June 2007
http://www.umn-mapserver-community.de

pgRouting (with the A*-Algorithm) and
the UMN MapServer

@ Kai Behncke (kbehncke@igf.uni-osnabrueck.de), Florian Thürkow (florian.thuerkow@ufz.de)

With this short tutorial the user may get a little help using UMN MapServer and pgRouting.
This tutorial is mainly based on know-how, which is imparted on the webpage http://pgrouting.postlbs.org.

Special thanks goes to Anton.
Further helpful sources:
Mailing list umn-mapserver.de:
http://freegis.org/pipermail/mapserver-de/2006-August/002433.html (containing 14 responses)

Mailinglist from UMN MapServers (English):
http://lists.umn.edu/cgi-bin/wa?A2=ind0612&L=mapserver-users&T=0&F=&S=&P=30653
(containing 8 responses)

Forum umn-mapserver-community.de:
http://www.selbstverwaltungbundesweit.de/mapserver/modules.php?name=Forums&file=viewtopic&t=331

This tutorial is written for Windows XP, works also on Linux-based systems (with the special adaptions).
You need to have basic knowledge in using/handling UMN MapServer, PostgreSQL/PostGIS and PHP/Mapscript.

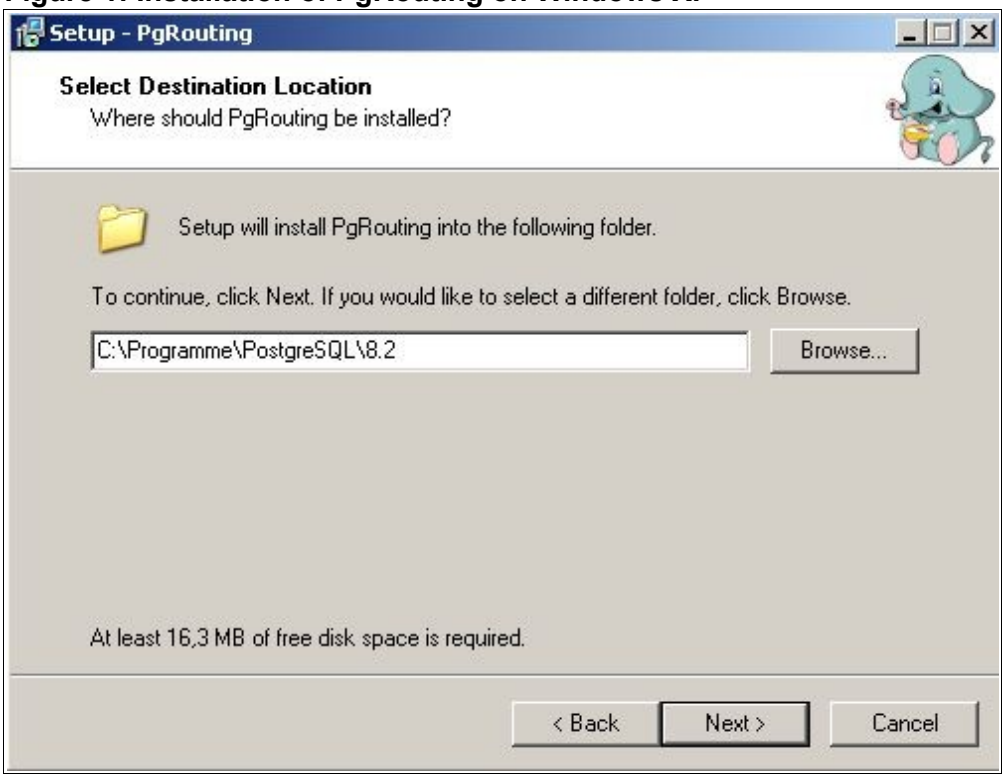
The following environment has been installed:

ms4w-Packet (2.2.3)
PostgreSQL 8.2.4 with PostGIS 1.1

What do we have to do now?

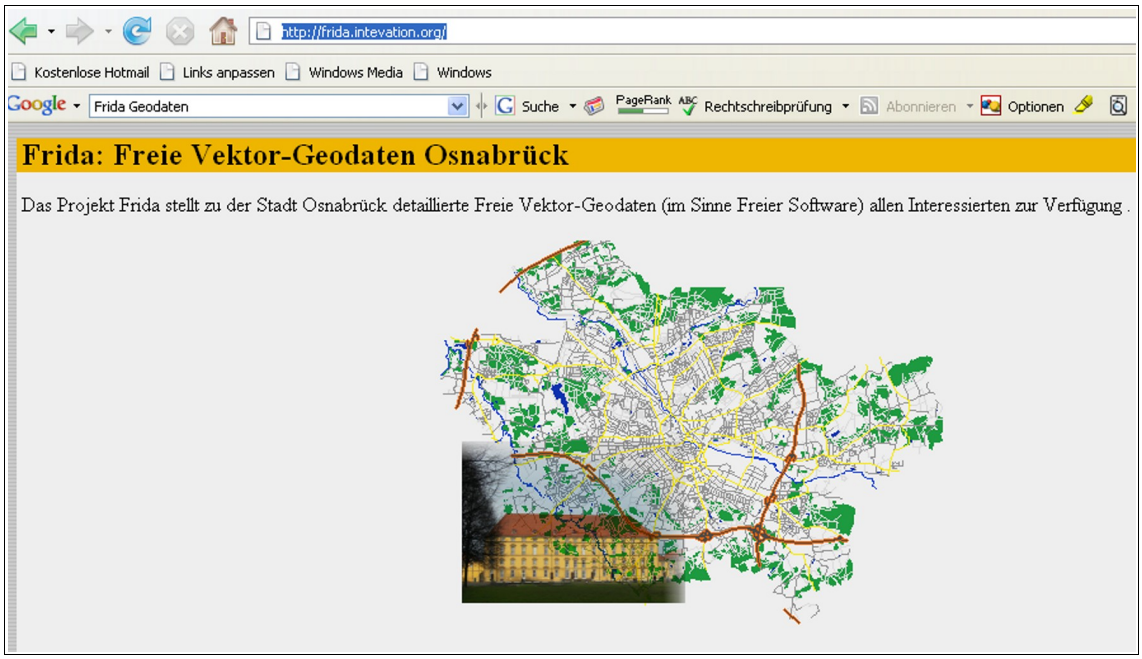
First of all, you need to download the pgRouting 1.0.0a-win32-installer from the homepage http://pgrouting.postlbs.org. Then you have to double click the package. The installation runs nearly automatically. The best way would be to install it into the directory C:\Programme\PostgreSQL\8.2 (fig. 1).

Figure 1: Installation of PgRouting on Windows XP



After that we need to get geodata.
Therefore we use the free geodata from the project „Frida“ (http://frida.intevation.org/) initiated by Intevation GmbH (fig. 2).

Figure 2: Homepage of the Frida-Data

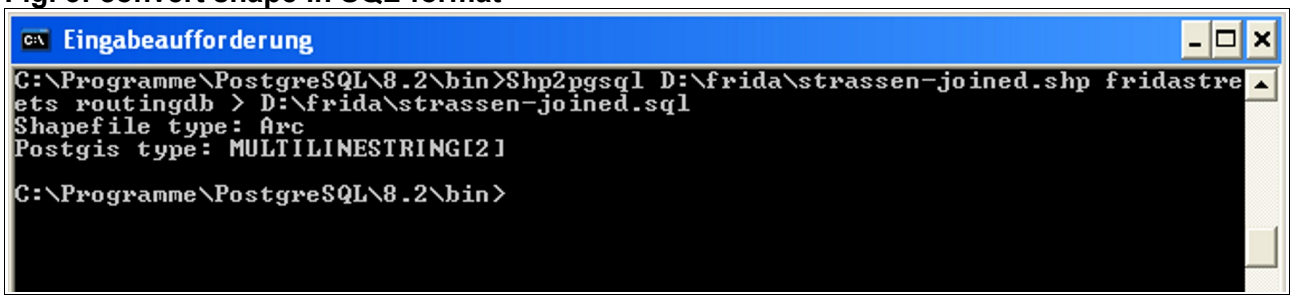


Download the data: frida-1.0.1-shp-joined.tar.gz and unzip it.

We need the „strassen-joined.shp“-data for the first step.
However, the data needs to be in sql-format to import it into a **PostgreSQL/PostGIS-database**, which you first have to construct.
Therefore you have to enter the following into the prompt (fig 3):

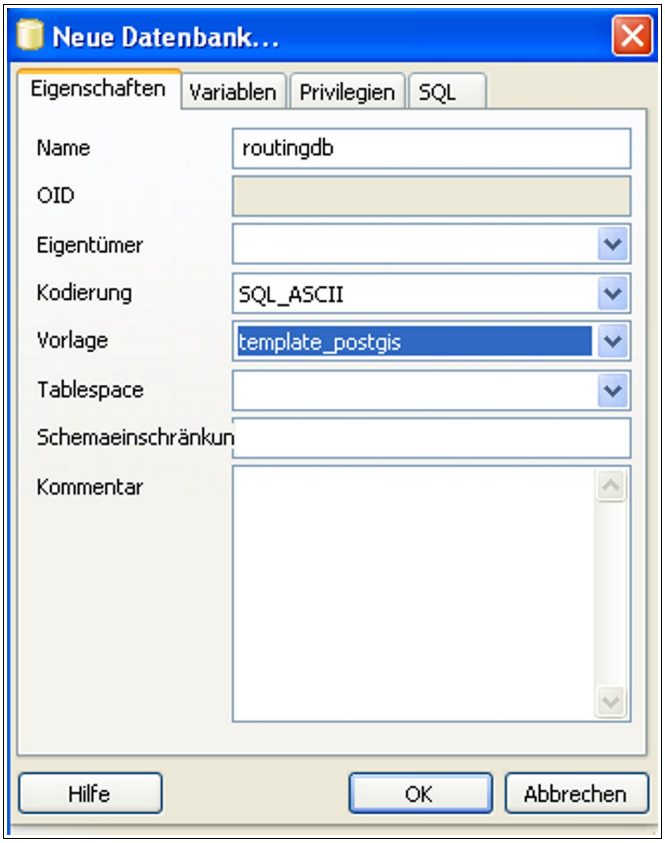
Shp2pgsql D:\frida\strassen-joined.shp fridastreets routingdb > D:\frida\strassen-joined.sql

Fig. 3: convert shape in SQL-format



In the following step we have to create a database with PostGIS-support (e.g. with the tool pgAdmin III).
Let’s call this database “routingdb” (Fig. 4).

Fig. 4: Create a database with PgAdmin III



After that the data- file “strassen-joined.sql” needs to be imported into the database. Therefore enter into the prompt:
psql -U postgres -f D:/frida/strassen-joined.sql routingdb

Fig. 5: Command to import the SQL-Data into the database

```
C:\Programme\PostgreSQL\8.2\bin>psql -U postgres -f D:/frida/strassen-joined.sql
 routingdb
```

The data of the table „fridastreets“ have the following structure:

Fig. 6: Original/basic structure of the Frida-Data

pgAdmin III Edit Data - PostgreSQL Database Server 8.2 (localhost:5432) - routingdb - fridastreets

Datei Bearbeiten Anzeigen Hilfe

100 Zeile

	gid [PK] serial	strshapeid smallint	strid smallint	strtypid smallint	strspuren smallint	strebene smallint	strname character var	the_geom geometry
1	1	1	1	5	0	0	kein Name vorh	0105000000010
2	2	2	1	5	0	0	kein Name vorh	0105000000010
3	3	3	1	5	0	0	kein Name vorh	0105000000010
4	4	4	1	5	0	0	kein Name vorh	0105000000010
5	5	5	1	5	0	0	kein Name vorh	0105000000010
6	6	6	1	5	0	0	kein Name vorh	0105000000010
7	7	7	1	5	0	0	kein Name vorh	0105000000010
8	8	8	1	5	0	0	kein Name vorh	0105000000010
9	9	9	1	5	0	0	kein Name vorh	0105000000010
10	10	10	1	5	0	0	kein Name vorh	0105000000010
11	11	11	1	5	0	0	kein Name vorh	0105000000010
12	12	12	1	11	0	0	kein Name vorh	0105000000010
13	13	13	288	4	0	1	Brockhauser Str	0105000000010

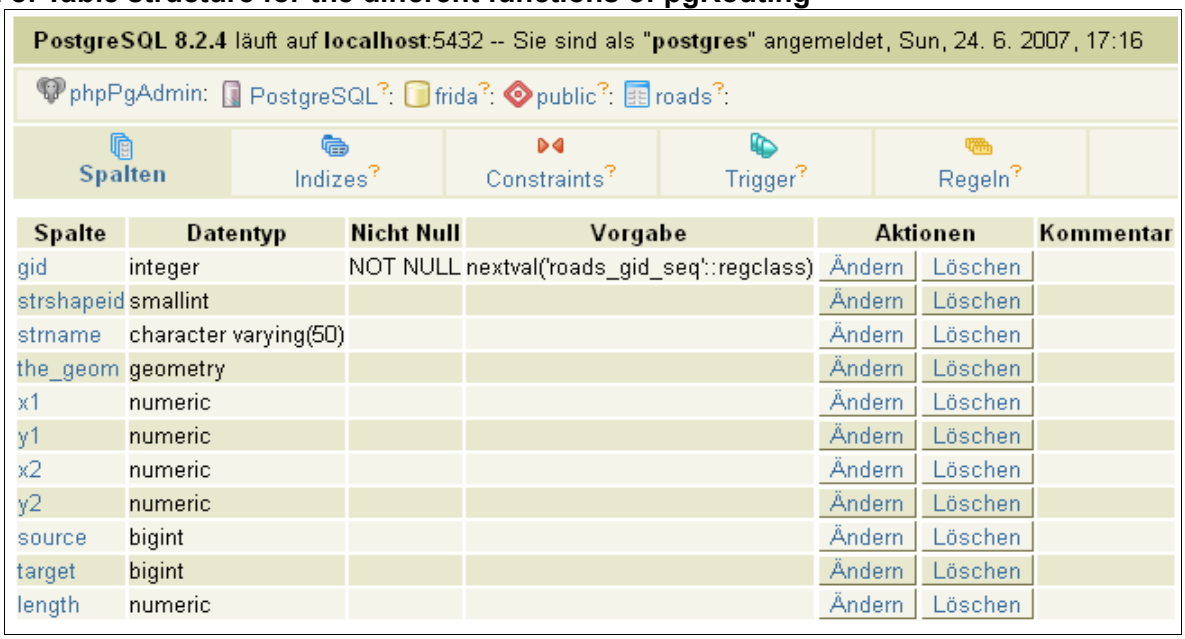
So far so good.
The database is not yet able to calculate routes. That is what we want to change and therefore we need to carry out the following commands:

psql -U postgres -f C:\Programme\PostgreSQL\8.2\share\contrib\routing.sql routingdb
As well as:
psql -U postgres -f C:\Programme\PostgreSQL\8.2\share\contrib\routing_postgis.sql routingdb

Fig. 7: Routing functions are created in the database

Now the database is prepared for routing. Nevertheless, that does not mean that it works in this connection.
To achieve different functions of pgRouting there needs to be a special table structure. Besides the gid and the geometry (the_geom) we also need the starting coordinates (x1, y1 each as its own column (data type numeric)) or the endcoordinates respectively (x2,y2 again each as its own column).
We also need the table columns “length“ (numeric) and “source” and “target” (bigint) (Fig. 8).

Fig. 8: Table structure for the different functions of pgRouting



Spalte	Datentyp	Nicht Null	Vorgabe	Aktionen		Kommentar
gid	integer	NOT NULL	nextval('roads_gid_seq'::regclass)	Ändern	Löschen	
strshapeid	smallint			Ändern	Löschen	
strname	character varying(50)			Ändern	Löschen	
the_geom	geometry			Ändern	Löschen	
x1	numeric			Ändern	Löschen	
y1	numeric			Ändern	Löschen	
x2	numeric			Ändern	Löschen	
y2	numeric			Ändern	Löschen	
source	bigint			Ändern	Löschen	
target	bigint			Ändern	Löschen	
length	numeric			Ändern	Löschen	

After having created those columns, we need to import the values for x1, y1, x2, y2. Therefore the following PHP-script has been written: (lies in the former downloaded folder as x_y_create.php).

```
<?php
$host = "localhost";
$port = "5432";
$dbname = "routingdb";
$user = "postgres";
$password = "postgres";

$con_string = "host=$host port=$port dbname=$dbname user=$user password=$password";
$con = pg_connect ($con_string);

//The code for getting x1 und y1
 $id_check = "SELECT max(gid)as gid from roads";
 $res_id_check = pg_query($con,$id_check);
   $count = pg_result($res_id_check,"gid");
   echo "Anzahl der Eintraegege in der DB: ".$count;
   echo "<br>";


for ($x=1;$x<=$count;$x++)
{

 $start = "SELECT astext(StartPoint(the_geom))as startpoint from roads where gid='$x'";
 $res_start= pg_query($con,$start);
   $start_ergebnis = pg_result($res_start,"startpoint");
   echo "<b>Geometrie $x</b></br>";
  echo "Anfangspunkte (x1,y1): ".$start_ergebnis;
  // echo "<br>";
  $array_01=array("POINT(",")");
  $array_02=array("","");

    for($r=0;$r<sizeof($array_01);$r++)
   {
    $start_ergebnis=str_replace($array_01[$r],$array_02[$r],$start_ergebnis);
    }
$explode=explode(" ",$start_ergebnis);
$x1=$explode[0];
$y1=$explode[1];
  echo "<br>";

//The code for  x2 und y2
  $end = "SELECT astext(EndPoint(the_geom))as endpoint from roads where gid='$x'";
 $res_end= pg_query($con,$end);
   $end_ergebnis = pg_result($res_end,"endpoint");
   echo "Endpunkte (x2,y2): ".$end_ergebnis;
   echo "<br>";
   echo "--------------";
   echo "<br>";

    $array_01=array("POINT(",")");
  $array_02=array("","");

    for($r=0;$r<sizeof($array_01);$r++)
   {
```

```
    $end_ergebnis=str_replace($array_01[$r],$array_02[$r],$end_ergebnis);
    }
$explode=explode(" ",$end_ergebnis);
$x2=$explode[0];
$y2=$explode[1];
  //Values are writte in the columns
    $werte_in_tabelle_schreiben="UPDATE roads SET x1='$x1',y1='$y1',x2='$x2',y2='$y2' where
gid='$x'";
 $res = pg_query($werte_in_tabelle_schreiben);

  }
?>
```

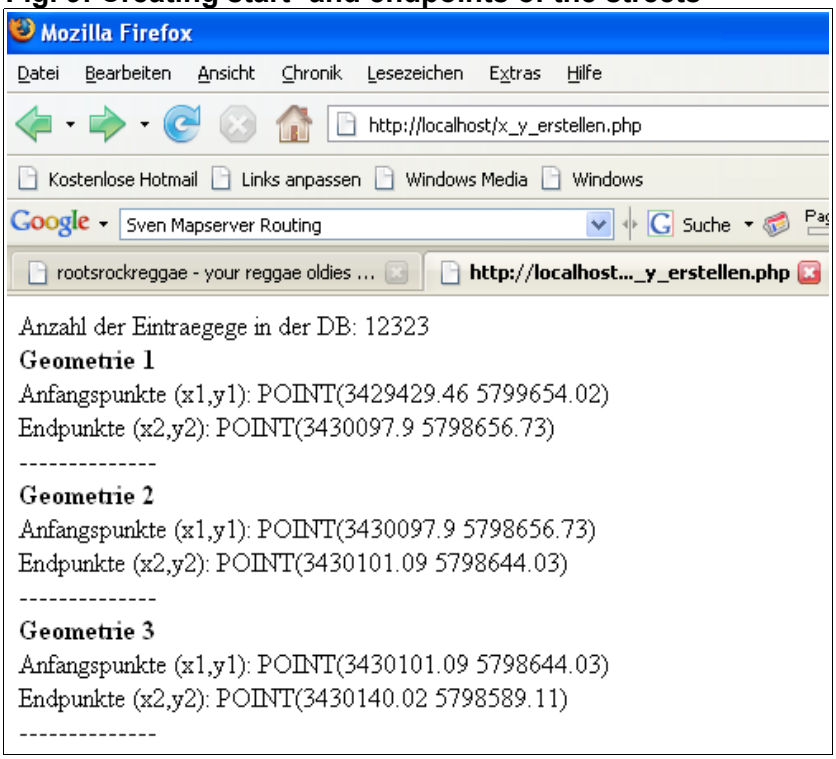
The script works quite simple.
It creates a connection to the PostgreSQL/PostGIS-database.
Then it reads out the number of the geometry-entries and writes the values of the vertices into the table in a loop. The coordinates are in the typical german Gauss-Krüger-system.
Next, you need to activate the script in the htdocs-directory of the Apache-webserver.

Attention: It could take a while until the coordinates are written into the database.
In case you work with the ms4w-Packet, you need to change the values in the PHP-configuration file “php.ini (C:\ms4w\Apache\cgi-bin)”.
Maximize the execution-time in line 255, e.g..:
max_execution_time=300;

...that’s the only way to get all the data records imported.

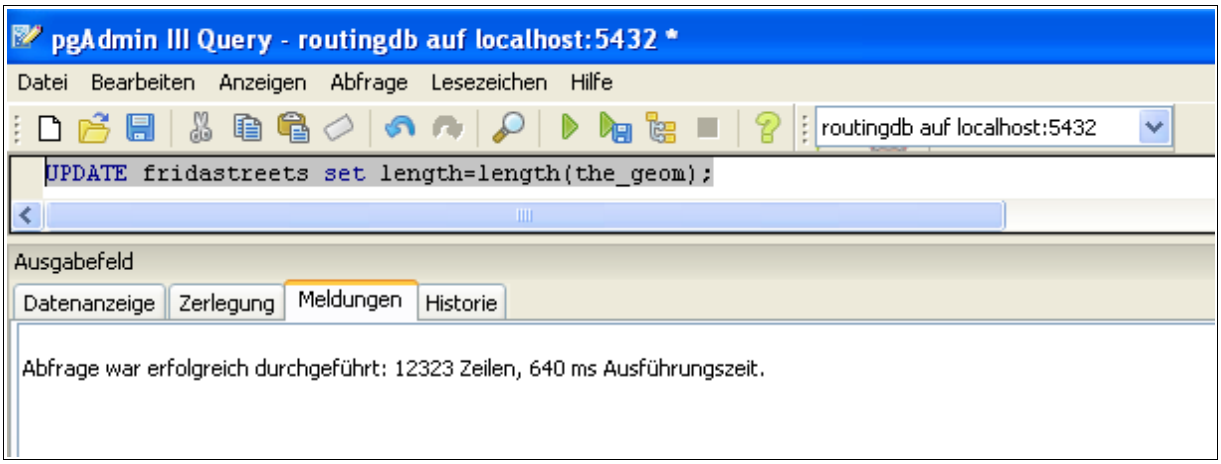
By starting the script a window should pop up on your screen in order to confirm the import (fig. 9). In this case all 12323 entries are verified.

Fig. 9: Creating start- and endpoints of the streets



All right! The geometries are in the database, time to get yourself a cup of coffee :-).
Afterwards we have to get the length-values.
This should be easy if you use the following SQL-command in the routingdb-database (fig. 10):
UPDATE fridastreets set length=length(the_geom);

Fig. 10: length-calculation

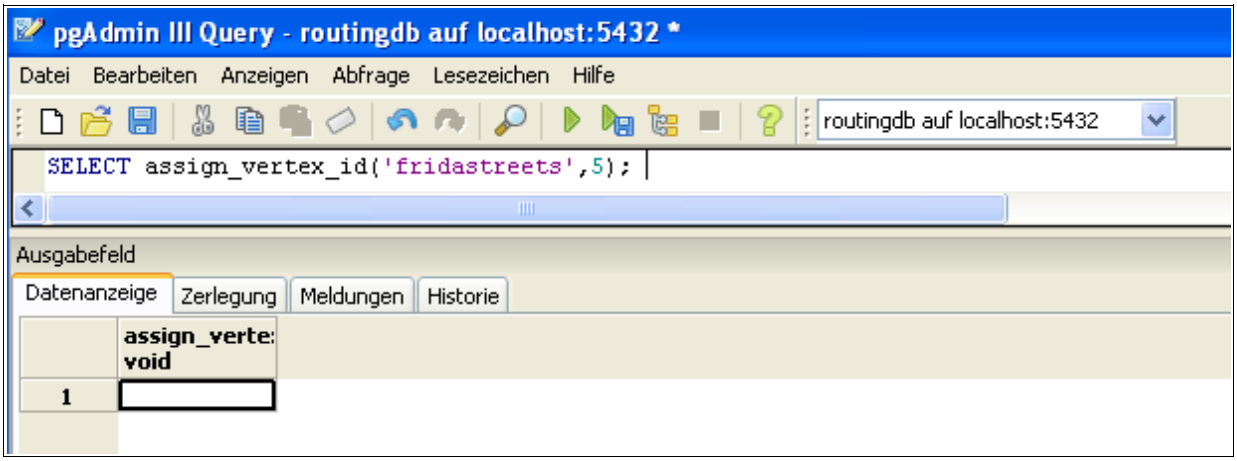


One thing is still missing...
In order to calculate the values for “source” and “target” we can use a predefined function:

SELECT assign_vertex_id('fridastreets', 5);

The number is variable. The number 5 represents a distance-room (5 meters), in which the nodes get the same vertexid.
However, that function expects the column-names to be „source_id“ & „target_id“ instead of „source“ or „target“. Of course we could modify the function but it`s easier to just rename the columns and start the function afterwards. It takes a while until this is finished and all entries are verified.

Fig. 11: Source/target-values calculating

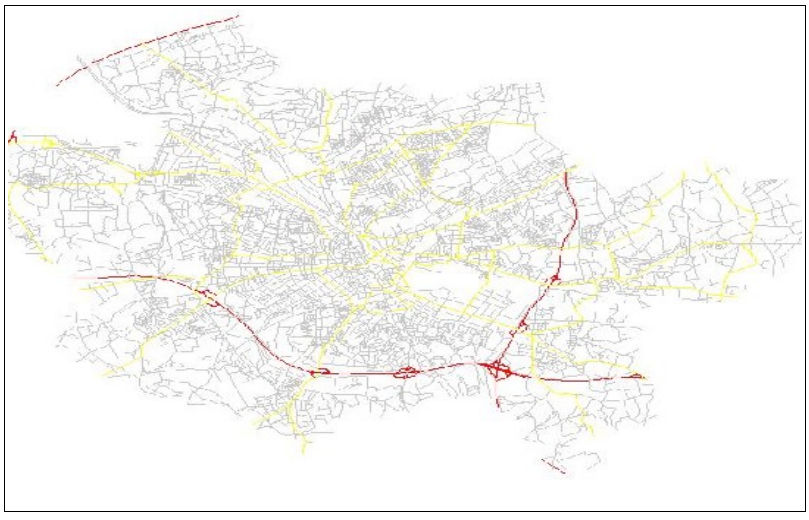


In the next step we rename source_id „source“ and „target_id“ „target“.
Ok, next thing we need is a PHP/Mapscript-Script and an adequate mapfile.
You can download this at: http://files.orkney.jp/pgrouting/sample/pgRouting-sampleapp.tar.bz.
We changed the data and adapted it to our geodata. They exist in the directory as “routing_os.map” or “phtmls/routing_os_frida.phtml” respectively.
The file “routing_os.map” is quite simple.
By default the Frida-data is visualized like this:

```
 LAYER
  NAME "roads"
  TYPE LINE
 CONNECTION "user=postgres password=postgres dbname=routingdb host=localhost port=5432"
 CONNECTIONTYPE postgis
 DATA "the_geom from fridastreets"

 STATUS DEFAULT
#LABELITEM 'strname'
CLASSITEM    'strtypid'
     CLASS
          EXPRESSION '1'
          STYLE
          COLOR          255 0 0
          END
     END
    CLASS
               EXPRESSION '3'
               STYLE
                    COLOR       255 255 0
               END
    END
    CLASS
               EXPRESSION /./
               STYLE
                    COLOR       200 200 200
               END
    END
END
```

Fig. 12: Visualisation of Frida-streets in the UMN MapServer



The Layer for the visualisation ot the route is called „path“.

```
 LAYER
    NAME "path"
    CONNECTION "user=postgres password=postgres dbname=frida host=localhost port=5432"
    CONNECTIONTYPE postgis
   STATUS ON
    TYPE LINE
    CLASS
     NAME "path"
         STYLE
         SYMBOL 'circle'
COLOR 255 0 0
SIZE 8
         END
    END
END
```

This layer is activated via PHP/Mapscript.
Take a look at the source-code of “routing_os_frida.phtml”.
By using this code a static extent is defined which can be changed with the variable $delta:

```
$delta=0;
$map_file=MAPFILE;
$map=ms_newMapObj($map_file);

$l=$map->getLayerByName("path");
if($l) {
  if($l && $start!=0 && $end!=0) {
$cx1=3429000;
$cy1=5787000;
$cx2=3444000;
$cy2=5800000;

   if($cx1!=0 && $cy1!=0 && $cx2!=0 && $cy2!=0 &&
     $cx1!=$cx2 && $cy1!=$cy2) {
    $minx = min($cx1,$cx2)-$delta;
    $miny = min($cy1,$cy2)-$delta;
    $maxx = max($cx1,$cx2)+$delta;
    $maxy = max($cy1,$cy2)+$delta;

    $map->setextent($minx,$miny,$maxx,$maxy);
```

Important is the call of the function „shortest_path_astar2_as_geometry_internal_id“
(which will only work, if the correct data-structure is in the table).

```
  $ll_x = $rectobj->minx;
    $ll_y = $rectobj->miny;
    $ur_x = $rectobj->maxx;
    $ur_y = $rectobj->maxy;

    $sql="the_geom from (select gid, the_geom from ".
        "shortest_path_astar2_as_geometry_internal_id('fridastreets', ".
        $start.", ".$end.", ".$ll_x.", ".$ll_y.", ".$ur_x.", ".
        $ur_y.")) as g using unique gid using SRID=-1";

    $l->set('data', $sql);
    $l->set('status', MS_ON);
```

The function itself is defined in the file “routing_postgis.sql” which we have written in the database.
Fundamental is the defining of the starting and ending points.
This works with numeric values in the formula:

```
<select name=start>
<option value=0 >W&auml;hle....</option>
<option value=7649 >Dom</option>
<option value=291 >Im Hone</option>
<option value=7750 >Kolpingstrasse</option>
<option value=7313 >Martinistr.</option>
```

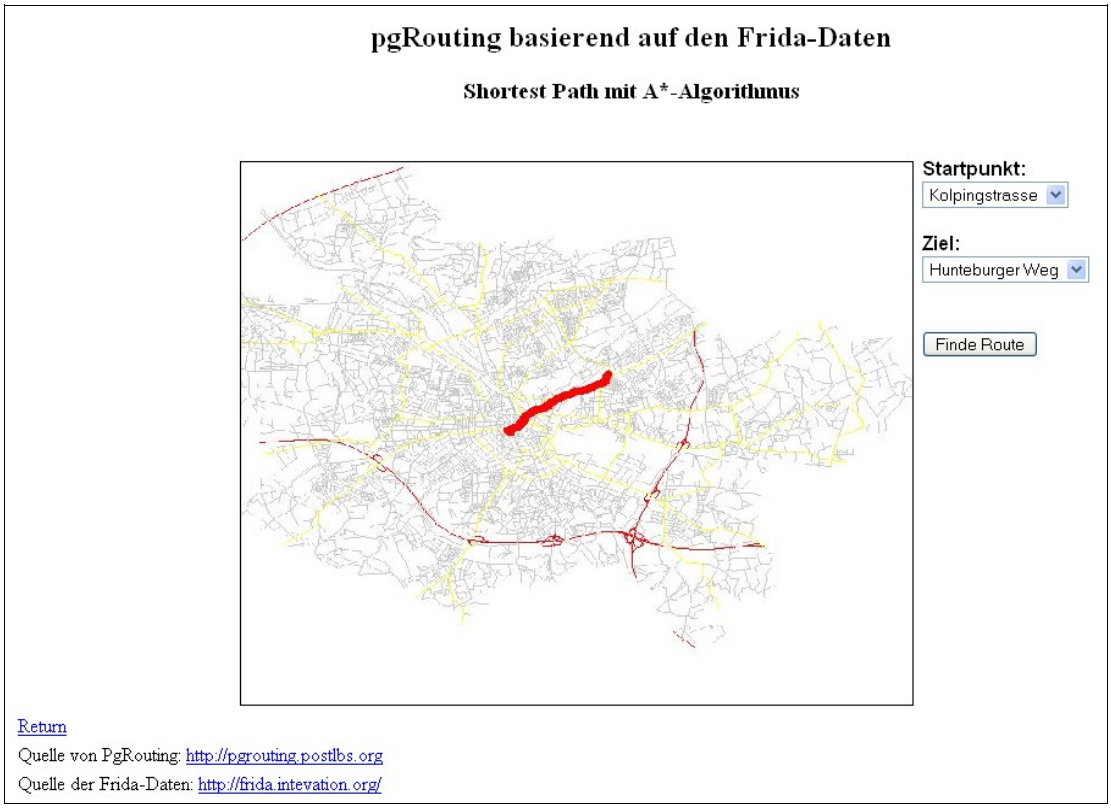
Attention: The values do not represent the gid in the table but the values of the source and the target-column (fig. 13).

Fig. 13: Source/target-values of Kolpingstreet

gid	strname	x1	y1	x2	y2	length	source	target
4778	Kolpingstr.	3435001.86	5793521.07	3434991.48	5793539.23	20.917217788298	7750	7751

After that you just have to choose two points from the application. Via the function shortest_path_astar2_as_geometry_internal_id the appropiate route is created „on_the_fly“ and visualized as layer „path“ in the mapfile (fig. 14).

Fig. 14: Route with pgRouting created



For further question concerning this topic, please use the forum:
http://pgrouting.postlbs.org

With best regards, Kai Behncke und Florian Thürkow