("\n\nHecho\n");
247        seleccion->touch(); //forzamos refresco
248      }
249      else
250      {
251         printf("\nNumero de argumentos incorrecto.Se esperan 7/8
    parametros\n");
252         printf("Uso:  cargarnube usarpaleta compR compG compB Tsup
    Tinf nombre forzar_datos\n");
253         printf("Uso:  cargarnube archivoBinario usarpaleta compR
    compG compB nombre forzar_datos\n");
254         printf("usarpaleta=1 se usará la paleta contenida en el
    fichero datospal.txt\n");
255         printf("usarpaleta=0 se usará el color en formato RGB pasado
    como compR compG y compB\n");
256         printf("Tsup es la temperatura que se asignara al color
    superior de la paleta de colores\n");
257         printf("Tmin es la temperatura que se asignara al color con
    indice 0 en la paleta de colores\n");
258         printf("nombre indica el nombre que se asignará al objeto
    seleccionado para futuras identificaciones del objeto.\n");
259         printf("forzar_datos=1 se considera a las imagenes de 24bpp
    como válidas para extraer información de temperatura\n");
260         return -1;
261      }
262      break;
263
264      case 2:  //comando "cargarmalla"
265      if(ncomando->num_argumentos==9) //creacion de malla en caso de
    bitmaps
266      {
267        SoSeparator *pmalla= generar_malla (
```

```
    "",atoi(ncomando->argumento[1]),
268         atoi(ncomando->argumento[2]),atoi(ncomando->argumento[3]),
269         atoi(ncomando->argumento[4]),atof(ncomando->argumento[5]),
270 
    atof(ncomando->argumento[6]),ncomando->argumento[7],atoi(ncomando->
    argumento[8]));
271 
272         pmalla->ref();
273         nraiz->addChild(pmalla); //se añade la malla al nodo raiz
274         printf("\n\nHecho\n");
275         seleccion->touch();
276         eviewer->viewAll(); //autoenfoque
277       }
278       else if(ncomando->num_argumentos==8)//creación en caso de
    archivos binarios
279       {
280          SoSeparator *pmalla= generar_malla ( ncomando->argumento[1]
    ,
281           atoi(ncomando->argumento[2]),atoi(ncomando->argumento[3]),
282           atoi(ncomando->argumento[4]),atoi(ncomando->argumento[5]),
283           0,0,ncomando->argumento[6],atoi(ncomando->argumento[7]));
284 
285         pmalla->ref();
286         nraiz->addChild(pmalla);
287         printf("\n\nHecho\n");
288         seleccion->touch();
289         eviewer->viewAll();//autoenfoque
290       }
291       else //ninguno de los anteriores
292       {
293          printf("\nNumero de argumentos incorrecto.Se esperan 7/8
    parametros\n");
294          printf("Uso:  cargarmalla usarpaleta compR compG compB Tsup
    Tinf nombre forzar_datos\n");
295          printf("Uso:  cargarmalla archivoBinario usarpaleta compR
    compG compB nombre forzar_datos\n");
296          printf("usarpaleta=1 se usará la paleta contenida en el
    fichero datospal.txt\n");
297          printf("usarpaleta=0 se usará el color en formato RGB pasado
    como compR compG y compB\n");
298          printf("Tsup es la temperatura que se asignara al color
    superior de la paleta de colores\n");
299          printf("Tmin es la temperatura que se asignara al color con
    indice 0 en la paleta de colores\n");
300          printf("nombre indica el nombre que se asignará a la malla
    seleccionado para futuras identificaciones del objeto.\n");
301          printf("forzar_datos=1 se considera a las imagenes de 24bpp
    como válidas para extraer información de temperatura\n");
302          return -1;
303       }
304       break;
305 
306       case 3: //comando cls.
307       if(ncomando->num_argumentos==1) //solo es necesario el nombre
    del comando
308       {
309         int numerohijos=nraiz->getNumChildren();
310         int contador=0;
311 
312         seleccion->deselectAll();//evita errores en caso de borrar
    nodos seleccionados
```

```
313         for (int i=0;i<numerohijos;i++) //recorremos todos los
    nodos hijos
314       {
315         SoNode *nodo=(SoNode *)nraiz->getChild((numerohijos-1)-i);
    //eliminando solo
316         if(nodo->getTypeId()==SoSeparator::getClassTypeId())
    //aquellos del tipo separador
317         {
318           nraiz->removeChild((numerohijos-1)-i);
319           contador++;
320         }
321       }
322       printf("\n%d objeto(s) eliminado(s)\n",contador);
323     }
324     else
325       printf("\n\nNo se necesitan parametros. uso: cls\n");
326     break;
327
328     case 4:  //comando suprimir
329       if(ncomando->num_argumentos==1)
330       {
331       if(seleccion->getNumSelected()!=0) //para el objeto
    seleccionado
332       {
333         int seleccionados=seleccion->getNumSelected();
334         for(int i=0;i<seleccionados;i++) //recorremos cada uno de
    los objetos seleccionados
335         {
336           SoNode *nodoSeleccionado=(SoNode *)  ( (SoPath *)
    (seleccion->getPath(i))->getTail());
337           char *nombretipo= (char *)
    nodoSeleccionado->getTypeId().getName().getString() ;
338           //obtención del nodo padre del objeto seleccionado
339           SoNode *nodoPadre= obtenerPadre( nodoSeleccionado,nraiz);
340           nraiz->removeChild(nodoPadre);
341           printf("\nobjeto %s borrado\n",nombretipo);
342         }
343         seleccion->deselectAll(); //evita errores en los borrados
344         printf("\n\nHecho.\n");
345       }
346       else
347        printf("\nNada a borrar.Ningun objeto seleccionado\n");
348       }
349       else
350        printf("\n\nNo se necesitan parametros. uso: suprimir\n");
351     break;
352
353     case 5: //eje de coordenadas
354
355       if(ncomando->num_argumentos==21)
356       {
357       //nuevo objeto eje de coordenadas
358       class ejecoordenadas *pnuevoeje=new class ejecoordenadas;
359
360         //posicion del eje
361         pnuevoeje->posX=(float)atof(ncomando->argumento[1]);
362         pnuevoeje->posY=(float)atof(ncomando->argumento[2]);
363         pnuevoeje->posZ=(float)atof(ncomando->argumento[3]);
364
365         //Número de marcas de cada uno de los ejes
366         pnuevoeje->marcasX=atoi(ncomando->argumento[4]);
```

```
367         pnuevoeje->marcasY=atoi(ncomando->argumento[5]);
368         pnuevoeje->marcasZ=atoi(ncomando->argumento[6]);
369
370         //valores de máximo y mínimo para cada uno de los ejes
371         pnuevoeje->maxvalueX=(float)atof(ncomando->argumento[7]);
372         pnuevoeje->maxvalueY=(float)atof(ncomando->argumento[8]);
373         pnuevoeje->maxvalueZ=(float)atof(ncomando->argumento[9]);
374
375         pnuevoeje->minvalueX=(float)atof(ncomando->argumento[10]);
376         pnuevoeje->minvalueY=(float)atof(ncomando->argumento[11]);
377         pnuevoeje->minvalueZ=(float)atof(ncomando->argumento[12]);
378
379         //longitud de las marcas para cada eje
380         pnuevoeje->longmarcaX=atoi(ncomando->argumento[13]);
381         pnuevoeje->longmarcaY=atoi(ncomando->argumento[14]);
382         pnuevoeje->longmarcaZ=atoi(ncomando->argumento[15]);
383
384
    pnuevoeje->tamanoFuente=(float)atof(ncomando->argumento[16]);
385
    pnuevoeje->separacionEtiqueta=(float)atof(ncomando->argumento[17]);
386
387         //Titulo para cada uno de los ejes
388         pnuevoeje->tituloX->sprintf("%s",ncomando->argumento[18]);
389         pnuevoeje->tituloY->sprintf("%s",ncomando->argumento[19]);
390         pnuevoeje->tituloZ->sprintf("%s",ncomando->argumento[20]);
391
392       SoSeparator *peje= generarEjeCoord(pnuevoeje);
393       ( (SoSeparator *)(nraiz->getByName("EjeCoord")) )->ref();
394       nraiz->addChild(peje);
395
396       printf("\n\nHecho\n");
397       seleccion->touch();
398       }
399       else
400       {
401         printf("\nNumero incorrecto de argumentos.Se esperan 20
    argumentos\n");
402           printf("Uso: eje coord X coordY coorZ marcasX marcasY
    marcasZ\n");
403         printf("     MaxvalorX MaxvalorY MaxvalorZ\n");
404         printf("     MinValorX MinValorY MinValorZ  longmarcaX
    longmarcaY longmarcaZ\n");
405         printf("     tamañofuente separacionEtiquetas tituloX
    tituloY tituloZ\n");
406       }
407
408     break;
409
410     case 6: //colorfondo
411       if(ncomando->num_argumentos==4)
412       {
413           SbColor colorfondo;
414           colorfondo.setValue((float)
    (atof(ncomando->argumento[1])/255.0),
415                               (float)
    (atof(ncomando->argumento[2])/255.0),
416                   (float) (atof(ncomando->argumento[3])/255.0) );
417           eviewer->setBackgroundColor(colorfondo);
418         printf("\n\nHecho.\n");
419       }
```

```
420       else
421       {
422         printf("\nNúmero de argumentos incorrecto.Se esperan 3
    argumentos\n");
423         printf("Uso:  colorfondo R  G  B      (formato 0-255)\n");
424       }
425     break;
426
427     case 7:  //translacion
428      if (ncomando->num_argumentos==4)
429      {
430       if(seleccion->getNumSelected()!=0)
431       {
432         int seleccionados=seleccion->getNumSelected();
433         for(int i=0;i<seleccionados;i++) //recorremos cada uno de
    los objetos seleccionados
434           {
435           SoPath *pathseleccionado=(SoPath *)
    (seleccion->getPath(i) );
436           if (pathseleccionado==NULL) return(-1);
437           SoNode *nodoSeleccionado=(SoNode *)  (
    pathseleccionado->getTail());
438           char *nombretipo= (char *)
    nodoSeleccionado->getTypeId().getName().getString() ;
439           //obtención del nodo padre
440           SoNode *nodoPadre= obtenerPadre( nodoSeleccionado,nraiz);
441           if (nodoPadre==NULL) return(-1);
442           SoNode *nodoTransform=
    seleccionarHijo(nodoPadre,"Transforma");
443           if (nodoTransform==NULL) return(-1);
444           //y se lo pasamos a la funcion
445           seleccion->deselect(pathseleccionado);
446           seleccion->touch();
447           translacion_objeto( (float) (atof(ncomando->argumento[1])
    ),
448                               (float) (atof(ncomando->argumento[2])),
449                                 (float) (atof(ncomando->argumento[3])),
450                       (SoTransform *)nodoTransform );
451           printf("\ntranslacion aplicada a objeto %s
    .\n",nombretipo);
452           }
453       printf("\n\nHecho.\n");
454       }
455       else
456       printf("\n\nNada a mover.Ningun objeto seleccionado\n\n");
457       }
458       else
459        printf("\n\n\nNúmero de parámetros incorrecto. Uso:
    translacion  X Y Z\n\n\n");
460     break;
461
462     case 8:  //rotacion
463      if (ncomando->num_argumentos==5)
464      {
465       if(seleccion->getNumSelected()!=0)
466       {
467         int seleccionados=seleccion->getNumSelected();
468         for(int i=0;i<seleccionados;i++)
469           {
470           SoPath *pathseleccionado=(SoPath *)
    (seleccion->getPath(i) );
```

```
471           SoNode *nodoSeleccionado=(SoNode *)  (
    pathseleccionado->getTail());
472           char *nombretipo= (char *)
    nodoSeleccionado->getTypeId().getName().getString() ;
473           //obtención del nodo padre
474           SoNode *nodoPadre= obtenerPadre( nodoSeleccionado,nraiz);
475           //y el SoTransform
476           SoTransform *nodoTransform= (SoTransform
    *)seleccionarHijo(nodoPadre,"Transforma");
477           //deselección del objeto al aplicar la rotación
478           seleccion->deselect(pathseleccionado);
479           seleccion->touch();
480           rotacion_objeto(  (float) (atof(ncomando->argumento[1])),
481                           (float) (atof(ncomando->argumento[2])),
482                             (float) (atof(ncomando->argumento[3])),
483                     (float) (atof(ncomando->argumento[4])),
484                     (SoTransform *)nodoTransform);
485           printf("\nObjeto %s rotado.\n",nombretipo);
486           }
487       printf("\n\nHecho.\n");
488       }
489         else
490       printf("\nNada a rotar.Ningun objeto seleccionado\n");
491       }
492      else
493      {
494       printf("\nNumero de argumentos incorrecto. Se esperan 4
    argumentos\n");
495       printf("Uso:\trotacion X Y Z angulo\n donde X Y y Z toman el
    valor 1 o 0");
496       printf("\tdependiendo de si se quiere o no aplicar la
    rotación al eje respectivo\n");
497       printf("Angulo en grados sexagesimales\n");
498      }
499     break;
500     case 9: //escalado
501      if (ncomando->num_argumentos==4)
502      {
503       if(seleccion->getNumSelected()!=0)
504       {
505         int seleccionados=seleccion->getNumSelected(); //guardamos
    el numero de objetos seleccionados
506         for(int i=0;i<seleccionados;i++) //recorremos cada uno de
    los objetos seleccionados
507           {
508           SoPath *pathseleccionado=(SoPath *)
    (seleccion->getPath(i) );
509           SoNode *nodoSeleccionado=(SoNode *)  (
    pathseleccionado->getTail());
510           char *nombretipo= (char *)
    nodoSeleccionado->getName().getString() ;
511           //intentamos obtener el nodo padre del nodo SoShape
    pasado
512           SoNode *nodoPadre= obtenerPadre( nodoSeleccionado,nraiz);
513           SoTransform *nodoTransform= (SoTransform
    *)seleccionarHijo(nodoPadre,"Transforma");
514           printf("\nel nodo padre se llama: %s\n",( char *)
    nodoPadre->getName().getString() );
515           //y se lo pasamos a la funcion
516           seleccion->deselect(pathseleccionado);
517           seleccion->touch();
```

```
518           escalar_objeto( (float) (atof(ncomando->argumento[1])),
519                           (float) (atof(ncomando->argumento[2])),
520                             (float) (atof(ncomando->argumento[3])),
521                       nodoTransform);
522           printf("\nEscalado aplicado a objeto %s .\n",nombretipo);
523           }
524       printf("\n\nHecho.\n");
525       }
526         else
527       printf("\nNada a escalar.Ningun objeto seleccionado\n");
528       }
529       else
530        printf("\n\nNúmero de parámetros incorrecto. Uso: ecalar
    factorX factorY factorZ\n");
531     break;
532
533     case 10:   //comando centro del objeto
534       if (ncomando->num_argumentos==4)
535       {
536         if(seleccion->getNumSelected()!=0) //si hay objetos
    seleccionados
537         {
538         int seleccionados=seleccion->getNumSelected();
539         for(int i=0;i<seleccionados;i++)
540           {
541           SoPath *pathseleccionado=(SoPath *)
    (seleccion->getPath(i) );
542           SoNode *nodoSeleccionado=(SoNode *)  (
    pathseleccionado->getTail());
543           char *nombretipo= (char *)
    nodoSeleccionado->getName().getString() ;
544           //obtenci´ñon del nodo padre
545           SoNode *nodoPadre= obtenerPadre( nodoSeleccionado,nraiz);
546           seleccion->deselect(pathseleccionado);
547           seleccion->touch();
548           //y llamada a la función convirtiendo los parámetros en
    el formato correcto
549           centro_objeto( (float) (atof(ncomando->argumento[1])),
550                         (float) (atof(ncomando->argumento[2])),
551                           (float) (atof(ncomando->argumento[3])),
552                       (SoSeparator *)nodoPadre);
553           printf("\nAsignado nuevo centro al objeto %s
    .\n",nombretipo);
554           }
555         printf("\n\nHecho.\n");
556         }
557         else
558         printf("\nNingun objeto seleccionado\n");
559       }
560       else
561       printf("Se esperan 3 parámetros. Uso: centro X Y Z.\n");
562     break;
563
564     case 11: //trackball-on
565       if (ncomando->num_argumentos==1)
566       {
567         if(seleccion->getNumSelected()!=0) //si hay objetos
    seleccionados
568         {
569         int seleccionados=seleccion->getNumSelected();
570         for(int i=0;i<seleccionados;i++) //recorremos cada uno
```

```
571           {
572           SoPath *pathseleccionado=(SoPath *)
    (seleccion->getPath(i) );
573           SoNode *nodoSeleccionado=(SoNode *)  (
    pathseleccionado->getTail());
574           char *nombretipo= (char *)
    nodoSeleccionado->getName().getString() ;
575           //obteniendo el nodo padre
576           SoNode *nodoPadre= obtenerPadre( nodoSeleccionado,nraiz);
577           //deseleccionándolo
578           seleccion->deselect(pathseleccionado);
579           seleccion->touch();
580           //y llamando a la función correspondiente
581           actdes_trackball(nodoPadre);
582           printf("\nManipulador asignado anuevo al objeto %s
    .\n",nombretipo);
583           }
584         printf("\n\nHecho.\n");
585         }
586         else
587         printf("\nNingun objeto seleccionado\n");
588       }
589       else
590         printf("No se espera ningún argumento.\nUso:
    manipuladorON\n");
591     break;
592
593     case 12: //trackball-off
594       if (ncomando->num_argumentos==1)
595       {
596         if(seleccion->getNumSelected()!=0)
597         {
598         int seleccionados=seleccion->getNumSelected();
599         for(int i=0;i<seleccionados;i++)
600           {
601           SoNode *nodoSeleccionado=(SoNode *)  ( (SoPath *)
    (seleccion->getPath(i))->getTail());
602           char *nombretipo= (char *)
    nodoSeleccionado->getName().getString() ;
603           //obtención del nodo padre
604           SoNode *nodoPadre= obtenerPadre( nodoSeleccionado,nraiz);
605           //llamamos a la función corrspondiente
606           actdes_trackball(nodoPadre);
607           printf("\nManipulador trackball desactivado en el objeto
    %s .\n",nombretipo);
608           }
609         printf("\n\nHecho.\n");
610         }
611         else
612         printf("\nNingun objeto seleccionado\n");
613       }
614       else
615         printf("No se espera ningún argumento.\nUso:
    trackball-off\n");
616
617     break;
618
619     case 13: //box-on
620       if (ncomando->num_argumentos==1)
621       {
622         if(seleccion->getNumSelected()!=0)
```

```
623         {
624         int seleccionados=seleccion->getNumSelected();
625         for(int i=0;i<seleccionados;i++)
626           {
627           SoPath *pathseleccionado=(SoPath *)
    (seleccion->getPath(i) );
628           SoNode *nodoSeleccionado=(SoNode *)  (
    pathseleccionado->getTail());
629           char *nombretipo= (char *)
    nodoSeleccionado->getName().getString() ;
630           //intentamos obtener el nodo padre del nodo SoShape
    pasado
631           SoNode *nodoPadre= obtenerPadre( nodoSeleccionado,nraiz);
632           //llamamos a la funcion
633           actdes_box(nodoPadre);
634           printf("\nManipulador tipo Box asignado al objeto %s
    .\n",nombretipo);
635           printf("El box se ha aplicado al nodo padre: %s
    \n",(nodoPadre->getName()).getString() );
636           }
637         seleccion->deselectAll();
638         printf("\n\nHecho.\n");
639         }
640         else
641         printf("\nNingun objeto seleccionado\n");
642       }
643       else
644         printf("No se espera ningún argumento.\nUso: box-on\n");
645     break;
646
647     case 14: //box-off
648       if (ncomando->num_argumentos==1)
649       {
650         if(seleccion->getNumSelected()!=0)
651         {
652         int seleccionados=seleccion->getNumSelected();
653         for(int i=0;i<seleccionados;i++)
654           {
655           SoPath *pathseleccionado=(SoPath *)
    (seleccion->getPath(i) );
656           SoNode *nodoSeleccionado=(SoNode *)  (
    pathseleccionado->getTail());
657           char *nombretipo= (char *)
    nodoSeleccionado->getName().getString() ;
658           //intentamos obtener el nodo padre del nodo SoShape
    pasado
659           SoNode *nodoPadre= obtenerPadre( nodoSeleccionado,nraiz);
660
661           seleccion->deselect(pathseleccionado);
662           seleccion->touch();
663           actdes_box(nodoPadre);
664           printf("\nManipulador desactivado en el objeto %s
    .\n",nombretipo);
665           }
666         printf("\n\nHecho.\n");
667         }
668         else
669         printf("\nNingun objeto seleccionado\n");
670       }
671       else
672         printf("No se espera ningún argumento.\nUso: box-off\n");
```

```
673     break;
674     case 15: //manipuladores-off
675       if (ncomando->num_argumentos==1)
676       {
677         //se desactivan todos los objetos que estén seleccionados
678         //y se eliminan todos los manipuladores
679         seleccion->deselectAll();
680         int borrados=desactivar_manipuladores(nraiz);
681         printf("\n\nManipuladores borrados : %d\n",borrados);
682         printf("\n\nHecho.\n");
683       }
684       else
685         printf("No se espera ningún argumento.\nUso:
    manipuladores-off\n");
686     break;
687
688     case 16: //luz-on
689       if (ncomando->num_argumentos==7)
690       {
691         if(seleccion->getNumSelected()!=0) //si hay objetos
    seleccionados
692         {
693         int seleccionados=seleccion->getNumSelected(); //guardamos
    el numero de objetos seleccionados
694         for(int i=0;i<seleccionados;i++) //recorremos cada uno de
    los objetos seleccionados
695           {
696           SoNode *nodoSeleccionado=(SoNode *)  ( (SoPath *)
    (seleccion->getPath(i))->getTail());
697           char *nombretipo= (char *)
    nodoSeleccionado->getName().getString() ;
698           //intentamos obtener el nodo padre del nodo SoShape
    pasado
699           SoNode *nodoPadre= obtenerPadre( nodoSeleccionado,nraiz);
700           //llamamos a la funcion
701           activar_LuzPosicional(nodoPadre,
    atof(ncomando->argumento[1]),
702
    atof(ncomando->argumento[2]),atof(ncomando->argumento[3]),
703
    atoi(ncomando->argumento[4]),atoi(ncomando->argumento[5]),
704             atoi(ncomando->argumento[6]) );
705           printf("\nLuz incluida en el objeto %s .\n",nombretipo);
706           }
707         printf("\n\nHecho.\n");
708         }
709         else
710         printf("\nNingun objeto seleccionado\n");
711       }
712       else
713         printf("\n\nNumero de argumentos incorrecto.Se espaeran 7
    arguemntos.\n");
714                 printf("Uso: luz-on   fila columna Z  R G B\n");
715         printf("Z=coordenada Z de la luz. R,G,B son las componentes
    de color de la luz\n");
716     break;
717     case 17:  //luz-off
718       if (ncomando->num_argumentos==1)
719       {
720         if(seleccion->getNumSelected()!=0)
721         {
```

```
722         int seleccionados=seleccion->getNumSelected();
723         for(int i=0;i<seleccionados;i++)
724           {
725           SoNode *nodoSeleccionado=(SoNode *)  ( (SoPath *)
    (seleccion->getPath(i))->getTail());
726           char *nombretipo= (char *)
    nodoSeleccionado->getName().getString() ;
727           //obtención del nodo padre
728           SoNode *nodoPadre= obtenerPadre( nodoSeleccionado,nraiz);
729           //llamada a la función correspondiente
730           desactivar_LuzPosicional(nodoPadre);
731           printf("\nLuz posicional desactivada en el objeto %s
    .\n",nombretipo);
732           }
733         printf("\n\nHecho.\n");
734         }
735         else
736         printf("\nNingun objeto seleccionado\n");
737       }
738       else
739         printf("No se espera ningún argumento.\nUso: luz-off\n");
740     break;
741
742     case 18: //posicionluz
743
744       if (ncomando->num_argumentos==4)
745       {
746         if(seleccion->getNumSelected()!=0)
747         {
748         int seleccionados=seleccion->getNumSelected();
749         for(int i=0;i<seleccionados;i++)
750           {
751           SoNode *nodoSeleccionado=(SoNode *)  ( (SoPath *)
    (seleccion->getPath(i))->getTail());
752           char *nombretipo= (char *)
    nodoSeleccionado->getName().getString() ;
753           //intentamos obtener el nodo padre del nodo SoShape
    pasado
754           SoNode *nodoPadre= obtenerPadre( nodoSeleccionado,nraiz);
755           //llamamos a la funcion
756           PosicionLuz(   (float) (atof(ncomando->argumento[1])),
757                       (float) (atof(ncomando->argumento[2])),
758                       (float) (atof(ncomando->argumento[3])),
759                   nodoPadre);
760           printf("\nLuz movida en el objeto %s .\n",nombretipo);
761           }
762         printf("\n\nHecho.\n");
763         }
764         else
765         printf("\nNingun objeto seleccionado\n");
766       }
767       else
768         printf("Se esperan 3 argumentos.\nUso: posicionluz X Y Z
    \n");
769      break;
770
771     case 19: //colorluz
772
773       if (ncomando->num_argumentos==4)
774       {
775         if(seleccion->getNumSelected()!=0) //si hay objetos
```

```
    seleccionados
776         {
777         int seleccionados=seleccion->getNumSelected(); //guardamos
    el numero de objetos seleccionados
778         for(int i=0;i<seleccionados;i++) //recorremos cada uno de
    los objetos seleccionados
779           {
780           SoNode *nodoSeleccionado=(SoNode *)  ( (SoPath *)
    (seleccion->getPath(i))->getTail());
781           char *nombretipo= (char *)
    nodoSeleccionado->getName().getString() ;
782           //intentamos obtener el nodo padre del nodo SoShape
    pasado
783           SoNode *nodoPadre= obtenerPadre( nodoSeleccionado,nraiz);
784           ColorLuz( atoi(ncomando->argumento[1]),
785                   atoi(ncomando->argumento[2]),
786                   atoi(ncomando->argumento[3]),nodoPadre);
787           printf("\nColor aplicado a objeto luz: %s
    .\n",nombretipo);
788           }
789         printf("\n\nHecho.\n");
790         }
791         else
792         printf("\nNingun objeto seleccionado\n");
793       }
794       else
795       {   printf("\nNúmero de argumentos incorrecto.Se esperan 3
    argumentos\n");
796       printf("Uso:  colorluz  R  G  B       (formato 0-255)\n"); }
797
798      break;
799     case 20: //luces-off
800       if (ncomando->num_argumentos==1)
801       {   //borramos todos los nodos manipuladores que se
    encuentren dentro del nodo raiz
802         borrar_nodos(nraiz,"Luz");
803         borrar_nodos(nraiz,"LuzManip");
804         printf("\n\nHecho.\n");
805       }
806       else
807         printf("No se espera ningún argumento.\nUso: luces-off\n");
808     break;
809
810     case 21: //crearmalla
811       if (ncomando->num_argumentos==9)
812       {   //borramos todos los nodos manipuladores que se
    encuentren dentro del nodo raiz
813
814         SbColor color;
815         color.setValue(
    atof(ncomando->argumento[4])/255.0,atof(ncomando->argumento[5])/255
    .0,atof(ncomando->argumento[6])/255.0);
816         SoSeparator *pmalla= crear_malla (
    atol(ncomando->argumento[1]),
817
    atol(ncomando->argumento[2]),atof(ncomando->argumento[3]),color,ato
    f(ncomando->argumento[7]),ncomando->argumento[8]); //creamos la
    malla
818         nraiz->addChild(pmalla);
819         ( (SoSeparator *)(nraiz->getByName("malla_grupo"))
    )->ref(); //referenciamos el nuevo obheto
```

```
820         printf("\n\nHecho\n");
821         seleccion->touch(); //forzamos refresco
822       }
823       else
824       {
825         printf("\n\n\nSe esperan 8 parametros.\n");
826         printf("Uso: crearmalla numFilas numColumnas valorZ R G B
    separacion nombremalla\n\n\n");
827       }
828     break;
829
830     case 22: //Comando cortarXfila
831       if (ncomando->num_argumentos==3)
832       {
833         if(seleccion->getNumSelected()!=0)
834         {
835         int seleccionados=seleccion->getNumSelected();
836         for(int i=0;i<seleccionados;i++)
837           {
838           SoPath *pathseleccionado=(SoPath *)
    (seleccion->getPath(i) );
839           SoNode *nodoSeleccionado=(SoNode *)  (
    pathseleccionado->getTail());
840           char *nombretipo= (char *)
    nodoSeleccionado->getName().getString() ;
841           //obtención del nodo padre
842           seleccion->deselect(pathseleccionado);
843           seleccion->touch();
844           SoNode *nodoPadre= obtenerPadre( nodoSeleccionado,nraiz);
845           cortarXfila ( atol(ncomando->argumento[1]),
846                       atoi(ncomando->argumento[2]),
847                  (SoSeparator *)nodoPadre);
848           }
849         printf("\n\nHecho.\n");
850         }
851         else
852         printf("\nNingun objeto seleccionado\n");
853       }
854       else
855       {
856       printf("\nNúmero de argumentos incorrecto.Se esperan 3
    argumentos\n");
857       printf("Uso:  cortarXfila fila  zonaAeliminar\n");
858       }
859     break;
860
861     case 23:  //cortarxcolumna
862       if (ncomando->num_argumentos==3)
863       {
864         if(seleccion->getNumSelected()!=0)
865         {$
866         int seleccionados=seleccion->getNumSelected();
867         for(int i=0;i<seleccionados;i++)
868           {
869           SoPath *pathseleccionado=(SoPath *)
    (seleccion->getPath(i) );
870           SoNode *nodoSeleccionado=(SoNode *)  (
    pathseleccionado->getTail());
871           char *nombretipo= (char *)
    nodoSeleccionado->getName().getString() ;
872           //obtención del nodo padre
```

```
873           seleccion->deselect(pathseleccionado);
874           seleccion->touch();
875           SoNode *nodoPadre= obtenerPadre( nodoSeleccionado,nraiz);
876           cortarXcolumna ( atol(ncomando->argumento[1]),
877                       atoi(ncomando->argumento[2]),
878                   (SoSeparator *)nodoPadre);
879           }
880         printf("\n\nHecho.\n");
881         }
882         else
883         printf("\nNingun objeto seleccionado\n");
884       }
885       else
886       {
887       printf("\nNúmero de argumentos incorrecto.Se esperan 3
    argumentos\n");
888       printf("Uso:  cortarXcolumna columna  zonaAeliminar\n");
889       }
890     break;
891
892     case 24:  //cortarxarea
893       if (ncomando->num_argumentos==5)
894       {
895         if(seleccion->getNumSelected()!=0)
896         {
897         int seleccionados=seleccion->getNumSelected();
898         for(int i=0;i<seleccionados;i++)
899           {
900           SoPath *pathseleccionado=(SoPath *)
    (seleccion->getPath(i) );
901           SoNode *nodoSeleccionado=(SoNode *)  (
    pathseleccionado->getTail());
902           char *nombretipo= (char *)
    nodoSeleccionado->getName().getString() ;
903           //obtención del nodo padre del nodo SoShape pasado
904           seleccion->deselect(pathseleccionado);
905           seleccion->touch();
906           SoNode *nodoPadre= obtenerPadre( nodoSeleccionado,nraiz);
907           cortarXarea (atol(ncomando->argumento[1]),
908                       atol(ncomando->argumento[2]),
909                   atol(ncomando->argumento[3]),
910                   atol(ncomando->argumento[4]),
911                   (SoSeparator *)nodoPadre);
912           }
913         printf("\n\nHecho.\n");
914         }
915         else
916         printf("\nNingun objeto seleccionado\n");
917       }
918       else
919       {
920       printf("\nNumero de argumentos incorrecto.Se esperan 4
    argumentos\n");
921       printf("Uso:  cortarxarea X1 Y1 X2 Y2\n");
922       }
923     break;
924
925     case 25: //infoObjeto
926       if  (ncomando->num_argumentos==8)
927       {
928         if(seleccion->getNumSelected()!=0)
```

```
929         {
930         int seleccionados=seleccion->getNumSelected();
931         for(int i=0;i<seleccionados;i++)
932           {
933           SoPath *pathseleccionado=(SoPath *)
    (seleccion->getPath(i) );
934           SoNode *nodoSeleccionado=(SoNode *)  (
    pathseleccionado->getTail());
935           char *nombretipo= (char *)
    nodoSeleccionado->getName().getString() ;
936           //obtención del nodo padre
937           seleccion->deselect(pathseleccionado);
938           seleccion->touch();
939           SoNode *nodoPadre= obtenerPadre( nodoSeleccionado,nraiz);
940
941             if(ncomando->num_argumentos==8)
942             {
943               //añadimos información al árbol nodal del objeto
    seleccionado
944               if( atoi(ncomando->argumento[7]) != 0)  //en caso de
    querer visulizar la informacion en el visor3d
945               {
946                 infObjeto *nuevaInfo=infoObjeto(
947                 atof(ncomando->argumento[1]),
948
    atof(ncomando->argumento[2]),atof(ncomando->argumento[3]),
949
    atoi(ncomando->argumento[4]),atoi(ncomando->argumento[5]),
950
    atoi(ncomando->argumento[6]),atoi(ncomando->argumento[7]),
951                 (SoNode *)nodoPadre);
952
953                 ((SoSeparator *)nodoPadre )->addChild( (SoNode
    *)(nuevaInfo->nodoInformacion));
954                 ( (SoSeparator *)(nodoPadre->getByName("Info"))
    )->ref();
955               }
956               else
957               {
958                 infoObjeto(atof(ncomando->argumento[1]),
959
    atof(ncomando->argumento[2]),atof(ncomando->argumento[3]),
960
    atoi(ncomando->argumento[4]),atoi(ncomando->argumento[5]),
961
    atoi(ncomando->argumento[6]),atoi(ncomando->argumento[7]),
962                 (SoSeparator *)nodoPadre);
963               }
964             }
965           }
966         printf("\n\nHecho.\n");
967         }
968        else   //si no hay seleccionado ningun objeto
969        printf("\nNingun objeto seleccionado\n");  //informamos
970       }
971       else
972       {
973       printf("\nNumero de argumentos incorrecto.Se esperan 7
    argumentos\n");
974       printf("Uso:  info coordX coordY coord Z compR compG compB
    mostrar\n");
```

```
 975        printf("mostrar=1 incluye la informacion en la ventana del
     visor3d\n");
 976        printf("mostrar=0 solo se muestra en la linea de
     comandos\n");
 977        printf("las componentes de color se indican en formato RGB
     (0-255)\n");
 978        }
 979      break;
 980
 981      case 26: //info-off
 982        if (ncomando->num_argumentos==1)
 983        {
 984          if(seleccion->getNumSelected()!=0)
 985          {
 986          int seleccionados=seleccion->getNumSelected();
 987          for(int i=0;i<seleccionados;i++)
 988            {
 989            SoPath *pathseleccionado=(SoPath *)
     (seleccion->getPath(i) );
 990            SoNode *nodoSeleccionado=(SoNode *)  (
     pathseleccionado->getTail());
 991            char *nombretipo= (char *)
     nodoSeleccionado->getName().getString() ;
 992            //obtención del nodo padre
 993            seleccion->deselect(pathseleccionado);
 994            seleccion->touch();
 995            SoNode *nodoPadre= obtenerPadre( nodoSeleccionado,nraiz);
 996            //borramos nodo de información
 997            borrar_nodos( nodoPadre,"Info");
 998            printf("\n\nOjeto Texto eliminado\n");
 999          }
1000          printf("\n\nHecho.\n");
1001          }
1002          else
1003          printf("\nNingun objeto seleccionado\n");
1004        }
1005        else
1006        {
1007        printf("\nNumero de argumentos incorrecto.No se espera ningun
     paramtero\n");
1008        printf("Uso:  info-off\n");
1009        }
1010      break;
1011      case 27://texturizar
1012        if (ncomando->num_argumentos==2)
1013        {
1014          if(seleccion->getNumSelected()!=0)
1015          {
1016          int seleccionados=seleccion->getNumSelected();
1017          for(int i=0;i<seleccionados;i++)
1018            {
1019            SoPath *pathseleccionado=(SoPath *)
     (seleccion->getPath(i) );
1020            SoNode *nodoSeleccionado=(SoNode *)  (
     pathseleccionado->getTail());
1021            char *nombretipo= (char *)
     nodoSeleccionado->getName().getString() ;
1022            //obtención del nodo padre del nodo SoShape pasado
1023            seleccion->deselect(pathseleccionado);
1024            seleccion->touch();
1025            SoNode *nodoPadre= obtenerPadre( nodoSeleccionado,nraiz);
```

```
1026            //llamamos a la función correspondiente
1027            texturizar( (SoSeparator
      *)nodoPadre,ncomando->argumento[1]);
1028            printf("\n\nOjeto Texturizado\n");
1029          }
1030          printf("\n\nHecho.\n");
1031          }
1032          else
1033          printf("\nNingun objeto seleccionado\n");
1034        }
1035        else
1036        {
1037        printf("\nNumero de argumentos incorrecto.Se espera un
      argumento\n");
1038        printf("Uso: texturizar nombre_imagen\n");
1039        printf("El nombre del archivo debe de ser la direccion
      absoluta\n");
1040        printf("formatos de imagen soportados (.tga), PIC(.pic), SGI
      RGB(.rgb, .bw) XWD (.xwd)\n");
1041          printf("los formatos siguientes estan parcialmente
      soportados: JPEG, GIF, TIFF y PNG\n");
1042        }
1043      break;
1044      case 28: //Filtro
1045        if( (ncomando->num_argumentos==9) ||
      (ncomando->num_argumentos==3) )
1046        {
1047          if(seleccion->getNumSelected()!=0)
1048          {
1049          int seleccionados=seleccion->getNumSelected();
1050          for(int i=0;i<seleccionados;i++)
1051            {
1052            SoPath *pathseleccionado=(SoPath *)
      (seleccion->getPath(i) );
1053            SoNode *nodoSeleccionado=(SoNode *)  (
      pathseleccionado->getTail());
1054            char *nombretipo= (char *)
      nodoSeleccionado->getName().getString() ;
1055            //obtención del nodo padre del objeto seleccionado
1056            SoNode *nodoPadre= obtenerPadre( nodoSeleccionado,nraiz);
1057            //llamamos a la función correspondiente
1058            if(ncomando->num_argumentos==3)//con 3 argumentos
1059            {
1060              SbVec3f color(-1,-1,-1); //se envían valores de color
      nulos(se mantienen los originales)
1061              filtro
      (atof(ncomando->argumento[1]),color,color,atoi(ncomando->argumento[
      2]),(SoSeparator *)nodoPadre);
1062            }
1063            else //con 9 argumentos
1064            {
1065              SbVec3f colorSup( atof(ncomando->argumento[2])/255.0,
1066                        atof(ncomando->argumento[3])/255.0,
1067                        atof(ncomando->argumento[4])/255.0 );
1068              SbVec3f colorInf( atof(ncomando->argumento[5])/255.0,
1069                        atof(ncomando->argumento[6])/255.0,
1070                        atof(ncomando->argumento[7])/255.0 );
1071              filtro
      (atof(ncomando->argumento[1]),colorSup,colorInf,atoi(ncomando->argu
      mento[8]),(SoSeparator *)nodoPadre);
1072            }
```

```
1073          }
1074          printf("\b\n\nHecho.\n");
1075          }
1076          else
1077          printf("\nNingun objeto seleccionado\n");
1078        }
1079        else
1080        {
1081        printf("\nNumero de argumentos incorrecto.Se esperan 6
      argumento\n");
1082        printf("Uso: filtro TempCorte (colorsup) (colorinf) tipo\n");
1083        printf("tipo: 0=pasa bajas, 1=pasa altas\n");
1084        printf("Los argumentos entre parentesis son opcionales.
      Colores en formato RGB \n");
1085        printf("Valores validos para colores 0-255 , para valores
      negativos se utilizara el mismo\n");
1086        printf("En caso de querer usar solo uno de los dos argumentos
      de color, el color que no interese\n");
1087        printf("usar se pasa con valores nulos (valores
      negativos)\n");
1088        }
1089      break;
1090
1091      case 29: //Filtro banda
1092        if( (ncomando->num_argumentos==13) ||
      (ncomando->num_argumentos==4) )
1093        {
1094          if(seleccion->getNumSelected()!=0)
1095          {
1096          int seleccionados=seleccion->getNumSelected();
1097          for(int i=0;i<seleccionados;i++)
1098            {
1099            SoPath *pathseleccionado=(SoPath *)
      (seleccion->getPath(i) );
1100            SoNode *nodoSeleccionado=(SoNode *)  (
      pathseleccionado->getTail());
1101            char *nombretipo= (char *)
      nodoSeleccionado->getName().getString() ;
1102            //obtención del nodo padre
1103            SoNode *nodoPadre= obtenerPadre( nodoSeleccionado,nraiz);
1104            //llamamos a la función correspondiente
1105            if(ncomando->num_argumentos==4)//con 4 argumentos
1106            {
1107              SbVec3f color(-1,-1,-1); //se envían valores de color
      nulos(se mantienen los originales)
1108              filtroBanda
      (atof(ncomando->argumento[1]),atof(ncomando->argumento[2]),color,co
      lor,color,atoi(ncomando->argumento[3]),(SoSeparator *)nodoPadre);
1109            }
1110            else //con 12 argumentos
1111            {
1112              SbVec3f colorSup( atof(ncomando->argumento[3])/255.0,
1113                        atof(ncomando->argumento[4])/255.0,
1114                        atof(ncomando->argumento[5])/255.0 );
1115              SbVec3f colorBanda( atof(ncomando->argumento[6])/255.0,
1116                        atof(ncomando->argumento[7])/255.0,
1117                        atof(ncomando->argumento[8])/255.0 );
1118              SbVec3f colorInf( atof(ncomando->argumento[9])/255.0,
1119                        atof(ncomando->argumento[10])/255.0,
1120                        atof(ncomando->argumento[11])/255.0 );
1121
```

```
      filtroBanda(atof(ncomando->argumento[1]),atof(ncomando->argumento[2
      ]),colorSup,colorBanda,colorInf,atoi(ncomando->argumento[12]),(SoSe
      parator *)nodoPadre);
1122            }
1123          }
1124          printf("\n\nHecho.\n");
1125          }
1126          else
1127          printf("\nNingun objeto seleccionado\n");
1128        }
1129        else
1130        {
1131        printf("\nNumero de argumentos incorrecto.Se esperan 6
      argumento\n");
1132        printf("Uso: filtrobanda TempInf TempSup (colorSup)
      (colorbanda) (colorInf) tipo\n");
1133        printf("tipo: 0=rechaza banda, 1=pasa banda\n");
1134        printf("Los argumentos entre parentesis son opcionales.
      Colores en formato RGB \n");
1135        printf("Valores validos para colores 0-255 , para valores
      negativos se utilizara el mismo\n");
1136        printf("color que la imagen original\n");
1137        }
1138      break;
1139
1140      case 30:  //referencia
1141        //flip-flop
1142        if  (eviewer->isFeedbackVisible()==TRUE ) //para
1143        eviewer->setFeedbackVisibility(FALSE); //muestrar el eje
1144        else
1145        eviewer->setFeedbackVisibility(TRUE);
1146
1147        seleccion->touch();
1148        printf("\n\nHecho.\n");
1149      break;
1150
1151      case 31:  //autoeje
1152        if (ncomando->num_argumentos==1)
1153        {
1154          if(seleccion->getNumSelected()==1)
1155          {
1156            SoPath *pathseleccionado=(SoPath *)
      (seleccion->getPath(0) );
1157            SoNode *nodoSeleccionado=(SoNode *)  (
      pathseleccionado->getTail());
1158            //obtención del nodo padre del objeto seleccionado
1159            seleccion->deselect(pathseleccionado);
1160            seleccion->touch();
1161            SoNode *nodoPadre= obtenerPadre( nodoSeleccionado,nraiz);
1162            //llamada a la función correspondiente
1163            if(nodoPadre!=NULL)
1164            autoEje( (SoSeparator *) nodoPadre );
1165            printf("\n\nHecho.\n");
1166          }
1167          else
1168          printf("\nNingun objeto seleccionado\n");
1169        }
1170        else
1171        {
1172        printf("\nNo se esperan argumentos\n");
1173        printf("Uso: autoeje\n");
```

```
1174       }
1175     break;
1176
1177     case 32:  //seleccionararea
1178      if (ncomando->num_argumentos==1)
1179      {
1180         if(seleccion->getNumSelected()==1)
1181         {
1182           if(capturaPuntos==2)
1183           {
1184           int seleccionados=seleccion->getNumSelected();
1185           SoPath *pathseleccionado=(SoPath *)
     (seleccion->getPath(0) );
1186           SoNode *nodoSeleccionado=(SoNode *)  (
     pathseleccionado->getTail());
1187           char *nombretipo= (char *)
     nodoSeleccionado->getName().getString() ;
1188           //obtención del nodo padre
1189           seleccion->deselect(pathseleccionado);
1190           seleccion->touch();
1191           SoNode *nodoPadre= obtenerPadre( nodoSeleccionado,nraiz);
1192
1193           printf("\n\nHecho.\n");
1194           }
1195           if(capturaPuntos==0) //si se acaba de llamar al comando
1196           {
1197             printf("\nActivada la captura de puntos. La captura de
     teclas queda deshabilitada");
1198             printf("hasta que se hayan capturado todos los puntos
     necesarios\n");
1199             capturaPuntos=1; //se empieza la captura de puntos
     (deshabilitando la captura de eventos de teclado)
1200           }
1201         }
1202         else
1203         printf("\nDebe de haber un objeto seleccionado\n");
1204     }
1205     else
1206     {
1207      printf("\nNo se espera ningun argumento\n");
1208      printf("Uso: captura\n");
1209     }
1210     break;
1211
1212     case 33: //reflejarv
1213       if (ncomando->num_argumentos==1)
1214       {
1215         if(seleccion->getNumSelected()!=0)
1216         {
1217         int seleccionados=seleccion->getNumSelected();
1218         for(int i=0;i<seleccionados;i++)
1219           {
1220           SoPath *pathseleccionado=(SoPath *)
     (seleccion->getPath(i) );
1221           SoNode *nodoSeleccionado=(SoNode *)  (
     pathseleccionado->getTail());
1222           char *nombretipo= (char *)
     nodoSeleccionado->getName().getString() ;
1223           //obtención del nodo padre del objeto seleccionado
1224           seleccion->deselect(pathseleccionado);
1225           seleccion->touch();
```

```
1226            SoNode *nodoPadre= obtenerPadre( nodoSeleccionado,nraiz);
1227            //llamada a la función correspondiente
1228            reflejarV( (SoSeparator *) nodoPadre );
1229            printf("\n\nOjeto reflejado verticalmente\n");
1230          }
1231          printf("\n\nHecho.\n");
1232          }
1233          else
1234          printf("\nNingun objeto seleccionado\n");
1235        }
1236        else
1237        {
1238        printf("\nNo se esperan argumentos\n");
1239        printf("Uso: reflejarv\n");
1240        }
1241      break;
1242
1243      case 34: //reflejarh
1244        if (ncomando->num_argumentos==1)
1245        {
1246          if(seleccion->getNumSelected()!=0)
1247          {
1248          int seleccionados=seleccion->getNumSelected();
1249          for(int i=0;i<seleccionados;i++)
1250            {
1251            SoPath *pathseleccionado=(SoPath *)
     (seleccion->getPath(i) );
1252            SoNode *nodoSeleccionado=(SoNode *)  (
     pathseleccionado->getTail());
1253            char *nombretipo= (char *)
     nodoSeleccionado->getName().getString() ;
1254            //intentamos obtener el nodo padre del nodo SoShape
     pasado
1255            seleccion->deselect(pathseleccionado);
1256            seleccion->touch();
1257            SoNode *nodoPadre= obtenerPadre( nodoSeleccionado,nraiz);
1258            reflejarH ((SoSeparator *) nodoPadre );
1259
1260            printf("\n\nOjeto reflejado horizontalmente\n");
1261          }
1262          printf("\n\nHecho.\n");
1263          }
1264          else
1265          printf("\nNingun objeto seleccionado\n");
1266        }
1267        else
1268        {
1269        printf("\nNo se esperan argumentos\n");
1270        printf("Uso: reflejarh\n");
1271        }
1272      break;
1273
1274      case 35: //invertirc
1275        if (ncomando->num_argumentos==1)
1276        {
1277          if(seleccion->getNumSelected()!=0)
1278          {
1279          int seleccionados=seleccion->getNumSelected();
1280          for(int i=0;i<seleccionados;i++)
1281            {
1282            SoPath *pathseleccionado=(SoPath *)
```

```
      (seleccion->getPath(i) );
 1283           SoNode *nodoSeleccionado=(SoNode *)  (
      pathseleccionado->getTail());
 1284           char *nombretipo= (char *)
      nodoSeleccionado->getName().getString() ;
 1285           //obtención del nodo padre del objeto seleccionado
 1286           seleccion->deselect(pathseleccionado);
 1287           seleccion->touch();
 1288           SoNode *nodoPadre= obtenerPadre( nodoSeleccionado,nraiz);
 1289           invertirColores( (SoSeparator *) nodoPadre);
 1290         }
 1291         printf("\n\nHecho.\n");
 1292         }
 1293         else
 1294         printf("\nNingun objeto seleccionado\n");
 1295       }
 1296       else
 1297       {
 1298       printf("\nNo se esperan argumentos\n");
 1299       printf("Uso: reflejarh\n");
 1300       }
 1301     break;
 1302
 1303     case 36:  //comando cargarmallat
 1304     if(ncomando->num_argumentos==9) //creación de malla en caso de
      bitmaps
 1305     {
 1306
 1307        SoSeparator *pmallaT= generar_mallaT (
      "",atoi(ncomando->argumento[1]),
 1308         atoi(ncomando->argumento[2]),atoi(ncomando->argumento[3]),
 1309         atoi(ncomando->argumento[4]),atof(ncomando->argumento[5]),
 1310
      atof(ncomando->argumento[6]),ncomando->argumento[7],atoi(ncomando->
      argumento[8]));
 1311       //se añade la malla al nodo raíz
 1312       nraiz->addChild(pmallaT);
 1313       ( (SoSeparator *)(nraiz->getByName("mallaT_grupo")) )->ref();
 1314       printf("\n\nHecho\n");
 1315       seleccion->touch();
 1316     }
 1317     else if(ncomando->num_argumentos==8)//creación en caso de
      archivos binarios
 1318     {
 1319        SoSeparator *pmallaT= generar_mallaT (
      ncomando->argumento[1] ,
 1320         atoi(ncomando->argumento[2]),atoi(ncomando->argumento[3]),
 1321         atoi(ncomando->argumento[4]),atoi(ncomando->argumento[5]),
 1322         0,0,ncomando->argumento[6],atoi(ncomando->argumento[7]));
 1323        //se añade la malla al nodo raíz
 1324       nraiz->addChild(pmallaT);
 1325       ( (SoSeparator *)(nraiz->getByName("mallaT_grupo")) )->ref();
 1326       printf("\n\nHecho\n");
 1327       seleccion->touch(); //forzamos refresco
 1328     }
 1329     else //ni 6 ni 7 argumentos
 1330     {
 1331        printf("\nNumero de argumentos incorrecto.Se esperan 8/7
      parametros\n");
 1332        printf("Uso:  cargarmallat usarpaleta compR compG compB Tsup
      Tinf nombre\n");
```

```
1333        printf("Uso:  cargarmallat archivoBinario usarpaleta compR
    compG compB nombre\n");
1334        printf("usarpaleta=1 se usará la paleta contenida en el
    fichero datospal.txt\n");
1335        printf("usarpaleta=0 se usará el color en formato RGB pasado
    como compR compG y compB\n");
1336        printf("Tsup es la temperatura que se asignara al color
    superior de la paleta de colores\n");
1337        printf("Tmin es la temperatura que se asignara al color con
    indice 0 en la paleta de colores\n");
1338        printf("nombre indica el nombre que se asignará a la mallaT
    seleccionado para futuras identificaciones del objeto.\n");
1339        printf("forzar_datos=1 se considera a las imagenes de 24bpp
    como válidas para extraer información de temperatura\n");
1340        return -1;
1341     }
1342     break;
1343
1344     case 37: //comando asignarpaleta
1345       if (
    (ncomando->num_argumentos==2)||(ncomando->num_argumentos==1))
1346       {
1347         if(seleccion->getNumSelected()!=0)
1348         {
1349         int seleccionados=seleccion->getNumSelected();
1350         for(int i=0;i<seleccionados;i++)
1351           {
1352           SoPath *pathseleccionado=(SoPath *)
    (seleccion->getPath(i) );
1353           SoNode *nodoSeleccionado=(SoNode *)  (
    pathseleccionado->getTail());
1354           char *nombretipo= (char *)
    nodoSeleccionado->getName().getString() ;
1355           //obtención del nodo padre
1356           seleccion->deselect(pathseleccionado);
1357           seleccion->touch();
1358           SoNode *nodoPadre= obtenerPadre( nodoSeleccionado,nraiz);
1359           if (ncomando->num_argumentos==2)
1360           asignar_Paleta( ncomando->argumento[1], (SoSeparator *)
    nodoPadre );
1361           else /uso /sin argumentos
1362           asignar_Paleta( "datospal.txt", (SoSeparator *) nodoPadre
    ); //se intenta cargar la paleta por defecto
1363         }
1364         printf("\n\nHecho.\n\n\n");
1365         }
1366         else
1367         printf("\n\nNingun objeto seleccionado\n");
1368       }
1369       else
1370       {
1371       printf("\n\nNumero de argumentos incorrecto\n");
1372       printf("Uso: asignarpaleta archivopaleta\n");
1373       printf("archivopaleta debe de contener el direcotior completo
    incluida extension del archivo\n\n\n");
1374       }
1375     break;
1376
1377     case 38: //comando v2bin
1378       if (ncomando->num_argumentos==2)
1379       {
```

```
1380         if(seleccion->getNumSelected()!=0)
1381         {
1382         int seleccionados=seleccion->getN$umSelected();
1383         for(int i=0;i<seleccionados;i++)
1384           {
1385             SoPath *pathseleccionado=(SoPath *)
     (seleccion->getPath(i) );
1386             SoNode *nodoSeleccionado=(SoNode *)  (
     pathseleccionado->getTail());
1387             char *nombretipo= (char *)
     nodoSeleccionado->getName().getString() ;
1388             //obtención del nodo padre
1389             seleccion->deselect(pathseleccionado);
1390             seleccion->touch();
1391             SoNode *nodoPadre= obtenerPadre(
     nodoSeleccionado,nraiz);
1392             //llamada a la función correspondiente
1393             V2Bin(ncomando->argumento[1], (SoSeparator *)
     nodoPadre);
1394           }
1395         printf("\n\nHecho.\n\n\n");
1396         }
1397         else
1398         printf("\n\nNingun objeto seleccionado\n");
1399       }
1400       else
1401       {
1402       printf("\n\nNumero de argumentos incorrecto\n");
1403       printf("Uso: v2bin nombreFichero\n");
1404       printf("nombreFichero debe de contener el direcotior completo
     incluida extension del archivo\n\n\n");
1405       }
1406     break;
1407
1408     case 39: //promedio
1409       if (ncomando->num_argumentos==1)
1410       {
1411         if(seleccion->getNumSelected()!=0) //si hay objetos
     seleccionados
1412         {
1413         int seleccionados=seleccion->getNumSelected(); //guardamos
     el numero de objetos seleccionados
1414           for(int i=0;i<seleccionados;i++) //recorremos cada uno de
     los objetos seleccionados
1415           {
1416             SoPath *pathseleccionado=(SoPath *)
     (seleccion->getPath(i) );
1417             SoNode *nodoSeleccionado=(SoNode *)  (
     pathseleccionado->getTail());
1418             char *nombretipo= (char *)
     nodoSeleccionado->getName().getString() ;
1419             //intentamos obtener el nodo padre del nodo SoShape
     pasado
1420             SoNode *nodoPadre= obtenerPadre(
     nodoSeleccionado,nraiz);
1421             printf("\n\nValor promedio %.4f\n",promedio(
     (SoSeparator *) nodoPadre));
1422           }
1423         printf("\n\nHecho.\n\n\n");
1424         }
1425         else
```

```
1426          printf("\n\nNingun objeto seleccionado\n");
1427        }
1428        else
1429        {
1430        printf("\n\nNo se esperan argumentos\n");
1431        printf("Uso: promedio\n");
1432        }
1433      break;
1434
1435      case 40: //histograma
1436        if ( (ncomando->num_argumentos==1) ||
      (ncomando->num_argumentos==4) )
1437        {
1438          if(seleccion->getNumSelected()!=0)
1439          {
1440          int seleccionados=seleccion->getNumSelected();
1441          for(int i=0;i<seleccionados;i++)
1442            {
1443            SoPath *pathseleccionado=(SoPath *)
      (seleccion->getPath(i) );
1444            SoNode *nodoSeleccionado=(SoNode *)  (
      pathseleccionado->getTail());
1445            char *nombretipo= (char *)
      nodoSeleccionado->getName().getString() ;
1446            seleccion->deselect(pathseleccionado);
1447            seleccion->touch();
1448            //obtención del nodo padre del objeto seleccionado
1449            SoNode *nodoPadre= obtenerPadre( nodoSeleccionado,nraiz);
1450            SoSeparator *Histograma;
1451            if (ncomando->num_argumentos==1)  //en caso de querer
      crear un histograma con títulos por defecto
1452              Histograma= (cargar_histograma ( (SoSeparator *)
      nodoPadre,NULL,NULL,ncomando->argumento[1]))->histograma;
1453            else
1454              Histograma= (cargar_histograma ( (SoSeparator *)
      nodoPadre,ncomando->argumento[1],ncomando->argumento[2],ncomando->a
      rgumento[3]))->histograma;
1455
1456            if (Histograma!=NULL)
1457            {
1458            Histograma->ref();
1459            ((SoSeparator *)nraiz)->addChild(Histograma);
1460            }
1461          }
1462          printf("\n\nHecho.\n\n\n");
1463          }
1464          else
1465          printf("\n\nNingun objeto seleccionado\n");
1466        }
1467        else
1468        {
1469        printf("\n\nNumero de argumentos incorrecto\n");
1470        printf("Uso: histo  tituloEjeX tituloEjeY  idHistograma\n");
1471        printf("Uso: histo\n");
1472        }
1473      break;
1474      case 41://posicionar
1475       if (ncomando->num_argumentos==4)
1476       {
1477        if(seleccion->getNumSelected()!=0)
1478        {
```

```
1479         int seleccionados=seleccion->getNumSelected();
1480         for(int i=0;i<seleccionados;i++)
1481           {
1482           SoPath *pathseleccionado=(SoPath *)
     (seleccion->getPath(i) );
1483           SoNode *nodoSeleccionado=(SoNode *)  (
     pathseleccionado->getTail());
1484           char *nombretipo= (char *)
     nodoSeleccionado->getTypeId().getName().getString() ;
1485           //obtención del nodo padre
1486           SoNode *nodoPadre= obtenerPadre( nodoSeleccionado,nraiz);
1487           SoNode *nodoTransform=
     seleccionarHijo(nodoPadre,"Transforma");
1488           seleccion->deselect(pathseleccionado);
1489           seleccion->touch();
1490           posicion_Objeto( (float) (atof(ncomando->argumento[1]) ),
1491                           (float) (atof(ncomando->argumento[2])),
1492                             (float) (atof(ncomando->argumento[3])),
1493                    (SoTransform *)nodoTransform );
1494           printf("\nObjeto %s posicionado .\n",nombretipo);
1495           }
1496       printf("\n\nHecho.\n");
1497       }
1498         else
1499       printf("\n\nNada a mover.Ningun objeto seleccionado\n\n");
1500       }
1501       else
1502        printf("\n\n\nNúmero de parámetros incorrecto. Uso: posicion
     X Y Z\n\n\n");
1503     break;
1504
1505     case 42://script
1506      if (ncomando->num_argumentos==3)
1507      {
1508         leer_script(ncomando->argumento[1],
     atoi(ncomando->argumento[2]) , (SoSeparator *)nraiz);
1509         printf("\nHecho\n");
1510      }
1511      else
1512      {
1513        printf("\nNumero de argumentos incorrecto. Se esperan dos
     argumentos.");
1514        printf("\nUso: script nombre_archivo intervalo");
1515        printf("\nel intervalo indica valor en segundos(numero
     entero\n");
1516      }
1517     break;
1518
1519
1520     case 43:  //selobjeto
1521      if (ncomando->num_argumentos==2)
1522      {
1523         SoInfo *inform=(SoInfo *)buscarObjeto(
     nraiz,ncomando->argumento[1]);
1524
1525         if(inform!=NULL)
1526         {
1527         //printf("Seleccionado objeto
     %s:\n",inform->getName().getString() );
1528           seleccion->select( inform);
1529         printf("Hecho\n");
```

```
1530         }
1531         else
1532           printf("\nNo existe ningun objeto con ese nombre\n");
1533        // printf("No se ha encontrado ningun objeto con ese
     nombre\n");
1534      }
1535      else
1536      {
1537        printf("\nNumero de argumentos incorrecto. Se 1
     argumento.");
1538        printf("\nUso: selobjeto nombre_grupo_figura");
1539      }
1540     break;
1541
1542     case 44:  //comando texturizar-off
1543      if (ncomando->num_argumentos==1)
1544      {
1545       if(seleccion->getNumSelected()!=0)
1546       {
1547         int seleccionados=seleccion->getNumSelected();
1548         for(int i=0;i<seleccionados;i++)
1549           {
1550           SoPath *pathseleccionado=(SoPath *)
     (seleccion->getPath(i) );
1551           SoNode *nodoSeleccionado=(SoNode *)  (
     pathseleccionado->getTail());
1552           char *nombretipo= (char *)
     nodoSeleccionado->getTypeId().getName().getString() ;
1553           //obtención del nodo padre seleccionado
1554           SoNode *nodoPadre= obtenerPadre( nodoSeleccionado,nraiz);
1555           SoNode *nodoTransform=
     seleccionarHijo(nodoPadre,"Transforma");
1556           seleccion->deselect(pathseleccionado);
1557           seleccion->touch();
1558           //se borra el nodo de textura
1559           borrar_nodos( (SoNode *)nodoPadre,"Textura");
1560           printf("\nObjeto %s posicionado .\n",nombretipo);
1561           }
1562       printf("\n\nHecho.\n");
1563       }
1564         else
1565       printf("\n\nDebe de haber un objeto seleccionado\n\n");
1566       }
1567       else
1568        printf("\n\n\nNúmero de parámetros incorrecto. Uso: posicion
     X Y Z\n\n\n");
1569
1570       break;
1571
1572     case 45: //v2txt
1573       if (ncomando->num_argumentos==2)
1574       {
1575         if(seleccion->getNumSelected()!=0)
1576         {
1577         int seleccionados=seleccion->getNumSelected();
1578         for(int i=0;i<seleccionados;i++)
1579           {
1580           SoPath *pathseleccionado=(SoPath *)
     (seleccion->getPath(i) );
1581           SoNode *nodoSeleccionado=(SoNode *)  (
     pathseleccionado->getTail());
```

```
1582            char *nombretipo= (char *)
    nodoSeleccionado->getName().getString() ;
1583            //intentamos obtener el nodo padre del nodo SoShape
    pasado
1584            seleccion->deselect(pathseleccionado);
1585            seleccion->touch();
1586            SoNode *nodoPadre= obtenerPadre( nodoSeleccionado,nraiz);
1587            v2txt(ncomando->argumento[1], (SoSeparator *) nodoPadre);
1588
1589          }
1590          printf("\n\nHecho.\n\n\n");
1591          }
1592          else
1593          printf("\n\nNingun objeto seleccionado\n");
1594        }
1595        else
1596        {
1597        printf("\n\nNumero de argumentos incorrecto\n");
1598        printf("Uso: v2txt nombreFichero\n");
1599        printf("nombreFichero debe de contener el direcotior completo
    incluida extension del archivo\n\n\n");
1600        }
1601      break;
1602
1603      case 46:  //h2txt
1604        if (ncomando->num_argumentos==2)
1605        {
1606          if(seleccion->getNumSelected()!=0)
1607          {
1608          int seleccionados=seleccion->getNumSelected()
1609          for(int i=0;i<seleccionados;i++)
1610            {
1611            SoPath *pathseleccionado=(SoPath *)
    (seleccion->getPath(i) );
1612            SoNode *nodoSeleccionado=(SoNode *)  (
    pathseleccionado->getTail());
1613            char *nombretipo= (char *)
    nodoSeleccionado->getName().getString() ;
1614            //obtención del nodo padre
1615            seleccion->deselect(pathseleccionado);
1616            seleccion->touch();
1617            SoNode *nodoPadre= obtenerPadre( nodoSeleccionado,nraiz);
1618            h2txt(ncomando->argumento[1], (SoSeparator *) nodoPadre);
1619            }
1620          printf("\n\nHecho.\n\n\n");
1621          }
1622          else
1623          printf("\n\nNingun objeto seleccionado\n");
1624        }
1625        else
1626        {
1627        printf("\n\nNumero de argumentos incorrecto\n");
1628        printf("Uso: h2txt nombreFichero\n");
1629        printf("nombreFichero debe de contener el direcotorio
    completo incluida extension del archivo\n\n\n");
1630        }
1631      break;
1632
1633      case 47:  //etiqueta
1634        if (ncomando->num_argumentos==10)
1635        {
```

```
1636          colorRGB nColor;
1637          nColor.componenteR=atof(ncomando->argumento[6])/255.0;
1638          nColor.componenteG=atof(ncomando->argumento[7])/255.0;
1639          nColor.componenteB=atof(ncomando->argumento[8])/255.0;
1640
1641          SoSeparator
      *nuevaEtiqueta=etiqueta(ncomando->argumento[1],atof(ncomando->argum
      ento[2]),
1642
      atof(ncomando->argumento[3]),atof(ncomando->argumento[4]),atof(ncom
      ando->argumento[5]),  // X Y Z
1643                  nColor,ncomando->argumento[9]);
1644
1645          nraiz->addChild(nuevaEtiqueta);
1646          printf("\n\nHecho.");
1647        }
1648        else
1649        {
1650          printf("\n\nnumero de argumentos incorrecto.Se esperan 9
      argumentos\n");
1651          printf("\nUso: etiqueta texto_etiqueta size_letra X Y Z R G
      B idEtiqueta\n");
1652        }
1653      break;
1654
1655      case 48: //autoenfoque
1656        eviewer->viewAll(); //llamada al método propio de
      SoWinExaminerViewer
1657      break;
1658
1659      case 49: //isoterma
1660          if (ncomando->num_argumentos==6)
1661        {
1662          if(seleccion->getNumSelected()!=0)
1663          {
1664          int seleccionados=seleccion->getNumSelected();
1665          for(int i=0;i<seleccionados;i++)
1666            {
1667            SoPath *pathseleccionado=(SoPath *)
      (seleccion->getPath(i) );
1668            SoNode *nodoSeleccionado=(SoNode *)  (
      pathseleccionado->getTail());
1669            char *nombretipo= (char *)
      nodoSeleccionado->getName().getString() ;
1670            seleccion->deselect(pathseleccionado);
1671            seleccion->touch();
1672            //se obtiene el nodo padre seleccionado
1673            SoNode *nodoPadre= obtenerPadre( nodoSeleccionado,nraiz);
1674            colorRGB ncolor;
1675            ncolor.componenteR=atof(ncomando->argumento[3]);
1676            ncolor.componenteG=atof(ncomando->argumento[4]);
1677            ncolor.componenteB=atof(ncomando->argumento[5]);
1678            //se llama a la función correspondiente
1679
      isoterma(atof(ncomando->argumento[1]),atof(ncomando->argumento[2]),
      ncolor,(SoSeparator *)nodoPadre);
1680          }
1681          printf("\n\nHecho.\n\n\n");
1682          }
1683          else
1684          printf("\n\nNingun objeto seleccionado\n");
```

```
1685        }
1686        else
1687        {
1688        printf("\n\nNumero de argumentos incorrecto\n");
1689        printf("Uso: isoterma Tinicial Tfinal colorFranja\n");
1690        }
1691      break;
1692
1693      case 50: //poscamara
1694        //simple reposicionamiento de la cámara incluida en el visor
      3D
1695        posCamara(
      atof(ncomando->argumento[1]),atof(ncomando->argumento[2]),
1696                        atof(ncomando->argumento[3]) );
1697        printf("\nHecho.\n");
1698      break;
1699
1700      case 51: //transcamara
1701        //translación de la cámara
1702        transCamara(
      atof(ncomando->argumento[1]),atof(ncomando->argumento[2]),
1703                        atof(ncomando->argumento[3]) );
1704        printf("\nHecho.\n");
1705        seleccion->touch();
1706      break;
1707
1708      case 52:
1709        //orientación
1710        orCamara(
      atof(ncomando->argumento[1]),atof(ncomando->argumento[2]),
1711
      atof(ncomando->argumento[3]),atof(ncomando->argumento[4]) );
1712        printf("\nHecho.\n");
1713      break;
1714    }
1715    return(0);
1716  }
1717
1718  /*****************************************************************
      *****/
1719  /* La función realiza un corte a la malla utilizando el parámetro
      fila */
1720  /* que indica a partir de que fila se corta
      */
1721  /* zona=0 indica que se elimina la parte superior y zona=1 la
      inferior */
1722  /* Para ello se crean nuevos vectores para contener solo la parte
      de   */
1723  /* datos necesaria (tanto para las coordenadas como para los
      colores   */
1724  /* de la malla) que sustituyen a los originales del objeto
      seleccionado*/
1725  /*****************************************************************
      *****/
1726  void cortarXfila (long fila,int zona,SoSeparator *grupo_figura)
1727  {
1728    int numColumnas,numFilas;  //uso general
1729    SoQuadMesh *newMalla;  //en caso de malla
1730    SoInfo *pInfo;       //en caso de nube de puntos
1731    SoPointSet *newNube;
1732
```

```
1733    char *nombre= (char *) (grupo_figura->getName().getString());
1734
1735    if(strcmp(nombre,"malla_grupo")==0)
1736    {
1737    //obtención del nodo SoQuadMesh
1738      SoQuadMesh *oldMalla=(SoQuadMesh
      *)(seleccionarHijo(grupo_figura,"malla")); //y los datos necesarios
1739    if(oldMalla!=NULL)
1740      {numColumnas=oldMalla->verticesPerRow.getValue(); //nº de
      columnas
1741      numFilas=oldMalla->verticesPerColumn.getValue();}  //y filas
1742    }
1743    else if(strcmp(nombre,"nube_grupo")==0) //en caso de nube de
      puntos
1744    {    //se obtiene el número de filas y columnas accediendo a los
      nodos de información
1745    pInfo=(SoInfo *)(seleccionarHijo(grupo_figura,"numColumnas"));
1746    numColumnas=atoi( pInfo->string.getValue().getString() );
1747
1748    pInfo=(SoInfo *)(seleccionarHijo(grupo_figura,"numFilas"));
1749    numFilas=atoi( pInfo->string.getValue().getString() );
1750    }
1751    else
1752    {
1753      printf("El elemento seleccionado no es una malla o nube de
      puntos. No se puede cortar.\n");
1754      return; //Fin de la función
1755    }
1756    //detección de parámetro fila superior al valor permitido
1757    if(fila>numFilas)
1758    {
1759      printf("Imposible.La malla/nube contiene %ld
      filas\n",numFilas);
1760     return;
1761    }
1762    //creamos nuevo nodo para contener los puntos del área de la
      malla que no
1763    //eliminaremos
1764    SbVec3f *puntos;
1765    SbColor *indicepaleta;  //Para contener el índice de los colores
      de la
1766                            //zona de la malla que no se elimina
1767
1768    if(zona==0)
1769    {
1770      puntos=new SbVec3f[(fila+1)*numColumnas]; //creamos un array
      con el espacio justo
1771      indicepaleta=new SbColor[(fila+1)*numColumnas];
1772    }
1773    else
1774    {
1775    puntos=new SbVec3f[numFilas*numColumnas - (fila-1)*numColumnas];
1776    indicepaleta=new SbColor[numFilas*numColumnas -
      (fila-1)*numColumnas];
1777    }
1778
1779    //obtenemos los puntos de la malla/nube  antigua
1780    SoCoordinate3 *oldCoordenadas=(SoCoordinate3 *)
      (seleccionarHijo(grupo_figura,"Coordenadas"));
1781    SoMaterial *oldMaterial=(SoMaterial *)
      (seleccionarHijo(grupo_figura,"material"));
```

```
1782    if(zona==0) //segun la zona a eliminar
1783    {
1784      for (long i=0;i< (fila+1)*numColumnas ;i++)
1785      {
1786        puntos[i][0]=oldCoordenadas->point[i][0];
1787        puntos[i][1]=oldCoordenadas->point[i][1];
1788        puntos[i][2]=oldCoordenadas->point[i][2];
1789        //y copiamos también los valores de color correspondiente
1790        indicepaleta[i]=oldMaterial->diffuseColor[i];
1791      }
1792    }
1793    else
1794    {
1795      for (long i=(fila-1)*numColumnas;i< numFilas*numColumnas;i++)
1796      {
1797
       puntos[i-(fila-1)*numColumnas][0]=oldCoordenadas->point[i][0];
1798
       puntos[i-(fila-1)*numColumnas][1]=oldCoordenadas->point[i][1];
1799
       puntos[i-(fila-1)*numColumnas][2]=oldCoordenadas->point[i][2];
1800        //y copiamos también los valores de color correspondiente
1801
       indicepaleta[i-(fila-1)*numColumnas]=oldMaterial->diffuseColor[i];
1802      }
1803      //y copiamos los valores de la zona de la malla por debajo de
       la
1804    }
1805
1806    //nodos que sustituiran a los originales
1807    SoCoordinate3 *newCoordenadas=new SoCoordinate3;
1808    SoMaterial *newMaterial=new SoMaterial;
1809    if(zona==0)
1810    {
1811    newCoordenadas->point.setValues(0,((fila+1)*numColumnas),puntos);
1812    newMaterial->diffuseColor.setValues(0,((fila+1)*numColumnas),indi
       cepaleta);
1813    }
1814    else
1815    {
1816    newCoordenadas->point.setValues(0,numFilas*numColumnas -
       (fila-1)*numColumnas,puntos);
1817    newMaterial->diffuseColor.setValues(0,numFilas*numColumnas -
       (fila-1)*numColumnas,indicepaleta);
1818    }
1819
1820    //en caso de malla_grupo
1821    if(strcmp(nombre,"malla_grupo")==0)
1822    {
1823      newMalla=new SoQuadMesh;
1824      newMalla->verticesPerRow.setValue(numColumnas);
1825      //segun la zona a cortar
1826      if(zona==0)  //zona inferior
1827      newMalla->verticesPerColumn.setValue(fila);
1828      else  //zona superior
1829      newMalla->verticesPerColumn.setValue(numFilas-fila+1);
1830      newMalla->setName("malla");
1831      newCoordenadas->setName("Coordenadas");
1832      newMaterial->setName("material");
1833      grupo_figura->replaceChild(oldCoordenadas,newCoordenadas);
1834      grupo_figura->replaceChild(oldMaterial,newMaterial);
```

```
1835          SoQuadMesh *oldMalla=(SoQuadMesh
      *)(seleccionarHijo(grupo_figura,"malla")); //obtención nodo malla
      antiguo
1836      grupo_figura->replaceChild(oldMalla,newMalla);
1837      //sustituimos las coordenadas anteriores por las nuevas
1838    }
1839    else if ( strcmp(nombre,"nube_grupo")==0 ) //si se trata de una
      nube de puntos
1840    {
1841      SbString *pcadena=new SbString;
1842      pInfo=(SoInfo *)(seleccionarHijo(grupo_figura,"numFilas"));
1843      newNube=new SoPointSet;
1844      newNube->setName("nube");
1845
1846      if(zona==0)  //zona inferior
1847        {
1848        pcadena->sprintf("%d",fila);  //se convierte el número de
      columnas nuevas a cadena de caracteres
1849        pInfo->string=pcadena->getString(); //y se guarda en el nodo
      de información
1850        //indicación del nuevo número de puntos de la nube
1851        newNube->numPoints.setValue(fila*numColumnas);
1852        }
1853      else  //zona superior
1854        {
1855        pcadena->sprintf("%d",numFilas-fila+1);//se convierte el
      núumero de columnas nuevas a cadena de carácteres
1856        pInfo->string=pcadena->getString();
1857        //indicación del nuevo número de puntos de la nube
1858        newNube->numPoints.setValue( (numFilas-fila+1)*numColumnas);
1859        }
1860      newCoordenadas->setName("Coordenadas");
1861      newMaterial->setName("material");
1862      grupo_figura->replaceChild(oldCoordenadas,newCoordenadas);
1863      grupo_figura->replaceChild(oldMaterial,newMaterial);
1864      //se obtiene el nodo de nube de puntos antiguo
1865      SoPointSet *oldNube=(SoPointSet
      *)(seleccionarHijo(grupo_figura,"nube"));
1866      grupo_figura->replaceChild(oldNube,newNube);
1867    }
1868  }
1869
1870  /*****************************************************************
      *****/
1871  /*   Corta el objeto seleccionado por la columna indicada,
      eliminando  */
1872  /*   la zona especificada mediante el argumento zona
      */
1873  /*****************************************************************
      *****/
1874  void cortarXcolumna (long columna,int zona,SoSeparator
      *grupo_figura)
1875  {
1876    int numColumnas,numFilas;  //uso general
1877    SoQuadMesh *newMalla;  //usado en caso de malla
1878    SoInfo *pInfo;       //para nube de puntos
1879    SoPointSet *newNube; //"      "" ""        ""
1880
1881    char *nombre= (char *) (grupo_figura->getName().getString());
1882    //  printf("\n\nNombre del objeto al que se intenta aplicar el
      comando cortarxarea es :%s\n\n",nombre);
```

```
1883    if(strcmp(nombre,"malla_grupo")==0)
1884    {
1885
1886      SoQuadMesh *oldMalla=(SoQuadMesh
      *)(seleccionarHijo(grupo_figura,"malla")); //y los datos necesarios
1887    if(oldMalla!=NULL)
1888      {numColumnas=oldMalla->verticesPerRow.getValue(); //nº de
      columnas
1889      numFilas=oldMalla->verticesPerColumn.getValue();}  //y filas
1890    }
1891    else if(
      (strcmp(nombre,"mallaT_grupo")==0)||(strcmp(nombre,"nube_grupo")==0
      ) )
1892    {
1893    pInfo=(SoInfo *)(seleccionarHijo(grupo_figura,"numColumnas"));
1894    numColumnas=atoi( pInfo->string.getValue().getString() );
1895
1896    pInfo=(SoInfo *)(seleccionarHijo(grupo_figura,"numFilas"));
1897    numFilas=atoi( pInfo->string.getValue().getString() );
1898    }
1899    else
1900    {
1901      printf("El elemento seleccionado no es una malla o nube de
      puntos. No se puede cortar.\n");
1902      return; //Fin de la función
1903    }
1904
1905    //detección de parámetro fila superior al valor permitido
1906    if(columna>numColumnas)
1907    {
1908      printf("Imposible.La malla/nube contiene %ld
      columnas\n",numColumnas);
1909     return;
1910  }
1911    //nodo para contener datos de la zona a NO eliminar de la malla
1912    SbVec3f *puntos;
1913    SbColor *indicepaleta;  //nuevo array para contener el índice de
      los colores de la
1914                            //zona de la malla que no se elimina
1915    //creamos vector para contener los vértices/puntos de la
      malla/nube
1916    if(zona==0)
1917    {
1918      puntos=new SbVec3f[columna*numFilas];
1919      indicepaleta=new SbColor[columna*numFilas];
1920    }
1921    else
1922    {
1923      puntos=new SbVec3f[(numColumnas-columna+1)*numFilas];
1924      indicepaleta=new SbColor[(numColumnas-columna+1)*numFilas];
1925    }
1926
1927    //obtención de los vértices/puntos de la malla/nube original
1928    SoCoordinate3 *oldCoordenadas=(SoCoordinate3 *)
      (seleccionarHijo(grupo_figura,"Coordenadas"));
1929    SoMaterial *oldMaterial=(SoMaterial *)
      (seleccionarHijo(grupo_figura,"material"));
1930    if(zona==0) //segun zona a eliminar
1931    {
1932      for (long i=0;i< numFilas ;i++)
1933      {
```

```
1934        for(long j=0; j< columna;j++)
1935        {
1936
    puntos[i*columna+j][0]=oldCoordenadas->point[i*columna+j+(numColumn
    as-columna)*i][0];
1937
    puntos[i*columna+j][1]=oldCoordenadas->point[i*columna+j+(numColumn
    as-columna)*i][1];
1938
    puntos[i*columna+j][2]=oldCoordenadas->point[i*columna+j+(numColumn
    as-columna)*i][2];
1939
1940        //y copiamos también los valores de color correspondiente
1941
    indicepaleta[i*columna+j]=oldMaterial->diffuseColor[i*columna+j+(nu
    mColumnas-columna)*i];
1942        }
1943      }
1944    }
1945    else
1946    {
1947      long salto=columna-1;
1948      for (long i=0;i< numFilas ;i++)
1949      {
1950        for(long j=0; j< (numColumnas-columna+1);j++)
1951        {
1952
    puntos[i*(numColumnas-columna+1)+j][0]=oldCoordenadas->point[i*(num
    Columnas-columna+1)+j+(i+1)*salto][0];
1953
    puntos[i*(numColumnas-columna+1)+j][1]=oldCoordenadas->point[i*(num
    Columnas-columna+1)+j+(i+1)*salto][1];
1954
    puntos[i*(numColumnas-columna+1)+j][2]=oldCoordenadas->point[i*(num
    Columnas-columna+1)+j+(i+1)*salto][2];
1955        //y copiamos también los valores de color correspondiente
1956
    indicepaleta[i*(numColumnas-columna+1)+j]=oldMaterial->diffuseColor
    [i*(numColumnas-columna+1)+j+(i+1)*salto];
1957        }
1958      }
1959    }
1960
1961    //creación de los nodos para cambiarlos por los originales
1962    SoCoordinate3 *newCoordenadas=new SoCoordinate3;
1963    SoMaterial *newMaterial=new SoMaterial;
1964    if(zona==0)
1965    {
1966    newCoordenadas->point.setValues(0,columna*numFilas,puntos);
1967    newMaterial->diffuseColor.setValues(0,columna*numFilas,indicepale
    ta);
1968    }
1969    else
1970    {
1971    newCoordenadas->point.setValues(0,(numColumnas-columna+1)*numFila
    s,puntos);
1972    newMaterial->diffuseColor.setValues(0,(numColumnas-columna+1)*num
    Filas,indicepaleta);
1973    }
1974
1975    //creación del nodo para la malla
```

```
1976    if(strcmp(nombre,"malla_grupo")==0)
1977    {
1978      newMalla=new SoQuadMesh;
1979      newMalla->verticesPerColumn.setValue(numFilas);
1980      //segun la zona a cortar:
1981      if(zona==0)  //zona inferior
1982      newMalla->verticesPerRow.setValue(columna);
1983      else  //zona superior
1984      newMalla->verticesPerRow.setValue(numColumnas-columna+1);
1985
1986      newMalla->setName("malla");
1987      newCoordenadas->setName("Coordenadas");
1988      newMaterial->setName("material");
1989      grupo_figura->replaceChild(oldCoordenadas,newCoordenadas);
1990      grupo_figura->replaceChild(oldMaterial,newMaterial);
1991      SoQuadMesh *oldMalla=(SoQuadMesh
      *)(seleccionarHijo(grupo_figura,"malla")); //obtención nodo malla
      antiguo
1992      grupo_figura->replaceChild(oldMalla,newMalla);
1993      //se cambiam las coordenadas anteriores por las nuevas
1994    }
1995    else if ( strcmp(nombre,"nube_grupo")==0 ) //si se trata de una
      nube de puntos
1996    {
1997      SbString *pcadena=new SbString;
1998      pInfo=(SoInfo *)(seleccionarHijo(grupo_figura,"numColumnas"));
1999      newNube=new SoPointSet;
2000      newNube->setName("nube");
2001
2002      if(zona==0)  //zona inferior
2003        {
2004        pcadena->sprintf("%d",columna);
2005        pInfo->string=pcadena->getString();
2006        //se indica el nuevo número de puntos de la nube
2007        newNube->numPoints.setValue(columna*numFilas);
2008        }
2009      else  //zona superior
2010        {
2011        pcadena->sprintf("%d",(numColumnas-columna+1));//conversión
      del número de columnas nuevas a cadena de caracteres
2012        pInfo->string=pcadena->getString();
2013        //se índica el nuevo número de puntos de la nube
2014        newNube->numPoints.setValue(
      (numColumnas-columna+1)*numFilas);
2015        }
2016      newCoordenadas->setName("Coordenadas");
2017      newMaterial->setName("material");
2018      grupo_figura->replaceChild(oldCoordenadas,newCoordenadas);
2019      grupo_figura->replaceChild(oldMaterial,newMaterial);
2020      //se obtiene el nodo de nube de puntos antiguo
2021      SoPointSet *oldNube=(SoPointSet
      *)(seleccionarHijo(grupo_figura,"nube"));
2022      grupo_figura->replaceChild(oldNube,newNube);
2023    }
2024  }
2025
2026  /*************************************************************
      *****/
2027  /*  Corta un zona rectangular definida por los argumentos
      X1,Y1-X2,Y2  */
2028  /*  del objeto seleccionado.
```

```
     */
2029 /*****************************************************************
     *****/
2030
2031 int cortarXarea (long X1,long Y1,long X2,long Y2,SoSeparator
     *grupo_figura)
2032 {
2033   //detección de valores no válidos
2034   if(Y1>Y2)
2035   {
2036     cortarXfila(Y1,0,grupo_figura);
2037     cortarXfila(Y2,1,grupo_figura);
2038   }
2039   else
2040   {
2041     cortarXfila(Y2,0,grupo_figura);
2042     cortarXfila(Y1,1,grupo_figura);
2043   }
2044   if(X2>X1)
2045   {
2046     cortarXcolumna(X2,0,grupo_figura);
2047     cortarXcolumna(X1,1,grupo_figura);
2048   }
2049   else
2050   {
2051     cortarXcolumna(X1,0,grupo_figura);
2052     cortarXcolumna(X2,1,grupo_figura);
2053   }
2054  return(0);
2055 }
2056
2057 /*****************************************************************
     *****/
2058 /*  Activa o desactiva un manipulador de tipo Trackball
     */
2059 /*****************************************************************
     *****/
2060
2061 void actdes_trackball(SoNode *nodoPadre)
2062 {
2063     SoNode *oldnode=seleccionarHijo(nodoPadre,"Transforma");
2064   if (oldnode==NULL) //en caso de no existir un nodo de tipo
     SoTransform si no existe ningun nodo Transforma o SoTrackballManip
2065     return;  //exit
2066
2067   //detección del tipo de nodo
2068   if (oldnode->getTypeId() == SoTransform::getClassTypeId() )
     //en caso de SoTransform
2069   {
2070     //creación de un nodo manipulador de tipo Trackball
2071     SoTrackballManip *newnode=new SoTrackballManip;
2072     //se guardan datos de posición, centro, etc.
2073     newnode->translation.setValue( ( (SoTransform *)
     oldnode)->translation.getValue() );
2074       newnode->rotation.setValue( ( (SoTransform *)
     oldnode)->rotation.getValue() );   //valor de rotación
2075     newnode->scaleFactor.setValue( ( (SoTransform *)
     oldnode)->scaleFactor.getValue() );  // factor de escala
2076     newnode->center.setValue( ( (SoTransform *)
     oldnode)->center.getValue() );  // centro del objeto
2077     newnode->scaleOrientation.setValue( ( (SoTransform *)
```

```
      oldnode)->scaleOrientation.getValue() );  //esc orientación
 2078
 2079     //se establece el mismo nombre para el nodo que el original
 2080     newnode->setName("Transforma");
 2081     //finalmente se sustituye el nodo SoTransform por el
      SoTrckballManip
 2082     ((SoSeparator *)nodoPadre )->removeChild(oldnode);
 2083     ((SoSeparator *)nodoPadre )->insertChild(newnode,0);
      //inserción del nodo en la posición correspondiente
 2084   }
 2085 }
 2086
 2087
 2088 /*****************************************************************
      *****/
 2089 /*  Activa/desactiva un nodo manipulador de tipo Box
      */
 2090 /*  en el objeto seleccionado. Si el objeto seleccionado no
      contiene   */
 2091 /*  ningún nodo del tipo SoTransform o SoTransformBoxManip
      */
 2092 /*  sale de la función. Si encuentra un nodo SoTransform lo
      sustituye  */
 2093 /*  por un nodo SoTransformBoxManip y le pone el mismo nombre,
      */
 2094 /*  es decir, "Transforma". Si encuentra un nodo de tipo
      */
 2095 /*  SoTransformBoxManip hace la inversa, lo sustituye por un nodo
      */
 2096 /*  de tipo SoTransform y le pone el mismo nombre "Transforma"
      */
 2097 /*  Antes de realizar los intercambios de nodos
      */
 2098 /*  se copia campo por campo para que las propiedades del objeto,
      */
 2099 /*  como son posición,rotación,etc no se modifiquen y el objeto
      */
 2100 /*  permanezca en el mismo lugar que estaba antes de llamar a la
      */
 2101 /*  función
      */
 2102 /*****************************************************************
      *****/
 2103 void actdes_box(SoNode *nodoPadre)
 2104 {
 2105     SoNode *oldnode=seleccionarHijo(nodoPadre,"Transforma");
 2106   if (oldnode==NULL) //si no existe ningún nodo Transforma (ya sea
      SoTrasnform o SoTransformBoxManip)
 2107     return;  //fin de la función
 2108
 2109   /identificación del tipo de nodo
 2110   if (oldnode->getTypeId() == SoTransform::getClassTypeId() )
      //si el nodo encontrado es un SoTransform
 2111   {
 2112     //creamos un nodo manipulador de tipo Box.
 2113     SoTransformBoxManip *newnode=new SoTransformBoxManip;
 2114     // se guardan todos los datos de posición,rotación, escalado,
      etc del nodo original
 2115     newnode->translation.setValue( ( (SoTransform *)
      oldnode)->translation.getValue() );
 2116     newnode->rotation.setValue( ( (SoTransform *)
```

```
      oldnode)->rotation.getValue() );   /
 2117     newnode->scaleFactor.setValue( ( (SoTransform *)
      oldnode)->scaleFactor.getValue() );
 2118     newnode->center.setValue( ( (SoTransform *)
      oldnode)->center.getValue() );
 2119     newnode->scaleOrientation.setValue( ( (SoTransform *)
      oldnode)->scaleOrientation.getValue() );
 2120
 2121     //estblecimiento del mismo nombre que el nodo original
 2122     newnode->setName("Transforma");
 2123     //finalmente se sustituye el nodo SoTransform por el
      SoTransformBoxManip
 2124     ((SoSeparator *)nodoPadre )->removeChild(oldnode);
 2125     ((SoSeparator *)nodoPadre )->insertChild(newnode,0);
      //insertándolo en la posición inicial de la jerarquía de nodos
      hijos
 2126   }
 2127
 2128   else if (oldnode->getTypeId() ==
      SoTransformBoxManip::getClassTypeId() ) //si el nodo existente es
      del tipo manipulador box->procedemos a desactivarlo
 2129   {
 2130     //creamos un nodo manipulador de tipo Transform.
 2131     SoTransform *newnode=new SoTransform;
 2132     //se guardan los datos de posición,rotación,etc del nodo
      SoTransformBoxManip en el nodo SoTransform
 2133     newnode->translation.setValue( ( (SoTransformBoxManip *)
      oldnode)->translation.getValue() );
 2134     newnode->rotation.setValue( ( (SoTransformBoxManip *)
      oldnode)->rotation.getValue() );
 2135     newnode->scaleFactor.setValue( ( (SoTransformBoxManip *)
      oldnode)->scaleFactor.getValue() );
 2136     newnode->center.setValue( ( (SoTransformBoxManip *)
      oldnode)->center.getValue() );
 2137     newnode->scaleOrientation.setValue( ( (SoTransformBoxManip *)
      oldnode)->scaleOrientation.getValue() );
 2138
 2139     //se establece el nombre
 2140     newnode->setName("Transforma");
 2141     //finalmente se sustituye el nodo SoTransformBoxManip por el
      SoTransform
 2142     ((SoSeparator *)nodoPadre )->removeChild(oldnode);
 2143     ((SoSeparator *)nodoPadre )->insertChild(newnode,0);
      //insertándolo como primer nodo hijo
 2144   }
 2145 }
 2146
 2147 /*****************************************************************
      *****/
 2148 /*  esta funcion recorre todos los nodos hijos del nodo raiz pasado
      */
 2149 /*  y si encuentra nodos de tipo SoTransformBoxManip los sustituye
      por */
 2150 /*  nodos de tipo SoTransform guardando previamente el estado de
      los   */
 2151 /*  campos. Por tanto esta funcion no necesota que haya ningun
      objeto  */
 2152 /*  seleccionado. Su función es igual a la de
      desactivar_manipuladores */
 2153 /*  pero en este caso solo desactiva los manipuladores de tipo Box.
      */
```

```
2154 /*  Devuelve un entero indicando el número de manipuladores Box
     */
2155 /*  eliminados
     */
2156 /*****************************************************************
     ****/
2157
2158 int desactivar_boxs(SoNode *nraiz)
2159 {
2160   int manipuladores_borrados=0;
2161   //se busca la existencia de algún nodo manipulador
2162   SoSearchAction Nbusqueda;
2163   Nbusqueda.setName("Transforma");  //nodo a buscar
2164   Nbusqueda.setInterest(SoSearchAction::ALL);
2165     Nbusqueda.apply(nraiz);  //nodo en el que buscar
2166     SoPathList ppaths = Nbusqueda.getPaths();
2167
2168   if (ppaths!=NULL)  // si se ha encontrado alguno
2169   {
2170     for(int i=0;i<(ppaths.getLength() );i++)
2171     {
2172       SoPath *ppath=ppaths[i];  //de cada camino (path)
2173       //obtenemos el nodo encontrado
2174       SoNode *oldnode=(SoNode *) ppath->getNodeFromTail(0);
2175
2176       //y comprobamos su tipo
2177           if ( oldnode->getTypeId() ==
     SoTransformBoxManip::getClassTypeId() )  //si el nodo encontrado es
     del tipo box
2178       {
2179         //se obtiene el nodo padre
2180         SoNode *nodoPadre=obtenerPadre(oldnode,nraiz);
2181         //creación del nodo para sustituirlos
2182         SoTransform *newnode=new SoTransform;
2183         //y copiamos los valores del oldnode al newnode
2184         newnode->translation.setValue( ( (SoTransform *)
     oldnode)->translation.getValue() );
2185         newnode->rotation.setValue( ( (SoTransform *)
     oldnode)->rotation.getValue() );
2186         newnode->scaleFactor.setValue( ( (SoTransform *)
     oldnode)->scaleFactor.getValue() );
2187         newnode->center.setValue( ( (SoTransform *)
     oldnode)->center.getValue() );
2188         newnode->scaleOrientation.setValue( ( (SoTransform *)
     oldnode)->scaleOrientation.getValue() );
2189         ppath->replaceIndex(nodoPadre,0,newnode);
2190         ((SoSeparator *)nodoPadre )->replaceChild(oldnode,newnode);
2191         //se establece el mismo nombre que el original
2192         newnode->setName("Transforma");
2193         manipuladores_borrados++; //cuenta de nodos manipuladores
     "desactivados"
2194       }
2195     }
2196   }
2197   return(manipuladores_borrados);
2198 }
2199
2200 /*****************************************************************
     *****/
2201 /*  esta funcion recorre todos los nodos hijos del nodo raiz pasado
     */
```

```
2202 /*  y si encuentra nodos de tipo SoTrackballManip    los sustituye
     por */
2203 /*  nodos de tipo SoTransform guardando previamente el estado de
     los   */
2204 /*  campos. Por tanto esta funcion no necesota que haya ningun
     objeto  */
2205 /*  seleccionado. Su función es igual a la de
     desactivar_manipuladores */
2206 /*  pero en este caso solo desactiva los manipuladores tipo
     trackball  */
2207 /*  Devuelve un entero indicando el número de manipuladores
     trackball  */
2208 /*  eliminados.
     */
2209 /*****************************************************************
     *****/
2210
2211 int desactivar_trackballs(SoNode *nraiz)
2212 {
2213   int manipuladores_borrados=0;
2214   //busqueda de algún nodo manipulador
2215   SoSearchAction Nbusqueda;
2216   Nbusqueda.setName("Transforma");
2217   Nbusqueda.setInterest(SoSearchAction::ALL);
2218     Nbusqueda.apply(nraiz);
2219     SoPathList ppaths = Nbusqueda.getPaths();
2220
2221   if (ppaths!=NULL)  // si se ha encontrado alguno
2222   {
2223     for(int i=0;i<(ppaths.getLength() );i++) //se recorren todos
     los paths
2224     {
2225       SoPath *ppath=ppaths[i];
2226       //obteniendo el nodo encontrado
2227       SoNode *oldnode=(SoNode *) ppath->getNodeFromTail(0);
2228
2229       //se comprueba su tipo
2230           if ( oldnode->getTypeId() ==
     SoTrackballManip::getClassTypeId() )  //si es del tipo trackball
2231       {
2232         //se obtiene el nodo padre
2233         SoNode *nodoPadre=obtenerPadre(oldnode,nraiz);
2234         //creación del nodo para realizar el cambio
2235         SoTransform *newnode=new SoTransform;
2236         //y copiamos los valores del oldnode al newnode
2237         newnode->translation.setValue( ( (SoTransform *)
     oldnode)->translation.getValue() );
2238         newnode->rotation.setValue( ( (SoTransform *)
     oldnode)->rotation.getValue() );
2239         newnode->scaleFactor.setValue( ( (SoTransform *)
     oldnode)->scaleFactor.getValue() );
2240         newnode->center.setValue( ( (SoTransform *)
     oldnode)->center.getValue() );
2241         newnode->scaleOrientation.setValue( ( (SoTransform *)
     oldnode)->scaleOrientation.getValue() );
2242         //y reemplazamos el oldnode por el newnode
2243         ppath->replaceIndex(nodoPadre,0,newnode);
2244         ((SoSeparator *)nodoPadre )->replaceChild(oldnode,newnode);
2245         //se establece el nombre del nodo como el original
2246         newnode->setName("Transforma");
2247         manipuladores_borrados++; //incremento de nodos
```

```
      manipuladores "desactivados"
2248        }
2249      }//cierre for
2250    }
2251    return(manipuladores_borrados);
2252  }
2253
2254
2255  /*****************************************************************
      *****/
2256  /* añade una nueva luz alobjeto seleccionado en la fila y  columna
      y   */
2257  /* altura Z indicada, con  el color facilitado mmediante las
      */
2258  /* componentes R,G,B. Si ya existia una luz se elimina y se crea
      una   */
2259  /* nueva. En caso de filas y columnas incorrectas se posiciona se
      toma */
2260  /* por defecto el valor central de las mismas.
      */
2261  /*****************************************************************
      *****/
2262
2263  void activar_LuzPosicional(SoNode *nodoPadre, float fila,float
      columna,float Z, unsigned char R,unsigned char G,unsigned char B)
2264  {
2265
2266    SoSearchAction Nbusqueda;
2267    Nbusqueda.setName("LuzManip");  //nodo a buscar
2268    Nbusqueda.setInterest(SoSearchAction::FIRST);
2269    Nbusqueda.apply(nodoPadre);
2270    SoPath * ppath = Nbusqueda.getPath();
2271     //obtención del nodo buscado
2272    if (ppath!=NULL)  // si existia una luz
2273     desactivar_LuzPosicional(nodoPadre); //se elimina, para crear
2274
2275      //un nuevo objeto manipulador de luz
2276      SoSpotLightManip *luzManip=new SoSpotLightManip;
2277      luzManip->setName("LuzManip");
2278
2279      //informacion del objeto
2280      class infObjeto *nuevaInfo=infoObjeto(0,0,0,0,0,0,1,nodoPadre);
2281      //posicionamos la luz en la fila y columna indicada
2282      SbVec3f coordenadas;
2283
2284      if( (columna>nuevaInfo->numColumnas) || (columna<0) )
      //coordenada X=columna del objeto
2285      {
2286        printf("\n\nno existe ese numero de columna.Valores validos
      0-%d\n",nuevaInfo->numColumnas-1);
2287        printf("La luz se posicionara en la columna central del
      objeto\n"); //posición predeterminada
2288
      coordenadas[0]=((nuevaInfo->maxValorX-nuevaInfo->minValorX)/2.0)+nu
      evaInfo->minValorX; //columna central del objeto
2289      }
2290      else
2291        coordenadas[0]=(
      (nuevaInfo->maxValorX-nuevaInfo->minValorX)/nuevaInfo->numColumnas
      )*columna; //posicionamiento sobre la columna deseada
2292
```

```
2293
2294      if( (fila>nuevaInfo->numFilas) || (fila<0) ) //coordenada
      X=columna del objeto
2295      {
2296        printf("\n\nno existe ese numero de fila.Valores validos
      0-%d\n",nuevaInfo->numFilas-1);
2297        printf("La luz se posicionara en la columna central del
      objeto\n");//posición predeterminada
2298
      coordenadas[1]=(((nuevaInfo->maxValorY-nuevaInfo->minValorY)/2.0)+n
      uevaInfo->minValorY); //columna central del objeto
2299      }
2300      else
2301        coordenadas[1]=(
      (nuevaInfo->maxValorY-nuevaInfo->minValorY)/nuevaInfo->numFilas
      )*fila; //posicionamiento sobre la fila deseada
2302
2303      //altura de la luz
2304        coordenadas[2]=Z;
2305
2306      ( (SoSeparator *) nodoPadre )->insertChild(luzManip,0);
      //manipulador de luz como primer nodo hijo
2307      luzManip->location.setValue(coordenadas);
2308
2309      luzManip->ref();
2310      seleccion->touch();
2311  }
2312
2313  /*****************************************************************
      *****/
2314  /* Desactiva (elimina) el nodo manipulador del objeto seleccionado.
      */
2315  /*****************************************************************
      *****/
2316
2317  void desactivar_LuzPosicional(SoNode *nodoPadre)
2318  {
2319
2320    SoSearchAction Nbusqueda;
2321    Nbusqueda.setName("Luz");  //nodo a buscar
2322    Nbusqueda.setInterest(SoSearchAction::FIRST);
2323    Nbusqueda.apply(nodoPadre);  //nodo en el que buscar
2324    SoPath * ppath = Nbusqueda.getPath();
2325      //obtención del nodo buscado
2326    if (ppath!=NULL)
2327    {
2328      //lo obtenemos
2329      SoNode *nodoEncontrado=ppath->getNodeFromTail(0);
2330      ( (SoSeparator *) nodoPadre )->removeChild(nodoEncontrado);
      //borramos el manipulador
2331      seleccion->touch();
2332    }
2333
2334    Nbusqueda.setName("LuzManip");  //nodo a buscar
2335    Nbusqueda.setInterest(SoSearchAction::FIRST);
2336      Nbusqueda.apply(nodoPadre);
2337      ppath = Nbusqueda.getPath();
2338    if (ppath!=NULL)  // si hay ningún manipulador
2339    {
2340      //lo obtenemos
2341      SoNode *nodoEncontrado=ppath->getNodeFromTail(0);
```

```
2342      ( (SoSeparator *) nodoPadre )->removeChild(nodoEncontrado); //y
      borramos
2343      seleccion->touch();
2344    }
2345  }
2346
2347  /*****************************************************************
      *****/
2348  /*  Posiciona la luz, del objeto seleccionado, en las coordenadas
      */
2349  /*  indicadas.
      */
2350  /*****************************************************************
      *****/
2351  void PosicionLuz(float X,float Y,float Z,SoNode *nraiz)
2352  {
2353    SoSearchAction Nbusqueda;
2354    Nbusqueda.setName("LuzManip");
2355    Nbusqueda.setInterest(SoSearchAction::FIRST);
2356    Nbusqueda.apply(nraiz);
2357    SoPath * ppath = Nbusqueda.getPath();
2358    ppath = Nbusqueda.getPath();
2359
2360    if (ppath!=NULL)  // si existe un nodo de luz
2361    {
2362      //lo obtenemos y lo posicionamos en las coordenadas
2363      SoNode *nodoEncontrado=ppath->getNodeFromTail(0);
2364      ( (SoSpotLightManip *) nodoEncontrado
      )->location.setValue(X,Y,Z); //indicadas
2365      seleccion->touch();
2366    }
2367  }
2368
2369  /*****************************************************************
      *****/
2370  /*  Establece el color de la luz del objeto seleccionado
      */
2371  /*****************************************************************
      *****/
2372  void ColorLuz(int compR,int compG,int compB,SoNode *nraiz)
2373  {
2374    SoSearchAction Nbusqueda;
2375    Nbusqueda.setName("LuzManip");
2376    Nbusqueda.setInterest(SoSearchAction::FIRST);
2377    Nbusqueda.apply(nraiz);
2378    SoPath * ppath = Nbusqueda.getPath();
2379    ppath = Nbusqueda.getPath();
2380
2381    if (ppath!=NULL)  // si existe un nodo de luz
2382    {
2383      //lo obtenemos y establecemos el color
2384      SoNode *nodoEncontrado=ppath->getNodeFromTail(0);
2385      ( (SoSpotLightManip *) nodoEncontrado )->color.setValue(
      (float)(compR/255.0),
2386        (float)(compG/255.0),(float)(compB/255.0) ); //indicado
2387      seleccion->touch();
2388    }
2389  }
2390
2391  /*****************************************************************
      *****/
```

```
2392 /*   Texturiza el objeto seleccionado con el archivo de imagen
     indicado*/
2393 /*****************************************************************
     *****/
2394
2395 int texturizar(SoSeparator *grupo_figura,char *fichTextura)
2396 {
2397   //comprobamos el tipo de objeto
2398   SoQuadMesh *malla=(SoQuadMesh *)
     seleccionarHijo(grupo_figura,"malla");
2399
2400   if(malla==NULL)
2401   {
2402     printf("Error no se puede texturizar el objeto seleccionado,no
     es una malla\n");
2403     return(-1);
2404   }
2405   //creación de los nodos necesarios para texturizar el objeto
2406   SoTexture2 *textura=new SoTexture2;
2407   textura->setName("Textura");
2408   textura->filename.setValue(fichTextura);
2409   grupo_figura->insertChild(textura,3);
2410   return(0);
2411 }
2412
2413 /*****************************************************************
     *****/
2414 /*  Elimina todos los nodos manipuladores (Trackball/Box) que
     encuentra*/
2415 /*  en el nodo raiz.
     */
2416 /*****************************************************************
     *****/
2417
2418 int desactivar_manipuladores (SoNode *nraiz)
2419 {
2420   int manipuladores_borrados=0;
2421   //buscamos nodos del tipo manipulador
2422   SoSearchAction Nbusqueda;
2423   Nbusqueda.setName("Transforma");
2424   Nbusqueda.setInterest(SoSearchAction::ALL);
2425     Nbusqueda.apply(nraiz);
2426     SoPathList ppaths = Nbusqueda.getPaths();
2427
2428   //actualización de los campos antes de borrar los manipuladores
2429   if (ppaths!=NULL)  //para cada manipulador encontrado
2430   {
2431     for(int i=0;i<(ppaths.getLength() );i++)
2432     {
2433       SoPath *ppath=ppaths[i];
2434       //lo obtenemos
2435       SoNode *oldnode=(SoNode *) ppath->getNodeFromTail(0);
2436
2437       //comprobamos si se trata de un tipo Box o Trackball
2438           if ( (oldnode->getTypeId() ==
     SoTransformBoxManip::getClassTypeId()) || (oldnode->getTypeId() ==
     SoTrackballManip::getClassTypeId()) )  //si es el caso
2439       {
2440         // se obtiene el nodo padre
2441         SoNode *nodoPadre=obtenerPadre(oldnode,nraiz);
2442         //se crea el nodo que lo substituirá
```

```
2443          SoTransform *newnode=new SoTransform;
2444          //copia de los valores de los campos antes de borrar el
      nodo manipulador
2445          newnode->translation.setValue( ( (SoTransform *)
      oldnode)->translation.getValue() );
2446          newnode->rotation.setValue( ( (SoTransform *)
      oldnode)->rotation.getValue() );
2447          newnode->scaleFactor.setValue( ( (SoTransform *)
      oldnode)->scaleFactor.getValue() );
2448          newnode->center.setValue( ( (SoTransform *)
      oldnode)->center.getValue() );
2449          newnode->scaleOrientation.setValue( ( (SoTransform *)
      oldnode)->scaleOrientation.getValue() );  //esc orientación
2450          //y reemplazamos el oldnode por el newnode
2451          ppath->replaceIndex(nodoPadre,0,newnode);
2452          ((SoSeparator *)nodoPadre )->replaceChild(oldnode,newnode);
2453          //se establece el mismo nombre que el del nodo substituido
2454          newnode->setName("Transforma");
2455          manipuladores_borrados++; //incremento de nodos
      manipuladores "desactivados"
2456        }
2457      }
2458    }
2459    return(manipuladores_borrados);
2460  }
2461
2462
2463  /*****************************************************************
      *****/
2464  /* Filtra los datos del objeto seleccionado utilizando como valores
      */
2465  /* de corte los parámetros pasados. tipo= 0 filtrado paso bajo 1
      paso  */
2466  /* alto. Los colores superiores e inferiores se expresan en formato
      */
2467  /* RGB(float).Para valores negativos de componentes RGB se utiliza
      el  */
2468  /* mismo color original del objeto y para valores superiores a 1
      los   */
2469  /* puntos filtrados tomaran el color del fondo de la pantalla.
      */
2470  /*****************************************************************
      *****/
2471  int filtro(float TempCorte,SbVec3f colorSup,SbVec3f colorInf,int
      tipo,SoSeparator *grupo_figura)
2472  {
2473    //obtención de las coordenadas de la malla
2474    SoCoordinate3 *coordenadas=(SoCoordinate3
      *)seleccionarHijo(grupo_figura,"Coordenadas");
2475    SoMaterial *material=(SoMaterial
      *)seleccionarHijo(grupo_figura,"material");
2476
2477    if(coordenadas==NULL)//si no se ha obtenido el nodo de
      coordenadas
2478    {
2479      printf("Imposible obtener el nodo de coordenadas del
      objeto\n");
2480      return(-1);
2481    }
2482    //caso de lectura correcta
2483    //obtencin del numero de puntos de la malla
```

```
2484    long numeroPuntos=coordenadas->point.getNum();
2485    SbVec3f v;
2486    //color del fondo de la pantalla
2487    SbColor colorFondo=(SbColor) eviewer->getBackgroundColor();
2488    SbVec3f colorAplicado;
2489
2490    for(long i=0;i<numeroPuntos;i++)
2491    {
2492      switch(tipo)  //evaluación tipo de filtro a usar
2493      {
2494        case 0: //paso bajo
2495           if ( coordenadas->point[i][2] > TempCorte )
2496         {
2497          v=coordenadas->point[i];
2498          v[2]=0; //temperatura=0
2499          coordenadas->point.set1Value(i,v);
2500          //Evaluación de las componentes de color
2501          for(int c=0;c<3;c++)
2502          {
2503            if(colorSup[c]<0) //caso de ser negativa
2504              colorAplicado[c]=material->diffuseColor[i][c]; //se
     aplica el mismo valor de componente del objeto
2505            if(colorSup[c]>1) //si se desea que tome el valor de
     fondo
2506              colorAplicado[c]=colorFondo[c];
2507            if( (colorSup[c]>=0) && (colorSup[c]<=1) ) //para valores
     dentro del rango 0-1 ( RGB 0 a 255)
2508              colorAplicado[c]=colorSup[c]; //se aplica ese valor a
     la componente correspondiente
2509          }
2510          material->diffuseColor.set1Value(i,(SbColor)colorAplicado
     );
2511        }
2512          else //para Temp inferiores a la de corte
2513        {
2514          //evaluacion de las componentes
2515          for(int c=0;c<3;c++)
2516          {
2517            if(colorInf[c]<0)
2518              colorAplicado[c]=material->diffuseColor[i][c];
2519            if(colorInf[c]>1)
2520              colorAplicado[c]=colorFondo[c];
2521            if( (colorInf[c]>=0) && (colorInf[c]<=1) )
2522              colorAplicado[c]=colorInf[c];
2523          }
2524          material->diffuseColor.set1Value(i,(SbColor)colorAplicado
     );
2525        }
2526       break;
2527
2528       case 1: //filtro paso alto
2529        if ( coordenadas->point[i][2] < TempCorte )
2530        {
2531         v=coordenadas->point[i];
2532         v[2]=0; //temeperatura=0
2533         coordenadas->point.set1Value(i,v);
2534
2535         //evaluación de las componentes de color
2536         for(int c=0;c<3;c++)
2537         {
2538           if(colorInf[c]<0)
```

```
2539              colorAplicado[c]=material->diffuseColor[i][c];
2540            if(colorInf[c]>1)
2541              colorAplicado[c]=colorFondo[c];
2542            if( (colorInf[c]>=0) && (colorInf[c]<=1) )
2543              colorAplicado[c]=colorInf[c];
2544          }
2545          material->diffuseColor.set1Value(i,(SbColor)colorAplicado
      );
2546        }
2547        else //temperaturas superiores a la de corte
2548        {
2549         //evaluacion de las componentes
2550         for(int c=0;c<3;c++)
2551         {
2552           if(colorSup[c]<0)
2553             colorAplicado[c]=material->diffuseColor[i][c];
2554           if(colorSup[c]>1)
2555             colorAplicado[c]=colorFondo[c];
2556           if( (colorSup[c]>=0) && (colorSup[c]<=1) )
2557             colorAplicado[c]=colorSup[c];
2558         }
2559         material->diffuseColor.set1Value(i,(SbColor)colorAplicado
      );
2560        }
2561       break;
2562     }
2563   }
2564   return(0); //ejecución sin errores
2565 }
2566
2567 /*****************************************************************
      *****/
2568 /*  filtra los datos del objeto seleccionado utilizando como
      valores   */
2569 /*  de corte los parámetros pasados. tipo= 0 filtrado rechaza
      banda, 1 */
2570 /*  pasa pasabanda. Los colores para la banda pasante, la inferior
      y la*/
2571 /*  superior se expresan en formato RGB, para valores negativos de
      */
2572 /*  componentes RGB se utiliza el mismo color original del objeto y
      */
2573 /*  para valores superiores a 1 los puntos filtrados tomaran el
      color  */
2574 /*  del fondo de la pantalla
      */
2575 /*****************************************************************
      *****/
2576
2577 int filtroBanda(float TempSup,float TempInf,SbVec3f
      colorSup,SbVec3f colorBanda,SbVec3f colorInf,int tipo,SoSeparator
      *grupo_figura)
2578 {
2579   //obtención de las coordenadas del objeto seleccionado
2580   SoCoordinate3 *coordenadas=(SoCoordinate3
      *)seleccionarHijo(grupo_figura,"Coordenadas");
2581
2582   SoMaterial *material=(SoMaterial
      *)seleccionarHijo(grupo_figura,"material");
2583   //obtención color de fondo de la pantalla
2584   SbColor colorFondo=(SbColor) eviewer->getBackgroundColor();
```

```
2585    SbVec3f colorAplicado;
2586
2587    if(coordenadas==NULL)
2588    {
2589      printf("Imposible obtener el nodo de coordenadas del
      objeto\n");
2590      return(-1);
2591    }
2592    //caso de lectura correcta
2593    //obtencion del número de puntos de la malla
2594    long numeroPuntos=coordenadas->point.getNum();
2595    SbVec3f v;
2596    for(long i=0;i<numeroPuntos;i++)
2597    {
2598      switch(tipo) //tipo de filtro
2599      {
2600      case 0: //Rechaza Banda
2601        //si se desea usar color Inf
2602        if ( coordenadas->point[i][2] < TempInf )
2603        {
2604          //evaluacion de las componentes
2605          for(int c=0;c<3;c++)
2606          {
2607            if(colorInf[c]<0) //si la componente pasada es negativa
2608              colorAplicado[c]=material->diffuseColor[i][c]; //se
      aplica el mismo valor de componente del objeto
2609            if(colorInf[c]>1) //si se desea que tome el valor de
      fondo
2610              colorAplicado[c]=colorFondo[c];
2611            if( (colorInf[c]>=0) && (colorInf[c]<=1) ) //para valores
      dentro del rango 0-1 ( RGB 0 a 255)
2612              colorAplicado[c]=colorInf[c]; //se aplica ese valor a
      la componente
2613          }
2614          material->diffuseColor.set1Value(i,(SbColor)colorAplicado
      );
2615        }
2616        //Si se desea usar color Sup
2617        if ( coordenadas->point[i][2] > TempSup )
2618        {
2619          //evaluacion de las componentes
2620          for(int c=0;c<3;c++)
2621          {
2622            if(colorSup[c]<0)
2623              colorAplicado[c]=material->diffuseColor[i][c];
2624            if(colorSup[c]>1)
2625              colorAplicado[c]=colorFondo[c];
2626            if( (colorSup[c]>=0) && (colorSup[c]<=1) )
2627              colorAplicado[c]=colorSup[c];
2628          }
2629          material->diffuseColor.set1Value(i,(SbColor)colorAplicado
      );
2630        }
2631
2632
2633        if ( (coordenadas->point[i][2] > TempInf)  &&
      (coordenadas->point[i][2] < TempSup) )
2634        {
2635          v=coordenadas->point[i];
2636          v[2]=0; //temperatura=0
2637          coordenadas->point.set1Value(i,v);
```

```
2638          //Evaluacion del color de banda
2639          for(int c=0;c<3;c++)
2640          {
2641            if(colorBanda[c]<0) //si la componente pasada es negativa
2642              colorAplicado[c]=material->diffuseColor[i][c]; //se
      aplica el mismo valor de componente del objeto
2643            if(colorBanda[c]>1) //si se desea que tome el valor de
      fondo
2644              colorAplicado[c]=colorFondo[c];
2645            if( (colorBanda[c]>=0) && (colorBanda[c]<=1) ) //para
      valores dentro del rango 0-1 ( RGB 0 a 255)
2646              colorAplicado[c]=colorBanda[c]; //se aplica ese valor a
      la componente
2647          }
2648          material->diffuseColor.set1Value(i,(SbColor)colorAplicado
      );
2649        }
2650      break;
2651
2652      case 1: //Pasa Banda
2653        //evaluacion valores inferiores
2654        if ( coordenadas->point[i][2] < TempInf )
2655        {
2656          v=coordenadas->point[i];//se hace 0 el valor de Z
2657          v[2]=0; //temperatura=0
2658          coordenadas->point.set1Value(i,v);
2659          //evaluacion de las componentes
2660          for(int c=0;c<3;c++)
2661          {
2662            if(colorInf[c]<0)
2663              colorAplicado[c]=material->diffuseColor[i][c];
2664            if(colorInf[c]>1)
2665              colorAplicado[c]=colorFondo[c];
2666            if( (colorInf[c]>=0) && (colorInf[c]<=1) )
2667                colorAplicado[c]=colorInf[c];
2668          }
2669          material->diffuseColor.set1Value(i,(SbColor)colorAplicado
      );
2670        }
2671        //Evaluacion valores superiores
2672
2673        if ( coordenadas->point[i][2] > TempSup )
2674        {
2675          v=coordenadas->point[i];
2676          v[2]=0; //temperatura=0
2677          coordenadas->point.set1Value(i,v);
2678          //evaluacion de las componentes
2679          for(int c=0;c<3;c++)
2680          {
2681            if(colorSup[c]<0)
2682              colorAplicado[c]=material->diffuseColor[i][c];
2683            if(colorSup[c]>1)
2684              colorAplicado[c]=colorFondo[c];
2685            if( (colorSup[c]>=0) && (colorSup[c]<=1) )
2686              colorAplicado[c]=colorSup[c];
2687
2688          }
2689          material->diffuseColor.set1Value(i,(SbColor)colorAplicado
      );
2690        }
2691        //Evaluación de los puntos que forman parte de la banda
```

```
2692        if ( (coordenadas->point[i][2] >= TempInf ) &&
    (coordenadas->point[i][2] <= TempSup ))
2693        {
2694          for(int c=0;c<3;c++)
2695          {
2696            if(colorBanda[c]<0)
2697              colorAplicado[c]=material->diffuseColor[i][c];
2698            if(colorBanda[c]>1)
2699              colorAplicado[c]=colorFondo[c];
2700            if( (colorBanda[c]>=0) && (colorBanda[c]<=1) )
2701              colorAplicado[c]=colorBanda[c];
2702          }
2703          material->diffuseColor.set1Value(i,(SbColor)colorAplicado
    );
2704        }
2705      break;
2706      }
2707    }
2708    return(0);  //ejecucion sin errores
2709  }
2710
2711
2712  void seleccionarArea(float X1,float X2,float Y1,float
    Y2,SoSeparator *grupo_figura,unsigned char autoeje)
2713  {
2714
2715    cortarXarea(X1,Y1,X2,Y2,grupo_figura);
2716    SoTransform *transforma=(SoTransform *)seleccionarHijo((SoNode
    *)grupo_figura,"Transforma");
2717    translacion_objeto(0,0,0,transforma);
2718
2719  /*  if(autoeje==1) //si se desea actualizar el eje de coordenadas a
    la nueva area seleccionada
2720    {
2721      SoSeparator *ejeCoord=(SoSeparator *)seleccionarHijo( (SoNode
    *)grupo_figura,"EjeCoord");
2722      //si el objeto tenia asociado un eje de coordenadas
2723      if(ejeCoord!=NULL)
2724      {
2725        grupo_figura->removeChild(ejeCoord);//se borra el eje antiguo
2726        autoEje(grupo_figura);//y se crea uno automaticamente
2727      }
2728    }
2729  */
2730  }
2731
2732  /*************************************************************
    *****/
2733  /*  Realiza una translación sumando el valor pasado a la posición
    */
2734  /*  actual del nodo SoTransform.
    */
2735  /*************************************************************
    *****/
2736
2737  void translacion_objeto (float X,float Y,float Z, SoTransform
    *pobjeto)
2738  {
2739    SbVec3f posicion;
2740    //primero obtenemos la posición actual del objeto
2741    if (pobjeto!=NULL)
```

```
2742   {
2743     posicion = pobjeto->translation.getValue(); //obtenemos la
     posicion actual del objeto
2744     posicion[0]=posicion[0] + X; //suma de las coordenadas
     indicadas
2745     posicion[1]=posicion[1] +Y;
2746     posicion[2]=posicion[2] +Z;
2747
2748
     pobjeto->translation.setValue(posicion[0],posicion[1],posicion[2]);
2749   }
2750 }
2751
2752 /*****************************************************************
     *****/
2753 /*  Establece el campo de translación con los valores indicados
     */
2754 /*****************************************************************
     *****/
2755 void posicion_Objeto (float X, float Y, float Z, SoTransform
     *pobjeto)
2756 {
2757   if (pobjeto!=NULL)
2758   {
2759     pobjeto->translation.setValue(X,Y,Z);
2760   }
2761 }
2762
2763 /*****************************************************************
     *****/
2764 /*  Establece el contenido del campo totation con los valores
     indicados*/
2765 /*****************************************************************
     *****/
2766 void rotacion_objeto (float X,float Y,float Z,float
     angulo,SoTransform *pobjeto)
2767 {
2768   //buscamos el nodo hijo del tipo SoTransform
2769   angulo=(float) ((angulo*PI)/180.0);   //angulo a aplicar
     (sexagesimales)
2770   if (pobjeto!=NULL)
2771     //obtenemos los valores de angulo actuales
2772   pobjeto->rotation.setValue(SbVec3f(X,Y,Z),angulo);  //cast
     obligatorio segun documentación COIN3D
2773
2774 }
2775
2776 /*****************************************************************
     *****/
2777 /*   Establece el campo de factor de escala a los valor pasados
     como   */
2778 /*   argumentos. Valores por defecto 1 1 1
     */
2779 /*****************************************************************
     *****/
2780
2781 void escalar_objeto (float factorX,float factorY,float
     factorZ,SoTransform *pobjeto)
2782 {
2783   if (pobjeto!=NULL)
2784   {
```

```
2785     if ( (factorX<=0) || (factorY<=0)|| (factorZ<=0) )
2786      {
2787        printf("\nError en  el factor de escala. El valor debe ser
      superior a 0\n");
2788        return(-1);
2789      }
2790     else
2791      pobjeto->scaleFactor.setValue(factorX,factorY,factorZ);
2792    }
2793    return(0); //ejecución sin errores
2794  }
2795
2796  /*****************************************************************
      *****/
2797  /*  Establece el campo center, del objeto seleccionado, con los
      valores*/
2798  /*  indicados.
      */
2799  /*****************************************************************
      *****/
2800
2801  void centro_objeto (float coordX,float coordY,float
      coordZ,SoSeparator *grupo_figura)
2802  {
2803    //obtención del nodo SoTransform
2804    SoTransform
      *transforma=(SoTransform*)seleccionarHijo(grupo_figura,"Transforma"
      );
2805    if(transforma!=NULL)
2806      transforma->center.setValue(coordX,coordY,coordZ);  //uso
      obligatorio cast a SbVec3f segun documentacion COIN
2807
2808  }
2809
2810
2811  /*****************************************************************
      *****/
2812  /*  Esta función realiza el reflejo vertical de una malla.
      */
2813  /*  permuta todas las columnas de la malla  recorriendo
      */
2814  /*  y modificando los valores de la coordenada Z necesarios
      */
2815  /*****************************************************************
      *****/
2816  void reflejarV(SoSeparator *grupo_figura)
2817  {
2818    //obtención del nodo de coordenadas
2819    SoCoordinate3 *coordenadas=(SoCoordinate3 *)seleccionarHijo(
      (SoNode *)grupo_figura,"Coordenadas");
2820    //obtención del nodo de la malla (SoQuadMesh)
2821    SoQuadMesh *malla=(SoQuadMesh *)seleccionarHijo( (SoNode
      *)grupo_figura,"malla");
2822    //nodo que almacena el enlace de color para cada vértice de la
      malla
2823    SoMaterial *material=(SoMaterial *)seleccionarHijo( (SoNode
      *)grupo_figura,"material");
2824
2825    unsigned long numFilas=(unsigned
      long)malla->verticesPerColumn.getValue();
2826    unsigned long numColumnas=(unsigned
```

```
      long)malla->verticesPerRow.getValue();
2827
2828    SbVec3f valor1; //almacenamiento temporal;
2829    SbVec3f valor2;
2830    float temp;
2831    SbColor colorPunto; //almacenamiento temporal del color
2832    SbColor color2;
2833    for(unsigned long i=0;i<numFilas;i++)
2834    {
2835      for(unsigned long j=0;j<(numColumnas/2);j++)//se asume posible
      truncamiento
2836      {
2837        valor1=coordenadas->point[i*numColumnas+j];  //guardamos
      valor inicial
2838        valor2=coordenadas->point[i*numColumnas+ (numColumnas-1-j)];
      //y final
2839
2840        temp=valor1[2];
2841        valor1[2]=valor2[2];
2842        valor2[2]=temp;
2843        //introducción de valores intercambiados
2844        coordenadas->point.set1Value( (i*numColumnas+j),valor1);
2845        coordenadas->point.set1Value( (i*numColumnas+
      (numColumnas-1-j)),valor2);
2846
2847
2848        //se reflejan tambien los colores
2849        colorPunto=(SbColor)material->diffuseColor[i*numColumnas+j];
2850        color2=(SbColor)material->diffuseColor[i*numColumnas+
      (numColumnas-1-j)];
2851        material->diffuseColor.set1Value( (i*numColumnas+j),color2);
2852        material->diffuseColor.set1Value( (i*numColumnas+
      (numColumnas-1-j)),colorPunto);
2853      }
2854    }
2855  }
2856
2857  /*****************************************************************
      *****/
2858  /*  Esta función realiza el reflejo horizontal de una malla.
      */
2859  /*  permuta todas las filas de la malla  recorriendo
      */
2860  /*  el nodo SoCoordinate3 y permutando los valores de las
      coordenadas Z*/
2861  /*****************************************************************
      *****/
2862  void reflejarH(SoSeparator *grupo_figura)
2863  {
2864
2865    //obtención del nodo de coordenadas
2866    SoCoordinate3 *coordenadas=(SoCoordinate3 *)seleccionarHijo(
      (SoNode *)grupo_figura,"Coordenadas");
2867    //obtención del nodo de la malla
2868    SoQuadMesh *malla=(SoQuadMesh *)seleccionarHijo( (SoNode
      *)grupo_figura,"malla");
2869    //nodo que almacena el enlace de color para cada vértice de la
      malla
2870    SoMaterial *material=(SoMaterial *)seleccionarHijo( (SoNode
      *)grupo_figura,"material");
2871
```

```
2872   unsigned long numFilas=(unsigned
     long)malla->verticesPerColumn.getValue();
2873   unsigned long numColumnas=(unsigned
     long)malla->verticesPerRow.getValue();
2874
2875   SbVec3f valor1;
2876   SbVec3f valor2;
2877   float temp;
2878   SbColor colorPunto;
2879   SbColor color2;
2880   for(unsigned long j=0;j<numColumnas;j++)
2881   {
2882     for(unsigned long i=0;i<(numFilas/2);i++)//se asume el posible
     truncamiento
2883     {
2884       valor1=coordenadas->point[i*numColumnas+j];  //guardamos
     valor inicial
2885       valor2=coordenadas->point[numColumnas*(numFilas-1-i)+j]; //y
     final
2886
2887       temp=valor1[2];
2888       valor1[2]=valor2[2];
2889       valor2[2]=temp;
2890       //introducción de valores intercambiados
2891       coordenadas->point.set1Value( (i*numColumnas+j),valor1);
2892       coordenadas->point.set1Value(
     (numColumnas*(numFilas-1-i)+j),valor2);
2893
2894
2895       //se reflejan también los colores
2896       colorPunto=(SbColor)material->diffuseColor[i*numColumnas+j];
2897
     color2=(SbColor)material->diffuseColor[numColumnas*(numFilas-1-i)+j
     ];
2898       material->diffuseColor.set1Value( (i*numColumnas+j),color2);
2899       material->diffuseColor.set1Value(
     (numColumnas*(numFilas-1-i)+j),colorPunto);
2900     }
2901   }
2902 }
2903
2904 /*****************************************************************
     *****/
2905 /* Devuelve un nodo separador que contiene todos los nodos
     necesarios  */
2906 /* para la representación de un eje de coordenadas con los
     parámetros  */
2907 /* undicados por el usuario
     */
2908 /*****************************************************************
     *****/
2909
2910 int autoEje( SoSeparator *grupo_figura )
2911 {
2912   //miramos si el objeto tiene un eje asociado
2913   SoSeparator *oldEje=(SoSeparator
     *)seleccionarHijo(grupo_figura,"EjeCoord");
2914
2915   if ( oldEje!=NULL ) //en caso de existir alguno
2916   {
2917     grupo_figura->removeChild(oldEje);
```

```
2918    }
2919
2920    //obtenemos malla
2921    SoQuadMesh *malla=(SoQuadMesh *) seleccionarHijo( (SoNode
      *)grupo_figura,"malla");
2922    SoTransform *transforma=(SoTransform *) seleccionarHijo( (SoNode
      *)grupo_figura,"Transforma");
2923
2924    //y el nodo de coordenadas
2925    SoCoordinate3 *coordenadas=(SoCoordinate3
      *)seleccionarHijo(grupo_figura,"Coordenadas");
2926      if(coordenadas==NULL)//en caso de error
2927        return(-1);//exit
2928
2929    SbVec3f v;
2930    v=coordenadas->point[0]; //primera coordenada de SoCoordinate3
2931
2932      //nueva instancia de ejecoordenadas
2933      ejecoordenadas *nuevoEje=new ejecoordenadas;
2934      //se obtiene información del objeto
2935      infObjeto *nuevaInfo=infoObjeto (0,0,0,0,0,0,1,(SoNode *)
      grupo_figura); //mostrar=1 para obtener solo los resultados
2936
2937      //adecuación de las propiedades del eje de coordenadas a las
      dimensiones
2938      //de la malla/nueb de puntos
2939
      nuevoEje->maxvalueX=(nuevaInfo->maxValorX-nuevaInfo->minValorX);
2940
      nuevoEje->maxvalueY=(nuevaInfo->maxValorY-nuevaInfo->minValorY);
2941      nuevoEje->maxvalueZ=nuevaInfo->maxValorZ+nuevaInfo->minValorZ;
2942
2943      nuevoEje->minvalueX=1;
2944      nuevoEje->minvalueY=1;
2945      nuevoEje->minvalueZ=nuevaInfo->minValorZ;
2946
2947      nuevoEje->colorejeX->componenteR=1;
2948      nuevoEje->colorejeX->componenteG=0;
2949      nuevoEje->colorejeX->componenteB=0;
2950
2951      nuevoEje->colorejeY->componenteR=0;
2952      nuevoEje->colorejeY->componenteG=1;
2953      nuevoEje->colorejeY->componenteB=0;
2954
2955      nuevoEje->colorejeZ->componenteR=0;
2956      nuevoEje->colorejeZ->componenteG=0;
2957      nuevoEje->colorejeZ->componenteB=1;
2958
2959      //numero de marcas para cada eje
2960      nuevoEje->marcasX=(int)(
      (nuevaInfo->maxValorX-nuevaInfo->minValorX)/15 );
2961      nuevoEje->marcasY=(int)(
      (nuevaInfo->maxValorY-nuevaInfo->minValorY)/15 );
2962      nuevoEje->marcasZ=(int)(
      (nuevaInfo->maxValorZ-nuevaInfo->minValorZ)/15 );
2963
2964        if( ((nuevaInfo->maxValorX-nuevaInfo->minValorX) <
      (nuevaInfo->maxValorY-nuevaInfo->minValorY)) &&
      ((nuevaInfo->maxValorX-nuevaInfo->minValorX) <
      (nuevaInfo->maxValorZ-nuevaInfo->minValorZ)) )
2965        {
```

```
2966
      nuevoEje->tamanoFuente=(nuevaInfo->maxValorX-nuevaInfo->minValorX)/
      8.0;
2967
      nuevoEje->longmarcaX=(nuevaInfo->maxValorX-nuevaInfo->minValorX)/10
      .0;
2968
      nuevoEje->longmarcaY=(nuevaInfo->maxValorX-nuevaInfo->minValorX)/10
      .0;
2969
      nuevoEje->longmarcaZ=(nuevaInfo->maxValorX-nuevaInfo->minValorX)/10
      .0;
2970          nuevoEje->separacionEtiqueta=(nuevaInfo->maxValorX)/12.0;
2971
2972        }
2973        if( ( (nuevaInfo->maxValorY-nuevaInfo->minValorY) <
      (nuevaInfo->maxValorX-nuevaInfo->minValorX)) && (
      (nuevaInfo->maxValorY-nuevaInfo->minValorY) <
      (nuevaInfo->maxValorZ-nuevaInfo->minValorZ) ) )
2974        {
2975
      nuevoEje->tamanoFuente=(nuevaInfo->maxValorY-nuevaInfo->minValorY)/
      8.0;
2976
      nuevoEje->longmarcaX=(nuevaInfo->maxValorY-nuevaInfo->minValorY)/10
      .0;
2977
      nuevoEje->longmarcaY=(nuevaInfo->maxValorY-nuevaInfo->minValorY)/10
      .0;
2978
      nuevoEje->longmarcaZ=(nuevaInfo->maxValorY-nuevaInfo->minValorY)/10
      .0;
2979
      nuevoEje->separacionEtiqueta=(nuevaInfo->maxValorY-nuevaInfo->minVa
      lorY)/12.0;
2980        }
2981        if( ((nuevaInfo->maxValorZ-nuevaInfo->minValorZ) <
      (nuevaInfo->maxValorX-nuevaInfo->minValorX) ) && (
      (nuevaInfo->maxValorZ-nuevaInfo->minValorZ) <
      (nuevaInfo->maxValorY-nuevaInfo->minValorY) ) )
2982        {
2983          nuevoEje->tamanoFuente=
      (nuevaInfo->maxValorZ-nuevaInfo->minValorZ )/8.0;
2984
      nuevoEje->longmarcaX=(nuevaInfo->maxValorZ-nuevaInfo->minValorZ)/10
      .0;
2985
      nuevoEje->longmarcaY=(nuevaInfo->maxValorZ-nuevaInfo->minValorZ)/10
      .0;
2986
      nuevoEje->longmarcaZ=(nuevaInfo->maxValorZ-nuevaInfo->minValorZ)/10
      .0;
2987          nuevoEje->separacionEtiqueta=(
      (nuevaInfo->maxValorZ-nuevaInfo->minValorZ )/12.0);
2988        }
2989
2990      //posicionamiento del eje de coordenadas
2991      nuevoEje->posZ=nuevaInfo->minValorZ;
2992      nuevoEje->posX=v[0];
2993      nuevoEje->posY=v[1];
2994
```

```
2995      //creación del eje de coordenadas
2996      SoSeparator *nodoEje=generarEjeCoord(nuevoEje);
2997      nodoEje->setName("EjeCoord");
2998
2999      nodoEje->ref();
3000      grupo_figura->addChild (nodoEje); //y lo incluimos en el grupo
      de la malla seleccionada
3001
3002  }
3003
3004  /*****************************************************************
      *****/
3005  /*  Resorre el nodo SoMaterial del objeto pasado, estableciendo los
      */
3006  /*  valores de color para cada vértice según la temperatura
      (coordenada*/
3007  /*  Z) y el valor de color contenido en el archivoPaleta
      */
3008  /*****************************************************************
      *****/
3009
3010  void asignar_Paleta(char *archivoPaleta, SoSeparator *grupo_figura)
3011  {
3012
3013    //se carga la paleta del archivo
3014    SbColor *pPaleta=cargar_Paleta(archivoPaleta,8);
3015    SoCoordinate3 *pCoordenadas=(SoCoordinate3 *)(
      seleccionarHijo(grupo_figura,"Coordenadas") );
3016
3017    //obtencion del numero de vértices/puntos de la malla/nube
3018    if(pPaleta==NULL) //en caso de error
3019    {
3020      printf("\n\nError al cargar la paleta de colores\n");
3021      return; //exit
3022    }
3023
3024    SoMaterial *material=(SoMaterial *)seleccionarHijo( (SoNode
      *)grupo_figura,"material" );
3025
3026    //obtención de información sobre el objeto (malla/nueb)
3027    class infObjeto *nuevaInfo=infoObjeto(0,0,0,0,0,0,1, ( SoNode *)
      grupo_figura);
3028
3029    datFichero *datosFichero=new datFichero();  //objeto para
      contener
3030    datosFichero->indicepaleta=new
      SbColor[nuevaInfo->numColumnas*nuevaInfo->numFilas];//los nuevos
      indices a la paleta cargada
3031    //recorremos todos los vértices de la malla/nube
3032    SbVec3f pPunto;
3033    float g;
3034    int indice=0;
3035    for(unsigned int i=0;i< (nuevaInfo->numFilas );i++)
3036    {
3037      for(unsigned int j=0;j< (nuevaInfo->numColumnas );j++)
3038      {
3039       //para cada valor de coordenadaZ (temperatura)
3040       pPunto=(SbVec3f)
      (pCoordenadas->point[i*nuevaInfo->numColumnas+j]);
3041
3042       //se calcula el índice correspondiente
```

```
3043        indice=(int)( (
      (pPunto[2]-nuevaInfo->minValorZ)*255)/(nuevaInfo->maxValorZ-nuevaIn
      fo->minValorZ) )+0.5;
3044
      datosFichero->indicepaleta[i*nuevaInfo->numColumnas+j]=pPaleta[indi
      ce];
3045      }
3046    }
3047    //se guardan los nuevos valores de color
3048    material->diffuseColor.setValues(0,nuevaInfo->numFilas*nuevaInfo-
      >numColumnas,datosFichero->indicepaleta);
3049  }
3050
3051
3052  /*****************************************************************
      *****/
3053  /*   Establece la posición de la cámara en las coordenadas
      indicadas   */
3054  /*****************************************************************
      *****/
3055  int posCamara(float X,float Y,float Z)
3056  {
3057    SoCamera *camara=(SoCamera *)eviewer->getCamera();
3058    if (camara==NULL) return(-1);
3059    camara->position.setValue(X,Y,Z);
3060  }
3061
3062  /*****************************************************************
      *****/
3063  /*  Incrementa/decrementa el valor de las coordenadas donde se
      */
3064  /*  encuentra situada la cámara
      */
3065  /*****************************************************************
      *****/
3066  int transCamara(float X,float Y,float Z)
3067  {
3068    static SoCamera *camara=(SoCamera *)eviewer->getCamera();
      //obtenemos la camara
3069    if (camara==NULL) return(-1);
3070    SbVec3f valorAnterior=camara->position.getValue();
3071
3072    valorAnterior[0]=valorAnterior[0]+X;
3073    valorAnterior[1]=valorAnterior[1]+Y;
3074    valorAnterior[2]=valorAnterior[2]+Z;
3075
3076    camara->position.setValue(valorAnterior);
3077    seleccion->touch();
3078  }
3079
3080  /*****************************************************************
      *****/
3081  /*  Establece la orientación de la cámara, rotando sobre los ejes
      */
3082  /*  indicados, el ángulo (en sexagesimales) indicado.
      */
3083  /*****************************************************************
      *****/
3084  int orCamara(float rotX,float rotY,float rotZ,float angulo)
3085  {
3086    SoCamera *camara=(SoCamera *)eviewer->getCamera(); //obtenemos la
```

```
      camara
 3087   if (camara==NULL) return(-1);
 3088   SbVec3f rot;
 3089   SbRotation rotacion;
 3090
 3091   rot.setValue(rotX,rotY,rotZ);
 3092   rotacion.setValue(rot, (float)((angulo*PI)/180.0) );//valor en
      sexagesimales
 3093   camara->orientation.setValue(rotacion);
 3094 }
 3095
```

```
 1 #include "stdAfx.h"
 2 #include <iostream>
 3 #include "gestionteclas.h"
 4 #include "stdio.h"
 5 #include "string.h"
 6
 7 bufferteclas::bufferteclas()
 8 {
 9   tamano=1;
10   cadena=new char[LONG_ARRAY];
11   numteclas=0;
12   flag_numeros=0;
13 }
14 //destructor
15 bufferteclas::~bufferteclas()
16 {
17   delete [] cadena; //liberamos memoria
18 }
19 //Funciones miembro
20 char * bufferteclas::get_cadena()
21 {
22   return(cadena);
23 }
24 int bufferteclas::get_tamano()
25 {
26   return(tamano);
27 }
28
29 void bufferteclas::add_caracter(char ncaracter)  //funcion que
   gestiona la pulsacion de teclas
30 {
31 if((ncaracter!=13)&&(ncaracter!='\b') )//primero miramos antes de
   nada que no se trate de la tecla INTRO ni borrar ni ALT
32   {
33       if(numteclas<LONG_ARRAY) //si tenemos espacio en el array de
   caracteres
34       {
35         if(ncaracter==',')  //correccion para los decimales (se
   considera el punto decimal tanto las comas como los
   puntos).Necesario para compatibilidad con el visor GUI de VBasic
36           ncaracter='.'; //se sustituye la coma por el punto
37       memcpy(cadena,&ncaracter,1);//escribimos el nuevo caracter en
   array temporal;
38       cadena++; //y apuntamos siguiente direccion
39       numteclas++; tamano++; //incrementamos variables de control e
   informacion
40       }
41       else //para evitar desbordamiento del array si
42       {    //consideraremos que los caracteres entrados son
   excesivos
43       printf("\ndemasiado larga...\n");
44         cadena=cadena-numteclas;//volvemos a la posicion inicial
   del array
45       reset(); //y restablecemos situacion original
46       }
47   }
48 if(ncaracter==13)   //si pulsamos la tecla INTRO
49   {
50   if(numteclas>0) //y hay alguna tecla almacenada en el array
51     {
52     *cadena='\0'; // indicamos que se ha terminado de capturar una
```

```
    cadena
 53     cadena=cadena-(numteclas); //volvemos a la posicion inicial del
    array
 54     reset(); //AL OBTENER LA CADENA REINICIAMOS LA INFORMACION DE
    LAS VARIABLES
 55     }
 56   }
 57 if(ncaracter=='\b')  //si se trata de la tecla de borrar
    (BACKSPACE)
 58   {
 59   if(numteclas>0) //y hay alguna tecla guardada en el array
 60     {
 61     cadena=cadena-1; //retrocedemos una posicion=borrar el ultimo
    caracter guardado
 62     tamano--;
 63     numteclas--;
 64     }
 65   }
 66 }
 67
 68 void bufferteclas::reset()
 69 {
 70   tamano=1;
 71   numteclas=0;
 72   flag_numeros=0;
 73 }
 74
 75 /* funcion de traduccion de codigos especiales SHIFT+numeros (de
    distribucion teclado US a ESP*/
 76
 77 char conv_shift (char codigo)
 78 {
 79   switch (codigo)
 80   {
 81   case '1':    //caracter alt+1
 82   return('!');
 83   break;
 84   case '2':    //caracter alt+2
 85   return('"');
 86   break;
 87   case '3':
 88   return('·');
 89   break;
 90   case '4':
 91   return('$');
 92   break;
 93   case '5':
 94   return('%');
 95   break;
 96   case '6':
 97   return('&');
 98   break;
 99   case '7':
100   return('/');
101   break;
102   case '8':
103   return('(');
104   break;
105   case '9':
106   return(')');
107   break;
```

```
108   case '0':
109   return('=');
110   break;
111   /*conversion caracteres especiales*/
112   case ',':
113   return(';');
114   break;
115   case '.':
116   return(':');
117   break;
118   case '-':
119   return('_');
120   break;
121   case 'º':
122   return('\\');
123   break;
124   }
125     /*mayusculas*/
126   if( (codigo>=65) && (codigo<=122) )
127   {
128     codigo=codigo-32;
129     return(codigo);
130   }
131
132 return(-1);  //retorno sin valor convertido
133 }
134
135 char *eliminar_espacios(char *cadena) //elimina los espacios de la
    cadena pasada
136 {
137   int i=0,indice=0;
138   int longitud=strlen(cadena);
139
140   char *temporal=new char [longitud+1];  //creamos una cadena
    temporal del mismo tamaño que la original
141
142   for(i=0;i<longitud;i++)  //para cada uno de los caracteres de la
    cadena original
143   {
144     if(cadena[i]!=' ')   //si no es un espacio
145     {
146     temporal[indice]=cadena[i];
147     indice++;
148     }
149   }
150   temporal[indice]='\0'; //cerramos cadena
151 return temporal;
152 }
153
```

```
#include "StdAfx.h"
#include <Inventor/SbBasic.h>
#include <Inventor/nodes/SoTransform.h>
#include <Inventor/Win/viewers/SoWinExaminerViewer.h>
#include <Inventor/nodes/SoSelection.h>
#include <Inventor/nodes/SoCoordinate3.h>
#include <Inventor/SbVec3f.h>
#include <Inventor/sensors/SoTimerSensor.h>
#include <Inventor/Win/SoWin.h>
#include <Inventor/SbColor.h>

#include "freeimage.h"
#include "leerfichero.h"
#include "figuras.h"
#include "colores.h"
#include "gestionteclas.h"
#include "gestioncomandos.h"
#include "nodos.h"
#include "string.h"
#include "stdlib.h"
#include "stdio.h"
#include "math.h"

FILE *fscript;  //Acceso global para el fichero script
extern infFichero infoFichero;
extern SoSelection *seleccion;
extern SoWinExaminerViewer *eviewer;
extern int contador=0;  //Almacenaje de las cuentas del Timer.
extern SoTimerSensor *Timer;


infFichero::infFichero(void)
{
  infFichero::altob=0;
  infFichero::altop=0;
  infFichero::anchop=0;
  infFichero::anchob=0;
  infFichero::bpp=0;
  infFichero::flag_leido=0;
  infFichero::numdatos=0;
  infFichero::BitmapOrigen=NULL;
}


datFichero::datFichero()
{
  datFichero::flag_leido=0;
  indicepaleta=new SbColor;
  paletaColores=new SbColor;
  punto=new SbVec3f;
}

/*****************************************************************
*****/
/*  Descompone el fichero de la imagen pasada y que genera 3
archivos  */
/*  temporales que contienen separadamente la cabecera, los datos y
*/
/*  la paleta si es que la imagen es "paletizada"
*/
/*****************************************************************
```

```
    *****/
 58
 59 int descomponer_fichero (FIBITMAP *imagen,char *nombreFichero)
 60 {
 61   unsigned int contador_bytes=0;
 62
 63
 64   FreeImage_Initialise();
 65   FREE_IMAGE_FORMAT fif=FreeImage_GetFileType(nombreFichero,0);
 66
 67   //Evaluación de tipo de archivo
 68   FILE *parchcab=fopen("datoscab.txt","w");
 69   if (parchcab==NULL) { printf("\n\nError al crear el archivo
    datoscab.txt\n"); return(-1); }
 70
 71   fprintf(parchcab,"%d\n",fif); //tipo de fichero.
 72   fprintf(parchcab,"%d\n",FreeImage_GetBPP(imagen));
 73   fprintf(parchcab,"%d\n",FreeImage_GetWidth(imagen));
 74   fprintf(parchcab,"%d\n",FreeImage_GetHeight(imagen));
 75   fprintf(parchcab,"%d\n",FreeImage_GetLine(imagen));
 76   fprintf(parchcab,"%d\n",FreeImage_GetPitch(imagen));
 77   //guardamos el archivo origen de los datos para futuras consultas
 78   fprintf(parchcab,"%s\n",nombreFichero);
 79
 80   printf("\n\nImagen Termográfica Origen de lis datos:
    %s\n",nombreFichero);
 81   printf("Tipo de Bitmap: %s\n",DescripcionTipoFich(fif));
 82   printf("Bits por pixel (BPP): %d\n",FreeImage_GetBPP(imagen));
 83   printf("Ancho de la imagen(en pixels):
    %d\n",FreeImage_GetWidth(imagen));
 84   printf("Alto de la imagen(en pixels):
    %d\n",FreeImage_GetHeight(imagen));
 85   printf("Ancho de la imagen(en bytes):
    %d\n",FreeImage_GetLine(imagen));
 86   printf("Alto de la imagen(en bytes)):
    %d\n",FreeImage_GetPitch(imagen));
 87
 88   fclose(parchcab);  //cierre fichero de información de la cabecera
 89
 90   //generación fichero de datos
 91   FILE *parchdat=fopen("datosimag.txt","w");
 92   //generación fichero de paleta
 93   FILE *parchpal=fopen("datospal.txt","w");
 94
 95   FREE_IMAGE_TYPE tipo_datos= FreeImage_GetImageType(imagen);//tipo
    de datos del bitmap
 96
 97
 98   unsigned int x,y;
 99   int i=0;
100   unsigned char dato_cap=0,temp=0;
101
102
103   if(tipo_datos==FIT_BITMAP)
104   {
105     switch(FreeImage_GetBPP(imagen))
106     {
107     case 1: //Monocromo.
108       for(y = 0; y <(FreeImage_GetHeight(imagen)) ; y++)
109         {
110         //puntero al comienzo de los datos
```

```
111         unsigned char *bytes_linea = (unsigned char
    *)FreeImage_GetScanLine(imagen, y);
112         if(y!=0) //scanline completada
113         fprintf(parchdat,"\n");  //comienzo nueva línea
114
115           for(x = 0; x < FreeImage_GetWidth(imagen); x++)
    //recorremos toda la scanline
116           {
117             //capturando cada nuevo dato de la imagen
118             if(contador_bytes!=FreeImage_GetWidth(imagen))
119             {
120             temp=bytes_linea[x] & (0x80);
121             temp=temp>>7;
122             fprintf(parchdat,"%d ",temp);
123             contador_bytes++;
124             }
125             if(contador_bytes!=FreeImage_GetWidth(imagen))
126             {
127             temp=bytes_linea[x] & (0x40);
128             temp=temp>>6;
129             fprintf(parchdat,"%d ",temp);//  bit1 =pixel1
130             contador_bytes++;
131             }
132             if(contador_bytes!=FreeImage_GetWidth(imagen))
133             {
134             temp=bytes_linea[x] & (0x20);
135             temp=temp>>5;
136             fprintf(parchdat,"%d ",temp);//bit2 =pixel 2
137             contador_bytes++;
138             }
139             if(contador_bytes!=FreeImage_GetWidth(imagen))
140             {
141             temp=bytes_linea[x] & (0x10);
142             temp=temp>>4;
143             fprintf(parchdat,"%d ",temp);//  bit3 =pixel3
144             contador_bytes++;
145             }
146             if(contador_bytes!=FreeImage_GetWidth(imagen))
147             {
148             temp=bytes_linea[x] & (0x08);
149             temp=temp>>3;
150             fprintf(parchdat,"%d ",temp);//  bit4 =pixel4
151             contador_bytes++;
152             }
153             if(contador_bytes!=FreeImage_GetWidth(imagen))
154             {
155             temp=bytes_linea[x] & (0x04);
156             temp=temp>>2;
157             fprintf(parchdat,"%d ",temp);//  bit5 =pixel5
158             contador_bytes++;
159             }
160             if(contador_bytes!=FreeImage_GetWidth(imagen))
161             {
162             temp=bytes_linea[x] & (0x02);
163             temp=temp>>1;
164             fprintf(parchdat,"%d ",temp);//  bit6 =pixel6
165             contador_bytes++;
166             }
167             if(contador_bytes!=FreeImage_GetWidth(imagen))
168             {
169             temp=bytes_linea[x] & (0x01);
```

```
170             fprintf(parchdat,"%d ",temp);//  bit7 =pixel7
171             contador_bytes++;
172             }
173           }
174           contador_bytes=0;//al terminar una línea de la imagen
175         }
176         fclose(parchdat); //cierre del fichero de datos de pixeles
177     fprintf(parchpal,"MONOCROMO");  //indicación tipo paleta
178     fclose(parchpal);
179     break;//fin generación archivos imágenes monocromo
180
181     case 4: //4 BPP
182         for(y = 0; y <(FreeImage_GetHeight(imagen)) ; y++)
183         {
184         unsigned char *bits = (unsigned char
    *)FreeImage_GetScanLine(imagen, y);
185         if(y!=0) //scanline completado
186         fprintf(parchdat,"\n");  //nueva línea del fichero
187
188           for(x = 0; x < FreeImage_GetWidth(imagen); x++)
189             {
190               //el nibble alto representa el indice a la paleta de
    colores del pixel de mas peso
191               if(contador_bytes!=FreeImage_GetWidth(imagen))
    //evitamos los nibbles de relleno
192                 {
193                 temp=bits[x] & 0xf0; //nibble alto
194                 temp=temp>>4;
195                 fprintf(parchdat,"%d ",temp);
196                 contador_bytes++;
197                 }
198               if(contador_bytes!=FreeImage_GetWidth(imagen))
    //evitamos los nibbles de relleno
199                 {
200                 temp=bits[x] & 0x0f; //nibble bajo
201                 fprintf(parchdat,"%d ",temp);
202                 contador_bytes++;
203                 }
204             }
205           contador_bytes=0;
206         }
207         fclose(parchdat); //cierre del fihcero de datos de la
    imagen
208         //para guardar paleta de la imagen
209           if(FreeImage_GetPalette(imagen)!=NULL) //si existe paleta
210           {
211             RGBQUAD *paleta=FreeImage_GetPalette(imagen);
    //puntero al comienzo de la paleta
212             for(i=0;i<16;i++) //4bpp=16 colores
213             {
214               fprintf(parchpal,"%d ",paleta[i].rgbRed);
215               fprintf(parchpal,"%d ",paleta[i].rgbGreen);
216               fprintf(parchpal,"%d\n",paleta[i].rgbBlue);
217             }
218           }
219         fclose(parchdat);
220         fclose(parchpal);
221       break; //fin generacion archivos para imágenes 4bpp
222
223     case 8: //8 BPP
224         for(y = 0; y <(FreeImage_GetHeight(imagen)) ; y++)
```

```
225         {
226         if(y!=0)
227         fprintf(parchdat,"\n");  //comenzamos una nueva linea en el
    fichero
228         unsigned char *bits = (unsigned char
    *)FreeImage_GetScanLine(imagen, y);
229         for(x = 0; x < FreeImage_GetWidth(imagen); x++)
230           {
231             //cada byte representa el índice a la paleta
232           fprintf(parchdat,"%d ",bits[x]);
233           }
234         }
235         fclose(parchdat);  //cierre fichero de datos
236
237         if(FreeImage_GetPalette(imagen)!=NULL)
238           {
239             RGBQUAD *paleta=FreeImage_GetPalette(imagen);
    //obtenemos posición
240             for(i=0;i<256;i++) //8bpp=256 colores
241             {
242               fprintf(parchpal,"%d ",paleta[i].rgbRed);
243               fprintf(parchpal,"%d ",paleta[i].rgbGreen);
244               fprintf(parchpal,"%d\n",paleta[i].rgbBlue);
245             }
246           }
247         fclose(parchpal);
248       break; //fin generación ficheros imágenes 8bpp
249
250     case 24:   //24 BPP
251             //cada componente de color RGB guardada en 3 bytes.
252       for(y = 0; y <(FreeImage_GetHeight(imagen)) ; y++)
253         {
254         if(y!=0)
255         fprintf(parchdat,"\n");  //nueva linea en el fichero
256
257         unsigned char *bytes_linea = (unsigned char
    *)FreeImage_GetScanLine(imagen, y);
258
259         for(x = 0; x < FreeImage_GetWidth(imagen); x++)
260           {
261             if(contador_bytes<FreeImage_GetWidth(imagen))
    //evitamos bytes de relleno
262             {   //notación little Endian (BGR)!!!
263             fprintf(parchdat,"%d ",bytes_linea[x*3+2]);
    //componente B
264             fprintf(parchdat,"%d ",bytes_linea[x*3+1]); //G
265             fprintf(parchdat,"%d ",bytes_linea[x*3]); //R
266             contador_bytes++;
267             }
268           }
269           contador_bytes=0; //restauración contador de bytes
270         }
271       fclose(parchdat);
272       fprintf(parchpal,"24BITS"); //indicación de 24bpp
273       fclose(parchpal);
274     break; //fin generación archivos imágenes de >16 mill.color
275     }
276   }
277   return(0);
278 }
279
```

```
280 /*****************************************************************
    *****/
281 /* Devuelve un objeto de tipo infFichero que contiene la
    información   */
282 /* de cabecera del fichero.
    */
283 /*****************************************************************
    *****/
284 class infFichero *cargar_info(char *nombreFichero)
285 {
286
287   //para almacenar la informacion referente al fichero (bitmap o
    binario)
288   infFichero *infoFichero=new infFichero();
289
290   //identificamos del tipo de fichero
291   if(strstr(nombreFichero,".bin")==NULL)
292   {
293     //ficheros de mapas de bits
294     FILE *pdatoscab=fopen(nombreFichero,"r");
295     if (pdatoscab==NULL)
296     {
297       printf("\nError al abrir el archivo de datos de cabecera\n");
298       infoFichero->flag_leido=0; //información inválida
299       return(infoFichero);  //error
300     }
301     fscanf(pdatoscab,"%d",&infoFichero->tipoFichero);
302     fscanf(pdatoscab,"%d",&infoFichero->bpp);
303     fscanf(pdatoscab,"%ld",&infoFichero->anchop);
304     fscanf(pdatoscab,"%ld",&infoFichero->altop);
305     fscanf(pdatoscab,"%ld",&infoFichero->anchob);
306     fscanf(pdatoscab,"%ld",&infoFichero->altob);
307
308     char dfdf[150]; //se guarda el
309     fscanf(pdatoscab,"%s",&dfdf);//nombre del archivo
310     infoFichero->BitmapOrigen.sprintf("%s",&dfdf);//origen de los
    datos
311
312     fclose(pdatoscab);  //cierre fichero de datos de la cabecera
313     infoFichero->flag_leido=1;//datos contenidos correctos
314
315   }
316   else //archivos binarios
317   {
318     FILE *pdatoscab=fopen(nombreFichero,"rb");
319     if(pdatoscab==NULL)
320     {
321       printf("\nError al abrir el archivo binario\n");
322       infoFichero->flag_leido=0; //información inválida
323       return(infoFichero);//error
324     }
325     int temp=0;
326     //identificación de fichero binario con el valor 255
327     infoFichero->tipoFichero=255;
328     //paleta por defecto
329     infoFichero->bpp=8;
330     fread(&infoFichero->anchop,2,1,pdatoscab);  //ancho
331
332     fread(&infoFichero->altop,2,1,pdatoscab);  //alto
333
334     infoFichero->anchob=infoFichero->anchop*2;
```

```
335     infoFichero->altob=infoFichero->altop*2;
336     fclose(pdatoscab);
337     infoFichero->flag_leido=1;// datos contenidos correctos
338   }
339   return(infoFichero); //sin errores
340 }
341
342 /*****************************************************************
    *****/
343 /*  Abre el archivo bitmap y identifica el formato del mismo. Si el
    */
344 /*  formato esta soportado devuelve un puntero para la lectura de
    la   */
345 /*  información y datos del archivo, en caso contrario devuleve
    NULL   */
346 /*****************************************************************
    *****/
347
348 FIBITMAP *abrir_bitmap(char *nombreFichero)
349 {
350   //deducción del formato por la signatura del mismo
351   FREE_IMAGE_FORMAT fif=FIF_UNKNOWN;
352   fif=FreeImage_GetFileType(nombreFichero,0);
353   //evaluación del tipo de archivo
354   if(fif==FIF_UNKNOWN) //si la signatura no es conocida
355   { //se intenta obtener el formato por medio de la extensión del
    archivo
356     fif=FreeImage_GetFIFFromFilename(nombreFichero);
357   }
358   //Si se ha identificado el formato y se tiene la capacidad de
    leerlo
359   {
360   FIBITMAP *imagen=FreeImage_Load(fif,nombreFichero,0);
361   return imagen;
362   }
363   //si el formato es desconocido o su lectura no esta soportada
364   return NULL; //devolvemos nulo.
365 }
366
367 /*****************************************************************
    *****/
368 /*  Carga los datos de la imagen en una estructura que contiene los
    */
369 /*  pixeles que conforman la imagen y los índices de a la paleta.
    */
370 /*  En caso de forzar_datos =1 se forzara a tratar las
    */
371 /*  imágenes de 24bpp como imágenes en escala de grises, en este
    caso  */
372 /*  sí se calculará la temperatura asociada a cada pixel de la
    imagen. */
373 /*****************************************************************
    *****/
374
375 class datFichero *cargar_datos(char *nombreFichero,int
    usarpaleta,SbColor colordefecto,float TempMax,float TempMin, bool
    forzar_datos)
376 {
377   int byte=0;
378   long i=0,j=0;
379   int separacion_puntos=1; //separación por defecto
```

```
380   int direccion=0; //dirección de lectura de los bytes N-O
381   int byte1=0,byte2=0,byte3=0;
382
383   if ( strstr(nombreFichero,".bin")==NULL ) //si archivo no es
    binario
384   {
385     infFichero *infoFichero=cargar_info("datoscab.txt"); //apertura
    archivo cabecera
386
387     if ( (infoFichero==NULL) || (infoFichero->flag_leido==0) )
388     {
389       printf("\nError cabecera no leida.Se debe de leer primero la
    cabecera.\n");
390       return(NULL); //Error
391     }
392
393     //Lectura correcta de la cabecera
394     //Creacion de nuevo objeto de tipo datFichero
395     datFichero *datosFichero=new datFichero();
396
397     //abrimos el archivo de datos
398     FILE *pdatosimag=fopen(nombreFichero,"r");
399     if (pdatosimag==NULL)
400     {
401       printf("\nError al abrir el archivo de datos\n");
402       return(NULL);  //error
403     }
404
405     //para contener los puntos de la representacion
    (nube/malla/mallat) de la imagen
406     datosFichero->punto=new
    SbVec3f[(int)(infoFichero->anchop)*(int)(infoFichero->altop)];
407
408     //vector para almacenar la paleta aplicada a la imagen
409     datosFichero->paletaColores=new SbColor[
    pow(2,infoFichero->bpp)]; //
410
411      //paleta para almacenar colores guardados segun el valor leído
    en el fichero de datos
412     datosFichero->indicepaleta= new
    SbColor[infoFichero->anchop*infoFichero->altop];  //la creamos del
    mismo tamaño que datos hay en el fichero
413
414     SbColor *paleta;
415
416     //carga paleta del archivo
417     if (infoFichero->bpp<=8) //1, 4, 8bpp
418     {
419       paleta=cargar_Paleta("datospal.txt",infoFichero->bpp);
    //cargamos paleta original
420       datosFichero->paletaColores=paleta;
421       if (paleta==NULL)
422       {
423         printf("\nNo se puede cargar la paleta de colores.\n");
424         printf("Se usara el color por defecto\n");
425       }
426       for (unsigned long u=0;u< pow(2,infoFichero->bpp);u++)
427               datosFichero->paletaColores[u]=paleta[u];
428     }
429     //forzar a 24 bpp como tonos de gris
430    if (forzar_datos==1)
```

```
431     {
432      if (infoFichero->bpp==24)
433      {
434        paleta=generar_paleta(0,255,256,"RGB"); //generación paleta progresiva de grises
435        if (paleta==NULL)
436        {
437          printf("\nNo se ha podido crear la paleta para la imagen de 24bpp.\n");
438          printf("Se usara el color por defecto\n");
439        }
440      }
441     }
442
443
444   if(infoFichero->bpp!=24)
445   {
446     for(i=0;i< (infoFichero->altop );i++)
447     {
448     for(j=0;j< (infoFichero->anchop );j++)
449       {
450         fscanf(pdatosimag,"%d",&byte); //lectura índice de color de cada pixel
451         //enlace color de ese pixel en el array necesario para el nodo SoMaterial
452         if( (paleta==NULL) || (usarpaleta==0) )
453
datosFichero->indicepaleta[i*infoFichero->anchop+j]=colordefecto;
454         else
455
datosFichero->indicepaleta[i*infoFichero->anchop+j].setValue(paleta[byte]);
456           datosFichero->punto[i*infoFichero->anchop+j][0]= (float)(j*separacion_puntos); //coordenada X del punto
457             //corrección para la dirección de lectura/representación
458             if(direccion==0) //direccion SO-NE
459
{datosFichero->punto[i*infoFichero->anchop+j][1]=(float)(i*separacion_puntos) ;}  //coord. Y
460             else //NO-SE
461
{datosFichero->punto[i*infoFichero->anchop+j][1]=(float) ((infoFichero->altop-i)*separacion_puntos) ;}
462             //calculo temperatura según valor leído
463             double temppixel=
(TempMax-TempMin)/(pow(2,infoFichero->bpp)-1);
464             temppixel=temppixel*byte+TempMin;
465
datosFichero->punto[i*infoFichero->anchop+j][2]=(float)temppixel;
466       }
467     }
468   fclose(pdatosimag);
469   datosFichero->flag_leido=1;
470   return(datosFichero);
471   }
472   else  //imagen 24bpp
473   {
474     for(i=0;i<infoFichero->altop;i++)
475     {
476       for(j=0;j<infoFichero->anchop;j++)  //notación little Endian
```

```
477       { //lectura directa sin bytes de relleno
478         int byte1=0,byte2=0,byte3=0;
479         fscanf(pdatosimag,"%d",&byte1);  //R
480         fscanf(pdatosimag,"%d",&byte2);
481         fscanf(pdatosimag,"%d",&byte3);
482
483         if(forzar_datos==1)
484         {
485           //se asume que la imagen está en escala de grises
486           if( (paleta==NULL) || (usarpaleta==0) )
487
    datosFichero->indicepaleta[i*infoFichero->anchop+j]=colordefecto;
488           else
489
    datosFichero->indicepaleta[i*infoFichero->anchop+j].setValue(paleta
    [byte1]);
490         }
491         else //caso contrario imagen =imagen no termográfica
492
    datosFichero->indicepaleta[i*infoFichero->anchop+j].setValue(
    (float)(byte1/255.0),(float)(byte2/255.0),(float)(byte3/255.0) );
493         datosFichero->punto[ (i*infoFichero->anchop+j)][0]=
    (float)(j*separacion_puntos);
494
495         if(direccion==0)
496         {datosFichero->punto[ (i*infoFichero->anchop+j)][1]=
    (float)(i*separacion_puntos) ;}  //coord. Y
497         else //NO-SE
498         {datosFichero->punto[ (i*infoFichero->anchop+j)][1]=
    (float) ((infoFichero->altop-i)*separacion_puntos) ;}
499
500         if(forzar_datos==1) //extracción de datos imagen 24bpp
501         {
502           double temppixel= (TempMax-TempMin)/(255.0);
503           temppixel=temppixel*byte1+TempMin;
504
    datosFichero->punto[i*infoFichero->anchop+j][2]=(float)temppixel;//
    calculo temperatura
505         }
506         else //caso contrario
507           datosFichero->punto [ (i*infoFichero->anchop+j)][2]= 1;
    //coord. Z=0
508
509       }
510     }
511   }
512   fclose(pdatosimag);
513   datosFichero->flag_leido=1;
514   return(datosFichero);
515    }//fin bitmaps
516
517    else if ( (strstr(nombreFichero,".bin")!=NULL) )//binarios
518    {
519        infFichero *infoFichero=cargar_info(nombreFichero);
520      //carga datos de cabecera
521     if ( (infoFichero==NULL) || (infoFichero->flag_leido==0) )
522     {
523       printf("\nError cabecera no leida.Se debe de leer primero la
    cabecera.\n");
524       return(NULL); //Error
525     }
```

```
526
527     //Lectura correcta
528     datFichero *datosFichero=new datFichero();
529
530     //apertura datos
531     FILE *pdatosimag=fopen(nombreFichero,"rb");
532     if (pdatosimag==NULL)
533     {
534       printf("\nError al abrir el archivo de datos binario\n");
535       return(NULL);  //error
536     }
537
538     //array para vértices
539     datosFichero->punto=new
    SbVec3f[(int)(infoFichero->anchop)*(int)(infoFichero->altop)];
    //=número de pixels
540     datosFichero->indicepaleta= new
    SbColor[infoFichero->anchop*infoFichero->altop];
541
542     fseek(pdatosimag,4,SEEK_SET);//posicionamiento en primer dato
    de temperatura
543     float dfloat;
544       for(i=0;i< (infoFichero->altop );i++)
545     {
546        for(j=0;j< (infoFichero->anchop );j++)
547       {
548         datosFichero->punto[i*infoFichero->anchop+j][0]= (float)
    (j*separacion_puntos); //coordenada X del punto
549         datosFichero->punto[i*infoFichero->anchop+j][1]=(float)
    (i*separacion_puntos) ;  //coord. Y
550         //lectura temperatura
551         fread(&dfloat,4,1,pdatosimag);  //altop
552         datosFichero->punto[i*infoFichero->anchop+j][2]=dfloat;
553
    datosFichero->indicepaleta[i*infoFichero->anchop+j]=colordefecto;
    //color por defecto
554       }
555     }
556     fclose(pdatosimag);
557       datosFichero->flag_leido=1; //datos válidos
558      return(datosFichero);
559    }
560   return(NULL);
561 }
562
563
564
565 /*****************************************************************
    *****/
566 /* Crea una paleta de colores guardando los datos del archivo
    */
567 /* nombreFichero de forma secuencial hasta alcanzar el tamaño
    indicado */
568 /* por 2^bpp,retornando el resultado. Devuelve NULL en caso
    contrario  */
569 /* ( archivo con menos datos que 2^bpp) o error en el acceso al
    archivo*/
570 /* nombreFichero.
    */
571 /*****************************************************************
    *****/
```

```
572
573 class SbColor *cargar_Paleta(char *nombreFichero,unsigned char bpp)
574 {
575
576   long i=0;
577   int byte=0;
578   FILE *pdatospal=fopen(nombreFichero,"r");
579   if (pdatospal==NULL)
580   {
581     printf("\nError al abrir el archivo de paleta de color\n");
582     return(NULL);//exit
583   }
584
585   SbColor *paleta; //array para contener paleta
586
587     if(bpp==1) //monocromo
588     {
589       paleta= new SbColor[2];
590       paleta[0].setValue(0,0,0);   //blanco
591       paleta[1].setValue(1,1,1);   //negro
592       fclose(pdatospal);  //cierre del fichero
593     }
594
595     if(bpp==4)  //imagen de 4bits
596     {
597       paleta= new SbColor[16];  //paleta 16 colores
598       for(i=0;i<16;i++)
599       {
600         fscanf(pdatospal,"%d",&byte);  //componente R
601         paleta[i][0]=(float)(byte/255.0);  //normalización
602         fscanf(pdatospal,"%d",&byte);
603         paleta[i][1]=(float)(byte/255.0);
604         fscanf(pdatospal,"%d",&byte);
605         paleta[i][2]=(float)(byte/255.0);
606
607       }
608       fclose(pdatospal);
609     }
610
611     if(bpp==8)
612     {
613       paleta= new SbColor[256];
614       for(i=0;i<256;i++)
615       {
616         fscanf(pdatospal,"%d",&byte);  //componente R
617         paleta[i][0]=(float)(byte/255.0);
618         fscanf(pdatospal,"%d",&byte);
619         paleta[i][1]=(float)(byte/255.0);
620         fscanf(pdatospal,"%d",&byte);
621         paleta[i][2]=(float)(byte/255.0);
622
623       }
624       fclose(pdatospal);
625     }
626   return(paleta);
627 }
628
629 /*****************************************************************
    *****/
630 /*  Devuelve una constante que ídentifica el formato de fichero
    bitmap.*/
```

```
631 /*  En caso de ser un formato no soportado devuelve -1
    */
632 /*****************************************************************
    *****/
633
634 const char *DescripcionTipoFich (FREE_IMAGE_FORMAT fif)
635 {
636   switch(fif)
637   {
638   case -1:
639     return("Desconocido (No Soportado)");
640   break;
641   case 0:
642     return("Archivo Bitmap de Windows o OS/2 (*.BMP)");
643   break;
644   case 1:
645     return("Windows Icon (*.ICO)");
646   break;
647   case 10:
648     return("ZSoft Paintbrush PCX bitmap format (*.PCX)");
649   break;
650   case 2:
651     return("Independent JPEG Group (*.JPG,*.JIF,*.JPEG,*.JPE");
652   break;
653   case 13:
654     return("Portable Network Graphics");
655   break;
656   case 17:
657     return("Truevision Targa Files (*.TGA,*.TARGA)");
658   break;
659   case 18:
660     return("Tagged Image File Format (*.TIF,*.TIFF");
661   break;
662   case 25:
663     return("Graphics Interchange Format (*.GIF)");
664   break;
665   case 30:
666     return("JPEG-2000 Codestream (*.J2K,*.J2C)");
667   break;
668   case 31:
669     return("JPEG-2000 File Format (*.JP2");
670   break;
671   }
672 }
673
674 /*****************************************************************
    *****/
675 /*  Función llamada mediante el comando script. Abre un fichero de
    */
676 /*  script, programa y activa el timer para la lectura de los
    comandos.*/
677 /*****************************************************************
    *****/
678
679
680 int leer_script(char *nombre,float retardo, SoSeparator *nraiz)
681 {
682   //primero buscamos el nodo Timer para establecer el intervalo de
    tiempo
683   //al deseado
684   if(Timer==NULL)
```

```
685   {
686     printf("\nError.Nodo del timer no encontrado\n");
687       return(-1);
688   }
689
690   fscript=fopen(nombre,"r");
691
692   if(fscript==NULL)
693   {
694     printf("\n\nError. no se puede abrir el archivo %s\n",nombre);
695     return(-2);
696   }
697
698   //correcto:
699   Timer->setInterval(retardo); //intervalo en segundos
700   Timer->schedule();  //activación
701 }
702
703 /*****************************************************************
    *****/
704 /*  Función de atención al nodo SoTimer. Ejecuta  periódica  y
    */
705 /*  secuencialmente los comandos contenidos en el fichero script
    */
706 /*****************************************************************
    *****/
707
708 void callbackTimer (void * userdata,SoSensor *)
709 {
710   char *cadena;
711   char caracter=0;
712   static unsigned char salir;
713   static long posicionArchivo=0; //almacenamiento posición del
    archivo script
714
715   SoSeparator *nraiz=(SoSeparator *) userdata;
716
717   //nuevo buffer para pulsaciones de teclado
718   bufferteclas nuevoBuffer;
719
720   fseek(fscript,posicionArchivo,SEEK_SET);  //establece posición de
    lectura
721   salir=0; //control de ejecución
722
723    while( (feof(fscript)==0)&&(salir==0)  )
724       {
725         caracter=fgetc(fscript);
726         if ( (caracter!=';') && (caracter!=10) && (
    feof(fscript)==0) )
727         {
728           nuevoBuffer.add_caracter(caracter);
729         }
730         if  ( (caracter==';') && (caracter!=10) && (
    feof(fscript)==0) )
731         {
732           nuevoBuffer.add_caracter(13);
733           ccomando *nuevo_comando=ordenar_comando(
    nuevoBuffer.get_cadena() ); //trata la cadena. Separa arguementos
734           posicionArchivo=ftell(fscript);  //almacenamiento
    posición actual
735           salir=1; //leída (¿ejecutada?)una línea
```

```
736
737             if(nuevo_comando==NULL)   //si no es un comando válido
738               {
739               printf("Error en posicion %d. Comando %s invalido o
    numero de argumentos incorrecto\n",posicionArchivo);
740               }
741             else  //gestión del comando
742             {
743               gestionar_comando(nuevo_comando,nraiz);  //Evaluación
    de los argumentos
744               printf("%s\n",nuevoBuffer.get_cadena());
745             }
746           }
747         }
748       if(feof(fscript)!=0)
749       {
750        Timer->unschedule(); //desactivación del timer*/
751        fclose(fscript);
752        posicionArchivo=0; //restitución de la posición
753        salir=1;
754       }
755 }
```

```
 1 #include "stdAfx.h"
 2 #include <Inventor/nodes/SoSeparator.h>
 3 #include <Inventor/nodes/SoQuadMesh.h>
 4 #include <Inventor/nodes/SoSelection.h>
 5 #include <Inventor/nodes/SoIndexedFaceSet.h>
 6 #include <Inventor/nodes/SoInfo.h>
 7 #include <Inventor/nodes/SoTriangleStripSet.h>
 8 #include <Inventor/nodes/SoDrawStyle.h>
 9 #include <Inventor/Win/viewers/SoWinExaminerViewer.h>
10 #include <Inventor/nodes/SoShapeHints.h>
11 #include <Inventor/nodes/SoCoordinate3.h>
12 #include <Inventor/nodes/SoMaterial.h>
13 #include <Inventor/nodes/SoBaseColor.h>
14 #include <Inventor/nodes/SoMaterialBinding.h>
15 #include <Inventor/nodes/SoTransform.h>
16 #include <Inventor/nodes/SoIndexedLineSet.h>
17 #include <Inventor/nodes/SoLineSet.h>
18 #include <Inventor/nodes/SoPointSet.h>
19 #include <Inventor/nodes/SoRotationXYZ.h>
20 #include <Inventor/nodes/SoFont.h>
21 #include <Inventor/nodes/SoText3.h>
22 #include <Inventor/nodes/SoNormal.h>
23 #include <Inventor/nodes/SoNormalBinding.h>
24 #include <Inventor/nodes/SoText2.h>
25 #include <Inventor/nodes/SoTranslation.h>
26 #include <Inventor/nodes/SoRotation.h>
27 #include <Inventor/nodes/SoSpotLight.h>
28 #include <Inventor/nodes/SoOrthographicCamera.h>
29 #include <Inventor/manips/SoSpotLightManip.h>
30 #include <Inventor/actions/SoSearchAction.h>
31 #include <Inventor/manips/SoTransformBoxManip.h>
32 #include <Inventor/manips/SoTrackballManip.h>
33 #include <Inventor/manips/SoDirectionalLightManip.h>
34 #include <Inventor/SbViewVolume.h>
35
36 #include "colores.h"
37 #include "figuras.h"
38 #include "leerfichero.h"
39 #include "gestioncomandos.h"
40 #include "nodos.h"
41 #include "datos.h"
42 #include "math.h"
43 #include "string.h"
44
45 extern infFichero infoFichero;
46 extern datFichero datosFichero;
47 extern SoSelection *seleccion;
48 extern unsigned char capturaPuntos;
49 extern SoWinExaminerViewer *eviewer;
50
51
52 #define  PI 3.14159265389
53 //constructor por defecto objeto ejecoordenadas
54 ejecoordenadas::ejecoordenadas()
55 {
56   posX=0; posY=0; posZ=0;   //posicionamiento del eje de
   coordenadas
57
58   maxvalueX=360;  minvalueX=0;
59   maxvalueY=240;  minvalueY=0;
60   maxvalueZ=100;  minvalueZ=0;
```

```
 61
 62    marcasX=10;
 63    marcasY=5;
 64    marcasZ=4;
 65
 66    colorejeX=new colorRGB;
 67    colorejeX->componenteR=1;
 68    colorejeX->componenteG=0;
 69    colorejeX->componenteB=0;
 70    colorejeY=new colorRGB;
 71    colorejeY->componenteR=0;
 72    colorejeY->componenteG=1;
 73    colorejeY->componenteB=0;
 74    colorejeZ=new colorRGB;
 75    colorejeZ->componenteR=0;
 76    colorejeZ->componenteG=0;
 77    colorejeZ->componenteB=1;
 78
 79    //longitud por defecto de las marcas divisorias
 80    longmarcaX= 10;
 81    longmarcaY= 10;
 82    longmarcaZ= 10;
 83    //tamano de la fuente por defecto
 84    tamanoFuente=20;
 85    separacionEtiqueta=10;
 86
 87    //titulo de los ejes
 88    tituloX=new SbString;
 89    tituloY=new SbString;
 90    tituloZ=new SbString;
 91
 92    tituloX->sprintf("%s","Pixels X");
 93    tituloY->sprintf("%s","Pixels Y");
 94    tituloZ->sprintf("%s","Temp (°C)");
 95  }
 96
 97  //constructor por defecto el objeto tipo informacion
 98  infObjeto::infObjeto(void)
 99  {
100    nodoInformacion=NULL;
101    nombre_grupo=NULL;
102    origen_datos=NULL;
103    posicion.setValue(0,0,0); //posicion por defecto
104    numColumnas=0;
105    numFilas=0;
106      maxValorZ=0;  //almacena los valores máximos del objeto
107    maxValorY=0;  //para cada una
108    maxValorX=0;  //de las coordenadas.
109    minValorX=0;
110    minValorY=0;
111    minValorZ=0;
112    vmedio=0;  //promedio de los valores de Z que componen el objeto
113
114  }
115
116  /*****************************************************************
     *****/
117  /* Función que devuelve el conjunto de nodos que conforman el eje
     de   */
118  /* coordenadas
     */
```

```
119 /*****************************************************************
    *****/
120 SoSeparator *generarEjeCoord (class ejecoordenadas *pejec)
121 {
122
123   int i=0;
124   int LongEtiqueta=0;
125
126   //nodo grupal que contendra el conjunto de nodos que forman el
    eje
127   SoSeparator *SepEjeCoord=new SoSeparator;
128   SoTransform *transforma=new SoTransform;
129   transforma->setName("TransformaEje");
130
131   SoFont *fuente=new SoFont;   //fuente común
132
133   SepEjeCoord->setName("EjeCoord");  //nombre identificativo
134   SepEjeCoord->addChild(transforma);
135
136   //separadores para cada eje
137   SoSeparator *ejeX=new SoSeparator;  //marcas y etiquetas del ejeX
138   SoBaseColor *colorX=new SoBaseColor;
139   SoDrawStyle *estiloX=new SoDrawStyle;
140   SoMaterialBinding *materialEnlaceX=new SoMaterialBinding;
141   SoCoordinate3 *coordenadasX=new SoCoordinate3;
142   SoLineSet *LineaX=new SoLineSet;
143   materialEnlaceX->value=SoMaterialBinding::OVERALL$;
144   ejeX->addChild(materialEnlaceX);
145   ejeX->addChild(estiloX);
146   ejeX->addChild(coordenadasX);
147   ejeX->addChild(colorX);
148   ejeX->addChild(LineaX);
149
150
151   SoSeparator *ejeY=new SoSeparator;  //"                    " ejeY
152   SoBaseColor *colorY=new SoBaseColor;
153   SoMaterialBinding *materialEnlaceY=new SoMaterialBinding;
154   SoDrawStyle *estiloY=new SoDrawStyle;
155   SoCoordinate3 *coordenadasY=new SoCoordinate3;
156   SoLineSet *LineaY=new SoLineSet;
157   ejeY->addChild(materialEnlaceY);
158   materialEnlaceY->value=SoMaterialBinding::OVERALL;
159   ejeY->addChild(estiloY);
160   ejeY->addChild(coordenadasY);
161   ejeY->addChild(colorY);
162   ejeY->addChild(LineaY);
163
164   SoSeparator *ejeZ=new SoSeparator;  //"                    " ejeZ
165   SoBaseColor *colorZ=new SoBaseColor;
166   SoDrawStyle *estiloZ=new SoDrawStyle;
167   SoCoordinate3 *coordenadasZ=new SoCoordinate3;
168   SoMaterialBinding *materialEnlaceZ=new SoMaterialBinding;
169   SoLineSet *LineaZ=new SoLineSet;
170   ejeZ->addChild(materialEnlaceZ);
171   materialEnlaceZ->value=SoMaterialBinding::OVERALL;
172   ejeZ->addChild(estiloZ);
173   ejeZ->addChild(coordenadasZ);
174   ejeZ->addChild(colorZ);
175   ejeZ->addChild(LineaZ);
176
177   SepEjeCoord->addChild(ejeX); //se añade eje X
```

```
178   SepEjeCoord->addChild(ejeY); //idem Y
179   SepEjeCoord->addChild(ejeZ); //para Z
180
181   SbColor colorEje;
182   //color para todos los elementos del eje X (marcas,linea del
    eje,texto...)
183   colorEje.setValue(pejec->colorejeX->componenteR,pejec->colorejeX-
    >componenteG,pejec->colorejeX->componenteB);
184   colorX->rgb.setValue(colorEje);
185
186   //ídem Y
187   colorEje.setValue(pejec->colorejeY->componenteR,pejec->colorejeY-
    >componenteG,pejec->colorejeY->componenteB);
188   colorY->rgb.setValue(colorEje);
189
190   //ídem Z
191   colorEje.setValue(pejec->colorejeZ->componenteR,pejec->colorejeZ-
    >componenteG,pejec->colorejeZ->componenteB);
192   colorZ->rgb.setValue(colorEje);
193
194   //separadores para las etiquetas
195   SoSeparator *SepTextoX=new SoSeparator;
196   SepTextoX->setName("SepTextoX");
197   SepTextoX->addChild(fuente);
198
199   SoSeparator *SepTextoY=new SoSeparator;
200   SepTextoY->setName("SepTextoY");
201   SepTextoY->addChild(fuente);
202
203   SoSeparator *SepTextoZ=new SoSeparator;
204   SepTextoZ->setName("SepTextoZ");
205   SepTextoZ->addChild(fuente);
206
207   ejeX->addChild(SepTextoX);
208   ejeY->addChild(SepTextoY);
209   ejeZ->addChild(SepTextoZ);
210
211   fuente->size=pejec->tamanoFuente; //tamaño fuente común
212
213   SoText3 *ptexto;
214   SbString *plabels;   //facilita la manipulación de cadenas
215   SoRotationXYZ *protacion; // necesario para la correcta
    orientación de las etiquetas
216   SoTranslation *ptranslacion;  //para posicionarlas en el sitio
    indicado
217   SoGroup *SepEtiqueta;
218
219   //posicionamiento etiquetas eje X
220   double incrementosX=(double) ((pejec->maxvalueX -
    pejec->minvalueX)/(pejec->marcasX)); //contiene la distancia entre
    cada marca del eje Y
221   for(i=0;i<pejec->marcasX+2;i++)
222   {
223     SepEtiqueta=new SoGroup;
224     ptexto=new SoText3;
225     plabels=new SbString;
226     ptranslacion=new SoTranslation;
227     protacion=new SoRotationXYZ;
228     ptexto->justification=(SoText3::CENTER);
229
230     //etiquetas para cada una de las marcas del eje X
```

```
231     if(i<pejec->marcasX+1)
232       {
233       plabels->sprintf("%.f",(float) ( pejec->minvalueX) +i *
    incrementosX);
234       LongEtiqueta=strlen( plabels->getString() );
235       ptexto->string=plabels->getString();
236       }
237     if (i==pejec->marcasX+1)
238       {
239       //etiqueta del titulo del eje X
240       ptexto->justification=(SoText3::LEFT); //alineación
241       ptexto->string=pejec->tituloX->getString();
242       LongEtiqueta=strlen( pejec->tituloX->getString() );
243       }
244
245     //Posicionamiento de las etiquetas en su lugar correspondiente
246     if(i==0)
247     {
248       ptranslacion->translation.setValue(SbVec3f
249           (pejec->posX,
250            pejec->posY - pejec->longmarcaX
    -LongEtiqueta-pejec->tamanoFuente-pejec->separacionEtiqueta,
251            pejec->posZ));
252     }
253     else
254     {
255
    ptranslacion->translation.setValue(SbVec3f((float)incrementosX,0,0)
    );
256     }
257     SepEtiqueta->addChild(ptranslacion);
258     SepEtiqueta->addChild(protacion);
259     SepEtiqueta->addChild(ptexto);
260     SepTextoX->addChild(SepEtiqueta);
261   }
262
263   //posicionamiento etiquetas eje Y
264   double incrementosY=double ((pejec->maxvalueY -
    pejec->minvalueY)/(pejec->marcasY)); //distancia entre marcas del
    eje
265   for(i=0;i<pejec->marcasY+2;i++)
266   {
267     ptexto=new SoText3;  //creamos nuevo texto3d
268     plabels=new SbString;  //y un nuevo objeto tipo string
269     ptranslacion=new SoTranslation; //lo mismo para las
    translaciones
270     protacion=new SoRotationXYZ; //y las rotaciones
271     ptexto->justification=(SoText3::CENTER);
272
273     if(i<pejec->marcasY+1) //si se trata de las etiquetas de las
    marcas divisorias
274       {
275       plabels->sprintf("%.f",(float) ((pejec->minvalueY)+i *
    incrementosY) );
276       LongEtiqueta=strlen( plabels->getString() );
277       ptexto->string=plabels->getString();
278       }
279
280     if (i==pejec->marcasY+1)  //máximo.
281       {
282
```

```
283       //etiqueta titulo del eje Y
284       ptexto->justification=(SoText3::RIGHT); //ajustado a la
    derecha
285       ptexto->string=pejec->tituloY->getString(); //establecemos el
    titulo del eje Y
286       }
287
288     //Posicionamiento de las etiquetas en su lugar correspondiente
289     if(i==0)
290     {
291     ptranslacion->translation.setValue(SbVec3f
292           (pejec->posX-pejec->longmarcaY-
    LongEtiqueta-pejec->separacionEtiqueta,
293            pejec->posY,
294            pejec->posZ) );
295     }
296     else
297     {
298
    ptranslacion->translation.setValue(SbVec3f(0,(float)incrementosY,0)
    );
299     }
300     //se añaden las etiquetas
301     SepTextoY->addChild(ptranslacion);
302     SepTextoY->addChild(protacion);
303     SepTextoY->addChild(ptexto);
304   }
305
306   //posicionamiento de las etiquetas del eje Z
307   double incrementosZ= ((pejec->maxvalueZ -
    pejec->minvalueZ)/(pejec->marcasZ));
308   for(i=0;i<pejec->marcasZ+2;i++)
309   {
310     ptexto=new SoText3;
311     plabels=new SbString;
312     ptranslacion=new SoTranslation;
313     protacion=new SoRotationXYZ;
314     ptexto->justification=(SoText3::CENTER);
315
316     //etiquetas para cada una de las marcas divisorias
317     if(i<pejec->marcasZ+1)
318       {
319       plabels->sprintf("%.1f",(float) (pejec->minvalueZ) +i*
    incrementosZ);
320       LongEtiqueta=strlen( plabels->getString() );
321       ptexto->string=plabels->getString();
322       }
323     if (i==pejec->marcasZ+1)
324       {
325       //etiqueta para el titulo del eje Z
326       ptexto->string=pejec->tituloZ->getString();
327       }
328
329     //Posicionamiento de las etiquetas en su lugar correspondiente
330     if(i==0)
331     {
332     protacion->axis=(SoRotationXYZ::X);
333     protacion->angle.setValue((float)(-3*PI/2)); //orientación y
334     ptranslacion->translation.setValue(SbVec3f //posicionamiento
    adecuado
335           (pejec->posX  -pejec->longmarcaZ-
```

```
    LongEtiqueta-pejec->separacionEtiqueta,
336             pejec->posY,
337             pejec->posZ));
338     }
339     else
340     {
341
    ptranslacion->translation.setValue(SbVec3f(0,(float)incrementosZ,0)
    );
342     }
343     //se añaden las etiquetas al nodo separador
344     SepTextoZ->addChild(ptranslacion);
345     SepTextoZ->addChild(protacion);
346     SepTextoZ->addChild(colorZ);
347     SepTextoZ->addChild(materialEnlaceZ);
348     SepTextoZ->addChild(ptexto);
349   }
350
351
352   //array para la línea del eje y cada una de sus marcas
353   SbVec3f *puntosejeX=new SbVec3f[(pejec->marcasX+1)*2+2]; //Para
    cada eje
354   SbVec3f *puntosejeY=new SbVec3f[(pejec->marcasY+1)*2+2]; //cada
    linea de marca necesita 2 vértices
355   SbVec3f *puntosejeZ=new SbVec3f[(pejec->marcasZ+1)*2+2]; //+2 más
    para la línea del eje
356
357 //Construcción "manual" de cada una de las líneas del eje
358   //punto inicial ejeX
359   puntosejeX[0][0]=pejec->posX;
360   puntosejeX[0][1]=pejec->posY;
361   puntosejeX[0][2]=pejec->posZ;
362
363   //punto final ejeX
364   puntosejeX[1][0]=pejec->posX+pejec->maxvalueX;
365   puntosejeX[1][1]=pejec->posY;
366   puntosejeX[1][2]=pejec->posZ;
367
368
369   //incial Y
370   puntosejeY[0][0]=pejec->posX;
371   puntosejeY[0][1]=pejec->posY;
372   puntosejeY[0][2]=pejec->posZ;
373   //final Y
374   puntosejeY[1][0]=pejec->posX;
375   puntosejeY[1][1]=pejec->posY+pejec->maxvalueY;
376   puntosejeY[1][2]=pejec->posZ;
377
378   //incial Z
379   puntosejeZ[0][0]=pejec->posX;
380   puntosejeZ[0][1]=pejec->posY;
381   puntosejeZ[0][2]=pejec->posZ;
382   //final Z
383   puntosejeZ[1][0]=pejec->posX;
384   puntosejeZ[1][1]=pejec->posY;
385   puntosejeZ[1][2]=pejec->posZ+pejec->maxvalueZ;
386
387   //líneas de marca para el ejeX
388   for(i=0;i<(pejec->marcasX+1);i++)
389   {
390     puntosejeX[2*i+2][0]=pejec->posX + incrementosX*i;
```

```
391       puntosejeX[2*i+2][1]=pejec->posY;
392       puntosejeX[2*i+2][2]=pejec->posZ;
393
394       puntosejeX[2*i+2+1][0]=pejec->posX +  incrementosX*i;
395       puntosejeX[2*i+2+1][1]=pejec->posY - pejec->longmarcaX;
396       puntosejeX[2*i+2+1][2]=pejec->posZ;
397     }
398
399     //para el Y
400     for(i=0;i<(pejec->marcasY+1);i++)
401     {
402       puntosejeY[2*i+2][0]=pejec->posX;
403       puntosejeY[2*i+2][1]=pejec->posY + incrementosY*i;
404       puntosejeY[2*i+2][2]=pejec->posZ;
405
406       puntosejeY[2*i+2+1][0]=pejec->posX  - pejec->longmarcaY;
407       puntosejeY[2*i+2+1][1]=pejec->posY + incrementosY*i;
408       puntosejeY[2*i+2+1][2]=pejec->posZ;
409     }
410
411     // y el Z
412     for(i=0;i<(pejec->marcasZ+1);i++)
413     {
414       puntosejeZ[2*i+2][0]=pejec->posX;
415       puntosejeZ[2*i+2][1]=pejec->posY;
416       puntosejeZ[2*i+2][2]=pejec->posZ + incrementosZ*i;
417
418       puntosejeZ[2*i+2+1][0]=pejec->posX- pejec->longmarcaZ;
419       puntosejeZ[2*i+2+1][1]=pejec->posY;
420       puntosejeZ[2*i+2+1][2]=pejec->posZ + incrementosZ*i;
421     }
422
423
424     //arrays para indicación de los vértices que delimitan las marcas
    de cada uno de los 3 ejes
425     int *verticesX=new int[1+pejec->marcasX+1];  //vertices del eje X
426     int *verticesY=new int[1+pejec->marcasY+1];  //...
427     int *verticesZ=new int[1+pejec->marcasZ+1];
428
429     //indicamos los vértices que forman cada marca  del eje X
430     for (i=0;i<(1+pejec->marcasX+1);i++)
431     {
432       verticesX[i]=2;  //N.C
433     }
434
435     //igual para el Y
436     for (i=0;i<(1+pejec->marcasY+1);i++)
437     {
438       verticesY[i]=2;
439     }
440
441     //y el Z
442     for (i=0;i<(1+pejec->marcasZ+1);i++)
443     {
444       verticesZ[i]=2;
445     }
446
447     //introducción de valores en los nodos correspondientes
448     coordenadasX->point.setValues(0,(pejec->marcasX+1)*2+2,puntosejeX
    );
449     coordenadasY->point.setValues(0,(pejec->marcasY+1)*2+2,puntosejeY
```

```
    );
450   coordenadasZ->point.setValues(0,(pejec->marcasZ+1)*2+2,puntosejeZ
    );
451
452   LineaX->numVertices.setValues(0,1+pejec->marcasX+1,verticesX);
453   LineaY->numVertices.setValues(0,1+pejec->marcasY+1,verticesY);
454   LineaZ->numVertices.setValues(0,1+pejec->marcasZ+1,verticesZ);
455
456   //devolvemos el resultado
457   SepEjeCoord->unrefNoDelete();
458   return SepEjeCoord;
459 }
460
461
462 //FUNCION PARA GENERAR MALLAS
463 SoSeparator *crear_malla(unsigned long nfilas, unsigned long
    ncolumnas,float Z,SbColor color,float separacion,char
    *pnombreMalla)
464 {
465
466   unsigned long i=0,j=0,indice=0;
467   float valor=0;
468   SoSeparator *SepMalla=new SoSeparator;
469   SbVec3f *vertices=new SbVec3f[nfilas*ncolumnas];
470    //recorremos cada una de las coordenadas considerando que la
    distancia (separacion) es la misma
471
472
473   for(i=0;i< (nfilas );i++)
474   {
475     for(j=0;j< (ncolumnas);j++)
476     {
477     vertices[i*ncolumnas+j][0]=(float) (j*separacion); //coordenada
    X del punto
478     vertices[i*ncolumnas+j][1]=(float) (i*separacion) ;  //coord. Y
479     vertices[i*ncolumnas+j][2]=Z ;
480     }
481   }
482
483
484
485   SoCoordinate3 *coordenadas=new SoCoordinate3;
486   coordenadas->point.setValues(0,nfilas*ncolumnas,vertices);
487   coordenadas->setName("Coordenadas");
488
489   SoTransform *transforma=new SoTransform;
490   transforma->setName("Transforma");
491
492   SoMaterial *material=new SoMaterial;
493   SoDrawStyle *estilo=new SoDrawStyle;
494   material->setName("material");
495   SepMalla->addChild(material);
496
497   material->diffuseColor.setValue(color); //color pasado al llamar
    la funcion
498   SoShapeHints *ShapeHints=new SoShapeHints;
499   ShapeHints->vertexOrdering=SoShapeHints::COUNTERCLOCKWISE;
500   SepMalla->addChild(ShapeHints);
501
502   SepMalla->addChild(estilo);
503   estilo->style.setValue(SoDrawStyle::FILLED);
```

```
504   estilo->lineWidth.setValue(5);
505   SepMalla->addChild(transforma);
506
507   //añadimos nodo de las coordenadas
508   SepMalla->addChild(coordenadas);
509   //y creamos una malla
510   SoQuadMesh *nobjeto=new SoQuadMesh;
511   nobjeto->verticesPerRow.setValue(ncolumnas);
512   nobjeto->verticesPerColumn.setValue(nfilas);
513
514   nobjeto->setName("malla");
515   SepMalla->addChild(nobjeto);
516   SepMalla->setName("malla_grupo");
517   //devolvemos el objeto malla
518   SepMalla->unrefNoDelete();
519   return SepMalla;
520 }
521
522
523 /*****************************************************************
    *****/
524 /*  Esta funcion utiliza para crear la malla el nodo SoQuadMesh
    */
525 /*  utilizando el color por defecto ( pasado en formato estándar
    RGB)  */
526 /*****************************************************************
    *****/
527 SoSeparator *generar_malla(char *nombreFichero,unsigned char
    usar_paleta,int compR,int compG,int compB,float TempMax,float
    TempMin,char *pnombreMalla, bool forzar_datos)
528 {
529
530     unsigned long i=0,j=0;
531   float temppixel=0;
532   float dato=0;
533   int direccion=0;  //direccion de lectura N-O(0) o S-E(1)
534
535   SbColor colorPdefecto;//usado en caso de no utilizar la paleta de
    colores
536   colorPdefecto.setValue(SbVec3f(
    (float)(compR/255.0),(float)(compG/255.0),(float)(compB/255.0)) );
537   infFichero *infoFichero;
538   datFichero *datosFichero;
539   SbString *pcadena=new SbString;
540
541   //nombre identificativo
542   SoInfo *nombreMalla=new SoInfo;
543   nombreMalla->setName("Nombre");
544
545   if(nombreMalla!=NULL)
546   nombreMalla->string=pnombreMalla;
547   else
548   nombreMalla->string=NULL;
549
550   SoInfo *ArchivoOrigen=new SoInfo; //almacena la cadena del
    archivo que supone el origen de los
551                                     //de cabecera, datos de la
    imagen.
552   ArchivoOrigen->setName("ArchivoOrigen");
553
554   if( (strstr(nombreFichero,".bin"))==NULL )
```

```
555   {
556     infoFichero=cargar_info("datoscab.txt");
557     if ( (infoFichero==NULL) || (infoFichero->flag_leido==0) )
    //en caso de error
558     {
559       printf("\n¡¡¡Error al abrir el archivo de cabecera!!!\n");
560       return(NULL); //exit
561     }
562     else //cargamos datos
563     {
564
565       datosFichero=cargar_datos
    ("datosimag.txt",usar_paleta,colorPdefecto,TempMax,TempMin,forzar_d
    atos);
566       if(datosFichero->flag_leido==0)
567       {
568         printf("\n¡¡¡Error al intentar cargar los datos\n!!!");
569         return(NULL);
570       }
571     }
572     ArchivoOrigen->string=infoFichero->BitmapOrigen;
573   }
574
575   else if( (strstr(nombreFichero,".bin"))!=NULL ) //para archivos
    binarios
576   {
577     infoFichero=cargar_info(nombreFichero); //cargamos información
    del fichero
578     if ( (infoFichero==NULL) || (infoFichero->flag_leido==0) )
    //en caso de error
579     {
580       printf("\n¡¡¡Error al abrir el archivo de cabecera!!!\n");
581       return(NULL); //exit
582     }
583     else //cargamos datos del fichero
584     {
585       datosFichero=cargar_datos
    (nombreFichero,usar_paleta,colorPdefecto,0,0,forzar_datos);
586       if(datosFichero->flag_leido==0)
587       {
588         printf("\n¡¡¡Error al intentar cargar los datos\n!!!");
589         return(NULL);
590       }
591     }
592     ArchivoOrigen->string.setValue(nombreFichero);  //se guarda el
    archivo de origen de los datos (en este caso binario)
593   }
594
595   //nodos necesarios para la creación de la malla
596   SoSeparator *resultado=new SoSeparator;
597   resultado->ref();
598
599   //se añade la información del archivo de origen
600   resultado->addChild(nombreMalla);
601   resultado->addChild(ArchivoOrigen);
602
603   SoShapeHints *ShapeHints=new SoShapeHints;
604   ShapeHints->vertexOrdering=SoShapeHints::COUNTERCLOCKWISE;
605   resultado->addChild(ShapeHints);
606
607   SoTransform *transforma=new  SoTransform;
```

```
608    transforma->setName("Transforma");
609    SoMaterial *material=new SoMaterial;
610    SoDrawStyle *estilo=new SoDrawStyle;
611    material->setName("material");
612    resultado->addChild(material);
613    resultado->addChild(estilo);
614    estilo->setName("estilo");
615    estilo->lineWidth.setValue(1);
616    resultado->addChild(transforma);
617
618    //enlace para los colores de la malla
619    material->diffuseColor.setValues(0,infoFichero->altop*infoFichero
     ->anchop,datosFichero->indicepaleta);
620
621    SoMaterialBinding *myMaterialBinding = new SoMaterialBinding;
622    myMaterialBinding->value = SoMaterialBinding::PER_VERTEX; //ambas
     caras
623    myMaterialBinding->setName("MaterialEnlace");
624    resultado->addChild(myMaterialBinding);
625
626    SoCoordinate3 *coordenadas=new SoCoordinate3; //para almacenar
     los puntos que forman la malla
627    coordenadas->setName("Coordenadas");
628    SoQuadMesh *malla=new SoQuadMesh;  //el nodo encargado de crear
     la malla
629
630    //establecimiento de los valores en los nodos correspondientes
631    coordenadas->point.setValues(0,infoFichero->anchop*infoFichero->a
     ltop,datosFichero->punto);  //introducimos las coordenadas de la
     malla
632    estilo->lineWidth.setValue(1);  //grosor predeterminadop para la
     línea en la reprsentación alámbrica de la malla
633    estilo->style.setValue(SoDrawStyle::FILLED); // representacion de
     la malla
634    malla->verticesPerRow.setValue(infoFichero->anchop);
635    malla->verticesPerColumn.setValue(infoFichero->altop);
636
637    resultado->addChild(coordenadas);
638    resultado->addChild(malla);
639
640    //nodos de información para Tmax y Tmin
641    SbString *cadena=new SbString;
642    SoInfo *tmin=new SoInfo;
643    SoInfo *tmax=new SoInfo;
644    cadena->sprintf("%f",TempMax);
645    tmax->string=cadena->getString();
646    tmax->setName("Tmax");
647
648    cadena->sprintf("%f",TempMin);
649    tmin->string=cadena->getString();
650    tmin->setName("Tmin");
651
652    resultado->addChild(nombreMalla);
653    resultado->addChild(tmax);
654    resultado->addChild(tmin);
655    malla->setName("malla");
656    resultado->setName("malla_grupo");
657    resultado->unrefNoDelete();
658    return resultado;
659 }
660
```

```
661 /*****************************************************************
    *****/
662 /* Crea un nodo separador para la representación de la nube de
    puntos  */
663 /* de la termografía.
    */
664 /*****************************************************************
    *****/
665 SoSeparator *generar_nube(char *nombreFichero,unsigned char
    usar_paleta,int compR,int compG,int compB,float TempMax,float
    TempMin,char *pnombreNube,bool forzar_datos)
666 {
667
668   unsigned long i=0,j=0;
669
670   SbColor colorPdefecto;//usado en caso de no utiizar la paleta de
    colores
671   colorPdefecto.setValue(SbVec3f(
    (float)(compR/255.0),(float)(compG/255.0),(float)(compB/255.0)) );
672   infFichero *infoFichero;
673   datFichero *datosFichero;
674   SbString *pcadena=new SbString;
675
676   //nodo informacion para contener el origen de la informacion de
    la figura
677   //nodo de inforamción para ponerle un nombre identificativo
    deseado a la malla
678   SoInfo *nombreNube=new SoInfo;
679   nombreNube->setName("Nombre");
680
681   if(nombreNube!=NULL)  // si el argumento pasado al llamar la
    funcion no es NULL
682   nombreNube->string=pnombreNube;
683   else
684   nombreNube->string=NULL;
685
686
687
688
689   SoInfo *ArchivoOrigen=new SoInfo; //almacena la cadena del
    archivo que supone el origen de los
690                       //de cabecera,datos de la imagen
691   ArchivoOrigen->setName("ArchivoOrigen");
692
693   if( (strstr(nombreFichero,".bin"))==NULL )
694   {
695     infoFichero=cargar_info("datoscab.txt");
696     if ( (infoFichero==NULL) || (infoFichero->flag_leido==0) )
    //en caso de error
697     {
698       printf("\n¡¡¡Error al abrir el archivo de cabecera!!!\n");
699       return(NULL); //salimos devolviendo nulo
700     }
701     else //caso contrario cargamos datos
702     {
703         //cargamos los datos del archivo a procesar
704     datosFichero=cargar_datos
    ("datosimag.txt",usar_paleta,colorPdefecto,TempMax,TempMin,forzar_d
    atos);
705     if(datosFichero->flag_leido==0)
706       {
```

```
707           printf("\n¡¡¡Error al intentar cargar los datos\n!!!");
708           return(NULL);//exit
709         }
710       }
711       ArchivoOrigen->string.setValue("datosimag.txt");
712     }
713     else if( (strstr(nombreFichero,".bin"))!=NULL ) //para archivos
    binarios
714     {
715       infoFichero=cargar_info(nombreFichero); //cargamos información
    del fichero
716       if ( (infoFichero==NULL) || (infoFichero->flag_leido==0) )
717       {
718         printf("\n¡¡¡Error al abrir el archivo de cabecera!!!\n");
719         return(NULL); //exit
720       }
721       else //carga de datos del fichero
722       {
723         datosFichero=cargar_datos
    (nombreFichero,usar_paleta,colorPdefecto,0,0,forzar_datos);
724         if(datosFichero->flag_leido==0)
725         {
726           printf("\n¡¡¡Error al intentar cargar los datos\n!!!");
727           return(NULL);
728         }
729       }
730       ArchivoOrigen->string.setValue(nombreFichero);
731     }
732
733     //nodos necesarios
734     SoSeparator *resultado=new SoSeparator;
735
736     resultado->addChild(nombreNube);
737     //se añade el nodo de informacion del origen de los datos
738     resultado->addChild(ArchivoOrigen);
739
740     SoTransform *transforma=new SoTransform;
741     transforma->setName("Transforma");
742     SoMaterial *material=new SoMaterial;
743     SoDrawStyle *estilo=new SoDrawStyle;
744     estilo->setName("estilo");
745     material->setName("material");
746     resultado->addChild(material);
747     resultado->addChild(estilo);
748     estilo->pointSize.setValue(1);
749     estilo->style.setValue(SoDrawStyle::POINTS);
750     resultado->addChild(transforma);
751
752     //enlace para los colores de la nube puntos
753     material->diffuseColor.setValues(0,infoFichero->altop*infoFichero
    ->anchop,datosFichero->indicepaleta);
754     SoMaterialBinding *myMaterialBinding = new SoMaterialBinding;
755     myMaterialBinding->setName("MaterialEnlace");
756     myMaterialBinding->value = SoMaterialBinding::PER_PART;
757     resultado->addChild(myMaterialBinding);
758
759     SoCoordinate3 *coordenadas=new SoCoordinate3;
760     coordenadas->setName("Coordenadas");
761     coordenadas->point.setValues(0,infoFichero->altop*infoFichero->an
    chop,datosFichero->punto);
762
```

```
763    //añadimos nodo de las coordenadas
764    resultado->addChild(coordenadas);
765
766    //nodo contenedor de puntos
767    SoPointSet *nube=new SoPointSet;
768    nube->setName("nube");
769    nube->numPoints.setValue(infoFichero->altop*infoFichero->anchop);
770
771    //para almacenar la información sobre las filas y las columnas de
       la nube
772    SoInfo *nFilas=new SoInfo;
773    SoInfo *nColumnas=new SoInfo;
774
775    pcadena->sprintf("%d",infoFichero->altop);
776    nFilas->string=pcadena->getString();
777    nFilas->setName("numFilas");
778
779    pcadena->sprintf("%d",infoFichero->anchop);
780    nColumnas->string=pcadena->getString();
781    nColumnas->setName("numColumnas");
782
783    resultado->addChild(nColumnas);
784    resultado->addChild(nFilas);
785
786    //nodos de información de temperatura máxima y mínima.
787    SbString *cadena=new SbString;
788    SoInfo *tmin=new SoInfo;
789    SoInfo *tmax=new SoInfo;
790    cadena->sprintf("%f",TempMax);
791    tmax->string=cadena->getString();
792    tmax->setName("Tmax");
793
794    cadena->sprintf("%f",TempMin);
795    tmin->string=cadena->getString();
796    tmin->setName("Tmin");
797
798    resultado->addChild(nombreNube);
799    resultado->addChild(tmax);
800    resultado->addChild(tmin);
801
802    resultado->addChild(nube);
803    resultado->setName("nube_grupo");
804
805    resultado->unrefNoDelete();
806    return resultado;
807  }
808
809  /*****************************************************************
       *****/
810  /* Crea un nodo separador para la representación de una malla de
       grids */
811  /* triangulados.
       */
812  /*****************************************************************
       *****/
813
814  SoSeparator *generar_mallaT(char *nombreFichero,unsigned char
       usar_paleta,int compR,int compG,int compB,float TempMax,float
       TempMin,char *pnombreMallat,bool forzar_datos)
815  {
816     unsigned long i=0,j=0;
```

```
817    float temppixel=0;
818    float dato=0;
819    int direccion=0;  //dirección de lectura N-O(0) o S-E(1)
820
821    SbColor colorPdefecto;
822    colorPdefecto.setValue(SbVec3f(
     (float)(compR/255.0),(float)(compG/255.0),(float)(compB/255.0)) );
823    infFichero *infoFichero;
824    datFichero *datosFichero;
825    SbString *pcadena=new SbString;
826
827    //para la identificación de la malla creada
828    SoInfo *nombreMallat=new SoInfo;
829    nombreMallat->setName("Nombre");
830
831    nombreMallat->string=pnombreMallat;
832
833    SoInfo *ArchivoOrigen=new SoInfo; //almacena la cadena del
     archivo que supone el origen de los
834                         //de cabecera,datos de la imagen.
835    ArchivoOrigen->setName("ArchivoOrigen");
836
837    if( (strstr(nombreFichero,".bin"))==NULL )
838    {
839      infoFichero=cargar_info("datoscab.txt");
840      if ( (infoFichero==NULL) || (infoFichero->flag_leido==0) )
     //en caso de error
841      {
842        printf("\n¡¡¡Error al abrir el archivo de cabecera!!!\n");
843        return(NULL); //exit
844      }
845      else //carga de datos
846      {
847      datosFichero=cargar_datos
     ("datosimag.txt",usar_paleta,colorPdefecto,TempMax,TempMin,forzar_d
     atos);
848      if(datosFichero->flag_leido==0)
849        {
850          printf("\n¡¡¡Error al intentar cargar los datos\n!!!");
851          return(NULL);//exit
852        }
853      }
854      ArchivoOrigen->string.setValue("datosimag.txt");
855    }
856    else if( (strstr(nombreFichero,".bin"))!=NULL ) //para archivos
     binarios
857    {
858      infoFichero=cargar_info(nombreFichero);
859      if ( (infoFichero==NULL) || (infoFichero->flag_leido==0) )
     //en caso de error
860      {
861        printf("\n¡¡¡Error al abrir el archivo de cabecera!!!\n");
862        return(NULL); //exit
863      }
864      else //carga de datos
865      {
866        datosFichero=cargar_datos
     (nombreFichero,usar_paleta,colorPdefecto,0,0,forzar_datos);
867        if(datosFichero->flag_leido==0)
868        {
869          printf("\n¡¡¡Error al intentar cargar los datos\n!!!");
```

```
870         return(NULL);//exit
871       }
872     }
873     ArchivoOrigen->string.setValue(nombreFichero);
874   }
875
876   //Creación de los nodos necesarios para la representación de
877   //la mallat
878   SoSeparator *resultado=new SoSeparator;
879   resultado->ref();
880
881   resultado->addChild(nombreMallat);
882   //se añade la información del archivo de origen
883   resultado->addChild(ArchivoOrigen);
884
885   //para almacenar el número de filas y columnas de la mallat
886   SoInfo *nFilas=new SoInfo;
887   SoInfo *nColumnas=new SoInfo;
888
889   pcadena->sprintf("%d",infoFichero->altop);
890   nFilas->string=pcadena->getString();
891   nFilas->setName("numFilas");
892
893   pcadena->sprintf("%d",infoFichero->anchop);
894   nColumnas->string=pcadena->getString();
895   nColumnas->setName("numColumnas");
896
897   resultado->addChild(nColumnas);
898   resultado->addChild(nFilas);
899
900   SoMaterial *material=new SoMaterial;
901   SoDrawStyle *estilo=new SoDrawStyle;
902   material->setName("material");
903   resultado->addChild(material);
904   resultado->addChild(estilo);
905   estilo->setName("estilo");
906   estilo->lineWidth.setValue(1);
907   SoTransform *transforma=new SoTransform;
908   transforma->setName("Transforma");
909   resultado->addChild(transforma);
910
911   SoNormalBinding *normbinding=new SoNormalBinding;
912   normbinding->value.setValue(SoNormalBinding::PER_VERTEX);
913   resultado->addChild(normbinding);
914
915   SoShapeHints *ShapeHints=new SoShapeHints;
916   ShapeHints->vertexOrdering=SoShapeHints::COUNTERCLOCKWISE;
    //visible ambas caras de la malla
917   resultado->addChild(ShapeHints);
918
919   //enlace para los colores de los vértices de la mallat
920   material->diffuseColor.setValues(0,infoFichero->altop*infoFichero
    ->anchop,datosFichero->indicepaleta);
921
922   SoMaterialBinding *myMaterialBinding = new SoMaterialBinding;
923   myMaterialBinding->value = SoMaterialBinding::PER_VERTEX_INDEXED;
    //MUUUUUUUUY IMPORTANTE PARA UNA CORRECTA VISUALIZACION
924   myMaterialBinding->setName("MaterialEnlace");
925   resultado->addChild(myMaterialBinding);
926
927   //creación de los grids que componen la mallat
```

```
928   int32_t *indices=new
    int32_t[(infoFichero->anchop-1)*(infoFichero->altop-1)*8];
929   SbVec3f *vnormals=new
    SbVec3f[(infoFichero->anchop)*(infoFichero->altop)*8];//normales
930
931   memset(vnormals,0,(infoFichero->anchop)*(infoFichero->altop) );
932   SoNormal *gridnormals=new SoNormal;
933   gridnormals->vector.setValues(0,(infoFichero->anchop)*(infoFicher
    o->altop),vnormals);
934
935   ----
936   | /|
937   |/ |
938   ----
939   for(i=0;i<(infoFichero->altop-1);i++)
940   {
941     for(j=0;j<(infoFichero->anchop-1);j++)
942     {
943
    indices[(i*(infoFichero->anchop-1)+j)*8]=(i*infoFichero->anchop+j);
944
    indices[(i*(infoFichero->anchop-1)+j)*8+1]=(i*infoFichero->anchop+j
    )+1;
945
    indices[(i*(infoFichero->anchop-1)+j)*8+2]=(i*infoFichero->anchop+j
    )+infoFichero->anchop+1;
946       indices[(i*(infoFichero->anchop-1)+j)*8+3]=SO_END_FACE_INDEX;
947
948
    indices[(i*(infoFichero->anchop-1)+j)*8+4]=(i*infoFichero->anchop+j
    )+infoFichero->anchop+1;
949
    indices[(i*(infoFichero->anchop-1)+j)*8+5]=(i*infoFichero->anchop+j
    )+infoFichero->anchop;
950
    indices[(i*(infoFichero->anchop-1)+j)*8+6]=(i*infoFichero->anchop+j
    );
951       indices[(i*(infoFichero->anchop-1)+j)*8+7]=SO_END_FACE_INDEX;
952     }
953   }
954
955   SoCoordinate3 *coordenadas=new SoCoordinate3; //almacena vértices
    de la malla
956   coordenadas->setName("Coordenadas");
957   coordenadas->point.setValues(0,infoFichero->anchop*infoFichero->a
    ltop,datosFichero->punto);
958   resultado->addChild(coordenadas);
959
960   estilo->lineWidth.setValue(1);  //grosor de línea preeterminado
    para la representación alámbrica de la malla
961   estilo->style.setValue(SoDrawStyle::FILLED);
962
963   SoIndexedFaceSet *mallaT=new SoIndexedFaceSet;
964   mallaT->coordIndex.setValues(0,((infoFichero->anchop-1)*(infoFich
    ero->altop-1)*8),indices);
965
966   //alamacenamiento de la información de Temperatura máxima y
    mínima
967   SbString *cadena=new SbString;
968   SoInfo *tmin=new SoInfo;
969   SoInfo *tmax=new SoInfo;
```

```
  970    cadena->sprintf("%f",TempMax);
  971    tmax->string=cadena->getString();
  972    tmax->setName("Tmax");
  973
  974    cadena->sprintf("%f",TempMin);
  975    tmin->string=cadena->getString();
  976    tmin->setName("Tmin");
  977
  978    resultado->addChild(nombreMallat);
  979    resultado->addChild(tmax);
  980    resultado->addChild(tmin);
  981    mallaT->setName("mallaT");
  982    resultado->addChild(mallaT);
  983    resultado->setName("mallaT_grupo");
  984    //devolvemos el objeto mallaT
  985    resultado->unrefNoDelete();
  986
  987    return resultado;
  988  }
  989
  990  /*****************************************************************
       *****/
  991  /*  Devuelve un objeto de la clase infObjeto conteniendo
       informacion   */
  992  /*  sobre la figura seleccionada.El objeto contiene un nodo con
       texto2D*/
  993  /*  para incluir en la escena gráfica (segun valores del argumento
       */
  994  /*  mostrar). mostrar=0 solo muestra la informacion del objeto en
       la   */
  995  /*  línea de comandos devolviendo NULL.1 devuelve la informacion en
       un */
  996  /*  objeto infObjeto,2 muestra la información en la linea de
       comandos y*/
  997  /*  devuelve la información en un objeto infObjeto.
       */
  998  /*****************************************************************
       *****/
  999
 1000  class infObjeto *infoObjeto(float posX,float posY,float posZ,int
       compR,int compG,int compB,unsigned char mostrar,SoNode
       *grupo_figura)
 1001  {
 1002
 1003    unsigned int numColumnas,numFilas;
 1004    unsigned char tamanoVertice;
 1005    float tamanoObjeto;
 1006    unsigned long numPuntos;
 1007    float maxValorX=0,maxValorY=0,maxValorZ=0;
 1008    float minValorX=0,minValorY=0,minValorZ=0,vmedio=0;
 1009    char *ArchivoOrigen;
 1010
 1011    infObjeto *pinfObjeto=new infObjeto;
 1012    SoSeparator *resultado=new SoSeparator;
 1013    SoTransform *transforma=new SoTransform;
 1014    SoOrthographicCamera * camara = new SoOrthographicCamera;
 1015    SoBaseColor *color=new SoBaseColor;
 1016    SoText2 *texto=new SoText2;
 1017
 1018    SoInfo *info; //para recuperación de información del objeto
 1019
```

```
1020    //obtención del tipo de malla
1021    char *nombre_grupo=(char *) ( (grupo_figura->getName()
      ).getString() );
1022
1023    if ( (strcmp( nombre_grupo,"malla_grupo")==0)||(strcmp(
      nombre_grupo,"mallaT_grupo")==0)
1024              ||(strcmp( nombre_grupo,"nube_grupo")==0) )
1025    {
1026      SoTransform *orgTransform=(SoTransform *)seleccionarHijo(
      grupo_figura,"Transforma");
1027      if (orgTransform!=NULL)
1028          pinfObjeto->posicion=orgTransform->translation.getValue();
1029
1030      if(strcmp( nombre_grupo,"malla_grupo")==0 )
1031      {
1032        SoQuadMesh *malla=(SoQuadMesh *)
      seleccionarHijo(grupo_figura,"malla") ;
1033        numColumnas=(malla->verticesPerRow).getValue();
1034        numFilas=malla->verticesPerColumn.getValue();
1035      }
1036      else //en caso de mallaT y nube de puntos
1037      {
1038        //se acceden a los nodos de información para obtgener el
      número de filas y columnas
1039        info=(SoInfo *)seleccionarHijo( grupo_figura,"numColumnas");
1040        numColumnas=atoi( info->string.getValue().getString() );
1041        info=(SoInfo *)seleccionarHijo( grupo_figura,"numFilas");
1042        numFilas=atoi( info->string.getValue().getString() );
1043      }
1044      //para todos los tipos de objeto
1045      info=(SoInfo *)seleccionarHijo( grupo_figura,"ArchivoOrigen");
1046      if(info!=NULL)
1047        ArchivoOrigen=(char *)info->string.getValue().getString();
      //obtención del archivo Origen
1048      else
1049        ArchivoOrigen=""; //nula
1050
1051      vmedio=promedio( (SoSeparator *) grupo_figura);
1052      tamanoVertice=(unsigned char)sizeof(SbVec3f);
1053      //valores máximos y mínimos
1054      SoCoordinate3 *coordenadas=(SoCoordinate3
      *)seleccionarHijo(grupo_figura,"Coordenadas");
1055      numPuntos=(unsigned long)coordenadas->point.getNum();
1056
1057      for(unsigned long i=0;i<(numPuntos);i++)
1058      {
1059       if (maxValorX < coordenadas->point[i][0])
      maxValorX=coordenadas->point[i][0];
1060       if (maxValorY < coordenadas->point[i][1])
      maxValorY=coordenadas->point[i][1];
1061       if (maxValorZ < coordenadas->point[i][2])
      maxValorZ=coordenadas->point[i][2];
1062      }
1063      minValorX=maxValorX;
1064      minValorY=maxValorY;
1065      minValorZ=maxValorZ;
1066      //obtención de los valores mínimos
1067      for(i=0;i<(numPuntos);i++)
1068      {
1069
1070       if (minValorX > coordenadas->point[i][0])
```

```
      minValorX=coordenadas->point[i][0];
 1071       if (minValorY > coordenadas->point[i][1])
      minValorY=coordenadas->point[i][1];
 1072       if (minValorZ > coordenadas->point[i][2])
      minValorZ=coordenadas->point[i][2];
 1073      }
 1074      //se guardan los datos
 1075      pinfObjeto->nombre_grupo=nombre_grupo;
 1076      pinfObjeto->numColumnas=numColumnas;
 1077      pinfObjeto->origen_datos=ArchivoOrigen;
 1078      pinfObjeto->numFilas=numFilas;
 1079      pinfObjeto->maxValorX=maxValorX;
 1080      pinfObjeto->maxValorY=maxValorY;
 1081      pinfObjeto->maxValorZ=maxValorZ;
 1082      pinfObjeto->minValorX=minValorX;
 1083      pinfObjeto->minValorY=minValorY;
 1084      pinfObjeto->minValorZ=minValorZ;
 1085      pinfObjeto->vmedio=vmedio;
 1086
 1087      if ( (mostrar==0) || (mostrar==2) )
 1088      {
 1089      //presentacion de los datos en la ventana de consola
 1090      printf("\nArchivo Origen: %s\n", ArchivoOrigen);
 1091      printf("Posicion del objeto en el espacio: %.f, %.f
      ,%.f\n",pinfObjeto->posicion[0],pinfObjeto->posicion[1],pinfObjeto-
      >posicion[2]);
 1092      printf("Nombre del grupo: %s\n",nombre_grupo);
 1093      printf("Numero Columnas; %ld\n",numColumnas);
 1094      printf("Numero de Filas: %ld\n",numFilas);
 1095      printf("Max valor X , Y  ,Z : %.3f, %.3f,
      %.3f\n",maxValorX,maxValorY,maxValorZ);
 1096      printf("Min valor X , Y  ,Z : %.3f, %.3f,
      %.3f\n",minValorX,minValorY,minValorZ);
 1097      printf("Valor promedio Z: %.3f\n",vmedio);
 1098      printf("Numero vertices: %ld\n",(numPuntos));
 1099      printf("Tamano vertice: %d bytes\n",tamanoVertice);
 1100      printf("Tamano objeto: %ld
      bytes",(numColumnas*numFilas)*tamanoVertice);
 1101      if( (numColumnas*numFilas)*tamanoVertice>1024 &&
      (numColumnas*numFilas)*tamanoVertice<1048576)
 1102        printf(" (%.2f
      Kb)\n",(float)((numColumnas*numFilas)*tamanoVertice)/1024 );
 1103      if( (numColumnas*numFilas)*tamanoVertice>1048576 )
 1104        printf(" (%.2f
      Mb)\n",(float)((numColumnas*numFilas)*tamanoVertice)/1048576 );
 1105      printf("MaxValor X Y Z:
      %.2f,%.2f,%.2f\n",maxValorX,maxValorY,maxValorZ) ;
 1106
 1107      }
 1108      //agrupación de las informaciones en un nodo
 1109       if ( (mostrar==1)|| (mostrar==2) )
 1110      {
 1111        SbString *cadena=new SbString[15];
 1112        cadena[0].sprintf("Archivo Origen: %s\n",ArchivoOrigen);
 1113        cadena[1].sprintf("Posicion espacial: %.f, %.f
      ,%.f\n",pinfObjeto->posicion[0],pinfObjeto->posicion[1],pinfObjeto-
      >posicion[2]);
 1114        cadena[2].sprintf("nombre del grupo: %s\n",nombre_grupo);
 1115        cadena[3].sprintf("numero de columnas: %d\n",numColumnas);
 1116        cadena[4].sprintf("numero de filas: %d\n",numFilas) ;
 1117        cadena[5].sprintf("Numero Vertices:
```

```
      %ld\n",numColumnas*numFilas) ;
1118        cadena[6].sprintf("tamano Vertice: %d bytes\n",tamanoVertice)
      ;
1119        cadena[7].sprintf("tamano objeto: %ld
      bytes\n",(numColumnas*numFilas)*tamanoVertice) ;
1120        //reasignación de unidades
1121        if( (numColumnas*numFilas)*tamanoVertice>1024 &&
      (numColumnas*numFilas)*tamanoVertice<1048576)
1122        {
1123        cadena[7].makeEmpty();
1124        cadena[7].sprintf("tamano objeto: %ld bytes (%.2f
      Kb)\n",(numColumnas*numFilas)*tamanoVertice,
1125          (float)((numColumnas*numFilas)*tamanoVertice)/1024);
1126        }
1127        if( (numColumnas*numFilas)*tamanoVertice>1048576 )
1128        {
1129        cadena[7].makeEmpty();
1130        cadena[7].sprintf("tamano objeto: %ld bytes(%.2f
      Mb)\n",(numColumnas*numFilas)*tamanoVertice,
1131          (float)((numColumnas*numFilas)*tamanoVertice)/1048576);
1132        }
1133        cadena[8].sprintf("MaxValor X Y Z:
      %.3f,%.3f,%.3f\n",maxValorX,maxValorY,maxValorZ) ;
1134        cadena[9].sprintf("MinValor X Y Z:
      %.3f,%.3f,%.3f\n",minValorX,minValorY,minValorZ) ;
1135        cadena[10].sprintf("Valor medio Z: %.3f\n",vmedio) ;
1136        //informacion sobre coordenadas y posicion del objeto
1137        if(orgTransform!=NULL)
1138        {
1139          SbVec3f valores; //para valores de retorno de las funciones
1140
1141          valores=orgTransform->center.getValue();
1142          cadena[11].sprintf("Centro del objeto:
      %.2f,%.2f,%.2f\n",valores[0],valores[1],valores[2]);
1143
1144          valores=orgTransform->translation.getValue();
1145          cadena[12].sprintf("Coordenadas absolutas del objeto:
      %.2f,%.2f,%.2f\n",valores[0],valores[1],valores[2]);
1146
1147          valores=orgTransform->scaleFactor.getValue();
1148          cadena[13].sprintf("Factor escala del objeto:
      %.2f,%.2f,%.2f\n",valores[0],valores[1],valores[2]);
1149
1150          float angulo;
1151          orgTransform->rotation.getValue().getValue(valores,angulo);
1152          cadena[14].sprintf("Rotacion: %.2f,%.2f,%.2f ->angulo=%.4f
      rad. (%.2f
      sexag.)\n",valores[0],valores[1],valores[2],angulo,(float)(angulo*(
      180/PI)));
1153        }
1154
1155        //nodos para el texto2D
1156        resultado->addChild(camara);
1157
      color->rgb.setValue((float)(compR/255.0),(float)(compG/255.0),(floa
      t)(compB/255.0));
1158        texto->string.setValues(0,10,cadena);
1159        transforma->setName("Transforma");
1160        resultado->addChild(transforma);
1161        resultado->addChild(color);
1162        resultado->addChild(texto);
```

```
1163
1164        //posicionamiento y reajuste del texto según visión de la
      cámara
1165        SbViewVolume vactual = camara->getViewVolume();
1166        SbVec3f pt = vactual.getPlanePoint(0.0f, SbVec2f(0.0f,
      0.95f));
1167
      transforma->translation.setValue(pinfObjeto->posicion[0],pinfObjeto
      ->posicion[1] , pinfObjeto->posicion[2]);
1168
1169        resultado->setName("Info");
1170        //se guardan el nodo resultado en el objeto infObjeto
1171        pinfObjeto->nodoInformacion=resultado;
1172        pinfObjeto->nodoInformacion->unrefNoDelete();
1173       }
1174    }
1175    return(pinfObjeto);
1176  }
1177
1178
1179
1180
1181  /*****************************************************************
      *****/
1182  /*      Crea una cadena de texto en 3D para incluir en la escena
      */
1183  /*      gráfica con el tamaño y en la posicion indicada por X
      */
1184  /*      Y  Z, y con el color indicado.
      */
1185  /*****************************************************************
      *****/
1186
1187  SoSeparator *etiqueta(char *cadena, float size,float X , float
      Y,float Z, struct colorRGB ccolor,char *identEtiqueta)
1188  {
1189    SbString pcadena;
1190    SoInfo *NombreEtiqueta=new SoInfo;
1191    SoSeparator *SepEtiqueta=new SoSeparator;
1192    SoBaseColor *color=new SoBaseColor;
1193    SoFont *fuente=new SoFont;
1194    SoText3 *texto3d=new SoText3;
1195    SoTransform *transforma=new SoTransform;
1196    transforma->setName("Transforma");
1197
1198    transforma->translation.setValue(X,Y,Z); //translacion a las
      coordenadas indicadas
1199
1200    SepEtiqueta->addChild(NombreEtiqueta);
1201    if(identEtiqueta!=NULL)
1202    NombreEtiqueta->string=identEtiqueta;
1203    else
1204    NombreEtiqueta->string=""; //cadena vacia
1205
1206    NombreEtiqueta->setName("Nombre");
1207    SepEtiqueta->addChild(transforma);
1208    SepEtiqueta->addChild(fuente);
1209    SepEtiqueta->addChild(color);
1210
1211    pcadena.sprintf("%s",cadena);  //se añade el texto
1212    texto3d->string=pcadena.getString();
```

```
1213
1214    //establecemos el color
1215    color->rgb.setValue(ccolor.componenteR,ccolor.componenteG,ccolor.
      componenteB);
1216    // y el tamaño de las letras
1217    fuente->size=size;
1218    SepEtiqueta->addChild(texto3d);
1219
1220    return(SepEtiqueta);
1221  }
```