CEIG'09, San Sebastián, Sept. 9-11 (2009)
C. Andújar and J. LLuch (Editors)

Edición y visualización de información vectorial en aplicaciones SIG

Jordi Torres, Jesús Zarzoso, María Ten, Rafael Gaitán, Javier Lluch

Abstract

Con la llegada de los Sistemas de Información Geográfica (SIG) en 3D, han aparecido nuevas formas de representar datos vectoriales mucho más intuitivas y realistas para los usuarios. En este artículo se presenta una librería multiplataforma y de código libre, que permite visualizar estos datos mediante símbolos tridimensionales como podrían ser árboles o postes eléctricos. Además, se presenta un sistema de edición interactiva mediante una librería de manipuladores para la edición de datos vectoriales y símbolos que permite ser fácilmente integrada en aplicaciones SIG.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.6]: Methodology and Techniques–Graphics data structures and data types—Computer Graphics [I.3.8]: Applications—Information Systems [H.0]: General—

1. Introducción

En la cartografía tradicional, se consigue representar el mundo real mediante una proyección vertical sobre un plano bidimensional, simplificando las diferentes entidades para formar lo que conocemos como mapa. Con la aparición de los Sistemas de Información Geográfica (SIG), estas proyecciones del mundo real se almacenan como imágenes *raster* o datos vectoriales en grandes bases de datos georreferenciadas, para posteriormente, ser consultadas, editadas y analizadas. Los datos vectoriales presentan la ventaja, frente a las imágenes *raster*, de aportar gran calidad gráfica en cualquier resolución. Además, la topología de estos datos permite utilizar ciertas herramientas de análisis muy costosas para datos *raster* o directamente inaplicables.

Un dato vectorial se define, en el ámbito de los SIG, como una tupla de una base de datos espacial, compuesta por una *característica geográfica* (tradicionalmente un punto, una línea o un polígono) a la que llamaremos *entidad* y además, una serie de atributos numéricos asociados a esta, como pueden ser índices de población, tráfico medio de una carretera, probabilidad de incendio de una zona, etc. A la hora de visualizar estos datos, muchos clientes SIG optan por una representación bidimensional muy similar a la cartografía tradicional, donde factores como el color o el grosor del contorno de las entidades pueden representar los atributos numéricos asociados. Con la aparición de nuevos clientes y herramientas 3D para SIG, podemos analizar estos datos desde una perspectiva diferente.

En una vista tridimensional no es necesario proyectar entidades del mundo real como edificios o árboles, sino que podemos representarlos directamente con modelos sintéticos sobre el terreno. De esta forma, no sólo tenemos una representación visual de la latitud y longitud de una entidad, sino que además podemos conocer su altitud. Al mismo tiempo, se simplifica la lectura de los mapas, haciendo a veces innecesario el uso de complejas leyendas. Sin embargo, en la actualidad no existe ningún estándar para trabajar con volúmenes sobre datos vectoriales, sino que los objetos tridimensionales se representan mediante entidades 2D con un atributo de altura o en su defecto como conjuntos de polígonos. Algunos clientes SIG implementan su propia topología para objetos 3D, el problema es que no es compatible con otras aplicaciones.

En este artículo vamos a presentar una librería multiplataforma y de código abierto para la visualización 3D de datos vectoriales que permite ser integrada en cualquier cliente SIG. Esta librería, basada en el grafo de escena de OpenSceneGraph, incorpora un sistema de tratamiento de simbología que permite, no sólo modificar los atributos propios de las entidades como pueden ser el color o el grosor del contorno, sino que además permite sustituir estas características por modelos sintéticos en tiempo de visualización, consiguiendo

DOI: 10.2312/LocalChapterEvents/CEIG/CEIG09/115-124

representaciones más intuitivas y realistas que con los métodos tradicionales.

Del mismo modo que un cliente SIG debe aportar los elementos necesarios para la visualización de datos vectoriales, es necesario que incorpore mecanismos para la edición de los mismos. Muchos clientes simplemente permiten su edición mediante aplicaciones de CAD externas o modificando directamente la base de datos. Aunque estos mecanismos pueden ser suficiente para aplicaciones 2D, complican bastante la edición si añadimos una tercera dimensión, pues muchas veces es necesario conocer datos reales del terreno, como la elevación, para posicionar correctamente ciertas entidades del mundo real. Para facilitar la edición de estos datos, se han desarrollado una serie de manipuladores que permiten la edición interactiva tanto de las características vectoriales como de los símbolos que las representan en la propia vista del cliente SIG. De esta forma, se pueden modificar los datos con herramientas muy similares a las presentes en las aplicaciones CAD y sin tener la necesidad de cambiar de aplicación o exportar los datos.

A continuación, se describirán las principales técnicas para *renderizar* datos vectoriales en el ámbito SIG, así como las diferencias que existen entre las entidades y el símbolo que se utiliza para representarlas. Se hablará sobre la importancia de los símbolos en clientes 3D y las posibilidades que ofrece el conjunto de herramientas desarrolladas. Posteriormente, se tratará sobre la edición interactiva, tanto de datos como de símbolos, mediante manipuladores desarrollados específicamente para este tipo de aplicación. Después, se explicará como se han integrado este conjunto de herramientas en un cliente SIG real y se comentarán los resultados obtenidos. Finalmente, se extraerán las conclusiones del trabajo desarrollado y se definirán las líneas de trabajo futuro.

2. Antecedentes

Los datos vectoriales son una parte fundamental de cualquier SIG debido a la cantidad de información que se puede obtener mediante su análisis. El formato de estos datos y su distribución viene marcado por los estándares del Open Geospatial Consortium (OGC) [Ogc94], como por ejemplo, Web Feature Service (WFS) [Ogc04] que define la interfaz necesaria para consultar y modificar bases de datos vectoriales mediante servicios web. A pesar de que este estándar permite almacenar la altitud, además de la longitud y la latitud, no permite definir verdaderas relaciones topológicas tridimensionales, pues fue concebido principalmente para ser utilizado bajo clientes web 2D.

Con la popularización de nuevas aplicaciones SIG en 3D, se ha intentado ganar realismo a la hora de visualizar los datos vectoriales. Estándares como CityGML [Ogc08] han surgido recientemente, por la necesidad de representar ciudades y paisajes urbanos fieles a la realidad. Sin embargo, hoy en día no se ha logrado crear un estándar para servicios web como WFS que soporte entidades tridimensionales y las operaciones necesarias para su análisis, quedando los esfuerzos del OGC en un borrador disponible en [Ogc05]. Muchos desarrolladores han optado finalmente por crear su propio formato de datos.

Existen dos métodos muy extendidos para renderizar datos vectoriales en aplicaciones SIG 3D. El primero, consiste en rasterizar previamente las entidades y trabajar con la imagen generada como si se tratará de datos *raster*, es decir, aplicándolos como una textura sobre el terreno. Este método tiene las mismas desventajas que los datos *raster*, pero tiene la ventaja de que los datos siempre quedan pegados sobre la superficie del terreno, evitando que queden por encima o por debajo del terreno. Algunas implementaciones permiten rasterizar los datos bajo demanda, creando modelos simples de multiresolución como en [OK02]. Otra implementación utilizando *shaders* está descrita en [BN08].

El segundo método consiste en renderizar directamente las entidades como primitivas geométricas. Esta estrategia tiene la ventaja de facilitar los métodos de edición interactiva, así como el uso de algunas herramientas de análisis. Sin embargo, aplicar un modelo de multirresolución a estas geometrías o lograr que se sitúen sobre la superficie de un modelo de terreno con distintos niveles de detalle, son tareas mucho más complicadas. Algunas implementaciones que se centran en solucionar estos dos problemas se describen en [ASZ07, WKW*03, AAJ06, SGK05, MS07].

Es importante aclarar que en el ámbito SIG, los datos vectoriales no sólo definen geometrías, sino como se ha explicado previamente, cada dato define una entidad y una serie de atributos numéricos o textos asociados. Las entidades suelen relacionarse con geometrías como puntos, líneas o polígonos, pero en realidad unicamente definen una o varias posiciones georreferenciadas y su relación topológica. También hay que diferenciar entre la geometría o característica que se almacena en la base de datos y el símbolo que se utiliza para representar esta característica en tiempo de visualización. Para entender mejor este concepto se puede recurrir a los mapas tradicionales. Imaginemos por ejemplo, que tenemos un mapa de nuestra ciudad y se quiere indicar en él las coordenadas de todas las farmacias que existen. Las coordenadas serían las entidades, que se pueden representar mediante puntos, sin embargo, es más útil sustituir los puntos en el mapa por cruces verdes de manera que se distingan facilmente de otras entidades. Estas cruces verdes se denominan *símbolos* y se pueden sustituir por otro objeto como una A en rojo que es el símbolo de farmacia en Alemania, sin afectar a los datos vectoriales.

El tratamiento de la simbología es un elemento común entre los clientes SIG 2D. Normalmente se representan los datos vectoriales por diferentes símbolos que ayudan al usuario a categorizar estas características vectoriales. Además se permite elaborar leyendas donde el color o el patrón de la geometría puede indicar el valor de alguno de los atributos asociados. Muchos desarrolladores se han limitado a expor-

tar esta funcionalidad a sus aplicaciones 3D, sin explotar las posibilidades que ofrece este nuevo espacio de trabajo, aportando como mucho algún mecanismo de extrusión de geometría. Por ejemplo, supongamos que se dispone de una serie de datos sobre los bosques de una región. Se pueden representar los polígonos que se definen en la información vectorial, como conjuntos de modelos sintéticos de árboles, variando su densidad en función de un atributo que indique la densidad de árboles real y su color en función de otro atributo que indica el peligro de incendio. En un espacio bidimensional no sería posible albergar tantos datos, es por ello que surge la necesidad de desarrollar una librería que permita el uso de simbología tridimensional para clientes SIG, de forma que los mapas creados tengan mayor realismo y sean capaces de proporcionar al usuario muchos más datos mediante nuevas herramientas de análisis.

La mayoría de los clientes SIG 3D proveen de los mecanismos necesarios para modificar los símbolos durante la visualización como en ArcGIS 3D Analyst [Esr99], pero no aportan ningún mecanismo para editar las entidades de forma interactiva desde la propia vista 3D. Para modificar estos datos, es necesario modificar directamente la base de datos, abrir una vista 2D o utilizar una herramienta CAD externa con la incomodidad que esto representa para el usuario final. Por tanto, es necesario aportar manipuladores de geometría intuitivos que permitan modificar los datos y ver los resultados sin necesidad de tener que cambiar de espacio de trabajo.

Dada la enorme cantidad de polígonos presentes en una vista 3D de una aplicación SIG, se ha optado por aprovechar los beneficios que aportan los grafos de escena, como OpenSceneGraph (OSG) [OB99] que incorpora mecanismos para facilitar el uso de técnicas de aceleración gráfica tales como: *LOD*, *view-frustrum culling*, *back-face culling* o *small feature culling*.

OpenSceneGraph dispone de un componente que facilita la edición interactiva de las geometrías presentes en el grafo de escena, llamada **osgManipulator**. En [ZTT*08] presentamos una primera versión de un sistema de edición que hacía uso de esta librería, llamada **osgVP-Manipulator**. Mediante este sistema, se facilita la creación y uso de los manipuladores disponibles en **osgManipulator**.

Finalmente, a la hora de integrar y probar estas herramientas en un cliente SIG, se ha optado por gvSIG [Cit03] dado que se trata de un cliente multiplataforma y de código abierto, desarrollado por la *Consellería de Insfraestructuras y Transporte* de la *Comunidad Valenciana*. Este cliente soporta la mayoría de los estándares del OGC y otros formatos de uso habitual. Recientemente se ha incorporado una extensión para añadir una vista 3D donde serán integradas las herramientas aquí descritas.

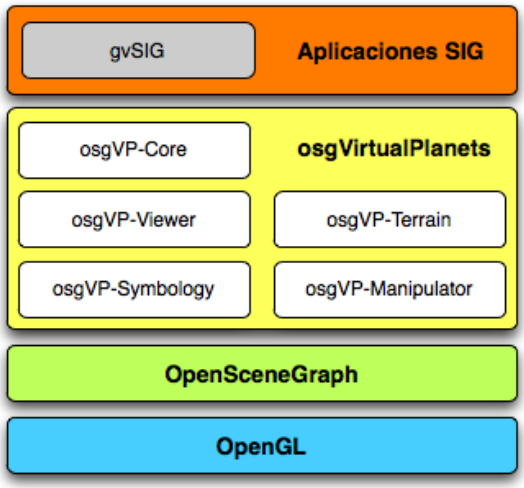


Figura 1: *Arquitectura general de la librería osgVirtualPlanets*

3. Desarrollo

En esta sección se describirá la arquitectura y desarrollo de la librería implementada, destacando las características más relevantes de cada uno de sus componentes. Finalmente, presentaremos un caso práctico de uso integrándola en el cliente gvSIG.

Con la finalidad de permitir incorporar una vista tridimensional a los clientes SIG, se ha desarrollado una librería que se denomina **osgVirtualPlanets** (osgVP) basada en el grafo de escena de OSG e implementada en C++. Esta librería incorpora una *API* en Java utilizando la tecnología *Java Native Interface* (JNI) que permite ser integrada con aplicaciones desarrolladas en este lenguaje. En la figura 1, podemos ver los diferentes componentes que se incluyen en esta librería:

- **osgVP-Viewer** permite añadir un contexto 3D a cualquier aplicación para renderizar escenas creadas mediante la librería o utilizando los nodos de OSG.
- **osgVP-Terrain** genera y maneja modelos de terreno a partir de datos de elevación e imágenes *raster* en formato SIG.
- **osgVP-Symbology** ofrece los mecanismos necesarios para visualizar datos vectoriales y símbolos asociados, así como para cambiar sus propiedades.
- **osgVP-Manipulator** permite manipular objetos de la escena como por ejemplo, datos vectoriales.
- **osgVP-Core** es el componente principal de la librería.

En las siguientes secciones, nos centraremos en los componentes **osgVP-Symbology** y **osgVP-Manipulator** que son los encargados de la visualización y edición de datos vectoriales.

3.1. Simbología

El objetivo principal del componente **osgVP-Symbology** es ofrecer una *API* a los desarrolladores que de soporte para la visualización de datos vectoriales con garantía de interactividad. Una buena estrategia a la hora de definir la simbología es un factor clave para aprovechar toda la potencia del grafo de escena utilizado en la visualización.

Tras el estudio de los requerimientos de los sistemas SIG actuales, se ha decidido definir los siguientes símbolos: puntos, polilíneas, polígonos, texto, símbolos compuestos y símbolos extruídos. Cada una de estas entidades básicas se especializa en otras específicas para poder abarcar cualquier tipo de geometría deseada para un símbolo. De este modo tendremos, por ejemplo, puntos 2D y puntos 3D, que a su vez se especializan en modelos geométricos más concretos.

En nuestro modelo, un símbolo puede contener muchas entidades, que serán representadas del mismo modo. Cada símbolo requiere de un tratamiento distinto para mejorar la velocidad de visualización, por eso en el proceso de construcción del grafo se ha utilizado el patrón de diseño *visitor* [GHJV95]. Esencialmente, cuando el símbolo acepta al *visitor*, éste recorre todas las geometrías contenidas en el símbolo, creando un grafo de escena eficiente y listo para ser visualizado. Esta mejora de eficiencia se logra mediante el uso de *vertex arrays*, *index arrays* y *primitive sets* reduciendo las llamadas a funciones y eliminando la redundancia de vértices compartidos.

El funcionamiento es el siguiente: un *vertex array* almacena toda la geometría de un símbolo, compactando así todas las entidades que se expresen mediante ese símbolo en un único vector. Más tarde, se declaran los *primitive sets* estableciendo una relación entre conjuntos de vértices y primitivas OpenGL. El grafo de escena es el encargado de gestionar estos *vertex arrays* por medio de *display lists*, almacenando en memoria de tarjeta gráfica la información geométrica.

Para obtener el máximo partido de este método se ha decidido utilizar *color arrays* para definir el color de cada vértice. De este modo, el color de cada entidad no influye en el número de símbolos declarados, ya que en el mismo símbolo se pueden almacenar entidades de diferentes colores. Factores que obligan a la diferenciación entre símbolos son el tamaño, el patrón de relleno, la anchura de línea, etc.

A continuación se detallan las decisiones referentes a la arquitectura e implementación adoptadas para tratar los problemas asociados a cada tipo de símbolo.

Puntos

Un requerimiento corriente en los SIG es poder expresar puntos geográficos por medio de símbolos que sean fácilmente reconocibles por el usuario final (como gasolineras, hospitales, hoteles, etc.). Para lograr este objetivo las clases básicas han derivado en otras específicas siguiendo el siguiente esquema:

- **Point2DSymbol**: pueden ser representados por medio de círculos, cuadrados, triángulos y en general, cualquier tipo de polígono. A estos elementos se les puede aplicar una textura, ya que en la construcción del grafo se detallan las coordenadas de textura de cada geometría.
- **Point3DSymbol**: se da soporte para el dibujado de figuras geométricas tridimensionales básicas como cajas, esferas, conos y cilindros, además de la posibilidad de cargar cualquier modelo geométrico complejo desde archivo y utilizarlo para expresar dichos puntos geográficos.

Otro requerimiento típico es poder representar puntos en diferentes unidades de medida: píxeles y metros. En el caso de tener que representar los puntos en píxeles se añaden nodos *AutoTransfrom* al grafo de escena para mantener el tamaño del símbolo en pantalla aunque cambie el punto de vista.

Texto

Un factor decisivo en la usabilidad de un SIG es poder representar texto masivamente sin perder la capacidad de interacción. En el modelo que se plantea en **osgVP-Symbology** se proporcionan tres maneras diferentes de representar texto en un mapa tridimensional:

- **Text2D**: esta entidad es útil en el caso de renderizar textos que no se solapen entre ellos, o en el caso en el que la velocidad de render sea un factor clave en la visualización de determinada zona geográfica. A estos objetos se les puede aplicar *billboarding*, con lo que se consigue que el texto siempre esté encarado hacia el observador y que, por tanto, sea legible independientemente del punto de vista del usuario.
- **Text3D**: cuando un usuario desea visualizar texto con volumen y que no deba estar encarado necesariamente al observador se debe utilizar la estructura de **Text3D**. La visualización de este tipo de objetos tiene un coste computacional alto, por lo que deberían ser utilizados en el caso de que los factores estéticos primen sobre la velocidad de visualización.
- **FadeText**: esta estructura ha sido diseñada para los casos en los que unos textos están ocluidos por otros, dificultando la interpretación por parte del usuario de los datos representados. Cuando la cantidad de textos que deben ser visualizados es relativamente grande, o se encuentran en una zona geográfica pequeña la escena puede resultar ilegible. Por eso **FadeText** hace desaparecer automáticamente y por medio de una transición suave los textos que quedan ocultos por otros.

La *API* de **osgVP-Symbology** permite asignar parámetros tales como el tamaño, el color, la alineación o el tipo de fuente a utilizar en la representación de cualquier tipo de texto.

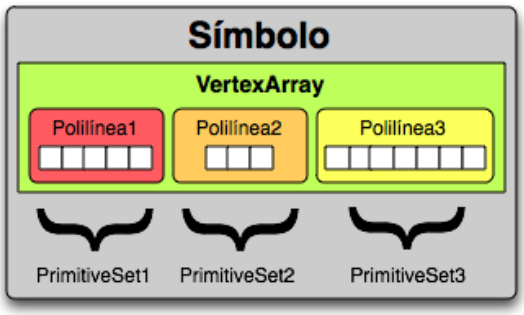


Figura 2: *Estructura interna de un símbolo de tipo polilínea*

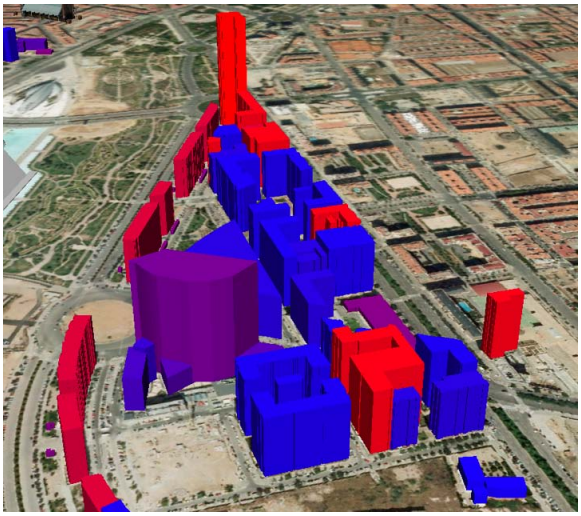


Figura 3: *Zona urbana de Valencia representada mediante símbolos de extrusión.*

Polilíneas y polígonos

La visualización de polilíneas y de polígonos es una actividad que los usuarios de SIG realizan con frecuencia. La figura 2 muestra la estructura interna de un símbolo de tipo polilínea, en el que varias entidades se compactan en el mismo *vertex array*.

Los usuarios podrán modificar el aspecto de una polilínea cambiando su grosor o patrón, y en caso de polígonos añadiendo textura.

Símbolos compuestos (Composite Symbols)

La combinación de símbolos es útil cuando se pretenda dibujar símbolos complejos o cuando se desee tratar varios símbolos como si fuesen uno sólo. En el modelo presentado, estas estructuras se han implementado siguiendo el patrón *composite*. Un ejemplo típico de la necesidad de este tipo de símbolos es la visualización de polígonos con borde con la finalidad de visualizar líneas fronterizas.

Geometrías definidas por extrusión

El componente **osgVP-Symbology** ofrece herramientas para visualizar entidades que tienen dimensión vertical en la realidad, como vallas, edificios, etc. Se ofrece un conjunto de herramientas de extrusión basado en un sistema de pilas de matrices que acumulan las transformaciones realizadas sobre la geometría original. Se dispone de extrusores especialistas en cada tipo de geometría que se desee extruir: puntos, polilíneas y polígonos.

- **PointExtruder**: permite al usuario convertir puntos en líneas. Estas estructuras se evidencian de verdadera utilidad cuando se usan combinadas con otros símbolos, por ejemplo en el caso de querer visualizar postes de electricidad o farolas. La direción y el color de la extrusión puede ser definida por el usuario.
- **PolylineExtruder**: esta entidad tiene la capacidad de transformar polilíneas en múltiples polígonos. Puede utilizarse masivamente en la representación de carreteras, vallas, etc. A estas estructuras se les puede aplicar textura, logrando una imagen más realista y permitiendo de este modo la fácil interpretación de los datos por parte del usuario.
- **PolygonExtruder**: la geometría resultante de la extrusión de un polígono puede utilizarse para representar edificios, estadísticas, simbolos volumétricos, etc. Este efecto se consigue mediante el uso de cintas de triángulos y asignación de coordenadas de textura. En la figura 3 se puede observar una estructura de este tipo para la visualización de una zona urbana.

3.2. Herramientas de edición

Una vez establecida la arquitectura necesaria para la visualización de la simbología 3D, se ha desarrollado un sistema que permite la edición de las entidades representadas por estos símbolos. El sistema es capaz de editar los puntos, líneas, polígonos y texto que forman las entidades 3D. Esta característica permite transformar de forma sencilla e interactiva las entidades presentes en los mapas. Esto será de utilidad, por ejemplo, si se quiere modificar la posición de una ciudad en un mapa (punto), cambiar el curso de un río (polilínea), o modificar los límites de una población (polígono). La realización de todas estas tareas es más fácil e intuitiva si se puede realizar visualmente en lugar de tener que editar los archivos que contienen esas entidades.

La implementación de los símbolos realizada por **osgVP-Symbology** hace que estos sean añadidos como nodos del grafo de escena. Por ello, se ha utilizado el componente

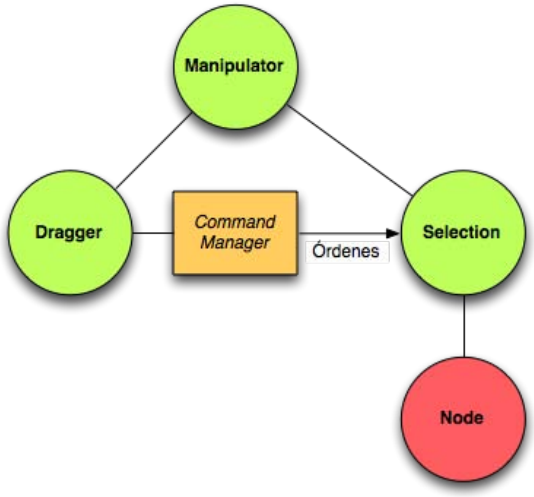


Figura 4: *Arquitectura del nodo Manipulator. En él podemos observar los elementos que lo forman: Dragger, Selection y CommandManager.*

osgVP-Manipulator para completar algunas de estas tareas de edición. Para llevar a cabo todas las transformaciones requeridas ha sido necesario realizar modificaciones en el componente, puesto que en un principio éste no permitía la edición individual de vértices. Actualmente, **osgVP-Manipulator** ofrece dos opciones para manipular un nodo. En primer lugar, se puede aplicar una transformación a toda la geometría incluida en el nodo. Por otro lado, se puede modificar sólo un conjunto de vértices de la misma.

A continuación, se explicará con más detalle las posibilidades que nos ofrece el componente **osgVP-Manipulator** como editor de grafos de escena y los cambios realizados para soportar la edición de conjuntos de vértices de las geometrías. Más tarde, se mostrará cómo adaptar la librería para que sea capaz de editar la simbología proporcionada por **osgVP-Symbology**.

Edición con Manipulator

El primer objetivo es que el sistema sea capaz de editar los nodos como una sola entidad, transformando las geometrías que se encuentran dentro de ellos. Esto es, todos los vértices de la geometría serán transformados conjuntamente. Para conseguir esto, se ha hecho uso del nodo **Manipulator** desarrollado en **osgVP-Manipulator**. Este nodo es especialmente útil si se quiere transformar toda la geometría al mismo tiempo. En la figura 4 se puede observar la arquitectura del subgrafo generado por el nodo **Manipulator**, y en la figura 5 se muestran distintos tipos de *dragger* de los que disponemos [ZTT*08], aplicados sobre un nodo.

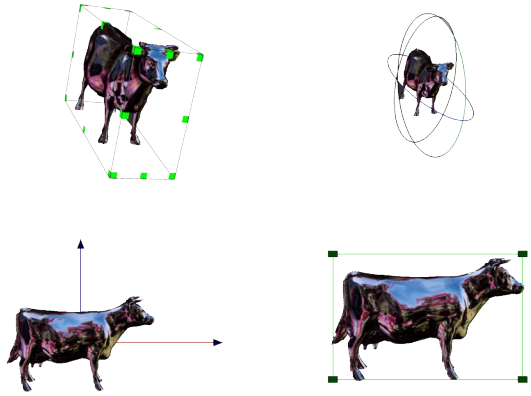


Figura 5: *Distintos tipos de draggers aplicados a un nodo.*

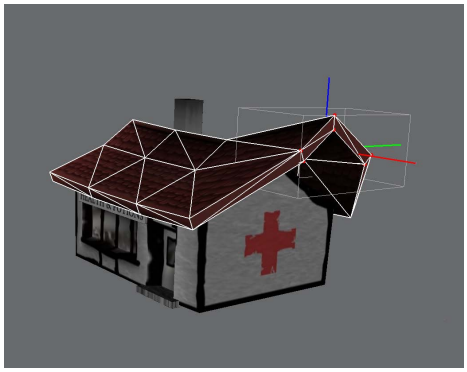


Figura 6: *Nodo GeometryManipulator. Podemos observar los 3 dragger de traslación 1D representando los ejes de coordenadas, y los 6 planos de traslación 2D que forman el MultiVertexDragger, representando la caja de inclusión que contiene los vértices a editar.*

Edición con GeometryManipulator

Para editar un subconjunto de vértices de un nodo, el sistema tiene que permitir la selección individual de los mismos. Para conseguir esto se ha desarrollado un nuevo nodo llamado **GeometryManipulator**. Este nodo toma como base la implementación de los manipuladores de **osgVP-Manipulator**, modificando su funcionamiento para adaptarse a estos nuevos requerimientos. Entre las modificaciones realizadas cabe señalar la creación de un nuevo tipo de *dragger* llamado **MultiVertexDragger**, el cual es una combinación de otros *draggers*. En la figura 6 podemos ver una captura de este tipo de estructura. También se ha creado un nuevo tipo de selección que se adapta a las nuevas órdenes generadas por el **MultiVertexDragger**, llamada **GeometrySelection**.

Los cambios realizados en la nueva selección residen en

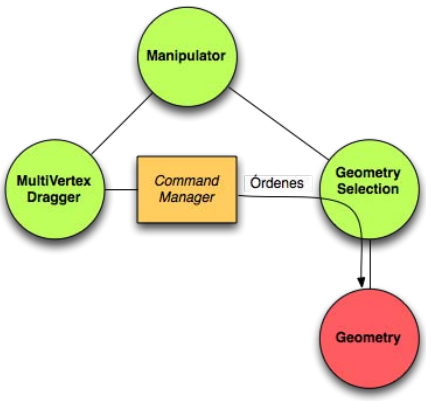


Figura 7: *Grafo generado al crear un nodo GeometryManipulator. Se puede apreciar cómo las órdenes recibidas por el GeometrySelection son aplicadas directamente sobre la geometría.*

cómo ésta procesa las órdenes que recibe. En este caso, las órdenes no modifican una matriz de transformación, como en el caso del nodo **Manipulator**, sino que directamente modifican el valor de los vértices de la geometría. Una diferencia importante es que esta selección no actúa sobre cualquier tipo de nodo, sólo puede actuar sobre nodos de tipo **Geometry**, que son los que almacenan la información sobre la posición de los vértices. Podemos ver un ejemplo que detalla la arquitectura del nodo **GeometryManipulator** en la figura 7.

Edición de las entidades

Anteriormente, se ha explicado cómo funciona la arquitectura del sistema de edición del grafo de escena y las diferentes posibilidades que nos ofrece. A continuación se procede a estudiar específicamente cómo editar las entidades representadas por los símbolos de **osgVP-Symbology**.

En el modo de visualización cada entidad puede representarse por un conjunto de diferentes símbolos, pero cuando una entidad pasa a estar en modo de edición, se le asocia una geometría básica (punto, polilínea o polígono) a la que se le pueden aplicar transformaciones afines mediante los manipuladores que aporta **osgVP-Manipulator**.

El paso de modo visualización a modo de edición se realiza mediante una selección interactiva de los símbolos dibujados en pantalla por medio del usuario. Estas transformaciones se hacen con el objetivo de modificar la geometría de la entidad, manteniendo la consistencia topológica con los datos vectoriales asociados.

Para hacer esta tarea más sencilla, cuando el modo de edición esté activo, se muestran puntos en el lugar correspondiente a cada una de las coordenadas pertenecientes a la geometría de las entidades, y todas las transformaciones se realizarán sobre dichos puntos. Los cambios realizados en las entidades se verán reflejados en los símbolos automáticamente, en cuanto el usuario vuelva al modo de visualización.

La edición de las entidades puede llevarse a cabo de modos diferentes según el tipo de manipulador escogido. El nodo **GeometryManipulator** permite realizar las operaciones de edición sobre los vértices en cualquier tipo de símbolo, al actuar directamente sobre las geometría propia de cada entidad. En el caso del nodo **Manipulator**, se transformarán al mismo tiempo todas las entidades que se encuentren representadas por el mismo símbolo. Esto es debido a las optimizaciones que realiza **osgVP-Symbology** en la generación del grafo de escena, agrupando todas las entidades en la misma geometría. En las figuras 8 y 9 podemos ver ejemplos de los dos nodos funcionando sobre las mismas entidades y las diferencias entre ellos.

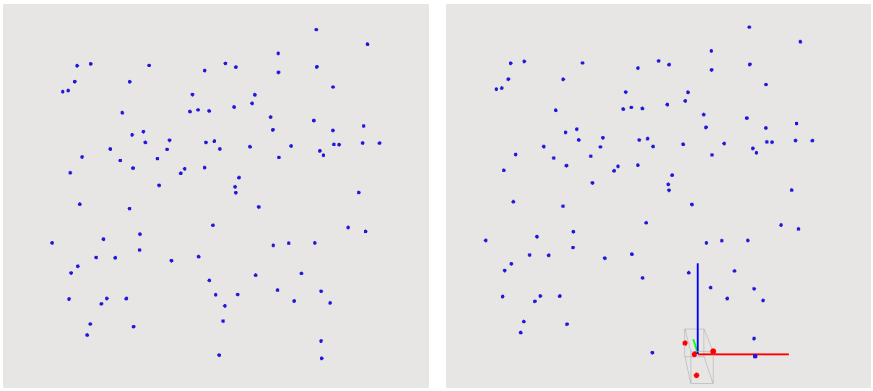
Por otro lado, la apariencia de los símbolos que representa a las entidades puede ser modificada por el interfaz gráfico de usuario que provea el sistema SIG en el que se implante la librería.

3.3. Integración en gvSIG

En esta sección se explicará cómo se ha adaptado la funcionalidad de este sistema de edición y visualización de simbología 3D dentro de la aplicación gvSIG. Para ello, se detallarán las herramientas creadas en la aplicación para incluir esta nueva extensión, y la arquitectura subyacente que permite su integración. El modelo de objetos utilizado para representar las geometrías por gvSIG es diferente al proporcionado por **osgVP**. Mientras el primero está enfocado a la representación de entidades presentes en los sistemas de información geográfica, el segundo está orientado a la representación tridimensional. Por tanto, el primer paso ha sido realizar un sistema que permita la conversión de las geometrías de un modelo al otro. Este sistema se ha desarrollado como una librería dentro del proyecto gvSIG llamada **libGPE-OSG**, siguiendo un patrón de conector.

El cliente gvSIG incorpora un sistema de extensiones para añadir nueva funcionalidad sin tener que modificar la estructura principal de la aplicación. Aprovechando este sistema se ha creado una nueva extensión llamada **ext3DCad** que permite utilizar la funcionalidad proporcionada por los componentes dentro del cliente gvSIG. De esta forma, cuando se instala la nueva extensión en el cliente aparece una nueva barra de herramientas con los siguientes elementos:

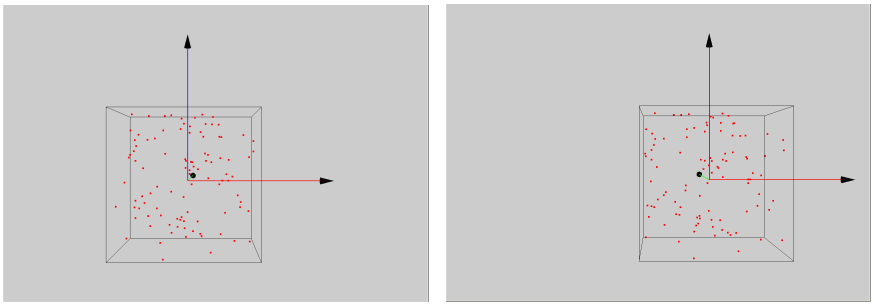
- **Selección**. Activa la herramienta para seleccionar las entidades que van a ser editadas.
- **Edición**. Cambia al modo de edición de las entidades mediante de los nodos **Manipulator** y **GeometryManipulator**.
- **Añadir Punto, Polilínea, Polígono, Texto**. Permite añadir una feature de estos tipos.



(a) En la siguiente capa de puntos todos los puntos son representados por el mismo símbolo.

(b) Se mueven cuatro de los puntos únicamente

Figura 8: *Edición de una capa de puntos con el nodo GeometryManipulator.*



(a) Capa de puntos, también representada con el mismo símbolo. En este caso se utiliza un dragger de traslación

(b) El nodo Manipulator los transforma conjuntamente, al estar toda la información almacenada en la misma geometría.

Figura 9: *Edición de una capa de puntos con el nodo Manipulator.*

Además de estas utilidades, se ha incorporado una herramienta para la edición *offline* de símbolos, que permite el uso de los nodos **Manipulator** y **GeometryManipulator** para cambiar tanto la geometría como la apariencia de los símbolos 3D que posteriormente se utilizarán para representar las entidades.

4. Resultados

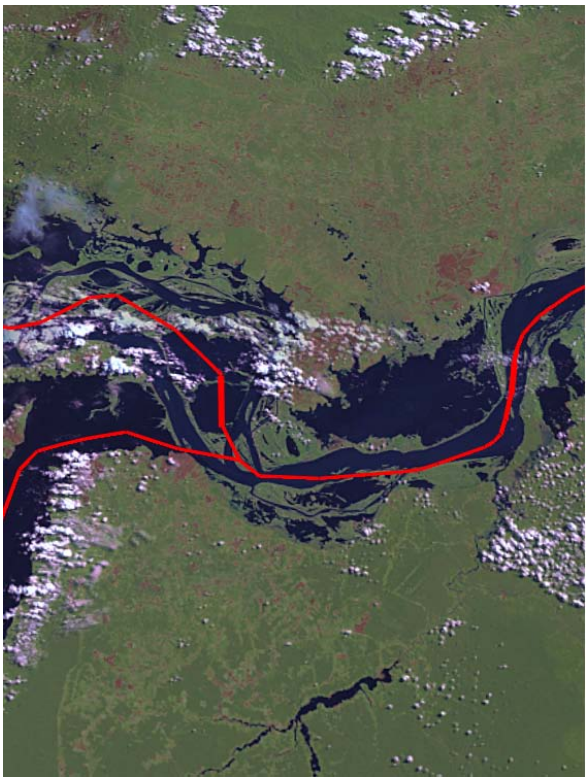
En esta sección se mostrarán los resultados obtenidos del desarrollo planteado en este artículo.

En la figura 10 podemos ver como editar una polilínea 3D utilizando las herramientas incorporadas al cliente gvSIG. Primero se debe cargar una capa que contenga datos vectoriales desde el propio cliente. Estos datos se renderizan utilizando un símbolo **PolilyneSymbol3D** de la librería **osgVPSymbology** como se muestra en la figura 10(a). Utilizando la herramienta de selección el usuario puede seleccionar qué elementos de la vista quiere editar. La entidad seleccionada muestra sus vértices para que puedan ser modificados como en la figura 10(b). Posteriormente, se puede seleccionar un conjunto de vértices de los disponibles de manera que se visualiza un *dragger* con el que podemos modificarlos como en la figura 10(c). Finalmente, la polilínea modificada se guarda en el formato de la capa vectorial, de manera que se pueda utilizar desde otras aplicaciones SIG.

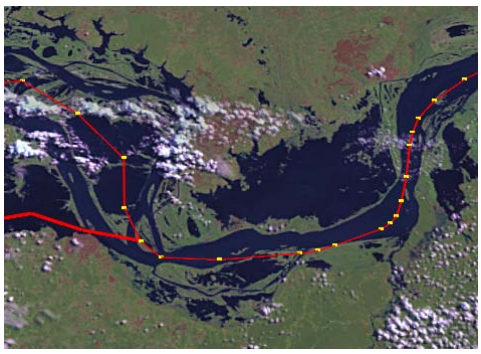
De manera análoga, en la figura 11 podemos ver que este mismo sistema sirve para otros tipos de datos vectoriales como los polígonos.

5. Conclusiones y trabajo futuro

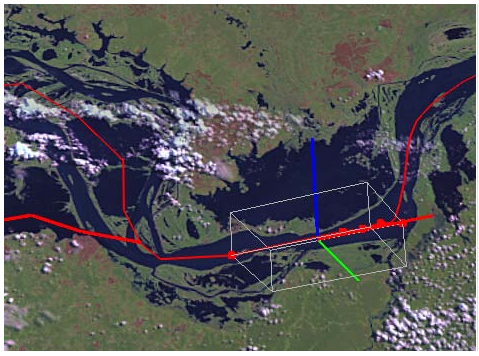
En este artículo se pone de manifiesto el aporte de la simbología tridimensional a la interpretación de datos vectoriales geográficos. También se ha presentado una librería que permite la visualización y edición interactiva de estos datos,



(a) Capa vectorial representando entidades de tipo polilínea mediante el símbolo PolilyneSymbol3D



(b) Una vez se ha seleccionado la entidad, se muestran los vértices a editar



(c) Se ha seleccionado un conjunto de vértices y se utiliza el dragger correspondiente. La polilínea queda modificada, actualizándose los datos vectoriales asociados.

Figura 10: *Ejemplo de edición de una polilínea en gvSIG con la librería osgVP.*

mediante un modelo escalable que ofrece una solución completa a los usuarios de aplicaciones SIG.

Por otro lado, se explica cómo mediante la implementación de un conector (**libGPE-OSG**) entre los modelos de datos de la librería y la aplicación, se ha podido añadir una extensión (**ext3DCad**) al cliente SIG gvSIG, capaz de llevar a cabo estas acciones.

El desarrollo, con licencia GPL, se ha orientado hacia la eficiencia, utilizando un grafo de escena (OpenSceneGraph) como base de la librería, característica todavía inédita en los sistemas SIG de código libre.

La implementación se ha realizado en C++ y se ha exportado toda su funcionalidad a Java, lenguaje en el que se desarrollan la mayoría de aplicaciones SIG licenciados bajo GPL, mediante JNI.

Como líneas de trabajo futuro se destacan la multirresolución de datos vectoriales 3D para su visualización masiva así como la implementación de geoprocesos para cálculos de volúmenes o la integración con otras aplicaciones SIG.

6. Agradecimientos

Este trabajo está siendo financiado por la *Conselleria d'Infraestructures i Transport* de la *Generalitat Valenciana* (Spain).

References

[AAJ06] ANUPAM AGRAWAL M. R., JOSHI R.: Geometry-based mapping and rendering of vector data over lod phototextured terrain models. In *14th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG)* (2006), pp. 1–8.

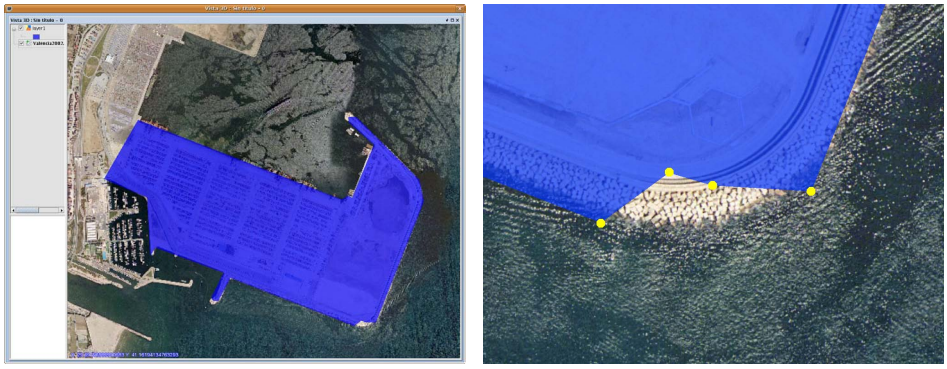
[ASZ07] A. SCHILLING J. B., ZIPF A.: Vector based mapping of polygons on irregular terrain meshes for web 3d map services.

[BN08] BRUNETON E., NEYRET F.: Real-time rendering and editing of vector-based terrains. In *Eurographics* (2008), vol. 27, pp. 311–320.

[Cit03] gvsig, 2003. http://www.gvsig.gva.es/.

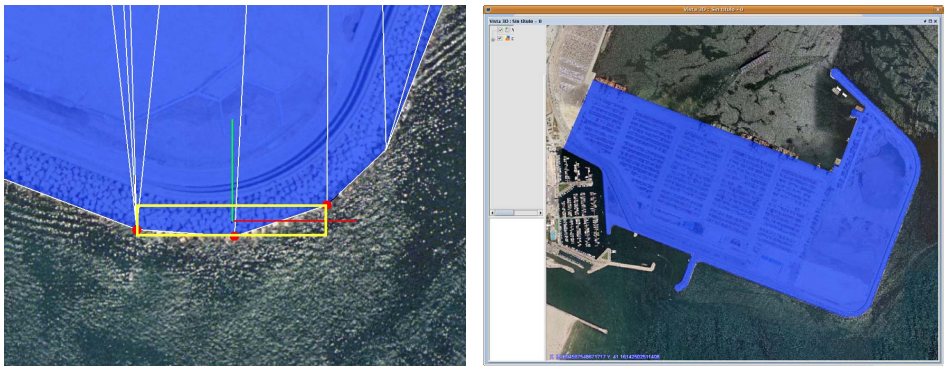
[Esr99] Arcgis 3d analyst, 1999.

[GHJV95] GAMMA E., HELM R., JOHNSON R., VLISSIDES J.: *Design Patterns. Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional Computing Series, 1995.



(a) En la imagen se muestran una capa raster del puerto de Valencia y una capa vectorial de polígonos, que contiene imprecisiones en su geometría.

(b) Zoom a la zona con imprecisiones, el polígono pasa a modo edición y se muestran los vértices de la entidad.



(c) Se ha seleccionado un conjunto de vértices y se ha corregido su posición por medio del dragger correspondiente.

(d) El polígono queda modificado, actualizándose los datos vectoriales asociados.

Figura 11: *Ejemplo de edición de un polígono en gvSIG con la librería osgVP.*

[MS07] MARTIN SCHNEIDER R. K.: Efficient and accurate rendering of vector data on virtual landscapes. *Journal of WSCG 15* (2007), 1–3.

[OB99] OSFIELD R., BURNS D.: Openscenegraph, 1999. http://www.openscenegraph.org.

[Ogc94] Open geospatial consortium, 1994. http://www.opengeospatial.org.

[Ogc04] Opengis web feature service (wfs) implementation specification, 2004. http://www.opengeospatial.org/standards/wfs.

[Ogc05] Web 3d service (draft implementation specification), 2005. http://www.opengeospatial.org/standards/dp.

[Ogc08] Opengis city geography markup language (citygml) encoding standard, 2008. http://www.opengeospatial.org/standards/citygml.

[OK02] OLIVER KERSTING J. D.: Interactive 3d visualization of vector data in gis. In *Proceedings of the 10th ACM international symposium on Advances in geographic information systems* (2002), pp. 107–112.

[SGK05] SCHNEIDER M., GUTHE M., KLEIN R.: Real-time rendering of complex vector data on 3d terrain models. In *In Proceedings of The 11th International Conference on Virtual Systems and Multimedia* (2005), pp. 573–582.

[WKW*03] WARTELL Z., KANG E., WASILEWSKI T., RIBARSKY W., FAUST N.: Rendering vector data over global, multi-resolution 3d. In *Joint EUROGRAPHICS - IEEE TCVG Symposium on Visualization 2003* (2003), pp. 213–222.

[ZTT*08] ZARZOSO J., TEN M., TORRES J., GAITÁN R., LLUCH J.: Edición vectorial de escenas 3d sobre openscenegraph. In *CEIG 2008: Congreso Español de Informática Gráfica* (2008), Eurographics Association, pp. 257–260.