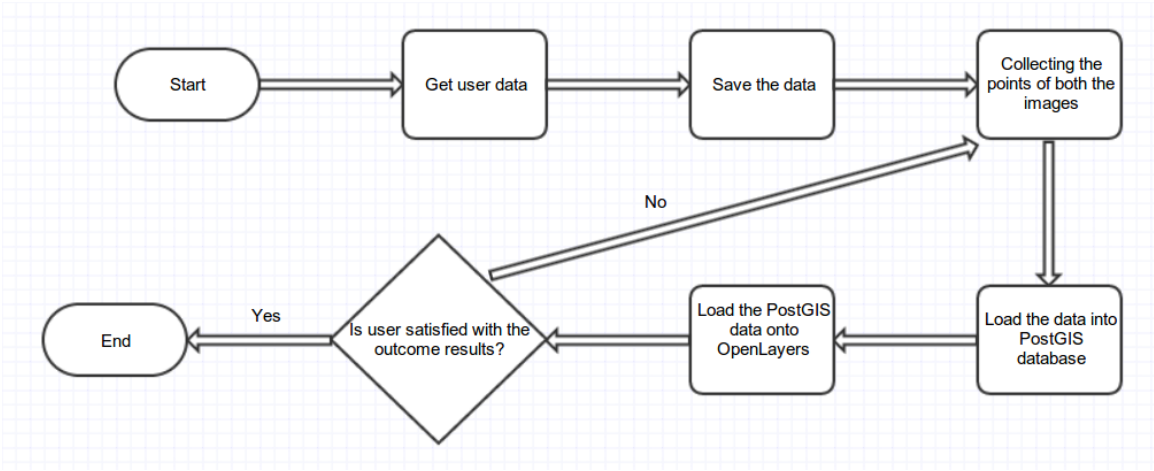
Orthorectifying EM data in reference to a thin section image

A brief introduction of the project:

As an extension to the previous project, the aim of the project is used to overlay the images spatially using different GIS tools. We are given a EM data and a thin section image and asked to reference the images.As we have a EM data(which is a high resolution image), we need to make sure that the base layer(image) can be made to geo-reference the points. So, for this to be clear we will be using jQuery for zoom levels so that the user can point-out the coordinates more easily and accurately. The GIS tools that we would be using are PostGIS, OpenLayers. The data is to be loaded into PostGIS database. Later the PostGIS data is loaded onto the Openlayers with EM data over the base layer based upon the coordinates provided by the user.

The flow of the project is shown below:



Getting the user data

index.php

Initially, we define a structure ‘class Image’ so that we can inherit it publicly and more easily. There are several variables that are defined inside the structure such as url, height, width etc.

The main application is processed from here. States are defined so that it will be more compatible. States are changed based on forms that are submitted by the user(or buttons that eventually send forms via JavaScript) which contain hidden input fields that requests for the next state.

getBase.php

$_SESSION['subImages'] = array();

$_SESSION['subCount'] = 0;

The above two variables are defined to keep track of number of subimages that are submitted by a particular user.

$_SESSION['dataDir'] = "data/";

The above variable is a temporary folder location to store the data. We get to know how this will be used in the next step.

The user here submits a hidden input field.

saveBase.php

A constructor $base = new Image(); is created.

From this constructor, we assign the details so as to save the

$baseFile = $_SESSION['dataDir']."base";

$base->url = $baseFile.".gif";

The $base->url is the desired path where the base image will be stored. Later the image is stored using imagegif.

getSub.php

The user selects an image and submits a hidden input field.

saveSub.php

This is similar to saving the base image but an additional information is being stored. $_SESSION['subCount'] = $_SESSION['subCount'] + 1;

The above variable is being incremented so as to keep track of number subimage is being used.

collectPoints.php

An id has been assigned to each element. The zoom function will be working based on this id's.

This plugin manages smooth zoom and pan on a given dom element.

$('<id>').smartZoom('zoom', scaleToAdd); this zooms to the given position. scaleToAdd is an argument for which we will be able to zoom the image to the current scale.

$('<id>').smartZoom('pan', pixelsToMoveOnX, pixelsToMoveOnY); this moves the image in simple terms this can be imagined as transformation of axis which here is the image. So, both pixelsToMoveOnX, pixelsToMoveOnY are given as arguments that contains a value.

One more point to notice is that the zoom function is disabled with the use mouse because the scaleToAdd argument changes and it becomes more difficult to identify.

myImg = document.getElementById("<id>");

myImg.onmousedown = GetCoordinates;

The GetCoordinates function uses the window.event method instead of .position() which gives the current coordinates of the element relative to the offset parent, to find the coordinates of the mouse when it is clicked. It also needs to take into account any scrolling and the position of the image inside the document so that the coordinates are always

relative to the top left of the image. The FindPosition function finds the position of the image tag within the page.

The points that have been collected are now passed. As the values are in javascript variables we need to assign them to php variables.

```
$.post( $("#myForm").attr("action"), $('#str').val(JSON.stringify(finarray)),
function(info){  $("#result").html(info); });
```

The above script converts the javscript array into php array.

On clicking, the data is loaded and then the array is converted to a javascript string.

The values that the are given by the user are stored into an array and then converted into a string that contains JSON text. I have used sessionStorage to store the string that can be accessed through an entire session.

displayResults.php

The values that are produced by the user are stored in the session and then accessed using JSON.pasre().

We have 8 values consisting of 4 coordinates that are provided by the user and 2 strings that are path to the images.

Let the coordinates of base image be (x_{b1}, y_{b1}) and (x_{b2},y_{b2}) and of subimage be (x_{s1}, y_{s1}) and (x_{s2},y_{s2}).

Based on the scale to which the subimage has to be overlaid is calculated and accordingly the picked coordinates are also calculated with respect to the base image.

To display images we need to create an image layer where one can view each image as a layer, so I used OpenLayers.Layer.Image constructor to show as an image layer.

So, after defining the layers we need to add layers in-order to display the images.

One of the features is to download the overlaid images. So, for this I have used an ajax request, which converts the javascript variable to php variable. The request is sent to the server side and received at the client side. The values sent by the server side are stored into a file. The overlaid image file is created based on the values that are taken from

the server side. The format in which the file has to be is written accordingly with the variables, html tags and constructors.

In similar to the above feature, one more feature is to download all the subimages that are part of the base image can also be downloaded as a single file. Here, we keep track of all the subimages and also the coordinates of the overlaid points that are necessary for the final output.