Tecnologías Geográficas para el Desarrollo Sostenible
Departamento de Geografía. Universidad de Alcalá, 2000: 16-30.

Desarrollo de una Librería de Clases Espaciales

Francisco Alonso Sarría
Area de Geografía Física. Universidad de Murcia

Resumen

Se presenta un intento de crear un conjunto de clases espaciales que sirvan para desarrollar aplicaciones de análisis espacial, modelización y SIG. Se pretende con ello explorar las posibilidades del software abierto en el campo de la Geografía cuantitativa.

1. Introducción

Uno de los principios fundamentales en que se basa la geografía cuantitativa es el análisis espacial de datos. Aunque parece aceptado que una de las herramientas más poderosas para llevar a cabo este tipo de análisis son los Sistemas de Información Geográfica, en realidad su diseño, más orientado hacia la cartografía, no les hace especialmente adecuado.

1.1. Análisis espacial

En los últimos tiempos parece establecerse una identificación entre análisis espacial y Sistemas de Información Geográfica. Sin embargo la teoría del análisis espacial y de los modelos de objetos y procesos espaciales es, desde luego, anterior al desarrollo de los SIG (Chorley, 1972; Davis & McCullagh, 1975; Getis & Boots, 1978). Esta teoría del análisis se ha ido desarrollando de forma no siempre paralela al desarrollo de los SIG y con la introducción en las publicaciones de programas de demostración (Bailey & Gatrell, 1995).

La aplicación de herramientas de análisis espacial ha tenido más difución en el campo de la geografía humana (Gamir Orueta *et al.*, 1995). No obstante también aparecen ejemplos en la geografía física (Chorley, 1972; Anderson, 1988). Mención especial merece el desarrollo en la hidrología de herramientas que podrían incluirse en el análisis espacial, como la morfometría de redes de drenaje iniciada por Horton (1932, 1945) y Strahler (1952, 1957) cuya evolución ha dado lugar a su utilización en la modelización hidrológica mediante herramientas como el Hidrograma Unitario Geomorfológico (Rodriguez Iturbe, 1993) o el desarrollo de técnicas fractales para el estudio de redes de drenaje (Rodriguez Iturbe & Rinaldo, 1997).

Además de su desarrollo individual, el análisis espacial se ha integrado en el desarrollo de los Sistemas de Información Geográfica (Fotheringham & Rogerson, 1994; Mather, 1993). Sin embargo se han lanzado algunas críticas contra esta integración en el sentido de que han supuesto un escaso avance en el análisis espacial en si mismo, se ha enfatizado el análisis estadístico y, en general conducen a una implementación simplificada de herramientas que puede conducir a errores a usuarios con escaso conocimiento de las mismas (Fotheringam, 2000).

Por otra parte, recientemente han empezado a desarrollarse en torno al grupo de la universidad de Leeds nuevas ideas en cuanto al análisis de datos espaciales, agrupadas bajo el nombre de *GeoComputación* (con C mayúscula). Según sus deesarrolladores, los SIG tal como están concebidos han heredado características que pueden considerarse de *ciencia dura* (Couclelis, 1998). Los seguidores de esta nueva idea postulan el desarrollo de técnicas computacionales que aprovechen las posibilidades de los ordenadores actuales, basadas en técnicas de Inteligencia Artificial (algoritmos genéticos, redes neuronales, sistemas expertos, lógica borrosa), menos paramétricas y más orientada al análisis exploratorio de datos (Openshaw, 2000).

1.2. Sistemas de Información Geográfica

En el desarrollo de los más recientes proyectos de SIG aparecen dos lineas fundamentales. En primer lugar programas desarrollados en universidades, con objetivos principalmente medioambientales y de gestión de recursos naturales, en segundo lugar proyectos desarrollados por la iniciativa privada que resultan en herramientas de tipo *desktop mapping*. Estas se orientan más a la administración y empresas en las que al usuario final no se le supone un especial dominio en el manejo de ordenadores o en las bases de los Sistemas de Información Geográfica.

Los primeros utilizan fundamentalemente modelos de datos de tipo raster, suelen caracterizarse por estructuras de datos más abiertas e incluso por ser proyectos de software abierto como en el caso de GRASS. Los segundos se basan primordialmente en modelos de datos vectoriales y, por tener en general un carácter más comercial, son mucho más cerrados en todos los sentidos. Esta división no pretende en absoluto ser sistemática sino que responde más bien a la percepción de lo que el mercado ofrece.

Las estrategías de integración del análisis espacial con los Sistemas de Información Geográfica pueden variar desde una independencia casi completa a una integración total (Nyerges, 1992). Resulta por tanto fundamental que un SIG que vaya a utilizarse para análisis espacial o modelización de procesos sea lo más abierto posible, tanto por lo que se refiere a las estructuras de datos, como a los códigos fuente. De ese modo se incrementa la flexibilidad de la integración.

Destaca el proyecto GRASS, que se ha adherido a la filosofía GNU de software libre, por el desarrollo de herramientas totalmente abiertas tanto en cuanto sus estructuras de datos como a los programas y las librerias utilizadas.

Una de las críticas más habituales a GRASS es su aspecto poco atractivo para su utilización por el *gran público*. Esta objección que tambien ha experimentado ArcInfo, especialmente tras la popularización de ArcView. Esta falta de salidas gráficas *atractivas* se debe, en parte, al uso de un sistema gráfico simple que garantiza la portabilidad (Neteler, 2000).

El problema se ha solucionado parcialmente con el desarrollo de NVIZ, un programa desarrollado con el lenguaje de scripts TCL y la libreria TK+ (es decir fuera de C++ y de las librerias de GRASS). Sin embargo este nuevo módulo de GRASS, incorporado a partir de la versión 5 (aún en fase beta), sigue siendo un programa de visualización 3D fundamentalmente raster. Por otra parte no soluciona el problema de la falta de programas de *desktop mapping*. Al estar basarse en un lenguaje interpretado su ejecución es más lenta, siendo su manejo a traves de terminales bastante complejo. XWINDOWS

Otra de las causas de la ausencia de salidas gráficas atractivas era la falta de herramientas sencillas de programación gráfica en Unix, sin embargo la popularización de linux ha desarrollado, y en parte también se ha apoyado, en el desarrollo de interfaces de usuario agradables a partir de librerias gráficas orientadas a objetos. Destacan, por su relativa facilidad de uso las librerias GTK+ y GDK desarrolladas en el proyecto GNOME (Harlow, 1999; Wright, 2000). Se dispone además de un entorno agradable para la programación de interfaces gráficas de usuario, se trata del programa GLADE (Wright, 2000) que, con una interfaz de usuario muy similar a la de Visual Basic, permite generar la IGU del programa con gran sencillez.

Finalmente resaltar la necesidad de herramientas de análisis espacial similares a R. Este es un lenguaje y un entorno para desarrollar y ejecutar análisis estadísticos interactivos (Gentleman & Ihaka, 1998). Es similar en cuanto a su presentación al lenguaje S, desarrollado por los laboratorios de

AT&T Bell, aunque internamente es distinto. Su mayor ventaja respecto a otras herramientas estadísticas es, además de constituir un lenguaje de programación específtfico, su orientación a objetos. De esta manera los resultados de los análisis realizados son objetos a los que se pueden aplicar funciones miembro para consultar, representar gráficamente, etc.

2. Objetivos

Partiendo de estos presupuestos se ha abordado el desarrollo de una libreria de clases espaciales que permita crear herramientas de análisis espacial, modelización y SIG de la forma más abierta posible. Los objetivos que se esperan conseguir a partir de este proyecto pueden resumirse en:

1. Incrementar la potencialidad de GRASS en el manejo de datos vectoriales.
2. Desarrollo de herramientas SIG con salidas gráficas en Xwindows que aprovechen las librerias GTK+ y GDK
3. Permitir el desarrollo de herramientas de SIG, análisis espacial, modelización y GeoComputación a partir de unos conocimientos básicos de C++.
4. Implementación facilmente comprensible por usuarios de SIG. En este sentido destaca la posibilidad de utilizar estas librerias en cursos avanzados de SIG
5. Obtener un código lo más independiente posible de la plataforma utilizada para programar.

Desarrollar una herramienta de análisis espacial orientada a objetos en el mismo sentido que R es un programa de análisis estadístico orientado a objetos.

Permitir el desarrollo de herramientas abiertas de tipo *Desktop mapping*.

3. Objetos espaciales

En el desarrollo de un proyecto relacionado con información espacial basado en un Sistema de Información Geográfica, los diversos elementos que aparecen en el paisaje son representados mediante modelos de objetos espaciales (Molenaar, 1998) que retienen lo esencial de la información espacial eliminando aquellos elementos que no se consideran relevantes en función de los objetivos o la escala del trabajo. Habitualmente se considera una clasificación de estos objetos en:

- ✓ Objetos puntuales (Vértices geodésicos, edificios, puntos de muestreo)
- ✓ Objetos lineales (rios, carreteras)
- ✓ Polígonos (municipios, areas homogeneas en cuanto a suelo, tipo de roca, vegetación, etc.)

- ✓ Superficies Relacionadas con la noción de variable regionalizada (Matheron), se trata de variables cuantitativas que adquieren valores diferentes en puntos diferentes del espacio pero manteniendo una estructura de autocorrelación, es decir cierta suavidad (topografía, pendiente, precipitación).

En los textos de SIG se suele hacer especial enfasis en no confundir los objetos geográficos tal como se han presentado antes con los objetos en el sentio de programación orientada a objetos. Sin embargo parece razonable desarrollar, en un entorno de SIG, clases en el sentido de la POO que respondan a los objetos geográficos clásicos de la literatura de SIG. Clases entendidas como modelos de objetos que contienen tanto sus propiedades (forma) como las funciones que les afectan (procesos).

Resulta interesante la incorporación de estos objetos espaciales dentro de los presupuestos básicos de la Teoría General de Sistemas. Podemos considerar un sistemas como una colección de elementos relacionados entre si mediante una serie de transferencias de materia o energía.

La unificación de todos estos criterios procedentes de campos científicos dispares lleva al planteamiento de clase espacial como aquella que cumple las siguientes características:

1. Representa una clase de elementos espaciales y por tanto algunas de sus variables deben codificar la ubicación espacial.

2. Tambien deben estar representadas las variables cuantitativas o cualitativas que definen el objeto.

3. Las funciones deben incluir tanto las funciones típicas de gestión de un SIG como los procesos de transferencia entre los elementos.

4. Sería deseable un cierto nivel de abstracción y recurrencia en las clases es decir clases que contengan objetos de otras clases de nivel inferior como variables (por ejemplo la clase linea contiene elementos de la clase punto

5. El criterio anterior implica que el número de elementos de una clase de nivel inferior es variable y al mismo tiempo tienen una existencia independiente de la de la clase superior, por tanto resulta aconsejable la utilización de punteros.

El punto dos del listado anterior implica un dilema de dificil solución. Las variables no espaciales son evidentemente diferentes de un objeto lineal a otro (carreteras y rios por ejemplo). Una solución seria la creación de la clase rio y la clase carretera herederas la clase linea. Sin embargo ello supondria una multiplicación de clases sin solucionar realmente el problema, ya que el tipo de variables de la clase rio puede variar en función del tipo de rio, tipo de estudio, etc.

La solución adoptada ha sido la creación de una estructura jerárquica en las que unas clases se relacionana con otras mediante relaciones de herencia o de agrupación (figura 1). El nivel más básico lo constituyen las *clases geométricas* (*punto*, *linea*); de ellas derivan por herencia las *clases espaciales* derivadas de las clases geométricas (*observatorio*, *tramo*, *polígono*).

Como norma general la clase espacial añade a la clase geométrica un identificador, un puntero a valores *float* y un entero que representa el numero de valore en coma flotante que se asignan al objeto.

Como complemento se crean dos clases en principio independientes. La clase *database* que no es más que un puntero a una matriz de valores float y dos enteros que definen las filas y columnas de la matriz de datos. La clase *raster* incluye un puntero a una matriz de puntos, una estructura *límites* que contiene la metainformación sobre la capa.

Finalmente en cualquier estudio sobre datos espaciales se suele trabajar con conjuntos de objetos de la misma clase que suelen representarse y almacenarse juntos. Por ello se han creado las clase *mapa de lineas* (para manejar isolineas), *mapa de observatorios*, *mapa de tramos* y *mapa de poligonos* que agrupan objetos de las clases respectivas. Realmente, lo que estas clases contienen es un puntero a un vector de elementos. Esto permite que los objetos individuales tengan una *existencia* independiente de la de los objetos que los agrupan.

La existencia de objetos *mapas de elementos* y de objetos *bases de datos* permite leer independientemente los ficheros de objetos espaciales y las tablas con sus características y enlazarlos posteriormente.

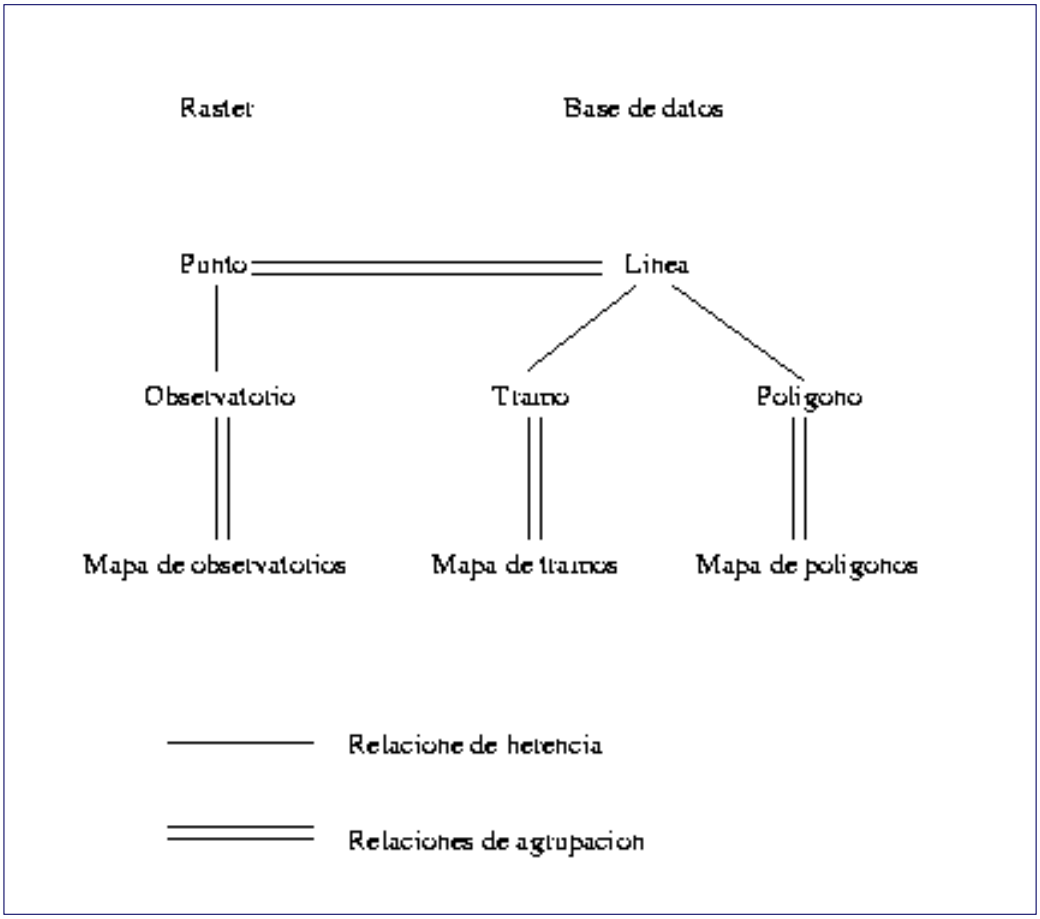

Figura 1. Esquema de clases utilizado

4. Gráficos

Cada una de las clases espaciales definidas previamente dispone de funciones miembro para su representación gráfica. Este tipo de funciones resultan más complejas de programar que cualquier otra en un sistema X-windows. Sin embargo el uso de las librerias GTK+ y GDK facilitan de forma considerable su uso.

5. Análisis de las diversas clases

Para mostrar como se ha construido esta librería se incluye su fichero *header* "clases_gis.h'' que muestra las 10 clases creadas asi como sus miembros (elementos y funciones miembro. El fichero "clases_gis.c'' contiene todas ellas excepto las que se refieren con la representación gráfica. Estas últimas se han almacenado en un fichero aparte "graficos.c'' para preservar la portabilidad en la medida de lo posible, ya que utilizan funciones de las librerias GTK+ GDK.

Se incluyen además una serie de funciones de diverso propósito, situadas al final del archivo que hacen uso de estas clases.

Puede verse claramente cómo el proyecto está todavía en fase de darrollo. Los módulos que aún están en fase de implementación aparecen marcados con el comentario //NYI.

FICHERO clases.h

```
#include <gdk/gdkkeysyms.h>
#include <gtk/gtk.h>


struct limites{float maxx;float maxy;float minx;float miny;int nrow;int
ncol;};

//CLASES

class punto{
  float x; float y;
 public:
  punto(float v1,float v2);
  punto(const punto &p);
  float get_x();
  float get_y();
  void set_x(float xx);
  void set_y(float yy);
  void show();
  void pinta(GtkWidget* d_a,GdkGC *pen,int relleno,int tam,struct limites
lim,int ven);
  float distancia(punto p);
```

```
  float angulo(punto p);
  punto operator=(punto p2);
};

class observatorio : public punto{
  int ide;float z; float *datos; int num;
 public:
  observatorio(float v1,float v2,float v3,float *dat,int n);
  observatorio(float v1,float v2,float v3);
  observatorio(float v1,float v2,float v3,int v4);
  observatorio(const observatorio &obs);
  void set(float v1,float v2,float v3,int v4);
  void linka(float *dat,int n);
  float get_dato(int i);
  int get_ide();
  float get_z();
  void set_z(float zz);
  int get_num();
  void set_num(int n);
  float* get_datos();
  void show();
  float get(int i);
  observatorio operator=(observatorio p2);
};

class linea{
  punto *elementos; int vertex; float z;char ide;
 public:
  linea(int n,float zz);
  linea(int n);
  linea(punto *el, int n);
  linea(const linea &line);
  ~linea();
  void set(int n);
  void set (char c);
  void set(int n,float x, float y);
  void set(punto *p);
  void set(float zz);
  punto get(int n);
  int get_vertex();
  char get_ide();
  float get_z();
  punto *get_pointer();
  void pinta(GtkWidget *d_a,GdkGC *gc, struct limites lim,int ven);  //NYI
  void show();
  float longitud();
  float longrect();
  linea operator=(linea l2);
};

class tramo: public linea{
  int ide; float *datos; int num;
 public:
  tramo(punto *el,int v, int id);
  void linka(float *dat,int n);
  float get_dato(int i);
  int get_ide();
  int get_num();
  void set_num(int n);
  float* get_datos();
```

```
};

class poligono: public linea{
  int ide; float *datos; int num;
 public:
  poligono(punto *el,int v, int id);
  void linka(float *dat,int n);
  float area();                                  //NYI
  float get_dato(int i);
  int get_ide();
  int get_num();
  void set_num(int n);
  float* get_datos();
  void pintapol();                                    //NYI
};

class mapa_lineas{
  linea *lines;  int numlines; char identif[10];
 public:
  mapa_lineas(int n);
  mapa_lineas(const mapa_lineas &ml);
  //mapa_lineas::~mapa_lineas();
  char *get_identif();
  void set(int n);
  void set(int n,linea *lin);
  void set(linea *l);
  linea *get(int n);
  punto get(int n1, int n2);
  int get_num();
  struct limites get_lims();
  void pinta(GtkWidget *d_a, struct limites lim,int ven);
  void leevectorgrass(char *iden,char *namefic);
  mapa_lineas operator=(mapa_lineas ml2);
};

class mapa_observatorios{
  observatorio *obs; int numobs;char identif[10];
 public:
  mapa_observatorios(int n);
  mapa_observatorios(const mapa_observatorios &mo);
  mapa_observatorios(char *iden,char *namefic);
  //mapa_observatorios::~mapa_observatorios();
  void leemapa(char *iden,char *namefic);
  struct limites get_lims();
  char *get_identif();
  void set(int n,observatorio ob);
  void set(observatorio *ob);
  void set(int n);
  observatorio *get(int o);
  int get_num();
  void pinta(GtkWidget *d_a, struct limites lim,int col,int ven);
  mapa_observatorios operator=(mapa_observatorios mo2);
  observatorio* what(float xx, float yy);
};

class mapa_tramos{          //CNYI
  tramo *lines;  int numtramos; char identif[10];
 public:
  mapa_tramos(int n);
  mapa_tramos(const mapa_lineas &ml);
```

```
  //mapa_tramos::~mapa_lineas();
  char *get_identif();
  void set(int n);
  void set(int n,linea *lin);
  void set(linea *l);
  linea *get(int n);
  punto get(int n1, int n2);
  int get_num();
  struct limites get_lims();
  void pinta(GtkWidget *d_a, struct limites lim,int ven);
  mapa_tramos operator=(mapa_lineas ml2);
};


class mapa_poligonos{         //CNYI
  poligono *poligonos;  int numpoligonoss; char identif[10];
 public:
  mapa_poligonos(int n);
  mapa_poligonos(const mapa_poligonos &ml);
  //mapa_lineas::~mapa_lineas();
  char *get_identif();
  void set(int n);
  void set(int n,linea *lin);
  void set(linea *l);
  linea *get(int n);
  punto get(int n1, int n2);
  int get_num();
  struct limites get_lims();
  void pinta(GtkWidget *d_a, struct limites lim,int ven);
  mapa_lineas operator=(mapa_lineas ml2);
};


class database{
  float *datos; int ncol;int nfil;char identif[10];
 public:
  database(int f, int c);
  database(int f, int c,float *dat);
  database(char *namefic);
  void leebase(char *iden,char *namefic);
  char *get_identif();
  void set(int f,int c,float dat);
  float get(int f, int c);
  void set_ide(int f,int valor);
  int get_ide(int f);
  void set_fc(int f, int c);
  int get_f();
  int get_c();
  float *get_datos(int i);
};

class raster{
  float *datos; float maxz; float minz;struct limites lim;char identif[10];
 public:
  raster(struct limites l);
  raster(struct limites l, float *dat);
  raster(int n);
  char *get_identif();
  void set_lims(float x1,float x2,float y1,float y2, int r,int c);
  void set_lims(struct limites l);
```

```
  void set_datos(float *dat);
  void set_dato(int f,int c,float z);
  void scalar(int op,float num);
  void booli(int op,float ref,float num);
  float get_dato(int f, int c);
  struct limites get_lims();
  void set_range();
  void pinta(gpointer d_a, struct limites lim,int ven);
  void grabagrass(char *namedir,char *namefic,int conv);
  void leegrass(char *identif,char *namedir, char *namefic);
};

//Funciones de manejo de angulos
float deg2rad(float x);
float rad2deg(float x);
float difang(float ang1,float ang2);
float dxdy2deg(float dx,float dy);
void deg2dxdy(float ang,float d,float *dx,float *dy);


//Funciones de lectura escritura de ficheros
void set_lim(struct limites *l,float mnx,float mxx,float mny,float mxy);
void set_lim(struct limites *l, raster mapa);
void set_lim(struct limites *l, mapa_lineas mapa);
void set_lim(struct limites *l, mapa_observatorios mapa);
short unsigned int grass2gcc(short unsigned int PA);
//short unsigned int *leeraster2b(char *namefic);
void leegrass(char *namedir, char *namefic,raster *mapa);
void leesurfer(char *namefic,raster *mapa);

void leevectorgrass(char *namefic, mapa_lineas *mapa);

void linkaobs(database *base, mapa_observatorios *mapa);


//Rasterizacion-vectorizacion
void rasteriza (float c1, float f1, float c2, float f2,int *fpoint, int
*cpoint,int *totp, struct limites lim);
int x2c(float x,struct limites lim);
int y2f(float x,struct limites lim);
float f2y(int c,struct limites lim);
float c2x(int c,struct limites lim);


//More complex stuff
planoeq(double aspect,double slope, struct triada *pl);

}
```

6. Resultados discusión y conclusiones

Puesto que todavía está en fase de desarrollo, aún faltan métodos por implementar y la compilación produce abundantes mensajes de advertencia que deben ser depurados. Al mismo tiempo el esquema conceptual de la libreria no está totalmente establecido sino que se mantiene abierto a las necesidades o sugerencias que puedan surgir.

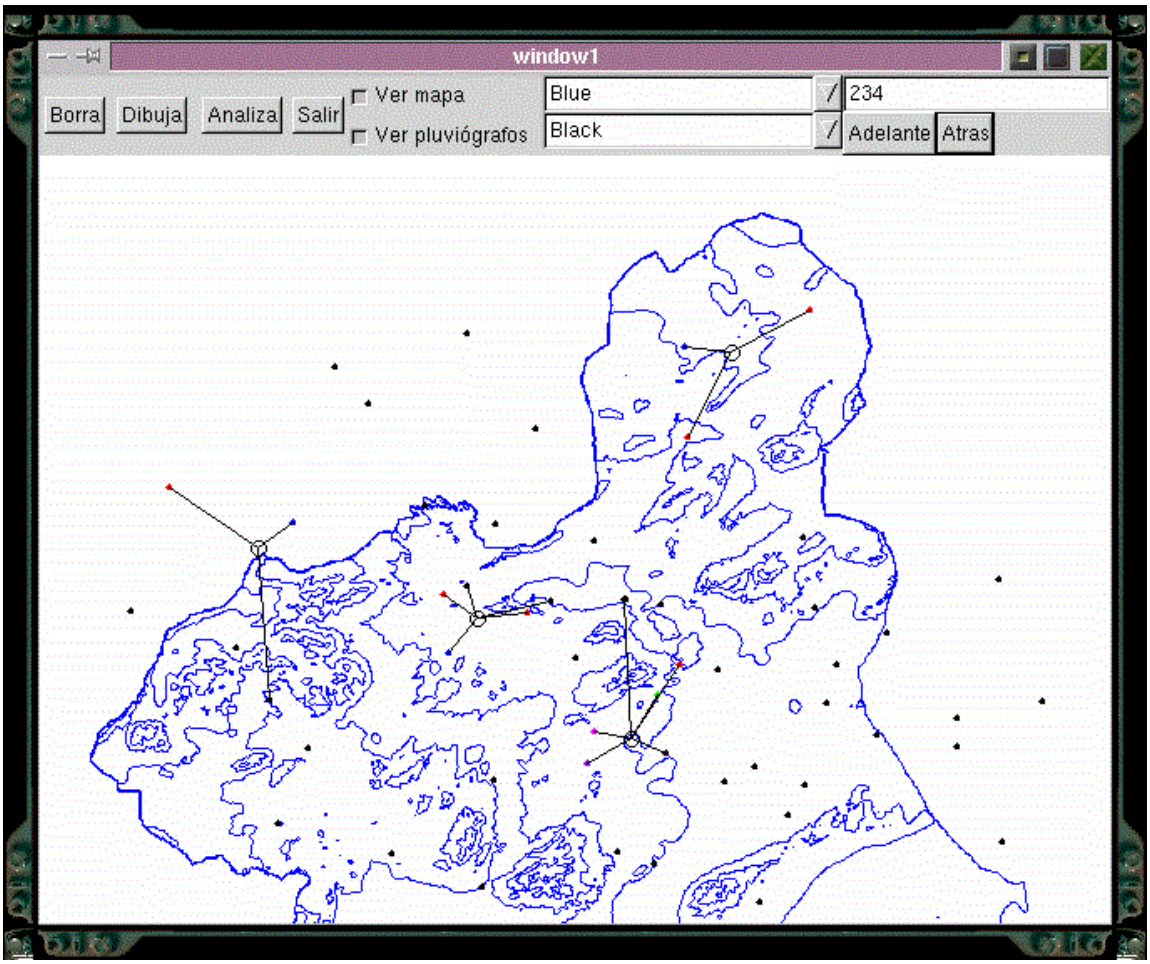

Figura 2. Pantalla del programa de análisis de episodios de precipitación

Sin embargo se han desarrollado algunos programas de forma experimental. En primer lugar un simple programa para verificar el buen funcionamiento de la libreria. Este programa permitió al mismo tiempo comprobar como la complejidad del código que ha de añadirse resulta considerablemente disminuida. En segundo lugar un programa para el análisis espacio-temporal de episodios de precipitación del que se presenta una pantalla en la figura 2. Este programa (Alonso Sarría, 2000) se muestra en esta misma reunión

La interfaz gráfica de usuario de ambos programas se llevó a cabo utilizando el programa *glade* mencionado anteriormente.

Se ha comprobado como a partir de esta libreria resulta sencillo programar herramientas para el análisis exploratorio de datos espaciales.

Entre las tareas que resulta necesario abordar en el futuro, destacan la integración con las librerias de GRASS,

7. Localización de las herramientas utilizadas

- ✓ Glade: http://glade.pn.org/
- ✓ GNU: http://www.gnu.org/
- ✓ GRASS: http://www.geog.uni-hannover.de/grass/index2.html
- ✓ GTK+: http://www.gtk.org/
- ✓ Linux: http://www.linux.org/
- ✓ R: http://www.stat.auckland.ac.nz/r/r.html

8. Bibliografía

Alonso Sarría, F. (2000) Desarrollo de un programa para el analisis espacio-temporal de episodios de precipitación *IX Congreso del grupo de métodos cuantitativos, sistemas de información geográfica y teledetección.*

Anderson,M.G. (1988) *Modelling Geomorphological Systems* John Willey & sons. Chichester.

Bailey,T.C. & Gatrell,A.C. (1995) *Interactive Spatial Data Analysis*. Longman. New York.

Chorley,R.J. (1972) *Spatial Analysis in Geomorphology* Methuen & co. Ltd. London

Couclelis,H. (1998) Geocomputation in context en P.A.Longley *et al.* (eds.) *Geocomputation: A primer*. John Willey&sons. Chichester: 17-30.

Davis,J.C. & McCullagh,M.J. (1975) *Display and Analysis of Spatial Data* John Willey&sons. London.

Fotheringham,A.S. (2000) GIS-based Spatial modelling: A step Forwards or a Step Backwards? en A.S.Fotheringham & M.Wegener (Eds.) *Spatial Models and GIS. New potential and New Models*. Taylor & Francis Group. London:3-20

Fotheringham,A.S. & Rogerson,P.R. (eds.) (1994) *Spatial Analysis and GIS*. Taylor & Francis Group. London.

Gámir Orueta,A.; Ruiz Pérez,M.; Seguí Pons,J.M. (1995) *Prácticas de análisis espacial* Oikos-tau. Barcelona.

Getis,A. & Boots,B. (1978) *Models of Spatial Processes* Cambridge University Press. Cambridge.

Harlow,E. (1999): *Desarrollo de aplicaciones Linux con GTK+ y GDK*. Prentice Hall. Madrid.

Horton,R.E. (1932) Drainage-basin characteristics, *EOS Trans. AGU*, 13:350-361.

Mather,P.M. (ed.) (1993) *Geographical information Handling: Research Applications*
John Willey&sons. Chichester.

Molenaar,M. (1998) *An introduction to the Theory os Spatial Object Modelling for GIS*. Taylor & Francis Group. London.

Neteler,M. (2000) *GRASS 5.0 Programmer Manual* http://hgeo02.geog.uni-hannover.de/grass/grass421/manuals/prgman42.pdf

Nyerges,T.L. (1992) Coupling GIS ans spatial analytic models en P.Bresnahan, E.Corwin&D.J.Cowan (eds.) *Procedings of the Fifth International Symposium on Spatial Data Handling*. Columbia,SC: University of South California: 534-542

Openshaw,S. (2000) GeoComputation en S.Openshaw&R.J.Abrahart (Eds.) *GeoComputation* Taylor & Francis Group, London 1:32

Rodriguez Iturbe,I. (1993) The Geomorphologic Unit Hydrograph en K.Beven&M.J. Kirkby (eds.) *Channel Netword Hydrology* John Willey&sons. New York: 226-241

Rodriguez Iturbe,I. & Rinaldo,A. (1997) *Fractal River Basins. Chance and Self Organization*. Cambridge University Press. Cambridge.

Strahler,A.N. (1957) Quantitative analysis of watershed geomorphology, *EOS Trans. AGU* 38:912-920

Wegener,M. (2000): Spatial models and GIS. en A.S.Fotheringham & M.Wegener (Eds.) *Spatial Models and GIS. New potential and New Models*. Taylor & Francis Group, London 3:20

Wrigth,P. (2000) *Beginning GTK+/GNOME Programming*. Wrox Press. Birmingham

Este trabajo se desarrolló durante una estancia postdoctoral en el Departamento de Geografía del King's College University of London.

Este trabajo se escribió originalmente en LaTex y se convirtió a RTF con Tex2RTF (http://web.ukonline.co.uk/julian.smart/tex2rtf)