

**WebGD: Framework
for Web-Based GIS/Database Applications**

by
Surya Halim

**Submitted to Oregon State University
In partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE
in Computer Science**

**Computer Science Department
Oregon State University
Corvallis, OR
June 2005**

Acknowledgements

I am very grateful to Dr. Toshimi Minoura for his support and guidance. On a personal note, I would like to acknowledge the support and love of my family and friends.

Abstract

We have developed a framework for Web-based GIS/database (WebGD) applications that allow users to *insert*, *query*, and *delete* data with map interfaces displayed by Web browsers. The framework uses such open source software packages as Minnesota MapServer, PostGIS, and PostgreSQL. With this framework, we can create the map interface of a new WebGD application only by providing *configuration files*. Those configuration files define the contents, appearances, and interactions of the *legend*, the *quickview mechanism*, and the *map function tools*. The map area needed for an application can be covered by multiple regions, each with its own set of map layers and region specific parameters such as a map projection and a map extent. Spatial references for different map regions can be dynamically switched.

1 Introduction

The Web-based GIS/database (WebGD) framework is a software framework for creating custom GIS applications rapidly. Traditionally, the map interface of an application that performs complex map operations had to be manually coded. The WebGD framework contains common code shared by many Web-based GIS applications. In addition to supporting map navigation, the framework allows *insertion*, *query*, and *deletion* of geographical features.

The WebGD framework uses the Internet map server developed at University of Minnesota (UMN MapServer) to generate map images. PostGIS, an extension to the PostgreSQL *object relational* database management system, is used to manipulate spatial data. These software tools are accessed through PHP-MapScript API.

In this report, we explain on how to create a WebGD application by using the WebGD framework, the organization of the WebGD framework, and the implementation details of the framework.

Section 2 provides an overview of a WebGD application. We describe the organization of the map interface of the Oregon Watershed Information System application in Section 3 and explain how to create such an application in Section 4. An overview of the WebGD framework is provided in Section 5. Finally, in Section 6 we discuss the implementation details of the WebGD framework.

2 Organization of a WebGD Application

The organization of a WebGD application is shown in Figure 2-1. A map operation that does not manipulate geometry features, i.e., *zoom-in*, *zoom-out*, or *panning*, is processed as follows.

1. The user action on the map is transmitted from the Web browser to the Web server as an HTTP GET or POST request.
2. The Web server activates a PHP script that handles the user action.
3. Inside the PHP script, various methods in PHP MapScript are called to prepare the new map parameters, such as the new map extent and the names of the layers to be displayed. The map drawing method in PHP MapScript is then called to instruct the MapServer to create the map image.
4. The MapServer requests data for features in a map layers whose data are stored in the PostgreSQL database.
5. The data for the features is returned.
6. The map image created is returned to the PHP script.
7. The HTML page generated by the PHP script is returned to the Web server.
8. The Web server transmits the HTML page including the new map image to the Web browser.

For a map operation that accesses or manipulates geometry features, i.e., *search by area*, *insert*, or *move point*, is processed as follows.

- a. Same as step 1 above.
- b. Same as step 2 above.
- c. The PHP script connects to the PostgreSQL database to perform a spatial operation with PostGIS-enabled SQL statements.
- d. The result of the spatial operation is returned to the PHP script by the SQL statements.
- e. Same as step 3 above.
- f. Same as step 4 above.
- g. Same as step 5 above.
- h. Same as step 6 above.

- i. Same as step 7 above.
- j. Same as step 8 above.

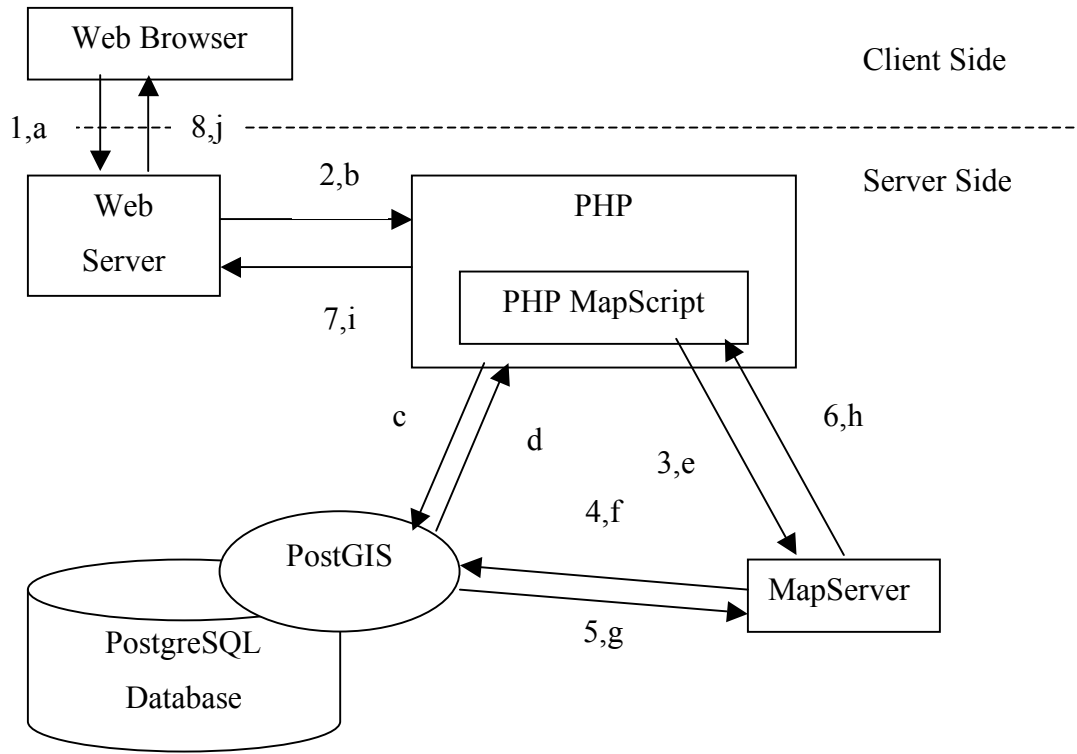


Figure 2-1: The organization of a WebGD application.

2.1 PHP

PHP (PHP: **H**ypertext **P**reprocessor) is a scripting language used for creating a dynamic website. This scripting language supports an extensive set of operations including those to access a database. PHP supports intuitive syntax for associative arrays, which are extensively used by the WebGD framework. PHP 4, which is used by the WebGD applications, also provides limited capabilities for object-oriented programming. PHP 5 provides full implementation of object-oriented programming features. More details can be found at the Web site for PHP [1].

2.2 PostgreSQL

PostgreSQL is an *object-relational* database management system (ORDBMS) used for storing both *spatial* and *non-spatial* data. Spatial data are stored in *well-known binary (WKB) format* as specified by the OpenGIS Consortium [2]. The types of the spatial data are POINT, MULTIPOINT, POLYGON, MULTIPOLYGON, LINESTRING, and MULTILINESTRING. More details can be found at the Web site for PostgreSQL [3].

2.3 PostGIS

PostGIS is an extension to PostgreSQL developed by Refractions Research [4] to perform spatial operations, such as computing intersections and unions, and testing containment properties among geographical features. More details can be found at the Web site for PostGIS [5].

2.5 PHP-MapScript

PHP-MapScript is a set of application programming modules created by DM Solutions [6] to use MapServer application with PHP. The information in the map configuration file is converted to a set of PHP objects. PHP-MapScript allows developers to query the values of the member variables of those objects to be queried with PHP object interface. More details can be found in the MapScript Documentation [7].

2.4 *University of Minnesota MapServer (UMN MapServer)*

UMN MapServer is an Internet map server, and it generates the map image to be displayed in a Web browser as specified in the map configuration file (map file) on the web browser. Parameters about the map are specified in a map file. More details can be found in the MapServer Documentation Project [8].

3 Interactive Map Interface

In this section we describe the Oregon Watershed Information System application that is built with the WebGD framework. With the map interface of this application, a user can obtain information on watersheds. In the following two subsections, we describe the layout of the map interface of the application and the operations supported by the map interface.

3.1 Layout of the Map Interface

The major components of the map interface are the map legend, the map display window, the reference map, the map tool set, the map scale bar, and the quickview mechanism as shown in Figure 3-1.

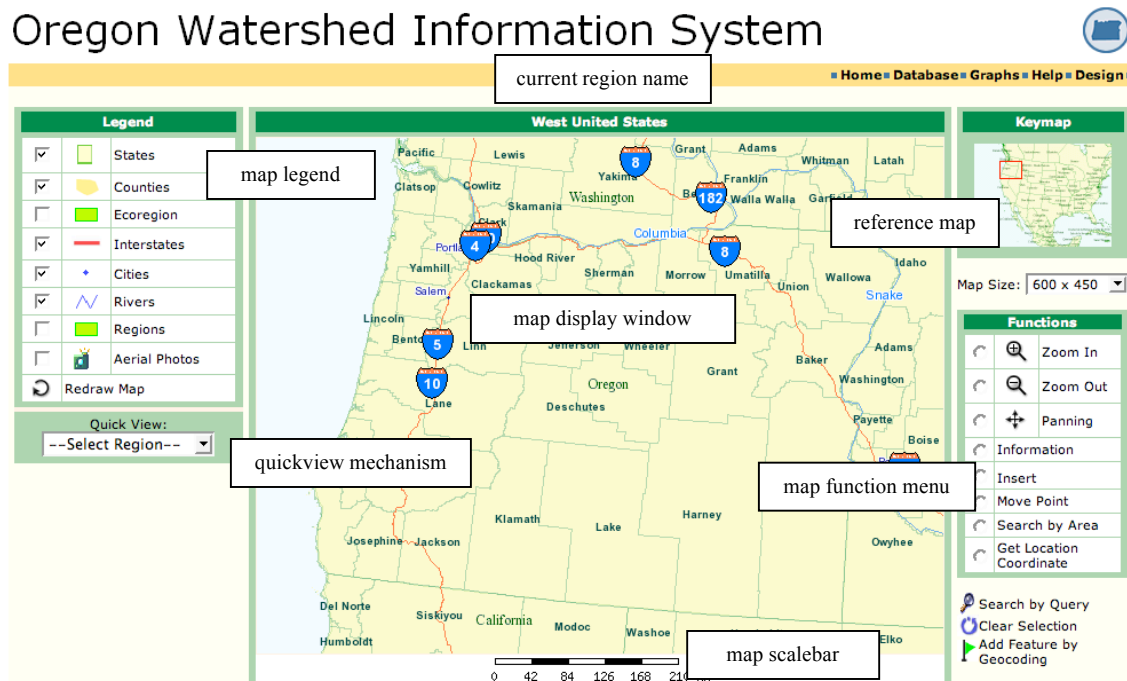


Figure 3-1: Map Interface

3.2 Using the Map Interface

We now describe each of the map interface components:

- **map display window**
This window is the place where the map image is rendered. The map image displayed is produced by UMN MapServer.
- **current region name**
This text shows the name of the current map region. The text is updated when the user performs a map operation that results in a change of the map region.
- **map legend**
This table lists all the layers that are available for the current map region. The user can turn on or off a layer by ticking the checkbox provided for it. The user can also set one active layer by clicking the name for that layer. The highlighted layer is active and spatial operations can be performed on it.
- **quickview mechanism**
This mechanism allows the user to zoom in/out directly to a location in the drop-down list.
- **reference map**
The smaller map shows the whole area of the current region. The current area being displayed in the map window is indicated by a shaded rectangle or by a crosshair marker.
- **map function menu**
The user can select a function to perform by the radio button for it. Then the user can perform the operation on the map. For example, to perform a zoom-in operation, the user first selects the zoom-in radio button, then the user creates a red rectangular zoom box by dragging the mouse. A zoom-out operation can be performed similarly. The user can also insert a new geometry feature on the layer currently active. The user can perform a spatial query on the layer currently active by drawing a bounding box on the map. In order to obtain information on an individual feature, the user can select `Information` and click on the feature on the map.
- **map scale bar**
The scale bar is updated automatically after each map operation.

4 Creating a WebGD Application

We use configuration files to customize the WebGD framework for an application. A region configuration file specifies the details of each region. A map layer configuration file customizes the look and action of each layer. In fact, what we call a layer here corresponds to a layer group in a map file. A layer group is a set of layers whose visibility can be controlled simultaneously. A quick view configuration file specifies the locations on the map where the user can go directly.

When the user performs a map operation that results in a change of the current region, the spatial reference is automatically switched to that of the new region in order to minimize map distortion. When the region changes, the layers in the map legend, the list of the map-navigation and data-manipulation commands, and the quick view list are reconfigured for the new region based on the configuration files for the new region. This reconfiguration is necessary when different regions require different sets of map layers, map commands, and quick view lists. `region_js_functions.inc` contains all the region-dependent JavaScript functions.

4.1 *Region Configuration File*

Whenever the map region changes due to a user action on the map interface, the region configuration file for the new region is loaded dynamically. Then the map interface is customized according to the new configuration file. A region configuration file for each map region includes the following definitions:

1. the map projection for the region,
2. the name of the region displayed on the map interface,
3. the unit of distance measurement,
4. the name of the map layer configuration file for the region, and
5. the name of the quick view configuration file.

For example, the region configuration file for the world region is defined as follows.

```
$region = array(  
    "gid" => 1,  
    "name" => "world",  
    "display_name" => "World",  
    "srid" => 4326,  
    "rank" => 1,  
    "units" => "MS_DD",  
    "mapfile" => "gmap75_world.map",  
    "quickview" => "world_qview.php",  
    "legend" => "world_maplayers.php",  
);
```

We now describe each of the option specified in the configuration file:

- **gid**
A unique number assigned to each region. This number is the primary key value of the row representing the region in table `regions` in the database.
- **name**
The name used to uniquely identify the region in the database. This name is in lowercase and may include underscore characters.
- **display_name**
The name of the region as displayed on the map interface. This name may include capital letters, white spaces, and other punctuation marks.
- **srid**
The spatial reference identifier for the map projection assigned to the region.
- **rank**
The priority number used in determining the region to be used. The region with the highest rank number is selected among the regions that encompass the map area to be viewed.
- **units**
The unit of distance for the region. The value is typically defined as part of the spatial reference definition. Possible values are `MS_DD` (decimal degree), `MS_METER`, and `MS_FEET`.

- `mapfile`
The map file to be loaded when the region is selected.
- `quickview`
The name of the quick view configuration file for the region.
- `legend`
The name of the map layer configuration file for the region.

4.2 Map Layer Configuration File

The map layer configuration file provided for a region specifies the layers to be included in the legend and their characteristics. It is possible for multiple map regions to share one map layer configuration file.

The map layer configuration file for the Oregon region, for example, contains the following definitions.

```
$layer_groups = array (
  'grp_eo_py' => array(
    'geom_type' => 'polygon',
    'table' => 'eo_py',
    'layer_selectable' => true,
    'gid_column' => 'gid',
    'geom_col' => 'the_geom',
    'legend_label' => 'EO Polygons',
    'search_script' => 'forms/eo/eo_py_eo_search.phtml',
    'select_script' => 'forms/eo/eo_py_eo_select.phtml',
    'edit_script' => 'forms/eo/eo_py_edit.phtml',
    'normal_layer' => 'eo_py',
    'searched_layer' => 'eo_py_searched',
    'checked_layer' => 'eo_py_checked',
    'selected_layer' => 'eo_py_selected',
    'img_src' => 'images/eo_poly.png',
    'img_width' => 26,
    'img_height' => 26,
    'onclick' => 'activate_layer("grp_eo_py")',
    'data_srid' => 6010
  ),
  . . .
)
```

The map legend displayed in the map interface according to this configuration file is shown in Figure 4-1.

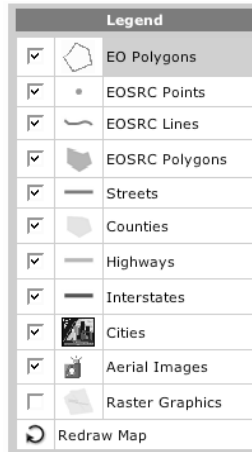


Figure 4-1: Map Legend for Oregon Region

We now explain each option used in a map-layer configuration file:

- `geom_type`
The type of geometry features contained in the layer. The value can be `polygon`, `multipolygon`, `linestring` and `multilinestring`. Exact spatial operations performed depend on this geometry type. For example, if the type is `point`, a point is inserted on the map with an `insert` operation. On the other hand, if the type is `polygon`, a polygon feature is inserted.
- `table`
The name of the table in the database that contains the geometry data for the layer.
- `layer_selectable`
The boolean option that determines whether spatial operations can be performed on the layer or not. Some layers, such as those for counties and highways in the example above, are not selectable for spatial operations.
- `gid_column`
The name of the primary key of the table that stores the geometry features for the map layer. The default value of this option is `gid`.
- `geom_col`
The name of the geometry column containing the geometry features in the layer. The default value of this option is `the_geom`.

- `legend_label`
The name of the layer in the legend.
- `search_script`
The name and the location of the search form associated with the layer.
- `select_script`
The name and the location of the select form associated with the layer.
- `edit_script`
The name and the location of the edit form associated with the layer.
- `normal_layer`
The name of the map layer, as defined in the map file, to be displayed when no highlighting occurs.
- `searched_layer`, `checked_layer`, `selected_layer`
The names of the layers used to highlight the geometry features when searching, checking, or selecting, respectively, occurs. The geometry features returned by a search operation are **searched for**, those whose checkboxes are turned on in the search result form are **checked**, and the geometry feature chosen for editing is **selected**.
- `img_src`, `img_width`, `img_height`
The source file name, the width, and the height of the icon in the legend.
- `onclick`
The name of the JavaScript event handler invoked when the user selects the layer in the legend.
- `data_srid`
The *spatial reference identifier (srid)*, a number that designates the map projection used by the geometry features in the map layer. If this srid is different from that of the current map region, then geometry features in the layer are reprojected before they are displayed on the map.

4.3 Quick View Configuration File

The quick view mechanism allows the user to select a map area directly. The user can go to the new map area quickly by selecting the name of the map area from the quick-view dropdown list.

Each entry in a quick view configuration file describes the name of the map area, the map projection used by the extent, and the extent of the area. The quick view configuration file for the Oregon region, for example, can be as follows.

```
$qview = array(  
  array(  
    'name' => 'World',  
    'srid' => 4326,  
    'extent' => '-180,-90,180,90'  
  ),  
  array(  
    'name' => 'United States',  
    'srid' => 4326,  
    'extent' => '-125,13,-65,53'  
  ),  
  array(  
    'name' => 'United States, East',  
    'srid' => 4326,  
    'extent' => '-102,22,-60,50'  
  ),  
  array(  
    'name' => 'United States, West',  
    'srid' => 4326,  
    'extent' => '-135,30,-105,50'  
  ),  
  ...  
)
```

The quick view list displayed in the map interface according to the configuration file above is shown in Figure 4-2.

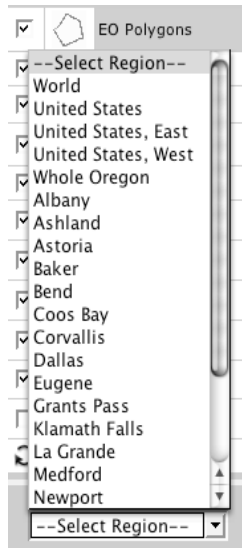


Figure 4-2: Quick View Selection of Map Areas

Each entry in a quick view configuration file can specify the following options:

- **name**
The name of the map area.
- **srid**
The spatial reference identifier for the `extent` explained below.
- **extent**
The `xmin`, `ymin`, `xmax`, and `ymax` values describing the positions of the lower left corner and the upper right corner of the map area.

5 Overview of the WebGD Framework

A map interface is refreshed according to the following steps.

1. The new map extent is determined based on the user action.
2. The mechanism of dynamic switching of spatial references determines the new map region.
3. The name of the map file to be loaded is read from the region configuration file.
4. The spatial reference of the region read from the region configuration file is set.
5. The new map extent is transformed to the new spatial reference system.
6. The new legend is created according to the map legend configuration file specified in the region configuration file.
7. The quickview configuration file whose name is in the region configuration file is processed.
8. A function in PHP MapScript is called to generate the map image with the parameters set in the preceding steps.

6 Implementation Details of the WebGD Framework

In this section we discuss the implementation details of the WebGD framework. The topics covered are the set of global variables defined in `gmap75.php`, the HTML form variables used by `gmap75_main.phtml`, the creation of the map legend, the interaction between the map legend and the map functions, the creation of dynamic quickview mechanism, the Zoom In mechanism, the Zoom Out mechanism, and the map Panning mechanism.

6.1 Global Variables Defined in `gmap75.php`

`gmap75.php` maintains in global variables the state of interaction including the substates for the map legend and the map command tool.

- `$default_selected_layer`
This variable holds the name of the layer to be selected by default when the application is initialized. The value is specified by `$default_selected_layer` in the map layer configuration file for each region. If none of the layers defined in the map layer configuration file can be activated, the value should be empty.
- `$layer_groups`
The value for this variable is copied from `$layer_groups` defined in the the map layer configuration file for the current region.
- `$qview_sel_opts`
This variable is a PHP string that holds the HTML drop-down list reformatted from the quick view configuration file.
- `$js_selectable_layers`
This variable is a PHP string that holds the names of the layers that can be activated in the legend. The string is in JavaScript array format.
- `$js_layers_point`, `$js_layers_polygon`, `$js_layers_line`
`$js_layers_point` contains the names of the POINT layers.
`$js_layers_polygon` contains the names of the POLYGON layers.
`$js_layers_line` contains the names of the LINE layers.

- `$js_all_legend_layers`
This variable is a PHP string that holds the names of all the the layer groups defined in the map layer configuration file.

6.2 *HTML Parameters in gmap75_main.phtml*

The following hidden parameters are defined in `gmap75_main.phtml`:

- `pointx, pointy`
`pointx` contains the x value of the point inserted on the map.
`pointy` contains the y value of the point inserted on the map.
- `extminx, extminy, extmaxx, extmaxy`
`extminx` contains the x value of the lower left corner of the map extent.
`extminy` contains the y value of the lower left corner of the map extent.
`extmaxx` contains the x value of the upper right corner of the map extent.
`extmaxy` contains the y value of the upper right corner of the map extent.
- `ZOOM_TO_SEARCHED`
If this variable is set after a search operation, the map automatically zooms in to the area show all the resulting features.
- `DO_HIGHLIGHT`
If this variable is set, the geometry features returned by a search operation are highlighted on the map.
- `layer_name`
This variable contains the name of the layer activated by the user in the legend list.
- `layer_type`
This variable contains the value of `searched`, `checked`, or `selected`.
- `list_gid`
This variable contains the list of *identifiers* (integers) of geometry features searched on the map.
- `region_name`
This variable contains the name of the current region.

- `last_cmd`
This variable contains the name of the last command performed on the map.
- `mapfile_switched`
This variable is set if the map file changes after dynamic switching of spatial reference.

6.3 Creation of the Map Legend

The HTML statements for the map legend is created as follows:

1. After the map layer configuration file associated with the current region, `oregon_maplayers.php`, for example, is read, the function `legend_file_read()` in `gmap75.php` is called to copy the `$layer_groups` array defined in the layer configuration file. `$region_name` is passed as the parameter to function `legend_file_read()`. Array `$layer_groups` is then available as a global variable in `gmap75.php` for other functionalities.
2. JavaScript look-up table `layer_webforms` in `rubber.js` is initialized from `$layer_groups` PHP array defined in the layer configuration file. This array designates for each layer the Web scripts that are activated when a user performs map operations as

```
layer_webforms["eosrc_point"]["edit_script"] =
"eosrc_point_edit.phtml".
```
3. The default selected layer is determined from the map layer configuration file. The variable `$default_selected_layer` in the map layer configuration file designates the layer to be activated in the legend when the map interface is initialized.
4. The HTML code for the legend table with checkboxes, icons, and layer names are produced. For this purpose, function `dynamic_legend()` defined in `gmap75.php` is called from `gmap75_main.phtml` defined in the framework.
5. The cell color for the default selected layer is highlighted if the user does not select another layer with the preceding map operation. Otherwise, the layer activated by the user is highlighted. Function `retain_layer_cmd()` defined

- in `gmap75.php` restores the selected layer when the map is refreshed. If the selected layer is of type POLYGON or LINE, and map function `Insert` has been selected, link `Done` is created.
6. When the `Done` link is visible, the user can insert nodes on the map for a polygon or line feature. JavaScript function `closeParcel()` in `region_js_functions.inc` is called when the `Done` link is clicked.
 7. The name of the current active layer is saved into hidden variables `$layer_name` and `$selected_layer` in `gmap75_main.phtml`.

6.4 Interaction between the Map Legend and the Map Functions

The map legend and the map functions interact as follows.

1. When the user activates a layer, the label for the activated layer must be highlighted. JavaScript function `layer_color_reset()` defined in `region_js_functions.inc` is called first to reset all the colors of the layers to the default color. JavaScript function `activate_layer()` defined in `region_js_functions.inc` is then called to change the cell color of the new active layer.
2. If the user performs a `Search-by-Area` operation, the search form for the active layer is opened. JavaScript variable `cmd` defined in `rubber.js` is set to `SEARCH_AREA` by the function `setCmdChecked()`. The name of the user selected layer is obtained from HTML hidden form variable `selected_layer`.
Look-up table
`layer_webforms[selected_layer][“search_script”]` determines the name of the search form to be used.
3. If the user inserts a point or polygon feature, link `Done` is made visible by JavaScript function `startParcel()` in `region_js_functions.inc`. The user can then insert nodes by clicking on the map. When the `Done` link is clicked, those nodes are inserted into the table containing the geometry features of the currently active layer. The action script used for inserting geometry feature created is determined from look-up table `layer_webforms` in `rubber.js`.

6.5 Creation of Dynamic Quickview Mechanism

The HTML statements for the dynamic quickview mechanism is created as follows.

1. The quick view configuration file for the current region is read. Function `quickview_file_read()` in `gmap75.php` then initializes global variable `$qview_sel_opts` for the region specified by parameter `$region_name`.
2. By printing variable `$qview_sel_opts`, function `dynamic_quickview()` creates the HTML drop-down list for the areas listed in the quickview configuration file.

6.6 Zoom-in Mechanism

We now describe the steps involved in performing a zoom-in map operation.

1. With the Zoom-In map function radio button checked, the user can press, drag, and release the mouse button to draw a zoom-box on the map image. When the user holds down the mouse button, JavaScript function `startRubber()` defined in `rubber.js` is called. Then function `moveRubber()` is called each time the mouse is moved. When the user releases the mouse button, function `stopRubber()` is called to record the screen coordinates of the zoom-box.
2. The zoom-box screen coordinates are stored in variables `x1`, `y1`, `x2`, and `y2` in `rubber.js`. Variables `x1` and `y1` represent the coordinate values of the lower left corner of the zoom-box, and variables `x2` and `y2` those of the upper right corner.
3. The values `x1`, `y1`, `x2`, and `y2` are converted into the format `x1,y1;x2,y2` and this string is saved in hidden form input variable `INPUT_COORD` defined in `gmap75_main.phtml`. `INPUT_TYPE` hidden form input variable is set to `auto_rect`.
4. The form containing the map interface is submitted.
5. In `gmap75.php`, the zoom-box coordinates are parsed from variable `$HTTP_FORM_VARS["INPUT_COORD"]`, and whether the coordinate values describe a zoom-box or a zoom-point is determined. A zoom-box is a zoom-point when the value of `x1` is equal to or very close to the value of `x2` and when the value of `y1` is equal to or very close to the value of `y2`.

6. Method `zoompoint()` for PHP-MapScript `$gpoMap` object is called if the zoom-box is a point. If the zoom-box is a rectangle, the method `zoomrectangle()` is called.
7. The map image is generated by `gmap75_main.phtml`.

6.7 Zoom-out Mechanism

Now we describe the steps involved in performing a zoom-out map operation.

1. Same as Step 1 of the Zoom In mechanism.
2. Same as Step 2 of the Zoom In mechanism.
3. Same as Step 3 of the Zoom In mechanism.
4. Same as Step 4 of the Zoom In mechanism.
5. The extent of the map before the map is refreshed is saved in hidden form variables `xmin`, `ymin`, `xmax`, `ymax` in `gmap75_main.phtml` in the framework. These variables are in the previous map coordinates.
6. In `gmap75.php`, the zoom-box coordinates are parsed from the variable `$HTTP_FORM_VARS["INPUT_COORD"]` as `x1`, `y1`, `x2`, and `y2`.
7. The width of the zoom-box `$rectWidth` is calculated as `x2 - x1`, and the height of the zoom-box `$rectHeight` is calculated as `y2 - y1`.
8. The map extent `xmin`, `ymin`, `xmax`, and `ymax` are copied into variables `$dfMinX`, `$dfMaxX`, `$dfMinY`, and `$dfMaxY` in `gmap75.php`.
9. The previous extent is set to the map object `$gpoMap` by method call `$gpoMap->setExtent($dfMinX, $dfMaxX, $dfMinY, $dfMaxY)`.
10. The image scale for the Zoom Out operation is computed as the ratios of the width and the height of the zoom-box in pixel and the width and the height of the map image (`$gpoMap->width` and `$gpoMap->height`). The computation occurs as


```
$scaleX = abs($gpoMap->width / $rectWidth);
$scaleY = abs($gpoMap->height / $rectHeight);
```
11. The width and the height of the map in map units are computed as `$muWidth` and `$muHeight` where

- ```

$muWidth =
 abs($gpoMap->extent->maxx - $gpoMap->extent->minx);
$muHeight =
 abs($gpoMap->extent->maxy - $gpoMap->extent->miny);

```
12. The zoom factor `$zoomFact` is computed as `MIN($scaleX, $scaleY)`.
  13. The new width and the height of the map are computed in map units as

```

$newWidth = $muWidth * $zoomFact;
$newHeight = $muHeight * $zoomFact;

```
  14. The new map scale after the zoom-out request is computed as

```

$newSF = $newWidth / $gpoMap->width;

```
  15. The new map extent after zoom-out request is computed as

```

$newXmin = $gpoMap->extent->minx -
 ($oPixelRect->minx * $newSF);
$newXmax = $newXmin + $newWidth;
$newYmin = $gpoMap->extent->miny -
 (($gpoMap->height - $oPixelRect->maxy) * $newSF);
$newYmax = $newYmin + $newHeight;

```
  16. Finally, the map extent is set with method

```

$gpoMap->setExtent($newXmin, $newXmax, $newYmin,
 $newYmax).

```

The map image is generated with `$gpoMap->draw()` method.

## 6.8 Map Panning Mechanism

We now describe the steps involved in performing the panning map function.

1. The user can press, drag, and release the mouse button to pan the map. The initial mouse position is recorded by JavaScript function `startRubber()`, the direction of the mouse movement by function `moveRubber()`, and the final mouse position by function `stopRubber()`. The horizontal direction (left or right) of the mouse movement is saved in form variable `DIRX`, while the vertical direction (up or down) is saved in variable `DIRY`.

2. The initial mouse position is stored in variables `x1` and `y1` in `rubber.js`. The final mouse position is stored in variables `x2` and `y2`.
3. Same as Step 3 of the Zoom In operation.
4. In `gmap75.php`, the mouse positions are parsed from variable `$HTTP_FORM_VARS["INPUT_COORD"]`. The mouse positions are then saved into variables `$x1`, `$y1`, `$x2`, and `$y2`.
5. Same as Step 8 of the Zoom Out operation.
6. Same as Step 10 of the Zoom Out operation.
7. Same as Step 9 of the Zoom Out operation.
8. The numerical signs for mouse movement directions `$xSign` and `$ySign` are computed as
 

```
$xSign = ($HTTP_FORM_VARS["DIRX"] == "left")?(-1):1;
$ySign = ($HTTP_FORM_VARS["DIRY"] == "up")?(-1):1;
```
9. The scale of the map `$scale` is computed as `MIN($scaleX, $scaleY)`.
10. The horizontal distance `$xAdjust` and the vertical distance `$yAdjust` between the initial and final mouse positions are computed as
 

```
$xAdjust = $xSign * $scale * ($x1 - $x2);
$yAdjust = $ySign * $scale * ($y2 - $y1);
```
11. The new map extent is computed as
 

```
$newXmin = $gpoMap->extent->minx + $xAdjust;
$newXmax = $gpoMap->extent->maxx + $xAdjust;
$newYmin = $gpoMap->extent->miny + $yAdjust;
$newYmax = $gpoMap->extent->maxy + $yAdjust;
```
12. The newly computed map extent is set with method call
 

```
$gpoMap->setExtent($newXmin, $newYmin, $newXmax,
$newYmax);
```
13. The map is generated with the `$gpoMap->draw()` method in `gmap75_main.phtml` in the framework.

## 7 Conclusion

In order to create Web-based GIS/database applications rapidly, we have developed the WebGD framework and used it to create 14 applications so far. The WebGD framework supports the following unique features.

1. Geometry features can be inserted and deleted through a set of Web forms.
2. Each map region can be customized with its own map legend, map operations and quickview list with a set of configuration files.
3. When the current region changes, the spatial reference used by it can be dynamically switched.
4. One framework is shared by many applications.

The WebGD framework also includes a form generator that allows web scripts for database access to be generated automatically from a database schema. Consequently, we can produce a new WebGD application by creating a database and by providing configuration files. A WebGD application generated thus supports the following functions:

1. Zoom In operation,
2. Zoom Out operation,
3. Map Panning operation,
4. Spatial query of geometry features on the map, and
5. Update and deletion of geometry features with a Web interface.

We can create an application that contains several tables with geometry features in about one month by using the WebGD framework. The Web scripts can perform more complex operation than the ones manually coded. A similar application would take one year to create if manually coded.

## References

- [1] PHP: Hypertext Preprocessor. <http://www.php.net>.
- [2] *OpenGIS Simple Feature Specification for SQL Revision 1.1*, Open GIS Consortium, Inc.
- [3] PostgreSQL: The world's most advanced open source database. <http://www.postgresql.org>.

- [4] Refrations Research. <http://www.refrations.net>.
- [5] PostGIS. <http://postgis.refrations.net>.
- [6] DM Solutions Group – Online Mapping Solutions and Services.  
<http://www.dmsolutions.ca>.
- [7] McKenna, Jeff, *MapServer PHP/MapScript Class Reference – Version 4.6*, DM Solutions Group Inc. <http://www.maptools.org>.
- [8] MapServer Documentation Project. <http://mapserver.gis.umn.edu/docs>.