University of Nevada
Reno

Identification of Tree Locations in Geographic Images

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science
in Computer Engineering

by

David T. Brown

Dr. Frederick C. Harris, Jr., Thesis Advisor

December, 2008

Abstract

This thesis describes an interactive image analysis algorithm for the correct placement of trees within a virtual reality wildfire visualization system. The wildfire visualization system is VFIRE (Virtual Fire in Realistic Environments), which uses fire simulation results produced by software programs such as FARSITE to place the user into an immersive, artificial environment where he or she can see graphical depictions of real or fictitious wildfire events. The purpose of VFIRE is to assist in training and planning for actual fires, and our purpose is to place trees within the artificial environment at locations corresponding to their locations in the real environment. Our goal was to achieve adequate tree-placement accuracy using any, arbitrarily chosen image, assuming no constraints on the minimum suitability of that image for tree detection. This goal led to the selection of a template-matching algorithm wherein the user selects a particular tree as a template and begins to search the image for similar-looking trees. Tree detection results are presented for a 1-meter panchromatic image of the current area of interest, Kyle Canyon, in Southern Nevada. It is difficult to perceive the aesthetic effect of tree-detection inaccuracies in terms of numeric error rates. Therefore, screen shots are provided representing tree-detection results placed into three broad, subjective categories: good, adequate, and poor. The results obtained were found to be roughly fifteen percent good, seventy percent adequate, and fifteen percent poor, where accuracy was directly related to the suitability of the terrain for tree detection in each particular region of the image.

Acknowledgments

This work is funded by the STTC CAVE Project (ARO# N61339-04-C-0072) at the Desert Research Institute.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

VFIRE is an application that uses Virtual Reality (VR) technology to allow a user to be immersed in an artificial environment where real or fictitious wildfire scenarios can be visualized [25]. The purpose of such simulations is to provide useful information about fire spread in a particular region under various wind, humidity, and other environmental conditions. Currently, the region of interest is Kyle Canyon, near Mt. Charleston, Nevada.

Since the simulation is intended to model an actual place, it is desirable that items within the simulation be located in the same place they would be located in the real environment. The purpose of this work is to accomplish such realistic item placement without the user having to specify the placement of each item individually. This project provides a facility to make the process more (though not fully) automatic. Some assistance from the user is still required.

We are primarily focused on the placement of trees because trees constitute a large proportion of the fuel in a wildfire and because trees dominate the landscape in the current VFIRE region of interest. Other items, such as buildings, can be placed in the simulation manually, but trees are currently the only items that can be placed automatically.

We perform automatic tree placement by interpreting user-supplied information with reference to an image file showing the region of interest, and this image file is supplemented with information from various vegetation attribute files as well. If either of these file sets is not available, tree placements will be less accurate. If neither

is available, tree placements will be completely random.

This project is currently configured to use satellite photographic image files showing the region of interest. Because satellite images tend to have low resolution, the current implementation cannot make use of algorithms requiring fine detail in the image. Instead, we use a method more similar to blob detection, which selects blobs based on size and shape, ignoring details of the interior of the blob.

It is the type and variety of data being used that distinguishes this work from others. Much work has already been done to develop general-purpose interactive image analysis tools, tools for locating trees within relatively high-resolution aerial photo images, and tools to locate trees within groves of vegetation having uniform appearance. This project, on the other hand, is similar to a general-purpose interactive image analysis utility that incorporates vegetation maps to compensate for poor image quality and outputs tree locations directly in georeferenced coordinates. In addition, this utility allows trees to be placed based on vegetation maps alone in cases where photographic images are not available.

The rest of this thesis is structured as follows: Chapter 2 provides background information on basic image processing methods and existing item placement tools and algorithms. Chapter 3 explains the conceptual operation of this project. The design of this project is presented in Chapter 4. Information on the current implementation of this project as well as the results obtained so far are given in Chapter 5. Chapter 6 presents our conclusions and identifies useful improvements to this project to be made in the future.

Chapter 2

Background

2.1 Geographic Data

Information regarding the surface of the earth can be produced in a number of different ways. In one such method, known as *remote sensing*, a camera or other sensing device is taken to a high altitude and aimed at the earth to collect information. Alternatively, data can be acquired by means of *proximate sensing*, which is performed by people using measuring devices at ground level [29]. For any application, the choice of methods and technologies is determined by the type, quality, and cost constraints of the required data. Much of the geographic data that is collected consists of geographic imagery such as maps and photographs.

2.1.1 Aerial Imagery

In the case of remotely sensed imagery, a common practice is to attach a downward-facing camera or other sensor to an aircraft and fly over the area of interest, taking pictures. This type of remote sensing, known as *aerial imagery*, has been done for almost as long as cameras have existed. Early aerial imagery was done by taking photographs from balloons. Subsequently, airplane-mounted cameras were used extensively for military reconnaissance in World War I [32]. Today, both airplanes and helicopters are used in the production of aerial imagery. Airplanes have the advantage that they can cover a given area more quickly, but helicopters can be positioned with greater control than airplanes [18].

In addition, unmanned aerial vehicles (UAV) are increasingly being used as platforms for aerial imagery. They can often be used to produce aerial imagery at a lower cost than can be done with manned aircraft [40]. UAVs encompass an extraordinary variety of vehicles, and the capabilities of UAVs vary greatly from one vehicle to the next. For example, their payloads may range from less than 1 kg up to several hundred kg, and their maximum altitudes may range from one thousand feet up to altitudes in excess of ninety thousand feet [27]. In spite of this diversity, UAVs do not always give the most cost-effective solution over all terrains at all altitudes, and the need for a reliable connection between UAVs and their remote pilots further limits their applicability. Therefore, there will likely continue to be some need to use manned aircraft in remote sensing for the foreseeable future.

2.1.2 Satellite Imagery

In addition to aircraft, spacecraft can also be used for the collection of remotely sensed data. Spacecraft have been used for remote sensing since the 1940's. At that time, cameras were placed inside V2 rockets acquired from Germany to take pictures of the Earth. These rockets did not reach orbit, but they were able to take photos at high altitude. Subsequently, in the 1960's, astronauts and cosmonauts took pictures of the Earth from their orbiting spacecraft [26]. Today, remote sensing from space is commonly done with Earth-orbiting, unmanned satellites.

Satellite imagery has several advantages and disadvantages compared to aerial imagery. Among the advantages of satellite imagery is the fact that the process of contracting to have aerial imagery produced requires planning to make sure that an aircraft is available and that weather conditions are acceptable on the acquisition date. On the other hand, the most recent satellite image of a region can generally be ordered with no preparation at all. For repeated imaging of the same region, aerial imagery can be more expensive because the aircraft has to be sent up each time an image is taken. A satellite has the advantage that it remains in orbit and can capture additional images with no additional cost [8].

Among the advantages of aerial imagery is the fact that it can be scheduled for a specific date. A satellite image of a particular region can only be produced on the dates when the satellite happens to be passing over it. In addition, aerial imagery often has much better resolution because it is taken much closer to the ground. Aerial imagery can have resolutions as good as about 2 inches. The best satellite imagery resolution is about 2.4 meters for CIR and about 0.6 meters for panchromatic images [8].

2.1.3 Types of Photography

In standard color photography, a camera is used that responds to light in a way similar to the way that the human eye responds to it. Such a camera will show a scene just as we would see it with our own eyes. However, there are other ways to depict a scene [5].

Another depiction commonly used in satellite imagery is CIR (Color Infrared) imagery, in which the green component is depicted as blue, the red is depicted as green, and near-infrared component is depicted as red. The result is a *false color* photo which allows us to see the infrared energy emitted or reflected from objects as the amount of red in the image [5]. Figure 2.2 shows a false color version of the true color image in Figure 2.1.

Images can also be shown in a panchromatic format. Originally, this term referred to standard black and white photography [13], where the brightness at any location in the image corresponds to the total combined visible-spectrum light energy detected by the camera at the corresponding location in the scene. Currently, the term has a looser definition, applying to any imagery where a large portion of the visible spectrum and, possibly, some portion of non-visible spectra are taken together to produce a monochrome image. For example, the IKONOS and QuickBird satellites each possess panchromatic cameras sensitive to light over nearly the entire visible spectrum plus the near infrared spectrum as well [15].

Geographic images are often captured using multispectral scanners. These are

Figure 2.1: True Color Image [5, page 45].

Figure 2.2: False Color Image [5, page 45].

scanners that can be set to selectively respond only to a particular light spectrum, visible or invisible [5]. A multispectral scanner can repeatedly scan the same scene each time in a different spectrum. The purpose is to detect selected features such as water or vegetation, which show up differently depending on the light spectrum. Hyperspectral scanners are very advanced multispectral scanners, having the ability to select very narrow light spectra from the available range. This allows a particular feature within the scene to be identified more precisely than can be done with multispectral scanners [9].

2.1.4 LiDAR

A height map is a mapping in which an altitude value is associated with each spot on a two-dimensional surface. LiDAR, (Light Detection And Ranging) is a form of surface scanning in which a laser beam is used to produce such a height map [41].

Aircraft-mounted equipment able to produce and detect laser beams is flown in a scanning pattern over the area of interest, and during the flight, the equipment projects laser beams toward the ground, measuring how long it takes for the reflections to return. The equipment records the delay times, which it then uses, in conjunction with its own altitude and projection angle, to determine the altitude of whatever the laser beam struck [42]. This process produces data in the form of a *point cloud*, a set of points in three-dimensional space, which can then be further processed into other formats that are easier to interpret [12].

LiDAR data are commonly used in forestry to create maps of forest canopy height, a task which is difficult to accomplish by other means [1]. It is also used for mapping beaches and shorelines [28]. Other uses of LiDAR include, transportation planning, oil exploration, and mining [12].

2.1.5 Digital Elevation Models

A Digital Elevation Model (DEM) is a digital file containing the information for a height map of the ground [46]. They can be derived from LiDAR data or from

ground survey data [38]. Several different types of DEMs are produced by the United States Geological Survey (USGS). These DEMs differ mainly in their coverage areas and data spacing, which is the horizontal distance between each consecutive height measurement [44]. USGS produces DEMs in blocks of 7.5, 15, 30, and 60 minutes of latitude corresponding to a 7.5, 15, 30, and 60 nautical miles square, respectively. DEMs are available with data spacing of 30m × 30m, and they are also available with spacings of 2×2, 3×3, 2×3, and 1×2 arc-seconds [45].

DEMs have a wide variety of applications including the modeling of water flow, creation of relief maps, rectification of aerial or satellite imagery, and rendering 3D visualizations [39].

A DSM (Digital Surface Model) is similar to a DEM except that it gives forest canopy elevation instead of ground elevation. Similarly, a DCHM (Digital Canopy Height Model) is created by subtracting the DEM from the DSM to produce a height map of a forest canopy with no height variations resulting from variations in ground elevation [16].

2.1.6 Vegetation Maps

Data regarding the amount and types of vegetation present in a geographic area can be compiled and displayed as a map. Such maps can be used for wildlife habitat assessment, resource management planning, and fire risk assessment and prioritization [19]. Many vegetation maps are produced by LANDFIRE, a joint project of the U.S. Department of Agriculture Forest Service and U.S. Department of the Interior [21]. Vegetation maps can be produced using data from many different sources. However, LANDFIRE relies heavily on remotely sensed data from the Landsat satellites [20]. Three different LANDFIRE vegetation maps are shown, each having its own color scheme. In Figure 2.3, the colors displayed correspond to the amount of vegetation coverage at each location on the map. The darker green areas have greater tree coverage, and the lighter green areas have less coverage. Other colors correspond to areas in which trees are not the dominant surface feature. In Figure 2.4, the colors

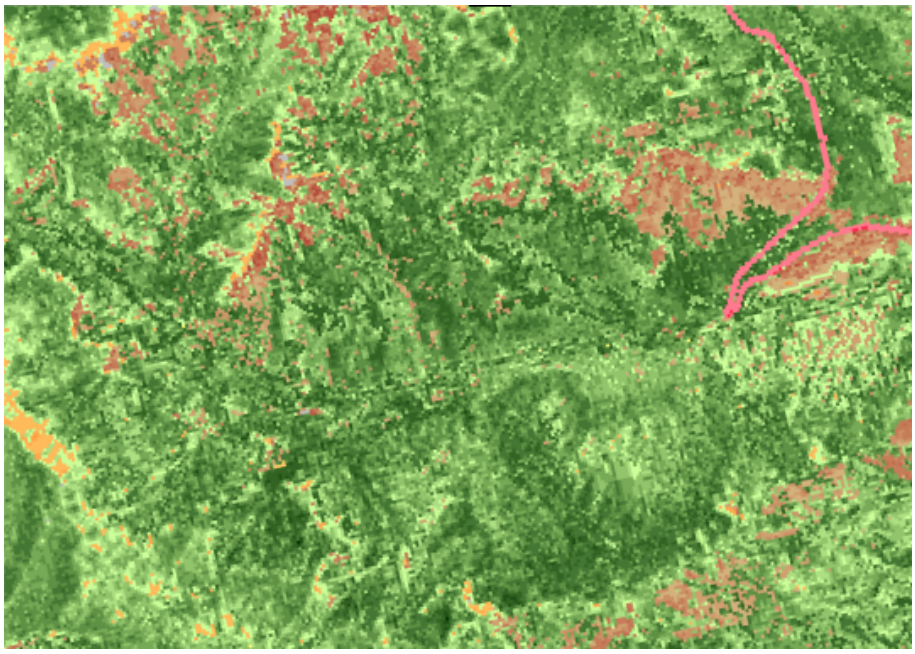

Figure 2.3: Vegetation Coverage of Kyle Canyon.

displayed correspond to the specific type of vegetation dominant in each location, and in Figure 2.5, the colors displayed correspond to the height range of vegetation dominant in each location.

In addition, LANDFIRE has maps depicting several other forest metrics [22]. These other metrics, such as canopy bulk density, canopy height, and canopy fuel weight, can be used, along with atmospheric and weather data, as inputs in fire simulation programs such as FARSITE, which is discussed in Section 2.4.

2.1.7 GDAL

Geographic images are commonly stored as raster data files, and there exist many different raster file formats in which these data can be encoded. Some common file formats include Arc/Info ASCII Grid, Arc/Info Binary Grid, and GeoTIFF [30]. With many different formats available, it is often helpful to use an abstraction library to allow access to the data without concern as to the particular file format used. GDAL (Geospatial Data Abstraction Library) is such a library. It allows data to be retrieved

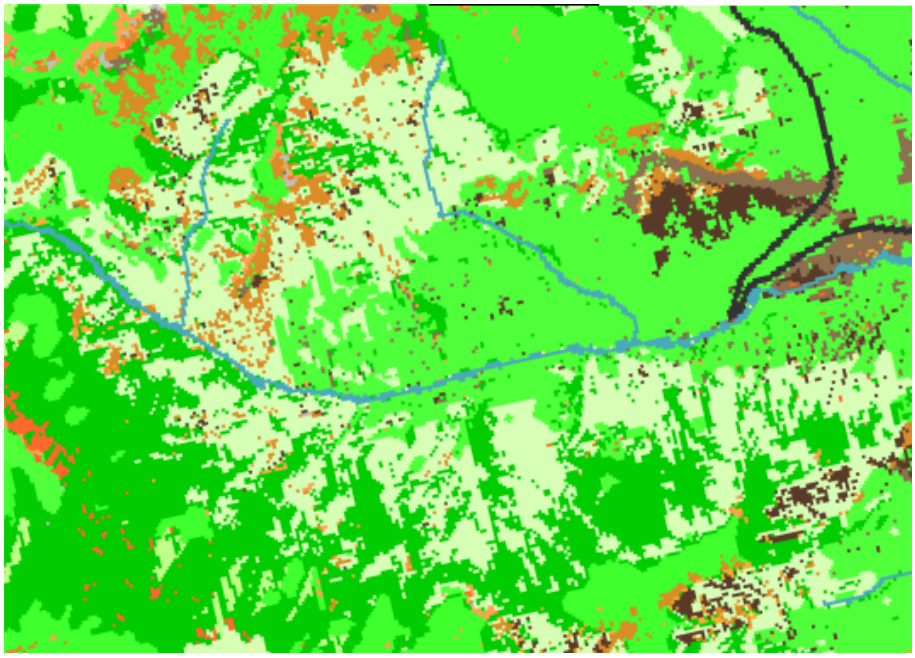

Figure 2.4: Vegetation Types for Kyle Canyon.

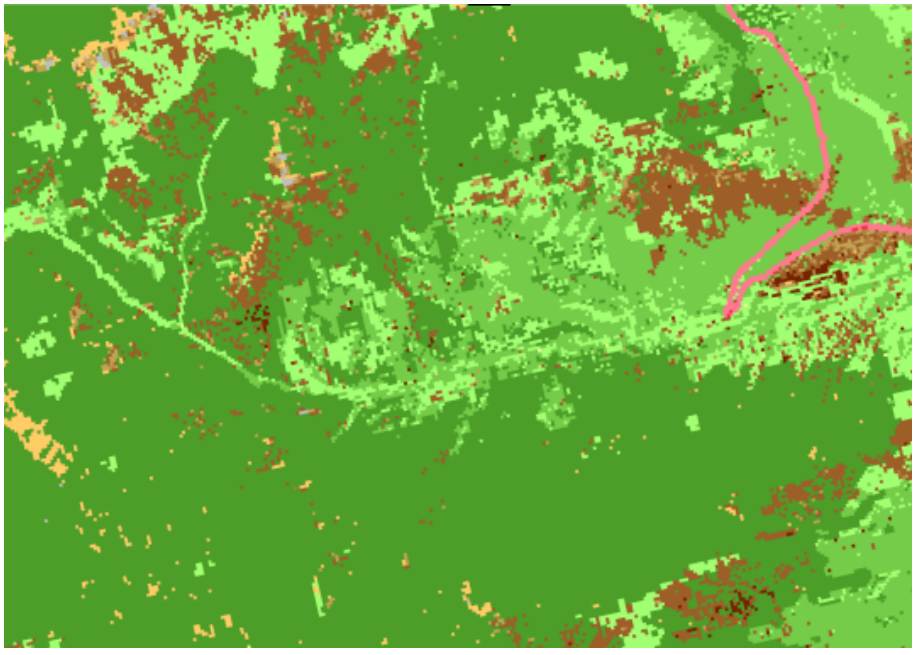

Figure 2.5: Vegetation Heights for Kyle Canyon.

using the same generic access commands for many different file formats including the ones listed above [31].

In addition to the raster data, itself, geographic images also have associated metadata. For example, images of the Earth's surface are generally accompanied by longitude and latitude or some other georeferencing information to establish what part of the world they depict. GDAL allows convenient access to this information as well.

2.2 Image Processing and Analysis

Image analysis can be defined as "the process of describing or evaluating an image based on its parts, properties, or relationships," and image processing can be defined as "the manipulation of images by computer" [43]. From these two definitions, it is apparent that image analysis is the process by which we gain useful information from an image. However, we may need to use techniques from image processing to manipulate the image in the course of acquiring this information. Therefore, our purpose is image analysis, but our methods often involve image processing.

Since images are stored as numeric arrays within a computer, image processing is, in practice, the manipulation of arrays of numbers within a computer. As shown in Figure 2.6, an array containing a matrix $\mathbf{X}$, representing the input image, is transformed by some process into the output image represented by matrix $\mathbf{Y}$, which may or may not have the same dimensions as the input.

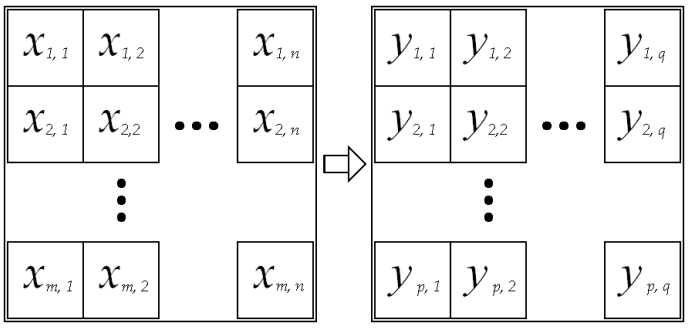

Figure 2.6: Generalization of Image Processing.

A great variety of methods to manipulate numeric arrays has been developed for the purpose of image processing, and there are several ways in which these methods can be categorized. For example, they can be divided into Fuzzy Logic or Discrete Logic methods. Alternatively, they can be divided into spatial-domain or frequency-domain methods. However, one of the most commonly used categorizations is to divide them into *point operations* and *neighborhood* operations.

2.2.1 Point Operations

Point operations are methods of manipulating numeric arrays such that each element in the output array is a function of the corresponding element in the input array [43]. The value of each output element does not depend on any of the elements surrounding the corresponding input element. Equation 2.1 gives the general form of a point operation.

$$y_{i,j} = f(x_{i,j}) \tag{2.1}$$

Point operations generally possess a greater degree of speed and simplicity than neighborhood operations. However, point operations are not always straightforward. They can be arbitrarily complex, depending on the complexity of the function f. A good example of a point operation is the process of changing the brightness of an image. Equation 2.2 shows how this might be done.

$$y_{i,j} = f(x_{i,j}) = \begin{cases} x_{i,j} + 50 & \text{if } x_{i,j} \leq 205 \\ 255 & \text{otherwise} \end{cases} \tag{2.2}$$

Equation 2.2 assumes an array of 8-bit numbers comprising either a monochrome image or the brightness component of a color image. The output image is constructed such that each output element is fifty units brighter than its corresponding input. If the input value is greater than 205, the output value is clipped at 255 because this value corresponds to the color white and white is the brightest that any pixel can be. Figure 2.8 shows the input image of Figure 2.7, noticeably brighter after the application of Equation 2.2.

Figure 2.7: Unprocessed Image.

Figure 2.8: Brighter Image.

2.2.2 Neighborhood Operations

Neighborhood operations are methods of manipulating numeric arrays such that each element in the output array is a function of a neighborhood of elements near the corresponding element in the input array [43]. These operations are inherently more complex than point operations, and they generally require more time to execute. However, this added difficulty is often unavoidable because there are many tasks that cannot be done with point operations alone. The advantage of neighborhood operations is that more information goes into the calculation of each output element, and a greater range of processes can therefore be implemented.

If we assume $\mathbf{X}_{i,j}$ to be a submatrix of $\mathbf{X}$, where i and j are indices into $\mathbf{X}$, as shown in Figure 2.9, then Equation 2.3 shows the general form of a neighborhood operation.

$$y_{i,j} = f(\mathbf{X}_{i,j}) \tag{2.3}$$

For simplicity, a 3×3 submatrix is shown in Figure 2.9. In practice, however, the submatrix can be of any desired dimensions, and the input element corresponding to the output element need not be at the center of the submatrix.

$X_{i-1,j-1}$	$X_{i-1,j}$	$X_{i-1,j+1}$
$X_{i,j-1}$	$X_{i,j}$	$X_{i,j+1}$
$X_{i+1,j-1}$	$X_{i+1,j}$	$X_{i+1,j+1}$

Figure 2.9: Example of input submatrix

We see from Equation 2.3 that the entire neighborhood of elements near the input element goes into the calculation of each output element. This process is done for every index pair (i, j) for which $\mathbf{X}_{i,j}$ is defined. The function, f, is commonly written such that each output element will be a weighted sum of the elements in the input neighborhood, as shown in Equation 2.4, where the $h_{m,n}$ coefficients are the weights specified in the function.

$$y_{i,j} = f(\mathbf{X}_{i,j}) = \sum_{m=0}^{2} \sum_{n=0}^{2} x_{i+m-1,j+n-1} \times h_{m,n} \tag{2.4}$$

A problem often occurs when attempting to process the elements along the edges of the image matrix. Depending on the dimensions of the submatrix, some portion of the neighborhood may lie outside the image matrix for input elements near the edge of the image. This problem can be overcome either by ignoring the edge elements, resulting in a smaller output image, or by padding the input image with a perimeter of extra elements. In the latter case, the values contained in the padding elements should be chosen to minimize their effect on the output values, though some effect may be unavoidable.

Equation 2.4 can be rewritten by taking the $h_{m,n}$ coefficients out of the function definition and putting them into an argument to the function. In this case, Equation 2.4 becomes Equation 2.5, and the function f is reduced to merely taking the dot product of two matrices for each output element [37], where $\mathbf{H}$ is the matrix of weights, a constant matrix having the same dimensions as submatrix $\mathbf{X}_{i,j}$.

$$y_{i,j} = f(\mathbf{X}_{i,j}, \mathbf{H}) = \mathbf{X}_{i,j} \cdot \mathbf{H} \tag{2.5}$$

In addition, if we think of the input image and the matrix of weights as discrete-domain functions of (i, j), the function f becomes almost equivalent to a two-dimensional convolution function, commonly denoted with an asterisk as in Equation 2.6.

$$y(i, j) = x(i, j) * h(i, j) \tag{2.6}$$

The major difference between a two-dimensional convolution and an array of matrix dot products is that the convolution process effectively flips h as if it were a function $(-i, -j)$ instead of (i, j) before multiplying and summing. However, this has no effect as long as the matrix of weights is symmetric about the center.

Several different terms are commonly used to refer to the $\mathbf{H}$ matrix. Such terms include *filter, filter mask, convolution mask,* and *convolution matrix.*

A good example of a neighborhood operation is the process of bluring an image. Figure 2.10 shows a simple filter mask that could be used for this purpose.

The output image is constructed such that each output element is the mean average of the input element and its eight surrounding elements. The filter mask

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Figure 2.10: A Blur Filter.

used in this example is commonly referred to as a *blur filter*. Figure 2.11 shows the input image of Figure 2.7 after three applications of the blur filter.

Figure 2.11: Blurred Image.

2.2.3 Image Segmentation

Image segmentation can be defined as "the process of dividing an image into regions" [43]. Since any region within an image is merely a group of pixels, the process of image segmentation can be thought of as a grouping problem. It is the process of grouping image pixels according to the objects they represent. For example, in

a landscape image, there will be a group of pixels representing one particular tree, another group of pixels representing another tree, and so on. In addition, there will be a group of pixels representing the ground and another group of pixels representing the sky. Therefore, each object depicted in the image is represented by a particular pixel group, and it is the purpose of image segmentation to put each pixel into its correct group, though it may not be known, initially, exactly what those groups represent. That determination is generally left to be made in a subsequent stage of processing [34].

In most cases, the purpose of image segmentation is to divide the image into regions so that the objects being represented by those regions can be correctly identified. In practice, however, image segmentation can be a very difficult task for computers to accurately perform automatically. As a result, many software tools have been developed that allow a human user to assist the computer in the segmentation process, but whether a system is automatic or interactive, two of the most commonly used segmentation algoritms are *thresholding* and *edge detection* [35].

2.2.4 Image Thresholding

Thresholding methods assign each pixel to a particular group based on its gray level or color. Image thresholding is so named because it requires one or more thresholds to be established. Once the thresholds have been established, all pixels that lie between the same two thresholds (or on the same side of the threshold, if there is only one) are assigned to the same pixel group. There are many different ways to threshold an image, but thresholding methods can be broadly considered to fall into two categories. There is *global thresholding*, in which a single set of thresholds is applied to the entire image, and there is *local thresholding*, in which the image is divided into subregions and each subregion has its own set of thresholds. The image segmentations produced by local thresholds are generally more accurate than those produced by global thresholds, but in either case, the accuracy is strongly dependent on the particular choice of threshold values. Although thresholding can be used on

color images, the effect is often easier to see in monochrome images. Figure 2.13 shows the image of Figure 2.12 after applying a global threshold of 50. All pixels darker than 50 are set to zero (black), and all pixels equal to or brighter than 50 are set to 255 (white).

Figure 2.12: Unprocessed Gray-Scale Image.

2.2.5 Edge Detection

Edge detection is the process of highlighting the boundaries between regions of an image. There are many different types and qualities of edge detection. In general, edge detection is accomplished by filtering an image with an edge-detection filter. One of the difficulties of edge detection is that the region representing a particular object may, itself, contain many edges and these edges may be highlighted as if they delineated separate objects. To overcome this problem, a *scale-space* representation is often used in the process of edge detection. In a scale-space representation of an image, some of the fine detail has been removed, preventing such details from producing any highlighted edges in the edge-detection stage. The removal of detail

Figure 2.13: Image After Applying Threshold of 50.

is commonly done by filtering the image with a Gaussian blur filter, which is derived from the Gaussian probability density function shown in Equation 2.7.

$$G(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{\frac{-(x-\mu)^2}{2\sigma^2}} \tag{2.7}$$

The value of sigma specifies the strength of the filter. A greater value of sigma corresponds to a greater loss of image detail. The reason for the use of a Gaussian filter is that a Gaussian filter will reduce the existing detail in an image without introducing any extraneous detail which may occur with other filters. Figure 2.14 shows a conceptualization of scale space, where f is a function of x, and the large variations in $f(x)$ diminish as sigma increases [47].

A common type of scale-space edge detection filter is the Laplacian of Gaussian (LoG) filter. It combines a Gaussian blur filter with a Laplacian edge detection filter to produce an edge-highlighted image at whatever level of detail is specified in the parameter sigma. Figure 2.15 shows an image after processed with a LoG filter using three different values of σ [36].

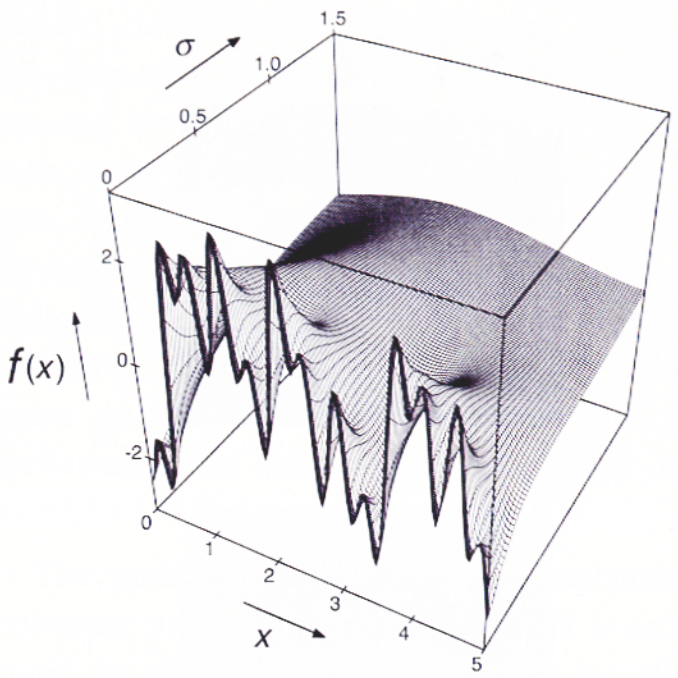

Figure 2.14: Conceptualization of Scale-Space [47].

Once the image has been segmented by whatever method, the next stage of analysis can begin. Often, this will be the process of identifying what each segmented region of the image actually represents.

2.2.6 Template Matching

Another image processing method that can be used as the initial step in an image analysis procedure is *template matching*. This method can be used when the appearance of an object is already known, but its location within an image needs to be determined. Template matching is a filtering process in which the filter mask is, itself, an image of the object that needs to be found. The filter mask is referred to as a *template*, and the output of the process is a *correlation image* containing a bright spot (or dark spot, depending on the algorithm) where the object is located. A second stage of processing searches the correlation image for bright spots to locate the object. If there is more than one instance of the object in the source image, each instance will have an associated bright spot in the correlation image, and the object

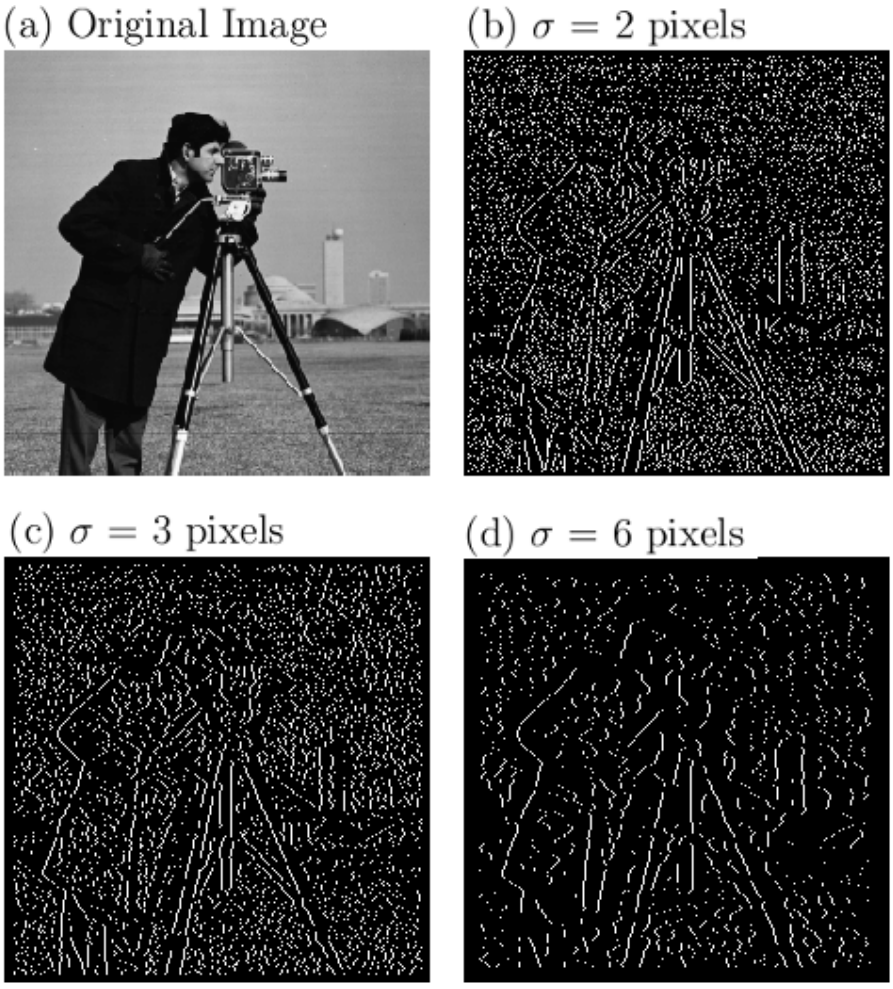

Figure 2.15: LoG Scale-Space Edge Detection at Different Values of Sigma [36].

will be identified once for each such spot. The objects in the image do not need to match the template perfectly. Closer matches produce brighter bright spots, and the algorithm can be tuned to accept any bright spots exceeding the brightness threshold established for the given application.

2.3 Virtual Reality

Virtual Reality can be defined as "the use of computer modeling and simulation that enables a person to interact with an artificial three-dimensional (3-D) visual or other sensory environment" [7]. As this definition indicates, virtual reality is not just visual. Technologies exist to provide virtual reality displays for other senses as well. However, virtual reality is most commonly associated with the computer graphics technologies used to create artificial 3D visual environments. In addition, virtual reality systems often include specialized input and output devices, designed specifically for that purpose.

To determine whether a particular technology or experience qualifies as virtual reality, four things need to be considered. A virtual reality experience must include a virtual world, immersion, sensory feedback, and interactivity. A virtual world is "a description of a collection of objects in a space and the rules and relationships governing those objects" [2]. Immersion is a state of mind in which the artificial environment is perceived as if it were real. The user has the perception of being inside the artificial world. This can be accomplished through strong mental engagement in a fictitious reality or physically by connecting one or more senses to an artificial reality system [2]. Sensory feedback is the sensory input that a person receives in response to his or her actions within an environment [2]. For example, in the physical environment, if a person touches an object, that person will feel what he or she is touching. This particular form of sensory feedback is known as *haptic* feedback. Similarly, if a person turns his or her head, the scene that he or she sees will change in response [2]. This is a form of visual feedback. Interactivity is the ability of objects in the environment to be affected by actions of the user [2]. For example, when a person

exerts a sufficient force on an object in the physical environment, it begins to move in response. Therefore, the physical environment would be considered interactive. Video games are also interactive because objects within the game can be moved, destroyed or otherwise affected by actions of the user. Similarly, a 3D graphical virtual world is interactive if it contains objects that can be moved or in some other way affected by the actions of the user [2].

2.3.1 Visual Depth Cues

It is common for virtual reality display devices to have the ability to produce the same visual cues that we receive from the physical world. These cues give us an impression of the size, proximity, rate of motion, and relative position of the objects we see.

Standard video displays are *monoscopic*, meaning that they display only one video stream at a time and both of the viewers eyes receive exactly the same image. Some visual cues such as interposition, size, linear perspective, and motion parallax can readily be produced by standard video display devices [2]. Interposition is the situation in which our view of an object is partially occluded by another, letting us know that the object blocking our view must be closer [2]. Size is a visual cue because objects appear smaller as they move away and larger as they come closer. Linear perspective is a cue that takes advantage of the linear path that objects appear to follow as they move closer to us or further away. Motion Parallax refers to the fact that, as we move, distant objects appear to be moving slowly and close objects appear to be moving fast [2]. Figure 2.16 shows the way changing position can affect depth cues.

Video display devices designed for use in virtual reality applications often provide a *stereoscopic* display, meaning that they display two different multiplexed video streams and that each of the viewers eyes receives only one of those video images. Each video image depicts the same scene from a slightly different perspective, simulating the human visual system in which the brain receives two slightly different images of the same scene, one image from each eye. The eyes must turn toward each other

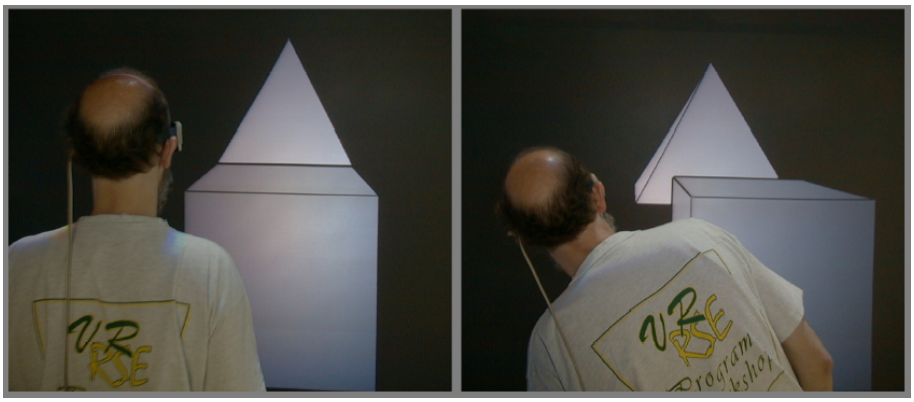

Figure 2.16: Effect of Perspective on Virtual Reality Display [2, page 120].

to force the differing images to coincide, and the brain estimates the distance to the object being observed from the amount by which the eyes are deflected toward each other. This form of depth perception is similarly induced by the differing images produced by stereoscopic displays.

2.3.2 Video Display Devices

With the addition of some virtual-reality graphics hardware to provide stereoscopic display and other VR capabilities, an ordinary desktop computer can be used as a virtual reality system. This is often referred to as a *fishtank* VR system [2]. Such systems have the advantage of being relatively inexpensive, but their small screens and stationary placement make them very limited compared to the other VR systems in common use [2].

Projection-based displays are usually much larger than fishtank system displays. They are stationary like fishtank systems, but because their screens are so much larger, they fill the user's field of view more fully. In addition, large projection-based displays have been constructed with multiple screens, each acting as one side of a partial or complete cube, large enough for people to walk inside. A three-sided projector system is shown in Figure 2.17.

Another type of VR display is the Head Mounted Display (HMD), which is worn on the head as shown in Figure 2.18. Head Mounted Displays are significantly less

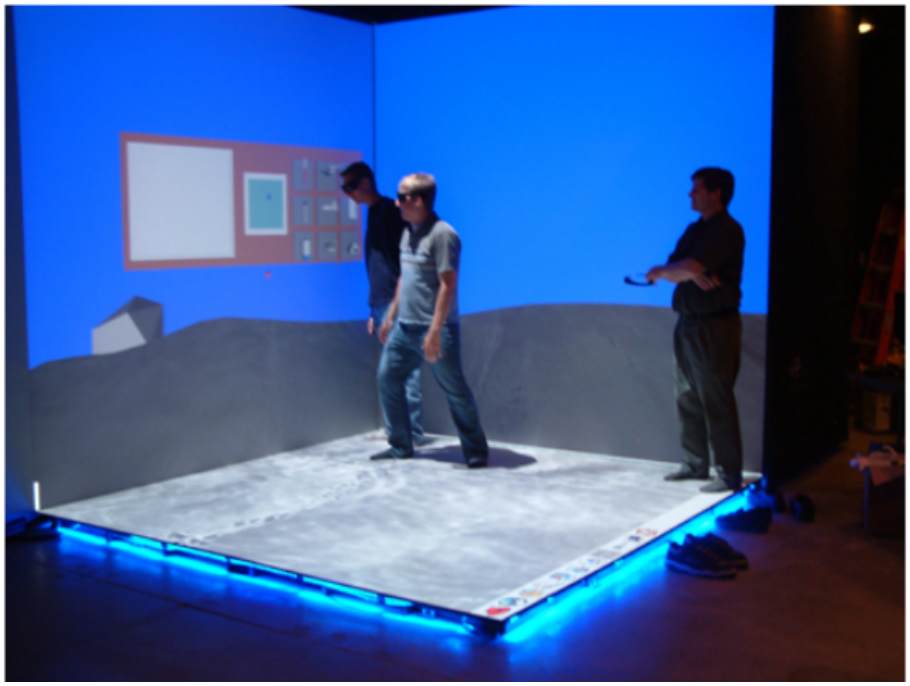

Figure 2.17: A Three-Sided Projector System [25].

expensive than large projection displays, and they usually have a better field of regard (FOR). The FOR is the percentage of all possible directions in which the user can orient his or her head without looking away from the VR display. The three-sided projection display in Figure 2.17 would be expected to have an FOR of fifty percent because the area covered by the two side projector screens and the floor projector screen is equal to the area left vacant by the open ceiling and two remaining open sides. An HMD, on the other hand, would be expected to have an FOR of one-hundred percent.

Although the FOR of an HMD is usually better than that of a multi-sided projector display, the multi-sided projector display usually has a wider field of view (FOV). The field of view is the angle associated with the user's viewing cone. A wider FOV means that the user can see more of the surrounding scene at one time. In addition, projection displays do not cause the unpleasant disorientation effects sometimes associated with HMDs. Finally, projection displays have the advantage that more than

Figure 2.18: A Head Mounted Display [2, page 14].

one person can view the same display at the same time.

2.3.3 Virtual Reality Input Devices

To allow the user to interact with the virtual world, it is necessary to use input devices to let the VR system know where the user is and what the user is doing. Among the most commonly used input devices are those used for head tracking. For VR projection displays, it is necessary to know the location and orientation of the user's head. This allows the system to give the correct view of the scene at the correct angle of view. Head tracking can be done using stationary sensors that communicate with a tracking device worn on the user's head. The communication link may be radio, optical, or even acoustic [2]. Figure 2.19 shows a head tracking device incorporated into a pair of stereoscopic goggles. The goggles work in concert with the stereoscopic display by alternately shuttering the view for each eye at the instant when a video frame intended for the other eye is being displayed. In this way, each eye receives only its intended video stream, and the user is unaware of the shuttering because it is too rapid to be perceptible. Incorporating the tracking device into the pair of goggles allows them to perform two tasks at once.

Another commonly used input device is the wand. The location and orientation of the wand can be tracked just like the goggles, and the wand can be used as a 3D mouse pointer, with buttons to initiate actions as would be found on a conventional mouse. Figure 2.20 shows a wand used for input in a projection display VR system.

A multi-sided projection system along with its associated tracking and other input devices together form a system known as a CAVE (Cave Automatic Virtual Environment). CAVEs are often described in terms of the number of projector screens they possess. For example, a four-sided CAVE would compose two-thirds of a complete cube and would have an FOR of about sixty-seven percent. Similarly, a six-sided CAVE would form a complete cube and would have an FOR of one-hundred percent.

Figure 2.19: A Pair of Head Tracking, Active Stereo Goggles [17].

Figure 2.20: A Virtual Reality Wand [17].

2.4 VFIRE

VFIRE is an application that uses virtual reality technology to allow a user to be immersed in an artificial environment where real or fictitious wildfire scenarios can be visualized. Unless otherwise noted, the material from this section is based upon the material in [25]. The purpose of these simulations is to provide useful information about fire spread in a particular region under various wind, humidity, and other environmental conditions.

Given that wildfires occur suddenly, without warning, and that the ways in which they spread are difficult to accurately predict, it is not surprising that a great deal of money and effort have been spent to develop mathematical models of wildfire spread patterns. However, only minimal resources have been applied to the purpose of visualizing simulations of these fire models.

Determining the accuracy of existing wildfire spread models is difficult, short of actually starting a wildfire to see if it burns as predicted by the model. Wildfire visualizations can be very useful for this purpose by comparing the computer visualization to video recordings of an actual fire. In addition, the visualizations can be used as a training tool for firefighters. Figure 2.21 shows a VFIRE visualization in progress.

VFIRE allows the user to travel throughout the entire area, either at ground level or flying above, observing the fire as it progresses. VFIRE also allows the user to go backward or forward in time, reversing or advancing the progress of the fire accordingly. Figure 2.22 shows a user interacting with the fire visualization.

VFIRE uses a virtual reality toolkit called FreeVR, created by William R. Sherman [48]. A virtual reality toolkit is essential in VR applications because it allows the computer to communicate with specialized VR hardware, not supported by most operating systems or graphics libraries.

Presently, VFIRE depicts wildfires using fire-spread simulation output data created by FARSITE. FARSITE is a fire behavior and growth simulator application

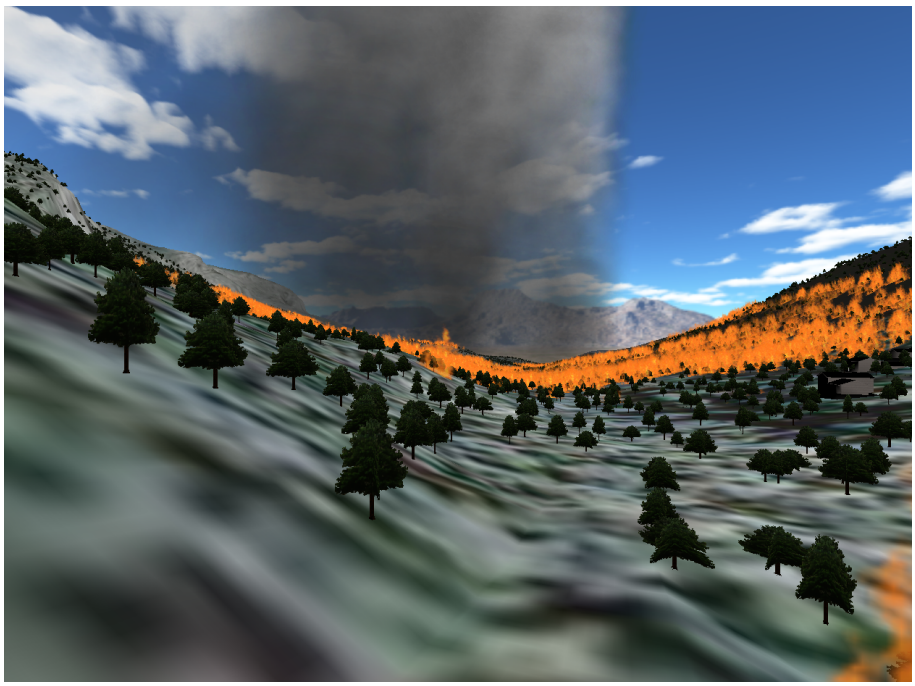

Figure 2.21: Fire Visualization in Progress [25].

Figure 2.22: User Interacting with VFIRE Visualization [25].

designed to be used by wildland fire planners and managers [10]. The FARSITE simulation output data contains six parameters of concern to VFIRE. However, the primary parameter is the time of arrival (TOA). The TOA data is a grid of numbers, each representing the time of day when the associated region (cell) of the terrain caught fire in the simulation. When the visualization time reaches or exceeds the time of arrival for any particular cell, VFIRE shows that cell catching fire. VFIRE shows the fire, the smoke, and eventually, the residual burnt vegetation.

Currently, VFIRE selects the location of each tree in the visualization randomly. Upon completion of this project, the arrangement of trees should more accurately reflect their positions in the real environment.

2.5 Related Work

It is often desirable to have knowledge of the locations, types, and sizes of trees in a particular area. For example, plantation managers may need to have accurate counts of their inventories in order to plan for their harvest activities. Forestry officials often require information about the ages, densities and types of trees in forested areas in order to assess the health of the forests and identify ongoing trends that may or may not be desirable. Similarly, it may be useful to know how much wood is available for harvesting in a particular area and, potentially, how much is available for combustion in the event of a fire. For these reasons, significant effort has been made to find fast and inexpensive ways of obtaining such data.

LiDAR data are not available in this project, and an examination of the techniques applied to LiDAR show relatively little applicability. In one instance, an effort was made to distinguish individual trees within a set of LiDAR data by clustering the point cloud in all three spatial dimensions [3]. In any area where the space between trees is sufficiently large, this algorithm could be expected to accurately locate individual trees. A similar approach could conceivably be applied to the photographic images available in this project by treating the brightness values as if they were height values in the LiDAR data. Using this approach, however, would likely result in a high

rate of false positive errors because brightness clusters may be produced by a variety of surface features besides trees, and without the height information contained in the LiDAR data, these false clusters would be indistinguishable from legitimate tree clusters. Similar limitations could be expected when trying to apply any LiDAR technique directly to photographic data.

Most of the algorithms that determine the locations of trees in photographic imagery do so by locating local maxima of the pixel values in the image [33]. A method of doing this was shown in [6], where a photographic image of a forested area is searched vertically, horizontally, and along both diagonals to find local maxima occurring in any of eight directions. One of the difficulties that occurs in using this method is that a single tree can produce more than one local maximum, resulting in two or more trees being detected where only one tree actually exists.

Various approaches have been developed to prevent such multiple detection. One such method is to apply a scale-space filter to the entire image and then extract the individual tree locations using a sophisticated thresholding technique [24]. In another approach, a scale-space filter is applied to the entire image and the individual tree locations are extracted by applying an edge-detection filter to reveal the curved edges enclosing the tree crowns. Analysis of the curvature of those edges produces the locations of likely trees [11]. In yet another method, a window is moved over the image in a scanning pattern and, at every location, the maximum pixel value inside the window is taken to be the local maximum for that region [14, 33].

The methods described in [11], [14], and [33], all benefit from the use of multispectral imagery. The different color bands in a multispectral image can be mixed in specific ways to allow certain types of material to appear brighter than others, allowing the image to be optimized for use in a particular algorithm.

In addition, all the methods described above use image data under constraints that can be exploited by the algorithm. For example, [24] uses aerial images of "even-aged homogeneous stands of Norway spruce." The other methods cited above also benefit from a relatively homogeneous scene in which there is scarcely anything to

be detected except the desired trees. Such visual similarity among the items to be detected greatly improves accuracy, and the methods cited above work well under those constraints.

In this project, however, the algorithm must be able to locate trees in images where no such constraints are assumed to exist and where multispectral imagery may not be available. The research done in [23] comes closer to this goal through the use of template matching. In this instance, a set of tree templates was created from a ray-traced, three-dimensional model based on an image of a Norway spruce forest. The image was filtered with the template, producing a correlation image that could be scanned for local maxima. This method produces good results as long as most of the trees in the image match the template reasonably well. This project also uses template matching, but the templates can be produced more quickly and easily, meaning that a template can readily be created for each group of similar-looking trees in an image. As a result, this project can be used as a stand-alone program to locate trees in a wide variety of images containing multiple groups of groupwise-uniform trees. The only preparation needed is to place the vegetation map data into the program in cases where a vegetation map is necessary.

There does exist image analysis software that can perform template matching [4]. However, these software packages would not be expected to list tree locations in geo-referenced coordinates nor assist the user by integrating vegetation map information into the image display. Finally, this project has the ability to make tree placements based on vegetation map data alone in cases where photographic imagery is not available.

Chapter 3

Placing Items in a Virtual Reality Visualization

The goal of this project is to place items, particularly trees, at locations within the VFIRE artificial environment corresponding to their true locations in the actual environment. Meeting this goal requires that the actual locations of most of the trees be known. In addition, trees, correctly placed, should have appearances similar to their appearances in the real environment, meaning that the heights, widths, and types of most of the trees must also be known. A further goal of this project is to place artificial structures such as houses and buildings in their correct positions within the VFIRE environment.

These goals are limited by the type and degree of detail of the available geographic image data. The available data consist of satellite images and LANDFIRE vegetation maps. The satellite imagery includes a panchromatic image at 1-meter resolution. It also includes red, green, blue, and near infrared images each at 4-meter resolution. There are several vegetation maps available, some being of minimal usefulness and some being of significant usefulness for the purposes of this project.

It is not assumed that the available data are necessarily adequate to fully meet the ultimate goals of the project, stated above. The 4-meter resolution images do not appear to have sufficient detail to distinguish trees. The 1-meter resolution image does appear to have marginally sufficient detail to distinguish trees. However, groups of trees in different regions of the image have different characteristic appearances, and

there exist many non-tree surface features in the image that can easily be mistaken for trees by a tree detection algorithm. For these reasons, this project uses an interactive process for tree detection, allowing the user to assist the computer in overcoming some of the limitations in the image data.

This project detects trees using a template matching algorithm, which allows the user to quickly adapt this project to the way trees look in any given image. This is possible because the algorithm is not tailored to a particular image. The user-defined template is tailored to the image, and the algorithm is tailored to the correlation pattern between the image and the template. As long as the trees in an image can be grouped into collections of groupwise-similar appearance, the user can select one representative tree from each group to serve as the template for that group. The system will then search for all items that look similar to the templates.

Although the trees in the 1m-resolution image are generally identifiable as trees, the tree types are not easily identifiable without prior knowledge of the types of vegetation in the area. Therefore, vegetation maps are used to supplement the vegetation type information not conveyed by the image itself. If the user clicks on the image, the system will display the information in the vegetation maps pertaining to the location where the mouse click occurred. In addition, the user can choose to have the photographic image of the geographic area highlighted using the colors of the vegetation map. This allows the user to see the dominant vegetation patterns along with the photo of the area at the same time.

Photographic images have already been obtained for the current area of interest, Kyle Canyon in Southern Nevada. However, it is possible that the project may be used in the future in an area for which photographic imagery will not be obtained. Given the low cost of vegetation maps, it is likely that they will be available even when no other data are available. Therefore, the system is also able to place trees based on vegetation map data alone. Tree placements, in this case, are made by placing trees randomly within each vegetation map cell according to the vegetation coverage, vegetation type, and tree height specified in the respective maps. The tree place-

ments produced in this way are less accurate than those produced from photographic imagery, but they are more accurate than completely random placements.

This project does not currently allow for the automatic detection of artificial structures. The template matching algorithm used for detecting trees does not work well for detecting houses and buildings in the area of interest because their appearances exhibit a high degree of diversity, preventing the creation of widely applicable templates. Furthermore, the number of such structures in the area is small enough to allow the user to efficiently place each one of them manually. Therefore, the system allows artificial structures (or trees, if desired) to be manually placed anywhere in the scene. It also allows items of any type to be manually deleted from the scene.

Chapter 4

Software Specification and Design

In this chapter, the features of this project are presented as a list of functional and nonfunctional requirements, as well as a diagram showing the use cases that are expected to occur in the process of tree detection. An explanation of the requirements and use cases is provided to clarify the process of using this project.

This chapter also presents the structure of this project both in terms of the classes used and in terms of the subsystems from which it is composed. The purpose of each class and each subsystem is explained, and information is provided regarding the ways in which each subsystem accomplishes its task.

4.1 Requirements

The purpose of this project is to find the locations of trees in an image. Given that there are no constraints on the type, resolution, or geographic area of the image, the system relies heavily on the judgment of the user to overcome tree-detection difficulties that cannot be anticipated or resolved within the algorithm. For this reason, this project uses an interactive detection procedure in which the user must be able to see what the system is doing in order to effectively guide the process. A significant portion of the program is devoted to the task of letting the user see the current status of the operation and what needs to be done next. In addition, a large portion of the program is devoted to allowing the user to control the way that tree detection occurs in order to minimize errors. Table 4.1 shows the functional

requirements.

The nonfunctional requirements are shown in Table 4.2. These features reflect some of the goals and constraints of the project, such as the use of C++ for optimum speed and the direct implementation of graphics display functions to maintain total control over memory consumption, which is of concern when viewing very large images.

4.2 Use Cases

The operations commonly used in the tree-detection process are shown in Figure 4.1. The use cases show that the user is given the flexibility to decide at runtime which geographic images to analyze, which combination of images to view at any moment, and which image to use as the source for any tree placements that are made. The user creates a template by drawing a highlighting mark over a particular tree. Each template is given a name, as well as a nominal height and width. These data are eventually placed in the output file for every tree detected using that template. The user can determine the approximate height and width either by visual inspection of the photographic image or by clicking on the image at the location of the template to view data from the vegetation maps. Multiple templates can be created for a single project, and the user can click on any existing template to select it as the active template. Subsequently, the user can filter the image using the active template to produce a correlation image. The system then scans the correlation image for bright spots, corresponding to likely trees. Finally, the system places a mark at the location of each tree detected. The user may then choose to have the current set of tree marks placed in an output file and stop the process or make changes to improve the accuracy and repeat the process.

F01	The utility shall allow the user to load a photographic image of a geographic area.
F02	The utility shall allow the user to display the photographic image.
F03	The utility shall allow the user to zoom and scroll the image.
F04	The utility shall load vegetation cover, vegetation type, and vegetation height maps if available.
F05	The utility shall display vegetation map information for user-specified locations.
F06	The utility shall allow the user to display one of the vegetation maps.
F07	The utility shall allow the user to display one of the vegetation maps on top of the image.
F08	The utility shall allow the user to select image regions to use as tree templates.
F09	The utility shall allow the user to select one template as the active template.
F10	The utility shall allow the user to edit the active template.
F11	The utility shall allow the user to associate a name, type, hight, and width for each template.
F12	The utility shall allow the user to place templates into groups.
F13	The utility shall allow the user to adjust tuning parameters for tree detection.
F14	The utility shall allow the user to filter the image to produce a correlation image.
F15	The utility shall allow the user to display the correlation image.
F16	The utility shall search the correlation image for trees.
F17	The utility shall search only within the region currently in view.
F18	The utility shall place a mark at each location where a tree is detected.
F19	The utility shall allow the user to place a tree mark in any location
F20	The utility shall allow the user to mark any location as the location of an artificial structure.
F21	The utility shall allow the user to delete any mark from any location.
F22	The utility shall allow the user to save all data in a project folder.
F23	The utility shall allow the user to load a project from an existing project folder.
F24	The utility shall allow the user to output all item locations to a file usable by VFIRE.
F25	The utility shall allow the user to create rough tree placements based only on vegetation maps.
F26	The utility shall mark tree placements made based on veg map data.
F27	The utility shall allow the user to output map-based tree placements to a file usable by VFIRE.

Table 4.1: Functional Requirements.

NF01	The utility shall directly implement graphics to create and manage templates.
NF02	The utility shall directly implement graphics to display, scroll, and zoom image.
NF03	The utility shall use bilinear interpolation to zoom into and out of the image.
NF04	The utility shall swap correlation images to and from disk to match currently active template.
NF05	The utility shall use the brightness component of the image for tree detection.
NF06	The utility shall perform contrast stretching on the correlation image.
NF07	The utility shall be implemented on the Linux platform.
NF08	The utility shall be written in C++.
NF09	The utility shall use GDAL to read image files.
NF10	The utility shall use FLTK for its GUI.

Table 4.2: Nonfunctional Requirements.

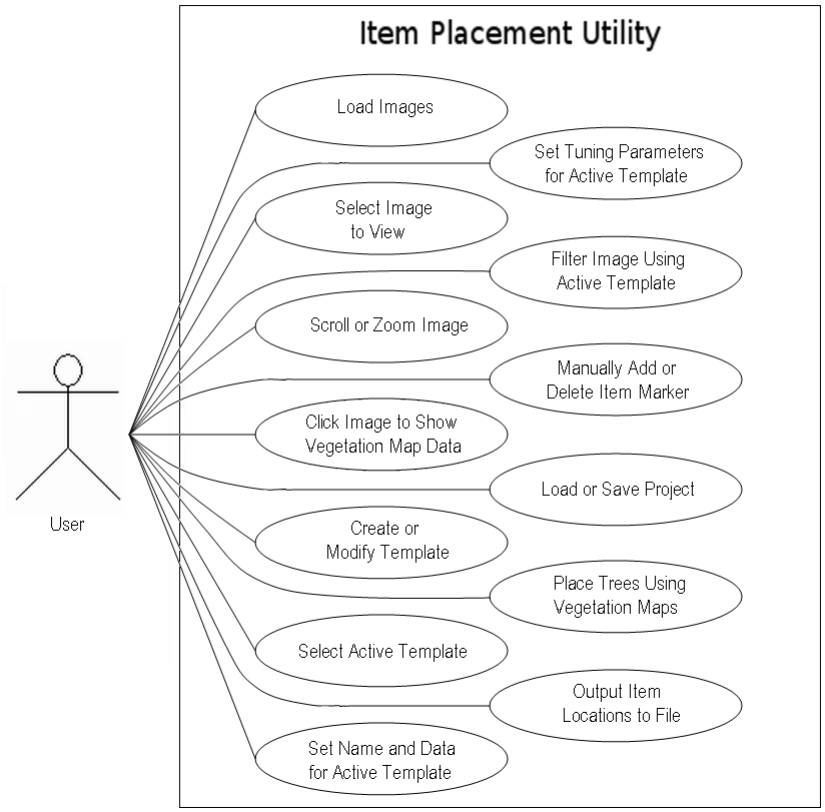

Figure 4.1: Use Cases.

4.3 Classes

Although the goal of this project is complex from an image analysis perspective, most of the data types used are fairly simple arrays of numbers. Therefore, only six classes are needed, and the classes are completely independent with no inheritance. Figure 4.2 shows the classes used in this project.

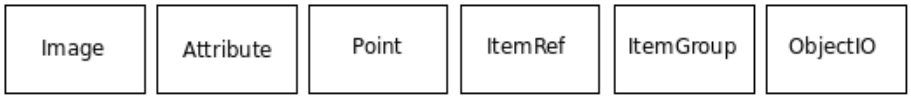

Figure 4.2: Classes Used by Utility.

An instance of the Image class is used to store the main image used in a tree-detection project. This is generally expected to be a photographic image of the area of interest, though other types of images can be used as long as they possess the same metadata. In addition, the system also uses three other instances of the Image class. The first of these is used to allow viewing of a vegetation map (if any), the second of these is used to allow viewing of the vegetation map simultaneously with the photographic image, and the third of these is used as a work space to hold temporary image data. Image class objects contain the raster data for the image as well as the georeferencing data that associates each location in the image with its corresponding location on the Earth's surface. The Image class also contains the information to let the display system know how to display the image, such as the current scroll position, zoom level, and whether the display buffer needs to be refreshed.

Instances of the Attribute class are used to store the vegetation maps for the system. Each vegetation map is stored as a raster of vegetation codes. The meaning of each code is stored in data tables contained in each instance of the class. For each possible vegetation code, the data tables give the associated display color. In addition, data tables also translate vegetation codes into information such as dominant vegetation type, percentage of vegetation cover, or vegetation height, depending on which vegetation map file the Attribute instance is being used to store. The display

system is not designed to use objects of the Attribute class. Therefore, vegetation maps are not displayed directly. Instead, the pattern of display colors associated with the raster of vegetation codes is scaled and copied to an object of the Image class, which is then displayed. The vegetation map can be copied onto a blank Image object or it be can mixed with photographic data already present in the Image object to produce an overlay of the vegetation map over the photographic image.

The Point class is used to keep track of individual pixels in an image. Each instance of the Point class stores the horizontal and vertical location of a single pixel, as well as its three color components. Multiple instances of the Point class are used to form templates.

Each instance of the ItemRef class is used to store an image template. ItemRef is a container class for objects of the Point class. Such collections of Point objects constitute the image templates that are used as filter masks to produce correlation images which are then scanned for bright spots corresponding to tree locations. ItemRef also contains the tuning parameters that govern which bright spots in the correlation image qualify as trees. These parameters determine how bright the spot must be, how wide it must be, and how much brighter it must be than the surrounding image. ItemRef also stores information about the area covered by each template. When the user chooses to filter an image, only the portion of the image currently in view is filtered, allowing the user to quickly test the accuracy of a newly created or newly altered template without waiting for the entire image to be processed. Finally, ItemRef stores a name, nominal height, and nominal width to be associated with any tree detected using that particular template.

ItemGroup is a container class for objects of the ItemRef class. The purpose of ItemGroup is to allow templates to be placed into separate groups, of which only one group is visible at a time. This allows the user to decide which templates and corresponding tree marks will be visible simultaneously with others and which ones will not. If the user wants a particular template and its associated tree marks to be seen only by itself, without any others on the screen at the same time, then this

template should be placed into its own separate instance of ItemGroup.

ObjectIO is the class used to read and write the binary tree-location files used by VFIRE. Instances of the ObjectIO class are used to write the geographic coordinates of each tree along with its height, width, and type. All the information is written in binary form to save disk space. ObjectIO can also be used to read these data from VFIRE.

4.4 Program Subsystems

This project consists largely of global functions rather than class methods. The functions that display images could have been included with the Image class, but the current configuration is very intuitive. A large collection of global functions, together, constitute the graphics display subsystem, and the graphics display subsystem accepts data from a small, fixed number of Image objects. The process of displaying images is straightforward and predictable in terms of which operations will be performed on which class objects. Figure 4.3 shows the arrangement of subsystems.

The File I/O subsystem consists of several functions that read and write the files used by the system. Vegetation map files and GeoTIFF image files are read by the system but never written. Tree-location files are written by the system but never read. Project data files and correlation image files are read and written by the system. GeoTIFF files are usually photographic images, and the File I/O subsystem relies on GDAL to read these files. GDAL has the ability to read and write other types of files, but this project currently uses it only for GeoTIFF files.

The Template subsystem is a set of functions that allow the user to create, edit, and manage the templates used for tree detection. The user creates templates by using the mouse pointer to draw highlighting marks on the desired portion of the image. The Template subsystem keeps track of which pixels have been highlighted, which template the pixel belongs to, and which group the template belongs to. As shown in Figure 4.3, the template subsystem relies on the graphics subsystem to draw the marks. The template subsystem also keeps the data in the active template current

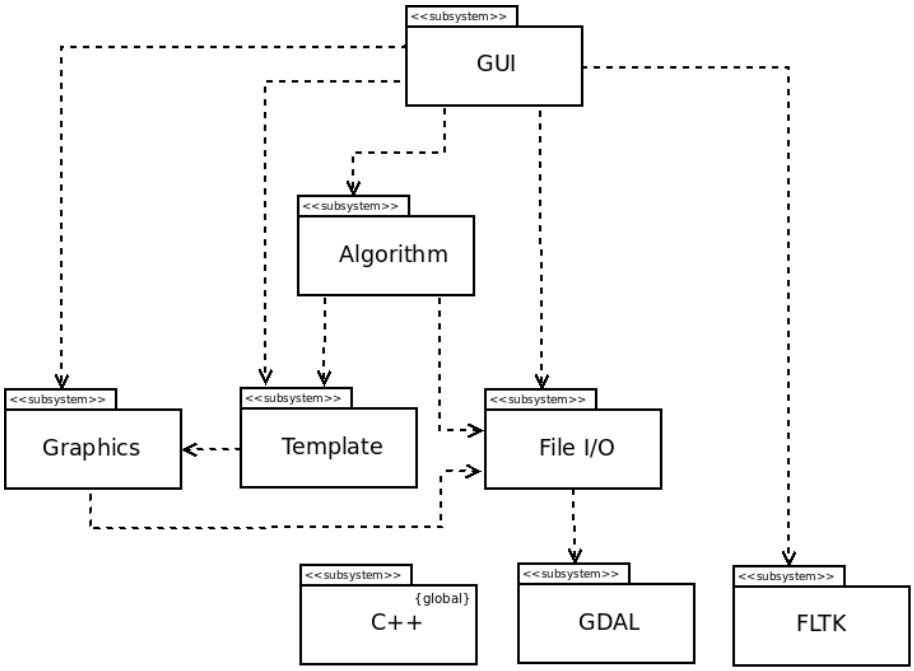

Figure 4.3: System Structure

as the zoom level or other view conditions change.

The Graphics subsystem includes all the functions that display images on screen. The Graphics subsystem can only process objects of the Image class, and to avoid running out of memory, this project uses only four instances of the this class. The first instance stores the original image. The second stores the original image mixed with the color pattern of one of the vegetation maps. The third stores the color pattern of the vegetation map by itself, and the third is a work space used for temporary storage of correlation images and to show tree placements made using vegetation map data alone. The graphics subsystem is always set to display one of these Image objects. The object being displayed at any particular time depends on the current view settings selected by the user. The need to display highlighting marks for the templates adds complexity to the display subsystem, and the need to display, at times, very large images adds further complexity. However, the graphics subsystem always displays one of the four Image objects listed above.

The functions in the Algorithm subsystem use the available templates and user-

controlled tuning parameters to detect the locations of trees within the image. This is done by filtering the image with the active template to produce a correlation image in which the brightness at any location varies directly as the similarity between the pixels in the template and the pixels in the corresponding neighborhood in the image. There are many ways to calculate the correlation values, but in this implementation, each pixel in the correlation image is calculated by taking the average difference between the two sets of pixels and subtracting this value from 255. Once the correlation image is produced, it is scanned for bright spots and they are marked as likely trees according to the current settings of the tuning parameters

Chapter 5

Implementation and Results

In this chapter, the software implementation of this project is presented. Examples are provided, demonstrating the use of the currently implemented features, and an explanation is given of the major design and implementation issues. Possible methods to optimize tree-detection accuracy are considered, and several examples of tree-detection results are provided. An exact, quantitative analysis of tree-detection accuracy is not provided because such results vary greatly among images and may vary significantly even within a single image. Instead of numeric detection results, screen shots are provided, showing the detection results obtained for portions of the panchromatic image of Kyle Canyon, Nevada. The screen shots are categorized to exemplify the good, adequate, and poor accuracy regions of the image. Good results were obtained in about fifteen percent of the image, adequate results were obtained in about seventy percent of the image, and poor results were obtained in the remaining fifteen percent. These screen shots are intended to indicate the level of accuracy that can be expected for images of similar resolution depicting scenes similarly suited for tree detection.

5.1 Detecting Trees

After loading a photographic image of the area of interest, the user identifies a group of similar-looking trees and selects one of them to serve as the template for the rest. The user then uses the mouse to draw a highlighting mark over that particular tree,

covering all pixels intended for inclusion in the template. The tree to be selected as the template is shown in Figure 5.1. The image is zoomed in so that the individual tree can be easily seen. The tree, itself, is the light-colored blob in the center of the image. The shadow of the tree is also visible. It is the dark, elongated shape pointing upward and to the left from the tree. It is often helpful to include, in the template, portions of the ground surrounding the tree or, as shown in Figure 5.2, the shadow of the tree.

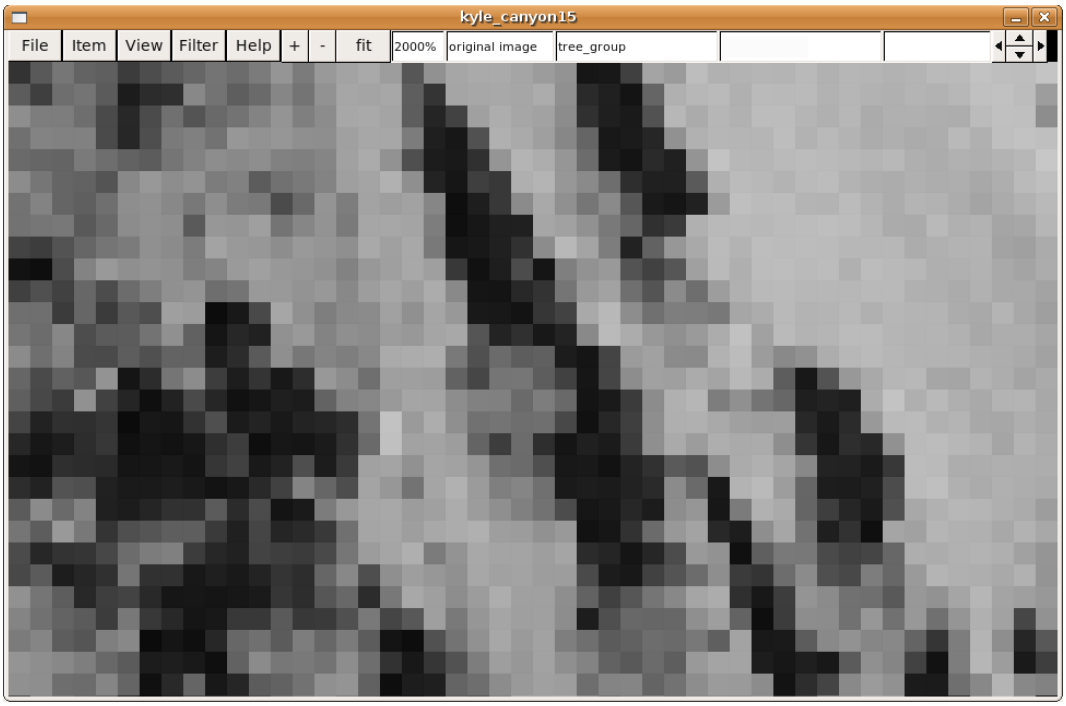

Figure 5.1: Enlarged View of Tree Centered in Window.

Figure 5.3 shows the image zoomed out to allow more of the area to be seen. The template can still be seen, highlighted in green. It is intended that the system will search for similar-looking trees and mark their locations.

Once the template has been defined, the user activates the filtering process. The result of the filtering process is a correlation image, as shown in Figure 5.4. The image looks red because the correlation image is monochrome and the data for it is stored in the first available array in the Image object being used as a temporary

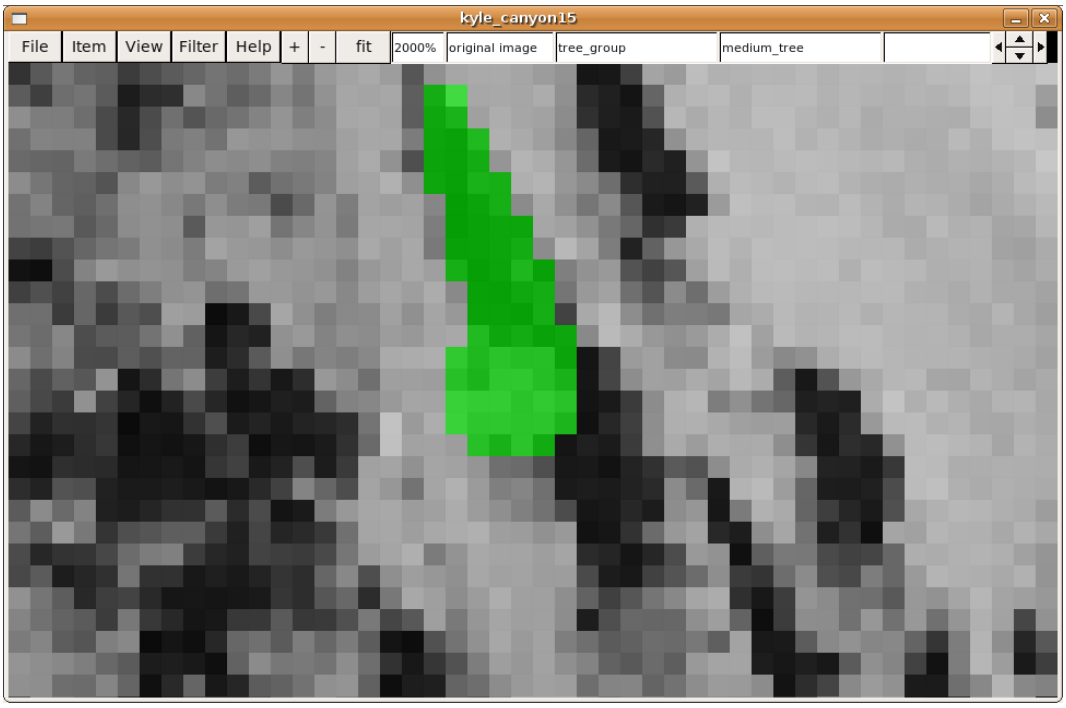

Figure 5.2: Enlarged View of Tree Highlighted by User.

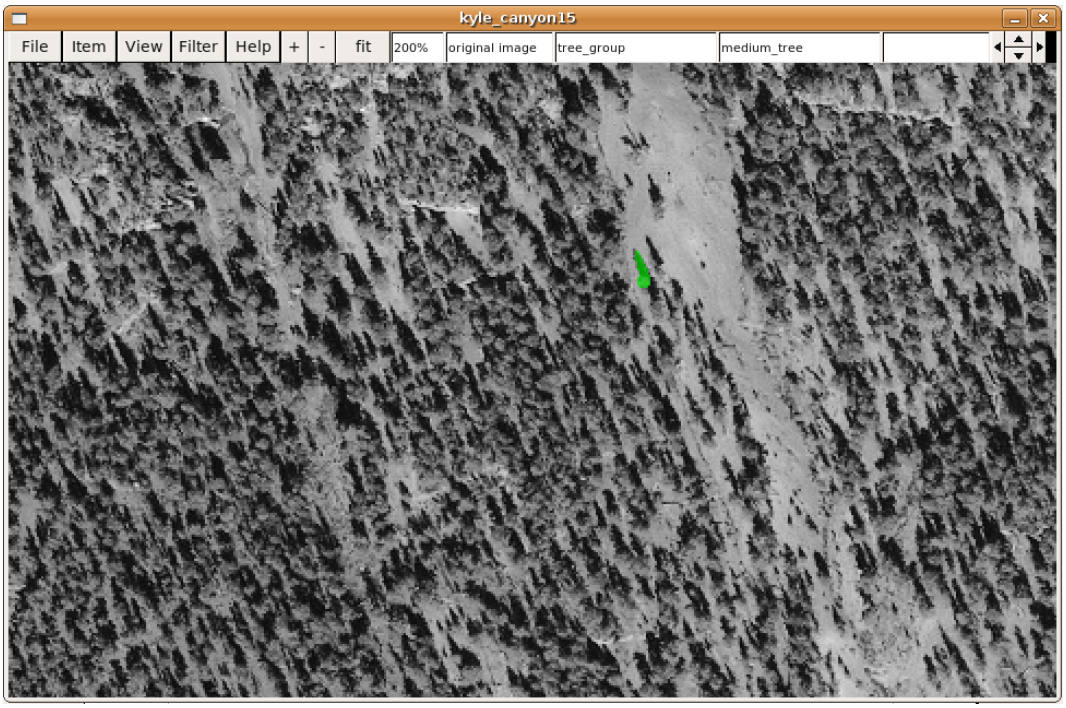

Figure 5.3: Tree Used As Template and Surrounding Area.

workspace. The second and third arrays (green and blue) are used for intermediate processing of the correlation image. The display function could have been modified to show only the red array of the workspace image, displaying it in grayscale. However, there are cases where it is desirable to see the other arrays as well, to verify that the processing algorithms are working properly. The bright spots in the correlation image of Figure 5.4 correspond to locations in the original image that look similar to the template tree. Filtering the original image to produce the correlation image is generally the most time consuming step in the tree-detection process. For this reason, the system filters only the portion of the image in view at the time of filtering. This way, the user can test the effectiveness of a template without waiting for the entire image to be filtered, and it can be determined quickly whether the current template is adequate. When it comes time to filter the entire image, the user clicks the "fit" button, which will fit the entire image into the window. With the entire image in view, any subsequent filter command will be applied to the entire image.

After creating the correlation image, the system begins searching for bright spots. It places tree markers in the locations of certain bright spots according to a set of user-adjusted tuning parameters. Figure 5.5 shows the placement of tree markers in the portion of the original image near the template.

The tuning parameters tell the system which bright spots qualify as trees and which ones do not. The controls are shown in Figure 5.6. Bright spots are initially found by identifying local maxima. Each pixel in the correlation image is surrounded by eight other pixels. If the center pixel is brighter than six of the surrounding pixels, then it is a peak in six directions, and if it is dimmer than two of the surrounding pixels, then it is a dip in two directions. Using the parameters of Figure 5.6, such a pixel would meet the first two criteria for inclusion as a likely tree. However, if it had more than two dip directions or fewer than six peak directions, it would fail. Since each pixel in the correlation image is a linear combination of other values, the word *moment* is used to refer to the pixel value itself. Figure 5.6 shows that each local maximum pixel must have a value of at least 190 on a scale of 0 to 253. The

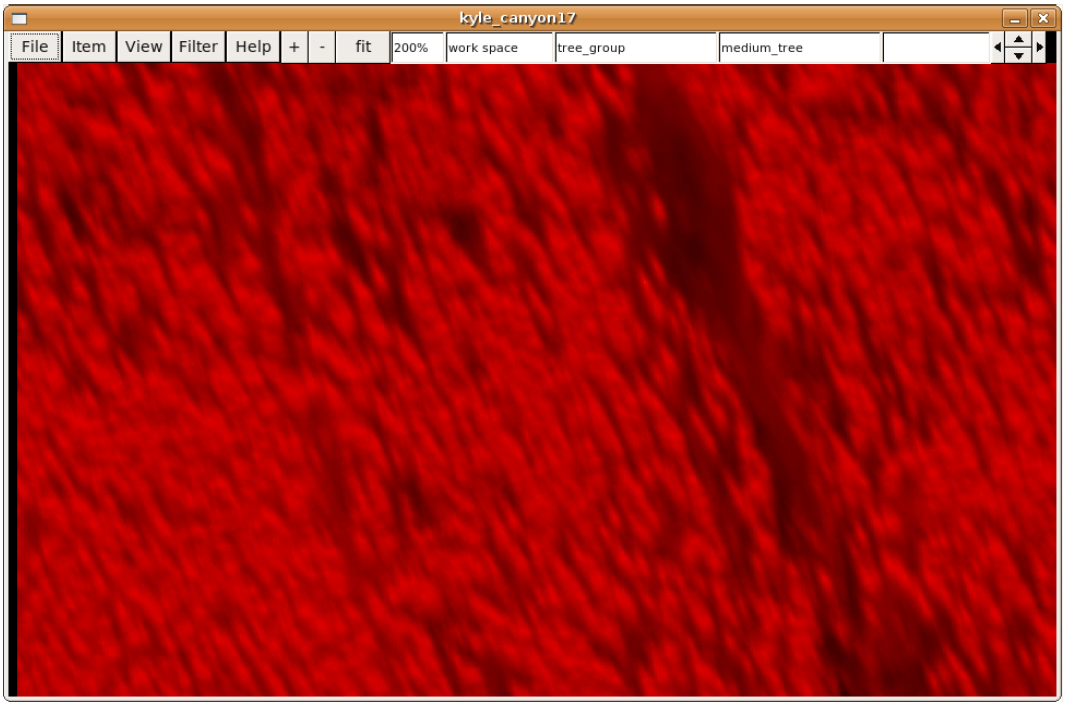

Figure 5.4: Monochrome Correlation Image Contained in Red Buffer of Workspace Image.

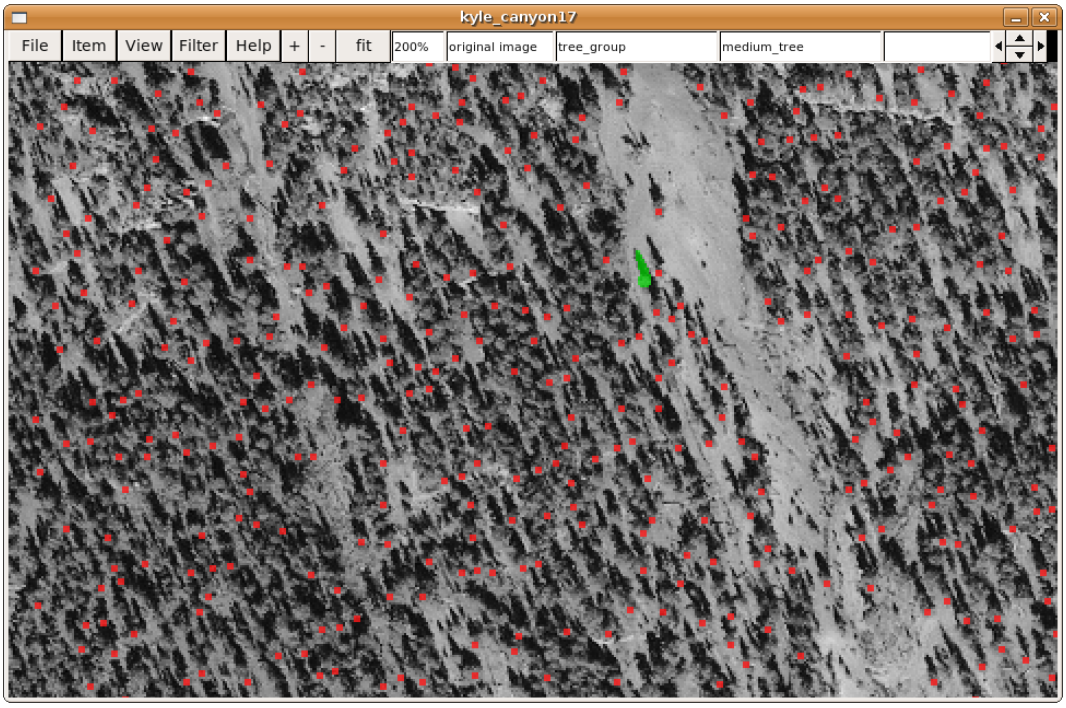

Figure 5.5: Likely Trees Marked After Processing.

maximum value for any pixel is 253 because the values 255 and 254 are reserved for manually placed or manually deleted items, respectively. Bright spots corresponding to valid trees are often surrounded by dark-colored perimeters. Figure 5.6 shows that a bright spot must be surrounded by a perimeter at least 18 units darker than the brightest pixel in order to qualify as a likely tree location. The distance between the perimeter and the brightest pixel usually falls within certain bounds. In the case shown in Figure 5.6, the perimeter must lie anywhere from one to thirteen pixels from the brightest pixel in the bright spot. In addition, the perimeter may be partial or it may completely enclose the bright spot. Estimating the extent of enclosure is done by scanning in eight directions from the brightest pixel. In the case of Figure 5.6, the perimeter must be detected in all eight directions for a given bright spot to qualify as a likely tree. It is possible for one tree to produce a bright spot with more than one local maximum. Therefore, a minimum distance between tree markers is set by the user. Tree markers occurring closer than the minimum distance from any other marker will be culled from the image. Finally, the user may wish culling to occur on a per-template basis, in which case the tree markers for one template are culled without regard to their distances to the tree markers from other templates. On the other hand, the user may consider all the tree markers to refer to the same type of tree, in which case the tree markers from one template should not be allowed to exist too close to marks from another template. In the case of Figure 5.6, the markers must be at least four meters apart and the markers for the active template are culled without regard to the markers of other templates.

As mentioned above, the user has the ability to manually place markers. The great majority of markers are used to represent trees, but they can be used to represent artificial structures as well, depending on the associated item code. In addition, the user can manually delete any marker of any type. The locations of manually placed or deleted items are stored in the correlation image itself. When a user places an item manually, the value 255 is placed in that particular location in the correlation image. This value is interpreted as a valid tree or artificial structure, and the automatic

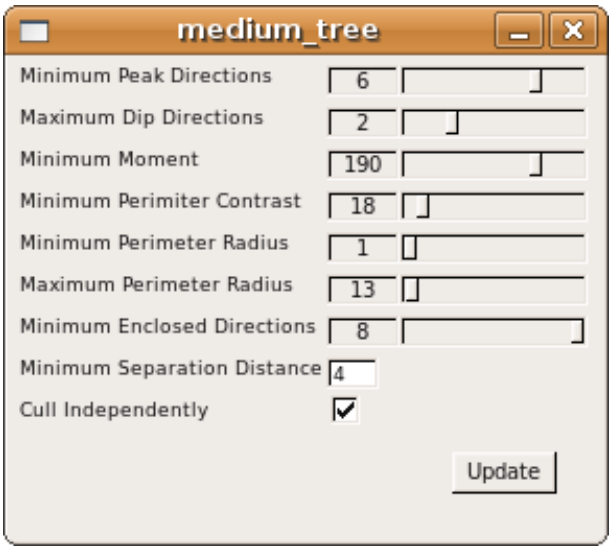

Figure 5.6: Tuning Parameter Controls.

system cannot override such placements even if its rules would not otherwise have placed an item in that location. Similarly, when the user manually deletes an item, the value 254 is placed in that particular location in the correlation image. This value is automatically interpreted to contain nothing, and the automatic system cannot override such deletions even if its rules would otherwise have placed an item in that location.

In the process of tree detection, large amounts of data may be produced. To allow the user to interrupt the process and resume it at a later time, the system allows the user to save the data in a project directory and reload it when needed. A project directory contains one project file, along with one correlation image file for each template in the project. The project file is a text file that contains the current values of the tuning parameters for each template as well as all other data associated with the templates. The correlation image files contain the raster data for each correlation image.

Once the tree-detection process is satisfactorily completed, the user outputs all tree and artificial structure locations to a binary file that can be used as input for the VFIRE application. To save space within the file and reduce the time it takes VFIRE

to read the file, tree types are expressed using an item code, as shown in Figure 5.7. The reference list of item codes is shown above the list of existing templates so that the user can place the correct code for each template before generating the placement file. The user also has the option of letting the tree type be selected by the vegetation type vegetation map. For example, if the user selects item code 14 in Figure 5.7, then, for each tree, code 14 will be replaced by whatever code matches the tree type in the vegetation map at the location of that particular tree. Each template is called a reference because the system refers to the data in the template to determine what to search for.

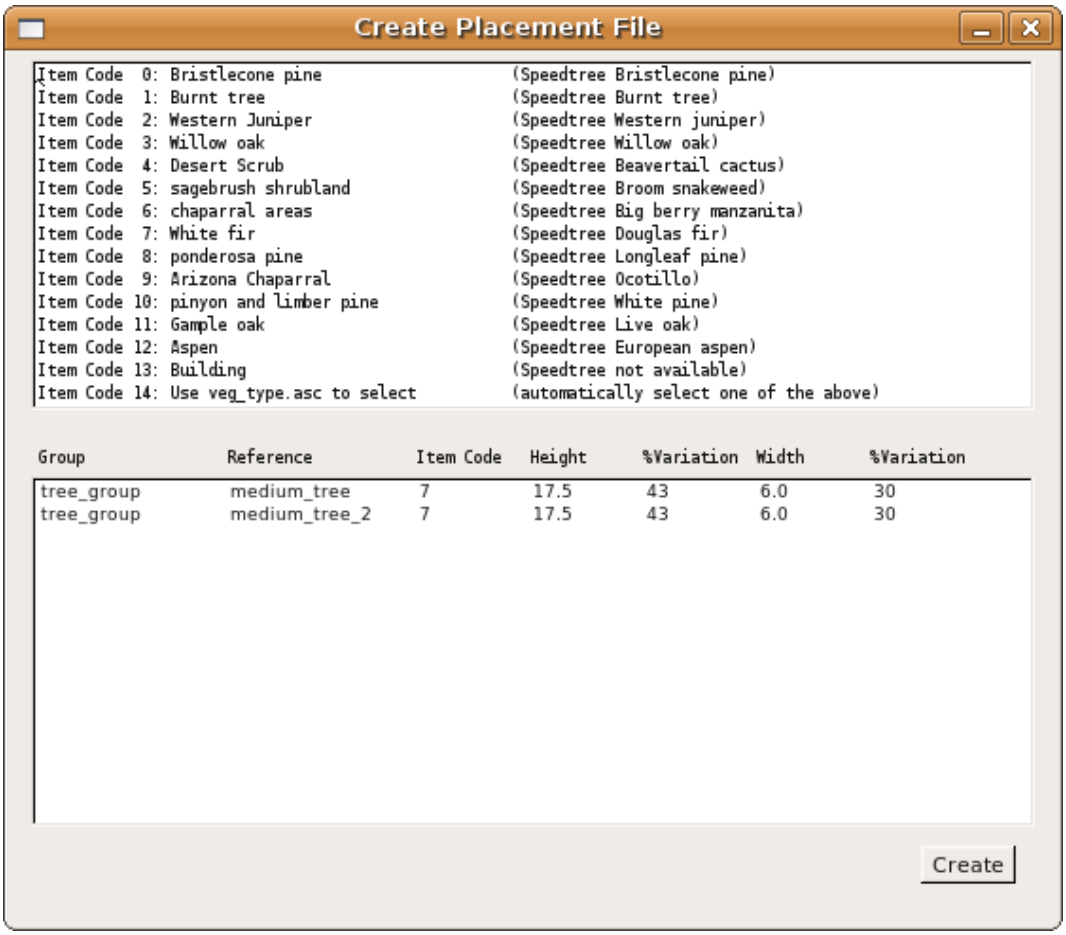

Figure 5.7: Placement File Information.

5.2 Using Vegetation Maps

In addition to the photographic imagery of the area of interest, the user can also view vegetation maps of the area. This project is currently able to handle three different vegetation maps. The first of these is the map showing vegetation cover. Each $5m \times 5m$ surface cell has a code that determines what percentage of the land in that cell is covered by vegetation. The second type of vegetation map is the vegetation type map. Each $5m \times 5m$ cell has a code specifying the dominant vegetation species within that cell. The third type of vegetation map is the vegetation height map. Each $5m \times 5m$ cell has a code specifying the range of tree heights within the cell. Each of these maps can be displayed by itself or as an overlay onto the photographic image of the area. Figure 5.8 shows the panchromatic image of Kyle Canyon, zoomed out far enough to show the entire image all at once. Figure 5.9 shows the vegetation type vegetation map for the same area, and Figure 5.10 shows the vegetation map as an overlay on the panchromatic image.

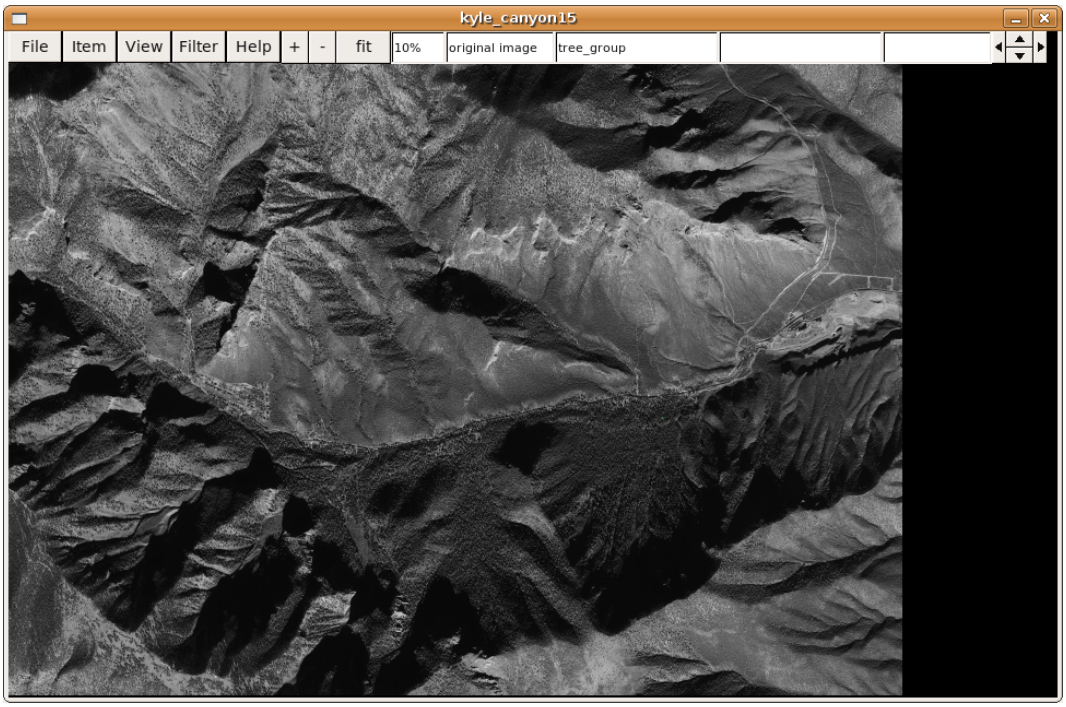

Figure 5.8: Photographic Image Alone.

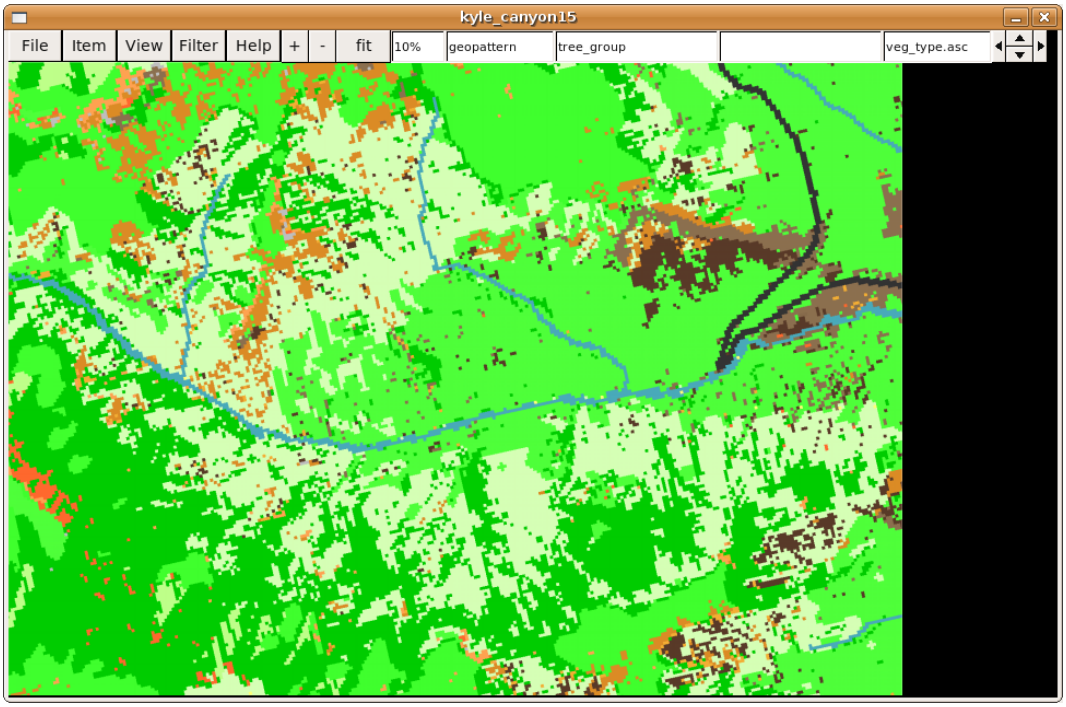

Figure 5.9: Vegetation Map Alone.

The image overlay is intended to help the user to determine tree types. Similar-looking trees found to exist on uniformly colored regions are probably the same type of tree. The tree type itself can be determined by clicking anywhere in the region. Figure 5.11 shows the text output that results from clicking on the image.

Vegetation maps can be used to make tree placements even when no photographic image is available. Figure 5.12 shows how this project can make tree placements based on the vegetation cover vegetation map.

The system draws a circle for every location where a tree is placed. This allows the user to see the results of the current settings and make adjustments if necessary. The density of tree placements in any cell is governed by the designated vegetation cover percentage of that cell. The vegetation type vegetation map may be used to determine tree types or the types may be generated randomly according to user-specified relative proportions as shown in Figure 5.13.

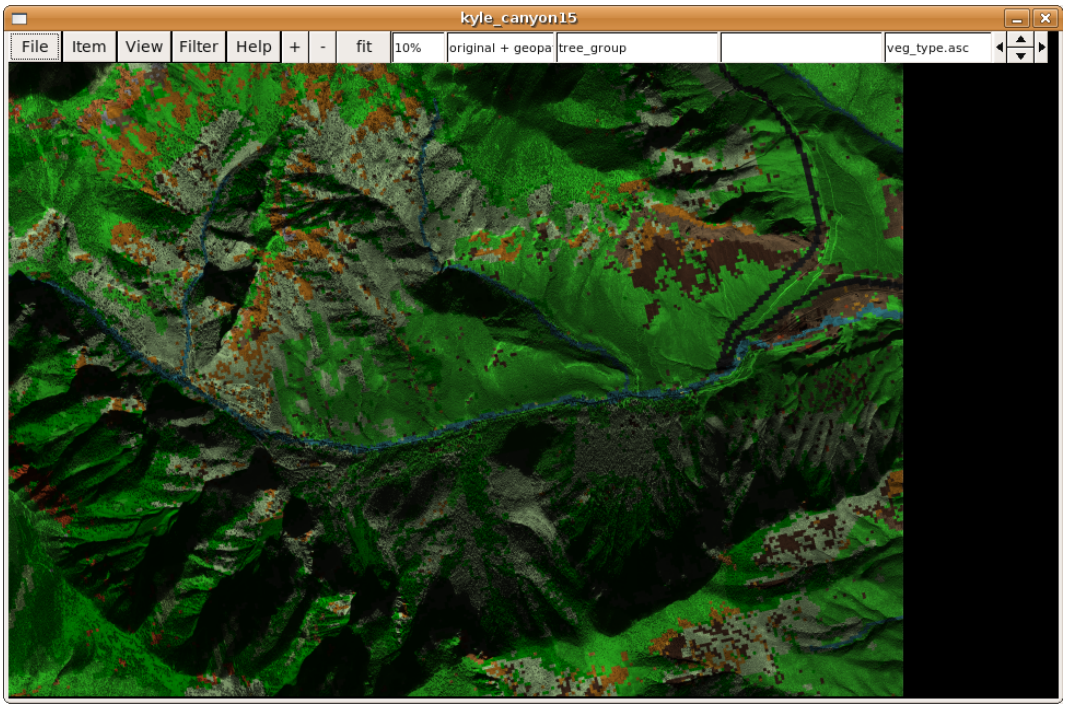

Figure 5.10: Photographic Image and Vegetation Map Together.

Figure 5.11: Terminal Text Output Produced When User Clicks on the Image.

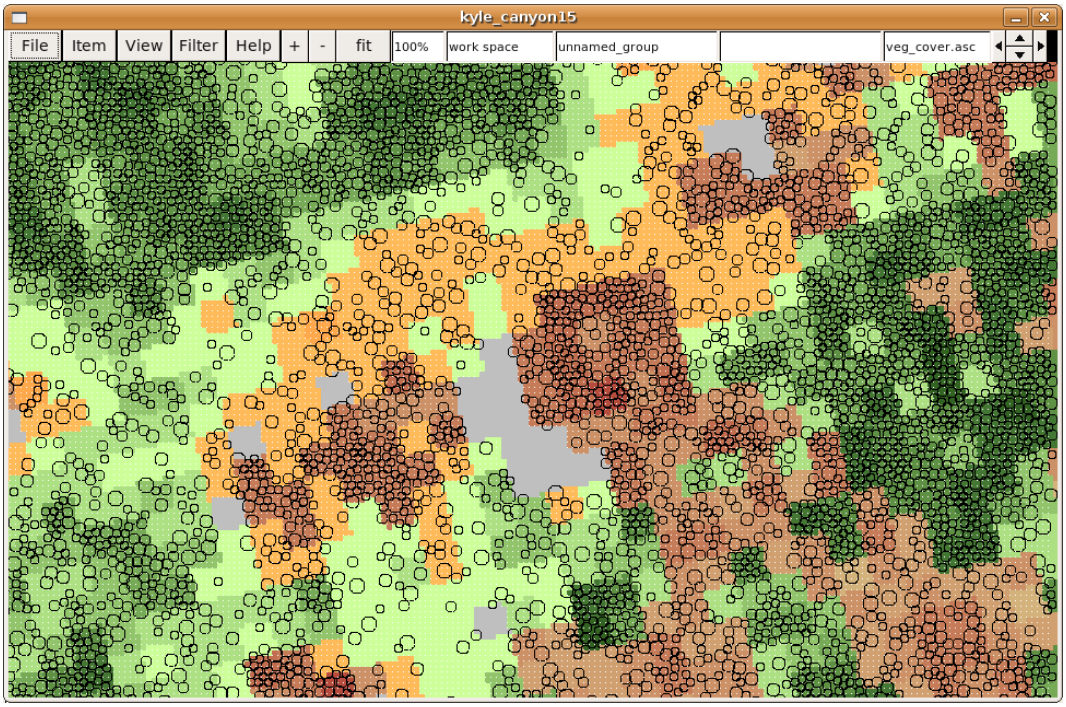

Figure 5.12: Tree Placements Made According to Vegetation Map Tree Densities.

5.3 Tree Detection Accuracy

When the system fails to mark the location of a tree, it is considered a false negative result, and when it marks a location where no tree actually exists, it is considered a false positive result. Ideally, the tuning parameters should be adjusted to balance each type of error so that both are kept below the maximum acceptable limit. In practice, however, it may not be possible to keep both types of errors within acceptable limits at the same time. A more generally applicable procedure, therefore, is to adjust the tuning parameters to put the number of false positives within acceptable limits and then create a new template to deal with the remaining false negatives. Figure 5.14 shows the same region depicted in Figure 5.5 after the inclusion of a second template. As shown in the image, adding the second template reduces the number of false negatives. The tree markers associated with the second template are shown in a slightly brighter red to indicated that they are associated with the currently active template. This image depicts the adequate, though not exceptional, level of accuracy

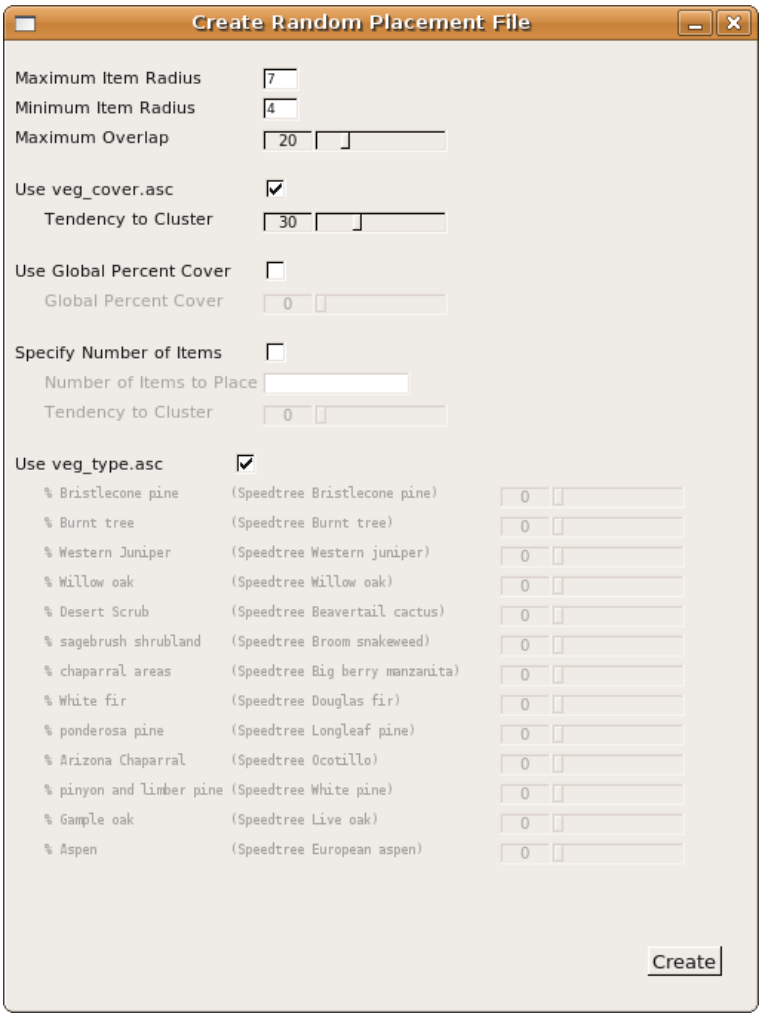

Figure 5.13: Random Placement File Information.

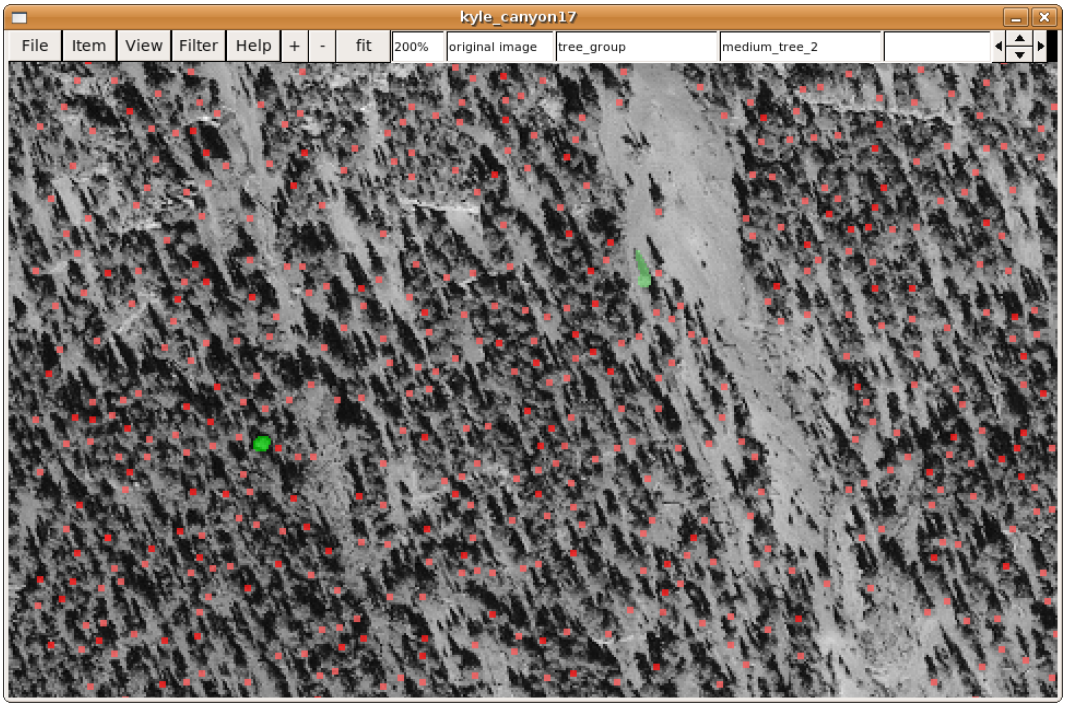

Figure 5.14: Likely Trees Marked After Processing with Two Templates.

found in about seventy percent of the image.

The second template takes advantage of the fact of being surrounded by shadow on all sides, unlike the first template, which has shadow only on two sides. The trees themselves both look like light-gray blobs. It is the nearby shadows that distinguish them from each other. The second template will more easily detect trees that are similarly surrounded by shadow. This is the reason why some portion of the area surrounding a tree is often included in the template. Including in the template some portion of the immediate area around a tree often produces darker perimeters around bright spots corresponding to trees in the correlation image. As the template approaches the edge of a tree during the filtering process, the shadow in the template overlaps the tree in the image and the shadow in the image overlaps the tree in the template. This situation produces a low correlation value, and it occurs mostly when the template is at the edge of a tree, resulting in a dark perimeter around the bright spot. As the template approaches the center of the tree, the similar regions tend to

coincide more, which produces the bright spot in the center. If the shadow or otherwise dissimilar area surrounding a tree is not included in the template, the bright spot will still be produced in the correlation image, but it is less likely to be enclosed by an extra dark perimeter. The darker perimeter is helpful because it makes the bright spot easier for the system to recognize as a likely tree. Still, there does remain some advantage in the practice of excluding any surrounding area from the template and including only the tree itself. The advantage is in reducing the false negatives caused by differences in nearby shadows. However, the disadvantage of doing this is in increasing the number of very conspicuous false positives, where there is nothing in the vicinity that could possibly be a tree. In general, both methods should be tried, keeping in mind the strengths and weaknesses of each. The templates shown in Figure 5.14 were chosen to produce satisfactory tree detection results in most portions of the image. In about fifteen percent of the image, however, false positives were found to be excessively high. Figure 5.15 shows an area where the straight-line shadows of burnt trees were mistaken for living trees.

Figure 5.16 shows an area where what appear to be large shrubs or small trees are commonly mistaken for the medium-sized trees being sought. About ten percent of the image is covered by this type of vegetation. These areas account for most of the false positives produced for this image.

In some areas, even a small number of false positives is unacceptable. Figure 5.17 and Figure 5.18 show two such areas. These are the two regions with a significant concentration of artificial structures. Clearly, a false positive occurring directly on top of a house or in the middle of a road would be unacceptable. Therefore, the system allows the user to manually remove each such error from the current project. This is not a lengthy process because these regions are so small. Manual correction would not be practical, however, if these regions were significantly larger.

In the remaining fifteen percent of the image, detection results were relatively good. Figure 5.19 and Figure 5.20 show two of the areas where the trees are spaced far enough from one another to show a single light gray blob coupled with a clearly

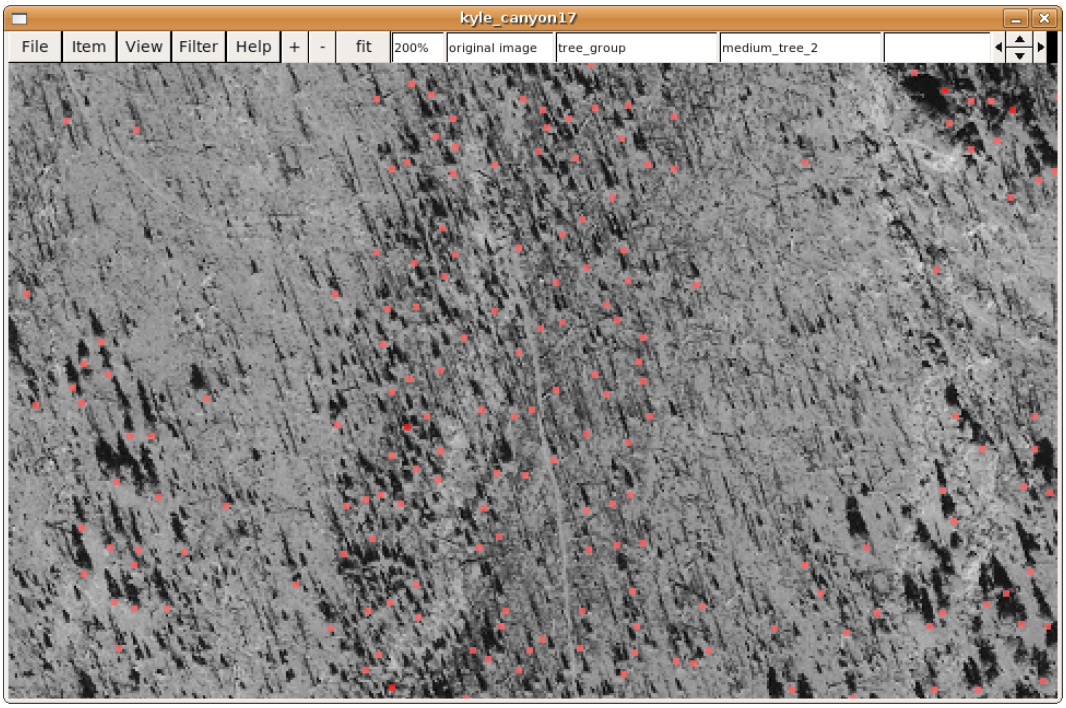

Figure 5.15: False Positives Produced by Burnt Trees.

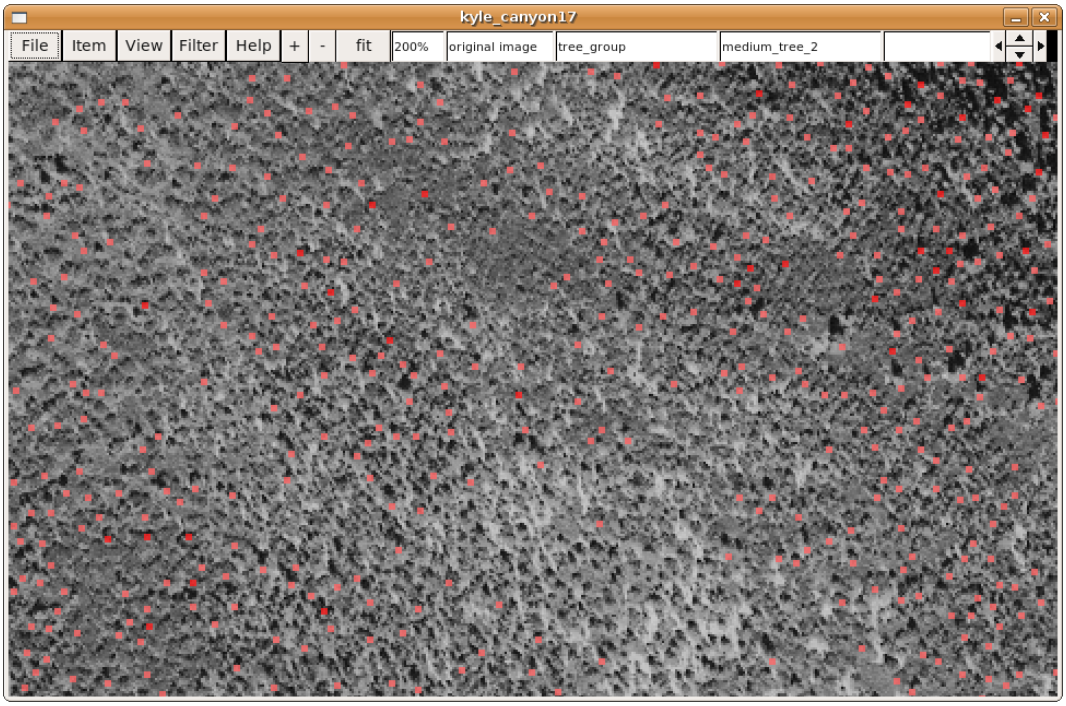

Figure 5.16: False Positives Produced by Small Trees and Shrubs.

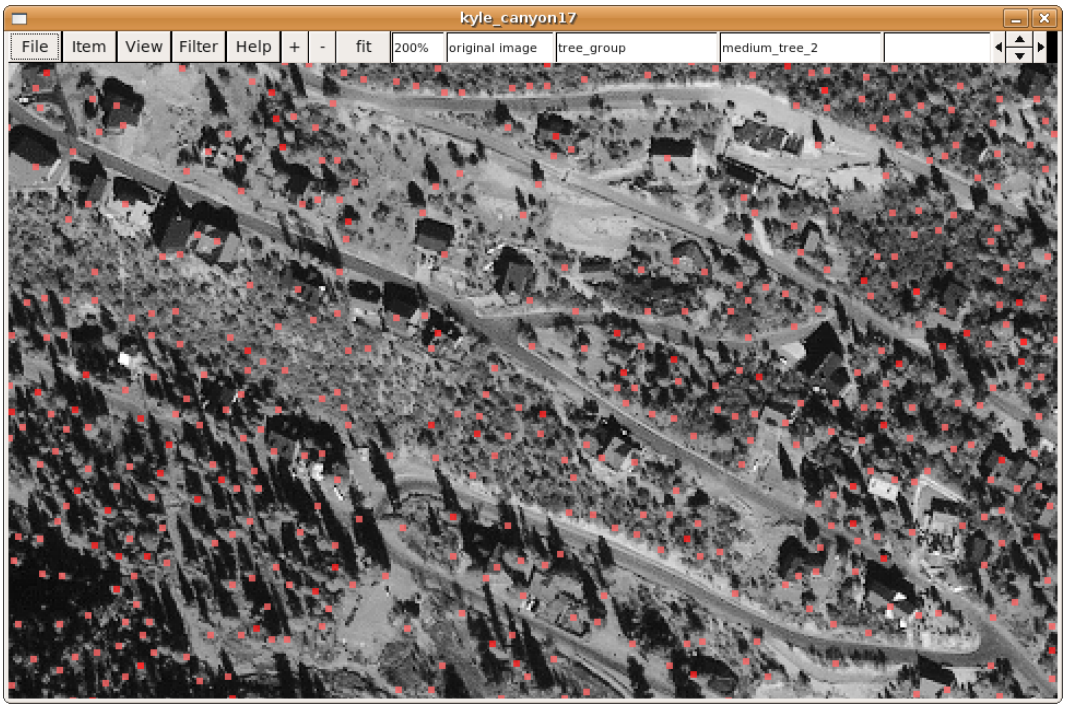

Figure 5.17: False Positives On Houses.

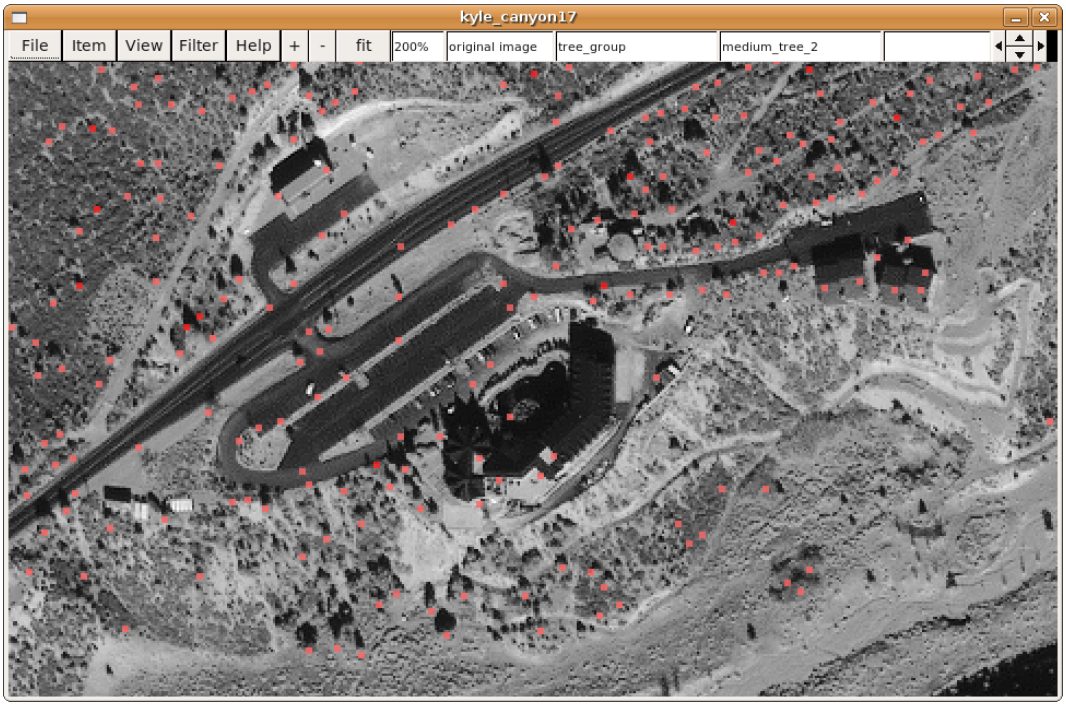

Figure 5.18: False Positives On Roads.

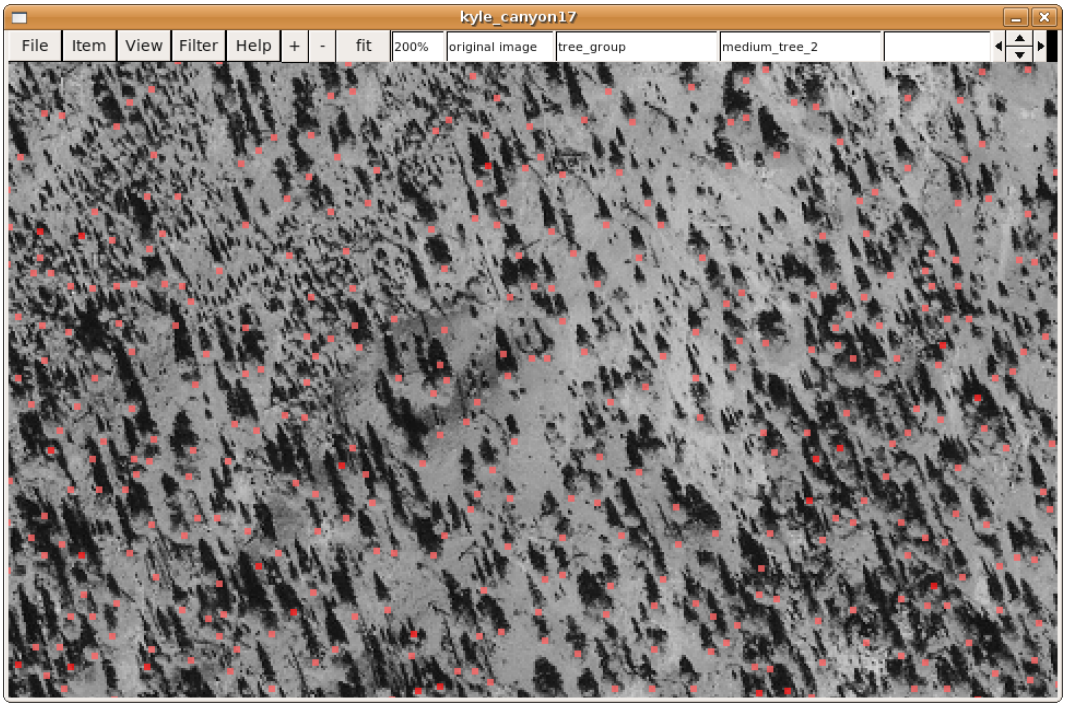

Figure 5.19: Good Results in Area with Few Items to Produce False Positives.

defined shadow. This is the best situation for tree detection. When the trees are too close together, the blobs and shadows interfere with one another and confuse the system.

There are areas of the image where tree detection accuracy is unknown. These are the regions entirely covered in shadow. Figure 5.21 and Figure 5.21 show some of the regions bordering the dark areas where tree detection is not possible.

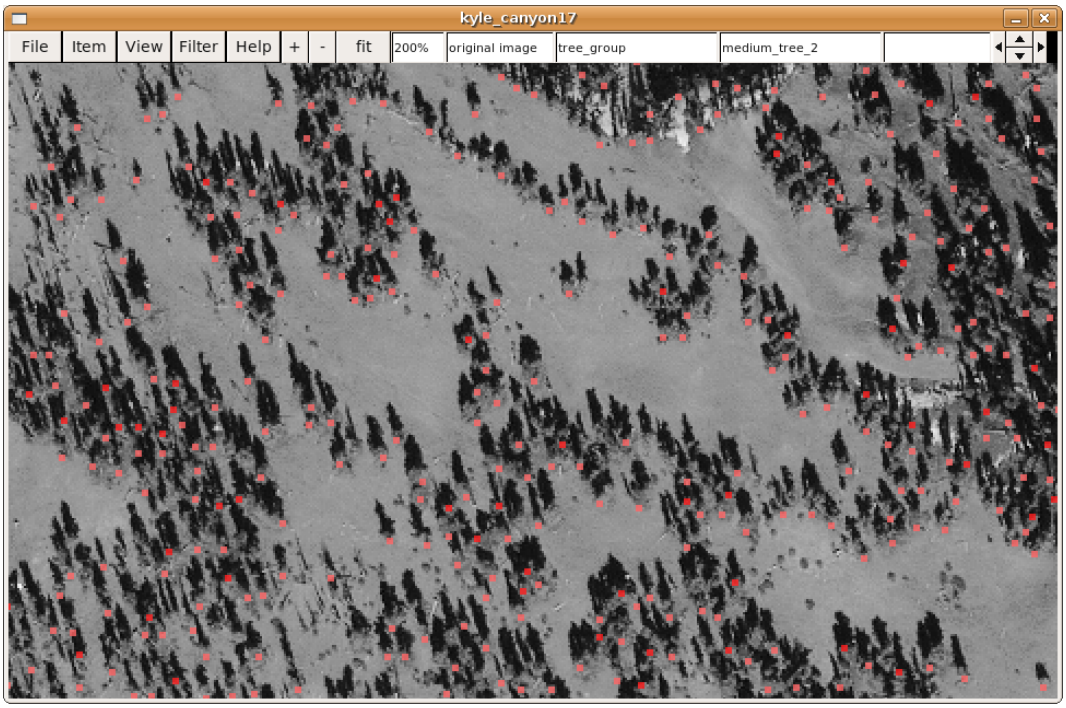

Figure 5.20: Good Results in Another Area with Few Items to Produce False Positives.

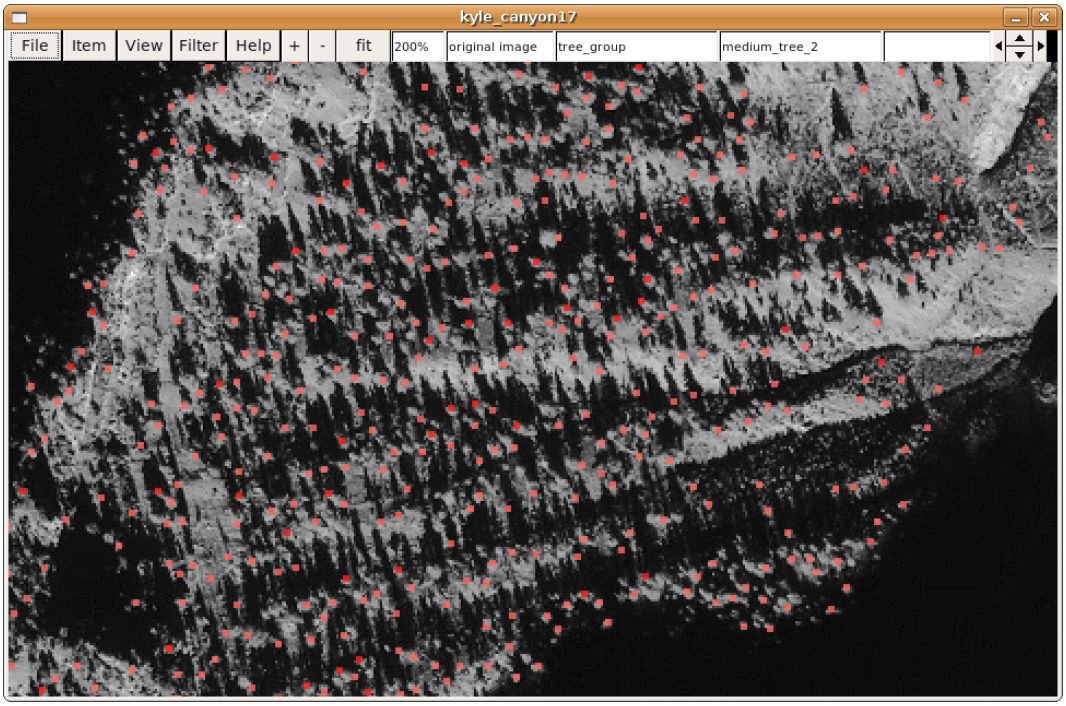

Figure 5.21: Portions of Image Obscured by Shadow.

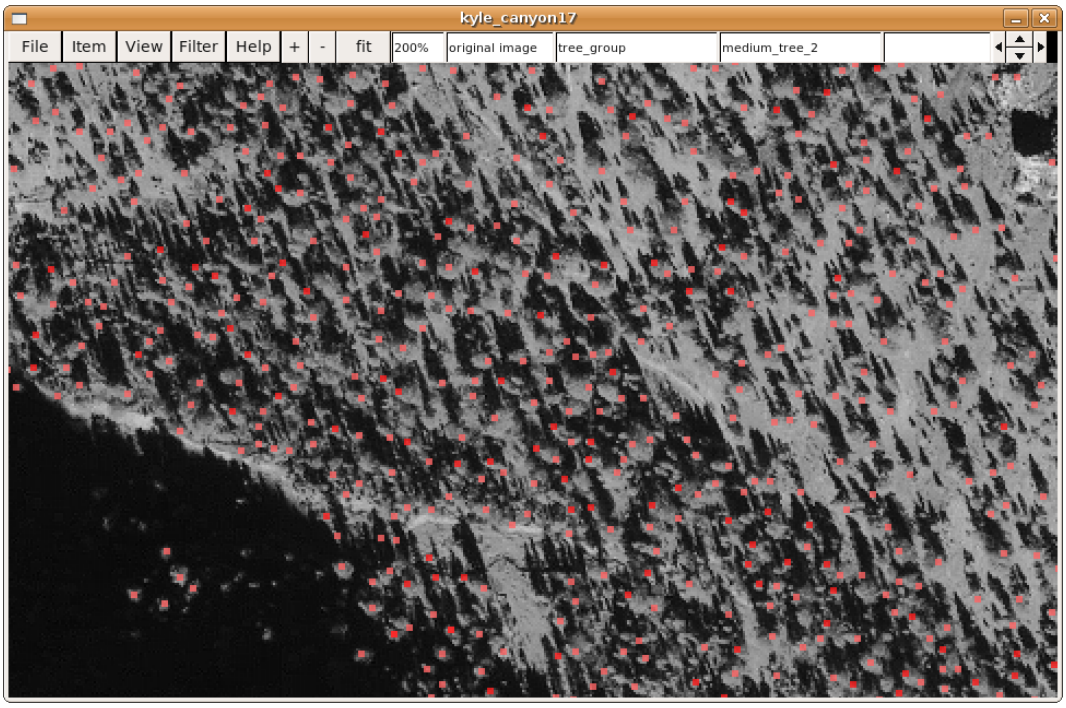

Figure 5.22: Additional Area Obscured by Shadow.

Chapter 6

Conclusions and Future Work

6.1 Conclusions

The purpose of this project is to develop an interactive image analysis system for the detection of trees in geographic images that may not be well suited to tree detection. The system relies on the judgment of the user to compensate for the lack of constraints on the factors that determine the suitability of any given automatic tree detection algorithm. All factors involved in the tree detection process are intended to be controlled by the user at runtime. Consistent with this goal is the selection of a template matching algorithm in which the user is fully responsible for defining the templates used in the detection process and all other parameters are controlled by the user as well.

Interactive tree detection using this system is a trial and error process. Ideally, the user continues this process until an acceptable level of accuracy achieved, though there is no guarantee that such accuracy will be attainable for any given image.

Using two templates, both referencing medium-sized trees in the panchromatic image of Kyle Canyon, tree detection accuracy was found to be good in about fifteen percent of the image, acceptable in about seventy percent of the image, and poor in the remaining fifteen percent of the image. These results were accomplished in a few hours including the time required to manually delete trees incorrectly placed on top of houses and on roads. Relatively few such manual corrections were necessary due to the great scarcity of houses and roads in the image. However, manual correction would

not be a practical solution for the overall problem of the many false positives occurring in less conspicuous areas. The total number of trees detected for the $8km \times 6km$ Kyle Canyon area is well in excess of 100,000. Assuming even a small percentage of these to have been placed incorrectly would likely require an inadvisable number of manual deletions. Other methods of dealing with this problem are discussed later in this chapter.

6.2 Future Work

This project currently provides the user with the ability to delete false positives one at a time, but there needs to be a way to delete large numbers of such errors all at once. Such a tool would be less precise, but it would be useful in areas where the number false positives is much greater than the number of correctly identified trees. Most likely, this would be accomplished by letting the user move the viewer to a particular region and then erasing all the tree markers within view. A better alternative to large-scale deletions would be to allow the user to change the tuning parameters from one region to the next. This would require a restructuring of the data structures to allow multiple sets of tuning parameters to be associated with each template and to allow each set of templates to be associated with a particular subregion of the image.

The *minimum item radius* tuning parameter appears to have little effect on the tree-detection results, and it is possible that this parameter should be removed. It was intended to allow excessively small trees to be eliminated from consideration. Further investigation of the usefulness of this parameter is warranted.

In the current implementation of this project, objects of the ItemRef class, the class that stores templates, are collected into groups and stored in objects of the ItemGroup class. The user selects one of the groups to be the active group. At any given time, only the templates in the active group will be visible, allowing the user to control which templates are simultaneously visible. It might be more convenient to include a flag in the ItemRef class telling the display system whether or not to display it. This would allow the elimination of the ItemGroup class. Alternatively,

some facility to allow the user to transfer templates from one group to another would make it easier to control which templates are visible at the same time. Currently, this can be done only by editing the project file. Although this is not a difficult process, a built-in facility would be helpful.

Using the utility could be made easier by allowing the user to choose the highlighting color used to specify templates. As shown in Figure 5.2, light green is currently used as the highlighting color, but this color could be difficult to see in images containing significant amounts of similar colors. In addition, it would be helpful to allow the user to cancel the filtering process. There is currently an indicator to show the current level of progress but no way of aborting the process prematurely.

The display subsystem is designed to allow the user to view very large image files without locking up. Zooming in and out is accomplished using bilinear interpolation to copy a rescaled subregion of the image raster to a fixed-size buffer. Scrolling is accomplished by changing the position of the view window over the buffer. When the view window reaches the edge of the buffer, a new portion of the image raster is copied to the buffer and the view window is restored to the center of the buffer. This results in an undesirable momentary pause while the copying and bilinear interpolation are being done. On computers that lack abundant memory, this system appears less likely to lock up than many other viewers. However, some means of eliminating these momentary pauses would be desirable if it could be done without greatly increasing memory consumption.

Currently, this project is used mainly to detect trees and other vegetation. Ideally, it could have been used in a more general capacity, to detect any type of item including but not limited to vegetation. In particular, the ability to detect houses and buildings would be desirable. Implementing such capability was investigated but not implemented because the algorithm would have to be very different from the one used for tree detection. All the houses in the image look very different from one another. In addition, the houses are often largely occluded by the surrounding vegetation. For these reasons, template matching is not likely to be successful. Instead,

an edge-detection filter would need to be applied and an algorithm used to distinguish the smooth, straight edges associated with artificial structures from the rough irregular edges associated with vegetation. Finally, a clustering algorithm would need to be applied to try to determine which edges belong to the same structure. Such an algorithm could be implemented in this project and could be useful when analyzing images containing large numbers of artificial structures, provided those structures are not located in very close proximity to one another.

Bibliography

[1] airbornelasermapping.com. Latest lidar news and web links. `http://www.airbornelasermapping.com/Features/ALMSpo02.html` (Accessed July 10, 2008).

[2] William R. Sherman Alan B. Craig. *Understanding Virtual Reality*. Morgan Kaufmann Publishers, San Francisco, CA USA, 2003.

[3] Felix Morsdorf Erich Meier Benjamin Ktz Klaus I. Itten Matthias Dobbertin Britta Allgwer. Lidar-based geometric reconstruction of boreal type forest stands at single tree level for forest and wildland fire management. *Remote Sensing of Environment*, 92(3):353–362, August 2004.

[4] C. Connolly. Latest developments in machine vision a review of image processing packages. *Sensor Review*, 23(3):193–201, 2003.

[5] Canada Centre for Remote Sensing. Fundamentals of remote sensing. `http://www.ccrs.nrcan.gc.ca/resource/tutor/fundam/pdf/fundamentals_e.pdf` (Accessed July 10, 2008).

[6] Darius S. Culvenor. Tida: an algorithm for the delineation of tree crowns in high spatial resolution remotely sensed imagery. *Computers & Geosciences*, 28(1):33–44, February 2002.

[7] Encyclopedia Britannica. Virtual reality. *Encyclopedia Britannica Online*. `http://www.britannica.com/EBchecked/topic/630181/virtual-reality` (Accessed July 10, 2008).

[8] eXtension. What are some considerations in determining whether I need aerial photography or satellite imagery? `http://www.extension.org/faq/30026` (Accessed July 10, 2008).

[9] eXtension. What is the difference between multispectral and hyperspectral imagery? `http://www.extension.org/faq/29837` (Accessed July 10, 2008).

[10] firemodels.org. firemodels.org - farsite. `http://www.firemodels.org/content/view/112/143/` (Accessed July 10, 2008).

[11] Tomas Brandtberg Fredrik Walter. Automated delineation of individual tree crowns in high spatial resolution aerial images by multiple-scale analysis. *Machine Vision and Applications*, 11(2):64–73, October 1998.

[12] Geospatial Solutions. A lidar primer. http://www.geospatial-solutions.com/geospatialsolutions/article/articleDetail.jsp?id=10275 (Accessed July 10, 2008).

[13] GISdevelopment.net. GIS Glossary. http://www.gisdevelopment.net/glossary/p.htm (Accessed August 19, 2008).

[14] Wulder M. Niemann K.O. Goodenough D.G. Local maximum filtering for the extraction of tree locations and basal area from high spatial resolution imagery. *Remote Sensing of Environment*, 73(1):103–114, July 2000.

[15] Le Wang Wayne P. Sousa Peng Gong Gregory S. Biging. Comparison of ikonos and quickbird images for mapping mangrove species on the caribbean coast of panama. *Remote Sensing of Environment*, 91(3–4):432–440, June 2004.

[16] Y. Hirata, K. Sato, A. Sakai, S. Kuramoto, and Y. Akiyama. The extraction of canopy-understory vegetation-topography structure using helicopter-borne lidar measurement between a plantation and a broad-leaved forest. *Geoscience and Remote Sensing Symposium, 2003. IGARSS '03. Proceedings. 2003 IEEE International*, 5:3222–3224 vol.5, 2003.

[17] Intersense. Is-900 wireless tracking modules. http://www.intersense.com/products/prec/is900/wireless.htm (Accessed April 1, 2007).

[18] JG Photography Group. Georgia aerial photography - aerial platform. http://georgiaaerialphotography.com/platform.htm (Accessed July 10, 2008).

[19] LANDFIRE. Landfire-faq. http://www.landfire.gov/faq.php?faq_id=3&sort_id=1 (Accessed July 10, 2008).

[20] LANDFIRE. Landfire-faq. http://www.landfire.gov/faq.php?faq_id=9&sort_id=1 (Accessed July 10, 2008).

[21] LANDFIRE. Landfire homepage. http://www.landfire.gov/index.php (Accessed July 10, 2008).

[22] LANDFIRE. Landfire national products. http://www.landfire.gov/products_national.php (Accessed July 10, 2008).

[23] M. Larsen. Individual tree top position estimation by template voting. In *Proc. of the Fourth International Airborne Remote Sensing Conference and Exhibition / 21st Canadian Symposium on Remote Sensing*, volume 2, pages 83–90, Otawa, Ontario, June 1999.

[24] Kim Dralle Mats Rudemo. Automatic estimation of individual tree positions from aerial photos. *Canadian Journal of Forest Research*, 27(11):1728–1736, 1997.

[25] Michael A. Penick. Vfire: Virtual fire in realistic environments. Master's thesis, University of Nevada Reno, Department of Computer Science and Engineering, Reno, NV 89557, May 2007.

[26] National Aeronautics and Space Administration. Earth observatory library: Remote sensing. http://earthobservatory.nasa.gov/Library/RemoteSensing/remote.html (Accessed July 10, 2008).

[27] National Consortium on Remote Sensing in Transportation. UAV2003 a road map for deploying unmanned aerial vehicles (UAVs) in transportation. http://www.ncgia.ucsb.edu/ncrst/meetings/20031202SBA-UAV2003/Findings/UAV2003-Findings-Final.pdf (Accessed July 10, 2008).

[28] National Oceanic and Atmospheric Administration. What is beach mapping data. http://www.csc.noaa.gov/products/nchaz/htm/intro.htm (Accessed July 10, 2008).

[29] Nicholas M. Short. Remote sensing tutorial introduction. http://www.fas.org/irp/imint/docs/rst/Intro/Part2_1.html (Accessed July 10, 2008).

[30] Open Source Geospatial Foundation. GDAL: GDAL - geospatial data abstraction library. http://www.gdal.org/formats_list.html (Accessed July 10, 2008).

[31] Open Source Geospatial Foundation. GDAL: GDAL - geospatial data abstraction library. http://www.gdal.org/ (Accessed July 10, 2008).

[32] Paul R. Baumann. History of remote sensing, aerial photography. http://employees.oneonta.edu/baumanpr/geosat2/RSHistory/HistoryRSPart1.htm (Accessed July 10, 2008).

[33] Pouliot D.A. King D.J. Bell F.W. Pitt D.G. Automated tree crown detection and delineation in high-resolution digital camera imagery of coniferous forest regeneration. *Remote Sensing of Environment*, 82(2):322–334, October 2002.

[34] G. J. Awcock R. Thomas. *Applied Image Processing*, pages 9–10. McGraw-Hill, Inc, New York, NY USA, 1996.

[35] G. J. Awcock R. Thomas. *Applied Image Processing*, page 127. McGraw-Hill, Inc, New York, NY USA, 1996.

[36] Robyn Owens. Stereo matching. http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/OWENS/LECT11/node5.htm (Accessed July 10, 2008).

[37] James E. Gentle J. Chambers W. Eddy S. Sheather. *Numerical Linear Algebra for Applications in Statistics*, page 57. Springer-Verlag, New York, NY USA, 1st edition, 1998.

[38] H. Mitasova W.M. Brown M. Hohmann S. Warren. Computing erosion modeling inputs from a dem. http://skagit.meas.ncsu.edu/~helena/gmslab/reports/CerlErosionTutorial/denix/Data/DEM_computing_erosion_modeling_input.htm (Accessed July 10, 2008).

[39] Satellite Imaging Corporation. Generating digital elevation models (dem) from satellite imagery. http://www.satimagingcorp.com/svc/dem.html (Accessed July 10, 2008).

[40] Sky-Borg Unmanned Aerial Vehicle Photography. Sky-borg unmanned aerial vehicle photography - about. http://www.sky-borg.com/about.htm (Accessed July 10, 2008).

[41] Songxin Tan. lidar. http://www.engineering.sdstate.edu/~tans/lidarremotesensing.htm (Accessed July 10, 2008).

[42] Spencer B. Gross. Lidar mapping technology at sbg. http://www.sbgmaps.com/lidar.htm (Accessed July 10, 2008).

[43] John B. Lane Louise McMonegal Russell Mike1 Jane Radatz Lawrence Reeker Charles Russell Paul Schmid Ernest Stalder Scott Tucker Mary Yee Ronald Berlack Susan Chonoles William Dupras Daniel Garvin Shirley Gloss-Soler John Goetz Virl Haas Robert Haralick Richard Leahy Philip Marriott Rollin Mayer Theo Pavlides Azriel Rosenfeld Leonard Seagren Sonja Peterson Shields Jack Sklansky Marti Szczur Fermin Trujillo Terry Weymouth. IEEE standard glossary of image processing and pattern recognition terminology. *IEEE Std 610.4-1990*, Mar 1990.

[44] United States Geological Survey. USGS digital elevation model (dem). http://edc.usgs.gov/guides/dem.html (Accessed July 10, 2008).

[45] United States Geological Survey. USGS Earth Resources Observation and Science (EROS). http://edc.usgs.gov/guides/glossary/c_d.html (Accessed July 10, 2008).

[46] United States Geological Survey. USGS terraweb for kids: Glossary. http://terraweb.wr.usgs.gov/kids/glossary.html (Accessed July 10, 2008).

[47] Vishvjit S. Nalwa. *A Guided Tour of Computer Vision*, page 96. Addison-Wesley Publishing Company, 1993.

[48] William R. Sherman. FreeVR: Virtual reality integration library. http://www.freevr.org/ (Accessed July 10, 2008).