FOSS4G 2007 - Victoria

Web-based Routing: An Introduction to pgRouting with OpenLayers[1]

Presenters:[2]

- Claude Philipona (Camptocamp SA)
- Daniel Kastl (Orkney, Inc.)

Time & Location:

- Room Sidney
- Monday, September 24 - 13:00 to 16:00

Introduction

pgRouting requirements

To install/compile pgRouting the following applications/libraries are required:

- pgRouting source code[3]
- C and C++ compilers
- PostgreSQL version >= 8.0
- PostGIS version >= 1.0
- The Boost Graph Library (BGL). Version >= 1.33 which contains the astar.hpp[4]
- The Genetic Algorithm Utility Library (GAUL)[5]
- Computational Geometry Algorithms Library (CGAL) version >= 3.2[6]

This installation has been done on Xubuntu 7.04 and is provided as a VMware image.

Workplace access

- Start VM Player and load the workshop virtual machine
- If root access is required, use

```
username: pgrouting
password: foss4g
```

1 An online version of the pgRouting part of the workshop handout is available at http://pgrouting.postlbs.org/wiki/WorkshopFOSS4G2007
2 With extensive assistence of Frédéric Junod and Anton Patrushev
3 http://pgrouting.postlbs.org
4 http://www.boost.org/libs/graph/doc/index.html
5 http://gaul.sourceforge.net
6 http://www.cgal.org

PostgreSQL / pgRouting

- Open terminal window

Create a routing database

We will create a database "routing" and load the PostGIS and pgRouting functions.

```
createdb -U postgres -E UNICODE routing
createlang -U postgres plpgsql routing

cd /usr/share/postgresql/8.2/contrib/
psql -U postgres -f postgis-1.2.1/lwpostgis.sql routing
psql -U postgres -f postgis-1.2.1/spatial_ref_sys.sql routing

psql -U postgres -f routing.sql routing
psql -U postgres -f routing_wrappers.sql routing
```

Load routing data

The data we use for routing is the GeoBase "National Road Network" dataset for British Columbia (nrn_rrn_bc_shp_en.zip)[7]. It is available in shape file format and can be loaded into PostgreSQL using "shp2pgsql".

A long list of attributes comes with the road network data, but most of them do not contain data. For this routing workshop we will only keep "gid", "id" and "the_geom".[8]

```
    Column    |       Type        | Modifiers
--------------+-------------------+-----------
 gid          | integer           |
 id           | integer           |
 the_geom     | geometry          |
```

- Import a clip (Victoria Downtown area) of the British Columbia road network data.[9]

```
psql -U postgres routing
\i /home/pgrouting/RoutingData/victoria.sql
```

- Add the geometry column (in case we want to reproject the data later):

```
ALTER TABLE victoria RENAME COLUMN the_geom TO geom;
SELECT AddGeometryColumn('victoria','the_geom',54004,'MULTILINESTRING',2);
UPDATE victoria SET the_geom=geom;
ALTER TABLE victoria DROP COLUMN geom;
```

7 http://geobase.ca/geobase/en/data/nrn/index.html
8 To overlay the route with Google Maps the data has been reprojected to EPSG 54004
9 A table named "victoria" will be created automatically.
The network extent is "-13737893, 6141906, -13728396, 6151394"

Prepare routing table for Dijkstra

- Add source, target and length column

```
ALTER TABLE victoria ADD COLUMN source integer;
ALTER TABLE victoria ADD COLUMN target integer;
ALTER TABLE victoria ADD COLUMN length double precision;
```

- Create network topology

```
SELECT assign_vertex_id('victoria', 0.001, 'the_geom', 'gid');
UPDATE victoria SET length = length(the_geom);
```

- Create indexes for source, target and geometry column

```
CREATE INDEX source_idx ON victoria(source);
CREATE INDEX target_idx ON victoria(target);
CREATE INDEX geom_idx ON victoria USING GIST(the_geom GIST_GEOMETRY_OPS);
```

- Run Shortest Path Dijkstra query

```
shortest_path( sql text,
               source_id integer,
               target_id integer,
               directed boolean,
               has_reverse_cost boolean )
```

(Source and target IDs are node IDs.)

  - Shortest path core function

```
SELECT * FROM shortest_path('
            SELECT gid as id,
                   source::integer,
                   target::integer,
                   length::double precision as cost
                  FROM victoria',
            25793, 128002, false, false);
```

```
 vertex_id | edge_id |       cost
-----------+---------+------------------
     25793 |   13956 | 142.820280785167
     23466 |   16779 | 134.657801293034
      5246 |    2657 | 119.354409882875
       ... |     ... | ...
```

  - Wrapper function without bounding box

```
SELECT gid, AsText(the_geom) AS the_geom
      FROM dijkstra_sp('victoria', 25793, 128002);
```

```
  gid   |                                the_geom
--------+-------------------------------------------------------------------------
  13956 | MULTILINESTRING((-13732937.9317993 6145945.06953108, ... ))
  16779 | MULTILINESTRING((-13732966.0956304 6145813.38991207, ... ))
   2657 | MULTILINESTRING((-13732986.2221944 6145695.7446972, ... ))
    ... | ...
```

  - Wrapper function with bounding box

```
SELECT gid, AsText(the_geom) AS the_geom
      FROM dijkstra_sp_delta('victoria', 25793, 128002, 3000);
```

Prepare routing table for A-Star

- Add x1, y1 and x2, y2 column

```
ALTER TABLE victoria ADD COLUMN x1 double precision;
ALTER TABLE victoria ADD COLUMN y1 double precision;
ALTER TABLE victoria ADD COLUMN x2 double precision;
ALTER TABLE victoria ADD COLUMN y2 double precision;

UPDATE victoria SET x1 = x(startpoint(the_geom));
UPDATE victoria SET y1 = y(startpoint(the_geom));
UPDATE victoria SET x2 = x(endpoint(the_geom));
UPDATE victoria SET y2 = y(endpoint(the_geom));
```

- Run Shortest Path A-Star query

```
shortest_path_astar( sql text,
                source_id integer,
                target_id integer,
                directed boolean,
                has_reverse_cost boolean )
```

(Source and target IDs are node IDs.)

  - A-Star core function

```
SELECT * FROM shortest_path_astar('
            SELECT gid as id,
                   source::integer,
                   target::integer,
                   length::double precision as cost,
                   x1, y1, x2, y2
                  FROM victoria',
            25793, 128002, false, false);
```

  - Wrapper function with bounding box

```
SELECT gid, AsText(the_geom) AS the_geom
      FROM astar_sp_delta('victoria', 25793, 128002, 3000);
```

Prepare routing table for Shooting-Star

- Add x1, y1 and x2, y2 column

```
ALTER TABLE victoria ADD COLUMN reverse_cost double precision;
UPDATE victoria SET reverse_cost = length;

ALTER TABLE victoria ADD COLUMN to_cost double precision;
ALTER TABLE victoria ADD COLUMN rule text;
```

- Run Shortest Path Shooting-Star query

```
shortest_path_shooting_star( sql text,
              source_id integer,
              target_id integer,
              directed boolean,
              has_reverse_cost boolean )
```

(Source and target IDs are edge IDs.)

  - Shooting-Star core function

```
SELECT * FROM shortest_path_shooting_star('
           SELECT gid as id,
                  source::integer,
                  target::integer,
                  length::double precision as cost,
                  x1, y1, x2, y2,
                  rule, to_cost
                FROM victoria',
           36339, 22921, false, false);
```

  - Wrapper function with bounding box

```
SELECT gid, AsText(the_geom) AS the_geom
      FROM shootingstar_sp('victoria', 36339, 22921, 5000,
                           'length',true,true);
```

OpenLayers routing map

- Open text editor (i.e. Gedit, Mousepad)
- Google key: you don't need a valid key if you're working on localhost
- Note: these examples are **not** compatible with the latest stable version of OpenLayers (version 2.5). If you want to update the code, please look at: http://trac.openlayers.org/wiki/SphericalMercator

Starting page (00.index.html)

- Create a basic html:
  - include OpenLayers and Google maps API javascript files
  - create an empty div to receive the OpenLayers map
  - create an empty `init` function. The function will be called on page load (see onload in the body tag)

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <style type="text/css">
        #map {
            width: 800px;
            height: 512px;
            border: 1px solid black;
        }
    </style>
    <script src="./OpenLayers-google/lib/OpenLayers.js"></script>
    <script src="http://maps.google.com/maps?file=api&v=2&key=foobar"></script>

    <script type="text/javascript">
        function init() { }
    </script>
  </head>

  <body onload="init()">
    <div id="map"></div>
  </body>
</html>
```

Design basic map layers (01.index.html)

- Create an OpenLayers map object
- Create and Controls to the maps (layer switcher and mouse position)
- Create a Google Satellite layer and add it to the map
- Recenter the map to a specified extent

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <style type="text/css">
        #map {
            width: 800px;
            height: 512px;
            border: 1px solid black;
        }
    </style>
    <script src="./OpenLayers-google/lib/OpenLayers.js"></script>
    <script src="http://maps.google.com/maps?file=api&v=2&key=foobar"></script>

    <script type="text/javascript">
        // global variables
        var map;

        function init() {
            map = new OpenLayers.Map('map', {projection: "EPSG: 54004",
                                             units: "m",
                                             maxResolution: 156543.0339,
                                             maxExtent: new OpenLayers.Bounds(-20037508,
                                                                              -136554022,
                                                                              20037508,
                                                                              136554022)

                                            });
            map.addControl(new OpenLayers.Control.LayerSwitcher());
            map.addControl(new OpenLayers.Control.MousePosition());

            // create and add layers to the map
            var satellite = new OpenLayers.Layer.GoogleMercator("Google Satellite",
                                                                {type: G_NORMAL_MAP});

            map.addLayers([satellite]);

            // set default position
            map.zoomToExtent(new OpenLayers.Bounds(-13737893,
                                                   6141906,
                                                   -13728396,
                                                   6151394));
        }
    </script>
  </head>

  <body onload="init()">
    <div id="map"></div>
  </body>
</html>
```

Create a Vector layer and draw a polygon on it (02.index.html)

- Create a custom style for the vector layer
- Create a Vector layer and add it to the map
- Create a WKT parser and read a polygon, this create an `OpenLayers.Geometry.Polygon` object
- Create a `OpenLayers.Feature.Vector` with the polygon and add it to the vector layer

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <style type="text/css">
        #map {
            width: 800px;
            height: 512px;
            border: 1px solid black;
        }
    </style>
    <script src="./OpenLayers-google/lib/OpenLayers.js"></script>
    <script src="http://maps.google.com/maps?file=api&v=2&key=foobar"></script>

    <script type="text/javascript">
         var result_style = OpenLayers.Util.applyDefaults({
             strokeWidth: 3,
             strokeColor: "#ff0000",
             fillOpacity: 0
         }, OpenLayers.Feature.Vector.style['default']);

        // global variables
        var map, parser, result;

        function init() {
            map = new OpenLayers.Map('map', {projection: "EPSG: 54004",
                                             units: "m",
                                             maxResolution: 156543.0339,
                                             maxExtent: new OpenLayers.Bounds(-20037508,
                                                                              -136554022,
                                                                              20037508,
                                                                              136554022)
                                            });
            map.addControl(new OpenLayers.Control.LayerSwitcher());
            map.addControl(new OpenLayers.Control.MousePosition());

            // create and add layers to the map
            var satellite = new OpenLayers.Layer.GoogleMercator("Google Satellite",
                                                                {type: G_NORMAL_MAP});
            result = new OpenLayers.Layer.Vector("Routing results",
                                                 {style: result_style});

            map.addLayers([satellite, result]);

            // create a feature from a wkt string and add it to the map
            parser = new OpenLayers.Format.WKT();
            var wkt = "POLYGON((-13737893 6151394, -13737893 6141906, -13728396 6141906,
-13728396 6151394, -13737893 6151394))";
            var geometry = parser.read(wkt);
            var feature = new OpenLayers.Feature.Vector(geometry);
            result.addFeatures([feature]);

            // set default position
            map.zoomToExtent(new OpenLayers.Bounds(-13737893,6141906,-13728396,6151394));
        }
    </script>
  </head>
  <body onload="init()">
    <div id="map"></div>
  </body>
</html>
```

"set start/end point" tool (03.index.html)

- Create two more vector layers with custom style
- Create the two drawing tools (`OpenLayers.Control.DrawFeature`), those tools can only draw point (`OpenLayers.Handler.Point`)
- The toggleControl function set the active tool

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <style type="text/css">
        #map {
            width: 800px;
            height: 512px;
            border: 1px solid black;
        }
    </style>
    <script src="./OpenLayers-google/lib/OpenLayers.js"></script>
    <script src="http://maps.google.com/maps?file=api&v=2&key=foobar"></script>

    <script type="text/javascript">

        var start_style = OpenLayers.Util.applyDefaults({
            externalGraphic: "start.png",
            graphicWidth: 18,
            graphicHeight: 26,
            graphicYOffset: -26,
            graphicOpacity: 1
        }, OpenLayers.Feature.Vector.style['default']);

        var stop_style = OpenLayers.Util.applyDefaults({
            externalGraphic: "stop.png",
            graphicWidth: 18,
            graphicHeight: 26,
            graphicYOffset: -26,
            graphicOpacity: 1
        }, OpenLayers.Feature.Vector.style['default']);

         var result_style = OpenLayers.Util.applyDefaults({
             strokeWidth: 3,
             strokeColor: "#ff0000",
             fillOpacity: 0
         }, OpenLayers.Feature.Vector.style['default']);

        // global variables
        var map, parser, start, stop, result, controls;

        function init() {
            map = new OpenLayers.Map('map', {projection: "EPSG: 54004",
                                             units: "m",
                                             maxResolution: 156543.0339,
                                             maxExtent: new OpenLayers.Bounds(-20037508,
                                                                              -136554022,
                                                                              20037508,
                                                                              136554022)
                                            });
            map.addControl(new OpenLayers.Control.LayerSwitcher());
            map.addControl(new OpenLayers.Control.MousePosition());

            // create and add layers to the map
            var satellite = new OpenLayers.Layer.GoogleMercator("Google Satellite",
                                                                {type: G_NORMAL_MAP});
            start = new OpenLayers.Layer.Vector("Start point", {style: start_style});
            stop = new OpenLayers.Layer.Vector("End point", {style: stop_style});
            result = new OpenLayers.Layer.Vector("Routing results",
                                                 {style: result_style});

            map.addLayers([satellite, start, stop, result]);
```

```
            // create a feature from a wkt string and add it to the map
            parser = new OpenLayers.Format.WKT();
            var wkt = "POLYGON((-13737893 6151394, -13737893 6141906, -13728396 6141906,
-13728396 6151394, -13737893 6151394))";
            var geometry = parser.read(wkt);
            var feature = new OpenLayers.Feature.Vector(geometry);
            result.addFeatures([feature]);

            // set default position
            map.zoomToExtent(new OpenLayers.Bounds(-13737893,6141906,-13728396,6151394));

            // controls
            controls = {
              start: new OpenLayers.Control.DrawFeature(start, OpenLayers.Handler.Point),
              stop: new OpenLayers.Control.DrawFeature(stop, OpenLayers.Handler.Point)
            }
            for (var key in controls) {
                map.addControl(controls[key]);
            }
        }

        function toggleControl(element) {
            for (key in controls) {
                if (element.value == key && element.checked) {
                    controls[key].activate();
                } else {
                    controls[key].deactivate();
                }
            }
        }
    </script>
  </head>

  <body onload="init()">
    <div id="map"></div>

    <ul>
      <li>
        <input type="radio" name="control" id="noneToggle"
               onclick="toggleControl(this);" checked="checked" />
        <label for="noneToggle">navigate</label>
      </li>
      <li>
        <input type="radio" name="control" value="start" id="startToggle"
               onclick="toggleControl(this);" />
        <label for="startToggle">set start point</label>
      </li>
      <li>
        <input type="radio" name="control" value="stop" id="stopToggle"
               onclick="toggleControl(this);" />
        <label for="stopToggle">set stop point</label>
      </li>
    </ul>

  </body>
</html>
```

Customize the tools (04.index.html)

- Create a custom `OpenLayers.Handler.Point` handle: SinglePoint. This handler only draw one point on the layer
- Add a method selector
- Create the “compute” button with the callback function (empty for now)

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <style type="text/css">
        #map { width: 800px; height: 512px; border: 1px solid black; }
    </style>
    <script src="./OpenLayers-google/lib/OpenLayers.js"></script>
    <script src="http://maps.google.com/maps?file=api&v=2&key=foobar"></script>

    <script type="text/javascript">

        var SinglePoint = OpenLayers.Class.create();
        SinglePoint.prototype = OpenLayers.Class.inherit(OpenLayers.Handler.Point, {
            createFeature: function(evt) {
                this.control.layer.removeFeatures(this.control.layer.features);
                OpenLayers.Handler.Point.prototype.createFeature.apply(this, arguments);
            }
        });

        var start_style = OpenLayers.Util.applyDefaults({
            externalGraphic: "start.png",
            graphicWidth: 18,
            graphicHeight: 26,
            graphicYOffset: -26,
            graphicOpacity: 1
        }, OpenLayers.Feature.Vector.style['default']);

        var stop_style = OpenLayers.Util.applyDefaults({
            externalGraphic: "stop.png",
            graphicWidth: 18,
            graphicHeight: 26,
            graphicYOffset: -26,
            graphicOpacity: 1
        }, OpenLayers.Feature.Vector.style['default']);

         var result_style = OpenLayers.Util.applyDefaults({
             strokeWidth: 3,
             strokeColor: "#ff0000",
             fillOpacity: 0
         }, OpenLayers.Feature.Vector.style['default']);

        // global variables
        var map, parser, start, stop, result, controls;

        function init() {
            map = new OpenLayers.Map('map', {projection: "EPSG: 54004",
                                             units: "m",
                                             maxResolution: 156543.0339,
                                             maxExtent: new OpenLayers.Bounds(-20037508,
                                                                              -136554022,
                                                                              20037508,
                                                                              136554022)
                                            });
            map.addControl(new OpenLayers.Control.LayerSwitcher());
            map.addControl(new OpenLayers.Control.MousePosition());

            // create and add layers to the map
            var satellite = new OpenLayers.Layer.GoogleMercator("Google Satellite",
                                                                {type: G_NORMAL_MAP});
            start = new OpenLayers.Layer.Vector("Start point", {style: start_style});
            stop = new OpenLayers.Layer.Vector("End point", {style: stop_style});
            result = new OpenLayers.Layer.Vector("Routing results",
```

```
                                                {style: result_style});

            map.addLayers([satellite, start, stop, result]);

            // create a feature from a wkt string and add it to the map
            parser = new OpenLayers.Format.WKT();
            var wkt = "POLYGON((-13737893 6151394, -13737893 6141906, -13728396 6141906,
-13728396 6151394, -13737893 6151394))";
            var geometry = parser.read(wkt);
            var feature = new OpenLayers.Feature.Vector(geometry);
            result.addFeatures([feature]);

            // set default position
            map.zoomToExtent(new OpenLayers.Bounds(-13737893,
                                                   6141906,
                                                   -13728396,
                                                   6151394));

            // controls
            controls = {
              start: new OpenLayers.Control.DrawFeature(start, SinglePoint),
              stop: new OpenLayers.Control.DrawFeature(stop, SinglePoint)
            }
            for (var key in controls) {
                map.addControl(controls[key]);
            }
        }

        function toggleControl(element) {
            for (key in controls) {
                if (element.value == key && element.checked) {
                    controls[key].activate();
                } else {
                    controls[key].deactivate();
                }
            }
        }

        function compute() { }

    </script>
  </head>
  <body onload="init()">
    <div id="map"></div>

    <ul>
      <li>
        <input type="radio" name="control" id="noneToggle"
               onclick="toggleControl(this);" checked="checked" />
        <label for="noneToggle">navigate</label>
      </li>
      <li>
        <input type="radio" name="control" value="start" id="startToggle"
               onclick="toggleControl(this);" />
        <label for="startToggle">set start point</label>
      </li>
      <li>
        <input type="radio" name="control" value="stop" id="stopToggle"
               onclick="toggleControl(this);" />
        <label for="stopToggle">set stop point</label>
      </li>
    </ul>

    <select id="method">
      <option value="SPD">Shortest Path Dijkstra - undirected (BBox)</option>
      <option value="SPA">Shortest Path A Star - undirected</option>
      <option value="SPS">Shortest Path Shooting Star</option>
    </select>
    <button onclick="compute()">Calculate Route</button>

  </body>
</html>
```

Ask for a route (05.index.html)

- Code the compute function:
  - Create a request object (result)
  - Use the OpenLayers LoadUrl function to call the php script via AJAX
  - The displayRoute function will be called with the output of the php script

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <style type="text/css">
        #map { width: 800px; height: 512px; border: 1px solid black; }
    </style>
    <script src="./OpenLayers-google/lib/OpenLayers.js"></script>
    <script src="http://maps.google.com/maps?file=api&v=2&key=foobar"></script>

    <script type="text/javascript">

        /* ... */

        function init() {
            /* ... */
        }

        function toggleControl(element) {
            /* ... */
        }

        function compute() {
            var startPoint = start.features[0];
            var stopPoint = stop.features[0];

            if (startPoint && stopPoint) {
                var result = {
                    startpoint: startPoint.geometry.x + ' ' + startPoint.geometry.y,
                    finalpoint: stopPoint.geometry.x + ' ' + stopPoint.geometry.y,
                    method: OpenLayers.Util.getElement('method').value,
                    region: "victoria",
                    srid: "54004"
                };
                OpenLayers.loadURL("./ax_routing.php",
                                   OpenLayers.Util.getParameterString(result),
                                   null,
                                   displayRoute);
            }
        }

        function displayRoute(response) {
            if (response && response.responseXML) {
                alert(response.responseXML);
            }
        }

    </script>
  </head>
  <body onload="init()">
    <div id="map"></div>

    <!-- ... -->

    <select id="method">
      <option value="SPD">Shortest Path Dijkstra - undirected (BBox)</option>
      <option value="SPA">Shortest Path A Star - undirected</option>
      <option value="SPS">Shortest Path Shooting Star</option>
    </select>
    <button onclick="compute()">Calculate Route</button>

  </body>
</html>
```

Show the routing result on the map (06.index.html)

- code the displayRoute function:
  - Erase the previous result
  - Get all the <edge> tag in the XML document
  - Parse the <wkt> content of each edge and add them to the result layer

```
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <style type="text/css">
        #map { width: 800px; height: 512px; border: 1px solid black; }
    </style>
    <script src="./OpenLayers-google/lib/OpenLayers.js"></script>
    <script src="http://maps.google.com/maps?file=api&v=2&key=foobar"></script>

    <script type="text/javascript">

        /* ... */

        function init() {
            /* ... */
        }

        function toggleControl(element) {
            /* ... */
        }

        function compute() {
            /* ... */
        }

        function displayRoute(response) {
            if (response && response.responseXML) {
                // erase the previous results
                result.removeFeatures(result.features);

                // parse the features
                var edges = response.responseXML.getElementsByTagName('edge');
                var features = [];
                for (var i = 0; i < edges.length; i++) {
                    var g = parser.read(edges[i].getElementsByTagName('wkt')
[0].textContent);
                    features.push(new OpenLayers.Feature.Vector(g));
                }
                result.addFeatures(features);
            }
        }

    </script>
  </head>

  <body onload="init()">
    <div id="map"></div>

    <!-- ... -->

    <select id="method">
      <option value="SPD">Shortest Path Dijkstra - undirected (BBox)</option>
      <option value="SPA">Shortest Path A Star - undirected</option>
      <option value="SPS">Shortest Path Shooting Star</option>
    </select>
    <button onclick="compute()">Calculate Route</button>

  </body>
</html>
```

Routing PHP script with XML output

- Open text editor (i.e. Gedit, Mousepad)

We will use a PHP script to make the routing query and send the result back to the web client.

The following steps are needed:

- Retrieve the start and end point coordinates.
- Find the closest edge to start/end point.
- Take either the start or end vertex of this edge (for Dijkstra/ A-Star) or the complete edge (Shooting-Star) as start of the route and end respectively.
- Make the Shortest Path database query.
- Transform the query result to XML and send it back to the web client.

PHP page template (basics)

```
<?php

  // Database connection settings
  define("PG_DB"  , "routing");
  define("PG_HOST", "localhost");
  define("PG_USER", "postgres");
  define("PG_PORT", "5432");
  define("TABLE",   "victoria");

  $counter = $pathlength = 0;

  // Retrieve start point
  $start = split(' ',$_REQUEST['startpoint']);
  $startPoint = array($start[0], $start[1]);

  // Retrieve end point
  $end = split(' ',$_REQUEST['finalpoint']);
  $endPoint = array($end[0], $end[1]);

  /* ... */

?>
```

Select closest edge

Usually the start and end point, which we retrieved from the client, is not the start or end vertex of an edge.

It is more convenient to look for the closest edge than for the closest vertex, because Shooting Star algorithm is "edge-based". For "vertex-based" algorithms (Dijkstra, A-Star) we can choose arbitrary start or end of the selected edge.[10]

```
// Find the nearest edge
$startEdge = findNearestEdge($startPoint);
$endEdge   = findNearestEdge($endPoint);

// FUNCTION findNearestEdge
function findNearestEdge($lonlat) {

  // Connect to database
  $con = pg_connect("dbname=".PG_DB." host=".PG_HOST." user=".PG_USER);

  $sql = "SELECT gid, source, target, the_geom,
                distance(the_geom, GeometryFromText(
               'POINT(".$lonlat[0]." ".$lonlat[1].")', 54004)) AS dist
          FROM ".TABLE."
          WHERE the_geom && setsrid(
                'BOX3D(".($lonlat[0]-200)."
                       ".($lonlat[1]-200).",
                       ".($lonlat[0]+200)."
                       ".($lonlat[1]+200).")'::box3d, 54004)
          ORDER BY dist LIMIT 1";

  $query = pg_query($con,$sql);

  $edge['gid']      = pg_fetch_result($query, 0, 0);
  $edge['source']   = pg_fetch_result($query, 0, 1);
  $edge['target']   = pg_fetch_result($query, 0, 2);
  $edge['the_geom'] = pg_fetch_result($query, 0, 3);

  // Close database connection
  pg_close($con);

  return $edge;
}
```

10 Later we can then create a substring of that edge and we can check, if we pass the same edge again on our route. If we pass it again, then we remove the edge from the path.

Routing query

```
// Select the routing algorithm
switch($_REQUEST['method']) {
```

- For Shortest Path Dijkstra

```
 case 'SPD' : // Shortest Path Dijkstra

   $sql = "SELECT rt.gid, AsText(rt.the_geom) AS wkt,
                length(rt.the_geom) AS length, ".TABLE.".id
           FROM ".TABLE.",
               (SELECT gid, the_geom
                   FROM dijkstra_sp_delta(
                       '".TABLE."',
                       ".$startEdge['source'].",
                       ".$endEdge['target'].",
                       3000)
                ) as rt
           WHERE ".TABLE.".gid=rt.gid;";
   break;
```

- For Shortest Path A-Star

```
 case 'SPA' : // Shortest Path A*

   $sql = "SELECT rt.gid, AsText(rt.the_geom) AS wkt,
                length(rt.the_geom) AS length, ".TABLE.".id
           FROM ".TABLE.",
               (SELECT gid, the_geom
                   FROM astar_sp_delta(
                       '".TABLE."',
                       ".$startEdge['source'].",
                       ".$endEdge['target'].",
                       3000)
                ) as rt
           WHERE ".TABLE.".gid=rt.gid;";
   break;
```

- For Shortest Path Shooting-Star

```
 case 'SPS' : // Shortest Path Shooting*

   $sql = "SELECT rt.gid, AsText(rt.the_geom) AS wkt,
                length(rt.the_geom) AS length, ".TABLE.".id
           FROM ".TABLE.",
               (SELECT gid, the_geom
                   FROM shootingstar_sp(
                       '".TABLE."',
                       ".$startEdge['gid'].",
                       ".$endEdge['gid'].",
                       3000, 'length', false, false)
                ) as rt
           WHERE ".TABLE.".gid=rt.gid;";
   break;

} // close switch

// Database connection and query
$dbcon = pg_connect("dbname=".PG_DB." host=".PG_HOST." user=".PG_USER);

$query = pg_query($dbcon,$sql);
```

XML output

OpenLayers allows to draw lines directly using WKT (Well-Known Text) format. This makes the XML output short and simple:

```
// Return route as XML
$xml  = '<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>'."\n";
$xml .= "<route>\n";

// Add edges to XML file
while($edge=pg_fetch_assoc($query)) {

  $pathlength += $edge['length'];

  $xml .= "\t<edge id='".++$counter."'>\n";
  $xml .= "\t\t<id>".$edge['id']."</id>\n";
  $xml .= "\t\t<wkt>".$edge['wkt']."</wkt>\n";
  $xml .= "\t\t<length>".round(($pathlength/1000),3)."</length>\n";
  $xml .= "\t</edge>\n";
}

$xml .= "</route>\n";

// Close database connection
pg_close($dbcon);


// Return routing result
header('Content-type: text/xml',true);
echo $xml;
```