# An Application for Tree Detection Using Satellite Imagery and Vegetation Data

David T. Brown[+*] Roger V. Hoang[+*] Matthew R. Sgambati[+*]
Frederick C. Harris, Jr.[+*]

Department of Computer Science and Engineering[+]
University of Nevada, Reno
Reno, NV 89557

Desert Research Institute[*]
Reno, NV 89512

{dtbrown, hoangr, sgambati, Fred.Harris}@cse.unr.edu

**Abstract**

Virtual reconstruction of large landscapes from satellite imagery can be a time-consuming task due to the number of objects that must be extracted. Poor image resolution and noise hinder automatic detection processes and thus must be corrected by the user. This paper describes an application that allows the user to guide the automatic detection of trees from satellite imagery and spatial vegetation data. The requirements of the system are specified and an architecture that satisfies these constraints is presented. The resulting application provides an intuitive computer-aided method for the selection and classification of trees.

Figure 1: Fire visualization in progress

## 1 Introduction

VFIRE [1, 3] is an immersive visualization application being developed to study the behavior of wildland fires and to train fire crews to better combat this phenomenon. Figure 1 is an example of the output generated by the program. The system attempts to generate realistic visualizations of simulated wildfires over real-world locations by combining satellite imagery and other spatial data. In order to create an accurate representation of the world, objects found in the satellite imagery should be placed correctly in the virtual world.

To do so, the type and location of these objects must first be extracted from the images. Unfortunately, determining this information for each individual object can be a very complicated and time-consuming task, especially due to the sheer number of objects such as trees. Computer vision techniques to automate this process can be effective. However, noise and other factors inhibit the accuracy of these techniques, leading to false positives and false negatives. Therefore, this application was developed 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, the application 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.

The remainder of this paper is structured as follows:

Section 2 outlines the requirements specification; Section 3 details the use cases for the application; Section 4 gives an overview of the components used for the tree detection algorithm; Section 5 discusses the relationships between the major subsystems of the application; Section 6 shows the results of the project; finally, Section 7 draws some conclusions and suggests paths for future work.

## 2 Requirements Specification

### 2.1 Functional Requirements

The functional requirements outline the necessary components that allow for the user to manipulate the view of the landscape, select candidate tree templates, and refine the results of the tree detection algorithm.

1. The utility shall allow the user to load a photographic image of a geographic area.
2. The utility shall allow the user to display the photographic image.
3. The utility shall allow the user to zoom and scroll the image.
4. The utility shall load vegetation cover, vegetation type, and vegetation height maps if available.
5. The utility shall display vegetation map information for user-specified locations.
6. The utility shall allow the user to display one of the vegetation maps.
7. The utility shall allow the user to display one of the vegetation maps on top of the image.
8. The utility shall allow the user to select image regions to use as tree templates.
9. The utility shall allow the user to select one template as the active template.
10. The utility shall allow the user to edit the active template.
11. The utility shall allow the user to associate a name, type, height, and width for each template.
12. The utility shall allow the user to place templates into groups.
13. The utility shall allow the user to adjust tuning parameters for tree detection.
14. The utility shall allow the user to filter the image to produce a correlation image.
15. The utility shall allow the user to display the correlation image.
16. The utility shall search the correlation image for trees.
17. The utility shall search only within the region currently in view.
18. The utility shall place a mark at each location where a tree is detected.
19. The utility shall allow the user to place a tree mark in any location
20. The utility shall allow the user to mark any location as the location of an artificial structure.
21. The utility shall allow the user to delete any mark from any location.
22. The utility shall allow the user to save all data in a project folder.
23. The utility shall allow the user to load a project from an existing project folder.
24. The utility shall allow the user to output all item locations to a file usable by V-FIRE.
25. The utility shall allow the user to create rough tree placements based only on vegetation maps.
26. The utility shall mark tree placements made based on veg map data.
27. The utility shall allow the user to output map-based tree placements to a file usable by V-FIRE.

### 2.2 Nonfunctional Requirements

The nonfunctional requirements 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.

1. The utility shall directly implement graphics to create and manage templates.
2. The utility shall directly implement graphics to display, scroll, and zoom image.
3. The utility shall use bilinear interpolation to zoom into and out of the image.
4. The utility shall swap correlation images to and from disk to match currently active template.
5. The utility shall use the brightness component of the image for tree detection.
6. The utility shall perform contrast stretching on the correlation image.
7. The utility shall be implemented on the Linux platform.
8. The utility shall be written in C++.
9. The utility shall use GDAL to read image files.
10. The utility shall use FLTK for its GUI.

## 3 Use Cases

The operations commonly used in the tree-detection process are shown in Figure 2. The use cases show that the user is given the flexibility to decide at run-time 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. This data is 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.

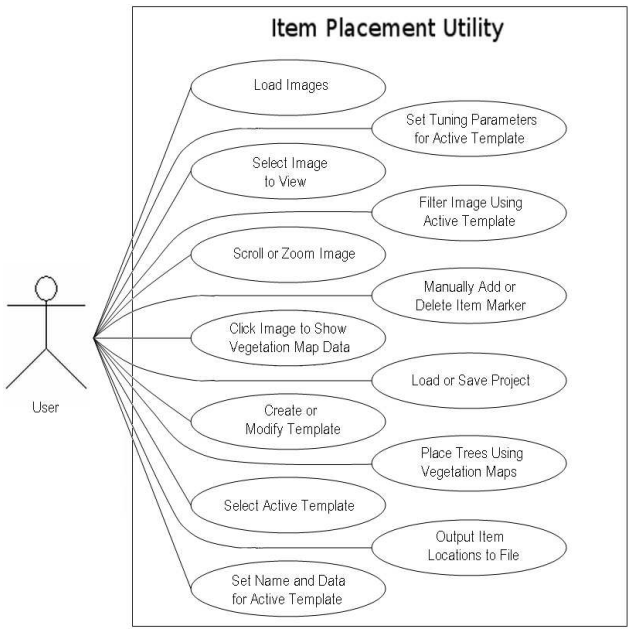


Figure 2: Use Cases

## 4 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 3 shows the classes used in this project.


Figure 3: Classes used by application

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 this data from VFIRE.

## 5 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

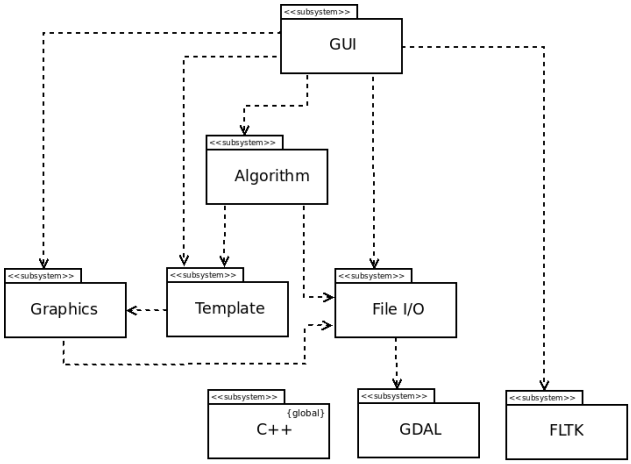


Figure 4: System Structure

is straightforward and predictable in terms of which operations will be performed on which class objects. Figure 4 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 [4] 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 [2] 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, the template subsystem relies on the graphics subsystem to draw the marks. The template subsystem also keeps the data in the active template current as the zoom level or other view conditions change.

The Graphics subsystem includes all the functions that display images on screen through FLTK [5]. 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.

## 6 Results

Figure 5 shows a screenshot of the developed application without any detected trees. The user can zoom into a particular area as shown in Figure 6, allowing the user to more easily outline a template used for tree detection. As the user selects template pixels in the image, they are highlighted as seen in Figure 7.

Once a template is created, the tree detection algorithm finds all objects that match the template to some user-specified threshold. The centers of the matches are then displayed as red dots like those shown in Figure 8. False positives can then be removed and missed trees added by the user.

The user is also allowed to have the application automatically generate randomized tree locations based on fuel data. This feature is useful for areas where trees cannot be detected due to shadows and/or poor image quality. Figure 9 demonstrates this function.

## 7 Conclusions

Detection of trees from images of large forested landscapes can be an extremely time-consuming task.

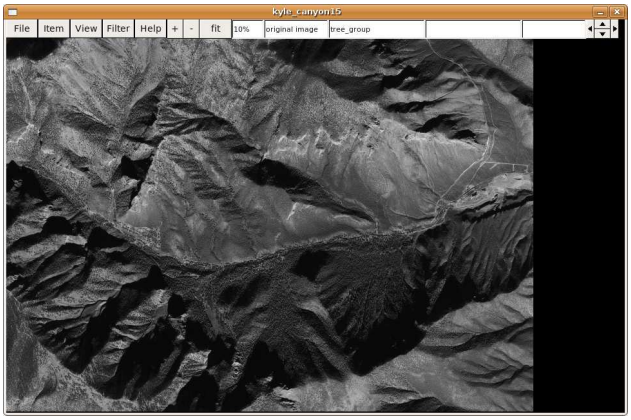


Figure 5: Application image before tree detection

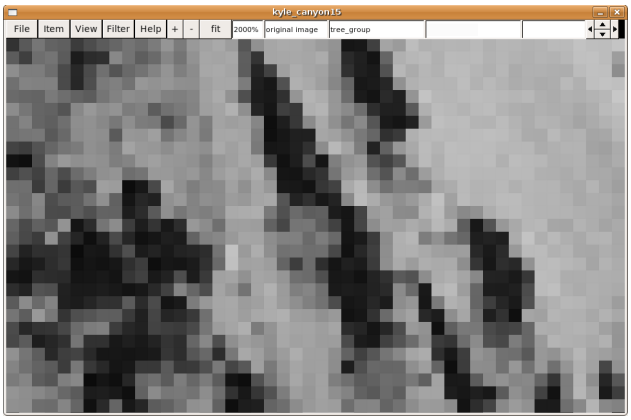


Figure 6: Enlarged view of tree centered in window

While automation is possible through computer vision, factors such as image resolution and noise generate inaccuracies. Thus, an application was developed that combines automation with user input to compensate for these problems. Despite the simplicity of the architecture, the application described in this paper effectively allows any user to quickly and straightforwardly detect trees by specifying templates and other parameters.

In the future, using the utility could be made easier by allowing the user to choose the highlighting color used to specify templates. As shown in Figure 7, 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.

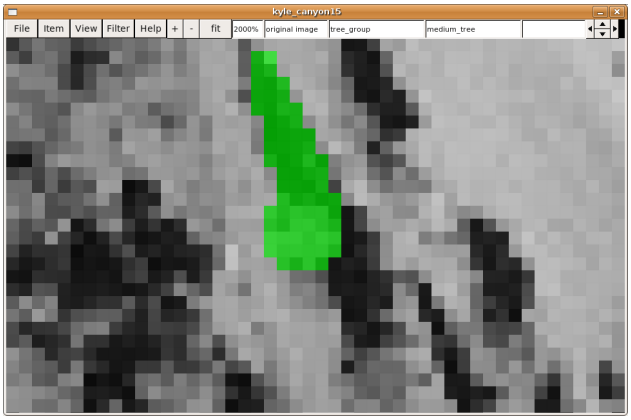


Figure 7: Enlarged view of tree highlighted by user

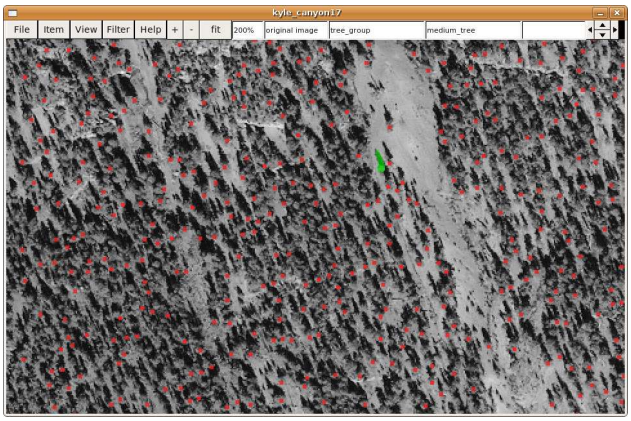


Figure 8: Likely trees marked after processing

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.

## Acknowledgements

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

## References

[1] 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.

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

[3] M.A. Penick, R.V. Hoang, F.C. Harris Jr, S.M. Dascalu, T.J. Brown, W.R. Sherman, and P.A. McDonald. Managing Data and Computational Complexity for Immersive Wildfire Visualization. *Proceedings of High Performance Computing Systems (HPCS'07)*.

[4] N. Ritter, M. Ruth, B.B. Grissom, G. Galang, J. Haller, G. Stephenson, S. Covington, T. Nagy, J. Moyers, J. Stickley, et al. GeoTIFF format specification GeoTIFF revision 1.0. *NASA, Jet Propulsion Laboratory, Pasadena, CA, and SPOT Image, Reston, Virginia (http://www-mipl. jpl. nasa. gov/cartlab/geotiff/geotiff. html)*, 1995.

[5] B. Spitzak et al. FLTK-The Fast Light Tool Kit. *URL: http://www. fltk. org*.

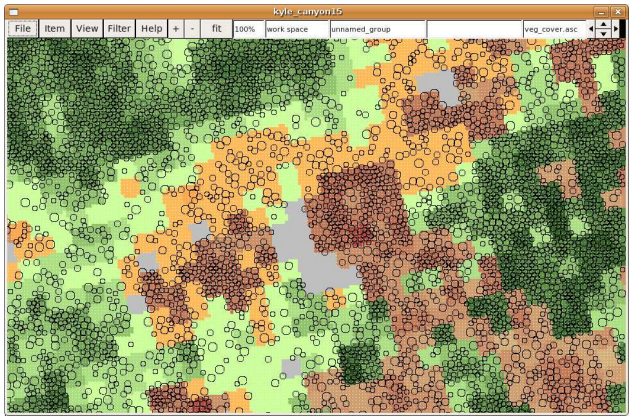


Figure 9: Random tree placements made according to vegetation data. Different colors represent different types of vegetation. The circles denote the location and size of trees generated.