The GEO System
Version 1.33

Tom Vijlbrief
TNO Institute for Perception
P.O. Box 23, 3769 ZG Soesterberg
The Netherlands
Email: TOM@IZF.TNO.NL

July 29, 1992

Abstract

This article describes the GEO System. This is a general purpose GIS (Geographical Information System) frontend for the Postgres object oriented database system. The system is implemented in C++ and uses the ET++ class library (which uses X11) for its graphics and data structures. GEO (and the Postgres and ET++ systems) are still being developed. The current prototypes are used for GIS research.


```
=============================================================================
Tom Vijlbrief
TNO Institute for Perception            Phone: +31 34 63 562 11
P.O. Box 23                             Fax:   +31 34 63 539 77
3769 ZG  Soesterberg                    E-mail: tom@izf.tno.nl
The Netherlands                         or: uunet!hp4nl.nluug.nl!tnosoes!tom
=============================================================================
```

Contents

List of Figures

1 What is GEO ?

GEO is being developed at our human factors laboratory in order to experiment with the various aspects of GIS (Geographical Informational) systems. E.g., what is a good way to compose queries (graphically), how do users interact with geo-related data, etc. The current prototype system is a frontend for the Postgres relational database system. The user can compose queries and look at Postgres relations. The result of the queries is graphically displayed on a map or shown in a tabular format in a scrollable window.

GEO is a Postgres frontend, i.e. Postgres databases should contain all data which the user wants to display. If a user wants some special functionality for his application, then he should extend the Postgres system by adding new types, new relations, new operators/functions and new rules. GEO can remain unmodified.

There is however an exception to this rule: One can define new GEO types. These GEO types (QueryShapes) specify how a Postgres object (tuple) should be represented on the GIS display. New GEO types are coded in C++ and should correspond to (new user-defined) Postgres types. One can also add new coordinate systems/projections and user-defined menu actions which have full access to all the internal datastructures. The details are discussed in the section 'Dynamic Loading'.

GEO can also display data which is stored in a special purpose Rtree database. This database stores static background map data which cannot be queried. This layer was needed because older Postgres releases were not so fast that these data could be retrieved fast enough. The current Postgres release (4.0) is much faster so this separation of data might no longer be needed.

Finally GEO allows the user to annotate maps with lines, symbols and text just as with standard drawing programs.

2 Getting started

2.1 Needed software

The current system is compiled with g++ 2.1 and uses a modified version of the ET++ 2.2 class library. Because these two are very huge and not stable yet, I have decided to distribute only a SPARC SunOs 4.1.1 binary. The modified ET2.2 sources are available in a separate tar, see the README file for details.

Even if you have only have one SPARC then you can use GEO on more workstations. The use of Postgres and X11 enables you to display the results of GEO on any workstation which uses X and to use any machine which runs Postgres for the data storage. (Let the environment variable PGHOST point to the host where Postgres runs.)

You can extend GEO with new types as discussed in the introduction. I have supplied the needed C++ header files on the distribution tape. You need access to a GNU C++ compiler if you want to create your own GEO types. If you don't want to do this you can go ahead without this (free) compiler.

Of course you need the Postgres DBMS which is also freely available. The current version of GEO uses the latest version (3.0). I assume that you know how to operate Postgres. If you do not, then read the Postgres documentation first.

GEO has been tested with Sun Openwindows, X11R5 and the 'olwm' and 'twm' window managers.

2.2 Loading the tape

The GEO distribution is written on a single 60Mb tape cartridge or you can obtain the release with anonymous ftp from postgres.berkeley.edu (this release is identical but smaller, it does not contain the wdb2 files). You need about 51 MBytes of disk space in order to load the whole tape. The GEO system itself is rather small, but the binary Rtree databases are huge. If you don't have enough disk space you can decide not to load these. Especially the 'wdb2' directory is very large (46.4Mb). The 'rtree' directory contains a much smaller data set describing Europe at a low resolution. This data set is probably sufficient if you want to experiment with GEO.

You can load the tape by typing: 'tar xvf /dev/rst8'. If you want to see the contents of the tape without loading it, then type: 'tar tvf /dev/rst8'. See the Unix manual page 'tar(1)' if you are not familiar with the 'tar' command.

Once you have loaded the tape in for example '/usr/geo', you should set the environment variable 'GEOHOME' to '/usr/geo' and set some additional ET++ environment variables. For the 'C Shell' enter:

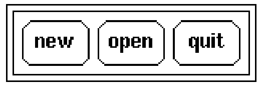


Figure 1: GEO Startup dialog

```
setenv GEOHOME          /usr/geo

setenv ET_DIR           $GEOHOME/et2.2
setenv ET_FONT_SIZE     14
setenv ET_DYN_PATH      .:$ET_DIR/dynmod
setenv ET_SRC_PATH      .:$ET_DIR/src
```

These commands are in the file 'LOGIN'. Copy them to your '.login' file.

2.3 Running GEO

Change directory to the GEO directory. Be sure that the current directory (.) is in your shell search path (PATH), this is usually the default. Now we are ready to start GEO for the first time. Just type 'geo' at your command line. GEO prints some diagnostics about fonts it cannot load and then displays a startup dialog box (Figure 1).

If this does not happen then you have probably not set the environment correctly. E.g. not setting 'ET_DIR' results in:

```
geo: Fatal in <FontManager::FontManager>: can't locate FontTab
Got signal: Segmentation fault
Abort (core dumped)
```

Now that you have GEO running, just press the left mouse button on 'quit', so that we can start configuring the Postgres database.

2.4 GEO options

You can start GEO with the '-Eb' option. This ET++ option enables the double buffering feature. This option is normally off so that you can see the maps being drawn, which is often nice when painting a complicated map takes a long time. The advantage of double buffering is that the drawing operations are somewhat faster, but you have less feedback. You should experiment to see if you like this option. ET++ also implements an image cache feature. This cache prevents costly redraws of rtree layers when parts of the map are uncovered by other windows. This results in a dramatic speedup. The sad point is that the current ET++

implementation has a bug which results in a small border of 'dirt' on the upper and left window edges, which I find unacceptable. By setting the geo option 'cache' (see the end of this section) and supplying the '-Eb' switch the cache is enabled. Just try it. (Note that just the rtree layers are cached, NOT the query or overlay layers, because they change too often!)

An important environment variable is 'GEODB'. This variable tells GEO in which postgres database (default: 'geo') to look for information. This information consists of user relations containing the data of interest but also of GEO relations with a name starting with 'geo_'. (E.g. 'geo_icons', 'geo_colors', etc.)

Geo options can be set by placing the keywords in the GEO environment variable (separated by ' ' or ',') or by supplying them on the command line as an option with '-' prepended. The geo options are:

cache Already described in this section.

merge Shows the 'map overview' (Figure 4) in the upper right corner of the main map window (Figure 3).

noover Disables the 'overlay' layer.

nortree Disables the 'rtree' layer.

sym Enables symbology in the 'overlay' layer.

browse Disables all GIS functionality and windows.

2.5 Initializing the Postgres database: 'geo'

The final step is filling the Postgres database 'geo'. I assume you have Postgres up and running. Change directory to $GEOHOME/postgres and type 'buildall'. This will create the Postgres database 'geo' and fill it with a few GEO relations and some demo relations.

We can also create four 'path' relations containing the borders, lakes, land and rivers in Europe. Change directory to '$GEOHOME/rtree' and type: 'makepostgres'.

We can also create two relations which are used by 'demo.new' and 'demo.new2'. These relations use the dynamic loadable 'MapShape.o' and are therefore much faster then there 'path'-equivalents. Type 'makemaps' to make these relations.

Finally we can extend Postgres with an example distance operator. Note that this step is optional. You can skip it if you don't want to experiment with queries which use these operators in their where-clause or you can add it at a later time when you are more experienced. Change directory to '$GEOHOME/postgres'. Now type: 'make install' to add the definitions of this operator to Postgres.

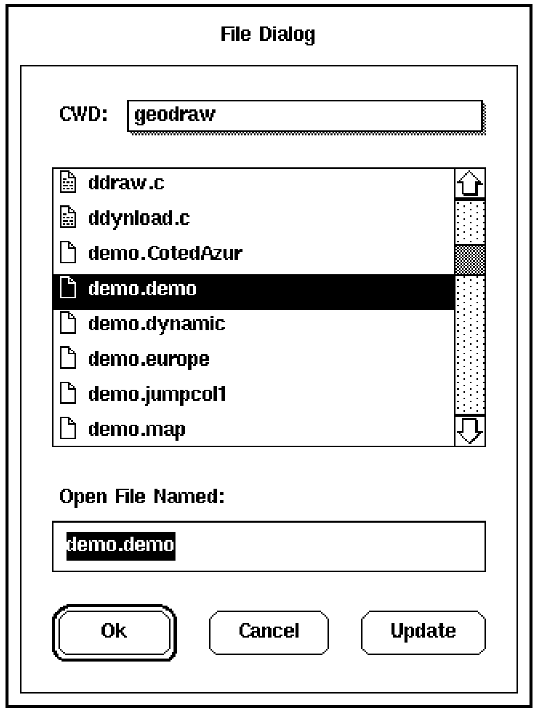


Figure 2: File Selector dialog

3 Operating GEO

3.1 A demonstration document

Now we are setup to look at a demo. Start GEO and click on 'open'. There appears a File Selector dialog (Figure 2).

Move the mouse cursor in the area that shows the list of filenames and type a 'd'. This will scroll to the files starting with a 'd'. Now click on 'demo.demo' and hit 'Ok'. GEO will execute a whole bunch of Postgres queries and finally displays a map with cities with more than 400000 inhabitants in Holland. (Figure 3).

The land and the borders are taken from the Rtree database, the cities are loaded from a Postgres relation.

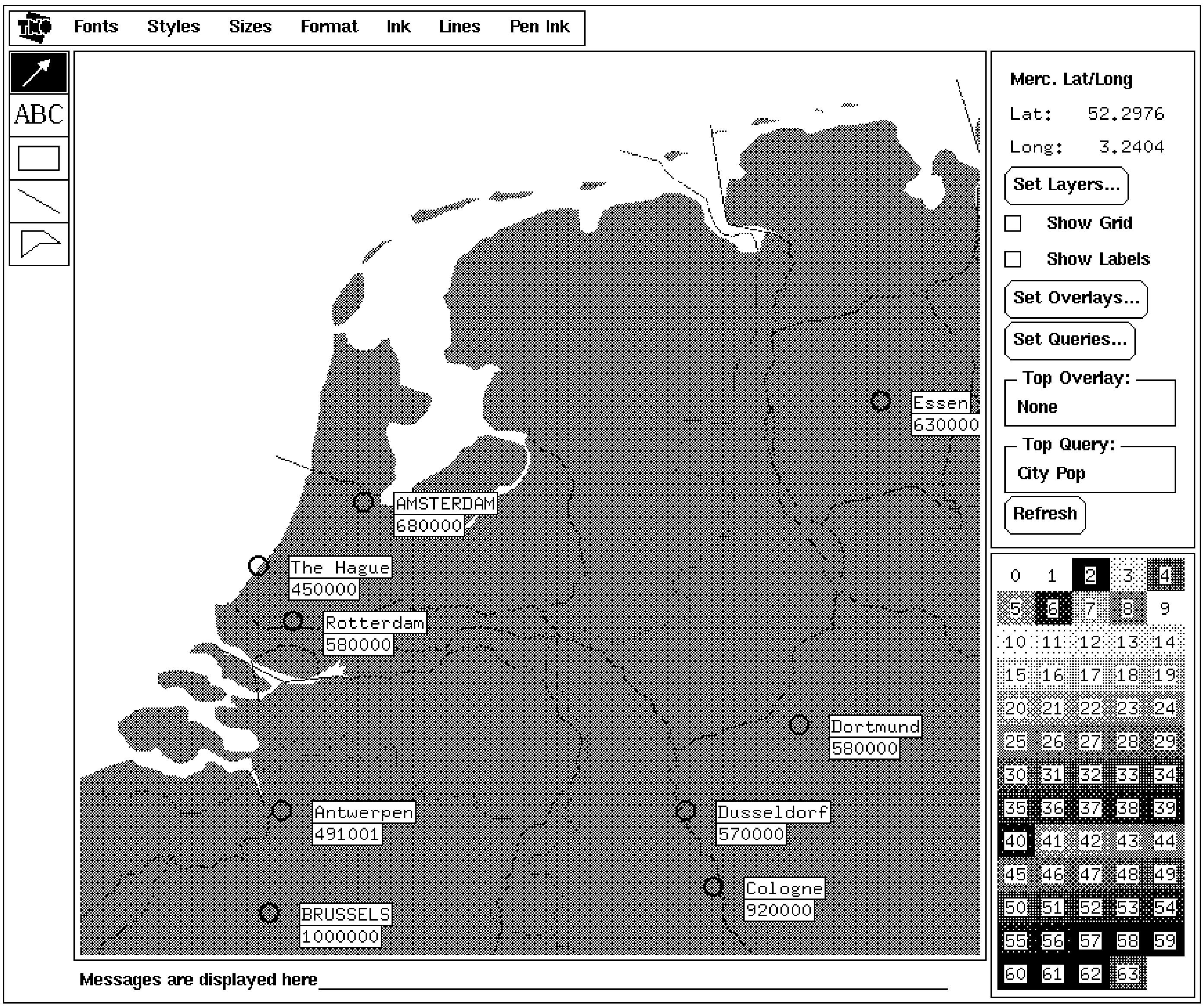


Figure 3: The Map Window

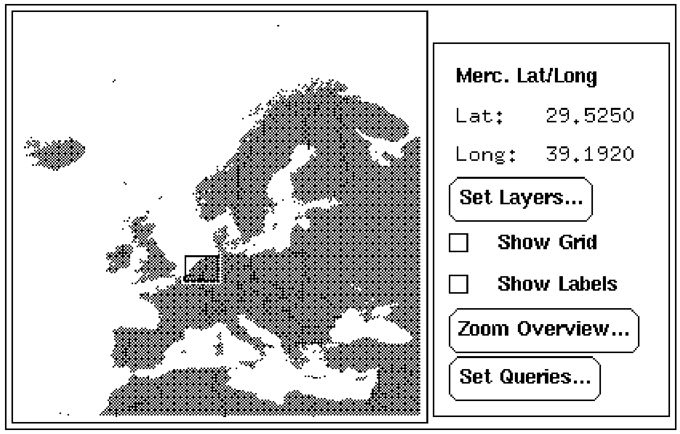


Figure 4: The Overview Window

3.2 The GEO Windows

3.2.1 The OverView Window

The OverView Window (Figure 4) shows an overview map. The area which is shown on the map window is indicated on this window by means of two white and black rectangles. One can zoom in on another area by selecting a box with the middle mouse button. Clicking the middle mouse button without moving the mouse, selects a new center for the displayed area. The items in the right sub window are:

Long: The longitude of the current mouse position is shown here.

Lat: The latitude of the current mouse position is shown here.

Set Layers... Hitting this button displays a 'Layer Selector' (Figure 6). With this dialog you can select the set of Rtree layers which is shown on the overview. It is generally wise to select just a few layers which can be drawn quickly. The left collection specifies the layers which are drawn. The item at the top of this list is drawn first, then the second item, etc. This means that the layer that you specify as the last layer will be drawn on top of all the others and should be the 'most important one'.

Show Grid This toggle toggles the drawing of a long/lat grid.

Show Labels This toggle toggles the drawing long/lat labels. Note that the labels are only shown when you have selected the 'Show Grid' toggle.

Zoom Overview... This button allows you to change the area which is shown in the overview. A zoom dialog will pop up (Figure 5). If you click on 'New Center', you should subsequently click with the middle mouse button in the overview on the new desired center. If you choose 'Zoom In', you should select a new area box with the middle mouse button. The 'Zoom Out' command requires no further actions. It just expands the current

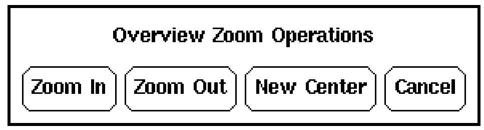


Figure 5: The Overview Zoom Dialog

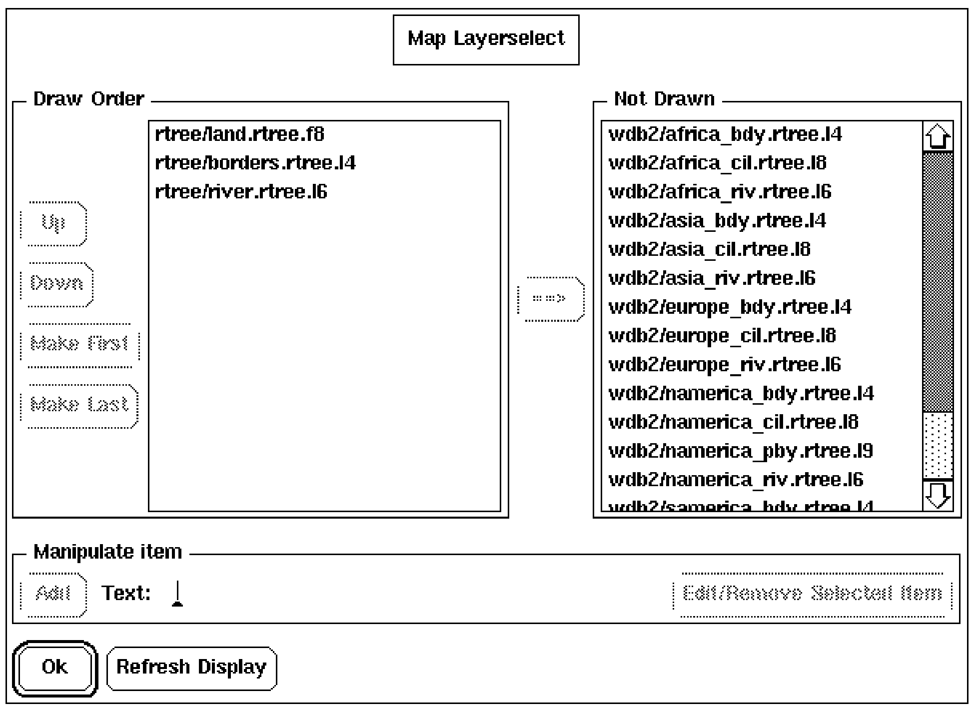


Figure 6: A Layer Selector

overview area with a factor two. Note that GEO does not allow you to select an area that would extent over the poles. It gives you a warning that the aspect ratio will become distorted. You can correct this and gain a correct aspect ratio by executing a 'zoom in' command.

Set Queries... This button shows a layer selector which is used to create or select new Postgres Queries. We'll discuss the details later.

3.2.2 The Map Window

The map window (Figure 3) is similar to the overview window (Figure 4), but it has two additional layers (queries and annotations) drawn on top of the rtree layer. The items which are specific for the map window are:

Right Mouse Button The right mouse button shows a popup menu. Most commands in this menu apply to the annotation layer, except the 'file' sub menu, which allows loading and saving of 'GEO documents'. A GEO document consists of the state of all GEO windows and dialogs. The entry 'application window' in the file sub menu shows the startup dialog (Figure 1).

Left Mouse Button The left mouse button operates on objects in the annotation layer and shows information about objects in the Postgres layer. Just click on the icon or line of a Postgres object and an information dialog will appear (Figure 9). This dialog will be discussed later in this chapter.

Top Menubar All commands in this menubar work on the currently selected item in the annotation layer.

Set Overlays... This button shows a layer selector which is used to create or select new annotation layers. It will be discussed in a separate section of this manual.

Top Overlay: This field shows the name of the top annotation overlay.

Top Query: This is the name of the top Query layer.

Set Queries... This button shows a layer selector which is used to create or select new Postgres Queries. We'll discuss the details later.

Refresh Hitting this button will refetch all displayed queries from the Postgres database. This is useful if data in the Postgres relations has been changed.

Async This button will update queries according to the Asynch Update information. See the section on "Dynamic Displays" for details.

Color Palette A color palette is shown in the bottom-right of this window. Colors are numbered from '0' to '63'. The actual RGB-values are taken from the 'geo_colors' relation or (if that relation does not exist) are taken from a small default set.

3.3 The Rtree layer

We'll first give some more details on the Rtree layer (Figure 6) which you already have encountered during the discussion on the overview window.

When you have created a new empty GEO document with the 'new' button of the startup dialog (Figure 1), then you can initialize the set of rtree layers by typing the name of each layer after 'Text:' and hitting the 'Add' button. A faster and less error prone way to do this is to load a file whose name ends with '.rtreeboot'. The file 'rtree.rtreeboot' in the GEO directory is such a file. It contains the set of rtree layers.

Each layer has encoded the color and type in its name. After the first '.' is the type of the layer. In the current GEO implementation this should always be 'rtree'. The two characters after the next '.' specify if the layer should be drawn as filled polygons ('f') or as polylines ('l'). The number at the end specifies the color.

The program 'Rinsert' in the 'rtree' subdirectory builds the binary tables. This program is used by the script 'makewdb' to create the 'wdb2' files. The exact format and use of these programs is not within the scope of this manual. Study the sources in the 'rtree' directory for details.

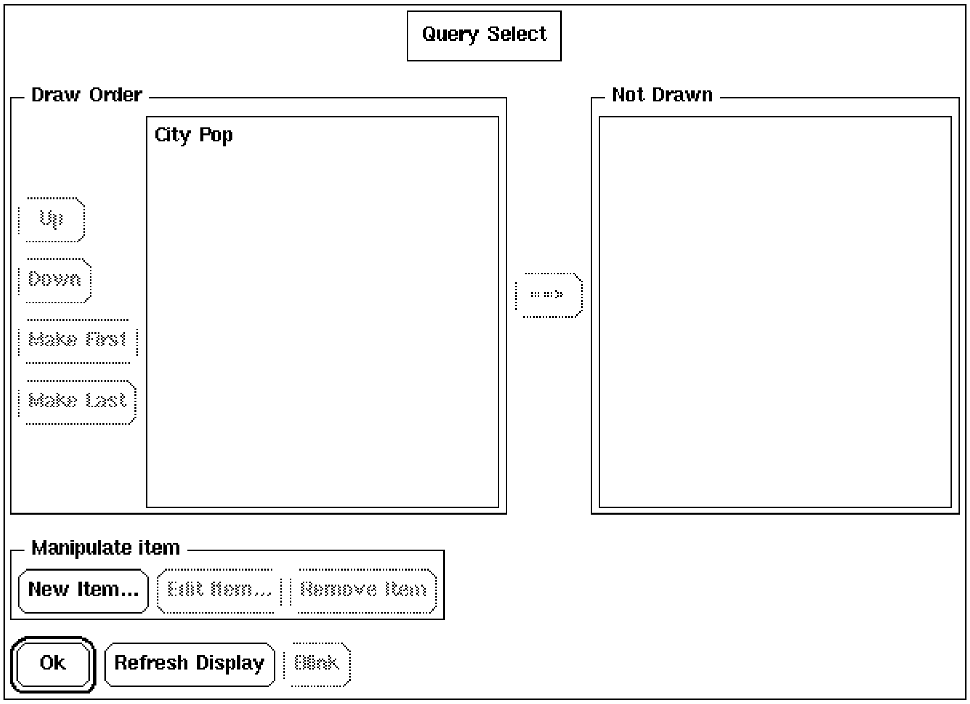


Figure 7: A Query Selector

4 Modifying the Query layer

We'll now look at how to manipulate the Query layer. Hit the 'Set Queries...' button of the main map window (Figure 3) and a query layer selector dialog (Figure 7) will appear. This selector is similar to the Rtree layer selector, except that the naming/editing of the layers is done in a separate dialog. The 'Refresh Display' button redraws the map display after you have shuffled the Query layers. (This is also obtained by closing this window with the 'Ok' button.) The 'Refresh Display' button does *not* refetch data from the Postgres relations (as the 'Refresh Queries' button in the map window *does*), except when queries are moved from the 'Not Draw' collection to the 'Draw Order' collection. It is not necessary to close the layer selectors with the 'Ok' button in order to execute other GEO operations, you may leave these windows open so they will always be quickly accessible.

Note that most of the GEO dialogs need not be closed with the 'Ok' ('Done') or 'Cancel' button, before you can execute commands on other windows. Exceptions are some dialogs in the 'Query Composer'. If you forget to close these dialogs then the other windows will 'beep'.

Editing of queries can be done by selecting the name of the query layer with the left mouse button in the 'Draw Order' collection and hitting 'Edit Item'. (Another way to do this is to double-click on the name of the query layer. Double clicking always means 'open/edit object' in ET++ and GEO.)

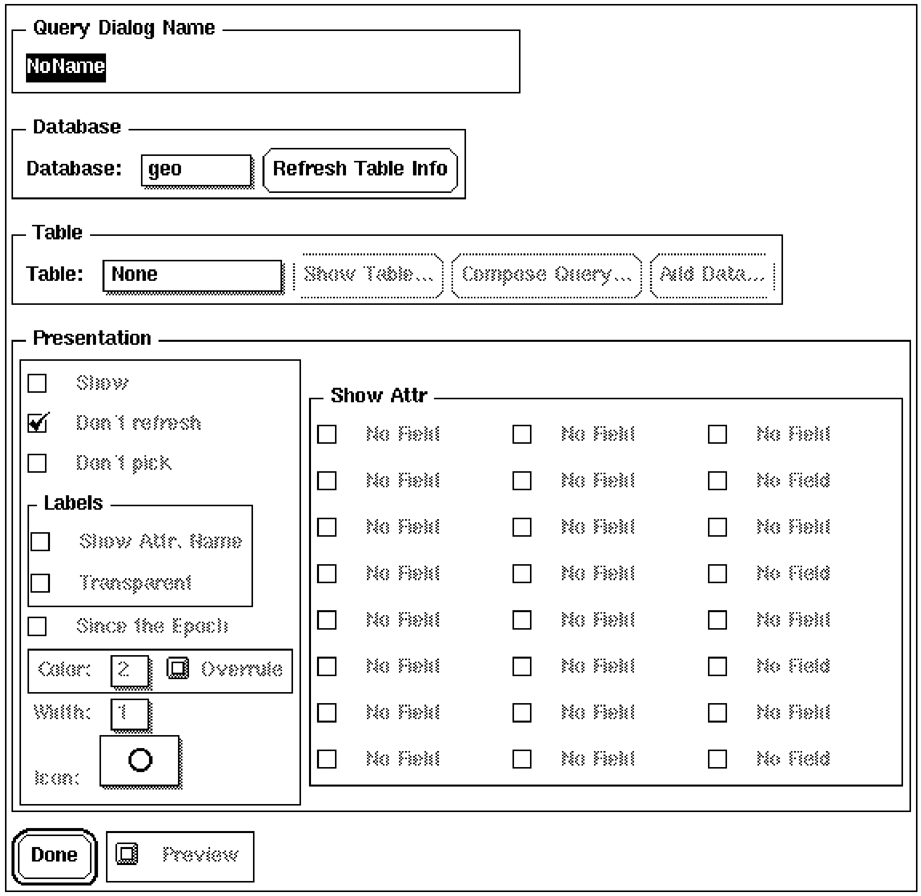


Figure 8: A Query Dialog

4.1 The Query Dialog

The next step is to click on the 'New Item' button with the left mouse button. A 'Query Dialog' will appear (Figure 8). We will now discuss the items in this dialog:

Query Dialog Name You can enter a name for the query here.

Database You can choose a Postgres database name from this popup menu (left mouse button). The default value is 'geo' or it is set from the 'GEODB' environment variable. Note that if you change the database, the GEO meta relations (like 'geo_colors', 'geo_icons', etc.) will still be taken from the 'geo' database (or from the relation specified by 'GEODB').

Refresh Table Info Refreshes the GEO meta relation cache. This is needed when you for example want to view a table you just created with the Postgres terminal monitor.

Table You can choose a table (relation) from the specified Postgres database. The 'Show Table', 'Compose Query' and 'Add Data' buttons will be discussed later in this section.

Show With this toggle you can display the result of Postgres queries on the map window. A Postgres relation is 'showable' when the relation:

1. has a 'geo_pline' attribute of type 'path'. A 'path' that is closed (the representation starts with '(1,...') is displayed as a filled polygon, a path that is not closed '(0,...)' is displayed as a polyline.
2. has a 'geo_loc' attribute of type 'point'.
3. is specified in the 'geo_dyninfo' relation. (See the section on 'dynamic loading'.)

Don't refresh This toggle can be made active if the 'show' toggle is on. It means that a query will not be refetched when the 'Refresh Queries' button is hit on the map window. It is useful for Postgres relations that do not change often (e.g. borders, mountains, etc).

Don't pick Disables the possibility to pick objects from this query.

Show Attr. Name Enabling this toggle will show the attribute name before each attribute value label (see 'Show Attr').

Transparent Draw the labels transparent instead of on a white box.

Since the Epoch This toggle specifies that all history data should be fetched from the Postgres relation. GEO will execute a '`retrieve (r.all) from r in relation[,]`'.

Color: With this popup item you can specify a color. If the relation has a 'geo_color' attribute (see for example the relation 'col1') then the color is taken from this attribute.

Overrule This radio button overrules the 'geo_color' attribute and allows you to pick a color from the 'Color:' popup menu.

Width: Similar to 'Color:' this popup menu can be used to specify a line width for 'geo_pline' relations. If the relation has an attribute 'geo_width' then the width is taken from that attribute.

Icon: If a relation is shown and it has a 'geo_loc' attribute then you can select an icon from this popup menu. The selected icon will be displayed at the geographical location as specified by the 'geo_loc' attribute. The available icons are loaded from the GEO meta relation 'geo_icons'. If this relation does not exist then a default set of icons is used. Look at the script 'fillicons' in the 'postgres' subdirectory if you want to add your own icons.

If a relation has an 'geo_icon' attribute (of type 'int2') then the displayed icon will be the icon with the ordinal number taken from this 'geo_icon' attribute. (Look at relation 'icon1' for an example.)

Show Attr Toggling these options specifies that you want to attach a label to the shown objects. The labels consists of 'attribute name:' followed by the actual value of this attribute.

Preview When this radio button is pressed each change to the display parameters (e.g. 'color' or the restricting where clause as specified with the 'Compose Query...' dialog) will immediately be made visible on the map window. If this button is off then the update of the map display will be postponed until you have hit the 'Ok' button of this dialog and subsequently have given a 'Refresh Display' (or hit 'Done' in the query layer dialog) or 'Refresh Queries' command. Note that the 'Preview' button will be switched off each time you end the query dialog, because it is a costly option.

Finally we will add a query to the map display. Choose 'col1' from the 'Table:' popup and wait till the 'Show Attr' fields are changed from 'No Field' to the correct attribute names. Toggle the 'Show' toggle and select an icon from the 'Icon:' popup. You can also toggle the 'Don't refresh' option. Finally hit the 'Done' button and on the map display will appear a collection of colored icons.

4.2 Viewing and Editing of Postgres relations

We will now look at some Postgres relations. The 'Show Table...' button of the query dialog (Figure 8) does allow us to examine and change Postgres relations. There are two ways to bring up the query dialog. One could select the query layer from the query layer dialog (Figure 7) and hit 'Edit Item' or you can just click on a displayed object of the specific relation.

Let's look at the 'col1' relation added to the display in the previous section. Click on one of the colored icons from the 'col1' relation with the left mouse button. An 'edit dialog' will appear (Figure 9).

The 'edit dialog' shows information about a tuple and allows you to edit this information. If you were just curious and wanted to know some details about this object, then you would hit 'Cancel' at this point and the dialog will disappear. Do this now, because you probably missed something. GEO gives you feedback about which object you did select. Click again on a object of the 'col1' relation, but do not release the mouse button until the selected icon is highlighted (drawn in color 63 (black)). This is the feedback GEO gives you. If you don't want the 'edit dialog' for this object (e.g. because you selected another object than you intended), then you just have to move your mouse slightly before releasing the mouse button.

GEO selects the object nearest to the point where you clicked, but the distance must be less than 20 pixels. When two objects have the same distance then priority is given to the object in the upper query layer.

The buttons in the 'edit dialog' are:

Create on Map Allows you to fill a 'point' or 'path' attribute by creating it on the map. For a point you just have to click the left mouse button on the position where you want the point to be located. For path attributes you have to click the left mouse button on each location where you want a node. Clicking twice on the same location (or closing

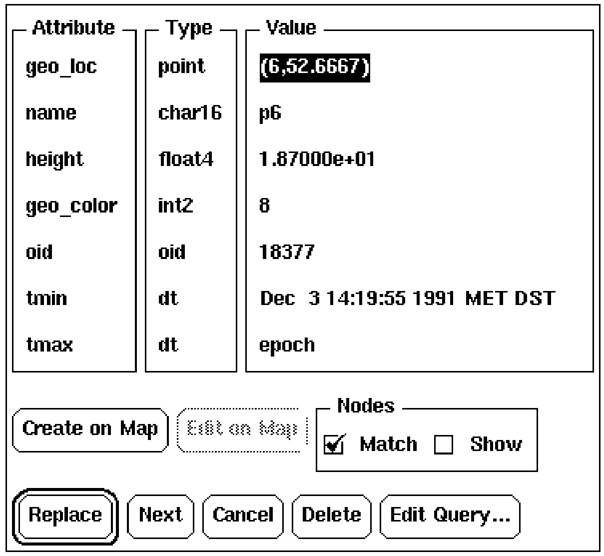


Figure 9: The Postgres Tuple Edit Dialog

a polygon by clicking again on the first point you entered) exits the 'create' operation and returns the input focus to the 'tuple edit' dialog.

Edit on Map Allows you to edit a 'path' attribute. You can drag nodes of the path with the left mouse button. Just clicking on a node without moving will remove the node from the path. Clicking on an arc between two nodes while pressing the Control-key will insert a new node. You can exit the 'edit' operation by just clicking the mouse while you are not pointing to a node. *Note that GEO will warn you if the generated path attribute exceeds 8000 characters. This is the maximum PostQuel query length. GEO will adjust the number of decimal places so that the attribute does not exceed this limit.*

Replace/Add Replaces/Adds the edited attributes in this tuple. Note that the text of this button is labeled 'Replace' or 'Add' depending on the current edit operation.

Next Used when adding new data to a database. The data is entered into Postgres and the edit dialog remains open. (The Add/Repace button closes the window). By using the next button you can enter a serie of new tuples. Note that choosing 'Next' while the 'Create on Map' button is enabled (because the text cursor is positioned on a 'path' attribute) will start the digitising mode (as if you had choosen 'Create on Map'). This allows you to enter a whole range of 'path' tuples in an easy way.

Nodes Show This toggle ('Off' by default) causes the nodes of paths to be marked with a highlighted rectangle.

Nodes Match This toggle ('On' by default) causes the start and end points of an entered path to be placed on the start/end points of nearby paths. The default maximum distance is 5 pixels. You can change this by setting the environment variable GEO-MARGIN to another value.

Cancel Dismiss this dialog.

Delete Delete this tuple from the relation.

Edit Query... Brings up the 'query dialog'.

The next step is to hit the 'Edit Query...' button from the 'edit dialog' because we want to take a look at this relation. The (familiar) 'query dialog' (Figure 8) will appear. Now hit the 'Show Table' button and a 'table dialog' (Figure 10) pops up.

If a relation is 'shown' on the map window then only the objects which are visible on this window (i.e. the objects whose bounding box intersects with the bounding box of the current area zoomed in on) are shown in the 'table dialog'. If you want to see all the objects of a relation then you must disable the 'Show' toggle in the query dialog (Figure 8).

The special buttons of this dialog are:

Refresh Refetches the data from the Postgres database.

Add Row... Displays the 'edit dialog' (Figure 9) and allows you to add a new tuple to this relation. You can step through the attributes by hitting the 'TAB' key. 'Shift-TAB' moves backwards. Note that the 'edit dialog' can also be displayed by using the 'Add Data' button in the 'query dialog' window.

Edit Row... After selecting a cell with the left mouse button, you can use the 'Edit Row' button to edit a row with the 'edit dialog' (or you can double click on a cell). Note that you can copy the contents of a selected cell to the ET++ Clipboard by selecting 'copy' from the right mouse button popup menu. The clipboard can be pasted to e.g. an editable text field in the 'edit dialog'.

While editing a tuple you cannot select attributes like 'oid'. They are read only. You can choose the 'Delete' command to remove a tuple from the relation.

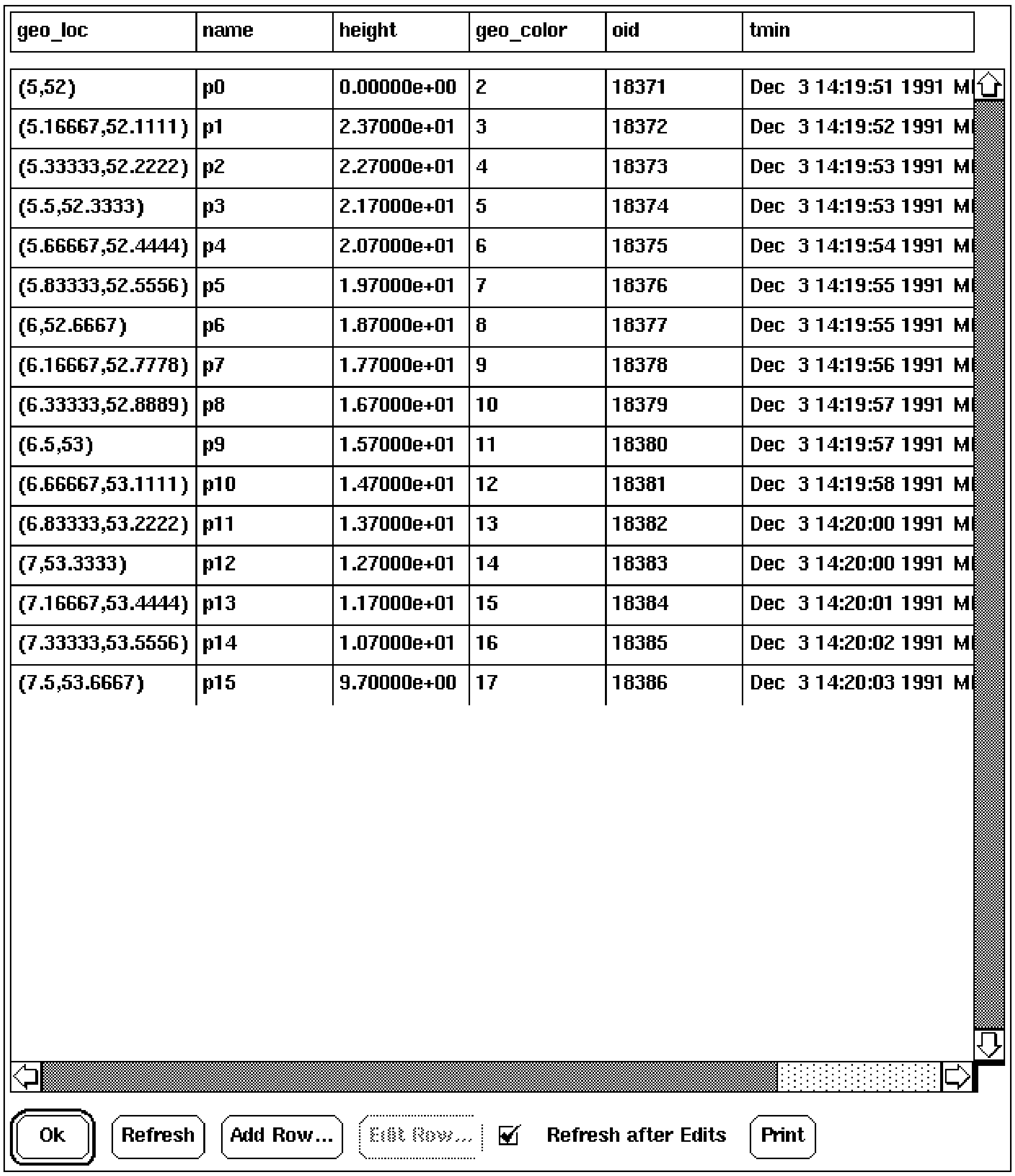
Refresh after Edits This toggle enables the automatic 'Refresh' action which is executed after rows have been editted or added.

Print The 'Print' command pipes the contents of the cell to the Unix 'lpr' command. If you want to pipe to another command then you may set the 'GEOPRINT' environment variable to this command.

Now hit 'Ok' to return to the 'query dialog' (Figure 8) so we can experiment with composing queries.

4.3 Composing 'Where' clauses

Now we'll look at how to compose and edit 'where' clauses for Postgres queries.



geo_loc	name	height	geo_color	oid	tmin
(5,52)	p0	0.00000e+00	2	18371	Dec 3 14:19:51 1991 MI
(5.16667,52.1111)	p1	2.37000e+01	3	18372	Dec 3 14:19:52 1991 MI
(5.33333,52.2222)	p2	2.27000e+01	4	18373	Dec 3 14:19:53 1991 MI
(5.5,52.3333)	p3	2.17000e+01	5	18374	Dec 3 14:19:53 1991 MI
(5.66667,52.4444)	p4	2.07000e+01	6	18375	Dec 3 14:19:54 1991 MI
(5.83333,52.5556)	p5	1.97000e+01	7	18376	Dec 3 14:19:55 1991 MI
(6,52.6667)	p6	1.87000e+01	8	18377	Dec 3 14:19:55 1991 MI
(6.16667,52.7778)	p7	1.77000e+01	9	18378	Dec 3 14:19:56 1991 MI
(6.33333,52.8889)	p8	1.67000e+01	10	18379	Dec 3 14:19:57 1991 MI
(6.5,53)	p9	1.57000e+01	11	18380	Dec 3 14:19:57 1991 MI
(6.66667,53.1111)	p10	1.47000e+01	12	18381	Dec 3 14:19:58 1991 MI
(6.83333,53.2222)	p11	1.37000e+01	13	18382	Dec 3 14:20:00 1991 MI
(7,53.3333)	p12	1.27000e+01	14	18383	Dec 3 14:20:00 1991 MI
(7.16667,53.4444)	p13	1.17000e+01	15	18384	Dec 3 14:20:01 1991 MI
(7.33333,53.5556)	p14	1.07000e+01	16	18385	Dec 3 14:20:02 1991 MI
(7.5,53.6667)	p15	9.70000e+00	17	18386	Dec 3 14:20:03 1991 MI

Figure 10: A Postgres Table Dialog

Bring up the 'query dialog' (figure 8) for the 'col1' relation as described in the previous section and be sure that you have enabled the 'Show' toggle. Also activate the 'Preview' button because we want to see immediately what the results of our query looks like. Once you have done this hit the 'Compose Query' button. A 'query composer dialog' will pop up (Figure 11).

The items in this dialog are:

Query Tree View The upper rectangle containing the word 'bool' is the place where you compose your 'where clause'.

Postgres Where Clause Viewer The middle rectangle contains the PostQuel text of the resulting 'where clause'. You cannot edit this text, except when you toggle the 'Manual Edit' toggle.

Query Active This toggle should be enabled if you want the 'where clause' to be active, i.e. you want that the tuples that are retrieved from the PostGres relation are restricted by the 'where clause' you composed.

Manual Edit This toggle allows manual editing of the 'where clause' text and disables composing in the 'Query Tree View'. Note that you can first compose the query in the normal way and switch to manual editing at a later time.

Lines This radio button allows you to specify how the lines between the nodes in the 'query tree' are drawn.

Layout This button changes the layout of the 'query tree'.

You edit/create a 'where clause' by applying transformations (rewriting rules) to the current 'query tree'. You must always select a node by clicking on it with the left mouse button. When you have done this, the node will be displayed highlighted and the menu entries of the popup menu (right mouse button) and the buttons on the right will be activated.

Select the 'bool' node (left mouse button) and press the right mouse button in order to look at the rewriting rules. (Note that most of these options are also available by using the buttons on the right). The rules displayed in the popup menu are:

Undo last change This operation will restore the 'query tree' to its previous state if you want to undo a mistake. (This command is also available by pressing the middle mouse button.)

Collapse/Expand Temporary hide the subtree originating at the selected node. This is undone by applying the 'expand' operation on a collapsed node. 'Expanding' and 'Collapsing' can also be obtained by clicking the middle mouse button on a tree node.

Edit... will popup a dialog (Figure 12) in which you can specify a simple boolean restriction, e.g. 'col1.color < 10', by means of two popup menus (left mouse button) and a editable text field. The contents of the text field is checked, see the description of 'Constant...'

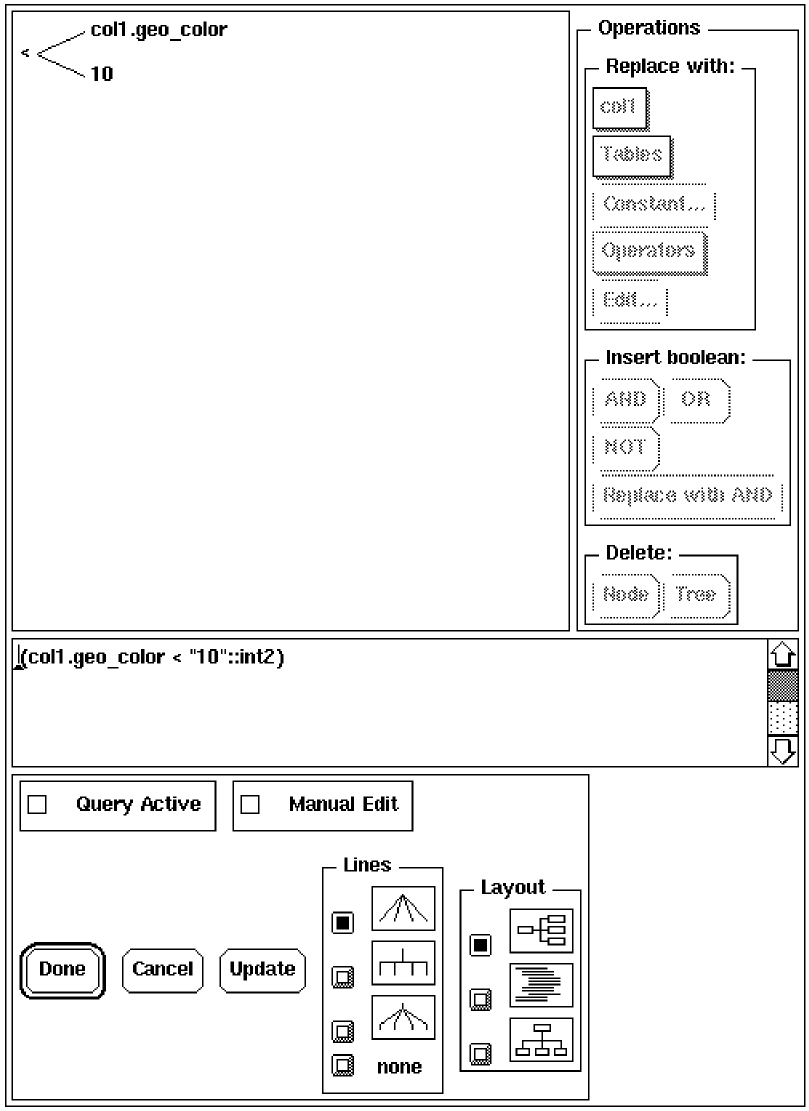


Figure 11: A Query Composer Dialog

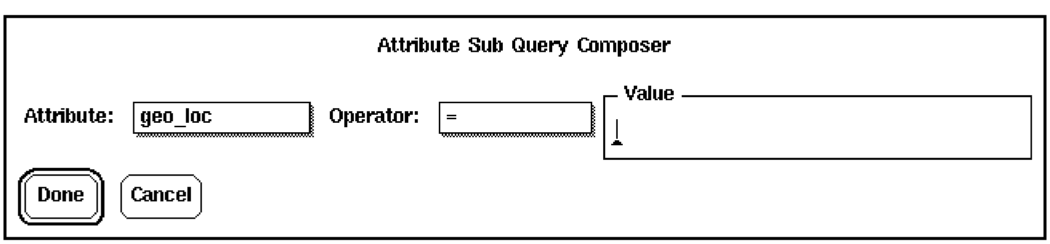


Figure 12: Boolean Restriction Dialog

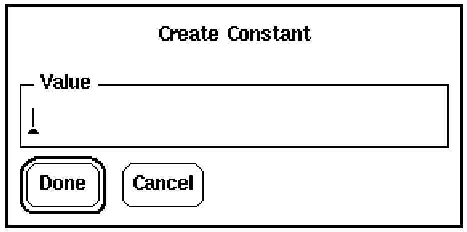


Figure 13: Constant Dialog

for details. It should be clear that this operation can only be applied to 'bool' nodes. Once a node has been 'edited' with this operation then double clicking on the node will reopen the dialog.

Delete node will delete the selected node from the tree or clear it if it is a leaf node. GEO will warn if you would destroy a initialized part of the tree.

Delete tree will delete the subtree rooted at the selected node.

Insert AND/OR/NOT These operations should be obvious. They can only operate on 'bool' nodes.

Replace with AND/OR Replace a 'or' or 'and' node with...

Constant... Creates a dialog which allows you to specify a constant (Figure 13). GEO checks if the entered string matches the regular expression which is specified in the relation 'geo_regexp'. If it does not match then GEO shows a warning and you'll have to change the entered value. Look at the script 'postgres/makeregexp' if you want to add regular expressions for your own or other Postgres types. Note that these regular expressions can also be used to check entered attribute values for specific relations by specifying "table.attr" (e.g. "address.phonenr") instead of a typename. Double clicking on a leaf node does the same.

Viewed Rect Inserts an "Postgres box type" with the viewed rectangle of your map. The coordinates of this box are adapted when you zoom or pan your map. You can use this "Viewed Rect" for optimising queries.

Table⇒ This sub menu allows you to select an attribute from a table. Note that you can also select attributes whose type matches the currently selected tree node.

relname⇒ This menu entry will be labeled with the name of the current relation. It is a shortcut for selecting attributes from the current relation.

Operator⇒ Allows you to:

- Replace the selected (sub)tree with an operator which has the correct result type.
- Replace an operator with another one which has the same result type and the same arguments.

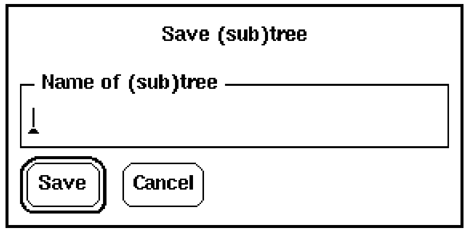


Figure 14: Subtree naming Dialog

The list of allowed operators is loaded from a Postgres relation and therefore contains all existing operators, including the ones you have defined yourself. In an earlier version of GEO it was also possible to call Postgres functions, but this feature has been eliminated because the current version of Postgres does not allow type checking the function arguments.

Predefined Queries⇒ Allows you to save the subtree rooted at the currently selected node or to replace the currently selected node with a tree which has been saved earlier with this command. If you choose 'Save (sub)tree' a dialog will popup, in which you have to enter the name you want to give to this query tree (Figure 14).

Now we'll compose a short query. Say that we only want do display objects from the 'col1' relation whose color attribute has a value less than 10. There are three ways we can compose this specific query:

- Because the query uses a standard '<' operator, we can use 'Edit':
  1. Select the 'bool' node by clicking on it with the left mouse button.
  2. Choose 'Edit...' from the popup menu and a dialog appears.
  3. Select 'color' from the first popup item.
  4. Select '<' from the second popup.
  5. Finally enter '10' in the right text field and hit 'RETURN' or click 'Ok'.
- Or we can compose the query in a more general top down way:
  1. Select the 'bool' node by clicking on it with the left mouse button.
  2. Choose 'Operator⇒' from the popup menu and select '$< (int2 < int2)$'.
  3. Select the upper 'int2' node and select 'col1⇒color' from the popup menu.
  4. Open (double click) the lower 'int2' node and enter '10' followed by 'RETURN' or click on 'Ok'.
- The last approach composes the query bottom up. The main advantage is that we don't have to know the types of the operands in advance when we choose an operator:

1. Select the 'bool' node and select 'col1⇒color' from the popup menu. An 'unknown operator' (? OPR ?) node will be inserted into the tree above the 'col1.color' node.
2. Select the '? OPR ?' operator node and select '< ($int2$ $<$ $int2$)' from the 'Operator⇒' popup menu.
3. Open (double click) the lower 'int2' node and enter '10' followed by 'RETURN' or click on 'Ok'.

Note that the middle window shows the PostQuel 'where clause' at each step. Figure 11 shows the final result for the second composition method.

The last step is to toggle the 'Query Active' toggle and click 'Done'. Now the query will be executed (if you did not forget to set the 'Preview' toggle) and the map display will only show 'col1' objects whose color attribute is less than 10.

4.4 Dynamic Displays

GEO can create dynamic map displays in which objects are redrawn as soon as they are changed. This implemented by defining rules on the relations of interest. These rules fill a special relation with the OID's of the changed objects. Every 5 seconds (this time can be changed by the user) GEO will inspect this relation and update the display. This is implemented by means of Asynchroneous Portals.

All you have to do is to enter the names of the relations which you want to monitor continuously into the relation 'geo_monitors'. Each relation has a flag specifying if you want to monitor the relation for:

deletions Set the flag to '1'.

replacements Set the flag to '2'.

appends Set the flag to '4'.

You can OR the flags if you want to monitor the relation for more than one condition. E.g. the flag '5' will display/remove each object from this relation as soon as it is created/deleted, but the object will not be redrawn if it changes.

If you want a dynamic display then you have to start GEO with the 'postgres/apgeo' script. This script installs all the rules and removes them when the GEO process has finished. Note that only the first opened Map window of a GEO process has the 'dynamic display' feature enabled. If you need more than one dynamic display then you have to start more than one GEO process on your workstation.

The script 'postgres/makeap' installs the needed relations (execute this script once for a specific database, this is already done for the 'geo' database) and 'postgres/buildcol1' contains an example. You may load 'demo.jumpcol1' and start the script 'postgres/jumpcol' for a demonstration. Note that because rules often don't work correctly on the oid that it is possible to maintain your own oid in an additional attribute. The option '-opoid=xxx' specifies the name of the attribute (the default is 'oid').

4.5 Dynamic Loading

GEO has two standard methods to display the result of Postgres queries. It can display an icon on a geographic location retrieved from an attribute with the name 'geo_loc' and it can display a polyline (polygon) from an attribute named 'geo_pline'.

Because these two methods are are probably not sufficient for every application, I have added the capability to GEO to dynamically link in your own methods. GEO can attach a binary object file to its running image with the GNU dynamic linking library.

GEO has been compiled with 'G++', GNU's C++ compiler. The loaded object files should be 'G++ compiled' files, which contain a definition of a subclass of GEO's 'QueryShape' class. It should also contain a normal 'C' function which is called by GEO to create the instances of your own defined 'QueryShape' subclass. This function is passed an array of character strings which contains the actual values of a user specified list of attributes.

The GEO directory contains a few C++ header files and the C++ source file 'IconShape.c' and its header file 'IconShape.h'. You may look at these two files to see how the two predefined drawing methods work. The 'DrawDyn.c' file contains an example implementation of a dynamically loadable drawing method. You can compile this file by executing the 'dyng++' script, e.g. 'dyng++ DrawDyn.c'.

In the 'postgres' subdirectory the script 'makedyninfo' initializes a demo relation 'dynamic'. (This has already been done.) Try to display the 'dynamic' relation. The objects are drawn near 'London'.

The 'geo_dyninfo' relation contains information about which Postgres relations should be drawn with dynamically loaded drawing methods. The relation has the following attributes:

relname Specifies the name of the relation.

relattr Contains a ',' separated list of attributes whose value should be passed to the user defined 'C' function (see attribute description of 'dynfunc').

bboxattr This attribute should contain a bounding box of the graphical representation of the object. If such an attribute does not exist then 'bboxattr' must have the value 'None' (note the capital 'N'), '' or 'null'.

dynfunc Specifies the 'C' function which should be called to allocate a new instance of your own defined 'QueryShape' subclass.

dynfile Contains the name of the binary object file.

the_oid Specifies the attribute in 'relname' that contains the 'oid'. This is needed for relations that are views because these have a 'null' oid. If the value of this attribute is 'None' (note the capital 'N'), '' or 'null' then the system attribute 'oid' is assumed to hold the oid.

is_bin Specifies that GEO should use a binary portal. This is much faster but note that it is no longer possible to label the objects with their attributes.

Relations may share their 'dynfunc' and 'dynfile' definitions. GEO will link the 'dynfile' only once. The scripts 'postgres/addbin' and 'postgres/bindyn' can be used to enter tuples into the geo_dyninfo relation.

A more sophisticated example of a dynamically loadable drawing method is contained in the file 'MapShape.c'. This implementation will draw shapes which are defined by passing the file (xyz.bin) and the offset within this file as arguments when constructing instances of this shape. These arguments and the bounding box (used for the Postgres rtree index) are stored as Postgres attributes. The MapShape will 'mmap(2)' the file in GEO's virtual address space and can therefore quickly access the polygon/polyline data. This implementation almost matches the speed of the special purpose GEO Rtree layer. The script 'rtree/makemaps' creates four maps which use these MapShapes.

4.5.1 Advanced customisation

GEO allows the user to implement new coordinate systems and projections. The default coordinate system is lat/long displayed with a Mercator projection. The file "CoordSys" contains the definitions of coordinate systems. These are dynamically linked just like QueryShapes. The exact details are not in the scope of this manual. Look at the example projection source for implementation details.

One may customise GEO by defining user menu entries for the right mouse button menu. The specifications of these dynamically linked GEO extensions are in the Postgres relation "geo_menu". Study the example "dynamic/DemoMenu.c" for details. Note that the code for user menu entries has access to all internal datastructures of the GEO process and can be used to adapt the system in virtual any way. ET++ knowledge and experience will be useful (get the ET++ release with its documentation). One can overrule virtual functions to modify GEO's behaviour. Study the GEO header files if you want to expiriment with these possibilities.

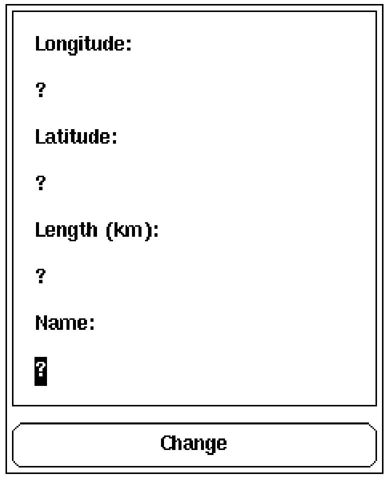


Figure 15: The Annotation Layer Dialog

5 The Annotation Layer

The annotation layer can be used to annotate a map. An annotation is a drawing just like the drawings which can be created with the standard drawing programs, except that the objects in the GEO annotation drawing have a geographical location. This means that when you zoom in on a map, you will simultaneously zoom in on the drawing.

Users can display several annotations at a time. The idea is that users may communicate by loading annotations which are created by others. Annotations are stored in normal ET++ files (a future version of GEO might store the annotations in Postgres relations). One can only edit the top annotation. GEO applies a locking scheme which should guard against simultaneous updates of annotations by different users. GEO creates lock files in the directory '$GEOHOME/locks' or it uses the environment variable 'GEOLOCKDIR' as the directory in which to create its lock files.

Clicking on 'Set Overlays...' results in a layer dialog which is similar to the rtree layer dialog (Figure 6). Here you can enter the filenames of the annotations you want to look at. if the file does not exist then GEO will display a warning and creates it.

Choosing 'Set Overlays...' (or loading a GEO document which contains annotation layers) will also display an annotation dialog (Figure 15).

I will now shortly list some details about the interaction with the annotation layer. The other manipulations should be obvious.

- One can create an object in the top annotation layer by clicking on one of the icons positioned to left of the map window (Figure 3).

- The pull down menus (left mouse button) from the upper menubar and the right mouse button pull down menu of the map window work on the currently selected object(s). Note that some of the menus in the menubar (e.g. 'Lines' and 'Fonts') are scrollable. Try to select a 'line entry' below the lowest and the menu will scroll.
- One can extend the current selection by 'shift-clicking'.
- Typing text while an object has been selected will automatically attach a textshape to that object.
- The annotation window (Figure 15 shows the geographical position of a selected object. You can also see the length of a (poly)line object in kilometers and see or change the name you have given to this object.
- Pressing 'Shift' or 'Control' while adjusting the form of a shape will limit the irregularity of the shape. Try it.

6 Conclusions

GEO is a versatile GIS frontend for Postgres. It is an 'open' system in the sense that it is extensible by the users by means of 'dynamic loading'.

I hope that by now it's clear what the GEO system does and how to operate it. You may use it freely to develop GIS applications, as long as you don't use it for commercial applications, try to sell it or in any other way misuse it.

If you give GEO to others then we would like to receive a message by (E-)mail, so that we know who uses the system. If you redistribute GEO, you should redistribute the whole system and not a part of it. Also a copy of this manual should be included.

See the GNU GENERAL PUBLIC LICENSE for details !!!

I welcome all suggestions for improvements, complaints and bug reports. New versions will appear from time to time. If you are interested then send me (E-)mail. I will ship new releases on tape and charge Fl. 200 (Dutch Guilders) (or $150) for mailing, media and administration costs. If you would like GEO for another CPU architecture then we can probably arrange something.

Again, I would really like to hear from you GEO users.

I would like to thank Peter van Oosterom (pjmg1@fel.tno.nl) who contributed to this GEO distribution the Rtree index code, the Postgres geometric datatypes and many interface and functionality suggestions.

```
Tom Vijlbrief

TNO Institute for Perception
P.O. Box 23
3769 ZG  Soesterberg
The Netherlands

E-mail: tom@izf.tno.nl
Fax:    +31 34 63 539 77
Phone:  +31 34 63 562 11
```

A Environment Options

The following options may be placed in the environment by means of the command 'setenv OPTION=value' (for string valued options) or 'setenv OPTION' (for boolean options). These commands apply to the C-Shell.

GEOHOME= Specifies the root directory of the GEO distribution.

GEODB= Specifies the default Postgres database.

GEOLOCKDIR= Specifies the lock directory for the 'overlay' layer.

GEOPRINT= Specifies the filter to print relations.

GEOCS= Specifies the Coordinate System.

GEOCSBASE= Specifies the Coordinate System description file (normally $GEOHOME/CoordSys).

GEONGRID= Specifies the number of grid lines to display.

GEOAP= Specifies the asynchroneous portal to use.

GEOAPOID= Specifies the attribute which contains the oid for relations that are monitored by asynchroneous portals.

GEOBBOX= Specifies the type of the bounding box area.

GEOHIDETYPE= Specifies a comma ',' separated list of types. Operators which operate on these types are hidden from the operator menu.

GEOCOLORS= Specifies the Postgres relation which contains the RGB mapping.

GEOSIMPLE Specifies that GEO should operate in a mode with less buttons and features.

B Command line Options

The following options may be placed on the command line. Note that you can also place them in the environment by placing them in the GEO environment variable in an ' ' or ',' separated list. E.g. ' setenv GEO "cs=MyProj,rtreeshare" '.

-cache Specifies that rtree layers should be cached in a backing store.

-merge Specifies that the main map window and the overview window should be displayed in one and the same window.

-nortree Hides the 'rtree' operations.

-noover Hides the 'overlay' operations.

-sym enables the use of military symbology in the overlay layer.

-browse Turns GEO into a simple Postgres database browser.

-simple See the environment option.

-rtreeshare Share the loaded rtrees between windows. This means that they will not be freeed when you close a map window and that they will be directly available for new windows the next time they are needed (an rtree is never loaded twice). The disadvantage is that when you access different rtrees sequentially the memory will fill up.

-ngrid= See the environment option.

-cs= See the environment option.

-ap= See the environment option.

-apoid= See the environment option.

-bbox= See the environment option.

-colors= See the environment option.

-margin= Specifies the pick margin for selecting objects.

-mapw= The main map window width in pixels.

-maph= The main map window height in pixels.

-overw= The overview map window width in pixels.

-overh= The overview map window height in pixels.

-hidetype= See the environment option.