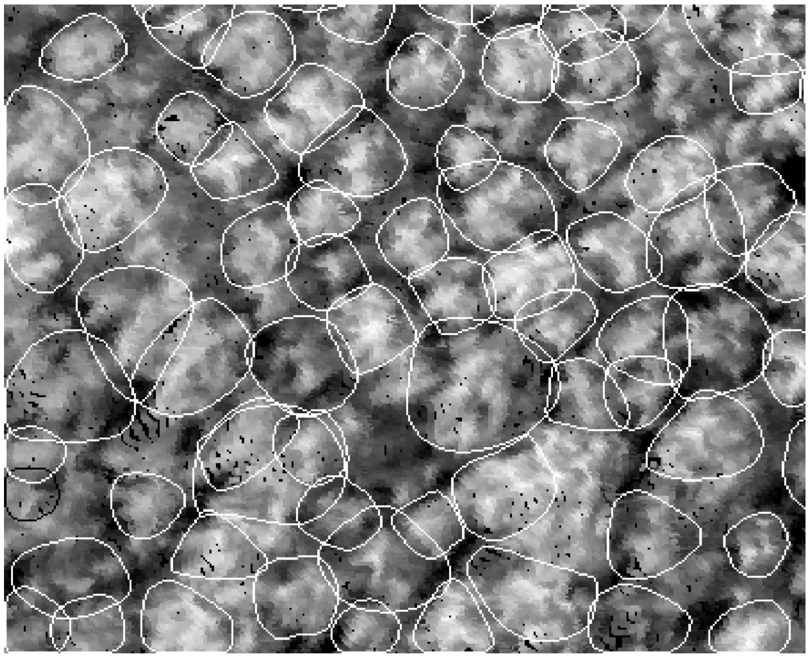
Automatic Tree-Crown Segmentation Using LM Filters and Balloons



Master's Project for

Bert Rawert

May 2004

Advisors: Ed Riseman and Howard Schultz

Table of Contents

Abstract *– This work proposes an algorithm capable of performing fully automatic segmentation of tree-crowns using information extracted from aerial imagery. First, a DEM (digital elevation map) and orthorectified image are computed from the imagery and corresponding estimates of camera orientation. Next, treetop locations are detected using a variation on a LM (local maximum) filter which fuses information from the DEM and the orthoimage. Then, at each treetop, a balloon (or a "snake" with an outward normal force) is initialized automatically, and is iterated until convergence. The balloon is influenced by forces from both the DEM and the orthoimage. The result is a fully automatic scheme for segmenting individual tree-crowns. A comparison with expert hand segmentations is also presented.*

I. Introduction

The goal of this Master's project report is to document the progress made so that the tools developed are useful to future students and staff, but also so that research can continue in the areas covered by this work. The specific research goal of this work is to develop an algorithm for automatically segmenting tree-crowns, or delineating a boundary around individual trees from derived DEMs and orthorectified imagery.

Automatic recognition of trees in almost any form is useful. Manual forest inventory is expensive and time consuming. Work presented here proposes an automatic tree-crown segmentation algorithm and a strategy for forest inventory that is highly cost effective. When the trees have been segmented, they can be counted, and their size (crown area, and other extrapolated measures) can be analyzed automatically. As a subject for future work, the individual trees also could be automatically classified by species. This pipeline greatly reduces the cost and time needed to collect forest inventory data. This particular work demonstrates that using 3D information in conjunction with 2D photographs can provide an improved result over using just one of the sources of information to perform automated forest inventory.

This report is divided into a number of sections. First, this introduction briefly outlines the work and the report. Next is a general discussion of the strategies used in the attempts to achieve the goal of this project. Following that, a literature review summarizes much previous research related to the work in this project. A more detailed description of the tree-crown segmentation algorithm comes next. Finally, results from some experiments are reported and discussed, and conclusions are drawn.

II. Strategies for Automatic Tree Crown Segmentation

Segmentation of objects in images is a frequently studied topic in computer vision research literature. It is the problem of identifying the boundaries of an object's appearance in an image. In this work, I have studied the problem of segmenting individual trees in aerial images taken of a forest canopy. If the locations, sizes, and types of all the trees in an image are known, then forest monitoring and inventory is much easier. Useful forest statistics can then be extracted from the images, too, such as biomass [Slaymaker99], which can be used to estimate the amount of carbon absorbed by a forest.

Tree segmentation is not an easy problem to solve manually or automatically. To segment the trees manually, an operator must sit in front of a computer and draw circles around, or trace the boundaries of, the trees. Sometimes this involves using multiple images and a workstation equipped with 3D display capabilities so the operator can be more accurate. Still, the trees can be hard to visually separate. There have been attempts to automate the process, but none of these have presented a satisfactory solution, either. The algorithm here looks to find a middle ground. It allows for complete automation of the process of tree segmentation, but it can also operate in semi-automatic mode, attempting to minimize user interaction by reducing it to one click per tree.

The usual approach to this, or any segmentation problem in computer vision, uses only the pixel values of a single image. Many use active contours (for example [Kass88], [Tang04], [Collet94], [Chuang01]), as does this work, but other strategies are often used, too, such as local maximum filters [Daley98], [Wang04], watershed segmentation [Wang04], and edge detection [Wang04]. Other strategies use only depth information ([Gong02], see Literature Review). Even Markovic, et al. [Markovic03], who attempt to combine depth and image information, operate on only one image at a time using an “edge combination” image computed using depth and image information, but this throws a lot of information away. The approach here uses a digital elevation map (DEM) computed from the sequence of images captured of the forest and the corresponding orthorectified images rather than the original aerial images. This approach uses not only the pixel values at a particular location, but also the height at that location. Combining these two sources of information better follows the process used by the human vision system, which makes use of stereo vision for depth information in addition to color information.

Having these sources of information, the problem is one that could be solved using “snakes.” A snake, in the computer vision sense, is a set of points (“snaxels”) whose locations can change iteratively from their initial positions to move closer to the boundary of an object. This works by allowing each point to move in a way that minimizes internal energies (based on the shape of the overall contour) and external energies (based on the data being segmented, such as the image intensity gradient). To get an intuitive feel of original snakes, one can imagine putting a noose around an object, and tightening it around the object. The points for the snake are like a sampling of points on the boundary formed by the loop of the noose. They move as the noose would, until they form a good sampling of points on the boundary of the object being segmented. To solve the problem of tree-crown segmentation, however, I use a snake whose internal energy causes it to expand from inside the tree-crown boundary until it fills the boundary. This type of snake is called a "balloon." Importantly, the external energies of the balloon in this research depend on both the pixel values in an orthorectified image and the height values of the corresponding DEM. This approach covers new ground in the use of snakes, and it is presented in detail in a following section.

Now, in order to use the balloon, it must be initialized. To do this, we employ a treetop finding strategy similar to a local maximum (LM) filter, but which attempts to find points which are local maxima in both the DEM (high spots) and orthorectified image (bright spots). These are used as treetops, and then the balloon is initialized as a circle whose radius is the minimum expected radius of any tree-crown in the image. The balloon iterates until convergence, thus segmenting the tree-crown. This process repeats until all the points found by the treetop search algorithm have been processed. Some segmentations are thrown out using tests to determine whether or not the segmentation is likely to be a tree.

Assumptions

The algorithm operates under some assumptions. First is a basic model for a tree-crown, which assumes that trees have one highest point (the treetop), and that the rest of the tree-crown forms a surface that tends to decrease in elevation along any direction moving away from the treetop. Further, we assume that the boundary of the tree-crown is roughly circular [Song03]. This yields a model that is roughly like a dome shape or a half-sphere to represent the tree-crowns. In addition to these physical assumptions, we assume the tree appears brightest near its center [Daley98], [Wang04], and tends to be darkest near its boundary when looking at it in the image. Based on this model, we attempt to segment a tree given a good guess at its treetop location. See Figure II.1 below to better understand the basic model:

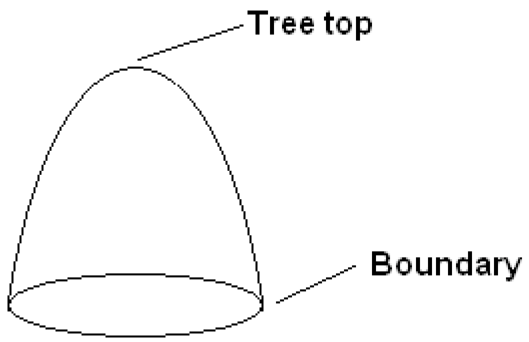


Figure II.1. *Basic model of the physical shape of a tree, as it appears in the DEM, consistent with our assumptions.*

III. Literature Review

III.A. Active Contour Models

The active contour models used in computer vision for segmentation purposes, known as snakes, have a long, successful history. The concept was originally presented in [Kass88]. A snake is "an energy-minimizing spline guided by ... image forces that pull it toward features such as lines and edges". The snake is represented by a sequence of snaxels, or nodes interpolated by the spline. These snaxels move iteratively, governed in each iteration by internal forces, which help control the continuity and curvature of the spline, and external forces, which draw it closer to image features. The iteration stops when a criterion is met; this is usually tested by thresholding net snake movement or the difference between successive contours. Snakes can be used to recover subjective contours, detect and compute stereo disparities, and track motion in video sequences. They need not be closed contours, but can be used to find edges, lines, and curves, in addition to complete closed contours around objects.

In the original formulation, snakes for object segmentation are initialized by supplying a set of snaxels which “closely” surround the object's boundary. Then, the snake slowly shrinks until it “locks on” to the image features which allow it to minimize its energy. One difficulty in using snakes is finding an initial set of points that are close enough to the pertinent image features. To solve this problem, a modification of the internal forces is proposed by Cohen [Cohen91] which causes the snake to expand until it locks on to the object boundary. This new snake model, called a “balloon,” need not be initialized close to the solution to converge. The constant balloon force will continue to expand the contour until the external forces on the contour balance against the balloon force to minimize the total energy.

Wang and Ghosh [Wang98] introduced a new model for the balloon force. This force is not simply the outward unit normal to the contour at each snaxel (as in the original balloon force [Cohen91]), but its magnitude varies spatially. The goal is to solve a problem that can occur with the original balloon force in some situations. The problem is that the balloon force can be too weak to overcome the image (external) force before the contour expands to the desired location, or the force could be too strong and expand the contour past the desired location. Wang and Ghosh solve this problem by first adding a weight function, not just a weight constant, to be multiplied by the normal force. This function depends on the external image forces. When this force drops below a threshold, then the weight on the balloon force becomes zero, and the contour stops expanding except possibly to fill in details of the contour.

The proposal and improvement of balloons has significantly enhanced the usefulness of active contours for image segmentation. However, even with the reduced initialization complexity yielded by the balloon method, contour initialization requires the specification of an initial contour by an outside source of knowledge; usually this is a human. Chuang and Lie [Chuang01] propose a method of automatic segmentation of multiple objects using snakes. To do this, they invent a snake initialization algorithm which moves “active points,” initially in an equally spaced grid arrangement, toward the local energy minima based on external snake forces. When the points stabilize, a clustering algorithm divides the points into groups. Then, the convex-hull is computed for each cluster, and this is used as the initial contour, at which point the snake “takes over” control of the points, which move according to the internal and external snake energies. This approach is a big improvement, but seems susceptible to deep “sinks” in the image, which could draw grid points away from other less powerful, yet interesting points. Also, since the convex hull is used, outliers could severely sabotage the initialization so that the contour is grossly distant from its intended object boundary. Were balloons used, however, this would not be such an issue.

Another problem that can plague snakes and balloons is the formation of loops. Ji and Yan [Ji99] discuss this. Their planned solution to this problem is to detect when the contour crosses itself, and then remove the loop. This is possible because they use an “attractable” snake model which requires snaxels to be in discrete locations, allowing storage of the complete contour in memory. This is the fast greedy snake algorithm proposed by Lam and Yah [Lam94], which is based on the original greedy snake algorithm proposed by Williams and Shah [Williams92].

As previously mentioned, a benefit of snakes is that they can segment subjective contours. Segmenting images of clouds [Collet94] supplies a great illustration for this. Clouds have unclear boundaries, so segmentation methods which look for edges will generally fail. However, a closed contour around a cloud can be obtained using snakes. The continuity and rigidity properties of snakes provide for a segmentation that makes sense when boundaries are unclear. When parts of the cloud boundary are particularly tough to discern, the snake will tend toward a smooth curve anchored by those snaxels in areas of the contour that are clearer.

Many successful applications of snakes result from research in biological and/or medical topics. One recent example, in addition to many of the works already cited, is that of Tang and Acton [Tang04], where the researchers use snakes to localize the boundaries of a blood vessel in microscopic images. The contributions of this application attempt to tackle some other problems that snakes encounter. One is that setting the weights on the internal and external forces is a complex, error-prone task. They solve this by implementing the snake as a B-spline which interpolates the snaxels. As a result, the rigidity and curvature force weights are implicit, and, while they cannot be directly controlled, they don't need to be, either. This simplifies the task of successfully applying snakes. The second problem they reduce is that snakes are often susceptible to noise points. A spurious bright spot in an image could snag the snake and prevent it from reaching the optimal boundary. To tackle this, the authors refine the snake's position on the image using a coarse-to-fine image scaling strategy. First, they heavily blur the image, and once the snake converges, they reduce the blurring and let the snake converge again. This repeats until the snake has converged in the original, non-blurred image.

A scaling mechanism is also addressed by Whitten [Whitten93], who relates the data scale to the internal constraints of an active contour (not necessarily a snake) for purpose of scale space tracking. By requiring only one parameter to control both the rigidity of the contour and the scale of the data, the scale space is more easily traversed.

The use of depth maps in combination with snakes is introduced by Markovic and Galautz [Markovic03]. In this work, the authors first compute a depth map of the scene in question using a stereo pair of images. Then, they compute an "edge combination image" which has only the edges detected in both the intensity image and the depth image by a Canny edge detector operating on each. They present results on two test scenes which appear to show that this method of combining edges results in a better segmentation. Unfortunately, the authors do not tell us how many snaxels are used in each situation (so we can assume that these vary). Further, they do not distinguish between a "depth image" and a "disparity image," nor do they tell us whether they compute an orthoimage before computing the edge combination image, or if they somehow align the depth map with the pixels of the intensity image. This makes it hard to get a feel for how successful their method is. However, there are much better ways to combine depth and intensity information without throwing away a lot of useful image detail than by simply applying the AND operator on the edge images.

III.B. Tree-Crown Detection and Segmentation

Tree-crown recognition from aerial images has been studied for decades because successful application offers substantial savings in effort, cost, and time over manual field sampling, and it is less subjective and less dependent on individual human operators, who may differ in their levels of experience. The goals of using aerial images in forestry are to reliably monitor forest characteristics in terms of counting trees, determining species and species distributions, and monitoring tree-specific properties such as size, volume, and age. In order to do achieve these goals, it is important to be able to detect trees and eventually delineate their boundaries (i.e. segment them) automatically.

One of the popular early strategies for treetop detection and counting was the use of local maximum (LM) filters. This simply uses the assumption that the image of a tree is brightest at the tree's highest point, the treetop. In Daley, et al. [Daley98] refine this approach by allowing for variable window size (VWS) LM filters. They use image semivariance to leverage image structure in determining the best LM window size to use at each location in an image. This yields higher tree-location accuracy for the detected treetops, though there were still a large number of omission and commission errors.

Another strategy for finding trees is a model-based approach. This approach constructs a model of a tree's appearance in an image or depth map using specific knowledge about the trees and forest being studied. Then, the data are searched for good fits with this model. Most of these approaches only look at the elevation (or range) information or the image intensity information and not both. One model-based approach presented by Gong, et al. [Gong02] is actually a manual approach with a well designed user interface. The user marks the tree's base elevation and treetop. Then, based on the dominant species, the system initializes the model with typical parameters for that species given the tree height. The user can then modify the parameters to better fit the tree.

Wang, Gong, and Biging [Wang04] propose a method for automatic tree-crown segmentation which integrates a treetop detection algorithm. In this work, treetops are identified by intersecting the results from two treetop detection algorithms. First, they perform local non-maximum suppression to find local peaks in the image which are then merged using eight-connected neighborhoods. Next, they find local maxima in a morphologically transformed distance (geodesic distance to nearest edge point) image. Then, they intersect these two sets of treetops to get a final set which is then used as a set of markers for a marker-controlled watershed segmentation. This produces a set of treetops and a set of segmented tree-crowns. Unfortunately, these methods are only valuable within a small field of view ($\pm$ 15° of nadir as stated in their paper) because they heavily depend on the treetop being near the center of the image of the tree-crown. This limitation should disappear if they correctly calibrate their camera and ortho-rectify their images. Also, this method tends to identify too many treetops within clumps, and it finds trees where there may not be any as observed in a comparison with human segmentations. This method does not use 3D information, though the authors mention that 3D could help improve the accuracy of their segmentations.

IV. Detailed Discussion of Tree-Crown Segmentation

The strategy for automatic tree-crown segmentation used here can be divided into two stages. First, treetops must be identified. Then, once the treetop is identified, the segmentation algorithm takes over and delineates the boundary of the tree corresponding to that treetop. In order to efficiently automate the entire process of tree-crown segmentation for a forest canopy DEM and orthorectified mosaic, there is some additional organizational processing that must control the work of the first two stages. This discussion will, therefore, also be divided into three sections. The reader should note that when the algorithm references data in the DEM or orthoimage, it is referring to the blurred versions of those images, as resulting from a Gaussian blurring with a specified radius.

IV.A. Treetop Detection

The minimum tree-crown radius is a parameter specified by the users, and is used repeatedly, including during treetop identification. The treetop algorithm used in this work is a modified version of the local maximum (LM) filter approach modified to use both the image and the DEM. This particular implementation depends more heavily on the orthoimage than on the DEM when finding treetops. First, a circular window, whose radius is that of the minimum tree-crown, moves across the image and the DEM, and local maxima are marked. Then, a second pass is made through the data. In this second pass, the circular neighborhood around each image maximum is searched for a marked local maxima in the DEM. The midpoint between the closest DEM maximum found in this neighborhood and the current image maximum is marked as a treetop. If there is no DEM maximum within the circular neighborhood around the image maximum, then the image maximum is marked as a treetop. This should result in a treetop count that is slightly higher than the actual number of trees in the scene. We will use this to our advantage later when automating the process. All DEM maxima not within the specified radius of an image maximum are thrown away. Below, Figure IV.A.1 shows the result of this algorithm overlaid on a highly blurred orthoimage.

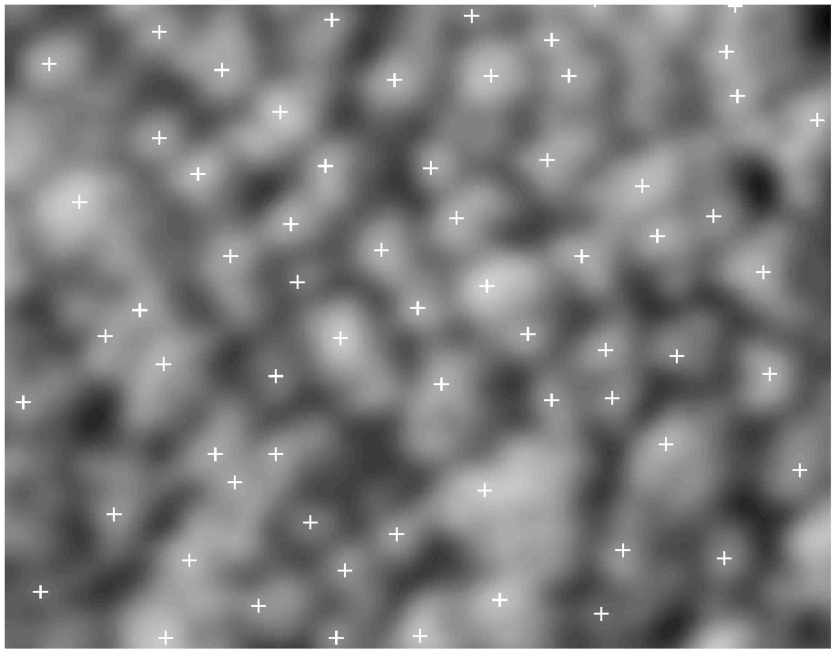


Figure IV.A.1. *Detected treetops marked with white crosshairs overlaid on a highly blurred orthoimage. This is the result of the treetop detection algorithm described in section IV.A.*

IV.B. Tree-Crown Segmentation with Active Contours

The tree-crown segmentation algorithm takes as input a treetop location and a number of parameters loosely characteristic of the data to be segmented. The tree-crown segmentation algorithm is a straightforward implementation of balloons [Cohen91], which are snakes [Kass88] whose tendency is to grow rather than shrink. An initialization scheme is also developed to initialize the balloon based on the location of the treetop. While the core algorithm is the snake algorithm, the external forces are governed by the data in both the orthorectified mosaic *and* the corresponding DEM. This is new ground in the use of balloons/snakes and in tree-crown segmentation, and it works quite well.

Balloon initialization requires the location of the treetop and a parameter specifying the radius of a circle associated with the smallest expected tree in the image. The balloon is then initialized as a set of snaxels evenly spaced around a circle of the radius specified, centered at the treetop location.

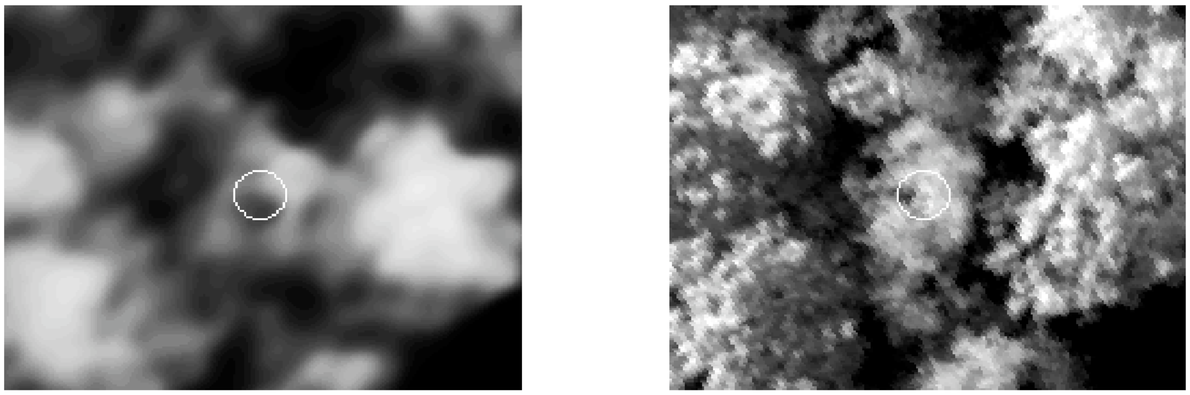


***Figure IV.B.1**. Example initialization overlaid on the DEM (left) and orthoimage (right). In the DEM, higher elevations appear brighter, with lower elevations darker. Note that the DEM contrast has been enhanced. This circle is centered on a user's mouse-click. The user was attempting to click near the "center" of the tree-crown*

A balloon is a closed curve represented as a set of points, "snaxels," which lie on the curve. The snake is a simulation of a set of physical points with mass (represented as snaxels) joined by (implicit) springs. The parameters of the snake govern how the points tend to move in relation to each other and how sharply the curve can bend (internal forces). They also describe the relative strength of the external forces, which depend on the data being segmented.

The energy functional of a snake or balloon, as originally proposed by Kass, Witkin, and Terzopoulos [Kass88], and as used in this work, follows:

$$\begin{aligned} E^{*}_{snake} &= \int_0^1 E_{snake}(v(s))\,ds \\ &= \int_0^1 E_{\text{int}}(v(s)) + E_{image}(v(s)) \end{aligned} \qquad (1)$$

where v(s) = (x(s), y(s)) is a parametric representation of the snake, E_{int} represents the internal energy, and E_{image} is the external energy. Kass, et al. proposed a third energy term, E_{con}, which is not used here, to allow other external forces to influence the curve, such as a spring connected on one end to the snake and on the other end to a fixed anchor position.

Internal Energy

The internal energy, which governs the shape of the snake, depends, not surprisingly, on the first and second derivatives of the curve. The internal energy term used follows:

$$E_{\text{int}} = \frac{(\alpha\,|v_s(s)|^2 + \beta\,|v_{ss}(s)|}{2} \qquad (2)$$

where α is the weight on the first-order term, and β weights the second-order term. By definition, the balloon force adds another term to the internal energy term, but we simply implemented it as an external force. The balloon force is produced by vector addition of the unit normal vector to the curve at the current position, and the vector pointing away from the nearest neighboring snaxel. This second vector helps keep the snaxels evenly spaced around the curve. The balloon force is below:

$$E_{balloon} = \frac{\hat{\boldsymbol{n}}(s) + }{2} \qquad (3)$$

where $\vec{r}$ is the normalized vector pointing away (repelling) from the nearest neighboring snaxel.

Clearly, the outward unit normal to the curve, $\hat{\boldsymbol{n}}(s)$ causes the balloon to expand.

External Energy

The external energy for the balloons in this work depends on both the image data and the elevation data. There are three external energy terms used in this implementation. First, as proposed by Kass, et al. [Kass88], we use the image intensity gradient as a force. The intensity energy term is simply the intensity gradient of the image at any point. Another energy used is the elevation according to the DEM. The last energy term is the simple image intensity. This moves the balloon toward darker regions of the image rather than brighter spots, which are likely to be treetops according to our assumptions (see subsection titled "Assumptions" in Section II above). Darker spots are more likely to be regions of shadow or space between trees. Since this dark-seeking energy is paired with the image intensity gradient, the tendency for the balloon to expand to encompass large areas of shadow should be reduced. Therefore, the external energy is below:

$$\begin{aligned} E_{image} = {} & w_1 \left| \nabla \left(G_\sigma (x, y) * I(x, y) \right) \right| \\ & + w_2 E(x, y) + w_3 I(x, y) \end{aligned} \qquad (4)$$

where G_σ is a Gaussian smoothing kernel, I(x,y) is the image intensity at point (x,y), and E(x,y) is the elevation at point (x,y). The weights on these three energy terms are parameters w_1, w_2, and w_3.

Iteration and Convergence

Snakes and balloons move iteratively. That is, the contour moves, the forces are re-evaluated, and the contour moves again. This continues until either a user ends the iterations, or, in a more automatic situation, convergence criteria are met. The latter strategy is used in this work.

To understand how the iterations proceed, recall that the balloon is a simulation of physical weights joined by springs. The goal is for the balloon to "slide down," surrounding a peak in energy, as influenced by the external forces. One of the forces is the elevation. Much like gravity influencing the simulated system, this force causes the weights, or snaxels, to move toward lower elevations. The other external forces function similarly, and they may work together with or counteract each other. The forces are determined by the underlying data (that is, the DEM and orthoimage), and are weighted by the weighting parameters. The energy of the snake is then computed according to equation 1. The snake moves, and the process begins again, stopping when convergence is reached.

There are several proposed convergence criteria in the literature. Trucco and Verri [Trucco98] present a test based on the proportion of snaxels that move in a each iteration. In conventional snake or balloon implementations, every snaxel moves in every iteration, though often the movements are by less than the width (or even half the width) of a pixel. The implementation used in the above mentioned book is not the variational calculus based approach used in the current work, but is based on the greedy implementation. This method requires snaxels to be at discrete locations and to move by whole-pixel distances in each iteration. It also uses different internal energy terms. This convergence criterion is not appropriate for implementations based on the variational calculus because all snaxels move (perhaps very small distances) in each iteration. Another approach is to apply a threshold to the minimum distance moved by any snaxel. That is, compute the displacement of each snaxel in the iteration, take the minimum of these displacements, and compare this value to a threshold, potentially stopping iterations. This approach can prematurely halt iterations because it is highly susceptible to outlier iterations in which one point may move very little while the rest of the points are moving quickly. Instead, this work looks at the mean displacement of all snaxels in the balloon. Then, this quantity is compared to a threshold. When the average movement drops below the threshold, the contour is determined to have converged, and iteration stops. See *Figure IV.B.2* for an illustration of this process.

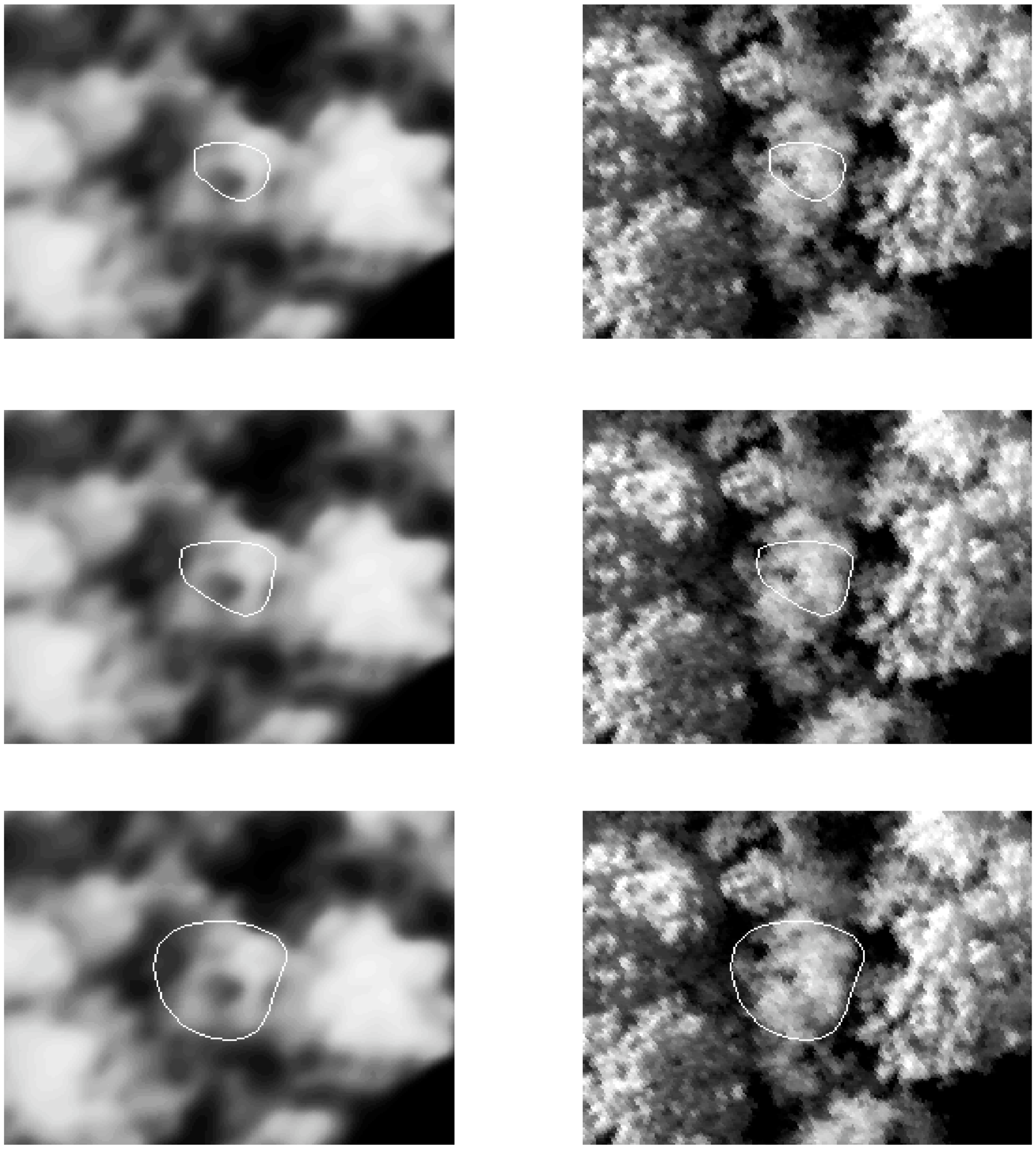


Figure IV.B.2. *DEM (top left) and orthoimage (top right) with balloon from Figure IV.B.1 after 2000 iterations . DEM (mid left) and orthoimage (mid right) with balloon from Figure IV.B.1 after 4000 iterations. DEM (lower left) and orthoimage (lower right) with balloon from Figure IV.B.1 after convergence (around 12,000 iterations, or about 3 seconds on a PIII 930 MHz PC). Note that the DEM contrast has been enhanced.*

IV.C. Automating the Segmentation Process

The first step in automatic segmentation is finding treetops. This process is described above in subsection A. The process described is geared to producing slightly more treetop candidates than there are trees in the scene. This increases the likelihood that all valid trees will be identified, but will leave some extraneous treetops that should be discarded in the process of segmenting the trees.

To segment the trees, the tree-crown segmentation algorithm using balloons is initialized as a circle surrounding the treetop location (as described in subsection B, above). The balloon then moves as influenced by its internal and external forces until it converges. This process takes place at each treetop identified by the treetop finding algorithm. When the region identified by a tree-crown segmentation includes another, as yet unprocessed treetop (as determined by the treetop detection algorithm), this treetop is discarded, and will not be used to initialize a balloon for additional tree-crown segmentation. To yield a reproducible result, the treetops are processed in the order of decreasing elevation.

This process may encounter problems. If the detected treetops are of low quality, then segmentations will also be of low quality. Some segmentations may take very odd shapes, especially if the treetop location is poorly placed. If the treetop is placed in an area of low elevation and dark intensity, the gradient-seeking force will drive it toward edges of other trees, so it will expand, but will form an odd shape. The shape is directed toward smoothly rounded contours by the internal forces. However, if the edges and valleys are strong enough, then these internal forces can be overpowered by the external ones. When this happens on a treetop, the segmentation's shape usually make sense because trees are generally round. When it happens in an area of shadow, however, this creates problems. Also, if the data aren't blurred enough, then far too many treetops will be identified, and tree-crowns will be segmented as smaller than appropriate because "bumps" in the data could cause the balloon to stop growing prematurely. Significant blurring is needed. In general, the Gaussian blurring kernel should be a square (or circle) with side length (or diameter) at least as big as the minimum expected tree-crown radius. Using a kernel side length that is larger than this, perhaps up to as large as the minimum expected tree-crown diameter will likely provide better results.

Validation

To validate the segmentation, some quick checks are made. In fitting with our assumptions, trees should appear brighter near the treetop than at the tree-crown boundary. Also, trees should be taller at the treetop (by definition) than on the boundary of the tree-crown. However, sometimes noise or holes in the data exist after, or as a result of, the processes of DEM extraction and orthorectification, so tree-crowns are only discarded if they fail to meet *both* of these tests. Further, if a tree is too small, it is discarded (smaller area than a circle with radius specified as minimum radius). Another check, which is perhaps the most difficult to formalize, attempts to discard trees that are too oddly shaped. Due to the assumption of roughly circular or elliptical tree-crowns within a loosely definable range of sizes, we can look at the ratio of the perimeter length of the tree-crown to its area. If this ratio is too large, then the shape is determined to be too irregular, and the segmentation is discarded.

Storage of Segmentations

Segmentations are stored in a simple ASCII text-based file format. Each tree-crown is stored as a polygon. The x- and y-coordinates of each polygon vertex are stored on a line. Each polygon is separated by a blank line in the text.

V. Experimental Results

To evaluate the accuracy of the algorithm developed in this work, we acquired some hand-segmented data. We chose this because hand segmentation is a common strategy for performing tree-crown segmentation in practice. Also, one of the first sanity checks performed on any algorithm is whether or not the segmentations look correct or make sense to a human observer. While it is nearly impossible to get 100% correct "ground truth" for this problem, humans are widely agreed to be better at delineating tree-crowns than algorithms at this point. Finally, since no other algorithms are known that segment trees using both intensity and depth information, comparison with other algorithms is not ideal as a validation activity for this algorithm. Humans are clearly able to perceive both depth and intensity simultaneously, so this is the best comparison available.

Human tree-crown segmentation from aerial images, as a strategy for forest inventory, has its drawbacks, however. First, different humans segment trees for different purposes and have different strategies. Some are segmenting with the sole goal of computing biomass. Others want to get correct tree counts. Some purposes are more suited toward generously sized segmentations while others are geared more for conservatively sized segmentations. This will yield highly different segmentations unless the notion of a "correct" segmentation is established in some form. Further, two humans with the "same" goal in mind will still produce different results due to individual differences and the frequent subjectivity of the problem. Perhaps most importantly, human segmentation is tedious, time consuming, and expensive. It yields varying segmentation quality over time for the same individual. One result of this is that we were able to retrieve a relatively small number of human segmentations in our evaluation phase due to time constraints. Upon examination of these segmentations, too, the quality is suspect in that it appears that many small trees were missed, and many trees were under-segmented (meaning that their crowns occupy more area than that reflected by their segmentation). This is an opinion held by several computer science and forestry researchers who have seen the data.

V.A. Hand Segmentation Data Collection

Below is Figure V.A.1 which shows the hand segmentations performed by a forestry expert (in fact, a coworker of the primary author of [Slaymaker99]), overlaid on the corresponding orthoimage. Note the irregularity of the shapes (see Figure V.A.2). Some segmentations are more detailed than others; some are more jagged than others. Note that some bright regions that appear to possibly be trees have been missed. This will also be a property of the automatically segmented data from the algorithm.

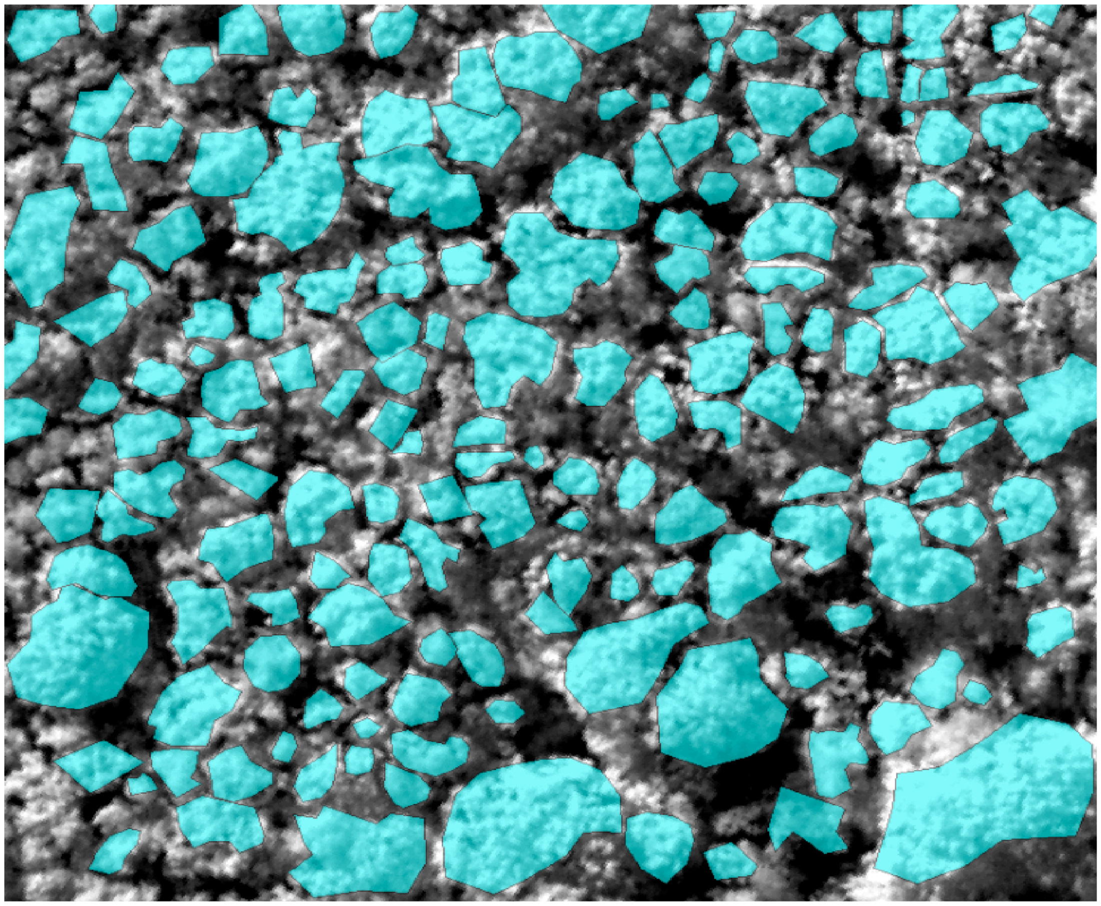


Figure V.A.1. *Hand segmentations overlaid on orthoimage. The hand segmentations are represented as closed polygons which are transparently shaded in a teal color.*

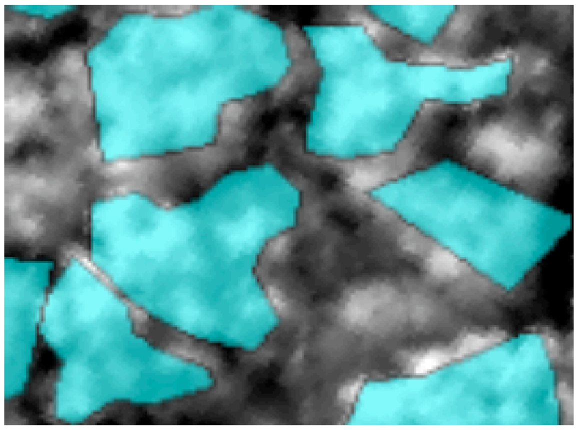


Figure V.A.2. *Irregularly shaped hand segmentations from left central area of Figure V.A.1. This has been enlarged to show the detail.*

These data were collected over a period of about two hours. The process was performed using the Imagine OrthoBASE version 8.6 and Stereo Analyst version 8.7 software packages produced by Leica Geosystems. To allow stereo viewing, NuVision 60GX Stereoscopic Wireless Glasses were used with the display set to a refresh rate of 120 Hz. To segment an individual crown, the user had to click a button indicating the beginning of a new polygon region of interest and set the mouse-cursor to an appropriate elevation. An appropriate elevation is one that is visibly similar to the elevation of the tree-crown. Next, the user had to click points to form the vertices of a polygon to enclose the crown of the tree. Finally, a double-click ended and closed the tree-crown polygon. Points and polygons could be (and were) edited or deleted if they were misplaced. Once all the segmentations were "complete" as determined by the user, they were stored into a standard shape file (.SHP file extension). Also, for convenience, they were exported to an ASCII file format listing the GPS coordinates of each vertex on a separate line, with a blank line and a header separating polygons.

The time required to perform hand segmentations could possibly be increased with a better user interface. However, even sophisticated user interfaces, such as the one presented in [Gong02] require the user to modify parameters to force a model to fit each of the trees. This seems likely to take even more time than manually clicking polygon vertices.

V.B. Automatic Segmentation Data Collection

The data formats required by the software implementing the algorithm presented in this work are different than those required by the GIS and stereoscopic presentation systems used in the hand segmentations. As a result, data generation and preprocessing must occur before using the software developed to perform automatic tree-crown segmentation. First, a DEM must be generated. The automatic segmentations presented here use a DEM generated by the Terrest 3D Reconstruction

System developed in part at the University of Massachusetts Amherst, which has proven to be accurate and robust. However, to use the tree-crown segmentation software, all that is needed is a registered DEM to go with the orthoimage. Once the DEM was obtained in this data collection, it and the orthoimage were smoothed using a 19-by-19 element Gaussian kernel. This allows the balloons to traverse a more smooth surface to ignore some of the intra-tree-crown brightness and elevation variations that could prematurely “snag” the contour as it expands, causing it to segment an area that is smaller than desirable.

Once this initial data generation and preprocessing is complete, the algorithm needs the parameters to specify how various forces relate and other behaviors progress. See Appendix A for a more detailed description of the parameters' functions. Here is a table which specifies the parameters and their values.

Parameter Name	***Value***	***Parameter Name***	***Value***
NumPoints	32	Alpha	1
Beta	24	InitRadius	12
Grad Weight	-5.25	Grad Blur Radius	19
Elev Weight	6	Balloon Weight	12
Intensity	1.25	Convergence	1

Table V.B.1. *Parameter values for automatic segmentation data collection.*

Using the InitRadius parameter, which is to specify the smallest expected tree-crown radius, the treetop detection algorithm identified 260 candidate treetops in the scene. Below, in Figure V.B.1 we see these treetops overlaid on the blurred orthoimage.

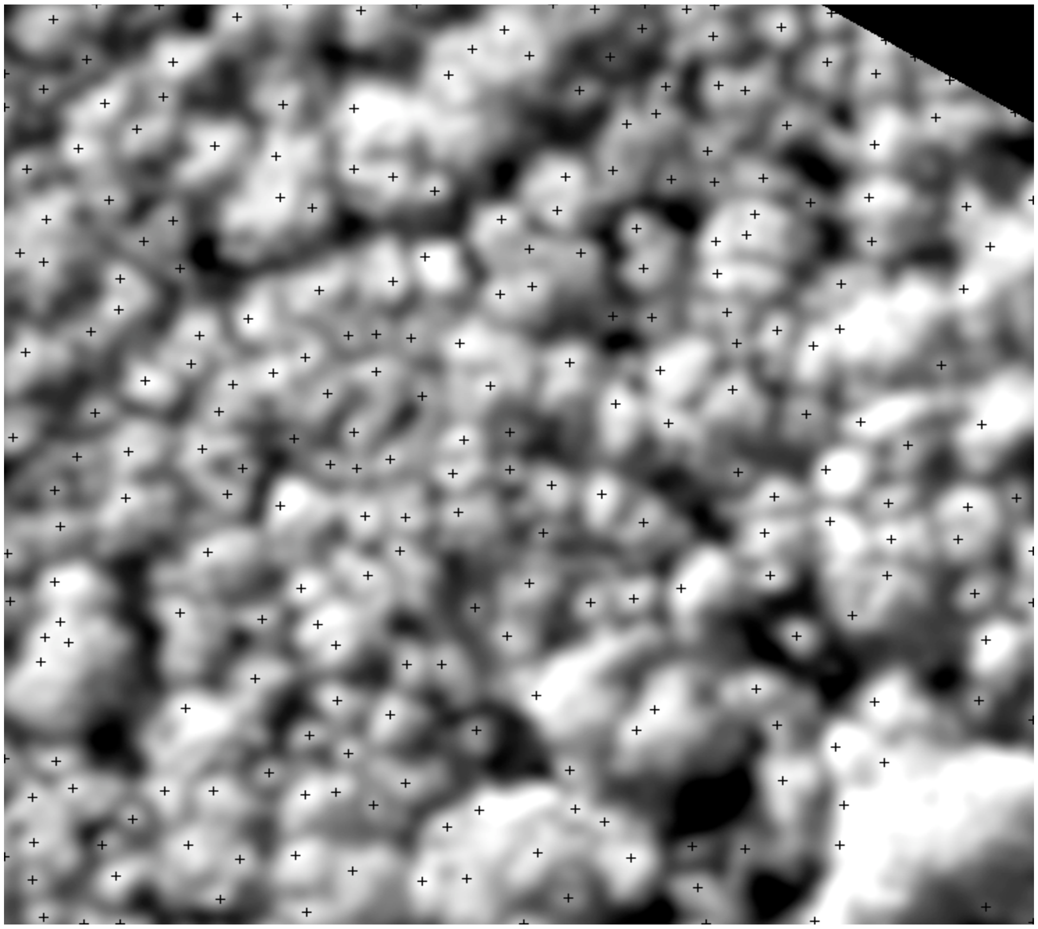


***Figure V.B.1**. Treetops overlaid as black crosshairs on the blurred orthoimage. This is the same area as presented in Figure V.A.1.*

Notice that no treetops were detected near the center of the large clump of trees in the lower right corner of the image. This is perhaps due to the loss of texture information in this area of saturated intensity which does not yield a maximum to be paired with any elevation maxima.

The final step in the data collection was running the tree-crown segmentation algorithm automatically at each of these treetops. This resulted in a final segmentation of 224 tree-crowns. This means that 36 detected treetops were discarded because they were overrun by other trees or because their corresponding segmentations were too irregular in shape, appearance, or elevation. The whole process, from start to finish, required 109 minutes on a Pentium III processor at 930 MHz running

Linux kernel version 2.4.20 and version 1.4.1_01 of the JRE (java). Figure V.B.2 shows the resulting segmentations.

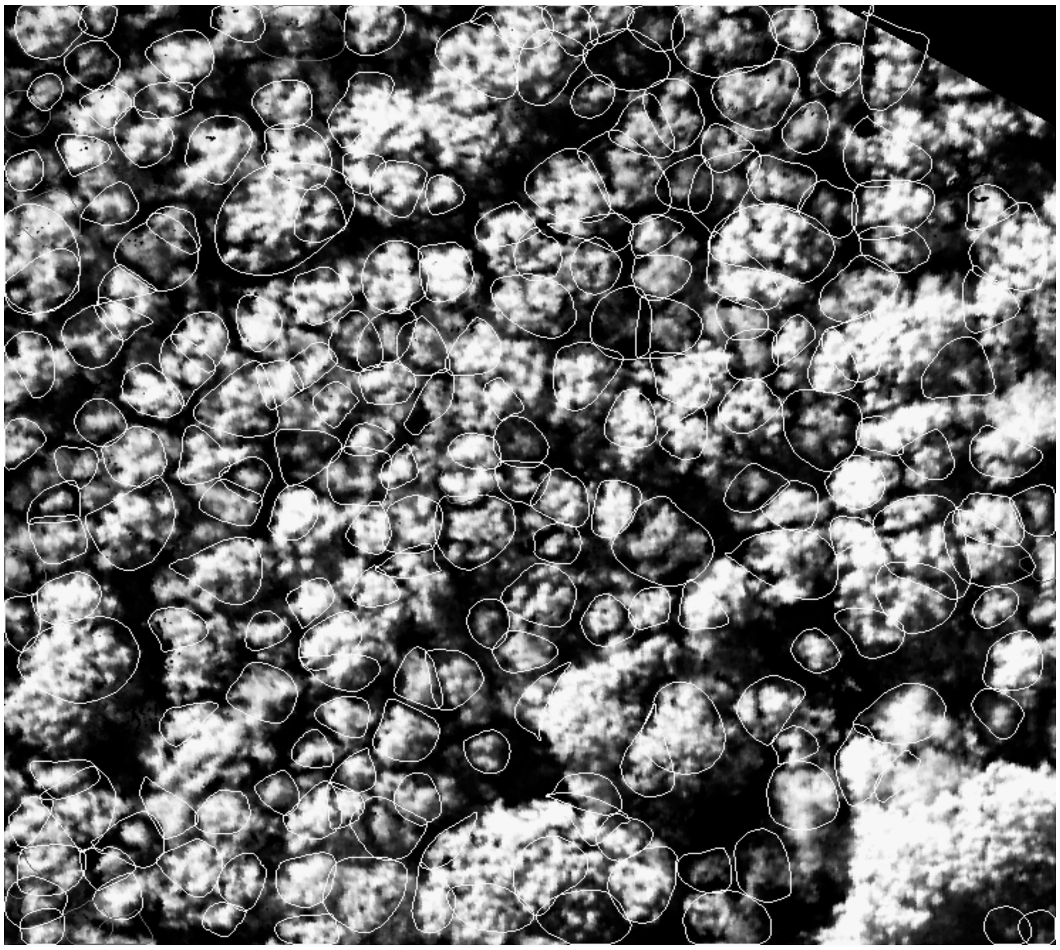


Figure V.B.2. *Segmentations overlaid as white closed contours on the orthoimage.*

The reader should note that the triangle of missing data in the upper right corner of the scene used with the algorithm is an artifact. Hand segmentations of tree-crowns in this region, and its corresponding area, were discarded from the comparison calculations presented next.

V.C. Comparison of Hand and Automatic Segmentations

Table V.C.1 summarizes the comparison of these two methods of tree-crown segmentation.

	Hand Segmentations	***Automatic Segmentations***
% of Total Area Marked as Tree	42.55	55.64
Number of Trees	161	224

Table V.C.1. *Comparison of hand segmentations to automatic segmentations.*

The human segmentation amounted to a smaller portion of the total area, than the automatic algorithm. The automatic algorithm also segmented about 40% more individual trees than the human. As already stated, the quality of the hand segmentations is in question. During hand segmentation, fatigue can begin to reduce the quality of segmentations and increase the likelihood of skipping smaller trees. Further, the algorithm here tends to include somewhat more shadow than is ideal in the segmentations of some trees. The automatic algorithm is also likely to break larger clumps of canopy area into more trees than the human did.

More hand-segmented data is needed, possibly from independent experts. In this comparison, parameters to the automatic algorithm were tuned and established before viewing the human segmentations. As with the individual differences in human segmentations mentioned above, there will be individual differences in tuning the parameters. If so desired, with further parameter improvements, the performance of the algorithm could be made to more closely mimic the hand segmentations, because the “ideal” segmentation being pursued by the algorithm with the parameter settings used was clearly different than what the human performing hand segmentations was seeking. It should be noted that the human performing the segmentations had not seen the segmentations produced by the automatic algorithm, and, again, the parameters to the automatic algorithm were established before seeing any segmentations by the human.

These results allow for an indirect comparison to another published automatic tree-crown segmentation algorithm. This comparison does not present a strong case, however, because we need more hand segmentations and more agreement about whether or not they are close to correct. Wang, et al. [Wang04] present an automatic algorithm which uses a more sophisticated treetop detection scheme in a unified framework with marker-controlled watershed segmentation. This algorithm does not use depth information at all (the images are projected into a surface of constant elevation). Also, and very importantly, the {Wang04] algorithm was run on a different data set. These researchers acquired ground truth by averaging across multiple human hand segmentation sets, and were operating on images with more, smaller-appearing trees. This comparison is difficult, too, because we do not know the dependence of their algorithm on scale, or whether the various smoothing and scale parameters were tuned before or after human segmentations. As a result of these complications, this comparison will only serve to tell us whether the algorithm presented in this current work is producing reasonable results, not which algorithm is better. Table V.C.2 presents their results in similar way to that of V.C.1.

	Hand Segmentations	***Automatic Segmentations***
% of Total Area Marked as Tree	29	41
Number of Trees	957	1122

Table V.C.2. *Comparison of hand segmentation to automatic segmentation as presented in [Wang04].*

Their algorithm also identified more area as part of a tree-crown than their hand segmentations, and it identified more individual trees than were found by their hand segmentations. Table V.C.3 presents a comparison of how far these were overestimated.

	Wang, et al.	***Us***
% More area than by hand	41	**30**
% More trees than by hand	**17**	40

Table V.C.3. *Comparison of Wang's errors to our errors.*

Table V.C.3 shows that the area segmented by our algorithm was closer to the area segmented by a human hand segmentation on our data than theirs was on their data. That is, the algorithm presented in [Wang04] segmented 41% more area than the average segmented by their human segmenters whereas ours segmented 30% more than our human segmenter did. Conversely, they identified a number of trees that is closer to the number identified in the human segmentation on their data than we did for our data. To be clear, the [Wang04] algorithm found 17% more individual trees than the average found by their human segmenters, and our algorithm found 40% more trees than our human segmenter. Again, this is only a very indirect comparison.

VI. Conclusions

Automatic tree-crown segmentation can be accomplished by using snakes and a treetop detection algorithm on both the DEM and the orthoimage in unison. This strategy yields reasonable numbers of trees, reasonable individual segmentations, and reasonable estimates of total tree-crown area for a data set. As this is the first approach of its kind, there is room for further research to increase the accuracy of the segmentations, the number of trees identified, and the speed of computation. This approach is faster than human segmentations from start to finish, but can also be done off-line, and therefore requires extremely low amounts of human supervision.

There is clearly not enough hand-segmented data for a good comparison to other algorithms. To truly compare this algorithm to other algorithms, a standardized data set with corresponding ground truth is needed.

VII. Notes for Future Research

To improve on the results achieved in this work, several avenues exist. Again, the general feeling is that more hand segmented data is needed because probably closer to 70-80% of the canopy area is occupied by tree-crowns rather than the relatively low number found by the human in these results. This means, however, that our algorithm also underestimated the area. There are several ways to attack this. One is to allow more treetops to be detected. One of the more obvious ways to do this is to allow isolated peaks in elevation to be marked as treetops like the isolated intensity peaks are marked. This is probably the best approach because, while the total area identified as part of a tree is less than desired, the individual tree-crown segmentations produced by this algorithm appear to be somewhat larger than ideal, so relaxing the parameters to allow for larger individual segmentations is not the best route. Because of this, the best way to get more tree-crown area is to find more trees. This could perhaps be done in a second pass through the data after the algorithm has completed, as explained below.

Another way to tackle this problem is to do post-processing after the algorithm above has completed. This could involve masking out the segmentations produced by this algorithm and analyzing what's left. Perhaps there are bright circular objects that didn't have enough depth information to pass the tree-validity tests. Perhaps there were high points that weren't bright enough to appear as trees. Performing thresholding and connected components analysis on the remaining elevation and intensity information may shed some light on whether or not this is a good path to follow. Thresholding the elevation values is far more tricky than intensity values, however, because of the existence of hills and other elevation changes due to causes other than variations in tree height. Perhaps some sort of neighborhood based, relative threshold would provide some results, but this is probably not necessary because easier methods probably exist. Preliminary thresholding of the orthoimage looks like it would indicate that most of the trees missed by the first pass of the automatic algorithm could be accurately classified as trees by the thresholding without introducing areas of shadow or ground cover.

Second, to reduce the tendency to segment a slightly larger region than ideal for many tree-crowns, the best solution is probably parameter optimization. This can be done ad hoc (by reducing the balloon or intensity forces, or increasing the gradient force), or in a more organized way. One way would be to let the balloon converge, then decrease the balloon force and the intensity force, and finally let the balloon converge again. This would allow the balloon to more easily shrink somewhat, if necessary, to move toward strong boundaries. A problem this introduces, however, is a higher potential for loops in the contour. To solve this problem, a new or highly altered version of the snakes/balloons algorithm would need to be used. See the literature review in section III for some papers that discuss this.

Third, depending on what the segmentations will be used to do, overlap of tree-crowns may or may not be useful. In some ways, the overlap is often more accurate than having completely disjoint tree-crowns. This is because trees to grow into one another and taller trees do occlude parts of lower trees. For the purpose of training a classifier, overlap is probably not a problem. If overlap is a problem, however, then there are some ways to deal with it. One way is to simply merge the two (or more) overlapping tree-crowns into the largest tree-crown. Another way is to give exactly half of the area to one crown, and the other half to the other. If one tree-crown is completely inside another, then this smaller tree-crown should most likely be discarded. Yet another way to handle overlap is to assign only the overlapping region to the larger tree (under the assumption that it is probably taller and is therefore occluding some of the smaller tree(s)), and then apply a threshold to determine if the remainder of the smaller tree is large enough to be considered its own tree, or if this remainder should be discarded or merged into the larger tree.

Finally, this strategy is dependent on the size of the tree-crowns. That is, performance degrades as tree-crown diameter increases. As a parameter to the algorithm, the minimum expected tree radius largely determines this. To reduce this dependency, one may attempt to segment trees at different scales. It is not clear exactly how to do this, however. Perhaps one strategy would be to blur the data with a larger Gaussian window to start, perform segmentation, reduce the blurring, and perform segmentation again on the areas not segmented first, repeating this iteratively. This may not work, however, because peaks in highly blurred data do not necessarily correspond to larger trees than peaks in less-highly blurred data. Another strategy would be to start with highly blurred data, allow the snakes to converge, then iteratively reduce the blurring on the data, and allow them to converge again, as proposed by Tang and Acton [Tang04] for tracking vessel boundaries in intravital microscopic imagery. This would yield better performance on larger trees and is likely to retain the current performance on smaller trees. The only drawback is that fewer trees would be identified. To solve this, new treetops should be identified at each stage of reduced blurring.

References

Chuang01: Chuang, C. and Lie, W., "Automatic Snake Contours for the Segmentation of Multiple Objects," *The 2001 IEEE Symposium on Circuits and Systems*, Vol. 2, pp. 389-392, 2001

Cohen91: Cohen, L., "On Active Contour Models and Balloons," *Computer Vision, Graphics, and Image Processing: Image Understanding*, Vol. 53, No. 2, pp. 211-218, 1991

Collet94: Collet, C., and Thourel, P., "Active Contour Models for Infrared Cloudy Shapes Segmentation," *OCEANS '94: 'Oceans Engineering for Today's Technology & Tomorrow's Preservation'*, 2, pp. 444-448, 1994

Daley98: Daley, N., Burnet, C., Wulder, M., Olaf Niemann, K., and Goodenough, D., "Comparison of Fixed-Size and Variable-Sized Windows for the Estimation of Tree Crown Position," *Proceedings of International Geoscience & Remote Sensing Symposium*, , pp. 1323-1325, 1998

Gong02: Gong, P., Sheng, Y., and Biging, G., "3D Model-Based Tree Measurement from High-Resolution Aerial Imagery," *Photogrammetric Engineering & Remote Sensing*, Vol. 68, No.11, pp. 1203-1212, 2002

Ji99: Ji, L., and Yan, H., "Loop-free Snakes for Image Segmentation," *1999 International Conference on Image Processing*, Vol. 3, pp. 193-197, 1999

Kass88: Kass, M., Witkin, A., and Terzopoulos, D., "Snakes: Active Contour Models," *International Journal of Computer Vision*, Vol. 1, pp. 321-331, 1988

Lam94: Lam, K. and Yah, H., "Fast Greedy Algorithm for Active Contours," *Electronics Letters*, Vol. 30, No. 1, pp. 21-23, 1994

Markovic03: Markovic, D., and Gelautz, M., "Video Object Segmentation Using Stereo-Derived Depth Maps," *27th Workshop of the Austrian Association for Pattern Recognition*, , pp. 197-204, 2003

Slaymaker99: Slaymaker, D., Schultz, H., Hanson, A., Riseman, E., Holmes, C., Powell, M., and Delaney, M., "Calculating Forest Biomass With Small Format Aerial Photography, Videography And A Profiling Laser," *ASPRS Proceedings of the 17th Biennial Workshop on Color Photography and Videography in Resource Assesment*, , , 1999

Song03: Song, C., and Woodcock, C., "Estimating Tree Crown Size from Multiresolution Remotely Sensed Imagery," *Photogrammetric Engineering & Remote Sensing*, Vol. 69, No. 11, pp. 1263-1270, 2003

Tang04: Tang, J., and Acton, S., "Vessel Boundary Tracking for Intravital Microscopy Via Multiscale Gradient Vector Flow Snakes," *IEEE Transactions on Biomedical Engineering*, Vol. 51, No. 2, pp. 316-324, 2004

Trucco98: Trucco, E. and Verri, A., Introductory Techniques for 3-D Computer Vision, ch. 5, 1998, Prentice Hall, ISBN: 0-13-261108-2

Wang04: Wang, L., Gong, P., and Biging, G., "Individual Tree-Crown Delineation and Treetop Detection in High-Spatial-Resolution Aerial Imagery," *Photogrammetric Engineering & Remote Sensing*, Vol. 70, No. 3, pp. 351-357, 2004

Wang98: Wang, H., and Ghosh, B., "Boundary Finding With New Balloon Models," *Proceedings of the 20th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, Vol. 20, pp. 686-689, 1998

Whitten93: Whitten, G., "Scale Space Tracking and Deformable Sheet Models for Computational Vision," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 15, No. 7, pp. 697-706, 1993

Williams92: Williams, D. and Shah, M., "A Fast Algorithm for Active Contours and Curvature Estimation," *CVGIP: Image Understanding*, Vol. 55, No.1, pp. 14-26, 1992

Appendix A: Instructions for Using the Tree-Crown Segmentation Program

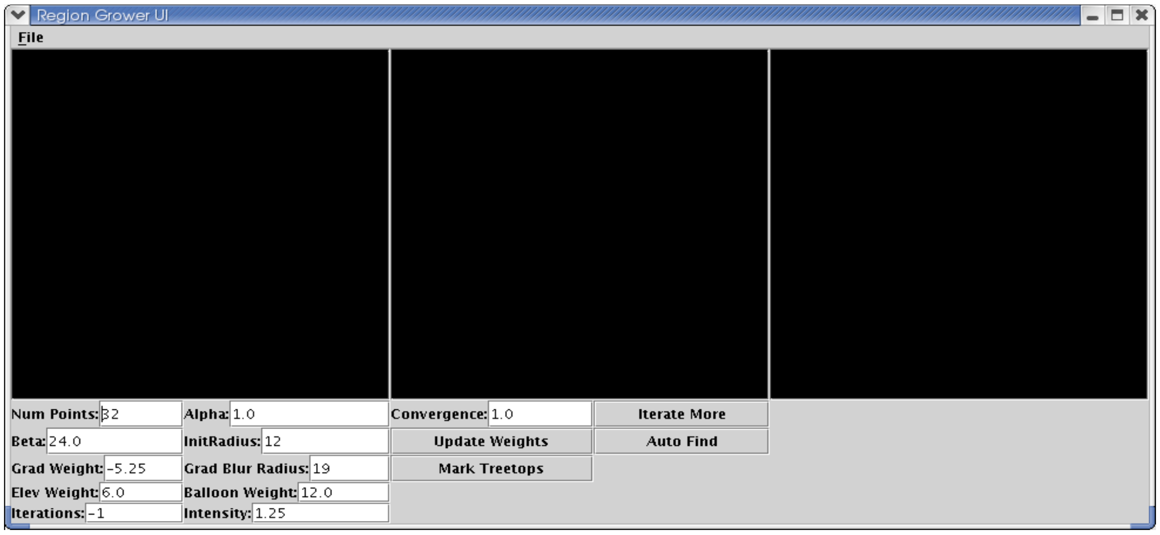
Step 1: Running the Tree-Crown Segmenter

The tree-crown segmenter is written in Java using version 1.4.1 of the JDK. So to run it, Java must be installed. Once java in installed and in the path, the tree-crown segmenter can be run as:

```
java -Xmx300m -Xms300m RegrowUI
```

The options -Xmx300m and -Xms300m tell the JVM to request a minimum of 300 MB of memory for use instead of the default amount (~64MB?). This is really dependent on the size of the data being processed. If out of memory errors occur, then you may want to increase the amount of memory requested by the JVM.

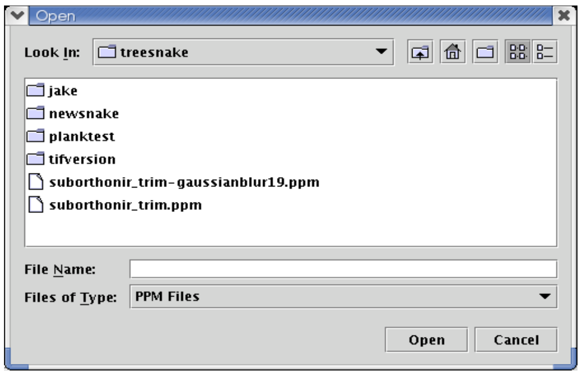
Once the command above is typed, the following screen appears:



Num Points: 32
Alpha: 1.0
Convergence: 1.0
Beta: 24.0
InitRadius: 12
Grad Weight: -5.25
Grad Blur Radius: 19
Elev Weight: 6.0
Balloon Weight: 12.0
Iterations: -1
Intensity: 1.25

Step2: Opening the Data

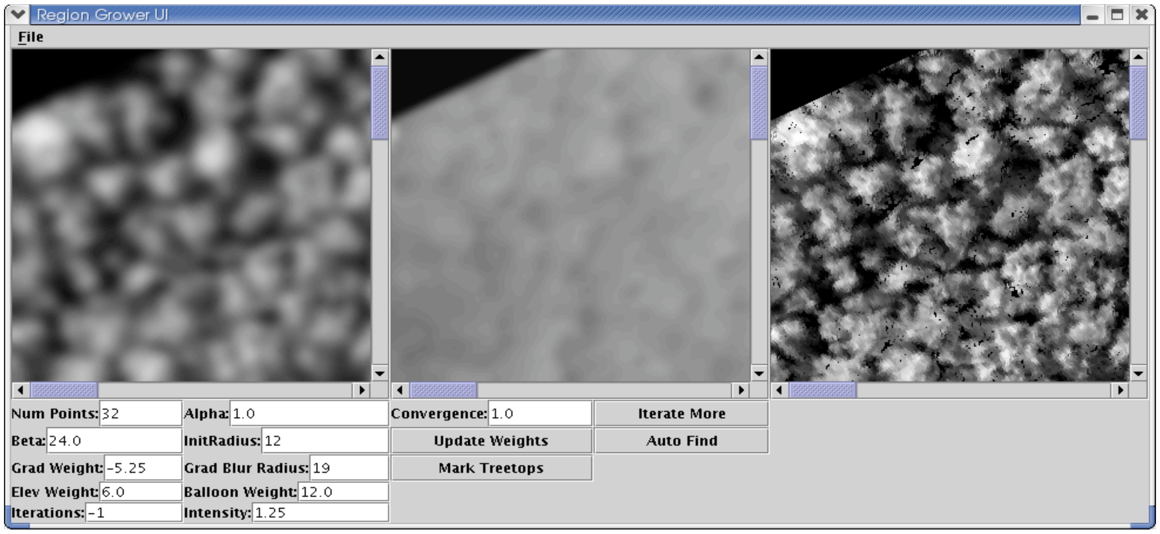
The tree-crown segmenter operates on two files: the blurred DEM and orthoimage. However, results are more informative if they are displayed on the non-blurred orthoimage. The tree-crown segmenter doesn't perform the pre-blurring of its input images. As a result, three files must be opened. To start, click File and then Open. The following screen appears:



Navigate to the folder containing the blurred version of the orthoimage and select it first, then click "Open". Notice that it must be in the PPM file format (you can convert from other types to the PPM type using the command-line utility 'convert' or with your favorite image manipulation program, such as the GIMP).

The next screen that appears after clicking "Open" is another Open dialog. Notice that this time, however, the Files of Type selector says "Digital Elevation Map (DEM) Files". This is to indicate that you should select the DEM. The DEM file format is simply a raw float array. The DEM filename should end in ".dem", and everything before ".dem" should be the same as the blurred orthoimage filename (up until the ".ppm"). The dimensions are assumed to be the same as the orthoimage. Navigate to and select the DEM. Click "Open".

The next screen that appears is a third Open dialog. Now is the time to open the non-blurred (original) orthoimage. Navigate to it, select it, and then click "Open". This ends the "Opening the Data" section, with the main window displaying the blurred (left) and non-blurred (right) orthoimage, and the DEM (center) as below:



Num Points: 32 Alpha: 1.0 Convergence: 1.0
Beta: 24.0 InitRadius: 12
Grad Weight: -5.25 Grad Blur Radius: 19
Elev Weight: 6.0 Balloon Weight: 12.0
Iterations: -1 Intensity: 1.25

Step3: Setting Parameters

In the main window, there are many settings that appear below the image displays. Below is a brief explanation of each of the parameters. Following this explanation is a discussion of how to set them.

- NumPoints: This is the number of snaxels used to represent the contour (snake)
- Alpha: This is the weight put on the first-order (continuity) internal force.
- Beta: This is the weight put on the second-order (curvature) internal force.
- InitRadius: This is an especially important parameter. The number entered here should be the expected radius of a circle which tightly surrounds the smallest tree that is expected to appear in your data and that you are interested in segmenting. This parameter is used for searching for treetops and some intermediate data smoothing.

- Grad Weight: This is the weight applied to the external force component due to the intensity gradient of the orthoimage (blurred).
- Grad Blur Radius: This is poorly named. It is actually the width of the window used to blur when computing the intensity gradient force and the elevation force.
- Elev Weight: This is the weight applied to the external force component due to the intensity gradient of the orthoimage.
- Balloon Weight: This is the weight applied to the balloon (expansion) force.
- Iterations: This tells the snake how many times to iterate. A value of -1 here indicates that it should iterate until convergence (mean point displacement after 1000 iterations is less than value of "Convergence" parameter).
- Intensity: This is the weight applied to the intensity force.
- Convergence: This is a threshold for the mean point displacement after 1000 iterations. It is used to determine when to stop iterating the snake.

Some of the parameters are easy to set. The Num Points parameter should be larger to capture more detailed contours, while smaller values are adequate and faster for simpler contours. The InitRadius and Grad Blur Radius parameters are also relatively easy to set. To determine the InitRadius, just measure the width in pixels of a small tree using an image processing program. The Grad Blur Radius should be at least as large as the InitRadius parameter, but 1.5 times larger works well. The Iterations parameter should also be easy to set. If you set it too low, you can always perform more iterations by optionally changing the value of this parameter and clicking the "Iterate More" button. Also, if you want to simply iterate until convergence, then enter -1. Also, the Convergence parameter is one of the easier ones to set. A value of 1.0 works well here, but you can make this larger to stop sooner, or smaller to stop iterating later.

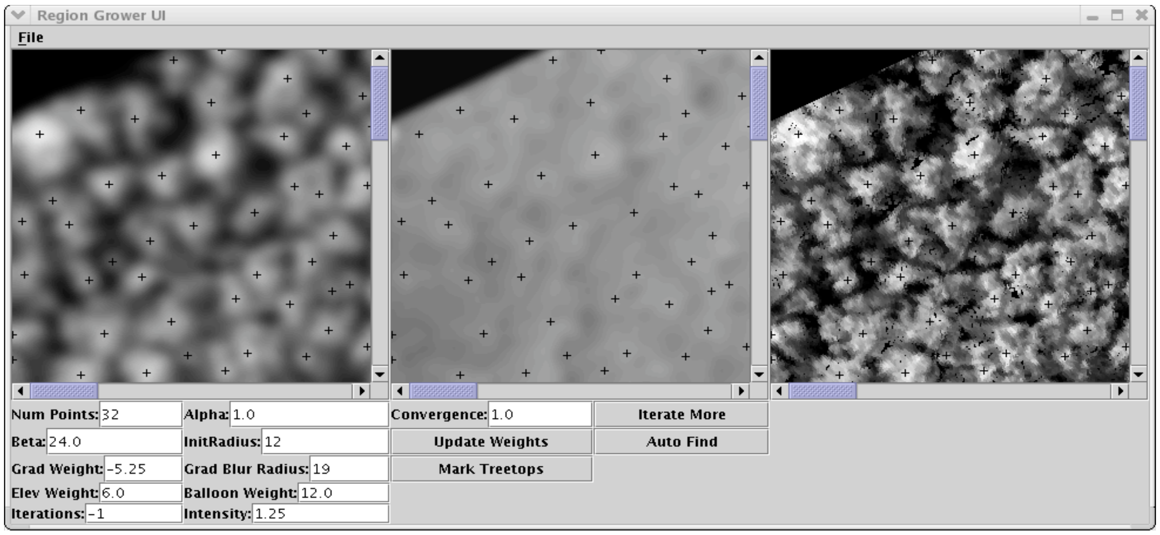
The weights on the internal and external snake forces are harder to tune. The values shown in the screen captures above work well together. For larger trees, the value of the alpha and beta parameters may need to increase. For complicated or detailed contours, beta may need to decrease. Grad Weight will also need to be tuned according to the data. Try by zeroing out the Elev Weight and Intensity parameters and find the best value of Grad Weight. Then, start tuning the Elev Weight parameter by finding the best value that works with the parameters chosen so far. You may play with reducing the Grad Weight as you tune the Elev Weight to get the best segmentations. Then, add the Intensity parameter to the mix, following a similar procedure. All the while, tune the Balloon Weight parameter so that the contour expands far enough, but not too far.

Step 4: Testing Parameters by Manual Treetop Identification

Trees can be segmented using the GUI (user interface) by issuing a single click on each of the treetops to be segmented. The first segmentation after setting the Grad Blur Radius parameter takes longer because of some pre-processing that occurs for the whole image. To test the setting of the InitRadius parameter, try setting Iterations to 0. Then, click a treetop (near the center of a tree-crown or on a bright spot in the DEM). A circle whose radius is the value of the InitRadius parameter is overlaid on the image centered at your click. If this circle appears too large or to small for the smallest tree you want to segment, then change InitRadius appropriately. Change Iterations back to -1 or whatever number of iterations you desire for the rest of the testing.

A good way to test many parameters at once is to iterate for a certain number of iterations without using the convergence detection supplied by the program. To do this, set Iterations to, say 3000. Then, click on a treetop. If the segmented region is too small, try clicking “Iterate More”. If the region appears to shrink after more iterations, or if it hasn't gotten any larger, try increasing the Balloon Weight parameter (or perhaps decreasing alpha). If it grows too quickly, try decreasing the Balloon Weight (or, perhaps increasing alpha). Continue clicking Iterate More until the snake movement slows considerably. If you noticed the snake slowed down when it got to about the right boundary, but then continued growing past it, you may want to slightly decrease the balloon force or increase the external component force that seems most appropriate to the particular boundary in question. If the segmentations are too smooth, try decreasing beta, or if they're too rough and sharply curving, try increasing beta. If the segmentations too often pass the boundary of the tree and extend into areas of dark shadow, try decreasing the Intensity weight and increasing the Grad Weight.

When the segmentations begin to appear reasonable, a final check is to test the treetop detection procedure which will be used in the automatic segmentations. To do this, click the “Mark Treetops” button. The process takes several minutes. If the resulting treetops look about like places where you would click if trying to segment the entire set of trees, then everything is fine. If not, then you may want to tweak the InitRadius parameter. If this still doesn't work, then you may want to try additional preprocessing steps, such as increasing blur window width, or median filtering. The results of clicking the Mark Treetops button should look something like the result below:



Num Points: 32
Alpha: 1.0
Convergence: 1.0
Beta: 24.0
InitRadius: 12
Grad Weight: -5.25
Grad Blur Radius: 19
Elev Weight: 6.0
Balloon Weight: 12.0
Iterations: -1
Intensity: 1.25

Step 5: Automatic Tree-Crown Segmentation

Once all the parameters have been set to your satisfaction, click the “Auto Find” button. This will segment all the trees. The result of all the segmentations will be overlaid on the three images displayed in the main window. It is possible to save a copy of this overlay image by clicking “File” then “Save”. First, you will need to enter a filename for the blurred orthoimage version, then the blurred DEM version, and finally the original orthoimage version of the overlay. Be sure not to overwrite your initial data files.

The segmentations are automatically stored in a file called “tree-crown-roi.txt” in the current directory. The segmentations are polygons. The file format is ASCII with the x- and y-coordinates of each vertex on its own line. Each polygon is separated by a blank line. To save segmentations from clicked treetops rather than automatic segmentations, click File -> Write ROI after each segmentation is complete. This will append the polygon to the “tree-crown-roi.txt” file.

Appendix B: Initial project description from proposal

This project involves analyzing aerial images taken of a forest, computing a digital elevation map (DEM) of the forest canopy, ortho-rectifying the images, constructing large photo-mosaics of the forest canopy, and then using this information to extract statistics such as counts, species, biomass, or other statistics describing the forest. This requires development of a reliable, robust methodology for undistorting images and producing accurate DEMs. Once a DEM is obtained, the images should be projected onto the DEM to produce a set of images which can be used to construct a large, geographically correct mosaic image of the forest. Finally, a major focus will be on developing a tree-crown segmentation algorithm so that trees can be identified for computing forest statistics.

Appendix C: Source code organization

There are three main classes. The user interface is implemented in the RegrowUI class. The snake is implemented in the Snake2D class. The SnakeSkin class takes input from the user interface and appropriately controls the snake and automatic segmentation process. The other classes help represent intermediate data, or intermediate operations on data. Javadocs are embedded in the code.