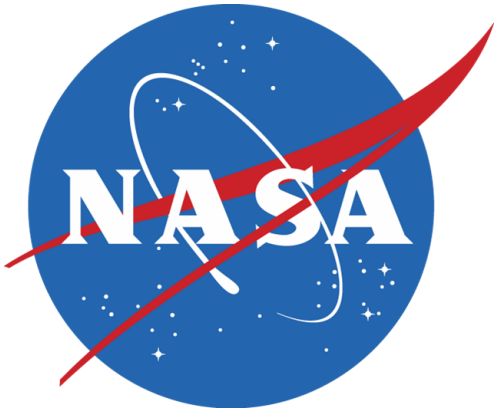


# A Web Based Impervious Cover Decision Making Tool

James S. Yang
University of Pennsylvania
Philadelphia, PA 19104
Stennis Space Center
Stennis Space Center, MS 39529-6000

National Aeronautics and
Space Administration

Stennis Space Center
Stennis Space Center, MS 39529-6000

**Acknowledgements**

I would like to thank the NASA Educations Program coordinator, Nancy Bordelon, and my mentors Duane Armstrong and Ted Mason for giving me the opportunity to work on this project. I would also like to thank several individuals for their support and contribution to the project, including James "Doc" Smoot, Gerry Gasser, and Shannon Ellis. Last and definitely not least, I'd like to give a HUGE thank you to Carolyn Owens for providing me with incredible amounts of technical support and advice throughout this project.

**Abstract**

This paper presents proof-of-concept geological analysis and decision making tool in the form of a web application. The application is designed to disseminate impervious cover data and supplemental GIS data. We discuss the necessity of such a tool and any anticipated limitations inherent in the method of generating useful data. We explore our implemented front and back end design as well as some planned features at a high and low level. Finally, we discuss recommendations for software tools that would supplement the application.

# 1 Introduction

Impervious surfaces are areas of land covered by urban development materials that are impermeable to water such as concrete and asphalt, or natural cover that has been densely packed as a result of urban development. These landscapes are closely studied environmental concerns because of the risk of ecological damage they pose to nearby sources of water and the atmosphere.

Rainfall over these areas drain into larger bodies of water, transporting harmful contaminants, such as garbage, sewage, and oils and exhaust particulates from cars, to nearby bodies of water. These contaimants can harm existing aquatic life in local ecosystems and make waters unsafe for humans. Materials that compose impervious surfaces also shed excess heat under rainfall, which when transferred to larger bodies of water, warms these waters, promoting deoxygenation and pH imbalance. This makes marine life more difficult to sustain. Finally, impervious surfaces displace natural tree cover that would normally attenuate urban climate fluctuations and reduce the amount of energy used to artifically control indoor climates.

In an effort to improve methods of surveying impervious surfaces and their impact on the environment, we want to create a decision making tool that provides impervious cover information supplemented with GIS data, such as percentage imperviousness (PIMP) calculations, amount of impervious cover, UTM coordinates of affected areas, and Hydrological Unit Code (HUC) boundaries of affected areas. This tool would help urban developers and researchers supplement envinronmental impact studies.

# 2 Data Production and Tool Design Motivation

The final application design depends heavily upon the availability of data and what types of data users will need. Impervious cover data are produced by analyzing large satellite images with various remote sensing algorithms. The nature of these data interpretation algorithms creates limitations on the availability of both input and output data, and the ability to honor requests for results in a timely manner. End users will be able to access the data collected throughout the process, as well as the final PIMP values, depending upon their needs. The following examines the limitations we face in terms of supplying data, and describe features to be added based on need and feasibility.

## 2.1 Data Availability and Other Obstacles

Data availability is affected by three factors: satellite coverage, image quality, and final product quality assurance. First, satellite images are neither ubiquitous nor always available. A satellite passes over a particular geographic location once every sixteen days, restricting user queries to those where satellite coverage is available. Second, satellite images are not always usable. Satellite images with excess cloud cover or interrupted transmission may be scrapped if the image has been affected around regions of interest. For obvious reasons, these images cannot be resampled. Finally, it is important to note that the end products are created from extremely large, high-dimensional datasets. The data production method is in no way immune to error, making QA a necessary step.

Other obstacles include the amount of input data necessary to produce desired outputs and natural disasters that may produce erroneous results. The current process to develop outputs requires two sets of satellite images representing the same region in two different seasons. This is to take into account the amount of surface area tree leaves represent in a region. These two time periods may not represent a unique event, i.e. natural disasters such as hurricanes, earthquakes, tornadoes or other acts of nature that can displace loose materials and affect remote sensing data.

Meeting client requests for output data will be primarily affected by output rates. The remote sensing algorithms used to produce impervious cover data are computationally heavy, so end products cannot be disseminated on the fly. Data extraction is also currently a manual process, where a human must interact with special software to produce results, which also throttles the process.

## 2.2 Application Front End Task Flow

We want the end-user to be able to accomplish the following tasks using the web application:

1. Request PIMP value of a specific point on a map
2. Request average PIMP value of a selected area
3. Save regions of interests
4. Subscribe to changes in impervious cover in areas of interest
5. Superimpose regions of interest with relevant GIS data such as HUC boundaries and UTM coordinates
6. Request specific data regarding a saved region

The user will employ these functionalities to analyze urban development projects or monitor existing impervious cover. A typical work flow should approximate the diagram depicted in Figure 1:

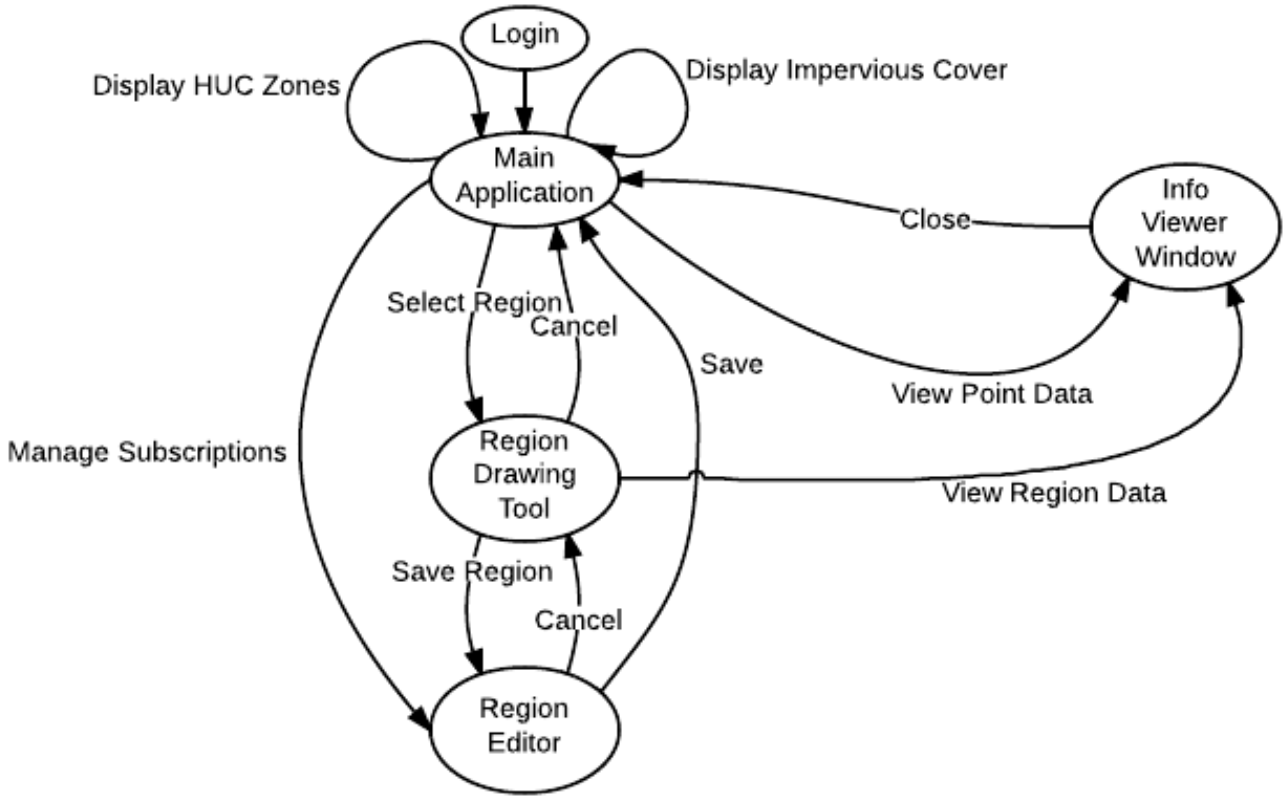



Figure 1: Typical end-user workflow.

### 2.3 Feasibility of Features

While there is not much that can be done about the availability of input data, streamlining the output production is possible so that the best can be made of what data are available. Automating the data gathering process is part of the effort, described in detail in Appendix A. This process, however, cannot be fully automated as of yet due to the lack of several tools. One of these tools is a method of automatically retrieving satellite images.

Based on the current constraints, we expect to be able to produce perennial data sets. Given the long duration of urban development projects, this is an acceptable value. Ideal climate and imaging conditions, which occur occasionally throughout any given year, may yield more data. Due to the unpredictability of available output data, users will be provided with data that match their search critieria the closest. Requests that include both date and geographic region will not be accepted.

With these limitations in mind, we discuss each feature listed in Section 2.2 according to their numerical label:

1. Users will be able to request the PIMP value of a specific point on a map, but with limitations. Requests for a particular date will be accomodated when possible. If the data requested does not exist, the closest match will be returned.

2. Users will be able to request the average PIMP value of a selected area with the same caveats discussed in Feature 1.

3. By creating a database of user profiles, we will be able to support saving regions of interest.

4. Using a messaging service such as email, users can subscribe to alerts about changes in impervious cover for areas of interest.

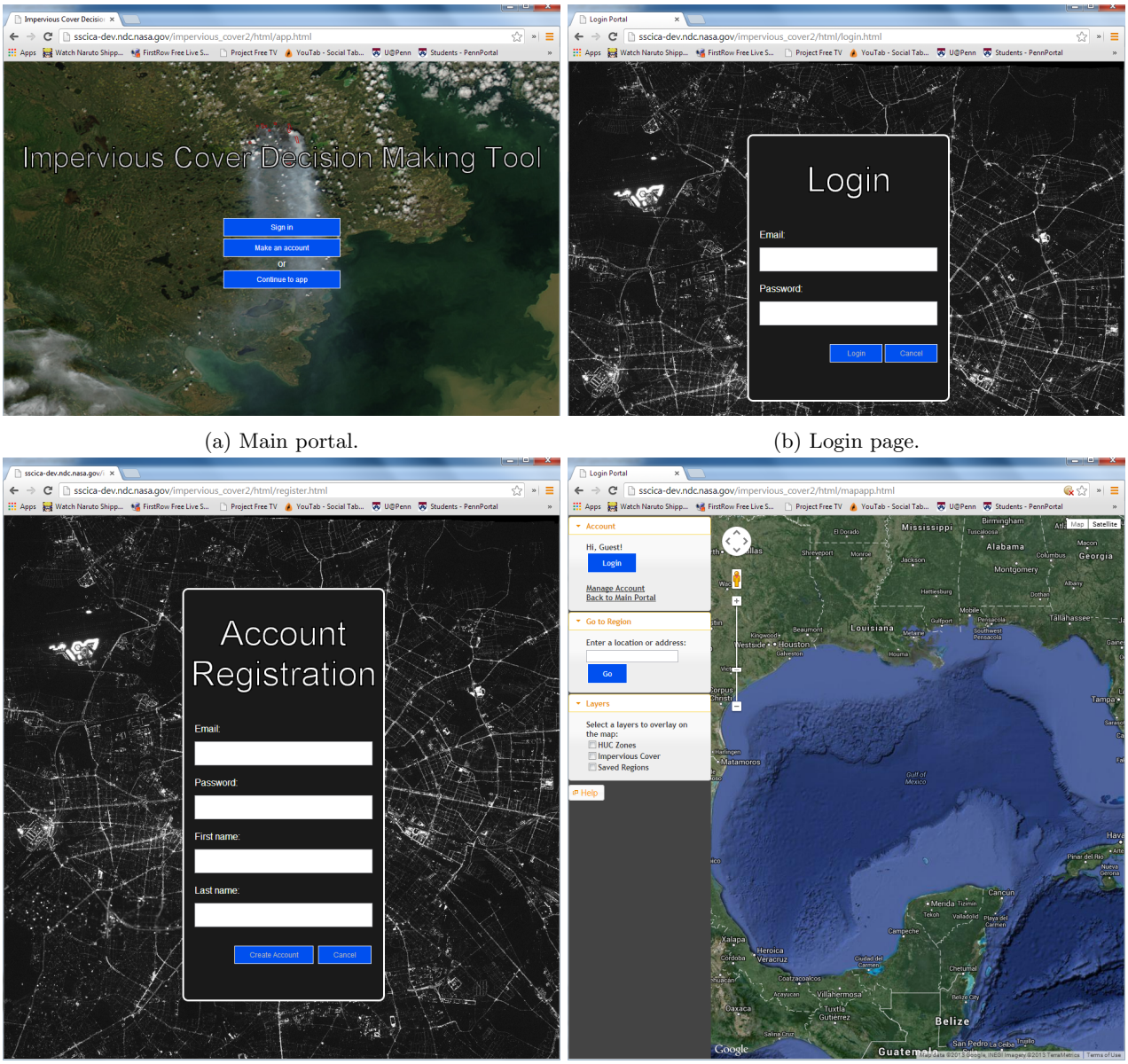
5. Since the PIMP calculation output product is a simple 2D matrix of values, it can be converted to an image that we can overlay on a map. The same overlaying method can be used to display subscribed regions of interest, HUC boundaries, UTM coordinates, and other supplemental GIS data.

6. Users will be able to request specific data for a saved region, including average PIMP value, coordinates of the region center, and the size of the region.

# 3 Application Front End Structure and Implementation

## 3.1 App Portal

We first created a main portal through which users may login to an existing profile, create a new profile, or proceed to the app without logging in. Figure 2a shows our main portal, Figure 2b shows our login page, Figure 2c shows our account registration page, and Figure 2d shows the main app portal.




(a) Main portal.

(b) Login page.

(c) Account registration page.

(d) Main app.

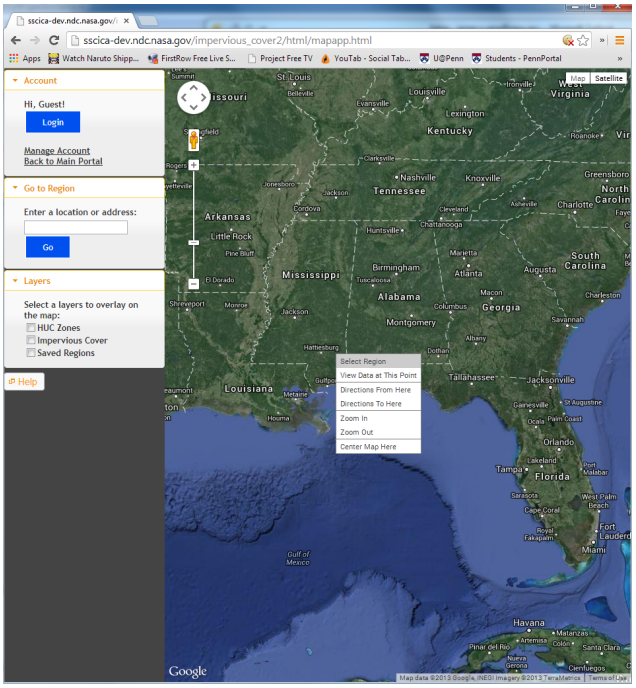
Figure 2: Sample web app screenshots.

## 3.2 Web-app Contents

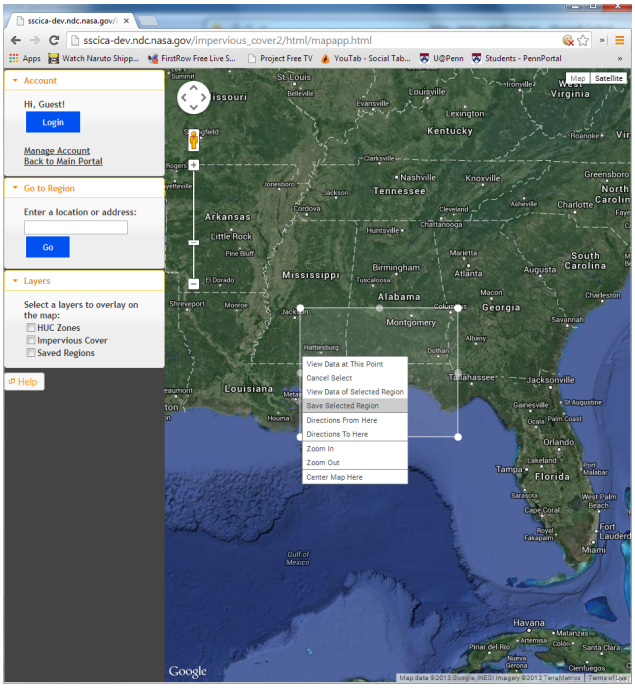
The web app consists of a canvas for displaying a map with GIS data and a control panel for accessing account preferences and map overlays.

### 3.2.1 Map Canvas

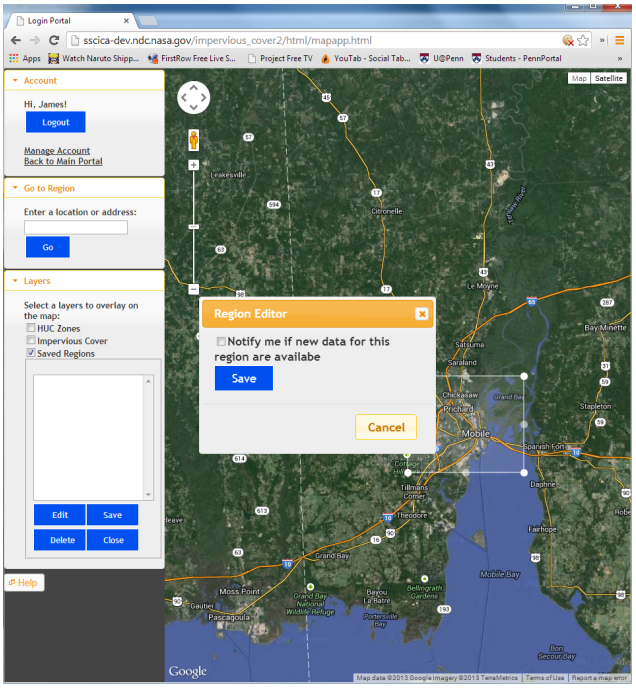
Through the map, one can display a context menu with options to select and subscribe to regions of interest or display data associated with a region of interest. The task flow for saving a region is demonstrated in Figures 3a, 3b, 3c, and 3d, and the task flow for displaying region data is demonstrated in Figures 4a, 4b, and 4c.



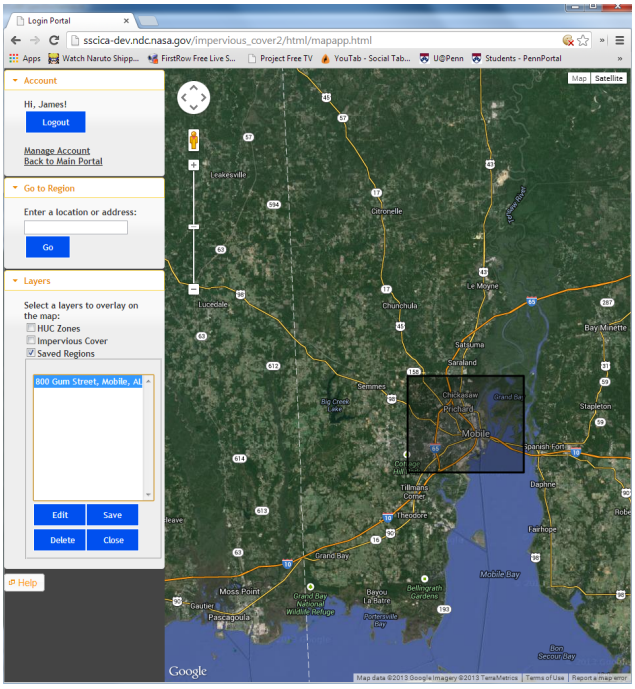

(a) Context menu with "select region" option.




(b) Saving selected region.

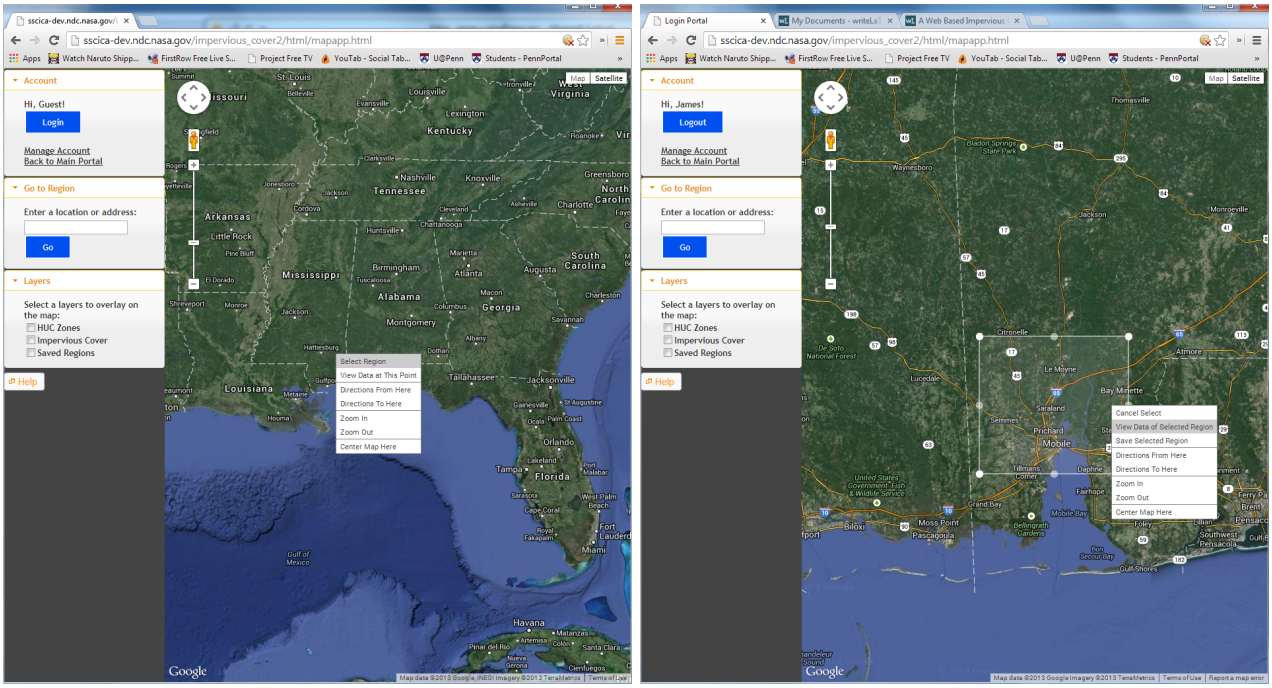



(c) Setting subscription options.



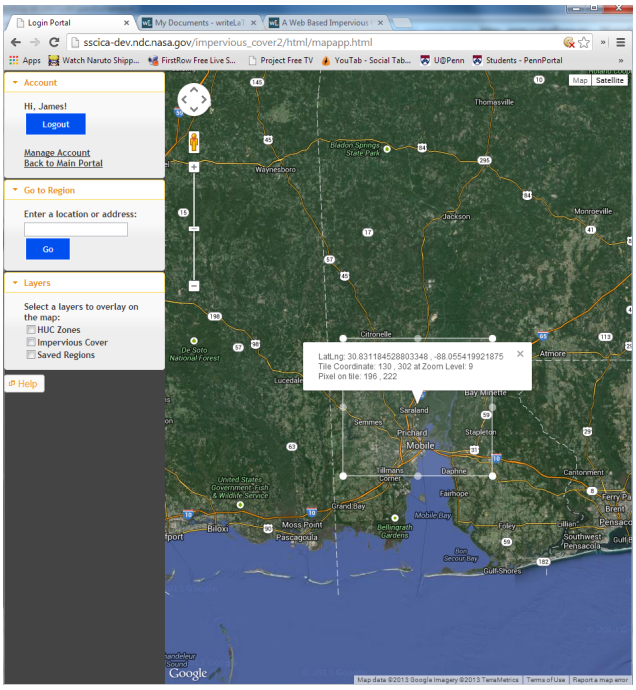

(d) Saved region added to subscriptions list in control panel.

Figure 3: Map canvas region save demonstration.



(a) Context menu with “select region” option.

(b) Selecting “view data for selected region” option.



(c) Info window with region information.

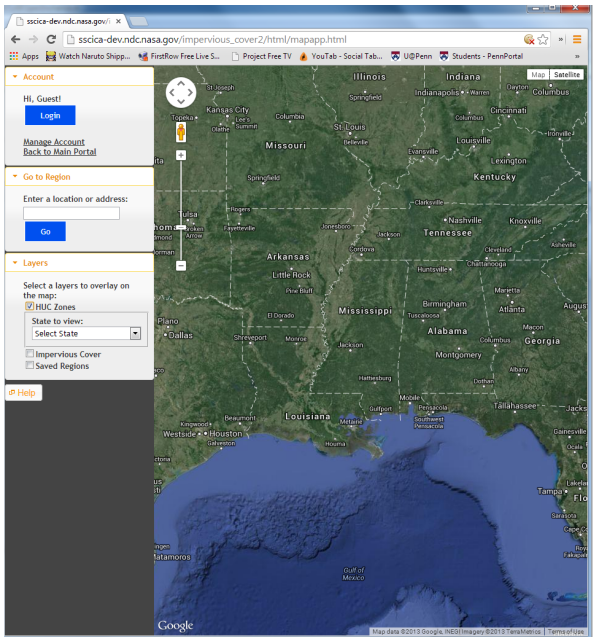
Figure 4: Map canvas region data display demonstration.

All data transactions between the client and the host server are initiated client-side via simple AJAX “post” and “get” queries. All queries are asynchronous with the host server, except for those that retrieve PIMP values because the values are displayed in a document element injected within a Javascript file. The document element would otherwise render before the host server can respond, ultimately displaying nothing. The impervious cover data are stored server-side as a single GEOTIFF image from which a percentage value can be obtained by referencing UTM coordinates. These queries must be performed synchronously so that client-side rendering does not time out. All subscription data are saved as JSON strings, making storing and parsing easy. Currently, we store region bounds as latitude and longitude coordinates, and a boolean value indicating whether or not the user has subscribed to that region.

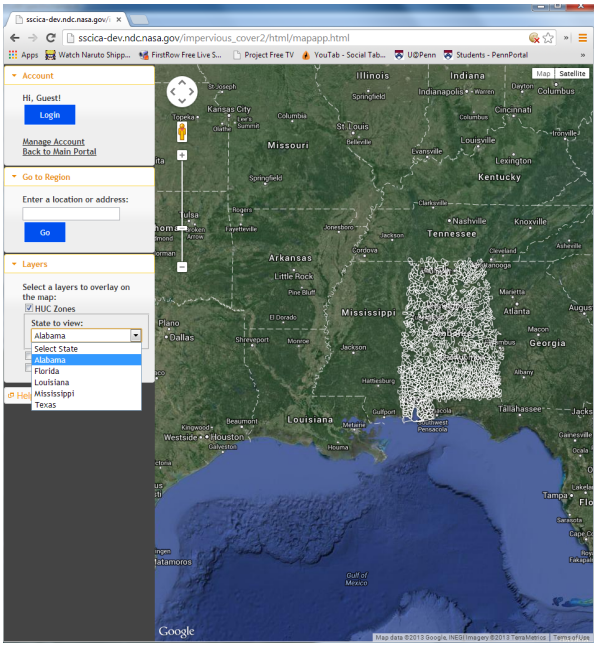
### 3.2.2 Control Panel

The control panel is to the left of the canvas and is used to access account information, center the map over any desired location, and display map overlays. Map overlays are rendered server-side.

HUC boundaries are stored in comma-separated value (CSV) tables in a server hosted by Google, and the layer images are rendered through a Google service called Fusion Tables. A rendering of HUC-12 zones is depicted in Figures 5a and 5b.



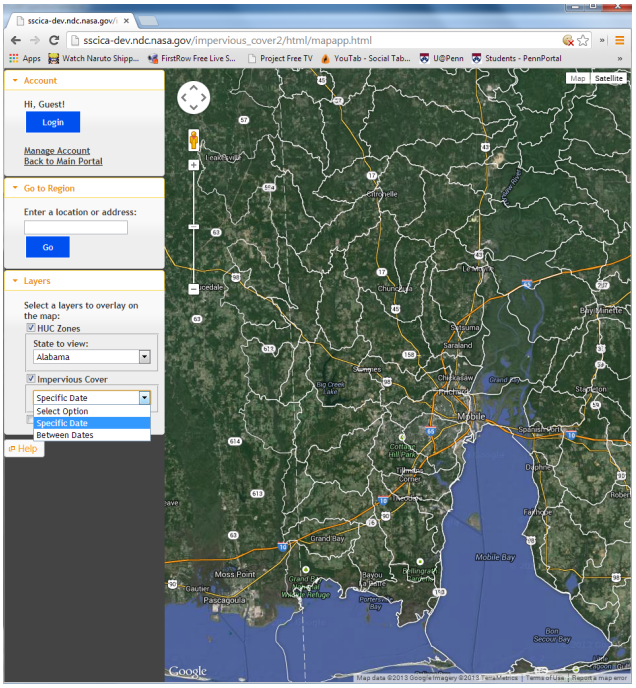
(a) Selecting HUC overlay.



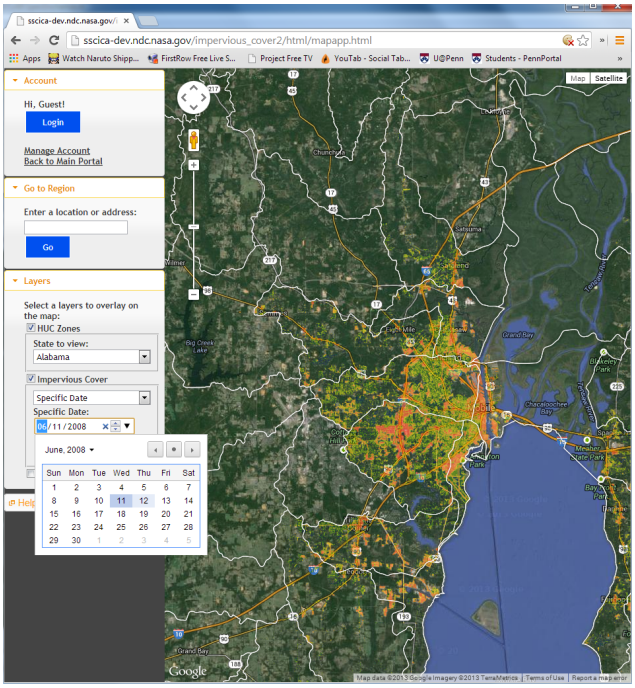
(b) Displayed HUC overlay.

Figure 5: HUC zone overlay demonstration.

The impervious cover image overlay consists of a set of pre-rendered image tiles that display on the canvas according to zoom level and location. The user can select a specific date, but if the data requested does not exist, the closest match will be returned. Figures 6a and 6b demonstrate this feature.



(a) Selecting impervious cover overlay.



(b) Displaying impervious cover overlay.

Figure 6: Impervious cover overlay demonstration.

# 4 Application Back End Structure and Implementation

The primary function of the web-app's back end is to provide resources for the front end and service client-side requests. For this, we set up a CGI-enabled Apache web server equipped with a Python engine, PHP server, and MySQL database server. Login is handled through a PHP session so as to access multiple web pages without handling passwords or other personal information. All client-side queries for retrieving and saving profile information are done through PHP scripts. Application data, however, is handled through two means. First is the Google image overlay engine which accesses overlay images on local and remote servers. In our case, our overlay images for impervious cover are stored in our own server, as are all the raw data. Second is Python's CGI support and the Python Imaging Library (PIL). We are able to read an image and access a specific pixel value or calculate the average value of a region of pixels, and then

return those values to the client using these tools.

# 5 Future Work

While much of the back end infrastructure is complete, there is still significant amounts of front end to back end integration left to do, as well as significant work on back end processing. These tasks include features in the user interface, extra infrastructure work for back end client request processing, and data preparation in an automated fashion. It is recommended, however, that any additional development of the user interface be halted until automation is complete. The functionality, layout, and flow of the user interface heavily depends upon what we expect to provide to the user, which depends upon the type of data that can be provided by the automation process.

## 5.1 Future User Interface Features

Fortunately, most of the user interface features are fairly easy to implement. For instance, users are currently unable to delete their accounts or do password recovery. Both involve very simple MySQL queries to accomplish. However, there are additional features that, while conceptually simple, may not be easy to implement.

One of the more advanced features to be implemented is event loop isolation. Ideally, the app would have adopted a state-machine model to indicate different modes of operation. However, during development many features were simply tacked on for experimentation's sake. Therefore, many HTML document elements are arbitrarily injected via Javascript, and as a result, many features are simultaneously active during different states of operation when some of them should be disabled. For example, when the region selection mode is active, certain circumstances will also leave the region drawing tool active when it should not be. A cleaner version of this app would involve a global state variable that would indicate what document elements would be inserted or removed and what event listeners would be added or removed.

Some simpler features yet to be implemented are region editing and a polygon drawer for alternative region selection. Currently, regions are named after the reverse geocoded approximate address of the rectangular region's center. The user should be able to rename the region. Also, the user may prefer to select regions of arbitrary shape. While this may make calculating impervious cover data difficult, it would make for a more intuitive tool.

## 5.2 Future Back End Client Request Processing

A highly desired feature is direct GEOTIFF access for retrieving PIMP values. Currently, requests for impervious cover data open the pre-rendered tile images and estimate the percent imperviousness based on a grayscale conversion of these RGB images. This does not provide an accurate value because these images consist of three-channel pixels that explicitly display colors of a heatmap. What is desired is a single channel height image with an associated color table so that each pixel contains a specific value directly related to percent imperviousness. To do this, we need a tool that can read GEOTIFF files. The Geospatial Data Abstraction Library (GDAL) provides tools to read and manipulate GEOTIFF files. The GDAL package also includes Python bindings that can be used in lieu of PIL for processing raw impervious cover data.

## 5.3 Future Data Preparation

One of the biggest challenges to overcome is fully automating data production. Currently, an automated process does not exist, but one can be implemented. Automation comes in two parts: creating the impervious cover product and creating client-side resources.

The algorithm described in Appendix A is a step towards automating impervious cover data production. It is simply a matter of converting the mathematics into functioning code. Since so many resources are available in convenient Python libraries, Python is the recommended language for automation.

A process that has been somewhat automated is image overlay generation. Since the map canvas used in the web app is a Google engine, we must produce products that are compatible with Google's tools, namely overlay tiles images. The map canvas that Google Maps displays is, for the sake of design simplicity, a set of images with no embedded data. Depending on the level of zoom on the map, these images are appropriately swapped with a tile with adjusted resolution and detail. The impervious cover overlay must be produced the same way. A convenient tool for creating these tiles is available with the GDAL Python bindings. A python script called `gdal2tiles.py` will take a GEOTIFF input and convert it to Google Maps overlay tiles.

# Appendices

## A Current Data Generation Process

The current process of generating impervious cover data consists of three main steps: spectral enhancement of Landsat data, terrain classification of raw and enhanced Landsat data, and assignment of imperviousness percentage values based on the classifications.

### A.1 Spectral Enhancement

Spectral enhancement algorithms include Normalized Difference Vegetation Index (NDVI), Principal Component Analysis (PCA), and Tasseled-Cap Transformation. These calculations provide additional information about each Landsat scene. NDVI tells us whether or not a scene contains live green vegetation. PCA provides the brightness and greenness of each scene. Tassled-Cap transformation isolates data related to vegetation studies.

**NDVI** NDVI is calculated using the red and near-infrared bands in a Landsat scene. Let $i_R$ and $i_{NIR}$ represent the visible red and near-infrared bands of a Landsat scene, where $i_R, i_{NIR}$ is an $m \times n$ pixel image. We produce a new image, $i_{NDVI}$, where

$$i_{NDVI} = \frac{i_{NIR} - i_R}{i_{NIR} + i_R}$$

**PCA** PCA is a simple eigenvalue problem performed on the six reflectance bands covering the visible blue, green, red, near-infrared (NIR), and mid-infrared frequencies. The process extracts the least correlated features of a set of images and produces a composite image of those features. The following algorithm produces the desired output, where our images are the six reflectance bands, $N = 6$, $m = 7161$, and $n = 8131$ for Landsat7 scenes.

Given a set of $N$ images $I = \{i_1, i_2, ..., i_N\}$ where each image is $m \times n$ pixels in size, we reshape each image into a vector of length $mn$. We do this by taking an image

$$i = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix}$$

and reshaping it to

$$i' = \{a_{1,1}, a_{1,2}, \cdots, a_{1,n}, a_{2,1}, a_{2,2}, \cdots, a_{2,n}, a_{m,1}, a_{m,2}, \cdots, a_{m,n}\}$$

Each reshaped image $i'_1, i'_2, \cdots, i'_N$ is put in an $N \times mn$ matrix

$$D = \{i'_1, i'_2, \cdots, i'_N\}^T$$

From this matrix, we find the mean $\mu$ of the average which involves averaging each column in $D$. We subtract this mean image from each row in $D$, giving us a mean-centered data matrix $U$.
We then calculate the covariance matrix from the mean-centered matrix,

$$\Sigma = \frac{U^T U}{N - 1}$$

and find the eigenvectors and eigenvalues of $\Sigma$ with the formula

$$\Sigma = \Phi \Lambda \Phi^T$$

where $\Phi$ is the eigenvector matrix from the eigenvalues derived from the following equation

$$det(\Sigma - \lambda I) = 0, \text{ and } \Lambda = \begin{pmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_N \end{pmatrix}$$

Using the first two column eigen vectors in $\Phi$, we create a composite image with a linear combination of each original image scaled to lowest correlation. Given

$$\Phi = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,N} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,N} \\ \vdots & & \ddots & \vdots \\ a_{N,1} & a_{m,2} & \cdots & a_{m,N} \end{pmatrix}$$

Let $v_1 = \{a_{1,1}, a_{2,1}, \cdots, a_{N,1}\}^T$, $v_2 = \{a_{1,2}, a_{2,2}, \cdots, a_{N,2}\}^T$, we produce our final images

$$i_{PC1} = I^T v_1, i_{PC2} = I^T v_2$$

each being $n$ length vectors. We resize the vector into an $m \times n$ matrix

$$i = \{a_{1,1}, a_{1,2}, \cdots, a_{1,n}, a_{2,1}, a_{2,2}, \cdots, a_{2,n}, a_{m,1}, a_{m,2}, \cdots, a_{m,n}\}$$

to

$$i = \begin{pmatrix} a_{1,1} & a_{1,2} & \cdots & a_{1,n} \\ a_{2,1} & a_{2,2} & \cdots & a_{2,n} \\ \vdots & & \ddots & \vdots \\ a_{m,1} & a_{m,2} & \cdots & a_{m,n} \end{pmatrix}$$

**Tassled-Cap Transformation** The Tassled-Cap Transformation provides a set of images indicating vegetation properties. The properties we are concerned with are brightness, greenness, and wetness, which provides information about a region's diversity in foliage, grass, and tree canopy cover. To obtain these three images, we take three linear combinations of the six visible bands, each weighted according a set of coefficients like Table 1. Table 1 refers to the coefficients to calculate the brightness, greenness, and wetness of a scene using Landsat4 data. For different sensor data, different coefficients must be used.

| | Band 1 | Band 2 | Band 3 | Band 4 | Band 5 | Band 7 |
|---|---|---|---|---|---|---|
| Brightness | 0.3037 | 0.2793 | 0.4343 | 0.5585 | 0.5082 | 0.1863 |
| Greenness | -0.2848 | -0.2435 | -0.5436 | 0.7243 | 0.0840 | -0.1800 |
| Wetness | 0.1509 | 0.1793 | 0.3299 | 0.3406 | -0.7112 | -0.4572 |

Table 1: Weights for Tassled-Cap Transformation of Landsat4 Data.

Therefore, we create the following matrix,

$$T = \begin{pmatrix} c_{1,1} & c_{1,2} & \cdots & c_{1,N} \\ c_{2,1} & c_{2,2} & \cdots & c_{2,N} \\ c_{3,1} & c_{3,2} & \cdots & c_{3,N} \end{pmatrix}$$

Where $c_{i,j}$ is a coefficient in the appropriate table, and $N$ is the number of bands being used. In the case of Landsat4 data, $N$ is 6.

Taking the vectorized images from PCA, $I$, we obtain the three desired outputs,

$$I_{TC} = I^T T^T$$

providing a $nm \times 3$ matrix. Each $nm$ length column is resized into an $m \times n$ image matrix as done in the last step of PCA.

## A.2 Terrain Classification

Terrain classification is a process that categorizes each point on a satellite scene as part of a certain type of terrain. There are two approaches for accomplishing this task. The first is an unsupervised classification that takes prepared data and clusters them according to thresholded multivariate statistical parameters. The second is a supervised classification that compares statistical values to actual proven data obtained from field work or hands-on image analysis by a trained professional.

In our process, we have two 12-channel images representing our satellite scene. Each image represents the same geographic location with and without leaf cover. Each image is composed of the following channels:

- 6 visible bands from raw satellite image
- 1 channel from NDVI
- 2 channels from PCA
- 3 channels from Tassled-Cap

These two images are joined into a single 24-channel image, where pixels can now be categorized and clustered together based on similarity in channel values.

**Supervised Classification** Supervised classification is an *a posteriori* analysis method that compares previously established classification data, known as training samples, with the unclassified data. The satellite scenes are converted from images to multivariate statistical parameters, and are masked with the training samples.

**Unsupervised Classification** Unsupervised classification is an analysis method that categorizes unlabeled data. Our process uses the ISODATA algorithm, a modified k-means algorithm that uses a supervisor to adjust the number of clusters on the fly by merging or splitting clusters using various heuristics such as thresholding point distances to cluster centers. The following describes a high-level overview of an example implementation.

We define a few algorithm parameters:

- $k_0$ = initial number of clusters. The number of clusters are expected to change as the algorithm progresses through each iteration.
- $t_{max}$ = maximum number of iterations. This algorithm is non-convergent, so we define some iteration limit $t_{max}$ to stop the minimization.
- $n_{min}$ = minimum number of points required to be considered a cluster.
- $L_{min}$ = minimum distance between two cluster centers.
- $P_{max}$ = maximum number of cluster pairs that can be merged per iteration. This is to prevent the merging of three or more overlapping clusters simultaneously.

We take the input image, $I$, that is $r$ pixels tall and $c$ pixels wide. For simplicity, we reshape this image into a $1 \times rc$ row vector

$$I' = \{\mathbf{a}_{1,1}, \mathbf{a}_{1,2}, \cdots, \mathbf{a}_{1,c}, \mathbf{a}_{2,1}, \mathbf{a}_{2,2}, \cdots, \mathbf{a}_{2,c}, \mathbf{a}_{r,1}, \mathbf{a}_{r,2}, \cdots, \mathbf{a}_{r,c}\}$$

or

$$I' = \{\mathbf{a}_1, \mathbf{a}_2, \ \ldots \ , \mathbf{a}_{rc-1}, \mathbf{a}_{rc}\}$$

We treat each pixel $\mathbf{a}$ as a vector in $\mathbb{R}^d$,

$$\mathbf{a}_p =< a_{p,1}, a_{p,2}, \ \ldots \ , a_{p,d} >$$

$$p \in \mathbb{Z} \mid 1 \leq p \leq rc$$

Given our images consist of 24 channels, $d$ is 24 for our purposes.

From an initial set of pixels, $i'$, we randomly sample $k = k_0$ initial cluster centers $Z = \{\mathbf{z_1}, \mathbf{z_2}, \ \ldots \ , \mathbf{z_k}\}$. We assign each pixel in $I'$ to its closest cluster center $z$, creating $k$ subsets $S_j \subseteq I'$, so that for any $\mathbf{a} \in I'$,

$$\mathbf{a} \in S_j \text{ if } \|\mathbf{a} - \mathbf{z}_j\| < \|\mathbf{a} - \mathbf{z}_i\|, \ \forall i \neq j$$

A cluster may consist of no fewer than $n_{min}$ points, so letting $n_j$ denote the number of points in $S_j$, we remove all clusters $S_j$ where $n_j < n_{min}$.

We now reassign the center point of each cluster to that cluster's centroid, or

$$\mathbf{z_j} = \frac{1}{n_j} \sum_{\mathbf{a} \in S_j} \mathbf{a}, \text{ for } 1 \leq j \leq k.$$

If clusters were removed prior to this step, we reassign points to their appropriate cluster centers and threshold the cluster sizes repeatedly until all clusters meet the size requirement.

We let $\Delta_j$ be the mean distance of points in $S_j$ to their cluster center $z_j$, and $\Delta$ be the mean of all $\Delta_j$.

$$\Delta_j = \frac{1}{n_j} \sum_{\mathbf{a} \in S_j} \|\mathbf{a} - \mathbf{z_j}\|, \text{ for } 1 \leq j \leq k$$

$$\Delta = \frac{1}{n} \sum_{j=1}^{k} n_j \Delta_j$$

If this is not the last iteration, then for each cluster $S_j$, we compute a vector $\mathbf{v}_j =< v_{j,1}, \ \ldots \ , v_{j,d} >$ where the $i$th component of $\mathbf{v}_j$ is the standard deviation of all $i$th components of the vectors $\mathbf{a} - \mathbf{z_j}$, $\forall \mathbf{a} \in S_j, 1 \leq j \leq k$, or

$$v_{j,i} = \left( \frac{1}{n_j} \sum_{\mathbf{a} \in S_j} (x_i - z_{j,i})^2 \right)^{1/2}, \text{ for } 1 \leq j \leq k \text{ and } 1 \leq i \leq d.$$

Let $v_{j,max}$ be the largest component in $\mathbf{v}_j$ and $\sigma_{max}$ be the largest allowable user-defined standard deviation of each point $\mathbf{a}$ from its cluster center along each component. For each cluster $S_j$, if the largest component in $\mathbf{v}_j$, $v_{j,max} > \sigma_{max}$ and if either

$$((\Delta_j > \Delta) \text{ and } (n_j > 2n_{min} + 2)) \text{ or } k \leq \frac{k_0}{2}$$

then $k$ is incremented and $S_j$ is split into two clusters by replacing its center with two cluster centers around $\mathbf{z_j}$ and separated. We go back to reassigning points to their appropriate cluster centers.

If no clusters are split, we determine all the pairwise distances between clusters,

$$d_{i,j} = \|\mathbf{z}_i - \mathbf{z}_j\|, \text{ for } 1 \leq i < j \leq k$$

and proceed to check if all distances meet the minimum intercluster distnace criterion, $d_{i,j} < L_{min}$. Any two clusters that do not meet this criterion are subject to merging, in which their points are grouped together with a new cluster center that is their weighted average of the original cluster centers, or

$$\mathbf{z}_{i,j} = \frac{1}{n_i + n_j}(n_i\mathbf{z}_i + n_j\mathbf{z}_j)$$

and decrement $k$. $k$ must not be decremented more than $P_{max}$ times. If we have not met our maximum iterations, then we go back to assigning points to their appropriate cluster centers.

## A.3 Percent Imperviousness Estimator

Urban materials and other terrian have particular spectral emission signatures. Based on the terrain classifications, we can finally assign imperviousness percentage values to each pixel using a look-up table. The National Land Cover Database, or NLCD, provides a database of terrain classes as well as their PIMP values.