CEIG'08, Barcelona, Sept. 3–5 (2008)
L. Matey and J. C. Torres (Editors)

Representación Vectorial 3D en un Sistema de Información Geográfica

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

Abstract

Se presenta una librería multiplataforma capaz de mostrar en un sistema de información geográfica datos vectoriales tridimensionales utilizando las técnicas de aceleración que ofrecen los grafos de escena. La representación de datos vectoriales es de especial interés, siendo un factor clave en diversas áreas del análisis geográfico. Se aportan herramientas de edición y visualización 3D capaz de dibujar puntos, líneas, polígonos, extrusión de geometrías y figuras geométricas básicas de una manera eficiente.

1. Introducción

La cartografía digital está experimentando en los últimos tiempos grandes avances debido a la popularización de los sistemas de información geográfica (SIG), cada vez más utilizados por las administraciones públicas y con fines más variados de los que se había venido utilizando tradicionalmente.

Por otro lado, los avances en el campo de visualización tridimensional, tanto en hardware como en software, han permitido que las imágenes, obtenidas mediante síntesis digital, tengan cada vez un mayor realismo así como una mayor capacidad de interacción.

Sin embargo, estos avances en informática gráfica no se ven plasmados en los SIG. La mayor parte de las veces estos sistemas presentan ortofotos superpuestas sobre un modelo digital del terreno, cuyos resultados no son muy realistas. La mayoría de las herramientas SIG incorporan algún tipo de visualización 3D, pero que desgraciadamente no colma las necesidades de los usuarios de este tipo de sistemas.

Este tipo de información suele ser almacenada en capas temáticas para satisfacer necesidades concretas de información. La forma de representar entidades vectoriales en la mayoría de los SIG es mediante: puntos, polilíneas, polígonos y multipolígonos. A estas entidades se les da el nombre de *features*. El objetivo del trabajo es mostrar este tipo de información con ciertas garantías. Las pruebas de integración se han realizado en gvSIG, un sistema multiplataforma y con licencia GPL.

Para lograr esta meta se ha decidido hacer uso del grafo de escena OpenSceneGraph (OSG). De esta manera se podrá agrupar features en estructuras de datos jerárquicas y acelerar el cálculo de la visibilidad.

2. Antecedentes

En esta sección se describirá sucintamente el estado actual de las tecnologías que sirven de base a este desarrollo, haciendo un breve repaso del estado actual de los Sistemas de Información Geográfica y de los modelos de datos que utilizan. También de los grafos de escena, explicando las técnicas de aceleración más importantes para la exploración de datos masivos.

La esencia de un SIG está constituida por una base de datos geográfica. Esta es, una colección de datos acerca de objetos localizados en una determinada área de interés en la superficie de la tierra. Existen diversas maneras de modelizar estas relaciones entre los objetos geográficos. Dependiendo de la forma en que ello se lleve a cabo se diferencian dos grupos principales: **formato raster** y **formato vectorial**.

El modelo de datos raster es especialmente útil cuando tenemos que describir objetos geográficos con límites difusos, donde los contornos no son absolutamente nítidos. En general, el modelo de datos vectorial es adecuado cuando trabajamos con objetos geográficos con límites bien establecidos, como pueden ser fincas, carreteras, etc.

En cuanto a desarrollos SIG que incorporen soporte tridimensional, por un lado existen aplicaciones como Google Earth, que cumpliendo las espectativas a la hora de visualización e interacción, adolecen de las herramientas necesa-

rias para ser consideradas aplicaciones técnicas de verdadera utilidad en análisis geográfico. Por otro lado existen aplicaciones propietarias como ArcMap (de la empresa ESRI) o MapInfo en las que la capacidad de interacción es bastante limitada y/o cuyo coste de licencia es muy elevado.También existen alternativas libres que incorporan visión tridimensional, como Grass o osgGIS. Aunque la mayoría de estos sistemas no son multiplataforma y algunos sólo incluyen una visualización 2D y media. Como ejemplo de SIG multiplataforma y de código libre se destaca gvSIG sobre el que se han hecho las pruebas de integración de la librería.

Además de las técnicas habituales de aceleración (véase [JF90] o [Zha98]), existen técnicas que, aunque menos conocidas, son muy útiles en el manejo de datos SIG. El **Small Feature Culling** [Far07] es una técnica derivada de la multirresolución [Hec94], consiste en eliminar de la escena aquellos objetos que son demasiado pequeños para ser vistos o que su tamaño excede un determinado umbral determinado por el usuario, acelerando de ese modo el cálculo de la visibilidad.

Actualmente existen muchas tecnologías que implementan grafos de escena, como Performer u OpenInventor. Las dos opciones que se han considerado más significativas y acordes con los objetivos propuestos son Java3D [Sun07] y OpenSceneGraph [OB99].

Por desgracia Java3D no aporta los últimos adelantos y mejoras incluidos en otros grafos de escena. Y, en algunos casos, no se obtiene la eficiencia necesaria en una aplicación de estas dimensiones.

El desarrollo OSG es un conjunto de librerías que proveen de organización de escenas y optimización del renderizado a aplicaciones tridimensionales. Esta escrita en ANSI C++ y hace uso del estándar de la industria OpenGL como API de bajo nivel. La mayor parte de OSG opera independientemente del sistema nativo de ventanas. Como resultado OSG es multiplataforma, pudiendo ejecutarse en la mayoría de sistemas operativos. OSG es de código abierto y está licenciado bajo la licencia LGPL, esto reporta muchos beneficios. De hecho OpenSceneGraph es uno de los grafos de escena disponibles de mayor rendimiento.

3. Análisis y diseño

Debe quedar claro que uno de los aportes fundamentales de este trabajo es la utilización de JAVA en la implentación, lo que permite la integración con gvSIG. En este apartado se abordará el problema inducido por el uso de este lenguaje, basado en máquina virtual, para el desarrollo de aplicaciones gráficas de alto rendimiento.

Para cumplir con el requisito de ser multiplataforma y ofrecer una API sencilla, la librería se está desarrollando en JAVA, esto permite que una máquina virtual implementada para cada arquitectura y sistema sea la que interprete el código del programa realizando las llamadas nativas pertinentes.

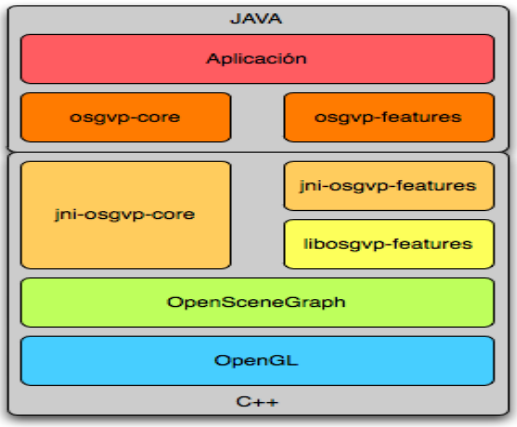

Figure 1: Capas de abstracción del sistema

Desgraciadamente el manejo que hace JAVA de la información 3D no es suficientemente eficiente para mostrar información vectorial en tiempo real con ciertas garantías.

Como se argumenta en [MC00] la principal desventaja de los lenguajes basados en máquina virtual, es que son más lentos que los lenguajes completamente compilados, debido a la sobrecarga que genera tener una capa de software intermedia entre la aplicación y el hardware de la computadora.

Existen aproximaciones a una solución para este problema ya implementadas, como es el *binding* de JAVA para OpenGL (JOGL), que no dejan de ser unos *wrappers* de de OpenGl para el lenguaje JAVA, con la inconveniencia de no implementar ninguna técnica de aceleración atribuible a cualquier grafo de escena.

3.1. Arquitectura del sistema

Para abordar la ineficiencia de JAVA en la representación de gráficos tridimensionales se ha decidido programar de forma nativa (en C++) y haciendo uso de la librería OpenSceneGraph, aquellas partes de la aplicación en las que se necesite rendimiento gráfico, creando más tarde un *wrapper*, o recubrimiento, que permita invocar a las funciones nativas desde JAVA.

La solución se ofrece en un paquete llamado **osgVirtualPlanets(osgVP)** que puede ser utilizado por programadores de aplicaciones SIG aportando un alto rendimiento en visualización de gráficos tridimensionales.

Se proponen, en principio dos librerías para proveer de visualización vectorial a la aplicación: osgvp-core y osgvp-features. Estas dos librerías proveerán de herramientas para la construcción del grafo de escena y para el dibujado de features.

En la figura 1 se muestra la arquitectura del sistema. El

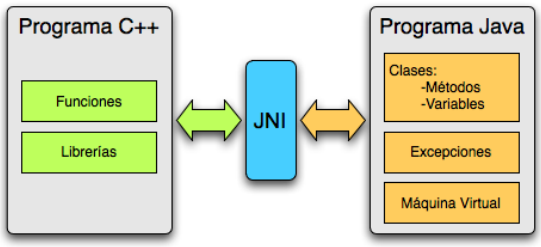

Figure 2: Esquema de funcionamiento de jni

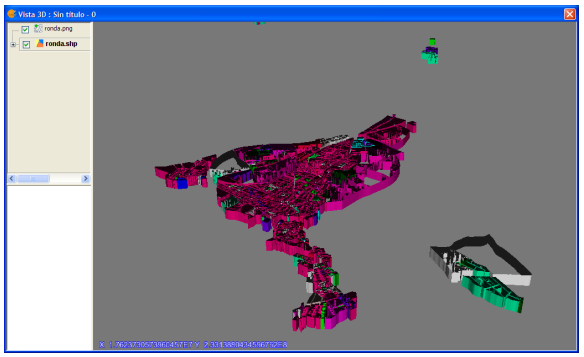

Figure 3: *Ejemplo de extrusión en funcionamiento sobre gvSIG.*

funcionamiento general es el siguiente: en la primera capa de abstracción se implementan en lenguaje nativo las herramientas para crear y manipular el grafo de escena así como añadir primitivas,almacenándose más tarde en forma de librería dinámica. Esta API puede ser utilizada por JAVA mediante JNI (Java Native Interface).Utilizando estas llamadas nativas se construyen las dos librerías de las nivel superior (osgvp-core y osgvp-features).

La librería **osgvp-core** integra los elementos necesarios para la construcción y optimización del grafo de escena, así como herramientas matemáticas para el manejo de datos vectoriales. El manejo de estas estructuras es bastante sencillo, simplemente añadiendo o quitando nodos al grafo podremos construir una escena.

Una buena estrategia a la hora de definir la simbología es determinante en la usabilidad de la librería **osgvp-features** . Diferentes tipos de features serán mejor expresadas por diferentes tipos de geometría.

Es necesario aprovechar la potencia de OpenGL en la rasterización de primitivas. Esto se puede lograr agrupando conjuntos de puntos o líneas en una sola primitiva, dando posibilidad de antialiasing, utilizando los algoritmos de teselación, permitiendo transparencia en cada una de las features, etc.

En aras de la simplicidad se ha decidido crear una capa de abstracción sobre OpenSceneGraph que dote de las funcionalidades requeridas por la aplicación final. El mapeo de la librería JAVA no es directo sobre clases OSG, estas clases se construyen a partir de la librería osgfeatures, implementada en C++.

4. Desarrollo

En esta sección se comentarán las decisiones de implementación más importantes razonando su validez. Primero se abordará el mecanismo utilizado para realizar las llamadas nativas, después, las características más destacadas de cada una de las librerías.

Las llamadas nativas se han implementado por medio de JNI (Java Native Interface). Como se puede observar en la figura 2, el sistema de tiempo de ejecución incluye el código necesario para cargar dinámicamente y ejecutar el código nativo que implementan estas llamadas. En el momento en el que se enlaza el módulo que contiene el código que implementan las llamadas, el procesador virtual captura el evento y se encarga de invocarlo. Para mayor información sobre la interfaz de métodos nativos se puede consultar [Lia99].

La librería osgvp-core incluye soporte para crear geometrías al estilo osg, pudiendo asignar diferentes tipos de primitiva a la lista de vértices que contiene. Estas listas de vértices se definen mediante el uso de vertexarrays, potenciando así el aprovechamiento del procesador gráfico. También contienen listas para almacenar las normales a cada vértice o coordenadas de textura.

Todas las features comparten una interfaz común, de modo que tareas como cambiar la transparencia o modificar los colores de cualquier feature resulte sencillo.

Siguiendo la mayoría de implementaciones de simbología SIG, se ofrece una API en la que los puntos pueden ser representados como tales (valores en píxeles), como quads (valores en metros) y mediante figuras geométricas (shapes y geometrías definidas por el usuario o cargadas desde base de datos).

El etiquetado es una de las operaciones más comunes en un SIG. Se ha dado la posibilidad de utilizar entidades de tipo *Text* para etiquetas y *FadeText*, que además permiten suavizado entre cambios de posición o contenido del texto.

En lo referente a extrusión de geometrías, se ha implementado un extrusor basado en pilas de matrices que acumulan las transformaciones que se aplican a la geometría original. A partir de aquí se han desarrollado extrusores especialistas en cada tipo de geometría a extruir, en un principio puntos, polilíneas y polígonos. Este tipo de objetos pueden utilizarse masivamente en un SIG, tanto para la creación de edificios, símbolos y visualización de información de diversa índole.

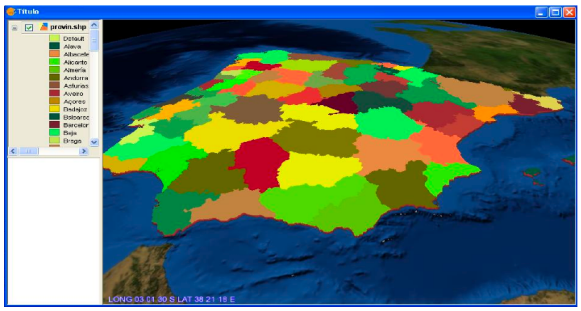

Figure 4: *Mapa político mediante el uso de polígonos teselados sobre gvSIG.*

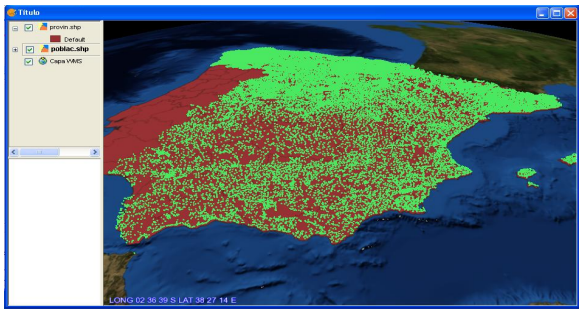

Figure 5: *Representación vectorial mediante puntos de las localides de españolas sobre gvSIG.*

La integración con gvSIG, ha resultado muy sencilla. Se han implementado nuevos cuadros de diálogo en Java, correspondientes a la interfaz 3D, y mediante la inclusión de las librerías de osgVP y realizando las llamadas adecuadas se han obtenidos resultados como los que se muestran a continuación.

5. Resultados

En esta sección se muestran algunas capturas de la integración de la librería con gvSIG.

En la figura 4 se puede apreciar el mapa político de España, separado por Comunidades. Para ello se hace uso exhaustivo de las rutinas de teselación que ofrece OpenGl.

La figura 5 muestra la inclusión de una capa de puntos utilizando la entidad PixelPoint.

6. Conclusiones y trabajo futuro

Se ofrece un completo *framework* de edición vectorial especialmente pensado para SIG capaz de mostrar información vectorial cartográfica tridimensional de manera efectiva y permitiendo la interacción. El análisis de la simbología utilizada en SIG ha sido fundamental en el desarrollo de las librerías.

El código es robusto, está bien organizado y forma parte del juego de herramientas osgVP. El desarrollo dirigido por tests y un buen sistema de construcción lo diferencian de otros frameworks del mercado, contando además con que osgVP tiene licencia GPL.

Se van a seguir en principio dos nuevas líneas en la investigación. Una atañe al exceso de tiempo que se consume implementando archivos JNI y creando nuevas capas de abstracción de OSG, la solución es la construcción de un nuevo Wrapper completo de OSG para Java que será generado automáticamente mediante el uso de introspección y reflexión de código. Otra línea de investigación es la multirresolución de features, lo que hará posible el aumento de la eficiencia de renderizado.

Se espera una buena acogida entre los programadores SIG, debido a la sencillez de utilización y a los resultados que se obtienen con pocas líneas de código.

7. Agradecimientos

Este trabajo está siendo financiado por la *Conselleria d'Infraestructures i Transport* de la *Generalitat Valenciana* (Spain) y en parte por el proyecto TIN2005-08863-C03-01 del Ministerio de Educación y Ciencia.

References

[Far07] FARINELLI P.: System and method for graphics culling, April 2007.

[Hec94] HECKBERT P.S. G. M.: Multiresolution modeling for fast rendering. In *Proceedings of Graphics Interface* (1994), pp. 43–50.

[JF90] J. FOLEY A. VAN DAM S. F. J. H.: *Computer Graphics: Principles and Practice. Second Edition in C.* Addison-Wesley, 1990.

[Lia99] LIANG S.: *Java Native Interface: Programmer's Guide and Reference*. Addison-Wesley Longman, 1999.

[MC00] MENDEZ R. M., CARBALLERIA F. G.: Arquitectura de la máquina virtual java. *Revista digital universitaria - UNAM 1*, 2 (October 2000).

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

[Sun07] Java 3d, 2007. http://java3d.dev.java.net.

[Zha98] ZHANG H.: *Effective Occlusion Culling for the Interactive Display of Arbitrary Models*. Tech. rep., 1998.