CONSTRUCCIÓN DE UN VISOR DE SHAPEFILES CON HERRAMIENTAS LIBRES
Quantum GIS, Qt y Python

Germán Alonso Carrillo Romero
geotux_tuxman@linuxmail.org
http://geotux.tuxfamily.org

Introducción

Con el ánimo de construir mi propio visor de datos espaciales en formato vectorial, me di a la tarea de buscar herramientas de *software* que me lo permitieran de la manera más libre posible.

En este artículo pretendo mostrar cómo construir un visor de datos vectoriales en formato *Shapefile* con algunas herramientas básicas de navegación empleando *software* libre.

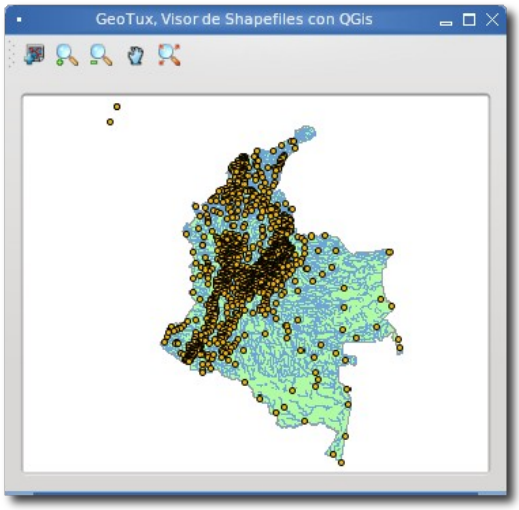

Figura 1. Visor de Shapefiles a realizar.

¿Para qué tener mi propio visor?

- Para poder acceder rápidamente a mis datos espaciales, sin tener que cargar los programas usuales, que pueden consumir recursos innecesarios en operaciones simples de consulta.

- Para aprender a emplear y a integrar las facilidades que presentan muchos de los proyectos de filosofía libre, demostrando así que el principal obstáculo en la implementación de estas herramientas es nuestro desconocimiento y no sus limitaciones funcionales.

¿Qué tipo de aplicación se va a generar?

Según Daniel P. Ames, quien está a cargo del desarrollo de componentes de Sistemas de Información Geográfica (SIG*)* en el proyecto *MapWindow*[1]*,* existen tres tipos de escenarios para desarrollar *software* SIG:

1. Desarrollo de extensiones y *plug-ins* que agregan funcionalidades a *software* SIG existente en el escritorio.
2. Desarrollo de herramientas *web* de visualización y generación de mapas.
3. Desarrollo de aplicaciones independientes en el escritorio empleando componentes SIG programables.

Nuestro visor de datos espaciales en formato vectorial será una aplicación independiente en el escritorio, empleando las librerías del proyecto *Quantum GIS*[2] que servirán como componentes SIG programables.

¿Qué herramientas de software se requieren?

- Sistemas Operativo: *GNU/Linux* (Con algunos ajustes puede emplearse en *Windows XP* o *Mac OS X*)
- Lenguaje de Programación: *Python* (Licencia: Compatible con *GNU/GPL*)
- Marco Multiplataforma de Desarrollo de Aplicaciones: *Qt* 4.4. (Licencia[3]: Dual, *GNU/GPL* y Comercial) y su herramienta gráfica *Qt Designer*.
- Python Bindings: *PyQGIS* (Licencia: *GNU/GPL*) y *PyQt4* (Licencia: Dual, *GNU/GPL* y Comercial).
- Librerías SIG: *Quantum GIS* (Licencia: *GNU/GPL*)

Python

Python es un lenguaje de programación interpretado, multiplataforma y orientado a objetos creado a principios de los años 90. Se caracteriza por tener una sintaxis clara (que da buena legibilidad al código), por tener tipado dinámico (no es necesario declarar el tipo de las variables) y por ser fuertemente tipado (no se puede tratar una variable como si fuera de un tipo distinto al que tiene).

Python es un lenguaje muy potente que ha incursionado en muchos campos, como por ejemplo el de los programas SIG, dotando de sencillez el desarrollo de funcionalidades y *plug-ins*.

Los lenguajes de programación interpretados son más lentos que los compilados debido a que un intérprete debe traducir el *script* cada vez que este se ejecuta. Sin embargo, *Python* maneja un mecanismo similar al de *Java*, pasando el código fuente a un código intermedio para que este sea el que se ejecute ante cualquier llamado posterior.[4]

En febrero de 2009 se lanzó la versión 3.0 de Python. Esta versión es incompatible hacia atrás por lo que los desarrollos con las versiones 2.x no podrán correr en la nueva. Cuando sea oportuno hacerlo (ahora no lo es pues las librerías necesarias no están migradas) actualizaré este artículo para que sea compatible del todo con la versión 3.0. En todo caso, eso será más adelante pues el proceso de migración del andamiaje de librerías puede tomar meses.

Qt y Qt Designer

Qt es un marco multiplataforma para el desarrollo de aplicaciones. Está escrito en *C++*. Se caracteriza porque permite que los proyectos compilados sean ejecutados en diversas plataformas como *GNU/Linux*, *Mac OS X*, *Windows* y *Windows CE*. *Qt* provee clases para el manejo de elementos de interfaz gráfica de usuario que son utilizados por múltiples aplicaciones, como por ejemplo, el virtualizador *VirtualBox*.

Qt Designer es una herramienta para diseñar y construir interfaces gráficas de usuario (*GUI*) a partir de componentes de *Qt*.

En marzo de 2009 se lanzó la versión 4.5 de Qt. Cuando sea oportuno actualizaré este documento para que funcione con la nueva versión.

PyQGIS y PyQt

Una de las grandes potencialidades de *Python* es su capacidad para tomar librerías existentes, escritas en *C* y *C++*, y hacerlas disponibles como módulos de extensión. Estos módulos de extensión son conocidos como *Bindings* para la librería[5].

Por ejemplo, *Quantum GIS* está escrito en *C++,* pero provee los *bindings* (*PyQGIS*) que permiten desarrollar aplicaciones independientes de escritorio y *plug-ins* para *Quantum GIS* empleando *Python*.

Para el desarrollo del visor de *Shapefiles* se emplearán los siguientes *bindings*:

- **PyQt4:** *Bindings* para *Qt4*. Permiten hacer uso de gran parte de las clases y objetos de interfaz de usuario del proyecto *Qt4* desde *Python*. Tiene licencia dual, como Qt, es decir, tiene una versión GPL y una versión comercial que permite cerrar los desarrollos.
- **PyQGIS**: *Bindings* para *QGIS*. Los provee el proyecto *Quantum GIS* desde su versión 0.9 en septiembre de 2007. Según *Martin Dobias*, desarrollador del mencionado programa para SIG, *PyQGIS* es un 99% idéntico a la *Interfaz de Programación de Aplicaciones* (*API)* en *C++. PyQGIS* está incorporado en el instalador de *QGIS* 1.0.

La API de Quantum GIS

El proyecto *Quantum GIS* no solo se ha enfocado en proveer una aplicación de escritorio para SIG, sino que también pretende disponer librerías SIG para ser empleadas en la elaboración de aplicaciones independientes en el escritorio, como es el caso de este visor de *Shapefiles,* y en el desarrollo de *plug-ins*.

Las librerías SIG de *Quantum GIS* son un conjunto de clases en *C++* que permiten acceder y manipular los objetos espaciales que un programa SIG necesita. Las librerías son:

- **Core** (Núcleo): Contiene las funcionalidades SIG.
- **Gui**: Está construida sobre la librería *Core*. Contiene controles reusables para la interfaz de usuario como por ejemplo el *Map Canvas*, la región en la cual se despliega y manipula el mapa.
- **MapComposer** (Generación de Mapas): Contiene funcionalidades para generar salidas gráficas.

Quantum GIS provee desde la versión 1.0, lanzada en diciembre de 2008, una *API* estable y compatible hacia atrás, esto implica que los desarrolladores tienen la certeza que al emplear la versión 1.0 no tendrán problemas de compatibilidad con futuros lanzamientos.

Para comenzar...

Para comenzar la construcción del visor, debemos instalar el *software* requerido. En este documento mencionamos la manera de instalar el *software* en la distribución *Ubuntu Linux*, para otras distribuciones puede hacerse uso de comandos similares o del gestor de descargas correspondiente. Para *Windows* puede utilizarse el paquete *OSGeo4W*[6] y hacer uso del instalador oficial de *PyQt4*[7].

Instalación en Ubuntu Linux

Esta instalación se lleva a cabo en *Ubuntu Linux* versión *Jaunty* (9.04), en todo caso puede servirte si tienes una versión más antigua, solo cambias el *jaunty* por *intrepid* (8.10)*, hardy* (8.04) o *gutsy* (7.10) en el primer paso.

1. Agregar al archivo /etc/apt/sources.list los repositorios para descargar QGIS:

```
deb http://ppa.launchpad.net/qgis/unstable/ubuntu jaunty main
deb-src http://ppa.launchpad.net/qgis/unstable/ubuntu jaunty main
```

2. Actualizar la lista de paquetes del gestor de descargas (desde la terminal):

```
sudo apt-get update
```

3. Instalar los programas necesarios (desde la terminal):

```
sudo apt-get install pyqt4-dev-tools python python-qt4 qt4-designer qgis
```

Ahora tenemos instalado el software que necesitamos, el siguiente paso es crear una carpeta en nuestro disco duro para guardar allí los archivos del visor de *Shapefiles*. Por ejemplo:

```
/home/german/visor_shapefiles_qgis/
```

Esta será la carpeta en la que almacenaremos todos los archivos que vamos a crear a continuación.

Probando las librerías

Antes de iniciar es bueno asegurarnos que podemos usar las librerías de *PyQt4* y *PyQGIS* en *Python* o de lo contrario podemos tener algunos percances.

Abrimos una terminal para fijar una variable de entorno (*PYTHONPATH*) que le diga a *Python* dónde encontrar los *bindings* de *QGIS*. Tecleamos en la terminal el siguiente comando:

```
export PYTHONPATH=/usr/share/qgis/python
```

Donde `/usr/share/qgis/python` es la ruta a los *bindings* de *QGIS*. Esta es la carpeta en donde se instalan por defecto pero pueden encontrarse en otra, dependiendo del proceso de instalación.

En *Windows* se debe usar el comando *set* desde una terminal, así:

```
set PYTHONPATH="C:\Archivos de programa\qgis\python"
```

O su correspondiente ruta.

Después de definir la variable de entorno *PYTHONPATH*, abrimos una consola de *Python* y tecleamos las siguientes líneas:

```
import PyQt4.QtCore
import PyQt4.QtGui
import qgis.core
import qgis.gui
```

Debemos obtener algo como esto:

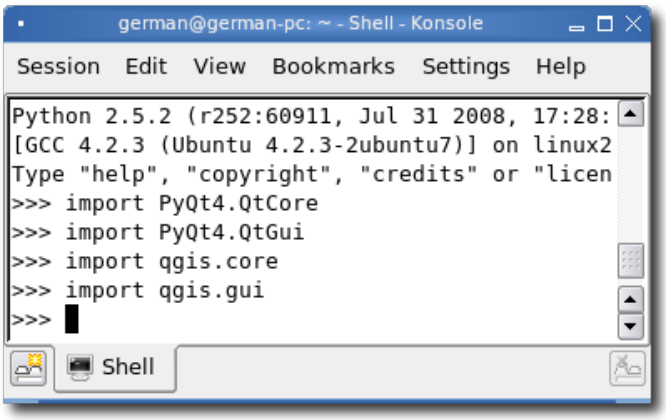

Figura 2. Consola de Python.

Si se obtienen errores es posible que la variable de entorno *PYTHONPATH* no esté correctamente definida por lo cual tendríamos que revisar la ruta a los *bindings* de *QGIS*.

Ahora podemos proceder a crear la interfaz para el visor de *Shapefiles*.

Creando la interfaz del visor

Para crear la interfaz del visor de *Shapefiles* utilizaremos el programa *Qt Designer*. Para abrir el programa podemos teclear [Alt + F2] y escribir *designer*. Se abre un diálogo para elegir el tipo de proyecto a diseñar. Elegimos *Main Form* (Forma Principal) y damos *click* en el botón *Create*.

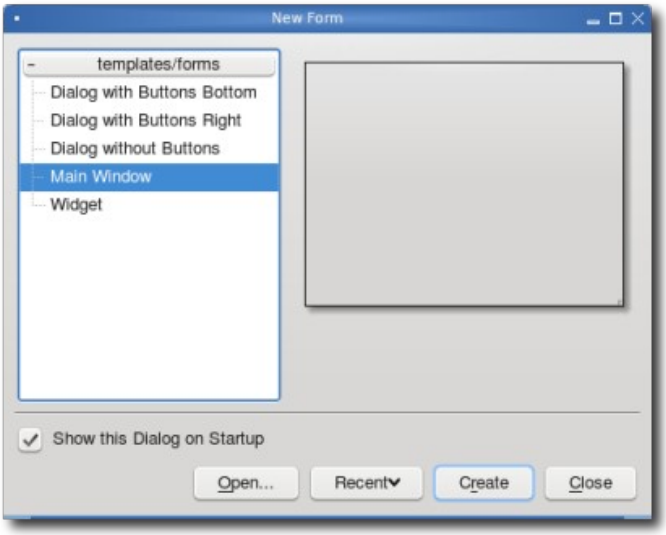

Figura 3. Diálogo New Form.

Si el diálogo *New Form* no se abre de forma automática podemos teclear [Ctrl + N] para desplegarlo.

Qt designer tiene por defecto una interfaz basada en múltiples ventanas, si lo queremos, podemos cambiar a una interfaz de una sola ventana para facilitar la visualización. Para ello vamos al menú *Edit*, a la opción *Preferences* y en la ventana que se abre seleccionamos *Docked Window* como *Modo de Interfaz de Usuario*.

En la sección de herramientas (en la parte izquierda de la interfaz) vamos al panel *Containers* y arrastramos un marco (*Frame*) a la ventana en donde aparece la forma principal:

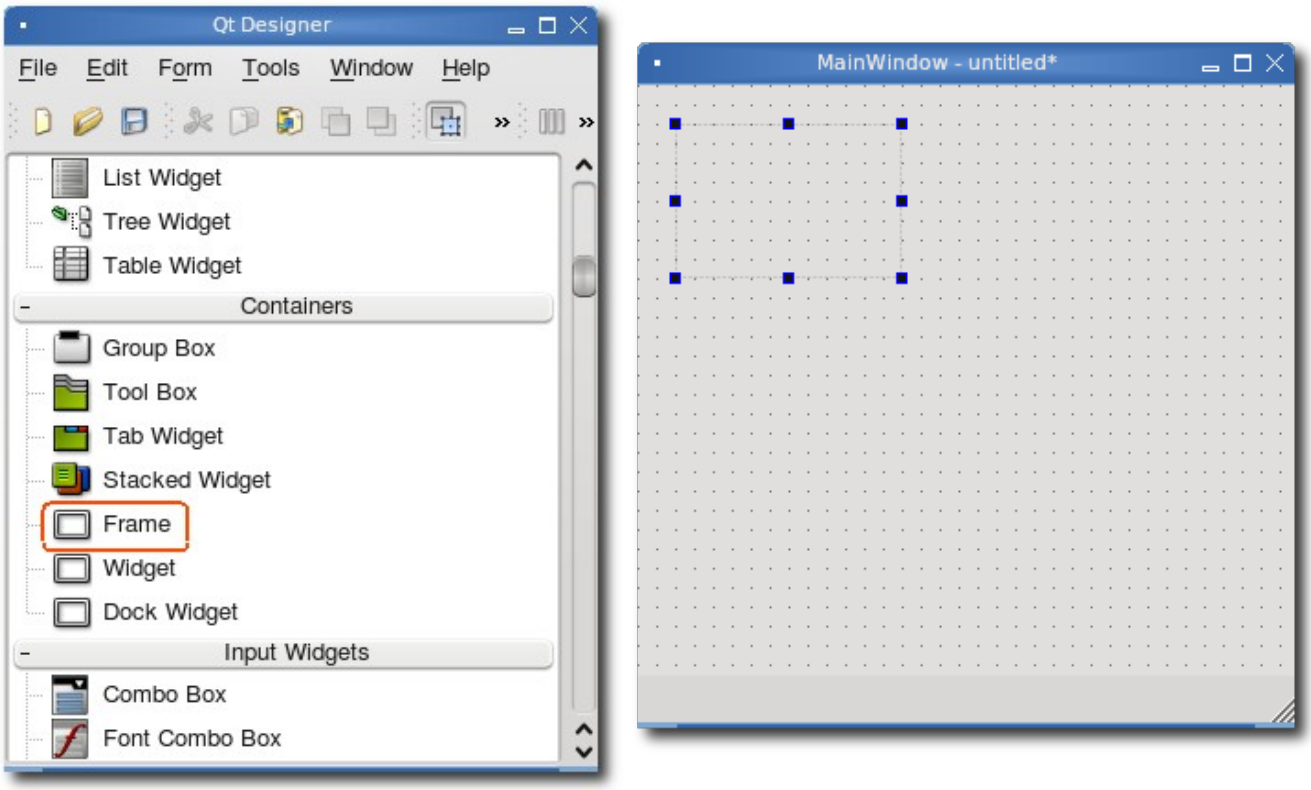

Figura 4. Panel *Containers* en *Qt designer* (Izquierda).
Forma principal con marco(Derecha).

El marco (*Frame*) es el lugar de la interfaz en donde estará el mapa.

Damos *click* en algún lugar de la ventana *Forma Principal* para quitar la selección del marco agregado y tecleamos [Ctrl + F5] para disponer los controles en una grilla (*Lay Out in a grid*), con esto, el marco se ajusta automáticamente a la extensión de la ventana.

Guardamos el archivo con el nombre `visor_shapefiles.ui` en la carpeta `/home/german/visor_shapefiles_qgis`

El archivo guardado está compuesto de etiquetas. Sin embargo, necesitamos un archivo de python que se encargue de construir la interfaz. Para ello existe la herramienta *pyuic4*, de *PyQt4*. Ejecutamos el siguiente comando en una terminal desde la carpeta creada para el visor:

```
pyuic4 -o visor_shapefiles_ui.py visor_shapefiles.ui
```

Con esto se crea el archivo `visor_shapefiles_ui.py`

Escribiendo el código del visor

Ahora estamos listos para escribir el código del visor de *Shapefiles* en *Python*.

Es importante conocer que en *Python* el indentado debe respetarse, algunos problemas con la ejecución del código pueden surgir si no se presta atención a este punto. Existen convenciones sobre el número de espacios para indentar en Python, se sugiere que sean cuatro espacios por nivel de indentación.

Algunas recomendaciones para escribir código en *Python* pueden consultarse en la traducción[8] de Raul González Duque del documento *Guía de Estilo del Código Python* de Guido van Rossum, creador de *Python*.

Para empezar con el código del visor de *Shapefiles* se deben importar las librerías necesarias, entre ellas la clase *Ui_MainWindow* que se encuentra en el módulo creado por el comando *pyuic4* (línea 9) :

```
1  import sys
2
3  from PyQt4.QtCore import *
4  from PyQt4.QtGui import *
5
6  from qgis.core import *
7  from qgis.gui import *
8
9  from visor_shapefiles_ui import Ui_MainWindow
10
```

Definimos una variable global que sirve para determinar el directorio de instalación de *Quantum GIS*. En *Ubuntu Linux* es `/usr` o `/usr/local`.

```
11  qgis_prefix = "/usr"
```

Definimos la clase *VisorShapefiles* que hereda de *QMainWindow* y *Ui_MainWindow* y definimos su método *__init__*. En el código podemos ver comentarios que nos explican en detalle las acciones que se realizan.

```
12
13 class VisorShapefiles(QMainWindow, Ui_MainWindow):
14
15     def __init__(self):
16         QMainWindow.__init__(self)
17
18         # Requerido por Qt4 para inicializar la UI
19         self.setupUi(self)
20
21         # Fijar el titulo
22         self.setWindowTitle("GeoTux, Visor de Shapefiles con QGis")
23
24         # Crear el Map Canvas
25         self.canvas = QgsMapCanvas()
26         self.canvas.setCanvasColor(QColor(255,255,255))
27         self.canvas.enableAntiAliasing(True)
28         self.canvas.useImageToRender(False)
29         self.canvas.show()
30
31         # Agregar el Map Canvas a la ventana principal, en el marco
32         self.layout = QVBoxLayout(self.frame)
33         self.layout.addWidget(self.canvas)
34
35         # Crear los comportamientos de los botones. Cuando el botón
36         #    se activa, se llama  el método apropiado de la clase
37         self.actionAddLayer = QAction(QIcon(qgis_prefix +
38             '/share/qgis/themes/classic/mActionAddOgrLayer.png'),
39             "Agregar capa", self.frame)
40         self.connect(self.actionAddLayer, SIGNAL("activated()"),
41             self.addLayer)
42
43         self.actionZoomIn = QAction(QIcon(qgis_prefix +
44             '/share/qgis/themes/classic/mActionZoomIn.png'),
45             "Acercar", self.frame)
46         self.connect(self.actionZoomIn, SIGNAL("activated()"),
47             self.zoomIn)
48
49         self.actionZoomOut = QAction(QIcon(qgis_prefix +
50             '/share/qgis/themes/classic/mActionZoomOut.png'),
51             "Alejar", self.frame)
52         self.connect(self.actionZoomOut, SIGNAL("activated()"),
53             self.zoomOut)
54
55         self.actionPan = QAction(QIcon(qgis_prefix +
56             '/share/qgis/themes/classic/mActionPan.png'), "Panear",
57             self.frame)
58         self.connect(self.actionPan, SIGNAL("activated()"), self.pan)
59
60         self.actionZoomFull = QAction(QIcon(qgis_prefix +
61             '/share/qgis/themes/classic/mActionZoomFullExtent.png'),
62             "Vista completa", self.frame)
63         self.connect(self.actionZoomFull, SIGNAL("activated()"),
64             self.zoomFull)
```

```
65
66          # Crear una toolbar y agregar los botones
67          self.toolbar = self.addToolBar("Mapa")
68          self.toolbar.addAction(self.actionAddLayer);
69          self.toolbar.addAction(self.actionZoomIn);
70          self.toolbar.addAction(self.actionZoomOut);
71          self.toolbar.addAction(self.actionPan);
72          self.toolbar.addAction(self.actionZoomFull);
73
74          # Crear las herramientas (tools) para el mapa
75          self.toolPan = QgsMapToolPan(self.canvas)
76          self.toolZoomIn = QgsMapToolZoom(self.canvas, False)
77          self.toolZoomOut = QgsMapToolZoom(self.canvas, True)
78
79          # Lista de capas del Map Canvas
80          self.layers = []
```

Definimos los métodos que utiliza cada botón de la *Toolbar* creada.

```
81
82      def zoomIn(self):
83          self.canvas.setMapTool(self.toolZoomIn)
84
85      def zoomOut(self):
86          self.canvas.setMapTool(self.toolZoomOut)
87
88      def pan(self):
89          self.canvas.setMapTool(self.toolPan)
90
91      def zoomFull(self):
92           self.canvas.zoomToFullExtent()
93
94      def addLayer(self):
95          # Definir el proveedor de los datos vectoriales (ogr)
96          layerPath = QFileDialog.getOpenFileName(self,
97              "Abrir shapefile", ".", "Shapefiles (*.shp)")
98          layerInfo = QFileInfo(layerPath)
99          layerProvider = "ogr"
100
101          # Crear el layer
102          layer = QgsVectorLayer(layerPath, layerInfo.fileName(),
103              layerProvider)
104
105          if not layer.isValid():
106              return
107
108          #Cambiar el color del layer
109          symbols = layer.renderer().symbols()
110          symbol = symbols[0]
111          symbol.setFillColor(QColor.fromRgb(192,192,192))
```

```
112
113         # Agregar el layer al registro
114         QgsMapLayerRegistry.instance().addMapLayer(layer);
115
116         # Fijar el extent al extent del primer layer cargado
117         if self.canvas.layerCount() == 0:
118             self.canvas.setExtent(layer.extent())
119
120         # Fijar el conjunto de capas (LayerSet) para el map canvas
121         self.layers.insert(0, QgsMapCanvasLayer(layer))
122         self.canvas.setLayerSet(self.layers)
```

Ahora definimos el método *main* del módulo. Este método es el que se ejecuta para poder ver la aplicación. Finalmente se controla que el módulo solo se ejecute cuando es llamado como *script* y no cuando es importado como módulo (línea 145).

```
123
124 def main(argv):
125
126     app = QApplication(argv)
127
128     # Inicializar las librerías de QGIS
129     QgsApplication.setPrefixPath(qgis_prefix, True)
130     QgsApplication.initQgis()
131
132     # Crear y mostrar la ventana principal
133     wnd = VisorShapefiles()
134     wnd.move(100,100)
135     wnd.show()
136
137     # Ejecutar un loop para correr el programa
138     retval = app.exec_()
139
140     # Salir
141     QgsApplication.exitQgis()
142     sys.exit(retval)
143
144
145 if __name__ == "__main__":
146     # Llamar el método principal que se encarga de crear la aplicación
147     main(sys.argv)
```

Guardamos el archivo como `VisorShapefiles.py` en la carpeta creada con anterioridad, la cual debe contener, como mínimo los archivos `VisorShapefiles.py` y `visor_shapefiles_ui.py`. Ahora podemos ejecutar la aplicación.

Ejecutando la aplicación

Después de haber completado los pasos anteriores con cierta dosis de fe, pues hasta este punto no hemos visto nada que se nos parezca a un mapa, podemos abrir el visor de Shapefiles ejecutando el siguiente comando desde la terminal de *Ubuntu Linux*:

```
python /home/german/visor_shapefiles_qgis/VisorShapefiles.py
```

El resultado es similar al siguiente:

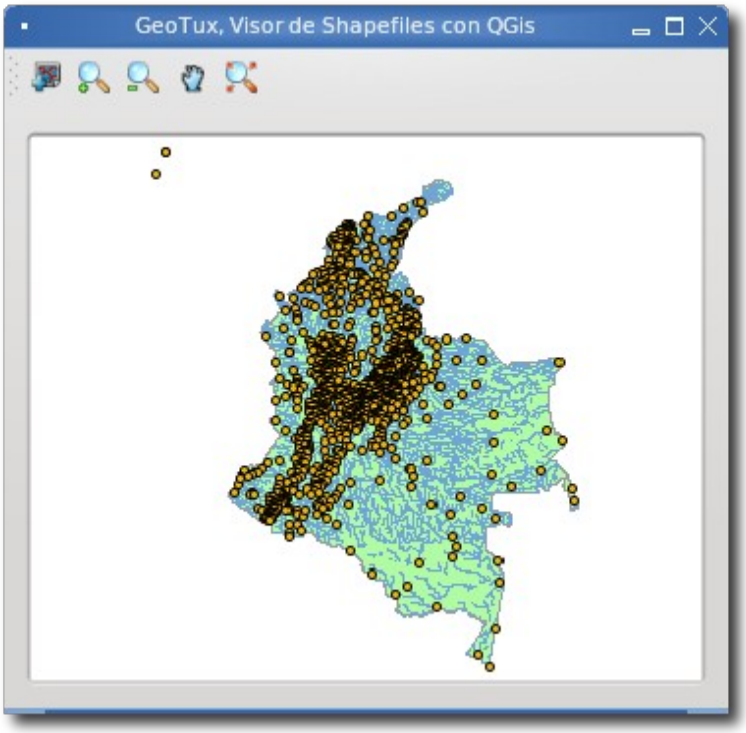

Figura 5. Visor de Shapefiles.

Archivos para descargar

En un principio el indentado puede traer algunos problemas. Para evitarlos podemos descargar el archivo `VisorShapefiles.py` y lo guardamos en la carpeta de la aplicación. El archivo puede descargarse desde aquí.

Podemos descargar un *Shapefile* de prueba aquí (archivo comprimido, 1.3 MB).

Conclusiones

Hemos generado nuestro propio visor de *Shapefiles* empleando un lenguaje de programación práctico, las librerías del proyecto *Quantum GIS* y el marco multiplataforma de desarrollo *Qt*. El visor generado puede correr en las plataformas *GNU/Linux*, *Windows* y *Mac OS X*, todo depende de tener los programas requeridos y de definir correctamente la ruta a los *bindings* de *QGIS*.

Construir una aplicación independiente en el escritorio empleando *PyQGIS* es posible gracias al esfuerzo que realizan los desarrolladores de *Quantum GIS* (especialmente *Tim Sutton* y *Martin Dobias*) en proveer las librerías y su documentación.

También es posible construir una aplicación independiente en el escritorio empleando las librerías de *QGIS* con el lenguaje de programación *C++* y *Qt*.

Este artículo está basado en el capítulo "Creating PyQGIS Applications" del manual de usuario versión 1.0 del proyecto *Quantum GIS,* en el artículo "Create a Standalone GIS Application 1" del blog de Gary Sherman (ver referencias consultadas) y en el artículo "Construcción de un visor de Shapefiles con herramientas libres: MapWinGIS y SharpDevelop" del autor.

Nota:

La *API* de *Quantum GIS* admite diversos formatos que pueden ayudar a enriquecer nuestro visor. Por ejemplo, podemos cargar geometrías de bases de datos de *PostgreSQL/PostGIS*, archivos ráster, servicios *web* de mapas *(WMS)* y archivos de texto delimitados por comas.

Referencias consultadas:

- *Quantum GIS. User, Installation and Coding guide*. Versión 1.0. 2009. Disponible en la URL: http://download.osgeo.org/qgis/doc/manual/qgis-1.0.0_user_guide_en.pdf (Última visita: Marzo, 2009)

- Sherman, Gary. *Creating a Standalone GIS Application 1.* Disponible en la URL: http://desktopgisbook.com/Creating_a_Standalone_GIS_Application_1 (Última visita: Marzo, 2009)

- *Python Bindings*. Quantum GIS Wiki. Disponible en la URL: http://wiki.qgis.org/qgiswiki/PythonBindings (Última visita: Marzo, 2009)

- *Quantum GIS API Documentation*. Disponible en la URL: http://doc.qgis.org/stable/index.html (Última visita: Marzo, 2009)

- González, Raúl. *Python para todos*. 2008. Disponible en la URL: http://mundogeek.net/tutorial-python (Última visita: Marzo, 2009)

- Carrillo, Germán. *Construcción de un Visor de Shapefiles con Herramientas Libres: MapWinGIS y SharpDevelop*. 2007 Disponible en la URL: http://geotux.tuxfamily.org/index.php?option=com_myblog&show=construcci%F3n-de-un-visor-de-shapefiles-con-herramientas-libres.html&Itemid=59 (Última visita: Marzo, 2009)

1 MapWindow es un proyecto compuesto por una herramienta de escritorio para SIG y un componente SIG programable llamado *MapWinGIS*. *MapWindow* tiene licencia *MPL* y está escrito en el lenguaje de programación *.NET*. Ver: http://mapwindow.org

2 *Quantum GIS* es un programa para SIG con licencia *GNU/GPL* escrito en el lenguaje de programación *C++*. La página *web* oficial del proyecto es: http://qgis.org

3 Desde la versión 4.5, *Qt* cuenta con triple licenciamiento: Licencia *LGPL* (Novedad), *GNU/GPL* y comercial.

4 Adaptado del libro “Python para todos” de Raúl González Duque.

5 Traducido de “What is SIP?”, disponible en internet en la URL: http://www.riverbankcomputing.com/software/sip/intro

6 Página oficial del proyecto OSGeo4W: http://trac.osgeo.org/osgeo4w

7 Página oficial de descargas de PyQt4: http://www.riverbankcomputing.com/software/pyqt/download

8 La traducción puede consultarse en Internet en la dirección: http://mundogeek.net/traducciones/guia-estilo-python.htm