Sistemas Informáticos
Curso 2002-03

# Generación de terrenos en 3D


**Realizado por:**

Mario Carballo Santos
Manuel Durán González
José Antonio Núñez Mendoza
Carmen Pardo Alhambra

**Dirigido por:**

Manuel Prieto Matías
Luis Piñuel Moreno
Roberto Lario-de-Llano

Dpto. Arquitectura de Computadores y Automática

Facultad de Informática
Universidad Complutense de Madrid

# Índice

# 1 Descripción del proyecto

Kilimanjaro Dem Viewer es un visor de archivos DEM que permite explorar a fondo las distintas características de la imagen. Permite sobrevolar el DEM, especificar el punto de luz, el nivel del mar, la escala de colores a utilizar así como salvar la pantalla actual en un archivo TGA. La aplicación es multiplataforma permitiendo su compilación en entornos tales como Windows, Linux…

Este proyecto surge ante la necesidad por parte del Departamento de Arquitectura de Computadores y Automática de una aplicación que permita visionar los archivos de elevación DEM en las distintas plataformas utilizadas (principalmente Windows y Linux).

English

Kilimanjaro Dem Viewer is an application for viewing DEM files that let's you explore completely the different characteristics of the image. Among its features, it lets you fly over the DEM, specify light type and position, sea level, color scale to be used and save screen state in TGA format. The application has been coded so that it is crossplatform and can be compiled either on Windows or Linux...

This project comes from the necessity from DACYA of an application that allows to view DEM heightfield files in different platforms (Windows and Linux mainly).

# 2 Metodología de trabajo

## 2.1 Fases de desarrollo

- Investigación
    - Dado que los conocimientos que se poseían acerca de OpenGL no eran demasiado extensos se realizó un profundo proceso de investigación acerca de las características de programación usando OpenGL con su correspondiente implementación de pequeños programas de prueba.
    - A su vez, se procedió a investigar acerca de formatos de archivo que resultaban desconocidos: TGA, USGS DEM etc.
    - Se realizarán además pequeñas pruebas de tecnología para aprender al desarrollo de interfaces utilizando las librerías facilitadas al comienzo del proyecto, GLUT y GLUI
- Análisis
    - Una vez conocidas las tecnologías a utilizar, sus ventajas y limitaciones, se realizó un análisis intenso de la aplicación a desarrollar. Este análisis se realizó en base a la especificación proporcionada. El resultado del análisis fue la estructuración en partes separadas de la aplicación y un orden de desarrollo e implementación de cada una de las partes según orden de importancia.
- Diseño
    - Tras obtener el análisis, se trató cada apartado por separado generando para cada uno un diagrama de clases. Sobre cada diagrama se meditó acerca de las mejoras que generaría el uso de patrones para implementar determinadas funcionalidades.
    - En paralelo con el diseño de clases, se produjo a esquematizar y definir el interfaz de interacción con el usuario que la aplicación debería tener.
    - Todas estas consideraciones quedaron reflejadas en un documento de diseño entregado a los directores del proyecto.

- Implementación

    - Llegado el momento de comenzar el proceso de implementación se desarrollaron pruebas en los dos entornos de desarrollo posibles, Borland C++ o Microsoft Visual C++ siendo este último el elegido por su gran compatibilidad con el desarrollo de aplicaciones usando OpenGL y GLUT.

    - El orden de implementación seguido, a *grosso modo* fue el siguiente:

        - Funcionalidad necesaria para cargar y visualizar ficheros DEM
        - Pequeño interfaz de prueba donde visualizar los ficheros
        - Resultado de los dos pasos anteriores se obtiene como resultado una pequeña aplicación que visualiza terrenos en dos dimensiones.
        - Estructura de clases necesarias para soportar el cargado de escena que contiene la mayor parte de la carga de la aplicación
        - Funcionalidad necesaria para las distintas vistas de escena
        - Interfaz completo con toda la funcionalidad
        - Funcionalidad necesaria para Cargar/Guardar la configuración del programa
        - Generación de sombras utilizando algoritmos más complejos que los implementados al iniciar la implementación

- Recursos

    - El principal recurso utilizado para el desarrollo de la aplicación ha sido una página web (Weblog). Mediante esta página, editable on-line por sus usuarios, se realizó la gestión de configuración así como se llevó siempre un diario de cambios de las distintas partes de la aplicación. Cada vez que se generaba una nueva versión del código, se subía dicho código a la página y se anotaban los cambios realizados.

    - Se definió por otro lado un estándar de documentación y nombrado de ficheros.

- Prueba y entrevista con el cliente.

    - Cada vez que se consiguió un hito importante si produjo una entrevista con el cliente (supervisor del proyecto) para comprobar el correcto funcionamiento de la aplicación y encauzar el desarrollo por el camino correcto, introduciendo los cambios necesarios a raíz de las correcciones del profesor.

- Generación de documentación

  - C++Doc: La herramienta utilizada para generar la documentación del proyecto es C++Doc. Dicha herramienta permite utilizar los comentarios del código debidamente formateados para generar un manual de referencia en formato HTML como salida.

    El formato de los comentarios utilizados por esta aplicación es muy similar al empleado por *javadoc.* Incluimos comentarios para cada una de las clases y métodos. Dicha documentación se incluye en la distribución del código del proyecto. En cada página referente a un método/clase existe un enlace al código en que aparece dicha definición.

# 3 Diseño de la aplicación

## 3.1 Tecnologías utilizadas

### 3.1.1 OpenGL, GLUT, GLUI

El proyecto ha sido desarrollado en el entorno Visual C++ 6.0. El apartado gráfico ha sido desarrollado en OpenGL utilizando las librerías GLUT y GLUI para diseñar el interfaz de interacción con el usuario.

GLUT es el OpenGL Utility Toolkit, una librería independiente del sistema para construir programas en OpenGL. Implementa un interfaz de programación simple (API) para OpenGL. GLUT hace considerablemente más fácil aprender y explorar la programación en OpenGL. El API de GLUT es portable, por lo que se puede escribir un único programa en OpenGL que funcione en ordenadores con Win32 y estaciones con X11.

GLUI es una librería de interfaz de usuario basada en GLUT-C++ que proporciona controles tales como botones, checkboxes, radiobuttons y spinners para aplicaciones realizadas en OpenGl. Es independiente del sistema de ventanas utilizado, dejando a GLUT toda la parte de gestión de eventos dependientes del sistema, tales como control de la ventana o del ratón.

### 3.1.2 XML

Por otro lado, puesto que la configuración de los distintos apartados del programa se puede salvar y posteriormente recuperar al iniciar, se ha utilizado la tecnología XML para almacenar las opciones elegidas por el usuario. XML (eXtensible Markup Language) es un lenguaje de marcado estándar que permite el intercambio de información en modo texto. Se ha escogido por la facilidad con que se pueden modificar los archivos que siguen el formato XML

(con un editor de texto es suficiente) y porque deseamos que sea extensible e independiente de la plataforma a utilizar. Asimismo, es una manera sencilla y cómoda de guardar/leer configuraciones almacenadas, evitándonos tener que hacer tratamiento de cadenas, etc.

Para validar un archivo XML se utiliza una DTD (Document Type Definition). Una DTD especifica la gramática que sigue nuestro archivo XML y por tanto, las reglas de construcción para tal archivo, esto es, qué elementos deben ir antes, con qué cardinalidad, etc.

El formato del archivo de configuración y su correspondiente DTD se mostrarán en detalle en la sección en que se explica el diseño de la aplicación.

### 3.1.3 Formatos de archivo: DEM, PGM, TGA.

#### ~ DEM

De la página que define la especificación para archivos USGS DEM:

- ***Background***

*The USGS Digital Elevation Model (DEM) data files are digital representations of cartographic information in a raster form. DEMs consist of a sampled array of elevations for a number of ground positions at regularly spaced intervals. These digital cartographic/geographic data files are produced by the U.S. Geological Survey ( USGS) as part of the National Mapping Program and are sold in 7.5-minute, 15-minute, 30-minute (also known as 2-arc-second), and 1-degree units. The 7.5- and 15-minute DEMs are included in the large scale category while 30-minute DEMs fall within the intermediate scale category and 1-degree DEMs fall within the small scale category.*

***Digital Elevation Model (DEM)***

*Large Scale*

> *The DEM data for 7.5-minute units correspond to the USGS 1:24,000- and 1:25,000-scale topographic quadrangle map series for all of the United States and its territories. Each 7.5-minute DEM is based on 30- by 30-meter data spacing with the Universal Transverse Mercator (UTM) projection. Each 7.5- by 7.5-minute block provides the same coverage as the standard USGS 7.5-minute map series.*
>
> *The 7.5-minute Alaska DEM data correspond to the USGS 1:24,000- and 1:25,000-scale topographic quadrangle map series of Alaska by unit size. The unit sizes in Alaska vary depending on the latitudinal location of the unit. The 7.5-minute Alaska DEM data consist of a regular array of elevations referenced horizontally to the geographic (latitude/longitude) coordinate system of the North American 1927 Datum (NAD 27) or the North American 1983 Datum*

*(NAD 83). The spacing between elevations along profiles is 1-arc-second in latitude by 2-arc-seconds of longitude.*

*The 15-minute DEM data correspond to the USGS 1:63,360-scale topographic quadrangle map series of Alaska by unit size. The unit sizes in Alaska vary depending on the latitudinal location of the unit. The 15-minute DEM data consist of a regular array of elevation referenced horizontally to the geographic (latitude/longitude) coordinate system of NAD 27. The spacing between elevations along profiles is 2- arc seconds of latitude by 3-arc-seconds of longitude.*

*Intermediate Scale*

*The 30-minute DEM data cover 30-minute by 30-minute areas which correspond to the east half or west half of the USGS 30- by 60-minute topographic quadrangle map series for the conterminous United States and Hawaii. Each 30-minute unit is produced and distributed as four 15- by 15-minute cells. The spacing of elevations along and between each profile is 2 arc seconds.*

*Small Scale*

*The 1-degree DEM (3- by 3-arc-second data spacing) provides coverage in 1- by 1-degree blocks for all of the contiguous United States, Hawaii, and most of Alaska. The basic elevation model is produced by or for the Defense Mapping Agency (DMA), but is distributed by the USGS, in DEM data record format. In reformatting the product, the USGS does not change the basic elevation information. The 1-degree DEMs are also referred to as 3-arc-second or 1:250,000-scale DEM data.*
*The EROS Data Center (EDC) also concatenated the 1- by 1-degree blocks for the contiguous United States in the Land Analysis System (LAS) environment using the elevation data from the photographic sources. This is referred to as the 1-degree DEM mosaic data set. Nine strips of concatenated imagery comprise the data set.*

- ***Data Characteristics***

*Large Scale*

*Each 7.5-minute unit of DEM coverage (based on the 7.5-minute quadrangle) consists of a regular array of elevations referenced horizontally in the UTM projection coordinate system. Elevation units are in meters or feet relative to National Geodetic Vertical Datum of 1929 (NGVD 29) in the continental U.S. and local mean sea level in Hawaii and Puerto Rico. The data are ordered from south to north in profiles that are ordered from west to east.*
*These horizontally referenced data may be NAD 27, NAD 83, Old Hawaiian Datum (OHD), or Puerto Rico Datum (PRD) of 1940.*

*The 7.5-minute Alaska DEMs consist of a regular array of elevations referenced horizontally to the geographic (latitude/longitude) coordinate system of NAD 27 or NAD 83. The data are ordered from south to north in profiles that are ordered from west to east. The unit of coverage corresponds to four basic quadrangle sizes for 1:63,360-scale graphics (depending on latitude):*

*Cell size limits*
*7.5- by 18-minutes -- State of Alaska north of 68°N latitude*
*7.5- by 15-minutes -- Between 62°N and 68°N latitude*
*7.5- by 11.25-minutes -- Between 59°N and 62°N latitude*
*7.5- by 10-minutes -- State of Alaska south of 59°N latitude*

*The 15-minute Alaska DEMs consist of a regular array of elevations referenced horizontally to the geographic (latitude/longitude) coordinate system of NAD 27 or NAD 83. The data are ordered from south to north in profiles that are ordered from west to east. Elevation units are in meters or feet relative to NGVD 29. The unit of coverage corresponds to four basic quadrangle sizes for 1:63,360-scale graphics (depending on latitude):*

*Cell size limits*
*15- by 36-minutes -- State of Alaska north of 68°N latitude*
*15- by 30-minutes -- Between 62°N and 68°N latitude*
*15- by 2.5-minutes -- Between 59°N and 62°N latitude*
*15- by 20-minutes -- State of Alaska south of 59°N latitude*

*Intermediate Scale*

*The 30-minute DEM data consist of a regular array of elevations referenced horizontally to the geographic (latitude/longitude) coordinate system of NAD 27 or NAD 83. The unit of coverage is a 30- by 30-minute block. Saleable units are four 15-minute DEMs covering a 30- by 30-minute area. Elevation data on the integer minute lines (all four sides) correspond to the same profiles on the surrounding eight blocks. Elevations are in meters or feet relative to NGVD 29.*

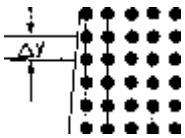
*Small Scale*

*The 1-degree DEM consists of a regular array of elevations referenced horizontally on the geographic (latitude/longitude) coordinate system of the WGS 72 (converted to WGS 84). The information content is approximately equivalent to that which can be derived from contour information represented on 1:250,000 scale maps.The unit of coverage is a 1- by 1-degree block. Elevation data on the integer degree lines (all four sides) overlap with the corresponding profiles on the surrounding eight blocks.*
*Elevations are in meters relative to NGVD 29 in the continental U.S. and local mean sea level in Hawaii. DEM accuracy information is provided in the Appendix.*
*The 1-degree DEM mosaic data set is characteristically the same as the source 1- by 1-degree DEM unit of coverage.*

- ***Spatial Resolution***

*Large Scale*

*The 7.5-minute DEM data are stored as profiles in which the spacing of the elevations along and between each profile is 30 meters. The number of elevations in a profile will vary because of the variable angle between the quadrangle's geographic boundary (neatline) and the UTM coordinate system. DEM data of low-relief terrain or generated from contour maps with intervals of 10 feet or less are recorded in feet while DEM data of moderate to high-relief terrain or generated from maps with terrain contour intervals greater than 10 feet are generally recorded in meters.*




***UTM Meter Grid for 7.5 DEM***

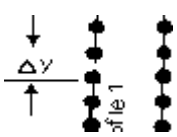
*The 15-minute DEM data are collected with a 2- by 3-arc-second spacing in latitude, and longitude, respectively. The first and last data points along a profile are at the integer degrees of latitude. Elevation data on the quadrangle neatlines (all four sides) share edge profiles with the surrounding eight quadrangles.*

*Intermediate Scale*

*Spacing of the elevations along each profile is 2-arc-seconds. The first and last data points are at the integer 15-minutes of latitude. A 15-minute profile will, therefore, contain 451 elevations.*

*Small Scale*

*Spacing of elevations along and between each profile of 1-degree DEMs is 3-arc-seconds with 1,201 elevations per profile. The exception is DEM data in Alaska, where the profile spacing varies depending on the latitudinal location of the DEM. Latitudes between 50 and 70 degrees north have spacings at 6-arc-seconds with 601 profiles per 1-degree unit and latitudes greater than 70 degrees north have spacings at 9-arc-seconds with 401 profiles per 1-degree unit.*
*The 1-degree mosaic data set spacing of elevation and profile data is the same as the data of conterminous U.S. 1-degree DEM data set.*




***Arc Second Grid for 1-degree DEM***

- ***Data Organization***

*A DEM file is organized into three logical records:*

- *Type A*

  *Contains information defining the general characteristics of the DEM, including DEM name, boundaries, units of measurement, minimum and maximum elevations, projection parameters, and number of type B records. Each DEM file has one Type A record.*

- *Type B*

  *Contains profiles of elevation data and associated header information. Each profile has a Type B record.*

- *Type C*

  *Contains statistics on the accuracy of the data.*

*Examples of a DEM record format may be found in the Appendix.*

***Entity and Attribute Information***

*A digital elevation model is composed of integer values representing a gridded form of a topographic map hypsography overlay.*

*Additional information on DEM specifications can be found in the USGS National Mapping Program's Standards for Digital Elevation Model (DEMs).*

***Classification Levels***

*DEM data are organized in three classification levels. Level-1 DEMs are elevation data sets in a standardized format. The intent is to reserve this level for 7.5-minute DEMs which are created by scanning National High Altitude Photography (NHAP)/NAPP photography. A vertical RMSE of 7 meters is the desired accuracy standard. A RMSE of 15 meters is the maximum permitted.*

*Level-2 DEMs are elevation data sets that have been processed or smoothed for consistency and edited to remove identifiable systematic errors. DEM data derived from hypsographic and hydrographic data digitizing, either photogrammetrically or from existing maps, are entered into the Level-2 category. A RMSE of one-half contour interval is the maximum permitted.*

*Level-3 DEMs are derived from DLG data by incorporating selected elements from both hypsography (contours, spot elevations) and hydrography (lakes, shorelines, drainage). A RMSE of one-third of the contour interval is the maximum permitted.*

***Digital Elevation Model Caveats***

*Large Scale*

*The majority of the 7.5-minute DEMs produced to date are categorized as Level-1 DEMs.*

*Intermediate Scale*

*All 30-minute DEMs derived from contours are Level 2. All 30-minute DEMs derived from 7.5-minute DEMs are Level 1.*

*Small Scale*

*All 1-degree DMA DTED-1 data have been classified as Level 3.*

## ~ PGM

Cada imagen PGM está formada por:

- Un "magic number" para identificar el tipo de fichero. El "magic number" de un fichero PGM son los dos caracteres "P5".
- Espacio blanco (blancos tabuladores, retornos de carro o saltos de línea).
- Una anchura, formateada como caracteres ASCII en decimal.
- Espacio blanco
- Una altura, de nuevo en ASCII decimal.
- Espacio blanco.
- El máximo nivel de gris (Maxval), de nuevo en ASCII decimal. Debe ser menor de 65536.
- Un carácter de "Newline" u otro carácter blanco.
- Una serie de anchura * Altura niveles de gris, recorriendo la imagen de izquierda a derecha. Cada nivel de gris es un número entre 0 y Maxval, con 0 correspondiendo al negro y Maxval, al blanco. Cada nivel de gris se representa en binario mediante 1 ó 2 bytes. Si Maxval es menor de 256, se emplea 1 byte. En otro caso, se usan 2 bytes. El byte más significativo es el primero.
- Cada nivel de gris es un número proporcional a la intensidad del pixel
- Los caracteres situados entre un "#" hasta el final de la línea, antes de la línea que contiene Maxval, son comentarios y se ignoran.
- En realidad hay otra versión del formato PGM que es bastante poco frecuente: el "plain" ("sencillo") PGM. El formato anterior, generalmente considerado el normal, se conoce como "raw" PGM. La diferencia en el formato "plain" es:
  - El "magic number" es "P2", en lugar de "P5".
  - Cada píxel se representa mediante un número decimal ASCII (de tamaño arbitrario).
  - Cada píxel tiene espacio blanco delante y detrás de él. Debe haber al menos un carácter blanco entre dos píxeles, pero no hay un máximo.
  - Ninguna línea debería tener más de 70 caracteres.

Éste es un ejemplo de una pequeña imagen en este formato:

```
P2
# feep.pgm
24 7
15
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 0
0  3  3  3  3  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15 15 15 15 0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0 15 0
0  3  3  3  0  0  0  7  7  7  0  0  0 11 11 11  0  0  0 15 15 15 15 0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 15  0  0  0 0
0  3  0  0  0  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15  0  0  0 0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0 0
```

**Nota de compatibilidad**: Antes de abril de 2000, un fichero PGM "raw" no podía tener un Maxval mayor de 255. Por lo tanto, no podía tener más de un byte por píxel. Los programas antiguos pueden tener esta restricción.

## ~ TGA

De la especificación de True Targa:

*A TGA file has a header that consists of 12 fields. These are:*

- *id (unsigned char)*
- *colour map type (unsigned char)*
- *image type (unsigned char)*
- *colour map first entry (short int)*
- *colour map length (short int)*
- *map entry size (short int)*
- *horizontal origin (short int)*
- *vertical origin (short int)*
- *width (short int)*
- *height (short int)*
- *pixel depth (unsigned char)*
- *image descriptor (unsigned char)*

*From all these fields we only really care about the image type, in order to find out if the image is uncompressed and it is not color indexed, the width and height of the image, the pixel depth, and finally the image descriptor that contains the image pixels.*

*Some possible values for the image type are:*

- *1 - colour map image*
- *2 - RGB(A) uncompressed*
- *3 - greyscale uncompressed*
- *9 - greyscale RLE (compressed)*
- *10 - RGB(A) RLE (compressed)*

*The only types that we're dealing with here are 2 and 3. As for the pixel depth, it represents the number of bits per pixel used, i.e. a greyscale image has 8 for pixel depth, where as a RGBA has 32.*

*One note of interest is that a TGA stores the pixels in BGR mode, i.e. the red and blue components are swapped, relative to RGB. This implies that we'll have to swap them when we load or save the image.*

## *3.2 Patrones de diseño*

Durante la confección de la estructura en clases de la aplicación se han utilizado diversas técnicas de diseño. Quizás la más reseñable sea la aplicación de patrones de diseño de software. Estos patrones permiten que la aplicación resultante sea más fácil de extender, reutilizar etc.

Un patrón de diseño describe una comunicación entre objetos y clases que es definida para resolver un problema general de diseño en un contexto particular. Un patrón de diseño identifica los aspectos clave de una estructura común confeccionada para que resulte más fácil crear un diseño orientado a objetos más reutilizable. El diseño con patrones identifica la participación de cada clase e instancia, sus roles y colaboraciones y la distribución de responsabilidades. Cada patrón de diseño se centra en un problema particular de diseño orientado a objetos. [L01]

## *3.3 Explicación del diseño*

El diseño de la aplicación se llevó a cabo teniendo como principal objetivo la construcción de código reutilizable y fácilmente extensible. Se construyó bajo en paradigma del diseño orientado a objetos y teniendo en cuenta la herramienta de desarrollo (Visual C++).

La aplicación constará de los siguientes componentes:

- Clases destinadas al manejo de datos. Entre ellas encontramos las clases que surgen al leer archivos dem. Todas las clases derivarán de una clase común, cuyo interfaz será el único que conozcan el resto de clases, de modo que la implementación específica quede ajena al resto de aplicación.

- Clases destinadas a la visualización y manejo de la aplicación (GUI). En ella incluimos tanto la clase principal como aquellas que se deriven de las respectivas cajas de diálogo destinadas a la configuración del programa.

- Clases destinadas al manejo de eventos. Según se ha concebido la aplicación, se podrá manejar la posición de la cámara, la intensidad y localización del punto de luz, nivel de agua, etc… Para una sencilla configuración de los mismos podremos conmutar entre distintos controladores que responderán al movimiento del ratón variando los parámetros necesarios en cada caso.

- Clases destinadas a visualización de los terrenos y demás elementos de la escena. Para ello tendremos clases que implementen operaciones típicas de OpenGL, tales como transformaciones de matrices, etc… además de los objetos propiamente dichos. Estos objetos derivan de una clase común que es la que proporciona los métodos que debe utilizar la aplicación para dibujar en pantalla. Cada elemento implementará su parte de código OpenGL según haya de dibujarse.

- Clases destinadas a aplicar sombras y otros efectos. Han de ser intercambiables de modo que el código quede lo menos acoplado posible. Cada una de ellas derivará de un interfaz común y proporcionará el método necesario para su aplicación.

### 3.3.1 Tipos de archivo

La aplicación es capaz de leer archivos USGS DEM y PGM. Se proporciona además una utilidad (y su código fuente) para convertir archivos en formato sdts dem a formato usgs dem. Se pueden incluir nuevos tipos de archivo de forma sencilla, tal y como quedará explicado en el apartado dedicado a puntos de extensión.

Para que el manejo de los diferentes tipos y su dibujado sean uniformes definimos una clase común de la cual derivan los tipos de fichero que hemos de soportar. Debemos implementar los métodos de abrir archivo, leerlo y obtener un punto determinado de la matriz de alturas según sea el caso.

Asimismo proporcionamos métodos para conocer información necesaria para la representación del archivo y otros datos adicionales. Tal información se encuentra en la cabecera de los archivos dem, y puede ser la máxima altura, mínima, ancho y largo del mapa, unidad de medida, etc.

Para obtener un objeto del tipo determinado según sea la opción que se haya escogido al abrir archivo utilizamos un patrón de tipo Factoría. En ella registramos los tipos de archivo que hemos de leer y devolvemos un objeto del tipo correspondiente según un parámetro.

Los archivos de salida que podemos utilizar de momento son los tga. Se salvará siempre la vista actual del archivo, según estén aplicadas las sombras y posicionada la cámara. Se dejará preparada la aplicación para incluir otros formatos, tales como jpg, tiff, etc.

### 3.3.2 Diseño del GUI

Dado que la librería proporcionada no ofrece cajas de diálogo predefinidas, definiremos unas clases base vacías con comportamiento vacío, tales como las de aceptar, aplicar y cancelar, etc… Dichas ventanas encapsulan todo el código OpenGL en clases C++.

Debe tener referencias a la escena que se está mostrando, así como a aquellos componentes de la escena individuales que sean necesarios para configurar otros elementos, ya sea la cámara o el punto de luz….

Se ha estructurado de la siguiente forma:

- Una ventana principal desde la cual controlar las opciones más comunes de la aplicación, esto es:
  - Shader a aplicar
  - Controlador utilizado
  - Resolución
  - Elementos de escena
  - Opciones de archivo
- Ventanas de configuración.
  - Una ventana por cada shader configurable, adaptada a los parámetros que debe configurar cada uno
  - Ventana de configuración de las propiedades de escena. En esta ventana se pueden configurar elementos más específicos, tales como los colores para los elementos de la escena, incidencia de la luz, etc.
- Ventanas de información:
  - Ventana para mostrar la *"Metainformación"* del archivo.
  - Ventana para mostrar mensajes.
  - Acerca de.

### 3.3.3 Control de eventos

Utilizaremos las funciones que proporcionan las GLUT para responder a los eventos. Todas las funciones llamarán a métodos homónimos implementados en los controladores. Son los controladores los encargados de reflejar los eventos de ratón y teclas en la configuración de la aplicación.

El funcionamiento de todos estos controladores será el mismo. Se tomará una muestra de las coordenadas del ratón en el momento en que ocurra un evento, y se tomará otra muestra en el momento en que ocurra otro evento que determine que debe haber un cambio. Se calculará la diferencia de cada uno de los parámetros de las muestras (ejes, teclas presionadas) y se aplicará la diferencia obtenida a cada una de las variables que se deban modificar por el controlador.

Puesto que queremos controlar el punto de luz, la posición de la cámara, nivel del agua y algún otro elemento que surja mediante el teclado o el ratón, necesitaremos los siguientes controladores:

- **Control de cámara:**

  Si se mantiene pulsado el botón izquierdo del ratón y se mueve el mismo, la vista de cámara cambia, variando el ángulo azimut y rotando el escenario según se mueva de izquierda a derecha y viceversa. Si se presiona el botón derecho y se sube o baja el ratón aumentará o disminuirá el tamaño del terreno (se hace zoom).

- **Control de iluminación:**

  Varía la posición del punto de luz. Según movamos el ratón con el botón izquierdo presionado el punto de luz se alejará, rotará alrededor del terreno, etc.

- **Control de nivel de agua:**

El nivel de agua aumenta o disminuye según se mueva el ratón con el botón principal presionado.

- **Control de cámara libre:**

Permite desplazarnos por el terreno en todos los sentidos posibles: rotaciones, traslación y traslación vertical.

- **Control de exageración vertical:**

Incrementa el factor por el que se multiplica la altura de cada uno de los puntos del terreno, lo cual permite que se puedan exagerar los cambios en la altura de los puntos según se mueva el ratón con el botón izquierdo presionado.

- **Control Crop:**

Si se selecciona una parte del terreno con este control seleccionado, se recorta dicha selección y se centra en la pantalla, convirtiéndose en la única parte visible del terreno. Cualquier cambio aplicado a la escena (ya sea presencia de agua, cambio de resolución, etc, sólo se hará presente en la parte visible mostrada.

### 3.3.4 Configuración de la aplicación

La clase d3d_AppConfigHandler es la encargada de manejar las operaciones de guardar/salvar la configuración de la aplicación. La tecnología utilizada en este caso es XML. La clase SceneView declara como "amiga" a esta clase para permitirle acceder a todos sus atributos.

La librería utilizada para la realización del fichero XML es Xerces, en concreto Xerces-C++ Versión 2.3.0.

Xerces-C++ es un analizador que valida XML escrito en un subconjunto portable de C++. Facilita la tarea de añadir a una aplicación la habilidad de leer y escribir datos XML. Contiene también una librería compartida para analizar, generar, manipular y validar documentos XML.

Xerces-C++ es fiel a la recomendación XML 1.0 y otros estándares asociados.

El analizador aporta alta interpretación, modularidad y escalabilidad. El código fuente, muestras y documentación API se facilita junto con el analizador. para favorecer la portabilidad, se ha tenido cuidado de usar mínimamente Templates, RTTI e #ifdefs.

#### ~ Estructura del fichero DTD

El fichero DTD utilizado para validar el fichero XML es el que sigue.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--DTD generated by XMLSPY v5 rel. 4 U (http://www.xmlspy.com)-
->
<!ELEMENT anaglyph (#PCDATA)>
<!ELEMENT application (scenePrefs, sceneElements, shaders)>
<!ELEMENT background (colorRGBA)>
<!ELEMENT colorRGBA (value,value,value,value)>
<!ELEMENT daylight (colorRGBA)>
<!ELEMENT gradient (gradientColor+)>
<!ELEMENT gradientColor (value, colorRGBA)>
<!ELEMENT grid (colorRGBA)>
<!ELEMENT gridVisible (#PCDATA)>
<!ELEMENT ambientLight (colorRGBA)>
<!ELEMENT diffuseLight (colorRGBA)>
<!ELEMENT specularLight (colorRGBA)>
<!ELEMENT horizon (colorRGBA)>
<!ELEMENT hsv (value,value,value,value)>
<!ELEMENT hud (colorRGBA)>
<!ELEMENT hudVisible (#PCDATA)>
<!ELEMENT lake (colorRGBA+)>
<!ELEMENT lightIconVisible (#PCDATA)>
<!ELEMENT nadir (colorRGBA)>
<!ELEMENT realisticLighting (#PCDATA)>
```

```
<!ELEMENT renderingStyle (#PCDATA)>
<!ELEMENT sceneElements (skyVisible, hudVisible, gridVisible,
waterVisible)>
<!ELEMENT scenePrefs (background, zenith, horizon, nadir, hud,
grid, water, ambientLight,diffuseLight,specularLight, shininess,
realisticLighting, lightIconVisible, renderingStyle)>
<!ELEMENT shaders (anaglyph, daylight, gradient, hsv, lake,
slope, terrace)>
<!ELEMENT shininess (#PCDATA)>
<!ELEMENT skyVisible (#PCDATA)>
<!ELEMENT slope (colorRGBA+)>
<!ELEMENT terrace (value, colorRGBA+)>
<!ELEMENT value (#PCDATA)>
<!ELEMENT water (colorRGBA)>
<!ELEMENT waterVisible (#PCDATA)>
<!ELEMENT zenith (colorRGBA)>
```

## ~ Estructura del fichero XML

El fichero que contiene la información es “app-config.xml” y se encuentra en el directorio raíz de la aplicación. Dicho archivo XML se valida con la DTD (Document Type Definition) especificada en el archivo “app-config.dtd” sito en el mismo lugar.

Los datos que conformar la configuración a guardar son:

- Preferencias de escena (ScenePreferences):
  - Background Color
  - Sky Zenit Color
  - Sky Horizon Color
  - Sky Nadir Color
  - Hud color
  - Grid color
  - Water Color
  - Ambient Light (aportación de luz ambiente al color de un punto).
  - Diffuse Light (aportación de luz difusiva al color de un punto).
  - Specularl Light (aportación de luz especualar al color de un punto).
  - Icono de luz visible
  - Estilo de renderizado (Flat, Smooth, Goraud, Wired)
  - Brillo
  - Luz realista seleccionada o no
- Elementos de escena visibles
  - Hud
  - Grid
  - Water
  - Sky

- Parámetros de cada shader configurable:

    - Anaglyph
    - DayLight
    - Gradient
    - HSV
    - Terrace
    - Slope
    - Lake

Un color está formado por 4 componentes :

- R : cantidad de rojo (de 0 a 1)
- G: cantidad de verde (de 0 a 1)
- B: cantidad de azul (de 0 a 1)
- Alpha: grado de transparencia (de 0 a 1)

Para almacenar un color en el archivo XML se pone la etiqueta <colorRGBA> y a continuación los 4 valores anteriores que conforman un color.

```
<colorRGBA>
        <value>0.000000</value> //Rojo
        <value>0.000000</value> //Verde
        <value>0.000000</value> //Azul
        <value>1.000000</value> //Alpha
 </colorRGBA>
```

Al almacenar información sobre algún componente formado únicamente por un valor, se toma por convenio poner el valor ente las marcas de inicio y fin del concepto sin necesidad de escribir la etiqueta <value>

```
<shininess>80.000000</shininess>
```

Si en el concepto aparece un valor y además uno o más colores, el valor viene entre etiquetas <value> y los colores con el esquema explicado anteriormente.

```
<concepto>
     <value>5.670280</value>
      <colorRGBA>
             <value>0.000000</value>
             <value>0.000000</value>
             <value>1.000000</value>
             <value>1.000000</value>
      </colorRGBA>
</concepto>
```

El esquema del fichero XML que genera KDV es el siguiente:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<!DOCTYPE application SYSTEM "app-config.dtd">
<application>

  <scenePrefs>
    <background>
     "Color del fondo"
    </background>
    <zenith>
     "Color del Zenit"
    </zenith>
    <horizon>
     "Color del horizonte"
    </horizon>
    <nadir>
     "Color del Nadir"
    </nadir>
    <hud>
     "Color del HUD"
    </hud>
    <grid>
     "Color del Grid"
    </grid>
    <water>
     "Color del agua"
    </water>
    <ambientLight>
      <colorRGBA>
      "Aportación de luz ambiente.
      </colorRGBA>
    </ambientLight>
    <diffuseLight>
      <colorRGBA>
      "Aportación de luz difusiva.
      </colorRGBA>
    </diffuseLight>
    <specularLight>
      <colorRGBA>
      "Aportación de luz especular.
      </colorRGBA>
    </specularLight>
    <shininess>80.000000</shininess>
    <realisticLighting>"Yes/No"</realisticLighting>
    <lightIconVisible>"Yes/No"</lightIconVisible>
    <renderingStyle>"Flat/Gouraud/Smooth/Wire"</renderingStyle>
  </scenePrefs>
  <sceneElements>
    <skyVisible>"Yes/No"</skyVisible>
    <hudVisible>"Yes/No"</hudVisible>
    <gridVisible>"Yes/No"</gridVisible>
    <waterVisible>"Yes/No"</waterVisible>
  </sceneElements>
```

```
<shaders>
    <anaglyph>Distancia para translación lateral entre capas.
   </anaglyph>
  <daylight>
    <colorRGBA>
   "Color base del shader DayLight"
    </colorRGBA>
  </daylight>
  <gradient>
    <gradientColor>
   "Altura relativa"
      <colorRGBA>
   "Color asociado a la altura anterior"
      </colorRGBA>
    </gradientColor>
    <gradientColor>
   "Altura relativa"
      <colorRGBA>
   "Color asociado a la altura en color"
      </colorRGBA>
    </gradientColor>
  </gradient>
  <hsv>
   "Valor de Low Start"
   "Saturación"
   "Value"
   "Range"
  </hsv>
  <lake>
    <colorRGBA>
   "Color de las zonas  de agua"
    </colorRGBA>
    <colorRGBA>
   "Color de las zonas de tierra"
    </colorRGBA>
  </lake>
  <slope>
    <colorRGBA>
   "Color de las zonas planas"
    </colorRGBA>
    <colorRGBA>
   "Color de las zonas verticales"
    </colorRGBA>
  </slope>
  <terrace>
     "Tamaño del paso (Step Size)"
    <colorRGBA>
   "Color 1"
    </colorRGBA>
    <colorRGBA>
   "Color 2"
    </colorRGBA>
    <colorRGBA>
   "Color Edge"
    </colorRGBA>
  </terrace>
```

```
  </shaders>

</application>
```

Mostramos ahora cómo quedaría un archivo XML de ejemplo siguiendo la especificación anterior:

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="no" ?>
<!DOCTYPE application SYSTEM "app-config.dtd">
<application>

  <scenePrefs>
    <background>
      <colorRGBA>
        <value>0.000000</value>
        <value>0.000000</value>
        <value>0.000000</value>
        <value>1.000000</value>
      </colorRGBA>
    </background>
    <zenith>
      <colorRGBA>
        <value>0.317647</value>
        <value>0.266667</value>
        <value>0.466667</value>
        <value>1.000000</value>
      </colorRGBA>
    </zenith>
    <horizon>
      <colorRGBA>
        <value>0.662455</value>
        <value>0.298704</value>
        <value>0.884582</value>
        <value>1.000000</value>
      </colorRGBA>
    </horizon>
    <nadir>
      <colorRGBA>
        <value>0.000000</value>
        <value>0.501961</value>
        <value>0.250980</value>
        <value>1.000000</value>
      </colorRGBA>
    </nadir>
    <hud>
      <colorRGBA>
        <value>0.000000</value>
        <value>1.000000</value>
        <value>0.000000</value>
        <value>1.000000</value>
      </colorRGBA>
    </hud>
    <grid>
      <colorRGBA>
```

```
        <value>1.000000</value>
        <value>1.000000</value>
        <value>1.000000</value>
        <value>1.000000</value>
      </colorRGBA>
    </grid>
    <water>
      <colorRGBA>
        <value>0.000000</value>
        <value>0.000000</value>
        <value>1.000000</value>
        <value>1.000000</value>
      </colorRGBA>
    </water>
    <ambientLight>
      <colorRGBA>
        <value>0.000000</value>
        <value>0.000000</value>
        <value>0.000000</value>
        <value>1.000000</value>
      </colorRGBA>
    </ambientLight>
    <diffuseLight>
      <colorRGBA>
        <value>1.000000</value>
        <value>1.000000</value>
        <value>1.000000</value>
        <value>1.000000</value>
      </colorRGBA>
    </diffuseLight>
    <specularLight>
      <colorRGBA>
        <value>0.400000</value>
        <value>0.400000</value>
        <value>0.400000</value>
        <value>1.000000</value>
      </colorRGBA>
    </specularLight>
    <shininess>80.000000</shininess>
    <realisticLighting>yes</realisticLighting>
    <lightIconVisible>yes</lightIconVisible>
    <renderingStyle>flat</renderingStyle>
  </scenePrefs>

  <sceneElements>
    <skyVisible>yes</skyVisible>
    <hudVisible>no</hudVisible>
    <gridVisible>yes</gridVisible>
    <waterVisible>no</waterVisible>
  </sceneElements>

  <shaders>
    <anaglyph>200.000000</anaglyph>
    <daylight>
      <colorRGBA>
        <value>0.000000</value>
```

```
      <value>0.091213</value>
      <value>0.762849</value>
      <value>1.000000</value>
    </colorRGBA>
  </daylight>
  <gradient>
    <gradientColor>
      <value>0.000000</value>
      <colorRGBA>
        <value>0.672655</value>
        <value>0.604728</value>
        <value>0.595774</value>
        <value>0.000000</value>
      </colorRGBA>
    </gradientColor>
    <gradientColor>
      <value>1.000000</value>
      <colorRGBA>
        <value>0.000000</value>
        <value>1.000000</value>
        <value>1.000000</value>
        <value>1.000000</value>
      </colorRGBA>
    </gradientColor>
  </gradient>
  <hsv>
    <value>0.000000</value>
    <value>1.000000</value>
    <value>1.000000</value>
    <value>360.000000</value>
  </hsv>
  <lake>
    <colorRGBA>
      <value>0.000000</value>
      <value>1.000000</value>
      <value>1.000000</value>
      <value>1.000000</value>
    </colorRGBA>
    <colorRGBA>
      <value>0.000000</value>
      <value>0.628732</value>
      <value>0.575779</value>
      <value>1.000000</value>
    </colorRGBA>
  </lake>
  <slope>
    <colorRGBA>
      <value>1.000000</value>
      <value>0.000000</value>
      <value>0.000000</value>
      <value>1.000000</value>
    </colorRGBA>
    <colorRGBA>
      <value>1.000000</value>
      <value>1.000000</value>
      <value>0.000000</value>
```

```
        <value>1.000000</value>
      </colorRGBA>
    </slope>
    <terrace>
      <value>100.000000</value>
      <colorRGBA>
        <value>0.000000</value>
        <value>0.000000</value>
        <value>1.000000</value>
        <value>1.000000</value>
      </colorRGBA>
      <colorRGBA>
        <value>1.000000</value>
        <value>1.000000</value>
        <value>0.000000</value>
        <value>1.000000</value>
      </colorRGBA>
      <colorRGBA>
        <value>1.000000</value>
        <value>0.000000</value>
        <value>0.000000</value>
        <value>1.000000</value>
      </colorRGBA>
    </terrace>
  </shaders>

</application>
```

#### *3.3.4..1 Guardar*

En el proceso de almacenado de la información de configuración, se genera un árbol DOM (Document Object Model) siguiendo la estructura anteriormente explicada. Para construir el árbol se accede a todos los valores del SceneView. Una vez construido el árbol se escribe en el fichero. La configuración se almacena en el momento que desee el usuario, utilizando la opción correspondiente desde el interfaz.

#### *3.3.4..2 Recuperar*

La configuración se recupera leyendo del archivo “app-config.xml” en el momento en que se inicia la aplicación, más concretamente cuando se crea el objeto SceneView. Si se desea intercambiar distintos archivos de configuración basta con renombrarlos a “app-config.xml”. Una vez analizado el documento y generado el árbol, se recorren cada uno de los nodos y se van actualizando los valores correspondientes en el SceneView, llamando a los métodos correspondientes para notificar al resto de elementos de la aplicación la nueva configuración.

# 3.4 Visualización de terrenos

## 3.4.1 Visión General

Se tiene un objeto que representa la escena que se ha de mostrar en cada momento. Esta escena está compuesta, en un principio, por el terreno, el punto de luz, el agua, coordenadas y otros elementos que puedan surgir. Puesto que no queremos cargar en exceso el pintado de los objetos recurrimos a la técnica del bounding box para delimitar que partes de la escena deben aparecer.

Para poder mostrar por pantalla de forma correcta todos estos elementos, el orden en que se hace el renderizado es importante. Puesto que no queremos llevar la cuenta de que elementos constan en nuestra escena, y tampoco queremos implicarnos en decisiones de control según los casos, utilizamos un patrón Composite.

El patrón composite nos permite tratar la escena como si fuese un árbol. En él, no se hace distinción entre nodos intermedios, raíz u hojas. Cuando se quiere renderizar la escena, se da la orden render a la raiz y será la encargada de propagarlo al resto de componentes de la escena. Además, puesto que haremos un recorrido de los hijos de izquierda a derecha, nos permite especificar qué elementos han de renderizarse antes que otros, según sea el orden en que se insertan en el árbol.

Todos los nodos descienden de una clase común llamada SceneNode. De ella hereda la clase ShapeNode (nodos que deben ser representados en la escena) y que contiene los métodos necesarios para que un objeto se pueda dibujar. Para cada nodo de este tipo, se indica el estilo de sombreado, el brillo, color especular, forma de dibujar polígonos… Los elementos que podemos dibujar gracias a esta clase son el grid, esferas, indicadores de coordenadas, cajas, texto.. Esta clase es la que utiliza OpenGL para pintar cada uno de los elementos. De esta clase también obtenemos nuestra clase principal, SG_Dem. Se trata de otro ShapeNode que utiliza el shader establecido en ese momento para aplicar los colores necesarios.

Si queremos pintar con las distintas opciones de OpenGL (wire, smooth, flat), utilizaremos las opciones del terreno. Estas opciones son compartidas por cada uno de los objetos de la escena. En dichas opciones se especifican también los colores que se deben utilizar para los elementos de la escena, su brillo, tipo de iluminación, etc.

Dividimos las clases según su comportamiento. Por un lado tendremos las clases que forman parte del grafo de la escena, estas son, aquellas que derivan de SceneNode.

Por otro lado las clases utilizadas como utilerías y que representan matrices de coordenadas, transformaciones, etc.

Para nuestra escena son importantes por tanto los siguientes elementos:

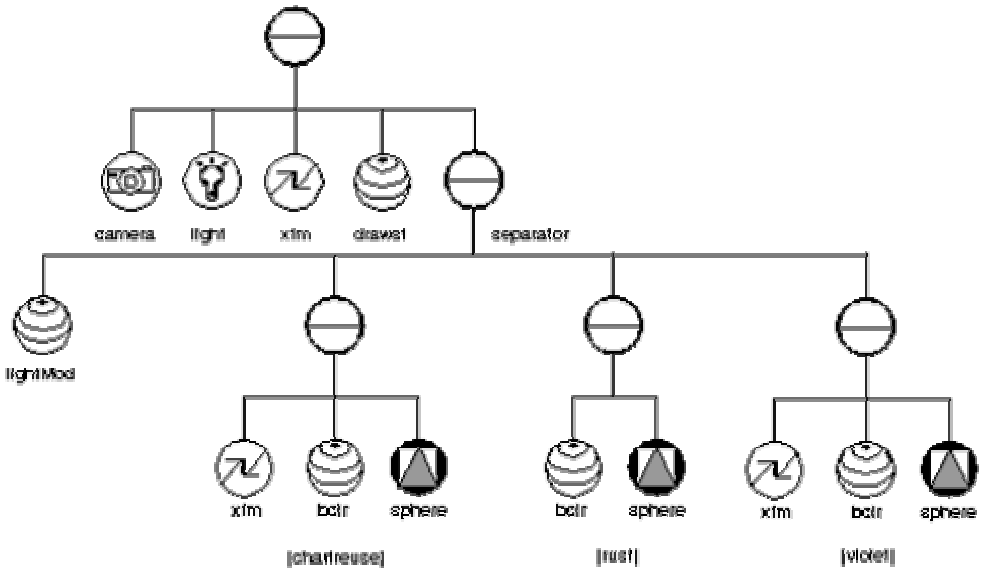
- Terreno a mostrar.
- Nivel del agua.
- Coordenadas o display.
- Punto de luz.

De igual modo, el terreno a mostrar será también un árbol. Dado que pensamos incluir vistas de cámara en 3D para la siguiente iteración, creemos conveniente dejar preparado el prototipo para realizar el renderizado correcto. Por tanto, a la hora de renderizar un terreno, lo primero que haremos será renderizar la vista de cámara, lo cual fija la vista del terreno, esto es, las coordenadas desde las que vamos a ver el terreno. Una vez fijado el campo de visualización, debemos mostrar aquellos elementos que necesitemos, estos son, la luz, el icono que muestra el punto de luz (si es visible), el terreno propiamente dicho (gris) y el cielo.

Para poder renderizar la escena al completo utilizamos los patrones Visitor e Iterator. Un objeto se encarga de recorrer toda la estructura de la escena, haciendo uso del iterador obtenido y visita cada uno de los nodos, realizando la operación correcta en cada caso. En una primera pasada cuenta el número de pasos necesarios para renderizar la escena. Cuando se haya completado esta operación, cambiará la operación que se realice al visitar por la de renderizar.

### 3.4.2 Grafo de escena

**¿Qué es un grafo de escena?**




Un grafo de escena es básicamente un árbol. Se representa con la raíz en la parte de arriba y las hojas en la parte de abajo. Comienza con un nodo que define el mundo virtual, ya sea en 2D o en 3D.

El mundo se divide entonces en una jerarquía de nodos que representa la agrupación espacial de los objetos, posiciones, animaciones o deficinión de relaciones lógicas entre los objetos. Las hojas del grafo de escena representan los objetos físicos en sí, las propiedades del material y su forma geométrica.

Un grafo de escena no es el motor gráfico de una aplicación, aunque puede formar parte de éste. Su principal propósito es representar los mundos en 3D y dar facilidad a la hora de renderizarlo. Los modelos físicos, la detección de colisiones, etc se dejan a otras librerías que se integrarán con el grafo de escena. El hecho de que el grafo de escena no incluya todas esas características ayuda a que el grafo de escena esté lo menos acoplado posible al código de la aplicación en que se utiliza.

## Beneficios del uso de un grafo de escena

Las principales razones por las cuales se utiliza un grafo de escena son *Rendimiento, Productividad, Portabilidad y Escalabilidad.*

- ***Rendimiento***

  Los grafos de escena proporcionan un marco excelente para maximizar el rendimiento gráfico. Un buen grafo de escena emplea dos técnicas, poda de los objetos que no se ven en la escena y ordenación del estado de las propiedades de los objetos, tales como texturas y materiales, de modo que objetos similares se dibujen juntos. Sin esa poda, la CPU, los buses y la GPU, quedarán sobrecargados por la cantidad de datos que necesitan para representar las escenas con precisión. La estructura jerárquica del grafo de escena hace que esta poda se procese eficientemente, por ejemplo, una ciudad entera podría quedar podada con unas pocas operaciones. Sin la ordenación del estado, los buses y la GPU estarán cambiando todo el rato su estado, bloqueando el pipeline y empeorando la productividad gráfica. Como las GPU se están haciendo más y más rápidas, el coste de bloquear el pipeline gráfico está creciendo, por lo cual los grafos de escena se están convirtiendo en algo muy importante.

- ***Productividad***

  Los grafos de escena requieren gran parte del trabajo duro necesario para desarrollar aplicaciones gráficas de alto rendimiento. El grafo de escena se encarga de los gráficos por ti, reduciendo lo que serían cientos de líneas en OpenGL a unas simples llamadas. Más allá, uno de los conceptos más poderosos en la programación orientada a objetos (POO) es la de la composición de objetos, encajada en el patrón de diseño Composite, que encuadra la escena perfectamente en la estructura de árbol y la hace un diseño altamente flexible y reutilizable – lo cual se reduce a que puede ser fácilmente adaptada para diseñar nuevas aplicaciones.

- ***Portabilidad***

  Encapsulan gran parte de las tareas de bajo nivel para renderizar gráficos y leer y escribir datos, reduciendo e incluso erradicando el código específico de la plataforma que requiere tu propia aplicación. Si el grafo de escena es portable, pasar de una plataforma a otra puede ser tan sencillo como recompilar el código.

- ***Escalabilidad***

  Acorde con ser capaz de manejar dinámicamente la complejidad de las escenas automáticamente, también hace posible manejar complejas configuraciones de hardware, tales como clusters o máquinas

destinadas a gráficos, o sistemas multiprocesador. Un buen grafo de escena permitirá al desarrollador concentrarse en su propia de aplicación mientras que el marco de renderizado de la escena maneja las configuraciones del hardware.

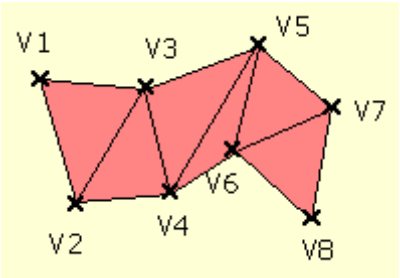
### 3.4.3 Proceso de visualización

Nos centramos ahora en el modo en que se genera el terreno y se visualiza por pantalla. Como ya sabemos, un terreno está formado por filas y columnas. Dado que podemos seleccionar en cada momento la resolución que deseamos utilizar (porcentaje de filas y columnas que se usan para generar el terreno), debemos llevar cuenta de qué filas y qué columnas del total se corresponden con nuestras filas y columnas para una determinada resolución.

Así, suponiendo que tengamos un DEM de 100x100 , para una resolución del 25% la primera columna que deberíamos mostrar sería la 25 real. Siempre que cambiamos la resolución se almacena esa correspondencia.

Además, no siempre se recorren todas las filas y las columnas. Entra en juego aquí el concepto de región visible. Una región visible está determinada por cuatro porcentajes normalizados entre 0 y 1, que delimitan la esquina inferior izquierda y superior derecha del total del tamaño del dem que son visibles.

Dicha región visible se puede modificar por medio del control crop.




A la hora de generar el terreno, utilizamos la técnica GL_TRIANGLES_STRIP, que permite ir definiendo los vértices generando pequeños triángulos y reutilizando puntos inferiores y anteriores para generar el nuevo triángulo. De nuevo, dichos vértices sólo se generan entre aquellas filas y columnas visibles.

Para determinar la posición en pantalla de un punto correspondiente a una fila y a una columna se hace una pequeña transformación de coordenadas. Se toman como base las coordenadas que delimitan el límite del terreno (bounding box). Se utiliza entonces el número de filas y columnas virtuales (dependen de la resolución utilizada), y el desplazamiento generado al dividir el número total de filas/columnas entre el número de columnas/filas virtuales. Utilizando ese desplazamiento, y el propio desplazamiento en coordenadas espaciales que proporciona el DEM (definen el aspecto o distancia existente entre dos puntos), obtenemos el punto en que se debe colocar el vértice calculado.

La coordenada Z del vértice depende del valor en ese punto para el DEM y de la exageración vertical definida en cada momento.

Para obtener el color de un vértice utilizamos una ecuación similar a la utilizada por OpenGL, pero sin tener en cuenta el tipo de material utilizado ni la luz emitida por el objeto. A medida que vamos generando las bandas de triángulos, utilizamos los vértices para determinar la normal en cada punto (gouraud si procede). Se calcula entonces el color para el punto utilizando el shader correspondiente (y modificando la normal y coordenadas del vértice si es necesario dada la naturaleza del shader). Dicho color queda atenuado en función de las características definidas para cada uno de los componentes de la luz, y de la incidencia de ésta sobre este punto.

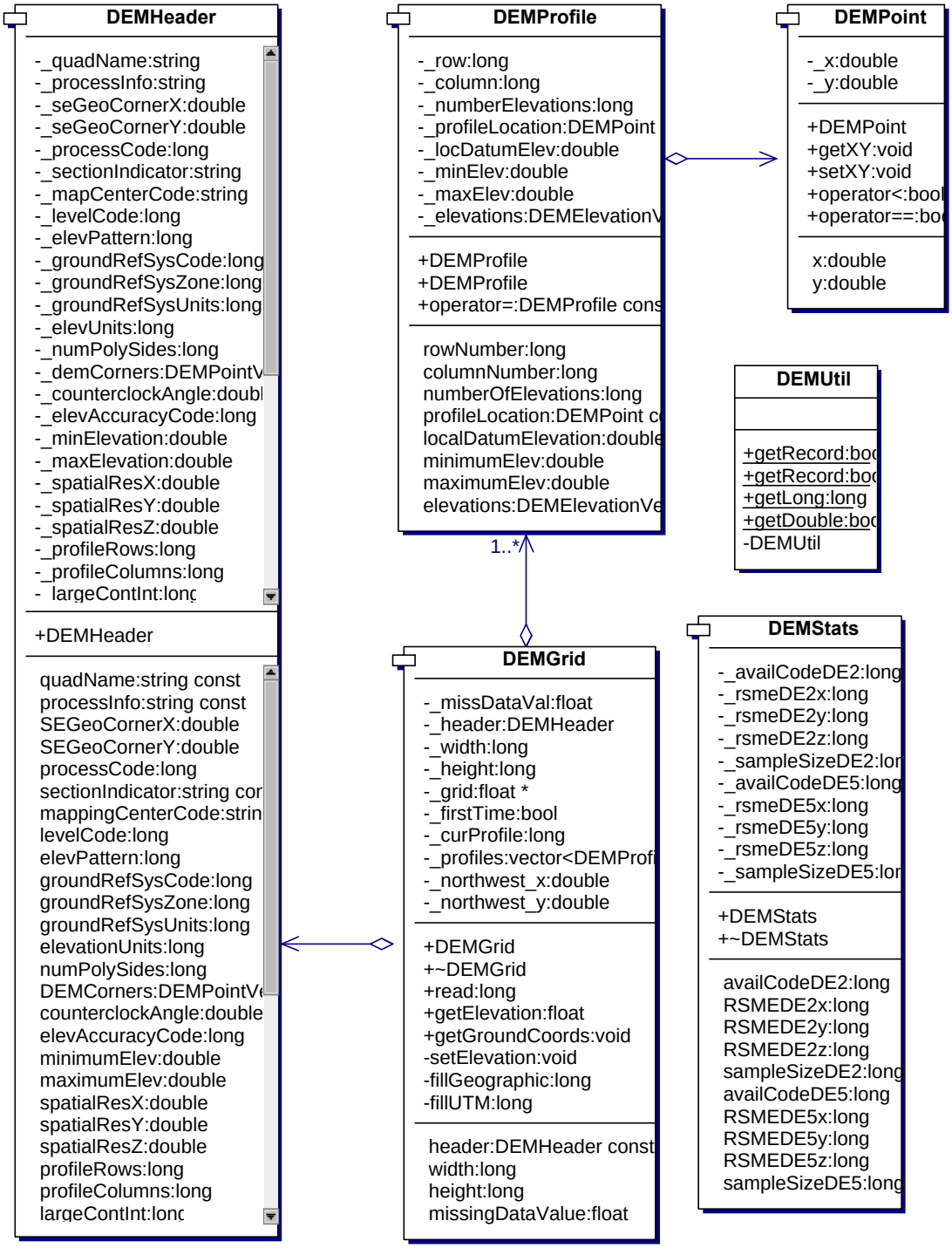
Una vez obtenido el color para el vértice, se comprueba la existencia de agua. Si la altura del vértice es menor que el nivel del agua, el color para ese punto se interpola con el del agua, utilizando el parámetro alpha del agua. Sin embargo, este color sólo se utiliza para guardarlo en la textura. La apariencia real de agua se consigue pintando una del color del agua y que tiene como límites los mismos que la región visible mostrada en ese momento.

Si tras esto hay que calcular sombras, se calculan y si resulta que el punto se encuentra en sombra, lo que se devuelve es el color resultante del producto entre el obtenido del shader y el porcentaje del mismo para luz ambiente.

Cada vez que generamos el color para un vértice, almacenamos dicho color en el buffer que utilizamos más tarde para almacenar la textura. Dicha textura sólo se actualiza cada vez que se renderiza la imagen en modo estático.

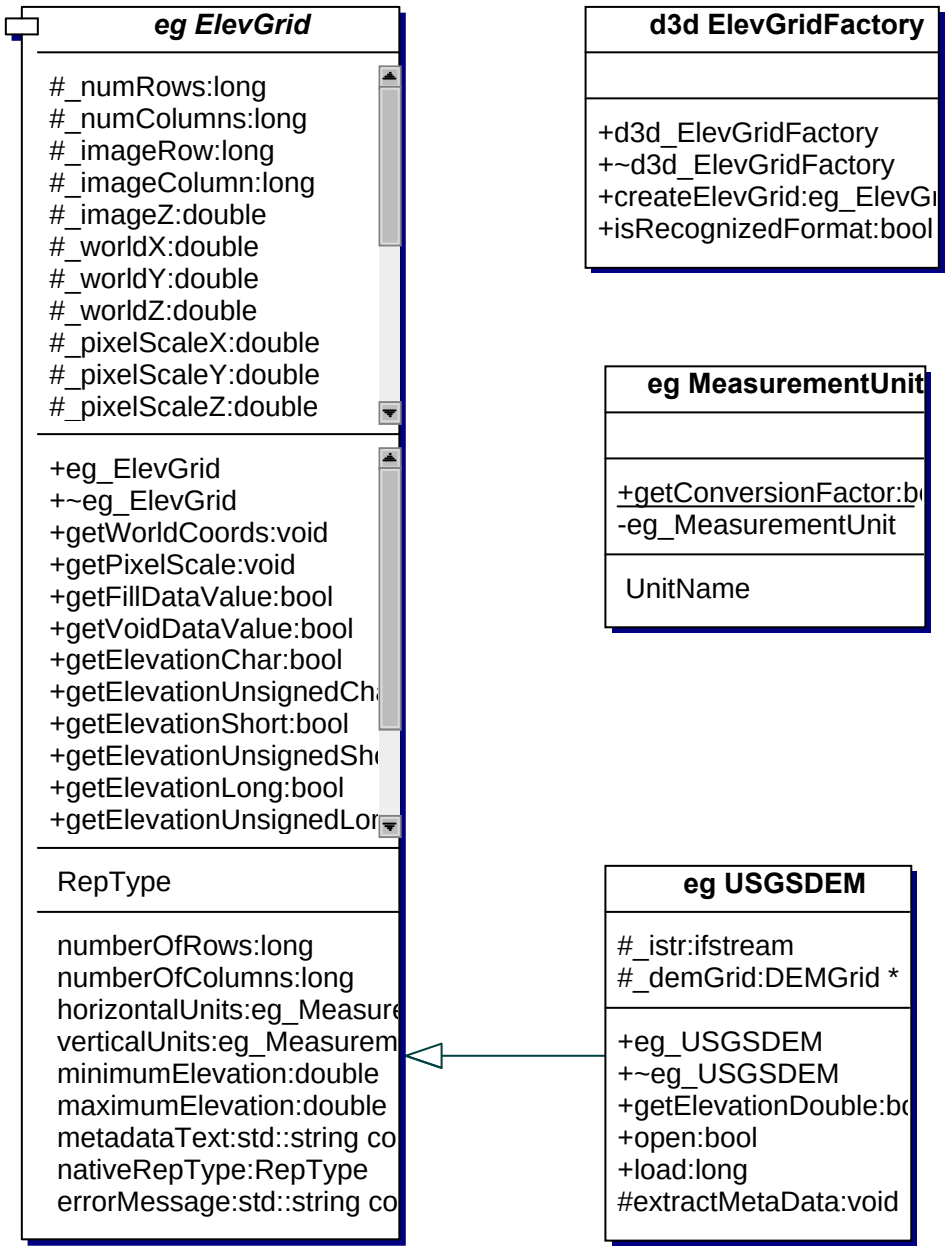
# 3.5 *Diagrama de clases*

## 3.5.1 Dem




Dentro de este directorio podemos encontrar las clases relacionadas con la lectura de archivos DEM. Así, encontramos clases correspondientes al DEMGrid, estructura que está formada por una cabecera (DEMHeader) y un conjunto de puntos (DEMPoints) resultantes de leer un fichero DEM.

### 3.5.2 ElevGrid




En este directorio se encuentra, por un lado, la clase eg_elevGrid que es el objeto en el que se almacenará el modelo de elevación digital para su manejo durante el transcurso de la aplicación. Aparece aquí el primer patrón aplicado, el patrón Factoría, explicado en el apartado “Patrones aplicados”. En este caso se utiliza para que se pueda recuperar de archivo no solo un fichero DEM USGS, sino que con pocas modificaciones la aplicación sería capaz de recuperar cualquier otro tipo de fichero. Como resultado de la aplicación de este patrón aparecen las clases d3d_elevGridFactory, eg_elevGrid. eg_USGSDEM y eg_PGM.

Cabe resaltar tres métodos de la clase eg_ElevGrid. Dichos métodos son los que hemos de implementar de forma particular para cada una de las clases que deriven de nuestra clase base, y que se acomoden al formato determinado para el cual se están creando.

- **Método Open**: Se debe llamar a éste método siempre que se desee abrir un archivo para ser leído. Se encarga de comprobar la existencia del archivo así como de leer el encabezado o los parámetros necesarios para cargar correctamente los datos. Asimismo, fija valores iniciales para aquellos parámetros que lo necesiten.

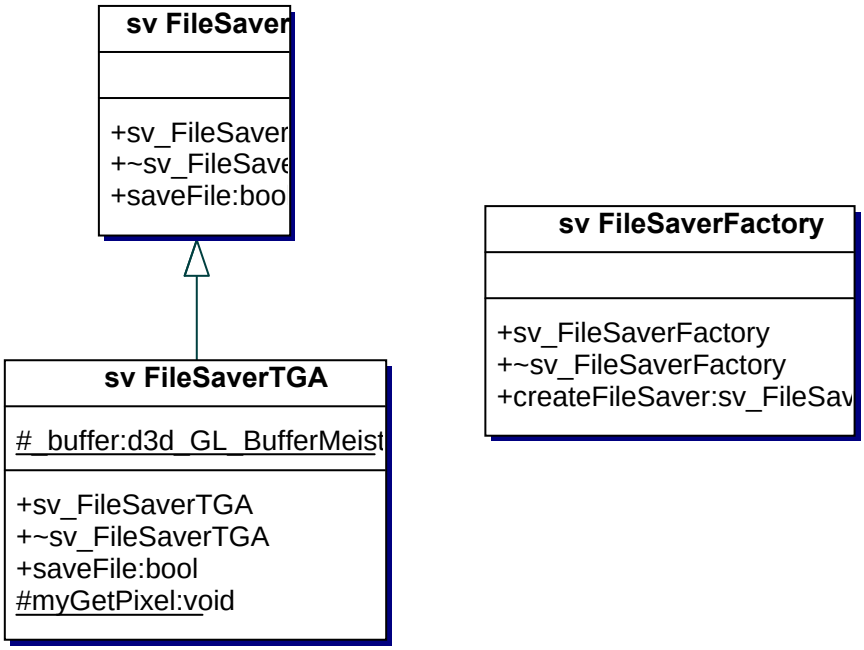
- **Método Load:** Permite especificar un parámetro en el que se indica si se desea cargar incrementalmente los datos o cargarlos todos de una sóla vez. Si se especifica carga incremental, se deben realizar sucesivas llamadas a éste método, que devuelve el número de pasos que faltan para terminar. En este método se obtienen los valores que se guardan en el buffer utilizado por la clase y que representan las elevaciones del dem.

- **Método getElevationDouble:** Puesto que cada clase puede utilizar la estructura de datos más oportuna en cada momento para almacenar los datos leídos, es necesario que sea la propia clase la encargada de definir cómo se han de devolver dichos valores.

La clase utilizada para cargar archivos en formato USGS DEM es eg_USGSDEM. Esta clase utiliza una librería para acceder y cargar todos los datos necesarios.

La clase utilizada para cargar archivos en formato PGM es eg_PGM. En el método *open* de dicha clase es donde se distingue el tipo de archivo que se está intentando leer, y se inicializan los parámetros necesarios. En el método *load,* se leen los datos según sea el tipo del archivo. Si se trata de un archivo de tipo P2, es decir, en ascii, se permite utilizar el parámetro incrementalLoad. En archivos en formato binario (tipo p5), no se utiliza dicho parámetro, pues la carga siempre es directa.

### 3.5.3 FileHandling

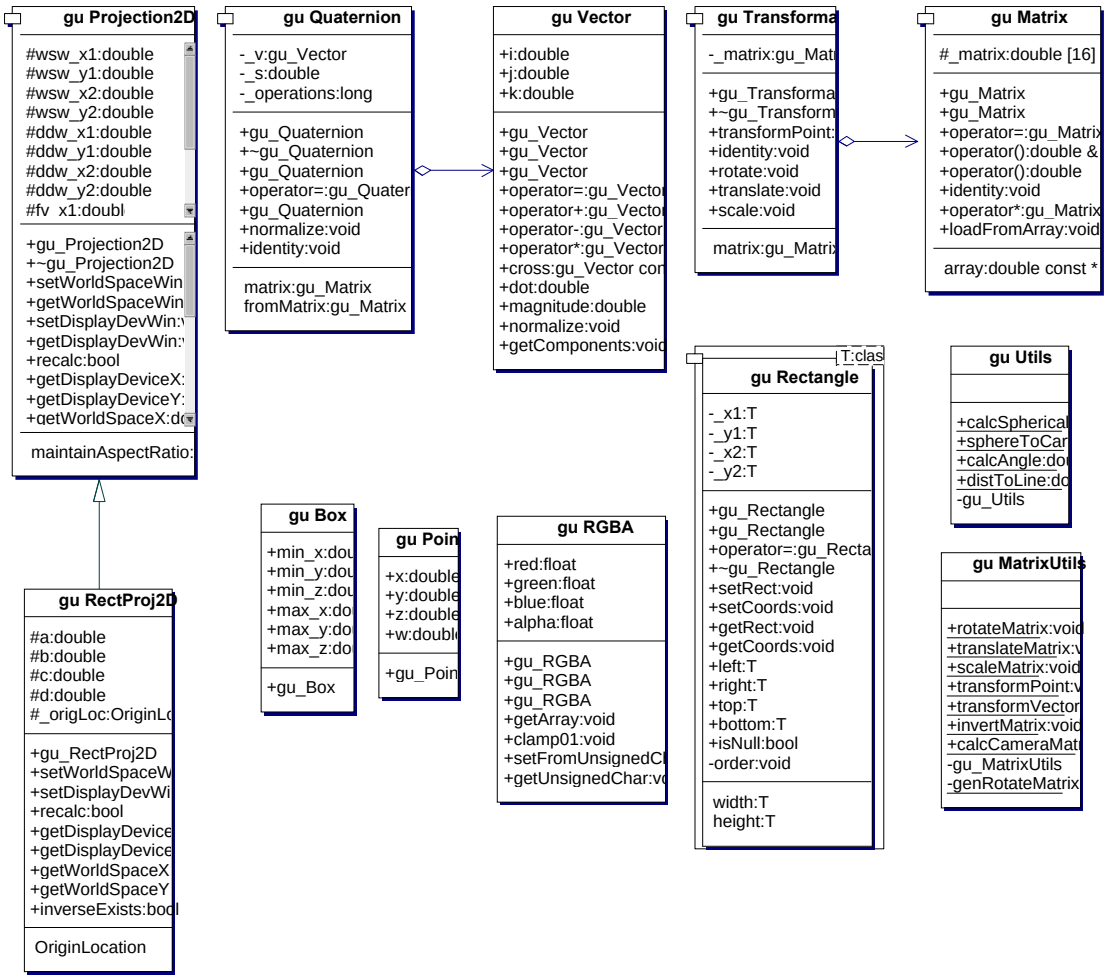



Aparecen aquí las clases relativas a la acción de guardar el fichero. Actualmente la aplicación permite salvar la textura aplicada al DEM con formato TGA. Aquí también se ha aplicado el patrón Factoría, lo que favorecerá que en un futuro se pueda extender fácilmente el código para poder salvar más formatos de archivo.

La clase sv_FileSaverTGA es una implementación de la clase sv_FileSaver. La factoría devuelve un objeto de este tipo si se le especifica como parámetro la extensión “tga”. Dicho objeto utiliza el buffermeister de la aplicación (buffer de píxeles) y las rutinas declaradas en sv_TGASave.

Están situados aquí también los archivos dir.h dir.cpp que son los encargados de facilitar un método de lectura de los ficheros existentes en un determinado directorio. La complicación de esta clase radica en que al ser una aplicación multiplataforma, se ha de implementar el listado de fichero tanto para sistemas operativos tipo Windows como para los tipo Linux. Esto se consigue utilizando las macros definidas por el compilador. Según el sistema utilizado, se usa una función u otra.

### 3.5.4 GraphUtils




En este grupo se encuentran las utilidades gráficas. Desde colores como gu_RGBA pasando por puntos gu_Points, cajas gu_Box o matrices gu_matrix que son utilizadas para realizar transformaciones gu_Transformation. Estos objetos son utilizados a lo largo del código de manera auxiliar por el resto de clases.

Las clases más importantes se explican a continuación.

La clase gu_Transformation es la base para todos los objetos que se van a utilizar para pintar la escena. Contiene una matriz que puede ser transformada mediante las rutinas escritas en la clase utilería gu_MatrixUtils.

La clase gu_Vector permite definir un tipo abstracto de datos que representa vectores. Se pueden realizar todas las operaciones típicas con vectores: producto escalar, producto vectorial, suma, resta, etc.. Se utiliza esta clase en multitud de sitios en la aplicación, ya sea para definir la dirección de un punto de luz, como de una normal, etc…

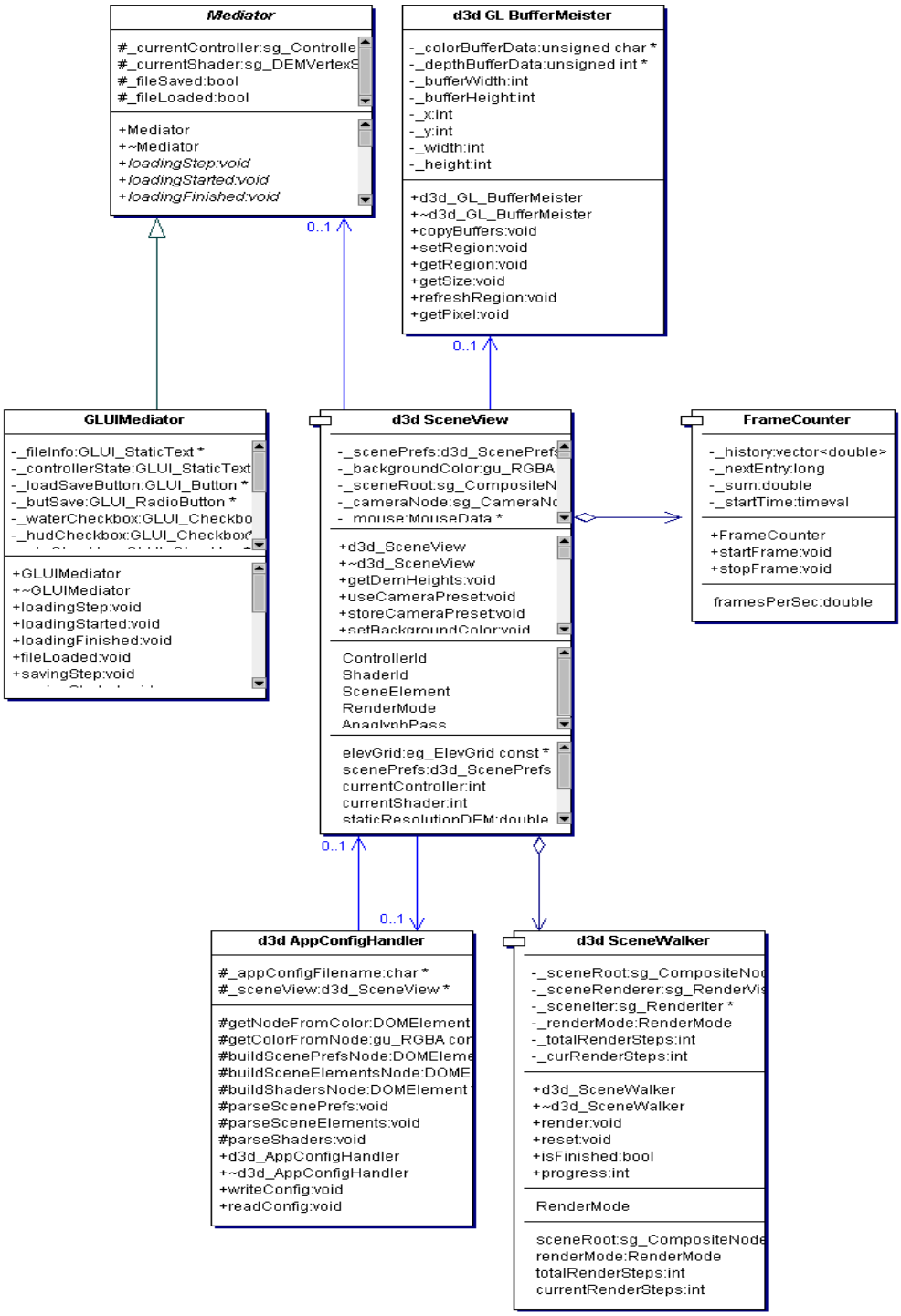
La clase gu_RGBA encapsula la información necesaria para representar un color, con los parámetros rojo, verde, azul y alpha. Se pueden realizar operaciones aritméticas entre colores. Ésta clase se utiliza también para representar los porcentajes para cada componente de la luz, esto es, ambient, diffuse y specular.

La clase gu_Point se utiliza cada vez que se va a definir un vértice. A dichos puntos se les puede sumar un vector de dirección para desplazarlos.

Mediante la clase gu_Rectangle podemos definir un rectángulo y representarlo como un objeto. Para representarlo, utilizamos las coordenadas que definen la esquina inferior izquierda y la superior derecha. Este rectángulo se utiliza a la hora de definir que región del DEM es visible.

### 3.5.5 Gui

Mostramos aquí lo más representativo de estas clases, siendo el diagrama completo demasiado extenso para mostrarlo.



Todas las ventanas han sido implementadas utilizando las librerías GLUT y GLUI.

La aplicación consta del interfaz principal y ventanas de configuración y auxiliares.

- **Interfaz Principal**

Definido en la clase Mainwindow. No se trata de un objeto, sino que está definido utilizando C estándar, sin encapsulación. Dicha ventana se crea en el método principal de la aplicación (main), al tiempo que se crea la aplicación. Mantiene referencias al objeto de tipo sceneView y al Mediator que ha de utilizar para configurar cada uno de sus componentes.

- **Ventana**

Es la clase base de la cual heredan todas las ventanas utilizadas. Dado que GLUI y GLUT necesitan utilizar funciones que se registren cada vez que hay un evento (“callback”), no podemos utilizar métodos definidos en un objeto, puesto que el cualificador del tipo del método no es equivalente al esperado por las funciones de las librerías.

Sin embargo, es conocida la utilidad y beneficio del uso de programación orientada a objetos. Para poder encapsular las funciones utilizadas por nuestras ventanas en clases, hemos de definir cada clase de tal modo que sólo exista una instancia de esa clase en cada momento. Dicho objeto se mantiene en una variable estática de la clase, con el propósito de que los métodos estáticos de la clase llamen a métodos propios de ese objeto.

Todas las llamadas a funciones GLUT/GLUI utilizan como parámetro un método estático definido en la clase. Este método estático hace referencia al objeto almacenado en la variable estática de la clase.

- **Ventanas de configuración de shaders**

Todas las ventanas que han de configurar los colores de un shader, siguen el mismo procedimiento. Se crea una ventana GLUT a la que se añaden los paneles GLUI necesarios. Mantenemos el vector de colores que hemos de configurar en cada clase. Utilizamos un spinner por cada componente del color, y cada vez que se modifican se actualiza el color de fondo de la ventana.

- **Mediator y GLUIMediator**

Utilizada para controlar la interacción de la aplicación con el interfaz. Siempre que hay un cambio que pueda afectar al interfaz, el objeto SceneView notifica este cambio al Mediator, que actualiza los elementos del interfaz en función de esos cambios. GLUIMediator es una implementación específica para el interfaz creado con GLUI.

- **d3d_GL_BufferMeister**

Mantiene en cada momento la información referente a los píxeles que se muestran por pantalla. Se utiliza como buffer accesible desde la clase principal para obtener los píxeles de pantalla o del buffer de textura y guardar así la información en un archivo.

- **d3d_SceneWalker**

Es la clase encargada de recorrer el grafo de escena y ordenar a cada objeto renderizarse. Mantiene la cuenta de los pasos necesarios para completar el renderizado y debe situarse en la raíz del grafo de escena cada vez que se desea recorrer el grafo. Hace un uso extensivo de los iteradores y los visitor.

- **FrameCounter**

Realiza los cálculos necesarios para determinar un framerate adecuado al mover la imagen. Utilizando este objeto se define la resolución interactiva que se debe utilizar.

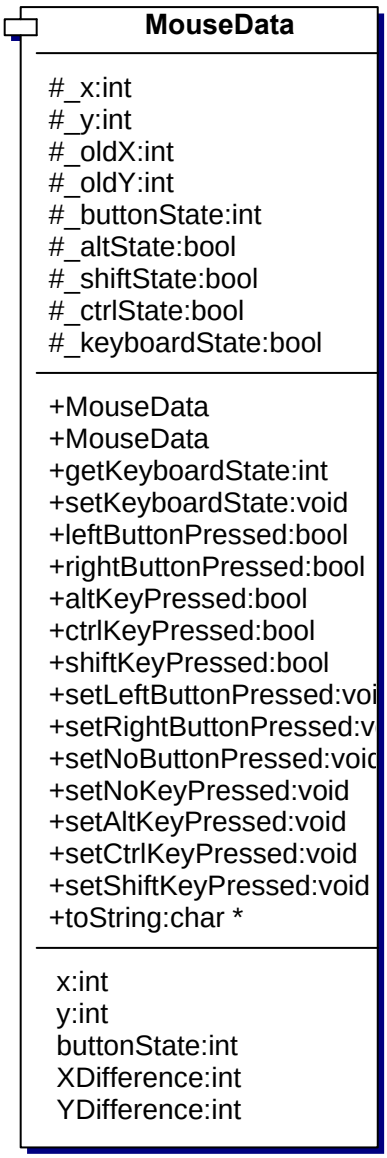
- **d3d_AppConfigHandler**

Encargada de realizar las operaciones relativas a la gestión de la configuración de la aplicación. Se define como clase “amiga” de la clase sceneView para poder acceder a los elementos que debe configurar. Implementada íntegramente usando Xerces XML. Permite leer y guardar la configuración.

- **d3d_SceneView**

La clase principal de la aplicación, y sobre la cual reside todo el control de la escena mostrada en la ventana. Crea el grafo de escena y mantiene referencias a cada uno de los elementos que entran en juego a la hora de pintar, esto es, los shaders, los controladores, la resolución, elementos visibles, etc… Siempre que se desea pintar, se fija el tipo de renderizado deseado (static/interactive), y se ejecuta el método startRender(). Dicho método prepara al sceneWalker y demás elementos que debe utilizar y comienza a llamar al método render(), que es el que realmente pinta los objetos mediante el scenewalker. Además, es la encargada de crear las ventanas de configuración de los shaders cuando sea necesario.

### 3.5.6 Mouse




En este grupo existe únicamente una clase MouseData que es la encargada de representar al ratón y los eventos que pueden tener lugar debido a su manejo. Este objeto MouseData también tiene en cuenta las teclas especiales del teclado como Shift, ctrl. o Alt.

### 3.5.7 SceneGraph

En este paquete se definen los elementos que conforman el grafo de escena, así como aquellas clases útiles para generarlo y recorrerlo. No se incluye captura de la estructura de clases de este paquete dada la gran extensión que tiene.

Nos centramos en las clases más significativas:

- **sg_DEM**

Utilizada para generar la representación del archivo DEM en 3D. Utiliza la información fijada en el sceneView para determinar las coordenadas de cada uno de los puntos que conforman el DEM. Contiene información acerca del shader actual, el nivel del agua, la región visible, resolución estática utilizada y del objeto eg_ElevGrid que se ha de representar.

La malla de puntos que componen el dem se genera mediante GL_TRIANGLES_STRIP. La creación de los vértices, así como determinar qué color se ha de asignar a cada uno ya quedó explicado con anterioridad. Se puede definir el método de coloreado de vértices que se desea utilizar, esto es, si se quiere utilizar FLAT (coloreando el polígono completo o mostrando la malla) o SMOOTH (con la opción de usar Gouraud).

- **sg_Controller**

Los controladores son los encargados de procesar la interactividad con el usuario. Cada movimiento hecho con el ratón es traducido por el correspondiente controlador, que se encarga de aplicar los cambios en el estado de la escena. de cada controlador. Los controladores reciben la información necesaria del ratón y procesan la entrada, actualizando los parámetros que almacenan. Todos los cambios se llevan a cabo en el método sync(). Aquellos controladores que implican movimiento de un objeto (luz, cámara y flyby), operan con la matriz del objeto, produciendo transformaciones.

- **sg_DEMVertexShader**

Son los objetos utilizados para aplicar color a un vértice. Necesitan como información las coordenadas del punto en que se va a pintar el vértice, la normal en ese punto y los puntos utilizados para generar esa normal. Dicha información se guarda en un struct. Aplican las transformaciones necesarias y devuelven el color correspondiente al vértice, sin tener en cuenta la presencia de agua o de luz.

- **sg_SceneNode y sg_ShapeNode**

Son la base de los elementos que forman el grafo de escena. El nodo de escena permite determinar qué objetos tiene por debajo y define métodos para saber si se encuentra activo un determinado elemento. Además, aporta funcionalidad para compatibilizar el nodo con el uso de patrones. Los elementos sg_ShapeNode, implementan aquellos objetos que se pueden representar en una escena, es decir, que tienen propiedades geométricas, de materiales, de colores, etc.

## *3.6 Puntos de extensión*

La facilidad de extensión ha sido uno de los puntos clave del diseño de la aplicación. Los puntos de extensión permiten añadir nueva funcionalidad y nuevos componentes a la aplicación, de forma fácil y rápida y de modo que el impacto sufrido por el resto de componentes de la aplicación a la hora de modificar el código sea mínimo.

Identificamos ahora aquellas partes de la explicación que pueden facilitar su ampliación:

- **Tipos de archivo leídos:**

  Se pueden incluir nuevos tipos de archivo partiendo de la clase eg_ElevGrid e implementando los métodos para abrir archivo, cargar archivo y obtener datos para una determinada coordenada. Una vez creado el nuevo tipo de archivo de elevación, se ha de registrar en la factoría con la extensión correspondiente.

- **Tipos de archivo guardados:**

  Se pueden guardar nuevos tipos de archivo siguiendo el método anterior, pero con la factoría para guardar.

- **Elementos de escena:**

  Se pueden incluir nuevos elementos en la escena de forma sencilla. Desde incluir nuevas luces hasta nuevas formas geométricas a representar, pasando por controladores, shaders, etc. Con todo se trabaja utilizando interfaces que hagan independiente el uso del objeto definido en cada momento del código y que por tanto faciliten crear nuevos objetos similares.

- **Ventanas de la aplicación:**

  Utilizamos una clase que encapsula las llamadas necesarias a la librería GLUT para crear ventanas. A partir de la clase Window, se pueden crear ventanas de configuración para cada shader, o para mostrar texto, etc. Basta con especificar las coordenadas y el tamaño, y redefinir en cada caso los métodos necesarios según el fin de la ventana. Así, por ejemplo, la lógica de todas las ventanas que configuran colores es similar, bastaría con seguir el procedimiento utilizado para crear cualquiera de ellas y tendríamos una nueva.

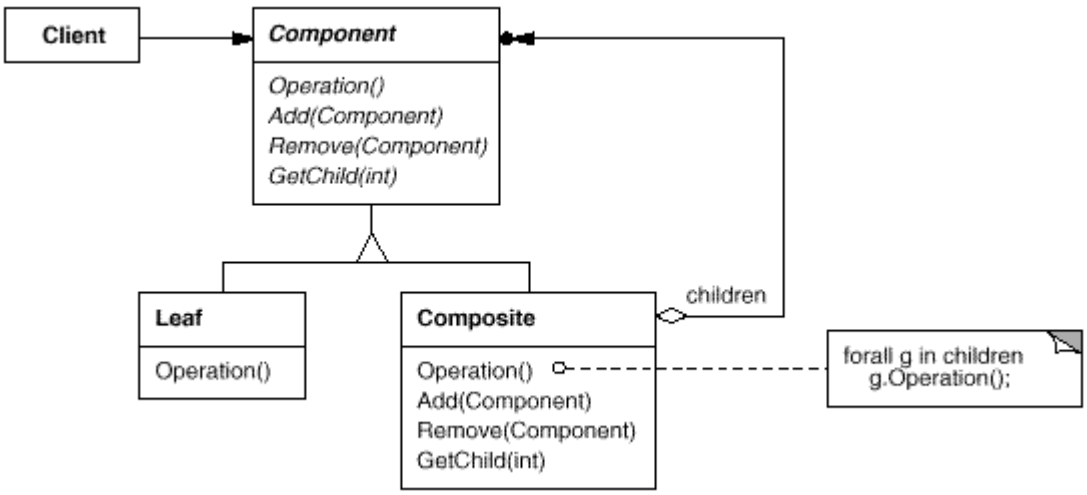
El uso de patrones también facilita la extensibilidad y reutilización. Se exponen a continuación los utilizados en la aplicación.

Los patrones utilizados en el diseño de la aplicación han sido Composite, Visitor, Factory Method, Iterator y Mediator. Cada uno de los campos en los que se han aplicado patrones constituye un punto de extensión de la aplicación resultando así mucho más fácil su ampliación para introducir posibles mejoras o reflejar posibles cambios en los requisitos. Pasamos a desarrollar cada uno de ellos y explicar su implementación concreta. [L01]

### 3.6.1 Composite

El objetivo de este patrón es componer objetos en estructuras de árbol para representar jerarquías. Este patrón permite a los clientes tratar objetos simples y compuestos de manera uniforme.

## ~ Estructura




## ~ Participantes

- Component: declara el interfaz para los objetos en la composición. Implementa el comportamiento por defecto para todas las clases. Define un interfaz para acceder y manejar sus componentes hijo.
- Leaf: representa los objetos simples en la composición. Una hoja no tiene hijos.
- Composite: define el comportamiento para los componentes con hijos. Almacena los hijos e implementa las operaciones de Component relativas a los hijos.
- Client: manipula los objetos de la composición a través del interfaz Component.

## ~ Aplicación concreta

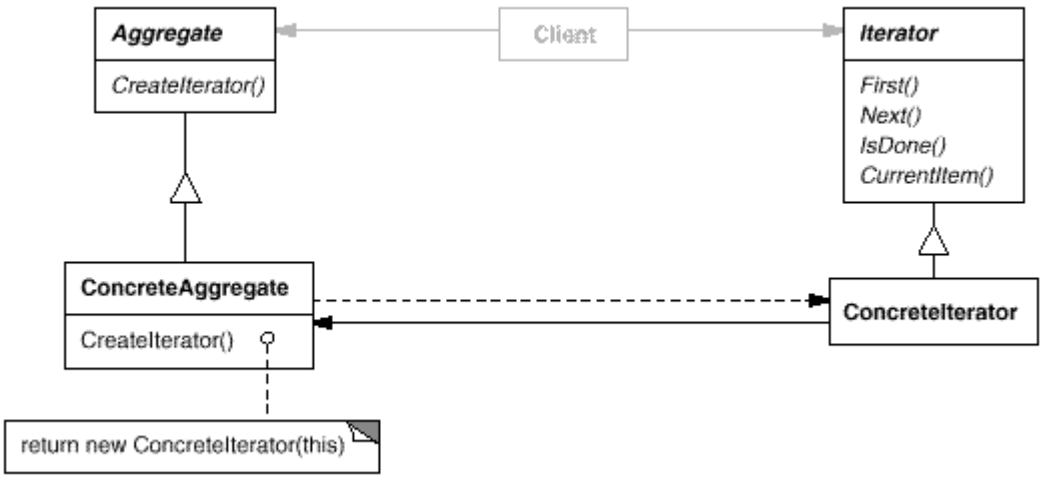
En la aplicación utilizamos este patrón para representar los objetos que forman parte de la escena Dichos objetos definen una jerarquía que se ha de seguir a la hora de pintar mediante OpenGL.

El rol “Component” lo asume la clase sg_SceneNode. El árbol definido mediante los objetos composite puede ser recorrido mediante un iterador y aplicar una determinada operación en cada nodo mediante un visitante.

### 3.6.2 Iterator

El objetivo de este patrón es facilitar un modo de acceso a los elementos de un objeto compuesto sin revelar su representación interna.

## ~ Estructura




## ~ Participantes

- Iterator: define un interfaz para acceder y atravesar elementos
- Concrete Iterator: implementa el interfaz Iterator y almacena la posición actual en el recorrido sobre el objeto Aggregate
- Aggregate: define un interfaz para crear un objeto iterador
- Concret Aggregate: implementa el interfaz de creación Iterator para devolver una instancia del Concrete Iterator apropiado.

## ~ Aplicación concreta

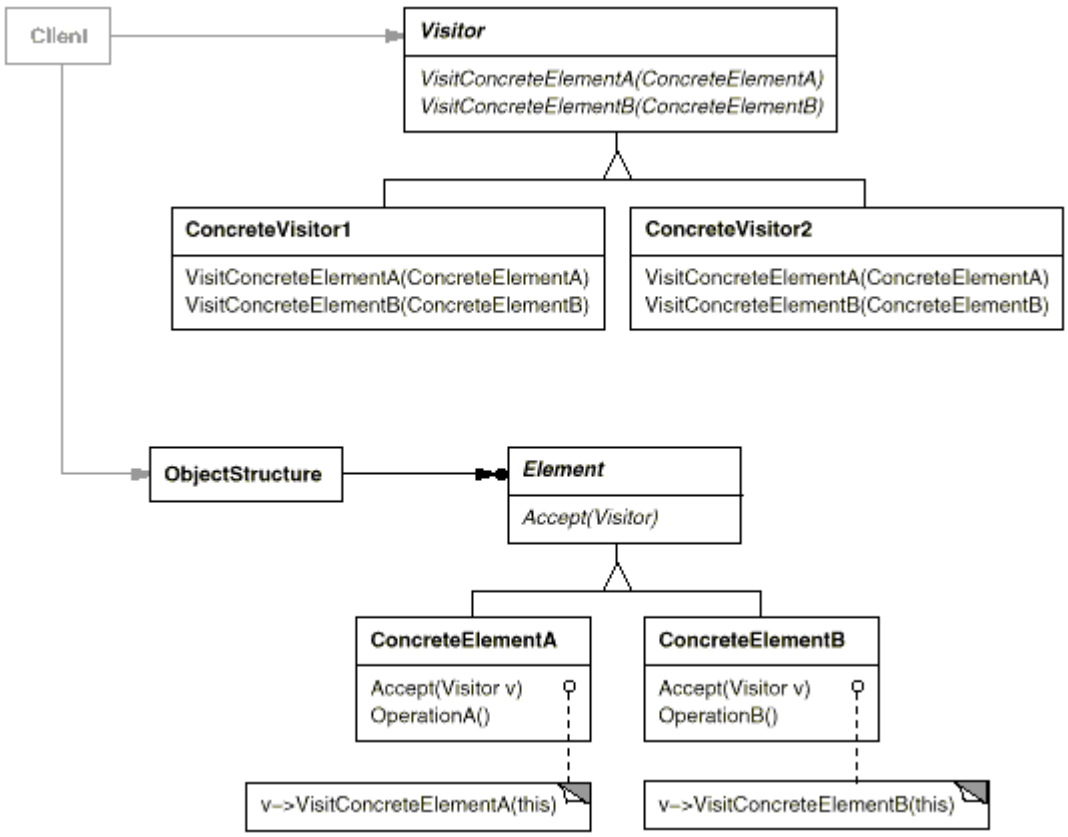
Utilizamos el patrón iterator para definir el orden en que se puede recorrer la escena. A partir de un iterador genérico derivamos un iterador para recorrer nuestra estructura de árbol y a partir de éste derivamos un tipo de iterador distinto según se recorra el árbol en preorden, inorden o postorden.

(Utilizaremos una modificación del iterador en preorden para renderizar la escena).

### 3.6.3 Visitor

Se utiliza para implementar operaciones que deben ser aplicadas en los elementos de una estructura. Visitor permite definir una nueva operación sin modificar las clases de los elementos en los cuales opera.

## ~ Estructura




## ~ Participantes

- Visitor: declara una operación Visitar para cada clase de ConcreteElement en el objeto estructura. El nombre y la signatura de la operación identifica la clase que envía la solicitud de visita al Visitor. Esto permite al visitante determinar qué clase concreta de Element ha de ser visitada. Así, el visitante puede acceder directamente al elemento a través de su interfaz.
- ConcreteVisitor: implementa cada operación declarada en la clase Visitor. Cada operación implementa un fragmento de un algoritmo definido para la correspondiente clase de objeto en la estructura. ConcreteVisitor facilita el contexto para el algoritmo y almacena su estado local. Este estado a menudo almacena resultados obtenidos durante la visita de la estructura.

- Element: define una operación Accept que recibe un Visitor como argumento
- ConcreteElement: comportamiento análogo
- ObjectStructure: puede enumerar sus elementos. A veces provee un interfaz a alto nivel que permite al Visitor recorrer sus elementos. Puede ser además, un Composite o una colección como un conjunto o una lista.

## ~ Aplicación concreta

Con el uso de este patrón lo que hemos conseguido es abstraer la distinción de objetos a la hora de aplicar una determinada operación sobre los elementos de la escena. Las posibles operaciones que podemos realizas son:

- Renderizar un nodo
- Contar número de pasos que hacen falta para renderizar un nodo
- Convertir el nodo a una cadena de caracteres.

Los roles en esta aplicación concrete van como sigue:

- El objeto Visitor en este caso particular se llama sg_SceneVisitor

- Concrete Visitor: tenemos tres implementaciones

    - Sg_StepCountVisitor que cuenta los pasos que hacen falta para renderizar.
    - Sg_TreePrintVisitor que genera una representación escrita en ASCII del árbol.
    - Sg_RenderVisitor que renderiza el árbol.

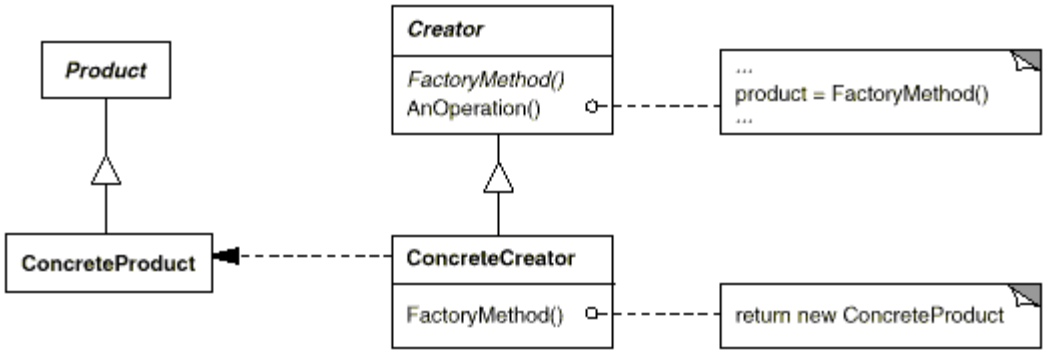
- El objeto Element en este caso está formado por sg_SceneNode y sus clases hijas:

    - Sg_CompositeNode
    - Sg_ManipulatorNode
    - Sg_CameraNode

### 3.6.4 Factory Method

Define un interfaz para crear un objeto, pero permite a las subclases decidir qué clase instanciar. Este patrón permite una instanciación delegada en las subclases.

## ~ Estructura




## ~ Participantes

- Product: define el interfaz para los objetos que el método factoría crea.
- ConcreteProduct: implementa el interfaz Product
- Creator: declara el método factoría, el cual devuelve un objeto de tipo Product. Puede también definir una implementación por defecto del método factoría que devuelve un ConcreteProduct por defecto
- ConcreteCreator: sobrescribe el método factoría para devolver una instancia de ConcreteProduct.

## ~ Aplicación concreta

Este patrón se ha aplicado en bastantes partes del código. Por un lado, existe una factoría para crear los objetos DEM que se han leído de archivo. De este modo, aunque actualmente se leen USGS DEM y PGM, la aplicación es fácilmente extensible para leer archivos SDTS DEM o futuros formatos.

Por otro lado, existe una factoría para guardar a archivo desde la aplicación. Análogamente al caso anterior, actualmente se guardan archivos TGA pero con mínimas extensiones se podría salvar en cualquier otro tipo de formato.

Los roles en esta aplicación concreta se dividen en dos grupos. Por un lado, para la factoría que crea los objetos DEM:

- El producto está formado por la clase eg_ElevGrid

- A su vez, Concrete Product posee dos implementaciones

    - Eg_USGSDEM para los archivos DEM del gobierno de los EEUU
    - Eg_PGM para los ficheros con formato PGM (Portable Greyscale Map)

- El rol de Creator lo asume eg_ElevGridFactory que es la encargada de crear los objetos del tipo adecuado.

Por otro lado, en la factoría encargada de salvar a archivo:

- El producto lo conforma la clase sv_fileSaver

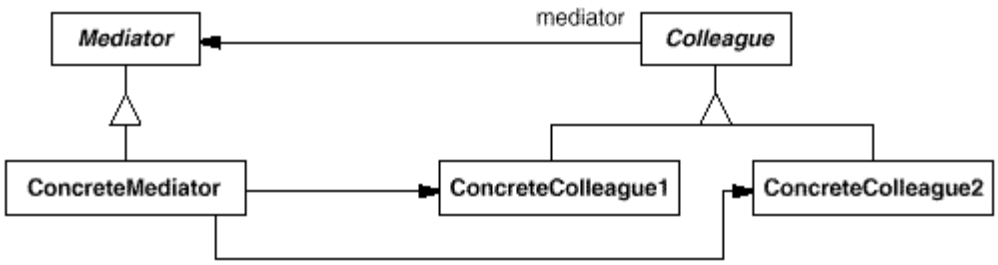
- Concrete Product por ahora solo tiene una implementación, sv_fileSaverTGA.

- El objeto creador en este caso es de tipo sv_fileSaverFactory,

### 3.6.5 Mediator

Este patrón permite la definición de un objeto que encapsula la comunicación entre un conjunto de objetos que interactúan. Mediator facilita el desacoplamiento manteniendo los objetos sin referencia explícita al resto. Así, permite cambiar sus interacciones independientemente.

## ~ Estructura




## ~ Participantes

- Mediator: define un interfaz para comunicarse con objetos Colleague
- Concrete Mediator: implementa un comportamiento cooperativo coordinando objetos Colleague. Conoce y mantiene sus Colleagues.
- Colleague classes: cada Colleague también conoce a su Mediator y se comunica con él cada vez que necesita comunicarse con otro Colleague.

## ~ Aplicación concreta

En este caso el patrón Mediator ha sido utilizado para aislar el interfaz del resto de la aplicación. Cuando se produce algún evento que hace neceesario un cambio de interfaz, este cambio es notificado al mediador que es el encargado de hacer los cambios pertinentes en pantalla. En caso de que en posteriores modificaciones, se decidiera cambiar la implementación del interfaz, esto no afectaría al resto de la aplicación. Este patrón no ha sido aplicado estrictamente sino que se ha realizado una adaptación del mismo a este caso particular.

El rol de “Mediator” en esta implementación lo lleva a cabo la clase Mediator de la cual deriva GLUIMediator que asume las funciones de “Concrete Mediator”.

Los cambios que actualmente se pueden notificar al Mediator y que vienen definidos en dicho interfaz son:

- Notificar el porcentaje de carga de un fichero.
- El proceso de carga de un fichero acaba de comenzar.
- El proceso de carga ha finalizado.
- El fichero X ha sido cargado con éxito
- Porcentaje del proceso de guardado de un fichero
- El proceso para salvar un fichero ha comenzado
- El proceso para salvar un fichero ha finalizado
- El fichero X ha sido salvado con éxito
- El controlador activo ha pasado a ser X
- El shader activo ha pasado a ser X. Si es configurable, se le notifica también
- La resolución seleccionada ha pasado a ser X
- El elemento de escena X ha pasado a ser visible si no lo era antes o viceversa.
- El estado del controlador activo ha cambiado
- La posición de cámara actual ha sido almacenada en la posición X
- El controlador de agua ha sido activado
- El controlador de agua ha sido desactivado
- La posición del ratón ha cambiado
- La región visible del terreno ha cambiado

# 4 Manual de usuario

## 4.1 ¿Qué es KDV?

Kilimanjaro Dem Viewer es un completo visor de modelos de elevación digital (DEM). Carga ficheros de elevación .dem (formato USGS DEM) y .pgm (tanto formato P2 como P5) y permite definir sobre ellos distintos sombreados así como moverse por el terreno o aplicarle ciertos parámetros. Es configurable en el 90% de sus parámetros permitiendo así una libertad casi total al usuario. El programa puede producir como salida la textura aplicada por el usuario al DEM o capturar la pantalla en cualquier momento produciendo en ambos casos un archivo en formato TGA.

## 4.2 ¿Cómo instalar?

Actualmente está disponible un instalador automático que se encargará de configurar el equipo para poder ejecutar correctamente KDV. Existen tres tipos de instalación disponible:

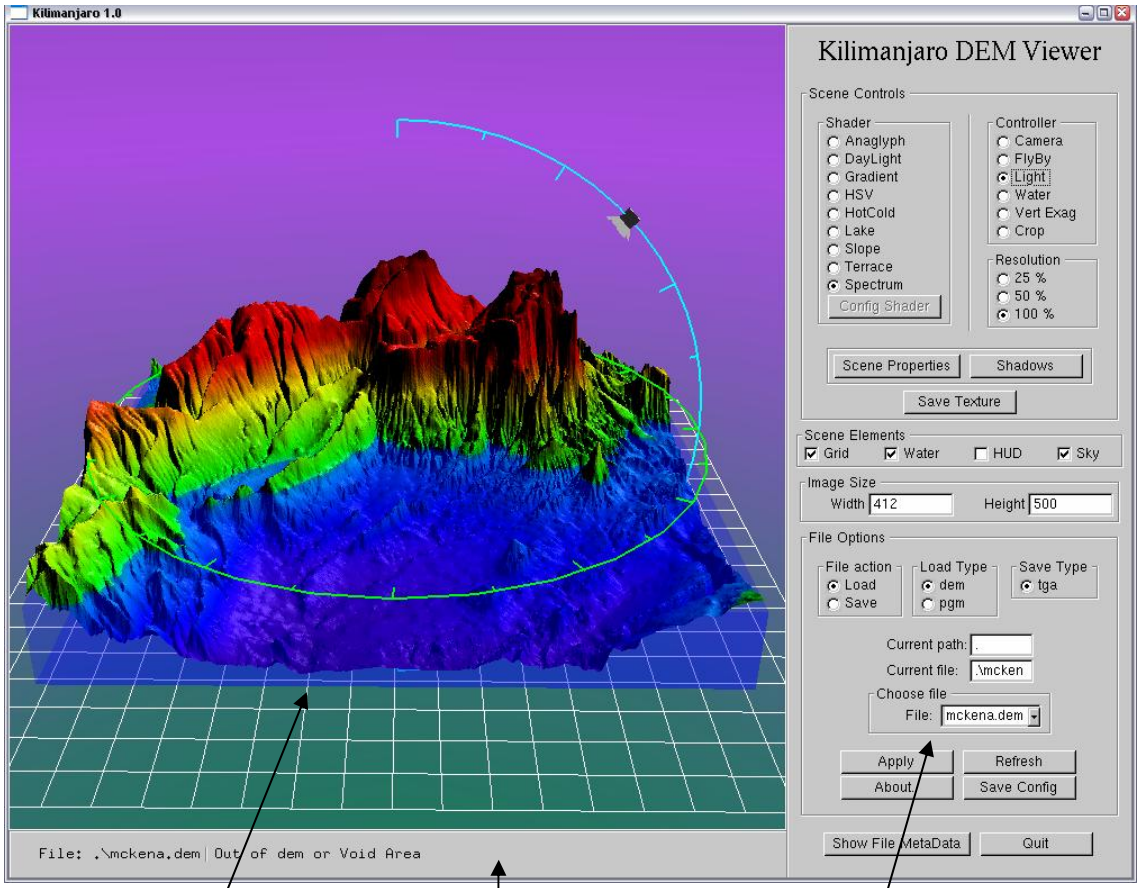
- Archivos binarios (para sistemas Windows) y este documento.
- Archivos binarios (para sistemas Windows), fuentes y documentación.
- Fuentes.

Para instalar KDV bastará con hacer doble clic sobre el instalador adecuado. Este instalador se encargará también de instalar, aparte del KDV, las librerías necesarias para su funcionamiento. Estas librerías son:

- msvcrit.dll
- msvcrt.dll
- glu32.dll
- glut32.dll
- xerces_2_2_0.dll
- opengl32.dll

## 4.3 Comenzando a utilizar KDV

Al lanzar la aplicación aparece la ventana principal donde se encuentra concentrada la mayor parte de la interacción con el usuario. Podemos distinguir tres grandes zonas:




Lienzo de Dibujo

Barra de información

Panel de opciones

- **Lienzo de dibujo**: es la zona de la ventana donde se dibujará el DEM y los elementos de escena seleccionados por el usuario. Todos los movimientos que se deseen realizar sobre el modelo de elevación se llevarán a cabo mediante pulsaciones de ratón sobre dicha ventana.

- **Barra de información**: ocupa la parte inferior de la ventana. En ella se nos informará de todas las acciones que se van realizando tales como cargar/salvar de/a fichero, las coordenadas del DEM en que se encuentra el puntero del ratón etc. La información que se ofrece al usuario cambiará según el controlador seleccionado.

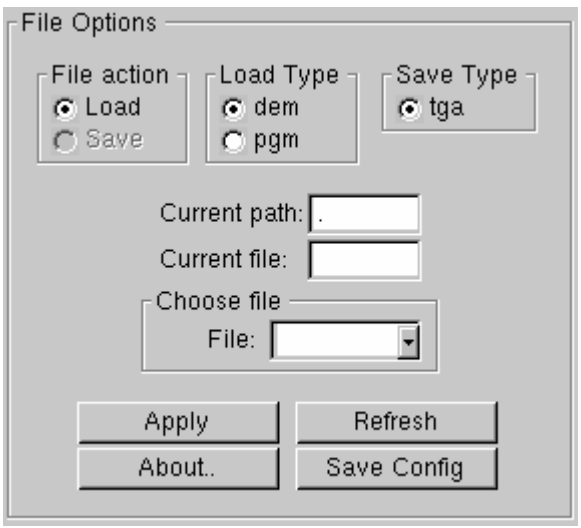
Cuando se produce alguna acción sobre el lienzo de dibujo (presionar alguno de los botones), se activa el controlador seleccionado, produciendo los consecuentes cambios en la imagen según sea su función. La información que se muestra en la barra de información refleja entonces el estado del controlador y/o los parámetros que modifica.

Si se realiza algún movimiento sobre el lienzo sin activar el controlador (ningún botón del ratón presionado), y siempre y cuando exista algún DEM en pantalla, muestra en cada momento las coordenadas relativas al DEM según se va moviendo el puntero por la superficie de éste.

- **Panel de opciones**: ocupa la parte derecha de la ventana. En este panel se encuentran todos los botones que permitirán configurar las distintas opciones del programa tales como elegir el controlador, aplicar una u otra sombra, cargar un fichero etc. Estas opciones se detallarán más adelante.

### 4.3.1 Cargar un DEM

Lo primero que habremos de hacer al lanzar la aplicación será leer el fichero que contiene el modelo a visualizar. Tras comprobar que la opción seleccionada en *File Action* es la adecuada, hemos de indicar el fichero a cargar. Para ello tenemos dos opciones:




- Indicar la ruta completa en la caja de texto *Current File*

- Utilizar la opción *Refresh*. Esta opción genera un listado de los ficheros DEM, con la extensión seleccionada en el cuadro Load Type, que se encuentran en el directorio especificado en el campo *Current Path*. Por tanto, si pulsamos *Refresh* y seguidamente desplegamos la lista *Choose File*, podremos elegir el fichero a cargar.

Tras especificar el fichero a cargar, pulsaremos *Apply* y podremos observar la evolución del proceso de carga en la barra de información.

Una vez cargado, podemos ajustar la resolución (25%-50%-100%) o seleccionar un controlador, o aplicar una determinada tonalidad.

### 4.3.2 Mostrar metainformación del DEM

Se puede visualizar la metainformación del DEM cargado mediante el botón *Show Metadata.* Dicha opción muestra una ventana que contiene toda la información del archivo. Para cerrar dicha ventana, basta con hacer clic sobre ella.

### 4.3.3 Elementos de la escena


Una vez cargado el fichero, en el lienzo de dibujo aparece su contenido. Además del DEM, en la escena podemos encontrar otros elementos. Podemos seleccionar cuáles de ellos deseamos visualizar en el cuadro *Scene Elements.* Todos los elementos de escena pueden ser configurados mediante la ventana *Scene Properties* detallada más adelante.

#### *4.3.3..1 Grid*

El Grid es la rejilla que aparece situada debajo del DEM. Permite observar las dimensiones relativas de cada porción del gráfico.

#### *4.3.3..2 Water*

En la escena el nivel de agua está asociado a una determinada altura. Si elegimos ver el agua, todas las regiones del DEM situadas por debajo de esta altura aparecerán cubiertas por agua.

#### *4.3.3..3 HUD*

Si seleccionamos este elemento, aparecerá en la pantalla sobre el DEM toda la información disponible acerca de altura, posición, etc. Dicha información se irá actualizando a medida que desplacemos el DEM.

Este controlador está inspirado en el sistema de visualización de la información de vuelo empleado en los aviones. Los aviones utilizan el HUD (Heads Up Display), un monitor que proyecta una imagen en una pantalla transparente en el frente de la cabina de mando. Con este display, el piloto

puede controlar los datos del vuelo y la información del radar sin apartar la vista del cielo.

#### *4.3.3..4 Sky*

Es el cielo situado detrás del gráfico. Se compone de tres partes configurables por separado: Nadir, Horizonte y Cénit.

### 4.3.4 Resolución

Hay que distinguir dos tipos de resolución:

- **Resolución interactiva (dinámica)**

  Tras cargar un archivo de elevación, se parte de un mínimo de resolución, que se va ajustando dinámicamente según se renderiza la imagen, esto es, cada vez que hay un movimiento con el ratón que modifique las características de la escena: cambio en la iluminación, ubicación de la cámara, parámetros de los shaders, exageración vertical, nivel del agua, etc. Dicha resolución aumenta siempre que hay un movimiento hasta un determinado límite, fijado por un umbral que permita una tasa de marcos por segundo (framerate) aceptable.

- **Resolución estática**

  Utilizada siempre que se renderiza la escena (el dem concretamente) en modo estático, es decir, tras dejar de manipular las características de la escena: iluminación, ubicación de la cámara, etc. La escena se renderiza en modo estático una vez han transcurrido dos segundos desde la última modificación (último renderizado en modo interactivo). Se puede seleccionar entre 25%, 50% o 100% de la resolución del DEM.

### 4.3.5 Controladores de imagen

Ahora que ya está el fichero visible en la pantalla, podemos realizar distintos movimientos sobre él utilizando los controladores existentes. Podemos ver los distintos controladores en el grupo “Controller” que se encuentra dentro del Panel de opciones en la parte superior derecha.

Procederemos ahora a la explicación de cada uno de los controladores.

#### ~ Camera

Este controlador nos permite mover el punto desde el cual estamos observando el DEM. Si seleccionamos este controlador pulsando sobre él, podremos realizar dos tipos de movimiento.

Por un lado, si pulsamos el botón izquierdo sobre el DEM y arrastramos sin soltar, podremos girar el DEM en cualquier dirección tanto en vertical como en horizontal. De este modo podremos observar el gráfico desde cualquier ángulo.

Si por el contrario, mantenemos pulsado el botón derecho y desplazamos el ratón, podremos realizar un zoom sobre la imagen. Si desplazamos el ratón hacia arriba, nos acercaremos a la imagen. Si desplazamos el ratón hacia abajo obtendremos el efecto contrario.

Existen posiciones de cámara almacenadas. Para recuperarlas, basta con pulsar los números del 0 al 9. Si se desea almacenar la posición de cámara actual, se puede hacer pulsando la tecla ALT simultáneamente con el número de posición en que se quiere almacenar.

Si se encuentra seleccionado este control, en la barra de información aparece información acerca de la altura, azimut y el radio de la esfera en que está situada la cámara.

#### ~ FlyBy

Activando este controlador podremos desplazarnos sobre la imagen simulando “caminar” por encima de ella.

Existen igualmente dos tipos de movimientos. Si mantenemos pulsado el botón izquierdo y arrastramos, esto nos permitirá mover el enfoque de la cámara en cualquier dirección, tanto giros horizontales como verticales.

Con el botón derecho podremos trasladarnos sobre cualquier punto de la imagen moviendo la posición de la cámara tanto alante y atrás como a los

lados. Para mover la cámara verticalmente, hay que mantener el botón derecho pulsado a la vez que se presiona la tecla Mayúscula.

Este control no genera información relevante que mostrar en la barra de información.

## ~ Light

Utilizando este controlador, podemos fijar la posición del foco de luz sobre el DEM. Así, podremos obtener el sombreado que se obtendría en cualquier época del año sobre el terreno. Para poder mover el foco de luz, este ha de estar seleccionado como visible en *Scene Properties*, detalladas más adelante. Para mover el foco de luz basta pinchar con el botón izquierdo y desplazar el ratón hasta posicionarnos en el lugar deseado. Al soltar, se renderizará la imagen generando el nuevo sombreado.

Mientras este control esté seleccionado, en la barra de información estará actualizada en todo momento la altura y azimut de la posición actual del foco.

## ~ Water

Activando este control podremos seleccionar la altura por debajo de la cual queremos que se encuentre el agua en el DEM. Para ello, basta con mantener pulsado el botón izquierdo y desplazar el ratón hacia arriba para incrementar el nivel, o hacia abajo para obtener el efecto contrario.

Mientras el botón izquierdo del ratón se encuentra pulsado, aparece en la barra de información el nivel de agua actual.

## ~ Vertical Exaggeration

Utilizando este control, podemos aumentar la altura relativa de cualquier punto del DEM. De este modo conseguiremos realzar el relieve pero de manera que se conserven las alturas relativas. El nivel 0 es el punto de partida, la altura del DEM original. El rango en que nos podemos mover va de -10 a +10. Para modificar la exageración vertical, basta con pinchar y arrastrar sobre el DEM con el botón izquierdo del ratón.

Mientras arrastramos sobre el DEM, aparece el valor de la exageración vertical en la barra de información.

### ~ Crop

Este control permite seleccionar únicamente una porción del DEM ocultando el resto. Al activar este control, obtenemos una vista vertical del DEM completo. Sobre ella seleccionaremos la porción deseada, mediante el método usual de selección: pinchar y arrastrar con el botón izquierdo. Al soltar, la porción seleccionada será renderizada y aislada del resto del DEM.

Para volver al DEM original, bastará con activar otra vez el control Crop. Una vez en este, podemos seleccionar una nueva porción, o cambiar de control sin seleccionar nada para recuperar el DEM completo.


Mientras se realiza la selección, se puede observar en la barra de información las coordenadas de la esquina inferior derecha del rectángulo de selección. Una vez seleccionada la región y realizado el crop, se actualiza el tamaño de la porción de terreno visible en el panel *Image Size.*

## *4.4 Cálculo de sombras realistas*

Se pueden calcular sombras realistas en función de la posición del punto de luz. Esas sombras se producen al realizar la proyección de cada elevación sobre el que incide la luz sobre el plano XY (el suelo). De este modo, podemos ver cómo ciertas partes de la montaña quedan oscurecidas por las sombras producidas por picos que se encuentren en la trayectoria de la luz hasta ese punto.

La intensidad de la sombra se puede modificar cambiando el parámetro correspondiente a la luz ambiente en las propiedades de la escena. Cuanto menores sean los valores para el rojo, verde y azul, más oscura será la sombra, y menos cercana al color que correspondería para ese punto.

Para aplicar el sombreado realista a la imagen, basta con presionar el botón *Shadows*. El efecto de este botón es renderizar la imagen una única vez con la resolución estática indicada y con los colores correspondientes al sombreado. La imagen queda entonces oscurecida según la posición del punto de luz. En caso de hacer clic o modificar algún parámetro que suponga renderizar de nuevo la imagen, el efecto de sombras se pierde y aparece la imagen que quedaría sin sombras.

## *4.5 Guardar textura*

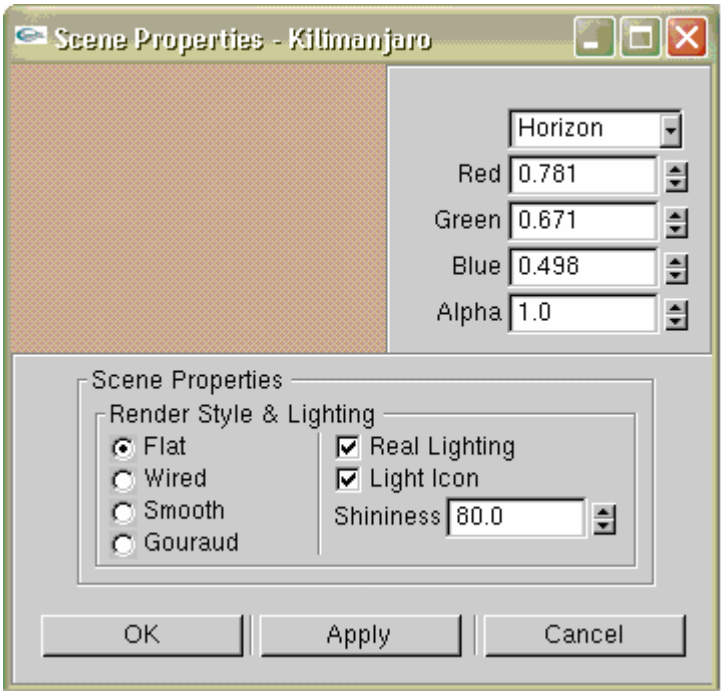
La acción de guardar la textura correspondiente al DEM que se está mostrando, con los colores producidos por el shader que se está aplicando en ese momento, así como efectos de luces, sombras y agua, se puede llevar a cabo mediante el botón *Save Texture.*

Al seleccionar dicha opción, la textura queda guardada en la misma ruta que el archivo correspondiente al DEM que se está mostrando. El nombre de archivo de la textura es el nombre del archivo original, seguido de "-texture". La textura se guarda en formato TGA.


El tamaño de la textura es el correspondiente al mostrado en las casillas Width (ancho) y Height (largo). Dicho tamaño se calcula en función de la resolución estática utilizada y la región visible del DEM para esa resolución.

# 4.6 *Configurar la aplicación*

## 4.6.1 Configurar los elementos de escena




Pulsando en el botón *Scene Properties* aparece esta ventana de configuración desde la cual podremos manipular los distintos elementos de escena. Arriba a la derecha tenemos una lista desplegable de los elementos de la escena cuyo color podemos configurar. Estos elementos son grid, nadir, horizonte, zenit, HUD, fondo, agua, luz ambiente, luz difusa y luz especular. Para estos elementos podemos elegir la cantidad de rojo, verde y azul así como el Alpha, que permite configurar el grado de transparencia. Todos estos valores varían en el rango [0.0..1.0]. Con un alpha de valor 1.0, la transparencia es nula y si su valor es 0.0 la transparencia es total.

Si la opción *Real Lighting* está seleccionada, se calcula el color para cada uno de los puntos del DEM en función de la incidencia de la luz sobre ese punto. De otro modo, se aplica el color sin más.

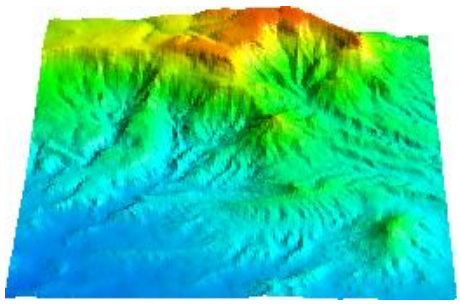
Si la opción *Light Icon* está seleccionada, aparece en la escena el icono que representa el punto de luz, el foco.

El valor correspondiente al campo *shininess* se utiliza a la hora de calcular la cantidad de color aportada por el reflejo de la luz, a valores más altos, más vivo es el color.

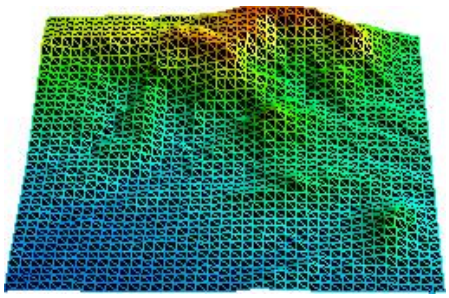
En la parte de debajo de la ventana de configuración podemos elegir el estilo de renderizado.

Éste puede ser:

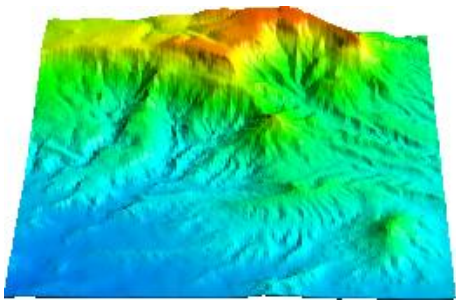
- **Flat** : selecciona el color almacenado de un solo vértice y se lo asigna a todos los fragmentos de píxel generados por rasterizar una primitiva simple



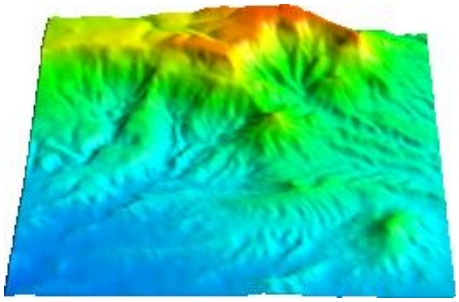
- **Wire**: muestra la malla de triángulos que forma el terreno sin aplicar el relleno.



- **Smooth** : interpola los colores almacenados de los vértices, asignando diferentes colores a cada fragmento de píxel resultante.



- **Gouraud**: calcula las normales en función del resto de normales que tiene alrededor. El resultado obtenido es una imagen de formas más suaves.

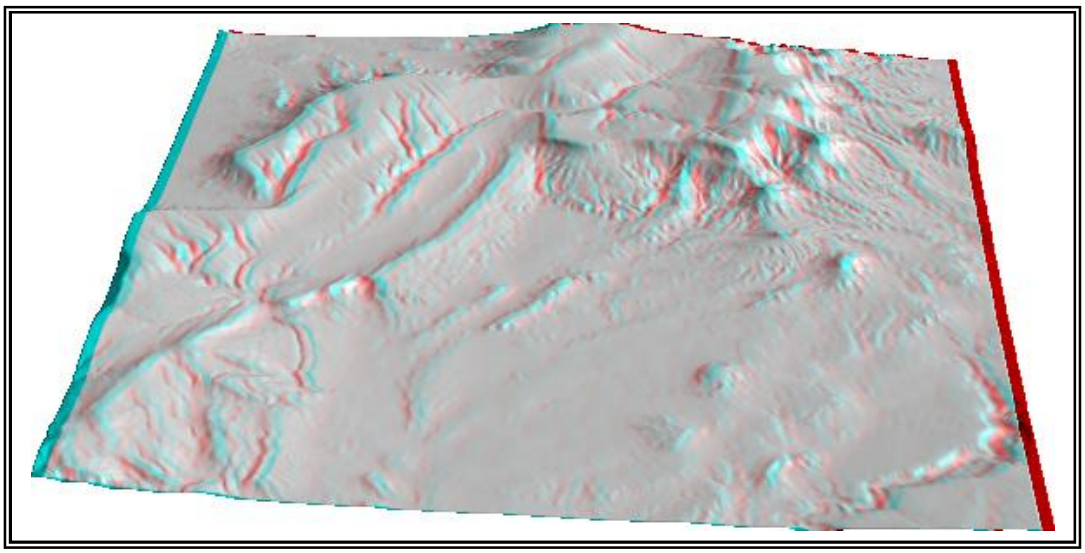


## 4.6.2 Configurar los Shaders

### *4.6.2..1 Anaglyph*

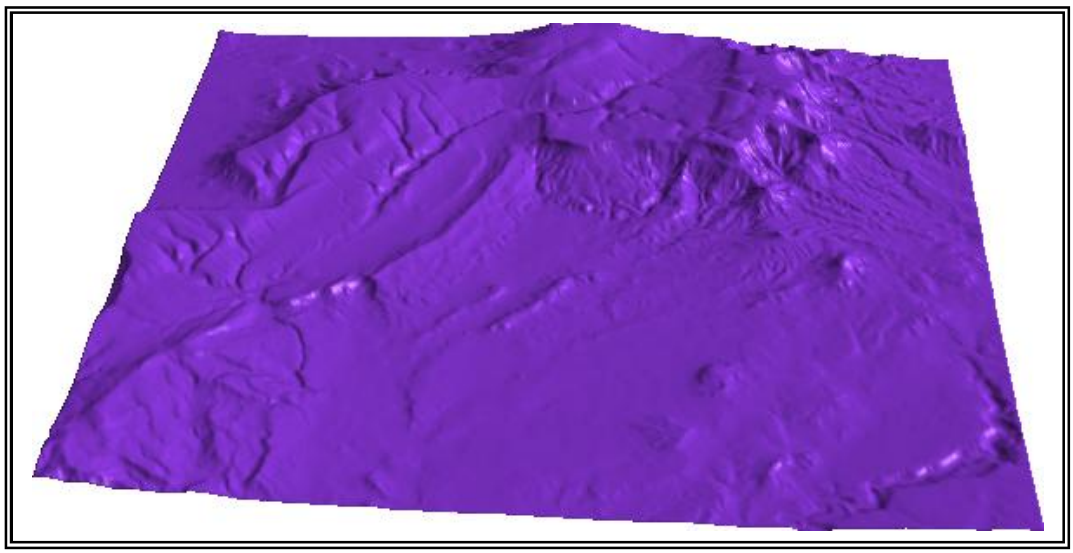
Este shader permite visualizar el DEM con gafas tridimensionales. Por ello dibuja el gráfico en 2 colores, rojo transparente y azul turquesa.


En su ventana de configuración podemos modificar el desplazamiento lateral entre las dos capas.



#### *4.6.2..2 DayLight*


Este shader utiliza un único color que varía en intensidad para dibujar el DEM. En su configuración podemos seleccionar el color de la superficie que se empleará.

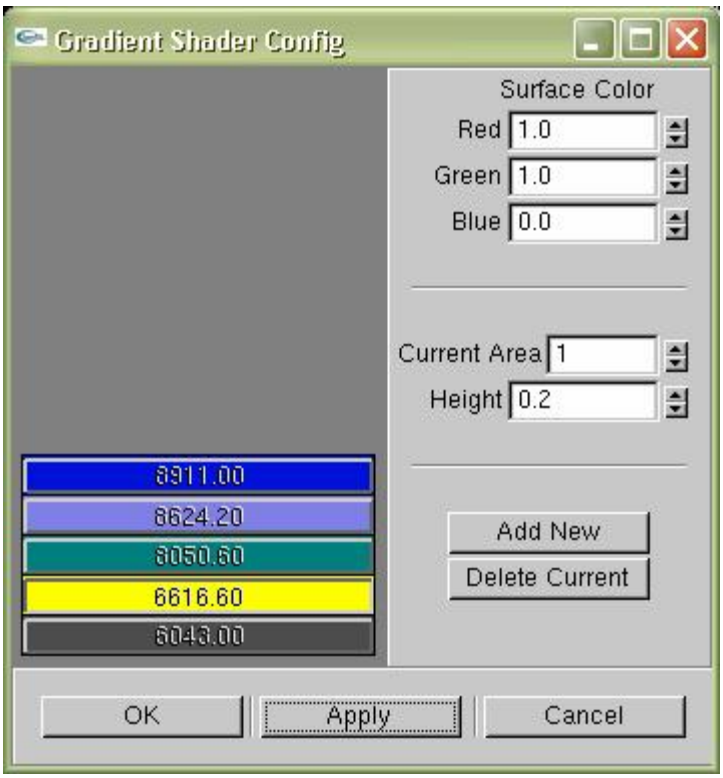


#### *4.6.2..3 Gradient*

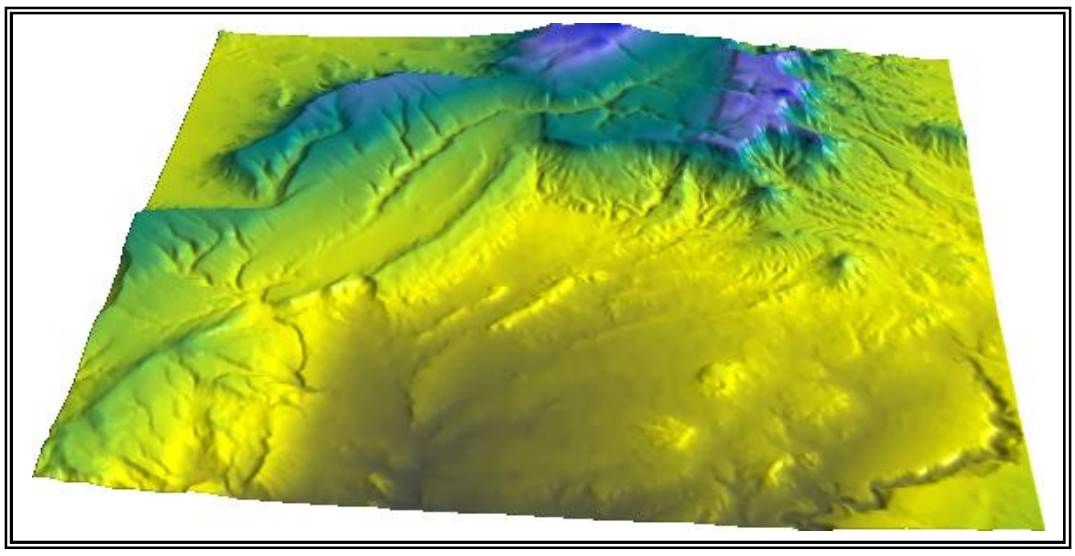
En este shader podemos definir un color para una altura determinada. El color para alturas intermedias se determina realizando una interpolación entre los 2 colores más próximos definidos.

Su ventana de configuración permite definir para todas las alturas que deseemos un color específico. Solo hemos de indicar la altura relativa deseada utilizando el spinner *Height* y a continuación especificar el color en función de sus 3 valores RGB, viendo los cambios reflejados inmediatamente en el botón.

En dicho botón aparece la altura absoluta calculada a partir de la altura del DEM y la altura relativa para ese color. A medida que modificamos la altura de un color, éste se recoloca automáticamente en función del resto de colores definidos. Para seleccionar un color, basta con pinchar sobre el botón que lo representa o utilizar el spinner Current Area.

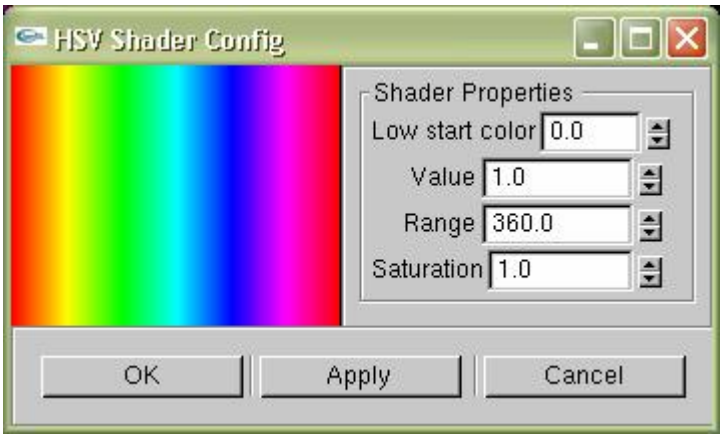



Para eliminar una altura basta con seleccionar el área pulsando sobre ella y presionar seguidamente el botón *Delete Current.* El mínimo de colores que esta ventana permite es uno.



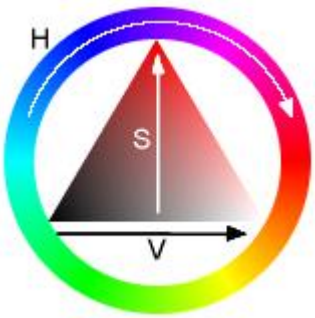
#### *4.6.2..4 HSV*

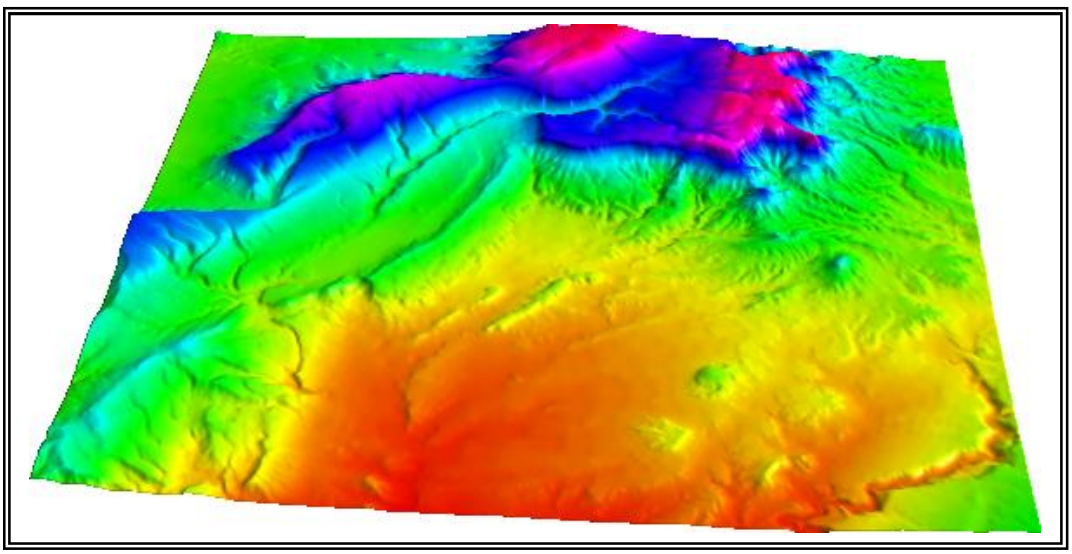
HSV quiere decir Hue Saturation Value, que son los parámetros configurables. Este shader es muy colorido y resulta útil para aportar detalles en DEMs que tienen poca variación de la elevación, es decir, la mínima y máxima elevación están próximas.




Los parámetros que podemos variar en este shader son:

- Low start color: establece dónde estará la menor elevación en el rango de colores de HSV
- Range (Hue): establece qué porción del rango HSV debe ser usada. Incrementando este parámetro, conduce a un color envolvente
- Saturation: permite especificar la intensidad del color, (qué cantidad de gris está presente). Su medida se expresa con el porcentaje de amplitud de la longitud de onda.
- Value: el brillo del color. Está expresado en función del porcentaje de propagación de la longitud de onda

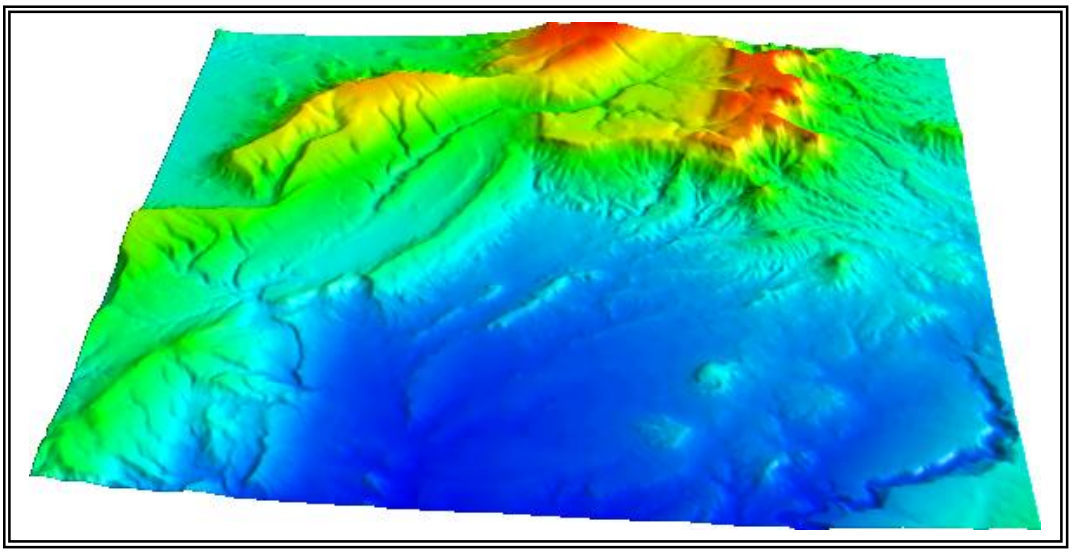


### *4.6.2..5 HotCold*

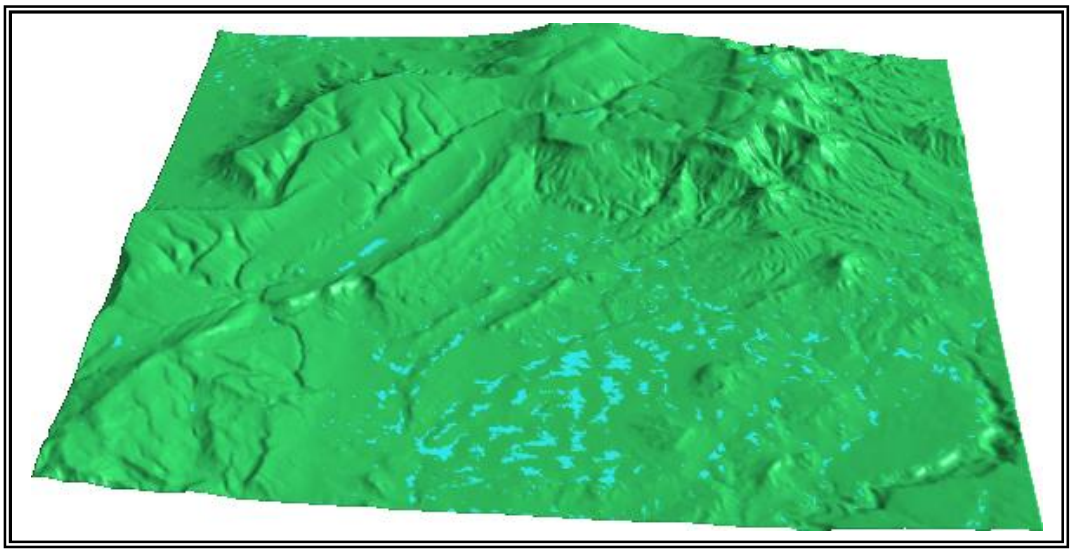
Este shader distingue divide el terreno en cuatro rangos de alturas. Asigna colores cálidos a las zonas altas y colores fríos a las bajas. La asignación de colores es:

- Para el cuarto superior utiliza :
  - color rojo: 1
  - color azul : 0
  - color verde: ponderado con la altura
- Para el tercer cuarto utiliza :
  - color rojo: ponderado con la altura
  - color azul : 0
  - color verde: 1
- Para el segundo cuarto utiliza :
  - color rojo: 0
  - color azul : ponderado con la altura
  - color verde: 1
- Para el cuarto inferior utiliza :
  - color rojo: 0
  - color azul : 1
  - color verde: ponderado con la altura



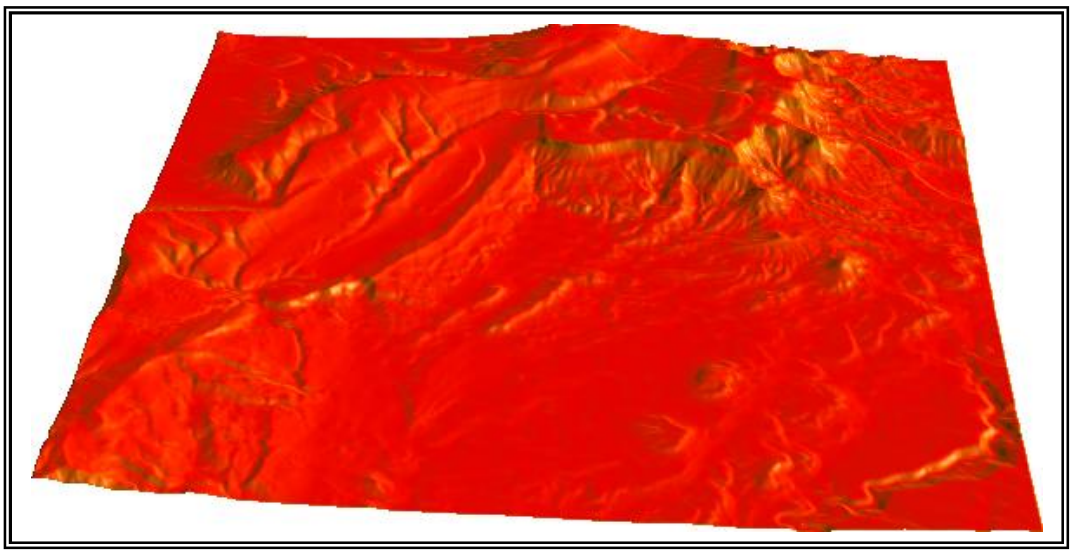
### *4.6.2..6 Lake*

Este shader aplica un único color para las zonas con pendiente (Land color) y otro para las zonas planas (Lake color). En su ventana de configuración podemos elegir qué dos colores usar.

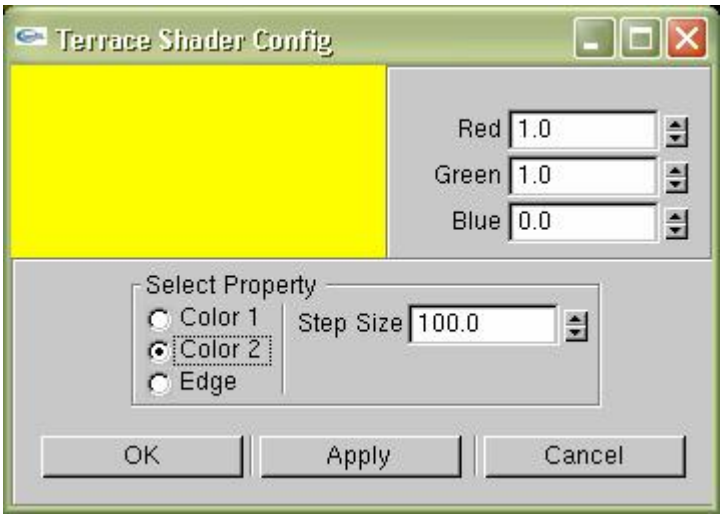
### *4.6.2..7 Slope*


Este shader aplica un color para las zonas planas y otro para las zonas con pendiente. Aplica a las zonas planas el color “Flat” y para las zonas con pendiente obtiene los porcentajes de color “Steep” y “Flat” a aplicar. Cuanto más vertical es una superficie, más porcentaje de “Steep” compone su color. La ventana de configuración permite seleccionar qué dos colores deseamos que el shader utilice.

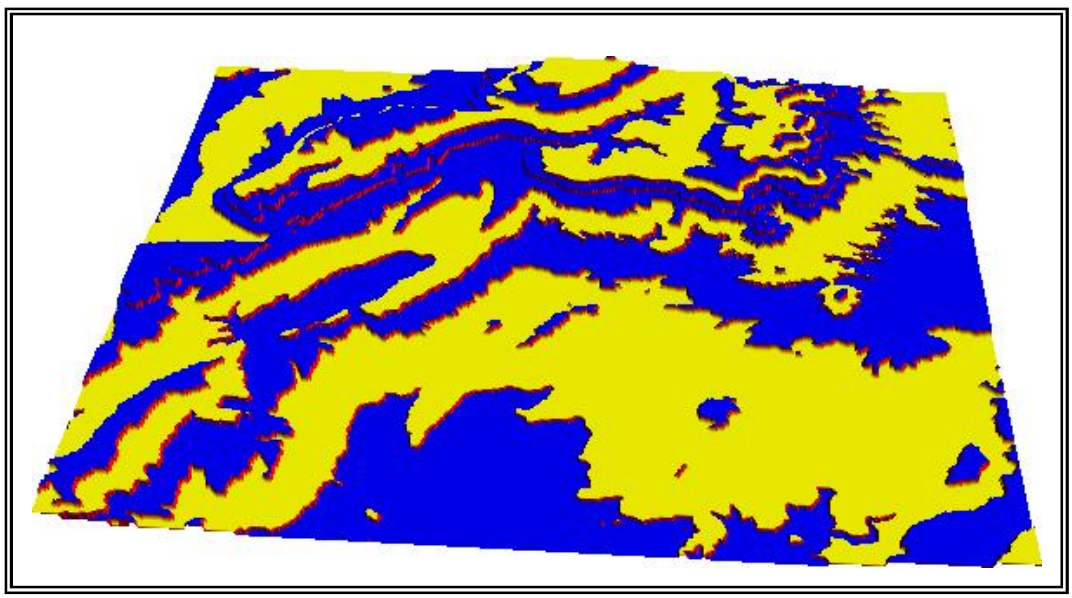


### *4.6.2..8 Terrace*

Este shader separa el Dem en distintas alturas y a cada una le asigna un color. Simula los mapas topográficos, que son una representación del relieve de la tierra. Tienen la cualidad de que por medio de las curvas de nivel se puede interpretar la forma de la superficie de la tierra, sabiendo si hay montañas, valles, ríos, riscos y demás cualidades del terreno que no se encuentran en un mapa normal. Así pues, en este caso, nos permiten conocer las alturas de todas las partes del DEM.

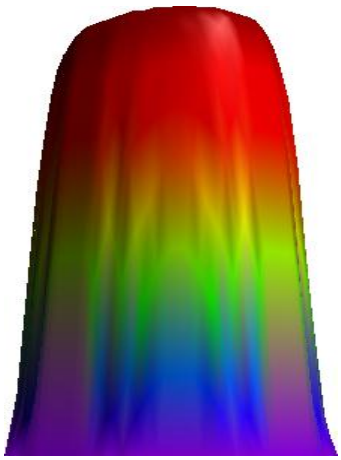



La ventana de configuración permite seleccionar los colores alternos de las capas, la línea de separación y la distancia entre capas.



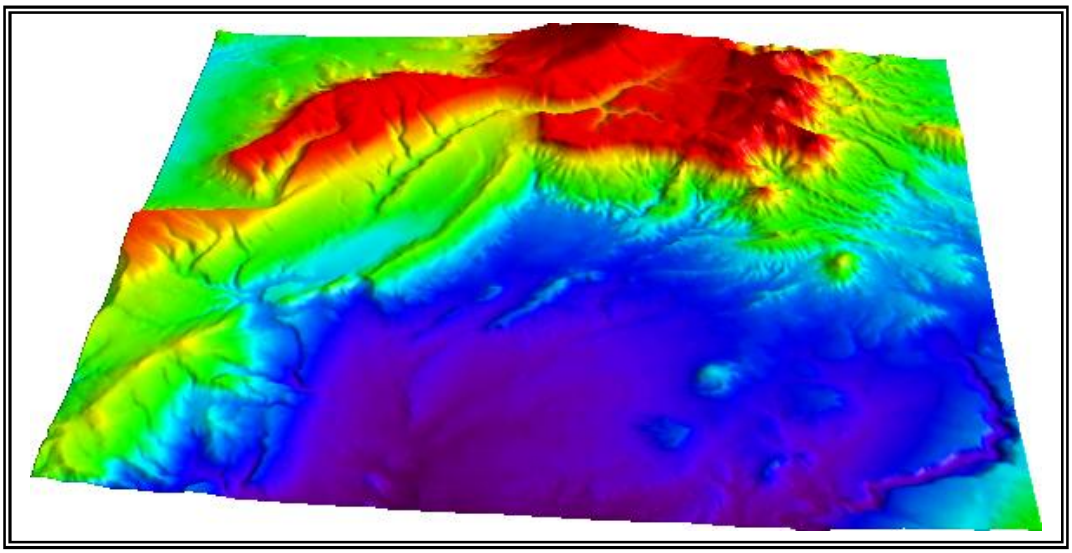
No es recomendable aplicar sombras a los escenarios generados con este shader pues se modifican las alturas relativas de los puntos para encuadrarlos en la zona correspondiente, y por tanto las sombras no se corresponderían con las de la imagen inicial.

#### *4.6.2..9 Spectrum*



Este shader aplica un determinado color del espectro a cada elevación del terreno. Para las zonas con menor elevación elige el azul y para aquellas zonas más elevadas, el rojo. El color se calcula según la longitud de onda y ésta está determinada por la altura del punto. La longitud de onda visible al ojo humano va desde 380 a 780 nanometros.

Dada su implementación determinista, este shader no es configurable.



# *4.7 Guardar a archivo*

El programa permite grabar en fichero tanto la imagen como la configuración actual del programa.

## 4.7.1 Guardar la configuración

KDV permite guardar la configuración actual del programa, esto es, la configuración de la escena y de cada uno de los shaders. Pulsando el botón *Save Config* se generará un archivo XML en el directorio de la aplicación que será cargado la próxima vez que ejecutemos KDV.

El nombre del archivo xml que controla la configuración es “app-config.xml”. Dicho archivo se encuentra en la misma ruta que el ejecutable de la aplicación. En caso de no encontrar este archivo a la hora de abrir la aplicación, ésta se inicia con valores por defecto para cada uno de los parámetros.

## 4.7.2 Guardar captura de pantalla

La imagen mostrada en el lienzo de dibujo se puede guardar en cualquier momento en un archivo con el formato de grabación seleccionado. El formato de salida disponible es TGA. Para salvar una captura, basta con seleccionar la opción *Save* en el panel *File Options*, y automáticamente cambiará el nombre del path, modificando la extensión para adaptarla al formato seleccionado. Si no se desea este nombre por defecto, se puede cambiar en el campo *Current Path.* De cualquier modo, una vez se seleccione la opción *Apply*, el archivo quedará guardado en la ruta indicada por el campo *Current Path.*

El ancho y alto de la imagen vienen determinados por el tamaño actual del lienzo de dibujo.

# *4.8 Otros*

Existen dos modos de visualización de la pantalla de la aplicación principal:

- **Modo Ventana:** La aplicación se muestra como una ventana. Para activar este modo desde el modo “Pantalla Completa”, hay que asegurarse de tener seleccionada la ventana de la aplicación (aún en formato Pantalla Completa), lo cual puede hacerse si se pincha sobre el lienzo de dibujo, y presionar la tecla ‘w’.

- **Modo Pantalla Completa:** Se puede pasar al modo pantalla completa para que la aplicación ocupe la totalidad de la ventana. Esto se puede hacer presionando la tecla ‘f’, siempre y cuando la ventana esté correctamente seleccionada mediante el procedimiento explicado en el punto anterior.

## 4.8.1 Acerca de

Si se selecciona la opción About, aparece una ventana con información sobre los autores de la aplicación.

## 4.8.2 Consideraciones finales

Se recomienda no utilizar el botón “X” de cualquier ventana de la aplicación, pues dadas las características de la librería utilizada para implementar la aplicación, si se cierra cualquier ventana mediante este procedimiento, puede ocasionar fallos en el resto de la aplicación o incluso cerrarla por completo. Utilizar por tanto los botones correspondientes o el botón *Quit* para salir de la aplicación.

# 5 Bibliografía

## 5.1 Libros


- 🕮 [L01] “Design Patterns” - *Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides*. Ed. Addison-Wesley
- 🕮 OpenGL programming guide : the official guide to learning OpenGL version 1.1 / Mason Woo, Jackie Neider, Tom Davis
- 🕮 OpenGL reference manual : the official reference document to OpenGL, version 1.2 / OpenGL Architecture Review Board ; editor, Dave Shreiner.


## 5.2 URLs


- http://edcwww.cr.usgs.gov/guides/dem.html Formato DEM
- http://www.lighthouse3d.com/opengl/terrain/index.php3?tgalib TGA
- http://netpbm.sourceforge.net/doc/pgm.html PGM
- http://msdn.microsoft.com/visualc/ Visual C++
- http://xml.apache.org/xerces-c/index.html Xerces
- http://fly.cc.fer.hr/~unreal/theredbook/ El libro rojo de OpenGL
- http://www.eecs.tulane.edu/www/graphics/doc/OpenGL-Man-Pages/opengl_index_spec.html Rutinas OpenGL
- http://www.sgi.com/Technology/openGL/ The OpenGL WWW Center
- http://www.reliefshading.com/analytical/index.html
- http://www.gamedev.net/reference/articles/article1817.asp
- http://www.gamedev.net/reference/articles/article678.asp
- http://www.gamedev.net/reference/articles/article1436.asp
- http://www.gamedev.net/reference/articles/article1936.asp
- http://www.opensg.org/index.EN.html Scenegraph
- http://sgl.sourceforge.net Scenegraph
- http://thorkildsen.no/faqsys/cates/math.html Algoritmos Varios
- http://easyweb.easynet.co.uk/~mrmeanie/rt/rt.htm Raytracing
- http://www.acm.org/jgt/papers/MollerTrumbore97/code.html Raytracing
- http://nervus.go.ro/common/fctsm.htm Sombras
- http://www.lighthouse3d.com/opengl/terrain/index.php3?introduction Terrenos 3D
- http://www.vterrain.org/Performance/lighting.html
- http://www.cs.unc.edu/~andrewz/comp236/hw5b OpenGL Lighting

## *5.3 Palabras clave*

Las siguientes palabras podrían ser utilizadas como palabras clave para realizar una búsqueda bibliográfica:

- OpenGL
- Terrenos
- Elevación
- GLUT
- GLUI
- Sombras
- Kilimanjaro
- DEM
- TGA

# 6 Autorización de difusión

Los abajo firmantes Mario Carballo Santos, Manuel Durán González, José Antonio Núñez Mendoza y Carmen Pardo Alhambra autores de la aplicación “Kilimanjaro Dem Viewer”, autorizan a la Universidad Complutense de Madrid a difundir y utilizar con fines exclusivamente académicos, nunca comerciales y mencionando expresamente a sus autores, tanto esta memoria, como el código, la documentación y el prototipo desarrollado bajo la asignatura Sistemas Informáticos.

Madrid a domingo, 06 de julio de 2003