UNIVERSITY OF CAPE TOWN

DEPARTMENT OF COMPUTER SCIENCE

CS/IT Honours Final Paper 2020

Title: Per Pixel Height Map Generation of Tree Orchards

Author: Daniel Bowden

Project Abbreviation: PLANER

Supervisor(s): Assoc. Prof. Patrick Marais

Category	*Min*	*Max*	Chosen
Requirement Analysis and Design	*0*	*20*	
Theoretical Analysis	*0*	*25*	
Experiment Design and Execution	*0*	*20*	15
System Development and Implementation	*0*	*20*	20
Results, Findings and Conclusions	*10*	*20*	15
Aim Formulation and Background Work	*10*	*15*	10
Quality of Paper Writing and Presentation	*10*		10
Quality of Deliverables	*10*		10
Overall General Project Evaluation (*this section allowed only with motivation letter from supervisor*)	*0*	*10*	
Total marks		**80**	

Per Pixel Height Map Generation of Tree Orchards

Daniel Bowden
BWDDAN001@myuct.ac.za
University of Cape Town
Cape Town, South Africa

ABSTRACT

Determining the heights of trees from a heightmap is a useful tool for agricultural farmers. A heightmap is an image representing a collection of the height values distributed evenly across a landscape. The heights of trees in an on orchard can be estimated, given such a heightmap, through using image processing methods. This involves the removal of the ground plane values from height data of farmland to leave you with a heightmap containing height information for just the trees. One issue that arises when implementing this process is the lack of ground truth data sets containing the known heights of the landscape that a particular orchard of trees is grown upon. Without knowing the true height of the ground below a tree one cannot evaluate whether image processing methods estimating the height of a tree were accurate or not.

We explore methods to create synthetic digital heightmaps which are sufficiently complex to capture the variation present in a broad range of tree orchards. Using these synthetic heightmaps we can store all the sufficient data for the land and the trees produced in order to rigorously test methods used for estimating the height of the land and the trees. It was found during this research that simulating features of landscapes and generating heightmap landscapes proved an effective way of evaluating the various methods involved with the process of estimating the height of trees in a farm orchard. Various pitfalls and improvements to the methods themselves became apparent as testing was undertaken using the maps we generated.

CCS CONCEPTS

• Computing methodologies → Computer Graphics, Modelling and simulation

KEYWORDS

Digital Elevation Model, GIS, Heightmap

1. INTRODUCTION

Knowing the heights of trees is an important factor in managing tree farmland. Tree height information can be used to monitor growth metrics, areas in danger of soil erosion and keep farms within possible greening regulations [1]. Many farmers would benefit greatly from being able to track the height of their tree orchards, however manual collection of tree height data is a slow tedious process. Many farmers instead monitor their farmland with drones commonly referred to as UAVs (Unmanned Aerial Vehicle). drones can quickly and efficiently gather large amounts of imagery and GIS (Geographic Information System) data that is useful to farmers in analysis of their crops, time saving and helping improve the yield of their farms [2]. One form of GIS data that can be recorded efficiently using UAVs is height data in the form of a heightmap. Height values of a landscape can be represented by an image of height values per pixel, a heightmap or DEM (Digital Elevation Model). Such a DEM recorded via UAV simply contains the highest value at each pixel in the image of land. This means this height could be the canopy of a tree, the ground around trees or any other object on the surface of the landscape. The height value found at the location of a tree canopy is therefore the height of the tree plus the height of the ground below it. Therefore, one can determine the height of each tree in such a DEM by subtracting the height of the ground from a height value where a tree resides in the image.

Image processing algorithms can be developed to implement this process, but without truly knowing the height of the ground below each tree these algorithms cannot be properly tested for their accuracy. More specifically, this research explores methods to generate sufficiently complex and variable DEMs to represent tree orchards with known height values in order to aid testing of ground and tree height estimation. We explore methods to algorithmically create different types of landscapes and store the resultant height data for testing. We then add simulated trees in a variety of ways, storing the position of each tree and their shapes for further testing. The generated DEMs need not fully simulate the realism of a tree orchard, but need to be complex enough to ‘stress’ test the robustness and fidelity of image processing algorithms developed to extract the height of the trees and ground.

Two types of image processing algorithms are tested using these synthetically produced heightmaps. Firstly, tree segmentation algorithms which determine whether a pixel on an image represents a tree or ground. Such algorithms produce a mask of the original image indicating where trees are present. Secondly, ground height estimation algorithms which estimate the height of the ground below a tree pixel are tested. These height estimation algorithms take the mask produced previously and interpolate the height of the ground where the tree canopies occlude the ground. As the second algorithm uses output from the first it is important that each is tested independently, so that errors in the mask from the first algorithm do not affect testing the second. There are several variables and requirements that our generated DEMs need to satisfy in order to produce sufficient variability of orchards, such as varying ground elevations and surface smoothness of the landscape. Other characteristics are also considered such as the pattern of growth of tree rows and the shapes and sizes of tree canopies. Tree orchards that are analyzed using the algorithms we aim to test will typically be grown on relatively even land surfaces and in neat rows however we aim to create

quite extreme versions of these characteristics to stress the algorithms rigorously.

This paper is laid out as follows. Section 2 provides background information about DEMs and producing procedural landscapes. Section 3 provides information on where previously height estimation has been undertaken in related work using DEMs. Section 4 describes the structure and implementation of our DEM generation system and our evaluation system that analyzes the results of tree and ground height estimation using the DEMs we generate. Section 5 provides insight into the design of the entire experiment. Details are provided here about what source data we were given the hardware used to conduct the experiment and a high-level view of the process involved in generating and analyzing our results. Section 6 provides significant results found when evaluating our DEMs and provides a discussion about the effect these results had and why they occurred. Section 7 we provide conclusions derived from the results and discussion had in section 6.

2. BACKGROUND

In this section we discuss the various aspects to the process of creating synthetic DEMs that simulate features found on a farm orchard. DEMs and how they are produced are discussed. We further explore methods to create DEM landscapes as well as tree vegetation for them.

2.1 Digital Elevation Models

Heightmaps in GIS are typically represented by either a Triangular Irregular Network (TIN) or a Digital Elevation Model (DEM). TIN models are computationally expensive, but highly accurate [3]. However, for our level of accuracy and efficiency they were deemed infeasible for our purposes. We therefore will be generating DEMs to represent our heightmaps. DEMs are images, typically top down greyscale images of landscapes, where each pixel in the image represents a height value of the landscape. Data can be recorded in different ways to produce a DEM.

The method used to capture our source data was that of LiDAR (Light Detection and Ranging) survey. LiDAR is a method for measuring distances and geographical data using an infrared laser light being aimed at a point on the ground and the reflection of this light being measured by a sensor. LiDAR is constantly improving in the potential resolution and amount of additional data that it can record[4] and thus is an effective way to record DEM data. A DEM can be stored digitally using the .tif format, otherwise known as a TIFF file. This is a computer file format which is widely supported for image manipulation and GIS to store raster graphics data for images.

2.2 Landscape Generation

For developing natural landscapes various methods can be explored using procedural generation [5]. Fractal generation, physical simulation and example-based learning can be used to create realistic varieties of landscape. However, the aim of this project does not require the level of realism that these methods produce, and such complexity will be a hinderance to the speed in which we generate images.

L-systems provide another procedural method that can be used to model forest or tree models. Lindenmayer systems or L-systems can efficiently be performed in parallel [6] to model geometric shapes that simulate natural features such as vegetation and the positioning of natural vegetation on landscapes. Again, however these methods provide a level of realism that we don't require to merit implementing for our purposes.

3. RELATED WORK

Much of the research that has gone into height extraction algorithms from heightmaps has been focused on urban areas. Edge detection and ground plane removal can be used to find the heights of buildings where there is a significant difference between the ground and non-ground objects and you have clear visibility of the ground in the majority of the image [7]. Our source images of tree orchards lack the defined shapes and visibility that urban areas have. This makes it more difficult for tree estimation methods to process these images accurately. We discuss below the two core parts to the tree and ground height estimation process, tree segmentation and ground estimation.

3.1 Tree segmentation

There is a two-part process that can be used for tree segmentation. Firstly, Simple Linear Iterative Clustering (SLIC), can prepare given DEM data by efficiently clustering adjacent pixels in the image of similar height [8] and for each cluster a second method, Watershed, finds whether a region in this cluster fits a tree canopy shape and creates a mask of this canopy indicating which are tree pixels and which are not. SLIC is an efficient pre-processing method that simplifies an image into a number of similar regions, with it you can set a desired number of pixels to cluster together and initializes the cluster centres uniformly to make it easier for the watershed to analyze each region. This works well for orchard trees as they will roughly be the same size across an orchard. The Watershed method operates by dividing an image up into catchment basins with each basin associated with a local minima [9]. Similarly, it can be used for hills with a local maxima as it simply divides by a change in the height gradient in areas in an image. For our purposes pixels that decrease gradually as you move away from a local maximum and then drop off drastically at a certain point, resemble a tree canopy and then a drop off to ground. The Watershed method is highly sensitive to small changes in pixel heights so the SLIC pre-processing is vital to improving the results of the watershed [10].

3.2 Ground Height Estimation

Interpolating the ground heights for a DEM can be done using inverse-distance weighting (IDW) interpolation which is a popular spatial interpolation model used in Geosciences [11]. IDW calculates a desired locations values based on a weighted average of neighboring points. This weighted average is based on the inverse of the distance to each point given to the weighting calculation [12]. As the weighting is based on these inverses it is more heavily influenced by the weights of pixels closer to a desired location than further away making this an effective technique for interpolating pixel values in DEMs with many small gaps between known pixels. Our tree pixels in our DEMs will have many gaps in between which are ground pixels and so can be effectively interpolated using this method.

4. DESIGN AND IMPLEMENTATION

In this section we outline the overall design of our system producing our DEMs and our system evaluating the effectiveness of these generated DEMs. We describe what additional development tools were used in developing our research. We also describe in detail how we implement each part of the DEM generation and evaluation process as well as the requirements that our software must meet in order for us to

produce results that are sufficiently complex and variable to test tree height estimation of a DEM.

4.1 Design

The goal of our software is to produce synthetic DEMs in the form of .tif files and to process these .tif files using external image segmentation algorithms and ground height estimation algorithms. Our produced DEMs will be created by first creating a landscape DEM with different surface elevations and bumpiness to emulate the irregularity of the ground that our source orchards are grown on. Secondly, trees will be added to the DEM to simulate various features found in a typical real-world orchard. We will be growing groups trees either in neat rows or along contour lines found in the DEM. Our DEMs are represented by 8bit floating point greyscale images where each pixel is a floating-point value of 0 to 255.We implement our methods using C++ with the aid of the OpenCV library. We produce varying ground elevations using mathematical functions to create smooth gradients and hills for our landscapes.

4.2 System Design

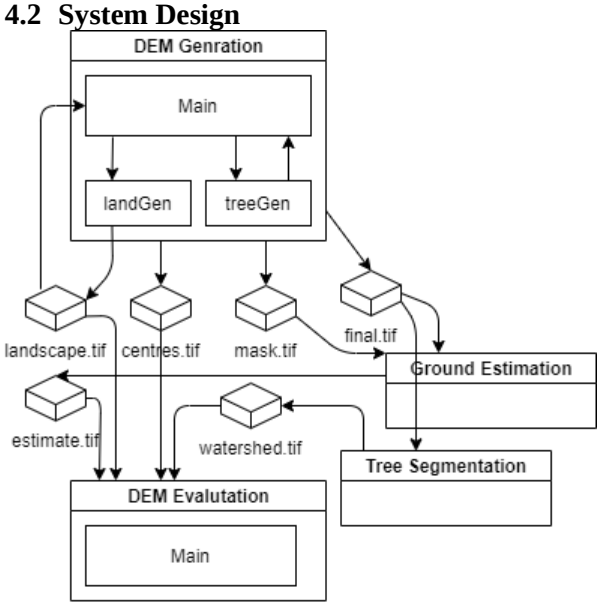


Figure 1: Architecture design for experiment system

The overall structure of the experiment system will be split up into a DEM Generation layer and a DEM Evaluation layer. The DEM generation layer contains the functionality to create landscapes and from the produced landscape imposes groups of trees to produce a final image.

DEM Generation steps:

1) We produce, using the landGen class, the underlying landscape.tif DEM containing the true ground heights for the DEM.
2) Using the treeGen class, groups of tree positions are produced. In the case of growing trees on contours the landscape.tif DEM is used to find contour lines and these contour lines are passed to treeGen.
3) Using the tree positions generated by treeGen, our main program draws trees on the landscape.tif DEM
4) Additionally, our main program produces a centres.tif DEM, containing the centres of each tree and a mask.tif DEM indicating exactly which pixels are tree pixels. These two DEMs are used for evaluation later.

From external tree segmentation methods, we receive a watershed.tif image containing the estimated positions of the trees given final.tif. The watershed.tif image is our only non-greyscale DEM and contains different colour values for each tree it finds. From external ground estimation methods, we receive an estimate.tif image of our estimated ground heights using the mask.tif and final.tif DEMs.

The DEM evaluation layer compares the outputs of the estimation methods as follows:

1) For tree segmentation, the watershed.tif DEM is compared against the centres.tif DEM. Bounding boxes are drawn from each centre pixel in centres.tif and these are compared with the bounding boxes for each colour in the watershed.tif image.
2) For ground estimation, the estimate.tif DEM and the landscape.tif DEM have their height differences compared. The ideal estimate.tif has as little difference to landscape.tif as possible.

4.3 Development Tools

The OpenCV library was used to edit, read and write our .tif file data. OpenCV provides a library of image editing functions that we make use of to format and process the raster data in our images, it also allows us to view the images we are working with in 2D. Aerialod is a viewing library we used for visualizing DEMs in 3D to better assess our progress and the look of the DEMs we produced in our research. DEMs can be read by most image editing software programs. We used GNU Image Manipulation Program (GIMP) for quick image edits and pixel colour probing for experimentation when developing our methods. QGIS is an open source GIS tool which we used to view our various source and generated data in 2D. QGIS also provides additional GIS details about the DEMs such as the full ranges of height values in each image and the number of layers in our source data, which had multiple layers. See the Software Links section for URL references for the software used.

4.4 DEM Feature Requirements

The trees and landscape of orchards have multiple important features that needed to be reproduced as they directly affect and test the strength of image processing algorithms. It is required that our landscapes simulated both flat land and gradual hilly slopes. We also developed steeper and more irregular land for experimentation purposes. Trees must have the ability to both be spread and overlap in their tree canopies. This overlapping of tree canopies creates a difficulty for the tree segmentation algorithm that must be overcome as well as reducing ground visibility for the ground height estimation algorithms. Trees added to the images resemble the tree canopy characteristics displayed in our source DEMs. Typically canopies in these DEMs are circular with some roughness around the edges that was reproduced in our generated canopies. The heights in the source canopies are highest at a point roughly in the middle of the canopy and reducing in height gradually moving outwards from the centres. This creates a semi-circular shape when seen from a side on view and our generated canopies resemble this. Trees are natural and therefore there needs to be some irregularity in the shapes of our generated trees, so a small amount of randomness was added to the height values produced. The way rows of trees are grown in our source

images are of two varieties, a straight-line row of a trees and trees grown along contour lines. We therefore generated two variations of tree groupings to emulate these two patterns.

4.5 Implementation

The following describes how we implement the generation of a tree orchard landscape DEM. This DEM is created in stages where first a landscape DEM is generated for just the ground height values. Then we generate positions for each tree we want to add to this landscape. Using these tree positions we finally calculate and draw the tree height values onto the landscape.

4.5.1 Landscape Generation

In producing our variations of land, we create an empty image of 500x500 pixels and change each pixel height in this grid by a calculated amount to produce an effect we desire such as a sloping hill or bumpy land. A resolution of 500x500 was chosen as this produces a sufficiently large enough image to contain a large area of landscape pixels as well as space for a tree orchard region within that which is sufficiently large enough to place enough trees to evaluate our generated DEMs. For simple variations we create flat and linear gradient slopes. More complicated variations make use of sin and cos functions to produce hills and ditches in the landscape.

4.5.2 Tree Positioning

In positioning our groups of trees, we implemented two methods. One to generate neat rows of trees in straight lines and another for producing trees along contour lines. Both methods involved a small amount of jitter in the positioning of trees to simulate the irregularity found in real life orchards. In generating our neat rows grid of trees, we simply loop through the available orchard area and based on the specified gap between trees and a random offset to represent jitter in its position we store set positions at which to spawn trees. The following pseudocode shows how we choose where to place trees for our neat rows:

```
treeGroup trees
createTreesGrid( treeWidth, gapX, gapY, gridWidthY, gridWidthX):
        randomDistX = random(- treewidth/4,  treewidth/4)
        foreach x-vaue i in the grid:
                xDist++
                randomDistY = random(- treewidth/4,  treewidth/4)
                foreach y-vaue j in the grid:
                        yDist++
                        if(xDist > gapX && yDist > gapY):
                                trees.add({ i + randomDistX , j + randomDistY })
```

The distances between trees are adjusted by a randomDist variable for both X and Y as well as a set minimum gap to divide the trees into spaced rows. these randomDist variables are based on the width of the tree to be able to dynamically work for different sized trees. Positions of trees are based on these randomDist variables as well as the incremented integer variables in each for loop for X and Y.

In order to generate trees along contours we first find contour lines in a given landscape image. OpenCV provides a findContours() function which returns a data structure containing the start and end points for each vector making up the contour lines in the image. A straight-line contour across the image found using findContours() would result in a single vector containing start and end points of this line while contours on a non-straight line contour would result in multiple vectors making up small lines that make up the total contour curve.

In order to detect neat contour lines, the image first needs to be Gaussian blurred and then normalized. The Gaussian blur acts as a low pass filter that we use to reduce the amount of noise which would otherwise produce unwanted contours lines by the findContour() function. We picked a normalization range of between 0 and 10 for our generated landscapes. Without normalizing the number of contours found is too sensitive because most DEMs contain large amounts of varying heights and the resultant contour data structure contains too many contours to realistically use as almost every pixel in the image becomes its own contour grouping. Once the contour lines have been found for the image, the following pseudocode shows how we add trees along contour lines:

```
vector<vector<Point>> contours = getContourVectors( ) //obtain all the vectors for the
contours in the image
treeGroup trees
createTreesContour( treeWidth, gap, gridWidthY, gridWidthX, ):
        bool [gridWidthY ][ gridWidthX ] availableSpawns //create a grid of possible spawn
positions and set them all to true
        foreach item  i in contours:
                foreach item j in the contours[ i ]:
                        posX = contours[ i ] [ j ].x
                        posY = contours[ i ] [ j ].y
                        if availableSpawns[ posX ] [ posY ] is true:
                                trees.add({ posX , posY })
                                foreach position within the range of treewidth/2 + gap:
                                        availableSpawns[ x ] [ y ] = false
```

A group of tree positions is created by looping through the contours vector<vector<Point>> data structure and for each contour point checking if the position is available to spawn a tree. If the position is free to spawn a tree then we make the surrounding pixels unavailable to spawn another tree and store this postion as the position for the tree. The surrounding pixels chosen to be made unavailable is based on the width of a tree and the desired gap. The irregular spacing we desire comes by virtue of the found contour vectors being of different distances. The ‘gap’ value prevents trees from being too close if there is a high density of contour line vectors.

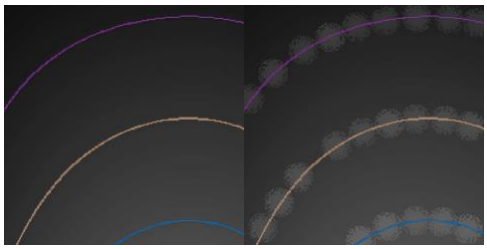


Figure 2: The contour lines for a hill landscape and the resultant added trees

Additionally, we add a random chance to drop a tree leaving a gap in the orchard to stress the image processing methods. For example, if the tree segmentation algorithm was designed in such a way that it relied on the distance being consistent between trees then this randomly dropped tree would allow us to see this downfall in the algorithm during evaluation of the results.

4.5.2 Tree Creation

The desired tree shape which resembles our source image is a circular tree shape with a local maximum that decreases as you

move away from the centre. We generated images with two types of trees, a 'simple' case and a 'complex' case.

The complex case for our trees will have a small degree of randomness to the canopy heights as well as gaps in the canopy close to the boundaries of the tree. This is to simulate irregularity of height values in the tree canopy as well as change the shape from being perfectly circular for each tree canopy.

The simple case type tree has no holes in their canopy and a smooth linear decrease in the canopy heights from the centre to the boundary of the tree. For an additional case we developed trees of much smaller size. Our simple and complex cases used trees roughly of a diameter of 30 pixels while our smaller tree case had trees with a diameter of 15 pixels. The following pseudocode represents how we generated our simple case tree.

```
drawEasyTree( radius ):
    int treeGrid[ radius*2][radius*2]
    foreach x-value i in treeGrid:
        foreach y-value j in treeGrid[ i ]:
            if treeGrid[ i ] [ j ] is inside circular shape:
                int diffX = abs( radius - i ) //absolute value
                int diffY = abs( radius - j )
                int dist = sqrt( diffX² + diffY² )
                float treeHeight = pow(abs(radius – dist), radius/50) x radius/2
                treeGrid[ i ] [ j ] = treeHeight
    return treeGrid
```

A circular two-dimensional tree grid is created based on the width of the tree. The circular shape comes from checking whether a pixel is within a circle based on the difference between the treewidth/2 for a radius and the current pixel being assigned (i, j). In the treeHeight calculation a logarithmic function is used based on a fraction of the tree radius and the value of 50 is chosen as a quotient as it was found through experimentation that this value would produce the smooth flattening out shape that we desired for the centres of our tree. The whole function is multiplied by the radius/2 to further improve the shape we produce in our tree.

The following pseudocode represents how we generated our complex case tree:

```
drawHardTree( radius , holeThreshold):
    int treeGrid[ radius*2][radius*2]
    foreach x-value i in treeGrid:
        foreach y-value j in treeGrid[ i ]:
            if treeGrid[ i ] [ j ] is inside circular shape
                int diffX = abs( radius - i ) //abs: absolute value
                int diffY = abs( radius - j )
                int dist = sqrt( diffX² + diffY² ) // sqrt: square root
                int canopyVariance = Random( -2 , 2)
                int canopyHoleChance = Random( 0, dist)
                if canopyHoleChance > holeThreshold: // holes can only occur
            passed a threshold distance away from the center and occur more
            often closer to the edges
                    float treeHeight = pow( abs ( radius – dist )
                    , radius/50 ) x radius/2 //pow (A, B): A to the power of B
                    treeGrid[ i ] [ j ] = treeHeight + canopyVariance
    return treeGrid
```

The implementation of the complex case has the added canopyVariance value which produces irregular heights for the tree canopy. The canopyHoleChance variable is designed to only produce holes for pixels at a distance away from the centre that pass a set threshold value meaning there are no holes in the tree until this threshold is reached. Basing the random canopyHoleChance variable from zero to the current distance from the centre means a pixel further from the centre has a higher chance to be a hole in the canopy, an effect we desire.

For our smaller trees a much simpler tree is generated as with less pixels the gradient of the tree canopy cannot contain too much variation due to the space limitation. This pseudocode indicates the generation of our smaller tree (complex case):

```
drawSmallTree( radius , holeThreshold):
    int treeGrid[ radius*2][radius*2]
    foreach x-value i in treeGrid:
        foreach y-value j in treeGrid[ i ]:
            if treeGrid[ i ] [ j ] is inside circular shape
                int diffX = abs( radius - i ) //abs: absolute value
                int diffY = abs( radius - j )
                int dist = sqrt( diffX² + diffY² ) // sqrt: square root
                int canopyVariance = Random( -2 , 2)
                int canopyHoleChance = Random( 0, dist)
                if canopyHoleChance > holeThreshold: // holes can only occur
            passed a threshold distance away from the center and occur more
            often closer to the edges
                    float treeHeight = abs ( radius – dist )
                    treeGrid[ i ] [ j ] = treeHeight + canopyVariance
    return treeGrid
```

The small tree generation has both the 'simple' and 'complex' cases mentioned previously, but the treeHeight calculation is a simple linear calculation based on the distance from the centre of the tree.

4.6 Evaluation Implementation

4.6.1 Evaluation by Segmentation

In evaluation the effectiveness of our DEMs to stress our segmentation algorithms we use a combination of the Jaccard Index otherwise known as Intersection over Union (IOU) and visual comparisons of the final watershed result from our segmentation. IOU tests are used to compare similarity and diversity in set of data [13] and can be implemented simply for our purposes by knowing the bounding box for our predicted tree pixels produced by our segmentation and the true bounding box for a particular tree in our generated image. IOU is computed by taking the area of an intersection box between two bounding boxes and dividing this by the union of the boxes. The intersection box is an area of overlap between the predicted bounding box and the true bounding box. The union of the two boxes can be found by adding the area of each box and subtracting the intersection area found previously. Our IOU calculation is done as follows:

I = Intersection Area, U = Union Area,

BAA = Box Area A, BAB = Box Area B,

IOU = Intersection over Union, n = current tree box index,

We can define:

$I(n) = BAA(n) \cap BAB(n)$

$U(n) = BAA(n) + BAB(n) - I(n)$

$$IOU(n) = \frac{I(n)}{U(n)}$$

4.6.2 Evaluation by Ground Estimation

For evaluating the effectiveness of our generated DEMs in testing the ground estimation methods we look at the average difference in heights between the true height values stored in our landscape.tif map versus the estimated heights for each estimated pixel in the output of our ground estimation (estimated.tif). We assess the height difference by looking at the average height difference over all the estimated pixels and its standard deviation as well as a visual comparison displaying how well each patch of ground below each tree was estimated. The ground estimation methods estimate the ground for which each tree in the image occludes the ground, therefore the underlying patches of ground occluded by each tree are the pixel areas we are interested in assessing.

To obtain each patch of ground, we first load all the pixels for each tree into a data structure (vector<vector<Point>> trees), where each tree vector contains a group of pixel positions associated with it. For each of the pixel positions the corresponding pixel heights in both the estimated.tif map and the landscape.tif map are obtained. For each patch of ground an average height is calculated based on the average height of each of its pixels. Using the average height for each patch we compute the average difference between the estimated ground heights and the true landscape heights and the standard deviation of this average difference. For our visual assessment we draw each patch to an image. We indicate using colours how accurately the ground was estimated. Green for a low difference between the heights (less than 0.5 difference), orange for moderate difference (between 0.5 and 2) and red for a high difference (above 2). These values were chosen through experimentation with the results being evaluated to show the significant areas in the DEMs where ground estimation failed.

5. EXPERIMENT DESIGN

Here we outline the background details for our research experiment such as the source data for tree orchards provided by Aerobotics, the hardware we use to perform our research and the high-level view of evaluating our results produced.

5.1 Hardware

Generation of test DEMs and evaluations were done on Ubuntu 19.10, intel i7-8550U (1.8 GHz x 8) and 12 GB RAM.

5.2 Source Data

For the overall project of tree height estimation source data was provided to us by Aerobotics, which conducts drone surveys of agricultural land for productivity monitoring and analysis. The images provided by Aerobotics are all 32-bit floating point uncompressed .tif images with multiple layers of resolutions. The resolutions given ranged from 10662x8520 down to 167x134, so a high level of precision was available to us. As the external image processing algorithms are to be used on these images the test DEMs we produce needed to emulate certain features that are displayed in our source data. For the most part our source data contains neat uniform rows of trees. Trees in the images are of a similar diameter and gaps between them are relatively uniform for a natural collection of objects.

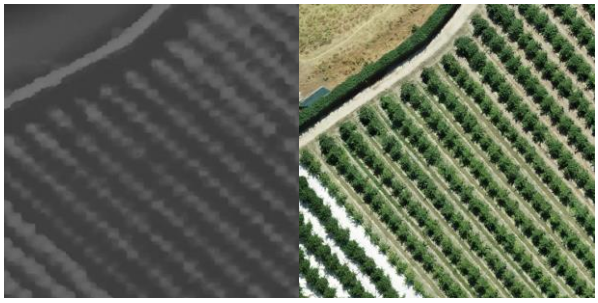


Figure 3: Aerobotics sourced heightmap and reference image.

5.3 Evaluation Design

The produced final images that represent our synthetically created tree orchards are accompanied by the underlying landscape heights (without the trees added). Additionally, the positions of each tree placed on the landscape is stored and a mask showing exactly which pixels in the final image are part of tree canopies. Using these accompanying images and the outputs of the external image processing methods we evaluate the effectiveness of our generated DEMs in testing image processing methods.

5.3.1 Evaluation of Testing Tree Segmentation

In evaluating the effectiveness of our test DEMs in testing tree segmentation algorithms we look at the number of trees flagged in segmentation versus the actual number of trees in our final image and compare the IOU values for the segmentation result versus the true image. Our DEM _Evaluation class reads in the centre positions from the centres.tif image and computes the bounding boxes for each tree based on these centres and the width of the tree we're working with in the current final.tif. We then read in the watershed.tif image produced by our tree segmentation and compute the bounding boxes for each unique colour in the watershed. Each watershed.tif grows regions of unique colour for each tree it has predicted which we can use to find its bounding box. We then compare each predicted bounding box against every true bounding box to find the highest IOU value for the predicted box and calculate the mean and standard deviation for the IOU's. Mean IOU values of above 0.75 are considered a successful prediction by segmentation with values of below 0.5 being low and indicating a failure to correctly segment the trees.

5.3.2 Evaluation of Ground Height Estimation

In predicting ground truth values, we compare the original land DEM (before adding trees), to the images produced by the ground height estimation. The height difference for each patch of ground occluded by a tree in the final image is analyzed. The average height of each patch is calculated for both the generated DEMs ground truth values and the estimated ground heights. The height difference between these averages is evaluated.

Additionally, we draw a DEM for visual evaluation where we display using colour coding which patches of ground were correctly estimated with a low average difference in heights and which were badly estimated with a high difference. Green for a good estimated patch, orange for acceptable and red for badly estimated patch.

6 RESULTS AND DISCUSSION

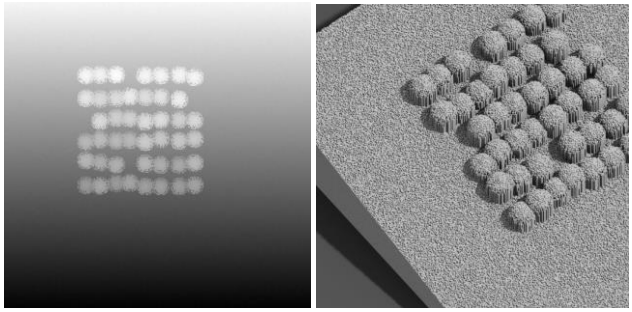
Here we list the most significant results and compare how well the tree estimation methods did for both tree segmentation and ground estimation.

Table 1: Summary of added tree variations.

Variation Type	Tree Diameter (pixels)	Tree Type (see section 4.5.2)	Overlapping Tree Canopies
Simple	30	Simple	Yes
Complex	30	Complex	Yes
Complex and Spread	30	Complex	No
Simple and Small	15	Simple	Yes
Complex and Small	15	Complex	Yes

Table 1 summarizes the differences between our five standard variations of added trees that we added to each landscape type. For example, the 'Complex' variation adds trees that are 30

pixels in diameter have a canopy shape and height distribution of the 'Complex' type tree described in section 4.5.2 and have trees who touch and sometimes overlap. These variations in combination with each of the landscape types we produced resulted in our final generated DEMs. See Appendix A to see 3D views of each landscape type for a better understanding on how the underlying landscape types look.



(A) (B)

Figure 4: final.tif DEM generated using the "Complex" variation on the "Steep Slope" landscape (A). 3D view of this DEM (B)

6.1 RESULTS

We produced a total of 30 different DEMs based on varying versions of landscapes and different variants of tree types. Additionally, we generated DEMs for experimentation which were not used in the recording of results and evaluation of external estimation methods. See Appendix B for all generated DEM resultant final.tif maps. Some examples of resultant final.tif DEM maps are displayed below:

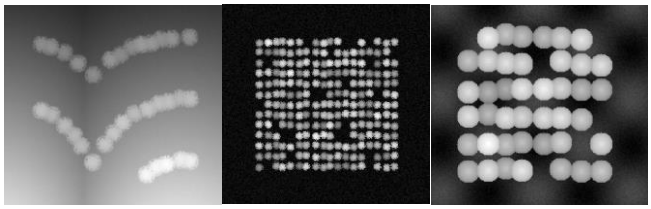


Figure 5: Example results of generated final.tif DEMs

Figure 5 shows some of the variety in our resultant segmented DEMs. Later we compare how these various generated DEMs 'stress' tested various tree and ground estimation methods. DEMs viewed in this way can be understood as lighter areas being of a higher height value and darker being lower height values. Circular brighter patches indicate trees and jagged edges to the trees indicate a 'complex' tree type being used.

6.1.1 Testing Tree Segmentation by IOU

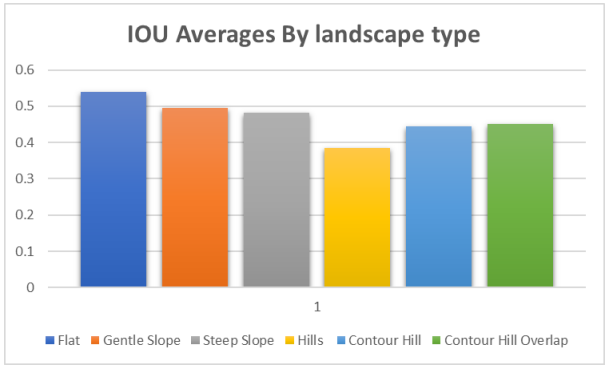
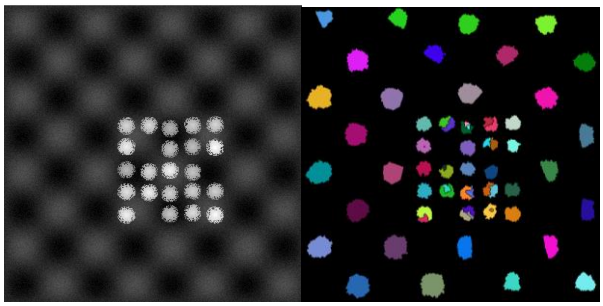


Figure 6: Graph showing IOU values averaged for each landscape type.

Figure 6 shows how accurate the segmentation results were in predicting which pixels were trees an which weren't. Of the landscapes two out of the six had their trees produced by our contour generation technique ("Contour Hill" and "Contour Hill Overlap"). Of the landscapes the one that segmentation had the most difficulty with was the "Hills" landscape with the lowest average IOU result of 0.39, while the most successful segmentation was done on the "Flat" landscape with an average IOU result of 0.54.

These IOU values are relatively low averaging 0.47. These low may be due to falsely flagged trees or multiple small trees predicted in a space where only one tree should be found.



(A) (B)

Figure 7: Final Test DEM image (A), Watershed tree segmentation result (B)

Another problem which this generated final DEM (A) clearly shows up is an issue with the watershed segmentation (B) where even low-lying hills were detected as being trees. This indicates the possible reason for such low mean IOU and an overestimate in the total number of trees found in the "Hills" type landscape .The dark grey larger pixels in figure 10 above represent low hills while the brighter white pixels are the trees who have significantly higher height values compared with the rest of the image and so it is a problem that these dark grey pixel groups were picked up as trees by our segmentation. This is due to the low hills having their own a local maxima so tree segmentation picks up each small hilltop as a grouping of canopy pixels.

This leads us to an immediate improvement to the watershed segmentation that can be done in thresholding the heights that can be detected and labeled trees within a range. it can be assumed that a similar issue occurs if there are large objects such as mountains with a local maxima present in the image as well.

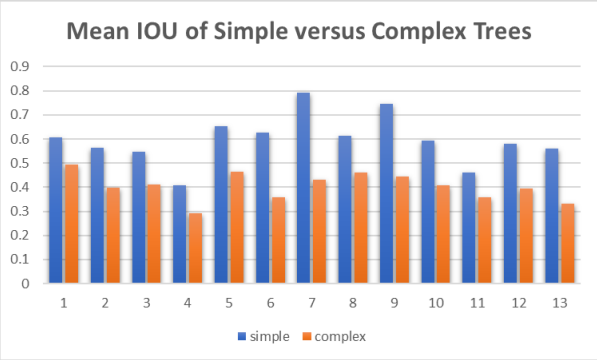


Figure 8: Graph comparison of the IOU values for Simple type trees versus Complex Trees.

Figure 8 shows the effect caused by using the more Complex type trees versus the Simple type trees. Included in the results are both small trees of diameter 15 and normal sized trees of diameter 30. Each simple and complex pair corresponds to tree images that were built with the exact same variables except with different tree types used. On average an IOU for all simple type trees was 0.6 with a standard deviation of 0.1 and for complex type trees the average IOU was 0.4 with a standard deviation of 0.06. Indicated in these results is a consistent decrease in IOU when using 'Complex' trees whose canopy varies greatly and who's boundary is not easily defined. This an ideal result which indicates a weakness in the tree segmentation methods in dealing with variable and noisy data.

In most DEMs with any 'complex' tree type one can see multiple watershed predicted trees within single trees. These multiple trees would produce a low IOU when compared against the final.tif DEM because these predicted trees have smaller bounding boxes than the true bounding boxes. This would also affect the number of trees estimated negatively. This is an important problem that needs to be addressed in segmentation as small variations in the tree canopy should not result in multiple predicted trees within a single tree. In this way our generated DEMs showed a significant problem with tree segmentation that needs to be addressed. In figure 7 this phenomenon can be seen where, when looking at the trees predicted in the watershed (B), multiple colours are drawn for a singular tree.

6.1.2 Testing Tree Segmentation by Number of Trees Found

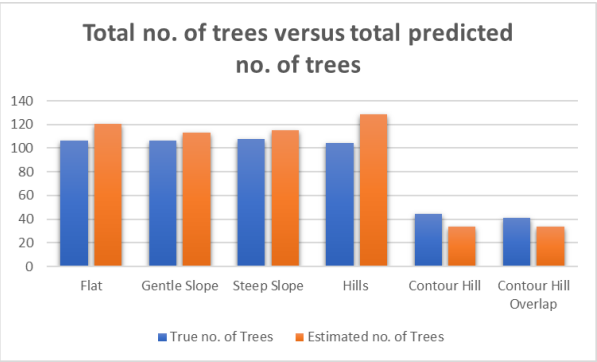


Figure 9: Graph showing the true number of trees in the final DEMs versus the estimated number of trees by each watershed.

Figure 9 shows how close the total number of actual trees for each landscape were to the predicted total number of trees for each landscape. Each total tree count is averaged from the five different variations of trees for each landscape type showed in table 1. The largest difference was apparent in the 'Hills' type landscape where there were 105 trees present, but tree segmentation detected 128 trees.

We can notice in these results that estimated trees totals (orange) for those grown along contours had less than the true number of trees (blue) in the DEM while consistently spaced row positioned tree groups had a higher estimated number of trees versus the true number of trees. This may be due to the fewer true number of trees in the contour images which were on average around 42 trees while the normal row pattern trees had on average 106 trees in their final.tif DEMs. The number of trees in the contour Images was perhaps too low in our resultant final DEMs and we should explore in the future a more similar tree count to the row patterns.

What can be noted from the total tree numbers is that the estimated number of trees was not affected too harshly by the angle of the slope. This can be seen from the results of our 'Flat', 'Gentle Slope' and 'Steep Slope' landscape types. This is not true for the IOU results which got significantly less as the angle of the slope got more extreme. Ranging from 0.54 for 'Flat' to 0.48 for 'Steep Slope. This indicates that tree segmentation is more difficult as the slope increases and means our generated DEMs successfully simulated this feature that may be present in natural orchard landscape.

6.1.3 Testing Height Estimation Results

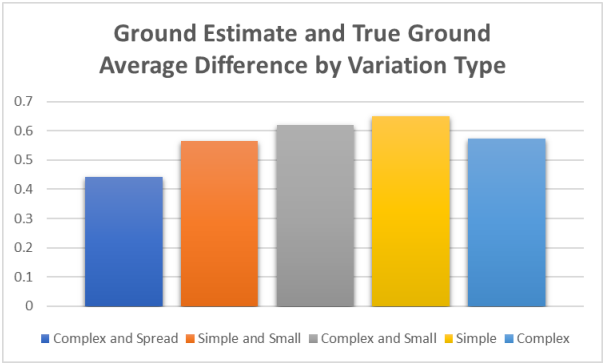


Figure 10: Graph showing the performance of ground estimation through the average difference in heights between the estimated ground and the true ground heights.

On average the differences in heights across all variations of maps was 0.57 with a standard deviation of 0.65. This is a relatively low difference and the ground estimation performed highly in interpolating the true height of the ground in most cases. Most successfully the spread-out tree DEMs ('Complex and Spread' in figure 10) was estimated with an average difference of 0.44 and a standard deviation of 0.54. This was the expected result as DEMs where trees have more ground between them allow for easier interpolation as there is more ground to interpolate with on the boundaries of each tree. The lowest performance was found with the 'Simple' tree type variation with an average difference of 0.65 and a standard deviation of 0.7 which was unexpected. While this is still a relatively low average difference there are several possible reasons for this. In the 'Complex' tree type there are holes near the edges of the tree, this could lead to more correctly interpolated ground if the ground estimation methods make use

of these holes. Additionally, more densely packed tree groups could result in worse results for interpolation as there is less ground to use to interpolate.

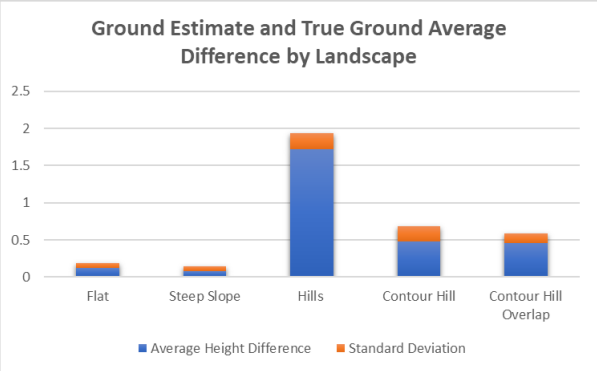
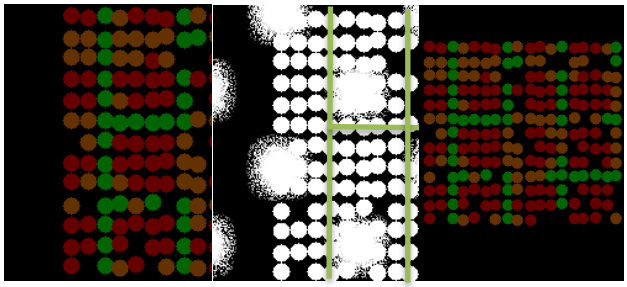


Figure 11: Graph showing the performance of ground estimation by landscape type.

Figure 11 clearly shows that the most significant factor that 'stressed' the ground estimation methods was the landscape type as the results vary greatly while figure 10 shows very slight differences based on tree type. The 'Flat' and 'Steep Slope' landscapes averaged differences of 0.12 and 0.08 with standard deviations of 0.08 and 0.06 respectively. This low average difference could be due to these landscapes having linear based landscape gradients and this being easier for interpolation in the ground estimation. This is highly likely as ground estimation would be able to interpolate quite simply using simple linear mathematical formulae.

Performing the worst by far was the 'hills' type landscape with an average difference of 1.72 and a standard deviation of 0.22. This landscape had the most varying and dynamic landscape so it is understandable that ground estimation resulted so poorly. Appendix 1 (Hills) displays this landscape type.

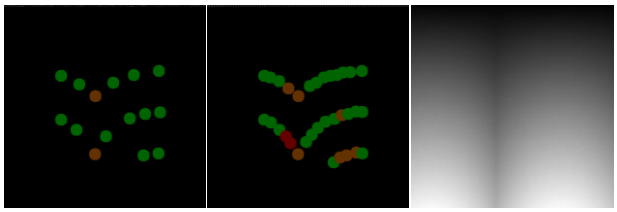


(A) (B) (C)

Figure 12: Piece of visual DEM indicating low average differences for interpolated area in green (A), normalized final.tif image with added green lines (B), full visual DEM indicating effectiveness of interpolation (C).

Figure 12 (C), displays in green, patches of pixels areas where average difference in ground was below 0.5 indicating a low difference. This interpolation is for the 'Simple and Small' (Table 1) tree variation on the 'Hills' landscape (Appendix A). There seems to be a pattern where interpolation is effective in straight lines along edges of the hills and ditches as displayed by figure 12 (B) where light patches are hills and dark areas in between are ditches. The green lines mark the corresponding green pixels in figure 12 (A).

Interpolation clearly struggles when the ground is more irregular and one cannot interpolate simply from checking one gradient angle. An average of multiple passes for ground estimation taken at different angles may improve the results for each patch of ground. The contour landscapes also have a significantly higher average difference when compared to 'Flat' and 'Steep Slope', but still much less than 'Hills'. The contour landscapes however are still very large sloping hills and so they don't create as much irregularity as the 'Hills' landscape does. While real world orchards will typically not be grown on such varying surfaces as 'Hills' they are still possibly grown on more irregular surfaces than the other four landscapes we compared.



(A) (B) (C)

Figure 13: Average differences visual DEM for spread trees on contours (A), average differences visual DEM for overlapping trees (B), underlying landscape.tif for each DEM (C)

In figure 12 we can see another example of a sudden change or irregular ground being a difficulty for the ground estimation. In figure 12 (A) and (B) the orange and red patches occur along the seam or dip between the two hills. We can see this in the underlying landscape figure 12 (C) where the dark line shows where the two hills dip down and meet. Interpolation clearly picked up a gradient, but with the sudden change to a hill going down to one going upwards the estimated ground height was interpolated incorrectly.

7. CONCLUSIONS

Our generated DEMs were able to pick up deficiencies in the image processing algorithms and so were an effective way to evaluate our image processing methods, however more rigorous testing could have been done by generating more variations of maps and adding more variability to each map produced. Other pitfalls include the lack of realism in our DEMs as they are still quite crude mathematically synthetic landscapes, and much could be done to further simulate natural features. There are many variables involved in natural landscapes even for areas as neat and uniform as tree orchards.

In tree segmentation our generated DEMs were able to pick up issues that arise with low lying hills and varying slope steepness in landscapes as well as the difficulty with detecting tree boundaries for densely packed trees versus spread trees. In ground estimation our generated DEMs showed that irregular hilly landscapes cause the greatest difficulty for interpolation and more densely packed trees also make it harder than spread trees to interpolate the ground.

Several pre-processing methods were used in preparing our synthetic DEMs. The normalizing of the image before finding contours using OpenCV made it significantly easy to find neat similar height contours in the image even when a large amount

of noise was present. The range that you normalize by can greatly simplify the DEM so that you can reproduce contours that fit the landscape at many levels of precision.

8. LIMITATIONS

Our source DEMs from Aerobotics were of tree orchards limited to specific small selection of tree types and farming techniques. With more varying source images we could have explored additional ways to generate DEMs that test image segmentation and interpolation techniques on more general tree orchards. With more time we also could have produced more variety in our landscapes and tree types. As tree segmentation and ground estimation was done externally, evaluating our generated DEMs by these methods relied on a third party to return the results of processing the generated DEMs. Another limitation was the unavailability of ground truth data in our source DEMs. If we wanted to generate DEMs we did not have access to the true ground values to accurately re-create a tree found in the source images.

9. FUTURE WORK

There are several features which are lacking in our implementation in our DEM Generation. How we draw a tree specifically can be improved upon and made to draw a tree that is more similar to those found in the source DEMs provided by Aerobotics as well as other tree types found typically in orchards. Testing with different sizes of trees can be more extensive as well as producing DEMs with much more trees in the added orchard. There are many more variations of landscape that can be produced. Landscapes of different types may discover more pitfalls with tree estimation methods and lead to more improvements of them. Finally, much more extensive variations using different resolutions can be done. It would be interesting to see how much higher or lower resolutions fair when processed by tree estimation methods. With larger resolutions the efficiency of these methods may be another factor of image processing that can be tested more harshly.

ACKNOWLEDGEMENTS

I would like to thank my project partners for their support throughout this project, Lynolan Moodley, who implemented the tree segmentation algorithms and Chiadika Emeruem, who implemented the ground estimation algorithms. I further thank Associate Professor Patrick Marais, my supervisor, for his essential guidance and insight throughout the project and Professor Hussein Suleman as our second reader. Additional thanks go to Michael Malahe and Aerobotics for the source DEM data and for detailed information on the source data and how it was obtained.

This work is based on the research supported wholly / in part by the National Research Foundation of South Africa.

SOFTWARE LINKS

OpenCV: https://opencv.org/

Aerialod: https://ephtracy.github.io/index.html?page=aerialod

GIMP: https://www.gimp.org/

QGIS: https://www.qgis.org/en/site/

REFERENCES

[1] Zarco-Tejada, P. J., Diaz-Varela, R., Angileri, V. and Loudjani, P. Tree height quantification using very high resolution imagery acquired from an unmanned aerial vehicle (UAV) and automatic 3D photo-reconstruction methods. *European Journal of Agronomy*, 55 (2014/04/01/ 2014), 89-99.

[2] Puri, V., Nayyar, A. and Raja, L. Agriculture drones: A modern breakthrough in precision agriculture. *Journal of Statistics and Management Systems*, 20, 4 (2017/07/04 2017), 507-518.

[3] Chen, Z. T. Systematic selection of very important points (VIP) from digital terrain model for constructing triangular irregular network. *Proceedings of AUTO-CARTO*, 8 (1987 1987).

[4] Lemmens, M. The Fierce Rise of Airborne Lidar (11/01/2017). Retrieved 19 September, 2020 from https://www.gim-international.com/content/article/the-fierce-rise-of-airborne-lidar

[5] Crause, J. *Fast, realistic terrain synthesis*. University of Cape Town, City, 2015.

[6] Boudon, F., Pradal, C., Cokelaer, T., Prusinkiewicz, P. and Godin, C. L-Py: An L-System Simulation Framework for Modeling Plant Architecture Development Based on a Dynamic Language. *Frontiers in Plant Science*, 3, 76 (2012-May-30 2012).

[7] Weidner, U. and Förstner, W. Towards automatic building extraction from high-resolution digital elevation models. *ISPRS Journal of Photogrammetry and Remote Sensing*, 50, 4 (1995/08/01/ 1995), 38-49.

[8] Xie, X., Xie, G. and Xu, X. High precision image segmentation algorithm using SLIC and neighborhood rough set. *Multimedia Tools and Applications*, 77, 24 (2018/12/01 2018), 31525-31543.

[9] Tarabalka, Y., Chanussot, J. and Benediktsson, J. A. Segmentation and classification of hyperspectral images using watershed transformation. *Pattern recognition*, 43, 7 (2010), 2367-2379.

[10] Haris, K., Efstratiadis, S. N., Maglaveras, N. and Katsaggelos, A. K. Hybrid image segmentation using watersheds and fast region merging. *IEEE Transactions on Image Processing*, 7, 12 (1998), 1684-1699.

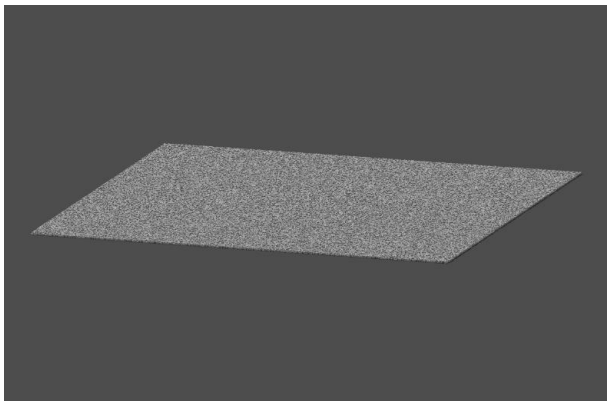
[11] Lu, G. Y. and Wong, D. W. An adaptive inverse-distance weighting spatial interpolation technique. *Computers & Geosciences*, 34, 9 (2008/09/01/ 2008), 1044-1055.

[12] Mei, G., Xu, L. and Xu, N. Accelerating adaptive inverse distance weighting interpolation algorithm on a graphics processing unit. *Royal Society Open Science*, 4, 9 (170436.

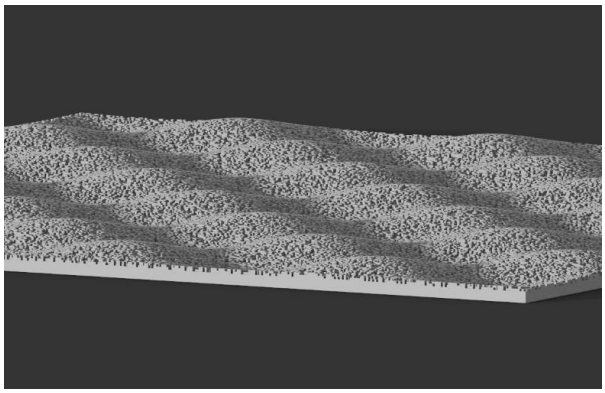
[13] Efendi, R., Samsudin, N. A., Deris, M. M. and Ting, Y. G. *Flu Diagnosis System Using Jaccard Index and Rough Set Approaches*. City, 2018.

Appendix A: Landscape Type 3D Views

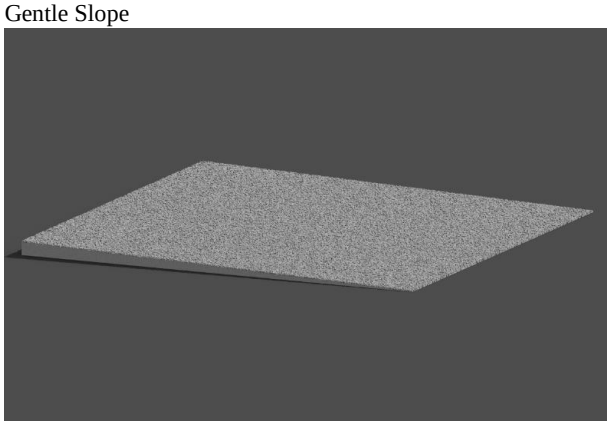
Flat



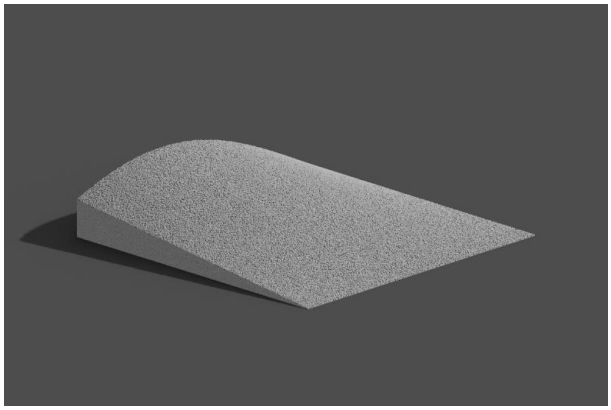
Hills



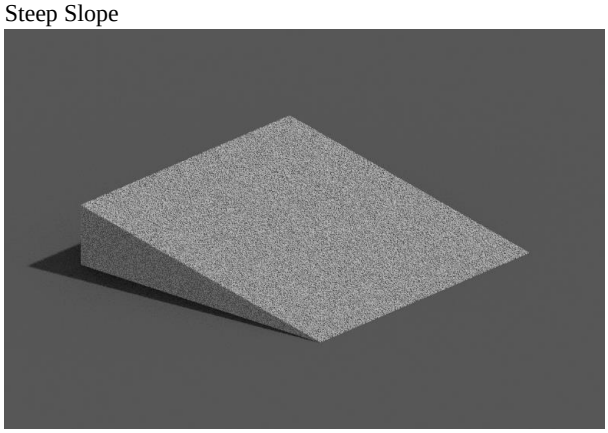
Gentle Slope



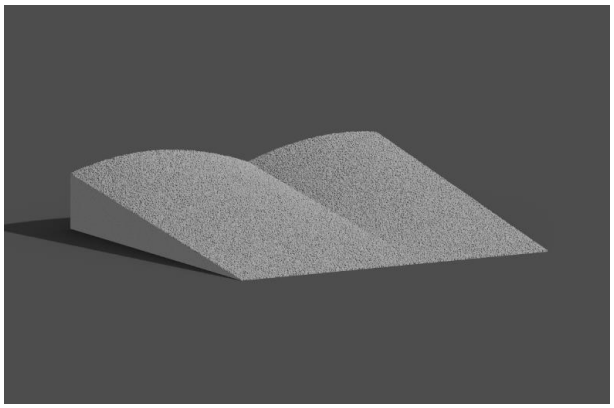
Contour Hill



Steep Slope



Contour Hill Overlap

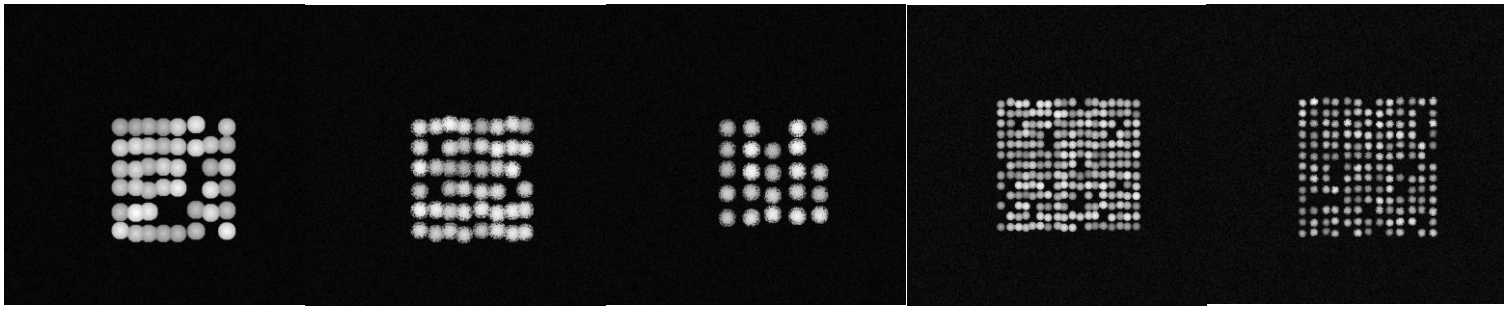


Appendix B: All Generated DEMs

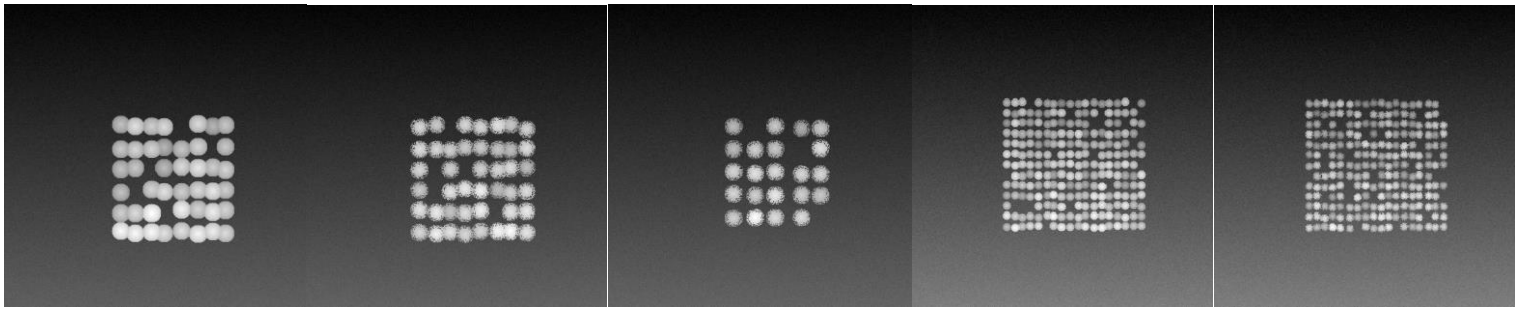
The five common variation types in section 6 table 1 are displayed for each landscape in the following order:

Simple, Complex, Complex and Spread, Simple and Small, Complex and Small.

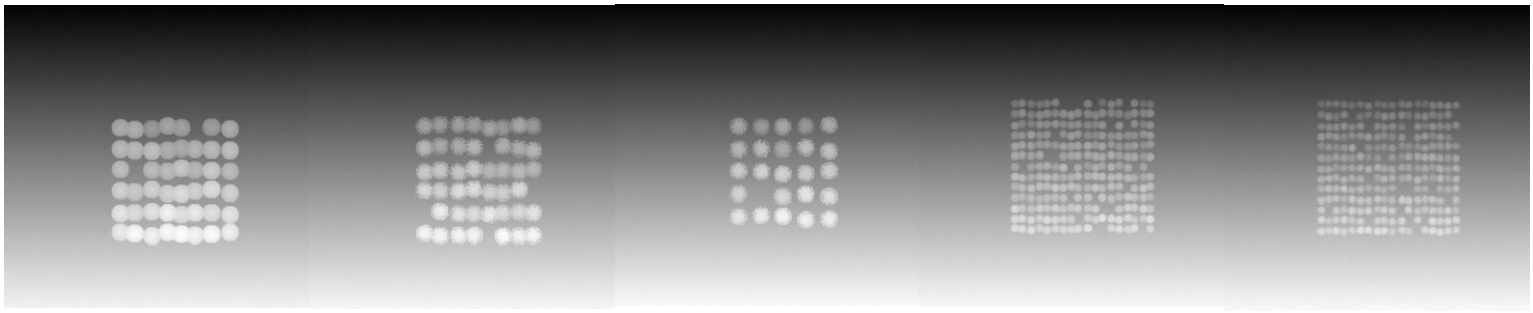
Flat:



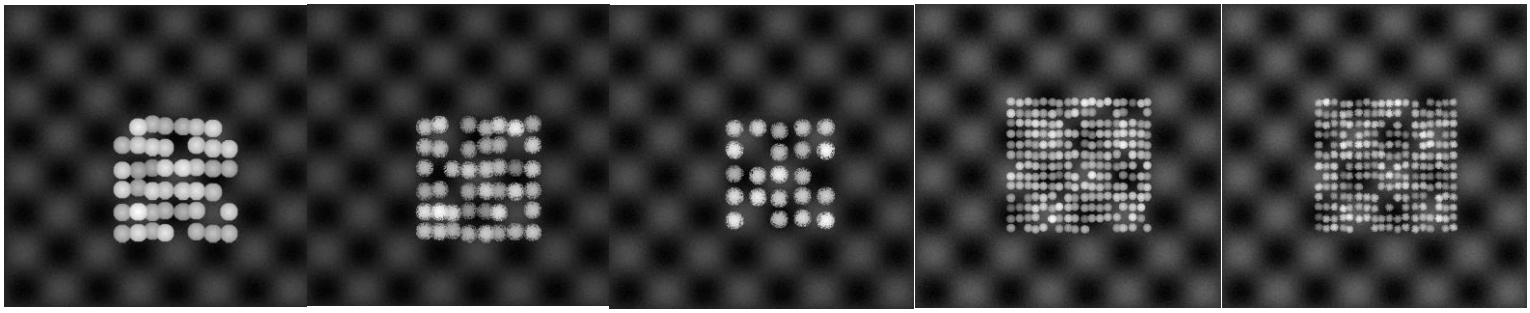
Gentle Slope:



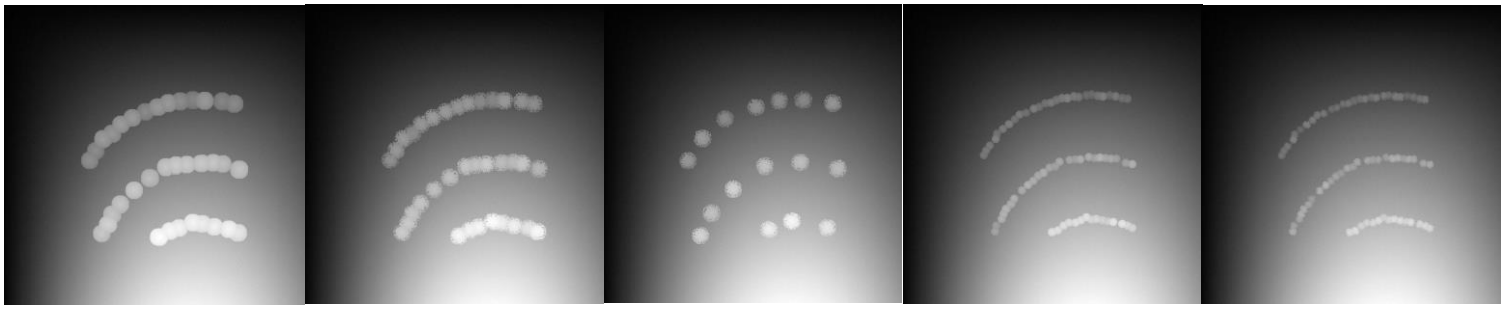
Steep Slope:



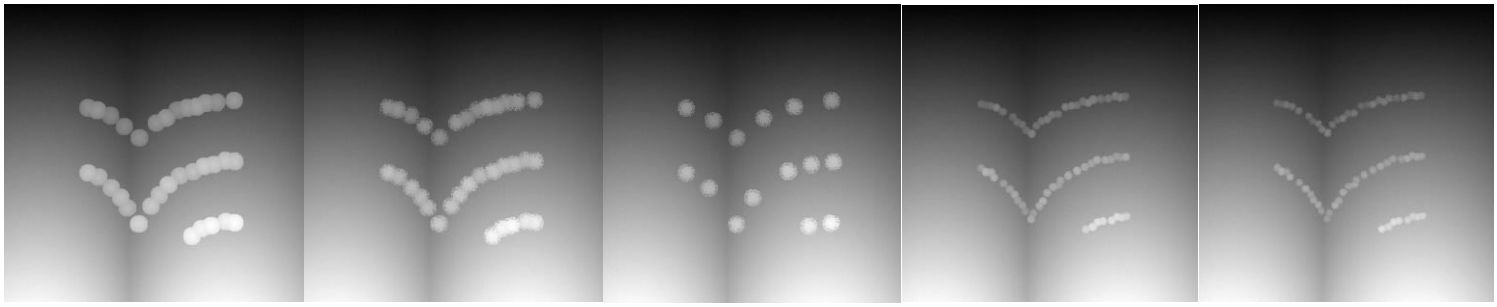
Hills:



Contour Hill:



Contour Hill Overlap:



Additional Experimental DEMS: