GGL2: Una alternativa real para el geoprocesamiento.

F. González Cortés [1] y V. González Cortés [2]

[1] Freelance, proyecto GearScape, Geomati.co, fernando.gonzalez@geomati.co.
[2] Freelance, proyecto GearScape, victorzinho@gmail.com

RESUMEN

GGL2 es un lenguaje de programación específico para el tratamiento de información geográfica que está especialmente diseñado para ser utilizado por profesionales del sector sin conocimientos avanzados de programación.
En este artículo se muestran las diferentes características que se han desarrollado en el transcurso del último año y que han permitido que GGL2 pase a ser una herramienta de geoprocesamiento flexible y productiva, capaz de interactuar con otras aplicaciones SIG, manejar diferentes tipos de datos, procesarlos y exportar los resultados de diversas maneras.
En efecto, la productividad del usuario se ve mejorada con la integración de GGL2 en gvSIG. Esta integración permite a GGL2 poder utilizar los datos disponibles en gvSIG desde el lenguaje así como mostrar los resultados en gvSIG de manera inmediata.
Uno de los desarrollos recientes más importantes ha sido el soporte raster, que se suma a la capacidad vectorial que ya tenía anteriormente el lenguaje y posibilita el tratamiento conjunto de ambos tipos de datos. Además, el acceso a nivel de muestra o píxel da una gran flexibilidad a la hora de aplicar filtros, operaciones de álgebra de mapas, etc.
También es posible el tratamiento de información en múltiples formatos, tanto tabulares (SHP, DBF, …) como jerárquicos (GPX, GML y XML en general), y todo ello con la misma sintaxis, lo cual proporciona una gran flexibilidad ya que el código es independiente de la ubicación y formato de los datos.
Así, por ejemplo, con GGL es posible realizar operaciones como modificar la coordenada Z de una capa vectorial en base a los valores de un modelo digital de terreno, realizar la resta de dos modelos digitales o filtrar las muestras de un raster en base a consultas sobre una capa vectorial, mostrando los resultados inmediatamente en gvSIG.

Palabras clave: *lenguaje, geoprocesamiento, GGL, GearScape, SIG, gvSIG, software libre.*

ABSTRACT

GGL2 is a domain-specific language for geographic data processing. It has been designed specifically to be used by geospatial technology professionals with no advanced programming knowledge.
This paper shows several features and improvements that have been developed through the last year. These changes have transformed GGL2 into a flexible and productive tool which is able to interact with other GIS applications, handle multiple data types, process them and export the results in several different ways.
Indeed, productivity is improved with the GGL2-gvSIG integration. This integration allows GGL2 to use gvSIG data directly from the language as well as to show results of geoprocesses in gvSIG immediately.
One of the most important recent developments is raster support which, together with GGL2 vectorial capabilities make it possible to process raster and vector data in the same algorithm. Furthermore, pixel level access provides high flexibility when applying filters, using map algebra, etc.
Moreover, GGL2 is able to read an write several formats, including tabular (SHP, DBF, …) and hierarchical (GPX, GML and XML in general) using the same uniform syntax. This provides a high flexibility since the GGL2 code is completely independent from the format and location of the processed data.
Thus, for example, with GGL2 it is possible to perform operations such as modify the Z coordinate of a vector layer according to the raster values of a digital elevation model (DEM), subtract two DEMs or filter raster samples depending on the values of a vector layer, showing the results in gvSIG immediately.

Key words: *language, geoprocessing, GGL, GearScape, SIG, gvSIG, free software.*

INTRODUCCIÓN

En este artículo vamos a mostrar los desarrollos realizados en GGL2, cómo se posicionan con respecto a los desarrollos anteriores y de qué manera contribuye esto a convertir a GGL2 en una alternativa más en el abanico de soluciones para geoprocesar con software libre. Mostraremos que GGL2 no sólo es capaz de resolver multitud de problemas relacionados con la geomática, sino hacerlo optimizando la productividad del usuario y maximizando el retorno de la inversión que supone el aprendizaje de un nuevo lenguaje. Además mostraremos algunos ejemplos de ello y valoraremos en qué medida se han seguido las líneas de trabajo futuro que se planteaban en la edición anterior [1].

Así, en primer lugar mostraremos las principales ventajas que se han añadido al lenguaje a lo largo del último año, como son la integración de GGL2 con gvSIG gracias al programa Google Summer of Code y la capacidad de procesar datos de tipo raster.

Posteriormente recordaremos brevemente las funcionalidades que ya existían en el lenguaje y cómo la integración con gvSIG y el soporte raster encajan con esas funcionalidades.

En los siguientes apartados mostraremos algunos ejemplos de las funcionalidades previamente expuestas y analizaremos de qué manera todo esto contribuye a la productividad y al retorno de la inversión por parte del usuario.

Por último valoraremos las conclusiones y expondremos brevemente las líneas de trabajo futuro que existen para GGL2.

INTEGRACIÓN CON GVSIG

La integración del lenguaje con gvSIG surge en el contexto del programa Google Summer of Code. Este programa consiste en potenciar la colaboración entre mentores, alumnos y organizaciones con el fin de desarrollar ideas con software libre. En el caso de GGL2, los autores de este artículo actuaron como mentor y alumno, mientras que gvSIG fue la organización que acogió la idea de integrar el lenguaje GGL2 con su aplicación de escritorio gvSIG Desktop.

Esta idea consiste en proporcionar al entorno de desarrollo de GGL2 la posibilidad de conectar con gvSIG. Como resultado, dicho entorno es capaz de obtener las capas definidas en gvSIG para usarlas en los algoritmos GGL2, así como mostrar los resultados de nuevo en gvSIG de manera completamente transparente para el usuario.

En lo que respecta a la obtención de datos desde gvSIG, las capas y tablas que estén definidas en gvSIG estarán disponibles en GGL2 como variables del lenguaje de forma implícita [1]. Sin embargo, puesto que el entorno de desarrollo está diseñado para permitir la conexión con varios SIG simultáneamente, es necesario diferenciar de alguna manera los datos que vienen de uno u otro. Así, las variables del lenguaje asociadas con datos de gvSIG tendrán como nombre el prefijo *gvsig_* seguido del nombre con el que se haya definido la capa, tabla, etc. en gvSIG. Por ejemplo, podemos crear una vista con una capa *vias* en gvSIG y mostrar el número de elementos de la capa con GGL2:

```
show gvsig_vias/@length;
```

En lo que respecta a la visualización de resultados producidos por GGL2 en gvSIG, la instrucción *show* del ejemplo anterior se transforma añadiéndole la construcción siguiente al final:

```
in gis as <escritor> '<nombre>';
```

donde *escritor* es el escritor que se va a encargar de escribir el resultado y *nombre* es el nombre que se le va a dar a la capa que se mostrará en la vista activa de gvSIG con el resultado de la operación.

Por ejemplo, es posible mostrar con GGL2 el buffer de la capa *vias* definida en gvSIG de la siguiente manera:

```
buffer_result = gvsig_vias select(v|ST_Buffer(v/the_geom, 10))
show buffer_result in gis as SHP 'buffer';
```

La ventaja de esta conexión es doble. Por un lado no es necesario escribir ninguna instrucción para leer los datos cargados en gvSIG, ya que estos se almacenan en variables implícitamente. Y por otro lado, no es necesario escribir los resultados a un fichero para luego importarlo en gvSIG, sino que la instrucción *show* realiza esta operación de manera automática.

SOPORTE RASTER

Una de las principales funcionalidades que se han añadido a GGL2 es la capacidad de procesar raster, básicamente añadiendo el tipo de datos básico *raster* e instrucciones para trabajar con variables de este tipo. En este apartado veremos cuáles son los formatos raster soportados por GGL2, así como las instrucciones que existen para manejar el raster a nivel de muestra, georreferenciarlo y acceder al valor del raster en una determinada coordenada.

En cuanto a los lectores que proporciona GGL2 para formatos raster encontramos ficheros TIFF georeferenciados (acompañados de ficheros *world file* .tfw), así como

ficheros ASCII Grid de ESRI (ASC), aunque un usuario con conocimientos de programación puede añadir nuevos lectores con facilidad.

Una vez leído el fichero raster, es posible realizar operaciones con sus muestras mediante la instrucción *rexp,* que recibe uno o más rasters como entrada y produce otro raster distinto como resultado. De esta manera, es posible realizar la suma de dos rasters distintos de la siguiente manera:

```
sum = rexp raster1, raster2 eval(raster1 + raster2);
```

Así, la instrucción *rexp* recibe *raster1* y *raster2* como entrada y produce un nuevo raster de las mismas dimensiones y cuyos valores son el resultado de aplicar la operación contenida en *eval* para cada una de las muestras, en este caso la suma. Notar que *raster1* y *raster2* son variables de tipo *raster* obtenidas de la lectura de dos ficheros TIFF o ASC. Además, en caso de que la operación a evaluar sea demasiado compleja, se proporciona una sintaxis similar a la de un algoritmo:

```
show rexp raster do (r1) {
        if (r1 < 100) {
                return 0;
            } else {
                return 1;
        }
}
```

donde tras la palabra reservada *do* se escribe, de manera opcional y entre paréntesis, el nombre de la muestra de entrada seguido de un bloque de instrucciones que utiliza dicha muestra como parámetro y acaba devolviendo otra muestra. Dicho bloque de instrucciones se evalúa para cada una de las muestras del raster de entrada, produciendo uno distinto de las mismas dimensiones y los valores evaluados como salida.

Además, GGL2 permite la georreferenciación de rasters. Para ello, cualquier variable de tipo *raster* cuenta con un hijo *georef* que contiene la información de georreferenciación. Este hijo no es más que un elemento que contiene 6 hijos que representan el ancho y alto del pixel en unidades del mapa, las rotaciones sobre los ejes X e Y, y las coordenadas X e Y del centro del píxel superior izquierdo. De esta manera, es posible por ejemplo acceder al ancho y alto del píxel de una variable *r1* de tipo *raster* de la siguiente manera:

```
show r1/georef/sizex;
show r1/georef/sizey;
```

Otra característica interesante es el acceso al valor del raster georeferenciado en una coordenada determinada. Para ello basta con utilizar la instrucción *at,* que recibe un raster como entrada, los valores X e Y de una coordenada como parámetros y devuelve el valor del raster en dicha coordenada:

```
show raster1 at (39.47, -0.38);
```

Por último, es importante destacar que la funcionalidad raster del lenguaje se puede integrar con las capacidades vectoriales de manera sencilla sin ningún problema. Por ejemplo, es posible crear un punto con coordenadas X e Y arbitrarias, y obtener la coordenada Z de un modelo digital de terreno:

```
x = 39.47;
y = -0.38;
show POINT(x y (raster1 at(x, y)));
```

Posteriormente se muestra un ejemplo más complejo que hace uso de esta capacidad para añadir la coordenada Z, obtenida a partir de un modelo digital de terreno, a una capa vectorial en 2D para posteriormente mostrarla en gvSIG.

OTRAS CARACTERÍSTICAS

Además de las nuevas características del lenguaje que se han desarrollado a lo largo del último año, y que hemos visto en apartados anteriores, existen también multitud de características que ya existían con anterioridad. Estas funcionalidades ya se discutieron en detalle en otras ediciones [1], por lo que no se entrará en profundidad en su discusión. Sin embargo, es importante recordarlas para obtener una visión global del lenguaje y observar cómo todas estas características pueden ser empaquetadas en algoritmos de forma que se maximice el retorno de la inversión, tal y como veremos en los siguientes apartados.

Una de las principales características del lenguaje son los operadores espaciales que se aplican sobre datos vectoriales para proporcionar una funcionalidad análoga a la del SQL espacial. Con estos operadores podemos realizar la selección de campos (*select*), el filtrado en función de una determinada condición (*filter*), agrupación de registros (*group by*), desagrupación de registros (*expand*) o unión de diferentes fuentes de datos (*join*). Ejemplos de estos operadores se pueden encontrar en [1]. Además, estos operadores vectoriales pueden interactuar con las capacidades raster del lenguaje como se ha visto en el apartado anterior y como se muestra más adelante con un ejemplo.

También es importante también recordar que GGL2 cuenta con un entorno de desarrollo que proporciona asistencia al usuario, tal como detección inmediata de errores, compilación automática o compleción de código. Esta plataforma ha sufrido algunas mejoras de usabilidad debido a la integración con gvSIG, tales como un nuevo asistente de ejecución de algoritmos, la herramienta para conectar y desconectar de gvSIG o la detección automática de errores relacionados con dicha conexión.

Por último, se ha desarrollado también una versión en línea de comandos para compilar y ejecutar programas escritos en GGL2 de manera ligera y sin necesidad de interfaz gráfica.

Todas estas características hacen permiten a GGL2 afrontar un gran abanico de problemas de geoprocesamiento. A continuación veremos la característica más importante de todas, que permite empaquetar en algoritmos los distintos scripts GGL2 y abstraerlos de los detalles concretos del problema: estructura de los datos, ubicación, formato, etc.

EMPAQUETADO DE ALGORITMOS

Antes de entrar en las principales ventajas del empaquetado de algoritmos, introduciremos la forma de acceder a los datos con GGL2 para proporcionar un contexto en el que se entienda mejor tanto el uso de algoritmos como las ventajas que proporcionan.

Acceso a estructuras de datos dispares

GGL2 permite el acceso a estructuras de datos muy dispares, tanto de tipo tabular como de tipo jerárquico o raster. Para ello proporciona dos instrucciones básicas de lectura y escritura:

```
read SHP 'data/buildings.shp' to buildings;
write SHP buildings to '/tmp/buildings.shp';
```

En ambas basta con especificar el lector/escritor utilizado (*SHP* en el ejemplo), el lugar donde se van leer/escribir los datos y la variable utilizada para leer/escribir.

Además, GGL2 proporciona multitud de lectores y escritores para los distintos formatos, vectoriales y raster. Entre ellos, cabe destacar el acceso a SHP, DBF, CSV, servicios web, TIFF, ASCII Grid de ESRI, XML, etc. Es importante notar que el lector de XML es genérico y nos puede servir para leer tanto KML como GPX como cualquier otro fichero XML con un esquema procesable por el lector.

Por si esto no fuera suficiente, GGL2 también proporciona al usuario la posibilidad de definir sus propios lectores y escritores para el formato que se requiere. Basta para ello definir un *reader* (o *writer* para escribir) y asociarlo a una clase Java que implemente dicha lectura (o escritura):

```
reader CSV maps to org.gearscape.ggl.readers.csv.CSVReader;
writer CSV maps to org.gearscape.ggl.readers.csv.CSVWriter;
```

Una vez leídos los datos, estos se asocian a una variable que tendrá un tipo inferido a partir de la estructura de los datos leídos. Si se tuviera que tener en cuenta el origen de los datos a la hora de implementar un geoproceso, como puede ocurrir en un lenguaje como Java, la reutilización de dicho geoproceso con distintas fuentes de datos se convertiría en una tarea tediosa.

Es por esto que GGL2 proporciona varios mecanismos que dotan de gran flexibilidad a los geoprocesos implementados de forma que su reutilización, y por tanto la productividad de los usuarios, se vean incrementados lo máximo posible.

Uno de estos mecanismos es la homogeneidad de la sintaxis del lenguaje. Independientemente de si los datos se han leído, por ejemplo, de un GPX, de un SHP o de un CSV, la sintaxis para manejar esas variables será homogénea; es decir, se puede utilizar el operador *filter* para filtrar los datos sin distinción del origen de los mismos.

Además, esta sintaxis homogénea puede encapsularse o empaquetarse en los denominados algoritmos. Estos algoritmos, gracias al polimorfismo de los parámetros, hacen que la reutilización del código y la definición de procesos repetitivos sean tareas mucho más simples.

Independencia de los algoritmos

Un algoritmo es un proceso que realiza determinados cálculos de manera independiente o, dicho de otra manera, la unidad mínima de reutilización que realiza una tarea significativa y completa. En concreto, GGL2 define los algoritmos como una serie de instrucciones que toma unos valores como entrada y produce otro valor como resultado.

La ventaja principal de los algoritmos es la reutilización de código. Es decir, es posible utilizar el mismo algoritmo con distintos valores de entrada y en distintos puntos del código. Un ejemplo claro de esto es el algoritmo predefinido en GGL2 *ST_Envelope*. Este algoritmo calcula el rectángulo mínimo de lados paralelos a los ejes de coordenadas y que contiene a la geometría enviada como parámetro. De esta forma, podemos reutilizar el algoritmo cuantas veces queramos simplemente con llamarlo en cualquier punto de nuestro programa y con cualquier geometría como valor de entrada. Además, una vez el algoritmo está implementado, como es el caso, no es necesario conocer la implementación del mismo; simplemente nos basta con saber qué hace y cómo llamarlo:

```
read SHP 'data/buildings.shp' to buildings;
foreach g in buildings/the_geom {
      show ST_Envelope(g);
}
```

Pero no es esta la única ventaja que proporcionan los algoritmos en GGL2. También es posible hacer uso del polimorfismo que se detalla en el apartado siguiente.

Polimorfismo

Entendemos por polimorfismo a la capacidad de un lenguaje de programación que nos permite manejar valores de distintos tipos utilizando una interfaz uniforme [2]. En el caso de GGL2, el polimorfismo nos permite asignar valores o estructuras complejas a variables cuyo tipo es más simple.

Por ejemplo, conociendo la definición de la *FeatureCollection* de OGC como elemento de GGL2:

```
define Feature as element {
      geometry the_geom,
      int id
};

define FeatureCollection as sequenceof Feature;
```

podemos leer un fichero *.shp* con multitud de campos (entre los que se debe encontrar uno *id* de tipo entero) y asignarle el valor leído a una variable de tipo *FeatureCollection*:

```
read SHP 'data/buildings.shp' to buildings;
FeatureCollection features = buildings;
```

De esta manera, a pesar de que la variable *buildings* pueda tener mucha más información, la variable *features* únicamente tendrá el campo geométrico y el identificador, todo ello sin hacer ningún uso adicional de memoria.

La principal ventaja de esta característica es el hecho de poder reutilizar código usando una variable cuyo tipo tenga sólo los campos que necesitamos para nuestro algoritmo, independientemente de que el valor referenciado por la variable contenga más información. Esto resulta realmente útil cuando se utiliza junto con los algoritmos. Así, podemos definir los parámetros de un algoritmo especificando un tipo determinado y luego llamarlo con cualquier variable que contenga al menos la información definida por el tipo.

Por ejemplo, es posible definir un algoritmo que reciba una *FeatureCollection* y devuelva la suma de áreas de todas las *features* cuyo identificador sea menor al segundo parámetro:

```
alg areaSum(FeatureCollection fs, double max) returns double {
      areas = fs filter(fs/id <= max)
                        select(a | area=ST_Area(a/the_geom));
      return sum(areas/area);
}
```

y posteriormente utilizarlo en un programa que haya leído cualquier secuencia de datos que se puedan asignar mediante polimorfismo a una *FeatureCollection*:

```
read SHP 'data/buildings.shp' to buildings;
show areaSum(buildings);
```

Así, es posible reutilizar el algoritmo *areaSum* con cualquier conjunto de datos leído que contenga geometrías e identificadores sin tener que reescribir el código o tener en cuenta la fuente de datos a la hora de implementar el algoritmo. Dicho de otra forma, podremos utilizar exactamente el mismo algoritmo tanto si hemos leído la *FeatureCollection* de un fichero GML o de un *shapefile*.

ALGUNOS CASOS PRÁCTICOS

En esta sección se muestran dos ejemplos que hacen uso de las características vistas en los apartados anteriores. El primero de ellos realiza la extrusión de una capa de edificios utilizando un campo altura, mientras que el segundo de ellos añade la coordenada Z a una capa 2D en función de un modelo digital de terreno.

Extrusión

Este ejemplo hace uso tanto de la integración con gvSIG, como del empaquetado de algoritmos. En primer lugar, se define un algoritmo que extrude una determinada geometría utilizando la altura enviada como parámetro:

```
alg extrudeBuilding(geometry building, double h) returns
sequenceof geometry {
      sequenceof geometry ret;
      geometry prev;
      process building coordinates as g do {
            if (prev is not null) {
                  x0 = ST_X(prev);
                  y0 = ST_Y(prev);
                  x1 = ST_X(g);
                  y1 = ST_Y(g);
                  add POLYGON((x0 y0 0, x0 y0 h, x1 y1 h,
                        x1 y1 0, x0 y0 0)) to ret;
            }
            prev = g;
      }

      add building to ret;
      add gexp building eval
            (g | POINT((ST_X(g)) (ST_Y(g)) h)) to ret;

      return ret;
}
```

El algoritmo recibe una geometría en 2D y una altura, y genera una secuencia de geometrías con todas las caras del edificio en 3D, el suelo y el techo. En este artículo no entraremos en detalle en la implementación del algoritmo. Sin embargo, en [3] es posible encontrar una explicación paso a paso del código del mismo.

Una vez disponemos del algoritmo, podemos utilizarlo en un sencillo programa para obtener la extrusión de todos los algoritmos de una capa:

```
read SHP 'data/buildings.shp' to buildings;
extruded = buildings
   select (b | the_geom = extrudeBuilding(b/the_geom, b/ALTURA))
   expand (a | a/the_geom);
show extruded in gis as SHP 'extruded';
```

En este programa primero se lee el fichero SHP de *data/buildings.shp*, que contiene un campo *ALTURA* que determina la altura de cada uno de los edificios. En la siguiente instrucción, mediante el operador *select*, se obtiene la extrusión de los edificios de la capa leída haciendo uso del algoritmo definido anteriormente. Puesto que para cada edificio se devuelve un secuencia de geometrías, el resultado del operador *select* es una secuencia de secuencias de geometrías. Para obtener una secuencia de geometrías “plana” se concatena el operador *expand* que realiza exactamente lo que buscamos. Una vez tenemos una secuencia de geometrías con

todas las caras, techos y suelos de los edificios, simplemente usamos la instrucción *show in gis* para mostrar el resultado en gvSIG con el nombre *extruded*:

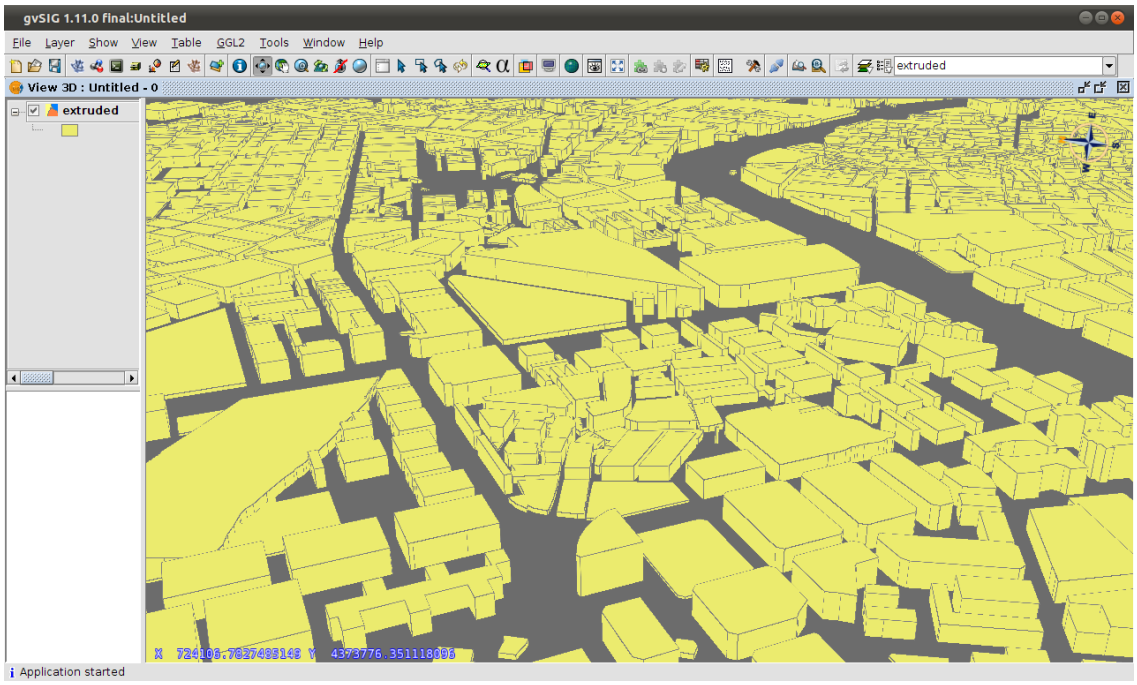

Figura *2:* Visualización de la extrusión con gvSIG.

Addz

Este ejemplo, al igual que el anterior, hace uso tanto de la integración con gvSIG, como del empaquetado de algoritmos. Sin embargo, en este ejemplo también se hace uso de las operaciones raster del lenguaje en conjunción con las capacidades vectoriales del mismo.

En primer lugar, se crea un algoritmo que añade la coordenada Z a la geometría en función del modelo digital de terreno enviado también como parámetro:

```
alg addZ(geometry g, raster dem) returns geometry {
      return gexp g do {
            x = ST_X(g);
            y = ST_Y(g);
            return POINT(x y (dem at(x, y)));
      };
}
```

Como se puede observar, simplemente se utiliza la instrucción *gexp* para, por cada punto de la geometría, devolver otro punto con las mismas coordenadas X e Y, y añadiendo la coordenada Z a partir del modelo digital de terreno.

Una vez hemos definido el algoritmo que añade la coordenada Z a una determinada geometría, es necesario aplicarlo a cada una de las geometrías de nuestra capa de entrada. Para ello basta con leer la capa vectorial, el modelo digital de terreno y utilizarlos en el operador *select*:

```
read TIFF 'data/dem.tif' to dem;
read SHP 'data/buildings.shp' to shp;
show shp select(addZ(shp/the_geom, dem)) in gis as SHP 'addz';
```

Como vemos en la última instrucción, la forma de mostrar la capa procesada en 3D en gvSIG es análoga a la utilizada en el ejemplo de extrusión. El resultado es el que se muestra en la Figura 3. Notar que en la figura se ha añadido una capa *zero* con la capa original a altura constante igual a 0 para dar una mejor perspectiva del resultado.

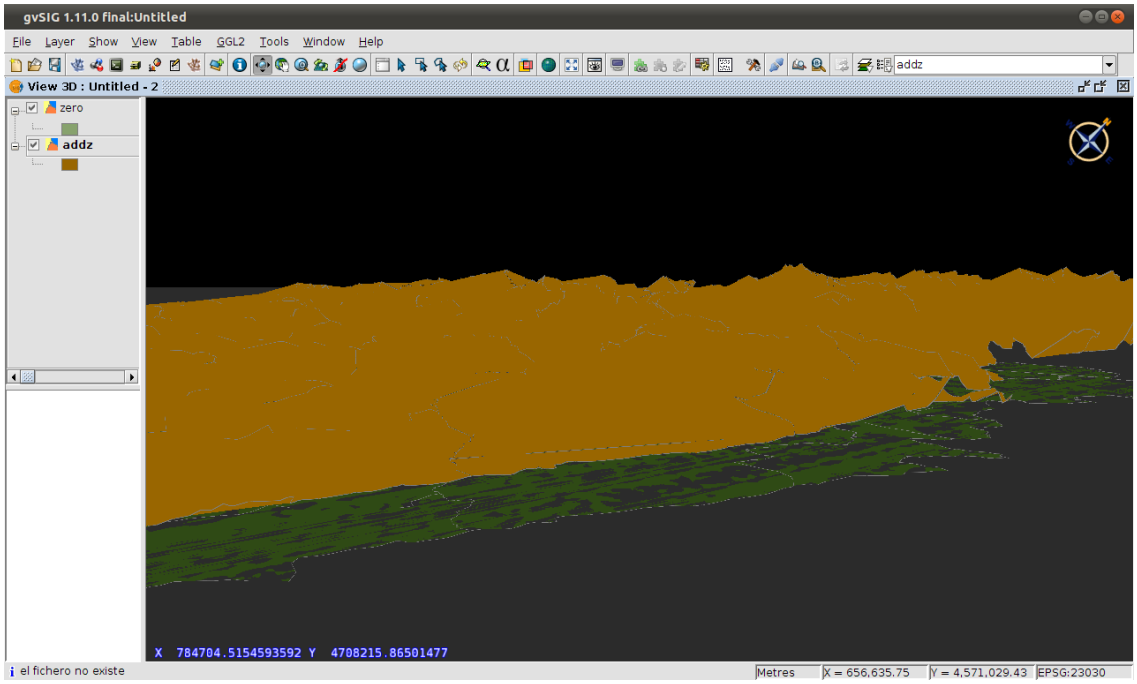

Figura *3*: Visualización de addZ con gvSIG.

RETORNO DE LA INVERSIÓN

En los puntos anteriores se ha podido observar de qué manera GGL2 ofrece soluciones para una gran variedad de problemas. A continuación se analiza el retorno de la inversión en el aprendizaje y uso de GGL2.

Inversión

La principal inversión al usar GGL2 es el aprendizaje. Pese a que GGL2 está especialmente diseñado para ocultar la mayor parte de detalles que ocurren detrás, su uso requiere la asimilación de conceptos que para los usuarios sin conocimientos de programación pueden no ser triviales, lo cual requiere tiempo y esfuerzo.

Además, la resolución de un problema de geoprocesamiento requiere la escritura de un script que lo solucione. Dicha escritura requiere cierto esfuerzo mental, algunas pruebas, posibles consultas a la documentación, etc.

Retorno

La primera ventaja de resolver un problema de geoprocesamiento con GGL2 es la repetibilidad. Como hemos visto anteriormente, el polimorfismo de tipos en GGL2 permite aplicar un mismo algoritmo a distintos datos de entrada mientras estos cumplan unos requisitos mínimos. Incluso es posible aplicar la solución a los mismos datos mucho tiempo después, sin necesidad de recordar los detalles de la solución. De esta manera, con GGL2 nunca es necesario resolver el mismo problema dos veces.

Por otra parte, al tratarse de un lenguaje de programación, GGL2 proporciona total flexibilidad a la hora de afrontar los problemas. Mientras que la utilización de

programas que presentan diálogos predefinidos para la realización de geoprocesos es más simple, el usuario está limitado por las posibilidades que ofrecen estos diálogos. En cambio, GGL2 proporciona un amplio conjunto de operadores que pueden ser combinados con total libertad mientras se respeten las reglas semánticas del lenguaje.

Además, dicha flexibilidad permite la creación de un cuerpo de soluciones cada vez mayor. De la misma manera que se puede reutilizar un algoritmo para resolver un problema, también es posible incluirlo en la solución de otro problema de manera que cada vez se pueden crear soluciones a problemas más complejos con menos esfuerzo. Por ejemplo, ahora que ya se conoce la solución al problema de la extrusión, sería posible crear una solución para cálculos de radiación solar que use la extrusión internamente.

Estas ventajas son mucho más beneficiosas cuando se trabaja en comunidad. Por una parte, por el hecho de haber mayor cantidad de scripts, las probabilidades de que haya un script para solucionar un problema dado son mayores. Por otro lado, cuanto mayor es la comunidad, mayor es la probabilidad de colaboración entre usuarios de distintos dominios. En una comunidad pluridisciplinar se crearán scripts para solucionar problemas propios de cada dominio y estos podrán ser utilizados por usuarios ajenos a dicho dominio sin necesidad de entender los detalles de implementación.

Además, las dinámicas propias de las comunidades de software en los usuarios se ayudan mutuamente se ven potenciadas en el caso de GGL2 por el hecho de que los scripts pueden ser fácilmente enviados en mensajes a las listas de difusión y de esa manera se puede dar y obtener soporte técnico más fácilmente.

CONCLUSIONES

En el presente artículo se han presentado algunas características de GGL2 que han sido recientemente añadidas o que son especialmente relevantes desde el punto de vista del retorno de la inversión.

Por el hecho de ser un lenguaje de programación se obtiene una gran flexibilidad que, unida a la capacidad de acceder a distintos tipos de datos vectoriales y raster, permite solucionar un gran abanico de problemas sin necesidad de terceras aplicaciones.

Dicha flexibilidad, junto con la facilidad para aplicar un algoritmo dado a distintos juegos de datos y en distintos formatos, permite construir incrementalmente una base de conocimiento reutilizable que es la base del retorno de la inversión en el aprendizaje y uso de GGL2.

TRABAJOS FUTUROS

Actualmente la base del lenguaje se considera casi completa. Existen algunas carencias en cuanto a procesamiento raster ya que no es posible tratar rasters multibanda, aunque es de esperar que en breve se solucione esta limitación.

A partir de este punto, las carencias principales son *readers* y *writers* para el acceso a datos, algoritmos, etc. pero todos estos artefactos ya no forman parte del lenguaje como tal sino de base de conocimiento que se construye sobre él y en el que se espera la participación de la comunidad que comienza a formarse.

Por tanto, uno de los objetivos dentro del equipo de desarrollo es la difusión de la solución por todas las vías posibles.

En el plano técnico, esto se traduce en la integración del lenguaje en distintos sistemas para los que se interprete dicha integración como interesante, notablemente el servidor de procesos WPS de 52º North, SEXTANTE y soluciones de escritorio como QGIS.

AGRADECIMIENTOS

Se agradece a la asociación gvSIG su papel como organización en el programa Google Summer of Code, así como la aceptación y el apoyo tanto de la idea de integración de GGL2 y gvSIG Desktop, como de mentor y alumno.

Se agradece también a Google tanto la inversión como la ayuda y el apoyo aportado para la consecución de la integración.

REFERENCIAS

[1] GONZÁLEZ, F.; GONZÁLEZ, V. (2011), “GGL2: Un lenguaje específico para SIG”, V Jornadas de SIG Libre, Girona.

[2] Polymorphism (computer science), Wikipedia (2012), http://en.wikipedia.org/wiki/Polymorphism_(computer_science)

[3] GONZÁLEZ, V. (Febrero 2012). *Extruding Buildings*. GGL2, http://ggl-two.blogspot.com/2012/02/extruding-buildings.html