

Automated Model Creation from Aerial Photography

Greg Rafalski

gxr9935@rit.edu

Center for Imaging Science, Rochester Institute of Technology

Carl Salvaggio, Advisor

ABSTRACT

Creating a model, instead of just making measurements, is a common task in photogrammetry today. Methods can vary depending on the requirements, data availability, and type of scenery. The amount of user input is one major differentiating factor. The methods discussed in this paper align closely with a typical workflow for urban mapping from aerial photographs.

1. INTRODUCTION

The concept of making measurements from photographs actually predates the advent of the chemical photographic process. Like with all sciences, photogrammetry benefitted from countless technological advances over the years. Heavier than air flight was one such advance that allowed for much wider survey collection. Aerial photography, and photogrammetry, became an increasingly important source of data for city planning, agriculture, and the military. Before the airplane, such photographs had to be taken from tall buildings, balloons, and pigeons.

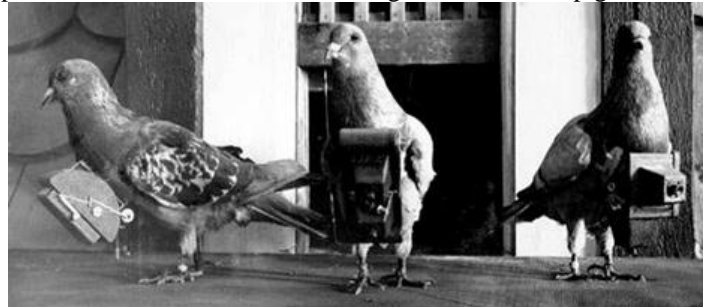


Figure 1 – World War One Aerial Reconnaissance Pigeons

Until approximately 1900, there were mapping and military uses for single-image photogrammetry, with stereo photographs were mainly used for novelties. Dirigibles and, later, aircraft made aerial photogrammetry practical. Multi-view imaging also became a reality with a stable imaging platform and the ability to plan and quickly execute flight lines over an area of interest. Cartographers no longer had to rely on completely manual surveys, and photographs could be taken from high altitudes without relying on mountains.

The methods for image acquisition and analysis continue to improve, but the basic idea is still to extract the best spatial information possible from image sets. Improvements came along in optics, film, flight planning, and analysis methodologies. The technologies exploited in this paper belong to analysis. This paper explores the ability for a computer to run a large number of calculations and produce a 3-dimensional model of the scene without user input. This is done with imagery from an uncalibrated camera and a relatively complex urban scene.

2. BACKGROUND

The data set used in this project comes from the Wildfire Airborne Sensor Program (WASP). The full data set was from a mapping mission over the city of Rochester with high overlap between images; about 70-90%. Seven of these images containing the Chester F. Carlson Center for Imaging Science building on the Rochester Institute of Technology campus were used. Each of these images comes with GPS/IMU data recorded with an Applanix system. A reduced resolution example of one of these images is shown below. The raw images are 4000x2762 pixels.



Figure 2 – Sample image used in this paper. The circled area is shown at full resolution at the bottom.

There are many uses for 3D models created from aerial imagery. Many techniques exist for creating these models requiring varying amounts of user input and metadata. As may be expected, the ultimate goal is a fully automated procedure that would produce high quality geoaccurate models. RIT CIS is one of many researchers on this topic and their efforts are documented here: <http://www.cis.rit.edu/~drn2369/>.

While still an active area of research, there are many companies actively using some of these techniques. Two products with large user bases are Google and Bing Maps. The Bing maps utility grew out of Microsoft Live Lab's Photosynth project which allows users to create virtual tours using photographs that are stitched together. As of 2008, Microsoft stated that an average sized city took two weeks to create using 5,000 CPUs.

Without metadata, the best model that can be produced will only be to scale. That is to say, the proportions will be correct, but it will be in an arbitrary coordinate space. In order to create a metric model from which measurements can be obtained, camera data such as the focal length and the pixel pitch of the sensor are required; or ground truth calibration. Using a process called bundle adjustment, camera positions can be calculated and the scale information provided by the camera parameters allows for accurate measurements. Adding in the camera position and orientation from a GPS/IMU unit will allow that metric model to reside in real-world latitude and longitude.

3. METHODOLOGY

A typical workflow for model creation, commonly called “Structure from Motion” is as follows:

- 1) Find point matches between each of the images. This is commonly performed with David Lowe’s Scale Invariant Feature Detection (SIFT)[8].
- 2) Execute bundle adjustment to refine the relative position and orientation of each of the cameras. Commonly performed with Bundler.
- 3) Run Clustering view for Multi-View Stereo and Patch-based Multi-View Stereo (CMVS/PMVS) to find and refine the 3-space location of as many points from the images as possible.

Running this workflow will result in a point cloud such as can be seen in Figure 3, which can be interpolated into a surface as in Figure 4.

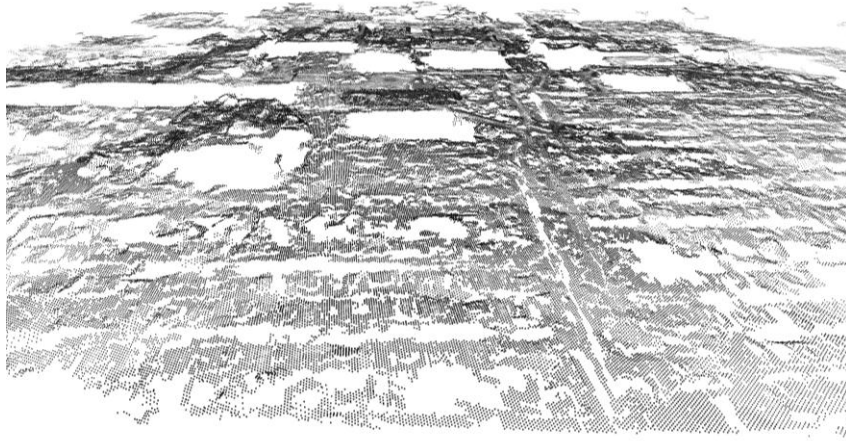


Figure 3 – PMVS Generated Point Cloud

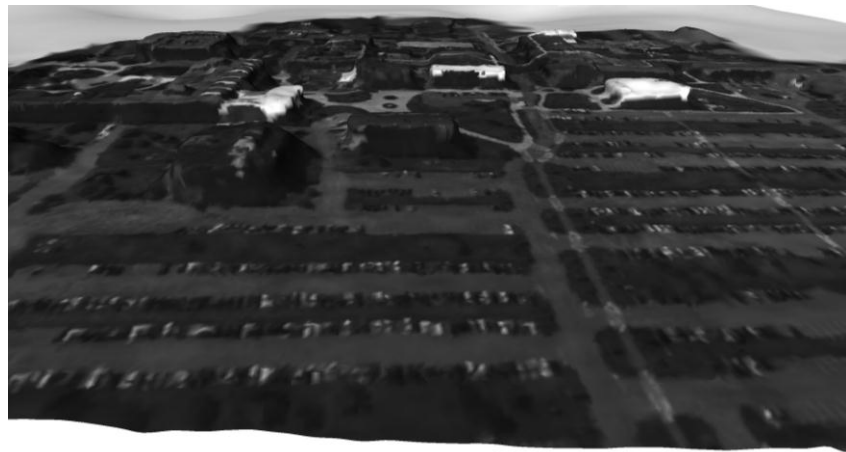


Figure 4 – Point Cloud Interpolated to a Colored Surface via Meshlab

Many algorithms for creating the final model start at the point cloud and attempt to refine edges and planes. This algorithm proposes an alternative workflow in which the edges are found before the point correspondence of step 3 above. This change ruled out using a pre-compiled CMVS/PMVS toolbox since that will look for all possible points and not just specified ones. The new workflow then looks like this:

- 1) Use camera metadata to determine image corner coordinates.
- 2) Use corner coordinates to determine which pairs overlap
- 3) Run Edge detection and convert the resulting binary image to a set of vertices representing continuous line segments.
- 4) Find point matches between each of the images using SIFTGPU.
- 5) Calculate the fundamental matrices for each pair.
- 6) Use the fundamental matrices to find transformation matrices that can be used to rectify each pair and save them to disk. There will now be a stereo pair for all combinations of overlapping images.
- 7) Transform the coordinates for all of the line segments found in step 3 to the new rectified coordinate system shared by each pair.
- 8) Go through the list of transformed points for each image and search for the corresponding point in its stereo pair.
- 9) If the point is found, use the photogrammetric parallax equation to determine its elevation.
- 10) Since the same point is likely to lie in multiple images, reduce errors by averaging the calculated elevations.
- 11) Use the camera metadata, the now-known point elevations, and trigonometry to determine latitude and longitude for all the points.

All of this is done in Matlab, except for SIFTGPU, which is a C++ plugin for Matlab called YASIFT.

A more detailed explanation of each step follows.

Step 1 – Determine Corner Coordinates

The camera metadata was provided as: Latitude (degrees), Longitude (degrees), Elevation (meters HAE), Roll (degrees), Pitch (degrees), and Yaw (degrees). Roll, pitch, and yaw were relative to a flat northerly flight path. The camera focal length was given as 55 mm, and the sensor pixel pitch was given as 9 microns.

From there, vectors were created that corresponded to the bore site (center) of the image, and all four of its corners. The area of interest was known to be relatively flat, so a single ground elevation was used, and the vectors were projected down until a point of intersection was found.

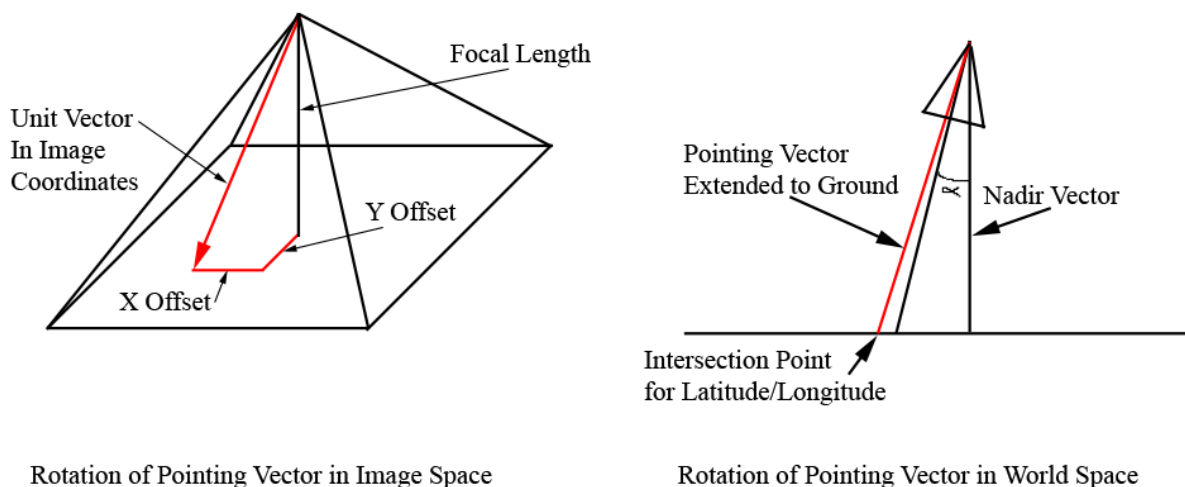


Figure 5 – Illustration of Vector/Plane Intersection

Step 2 – Find Overlapping Pairs

Step 2 was unnecessary for the 7-image dataset used since they all overlapped, but adding this step would allow for extending this code to much larger datasets where the majority of images won't overlap. In that case the millions, possibly billions, of calculations performed in step 4 would be saved.

Because the image footprints on the ground will most likely not be perfect rectangles, it is not a straightforward calculation, and is similar to the classic “point in polygon” problem. The simplest solution was to calculate the intersection of each possible pair's edges and if an intersection was found, the two shapes were considered to overlap [5].

Figure 6 below shows the image footprints on the ground, calculated with the method described in step 1. The coordinates at the corners are what is used to calculate image overlap in this step. The red circles represent the bore sight of the camera at each location.

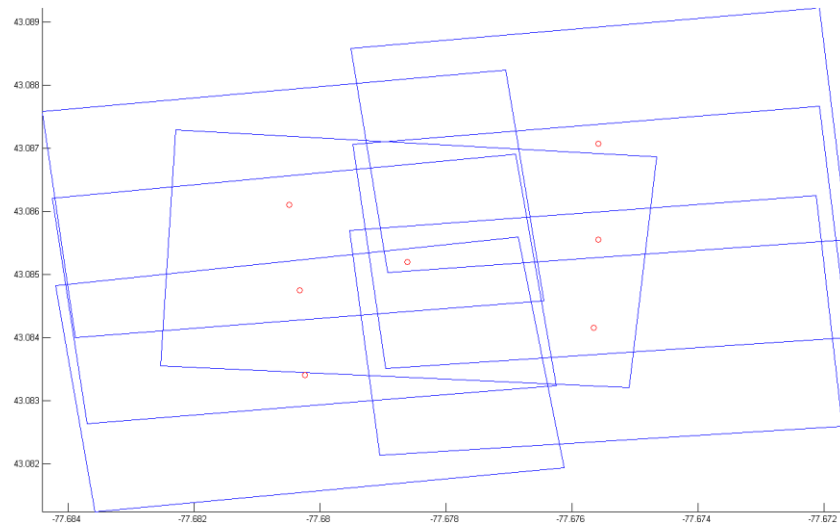


Figure 6 – Image Footprints

Step 3 – Find a List of Line Segments

This is the most unique addition to the workflow. In this step the possible edges to search for are found before searching for point correspondences. There are three benefits to this. First, finding a list of vertices will drastically reduce the number of correspondence searches performed, and computation time required, in step 8. Second, finding corners and edges in a 2D image is simple. Searching for distinct edges and corners is a difficult task to perform on a point cloud. Any refinement done on the point cloud to reduce calculation errors (i.e. to make planes flat) will also result in rounded edges. Look closely at figure 4 and note there are no sharp edges. Third, storing a list of all the linked lines in an image is essentially building the wireframe model. The resulting list only needs the elevation for each vertex.

This is performed by first creating a binary edge image with the Canny edge detector. The edge pixels are then iteratively searched for adjoining pixels and grouped. Any group that contains less than a specified number of pixels is removed[6].

Once each group is identified, the algorithm attempts to simplify the groups. The beginning and end points are easily set. From those end points, a single line is constructed, and each of the edge pixels is tested to see if they fall farther away than a given tolerance. If they do, a vertex is added to the line, nearest to the outlying pixel. The new set of lines is tested again, and the process is repeated until all of the original pixels fall within the specified distance to the new lines. Each of these line segments is stored, one by one, into a cell array for individual retrieval later. Below is an example of the image from Figure 2 with its detected line segments with a minimum length of 15 pixels.

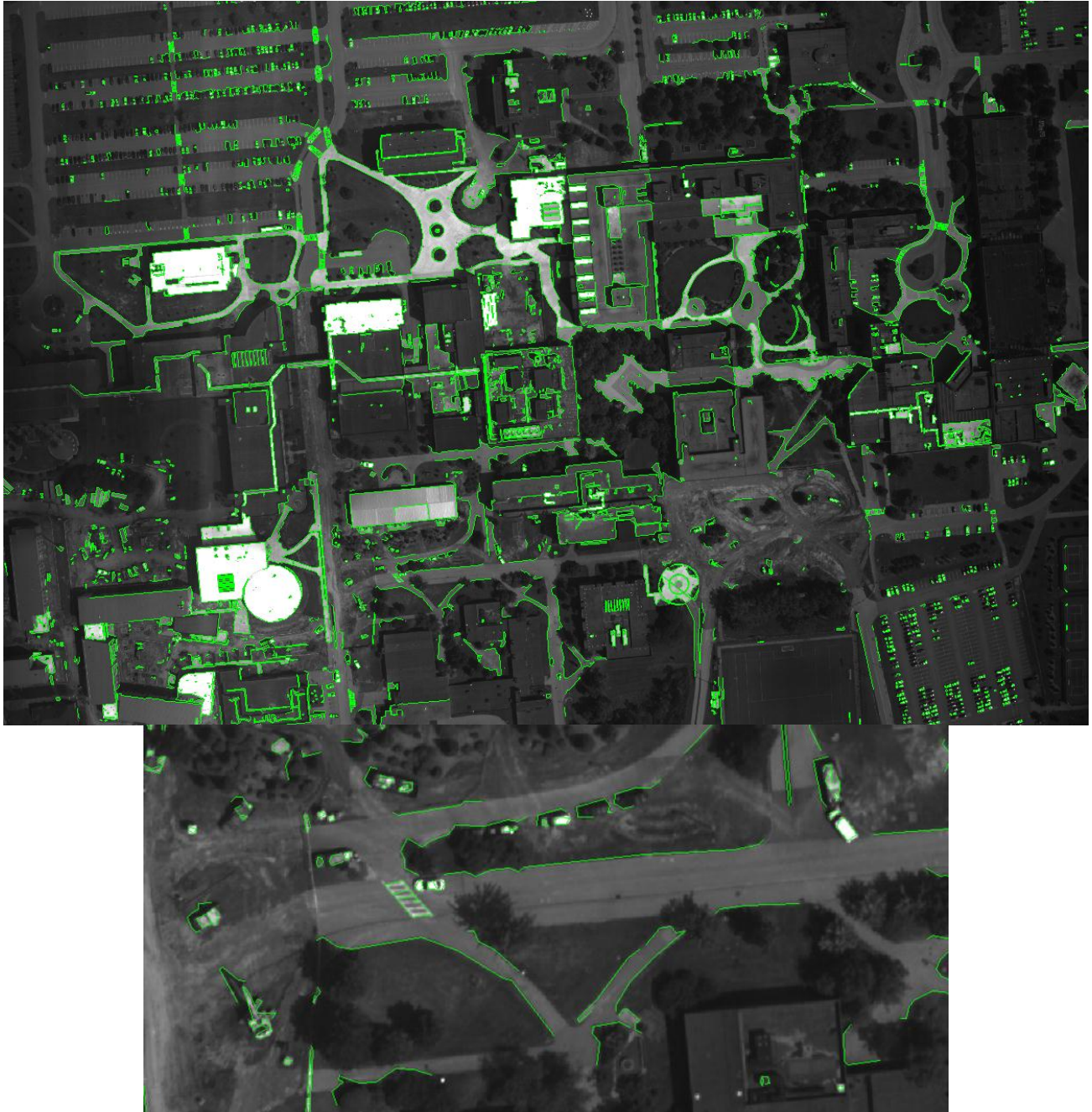


Figure 7 – Lines created from edge detection. The lower portion shows the same enlarged portion as figure 2.

Step 4 – Find Initial Point Matches

David Lowe's Scale Invariant Feature Transform (SIFT) is a wonderful tool that is used for video stabilization, feature tracking and image matching. SIFT will characterize a pixel by looking at its surrounding area and calculate 128 different values that represent that pixel. It will do this for all of the pixels in all of the images, and pick which pixels correspond to each other. Because of the way the characteristics are chosen, it can find matches regardless if the images are from different angles, or sizes. Finding these corresponding points will lay the foundation for the remaining portions of the workflow.

SIFT works great, but its speed is greatly increased when calculations are done in parallel on the graphics card. Changchang Wu developed this GPU implementation, and Parag Mital ported it over for use in Matlab [7]. This step can be done without GPU processing, but the increased speed is too good to pass up. There are two factors to consider when performing SIFT on a graphics card: 1) Adding more GPU cores will mean faster results, but 2) Adding more memory on the graphics card will mean more accurate results. Consideration number 2 is true because if the code detects that it will not be able to store all of the information it needs into a contiguous block of memory, it will reduce all of the images to half resolution.

Step 5 – Create Fundamental Matrices

The fundamental matrix describes the epipolar relationship between two images. If you pick a pixel in one image, you can use the fundamental matrix to find a line along which the same point lies. Calculating the fundamental matrix requires at least 7 matched points between the two images. These matched points come from SIFT in step 4.

SIFT works well, but it will find some mismatched pairs. To correctly calculate the fundamental matrix, these spurious matches must be minimized. The technique used here is RANdom SAmple Consensus (RANSAC). The details of RANSAC will be left out, as this is a common step in model creation[1]. The entire fundamental matrix calculation was performed in this case with the Computer Vision toolbox for MatLab 2012. The version of MatLab is specified because many of the functions in the Computer Vision toolbox changed from 2011. The two major changes affected the point matches in this workflow: The coordinate system switched from x-y to row-column, and went from a row vector to a column vector. The original x-y coordinate system allowed for negative values, where row-column is more typical of a MatLab representation of a matrix.

Step 6 – Image Rectification

With the fundamental matrices now known, the transformation matrices for each image pair can also be calculated. The transformation matrices will warp the two images such that all of the parallax is in the horizontal direction in the two images. The MatLab command for this is “imtransform”. This is how stereo pairs are formed, and these images could now be displayed together as a red/cyan anaglyph. An example showing the same image as Figure 2 and one of its stereo mates is shown below.

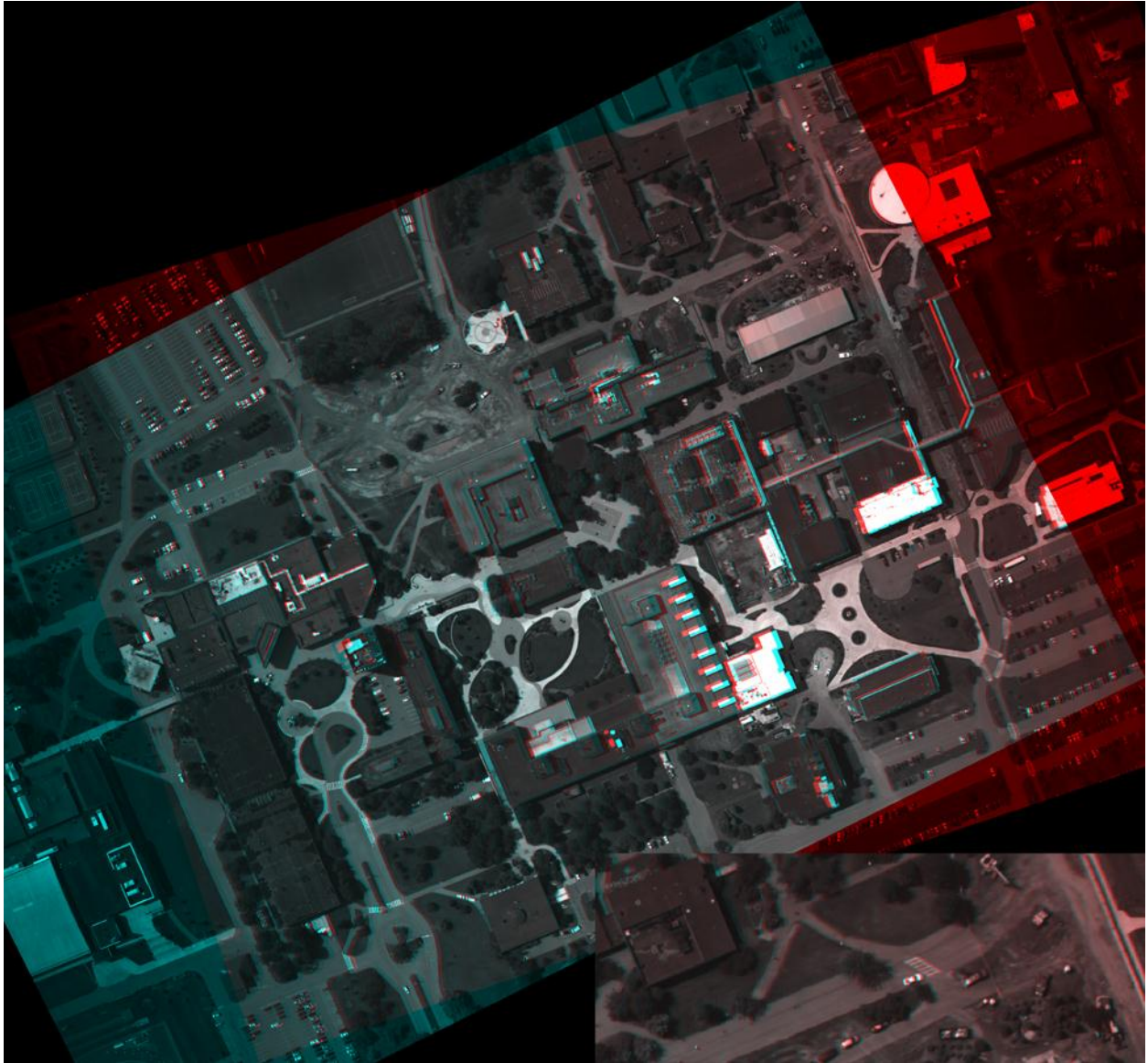


Figure 8 – Stereo pairs in a red/cyan anaglyph. Inset at lower right shows the same enlarged portion as figure 2

Step 7 – Transform Line Segments

The same transformation matrices used to create the stereo pair can also be used to move the lines found in step 3 to the stereo coordinate system. This makes use of the “tformfwd” command. In the background, MatLab performs the transform using the homogenous form of the original X-Y point locations.

This transform will put each of the line vertices from step 3 into a shared, but arbitrary, coordinate system. The bounding box containing the two images in this coordinate system was saved as an output of the “imtransform” function. For use in calculations, the transformed line segments must be shifted to the pixel row-column system. That is a simple addition/subtraction applied equally to all of the output values, and was used to overlay the lines in Figure 9.

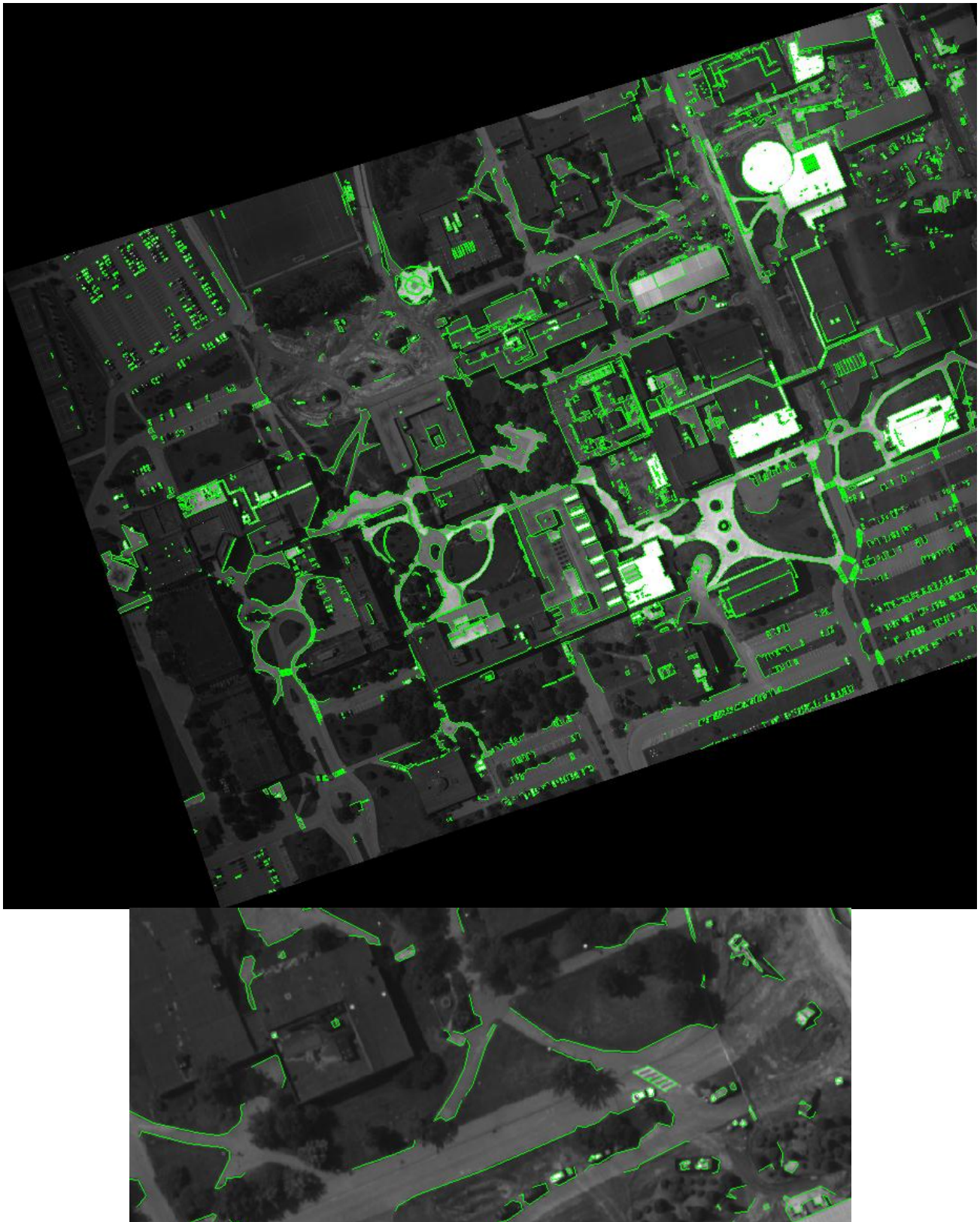


Figure 9 – Edge lines overlaid onto the rectified image. The lower portion shows the same enlarged portion as figure 2.

Step 8 – Point Correspondence for Line Segments

With all of the lines in the shared coordinate system of the stereo pair, all of the necessary information is now present to search for the corresponding points in the second image. Because the rectification step moved the parallax into the horizontal direction, the search area has been drastically limited. The matching point in the mate image should lie in the same row as the input point of the base image.

As mentioned previously, because the point search does not run through the typical workflow, the more robust CMVS/PMVS tool cannot be utilized. Instead a customized search algorithm was created.

Each vertex of the detected lines is taken individually. A 5x5 kernel using the surrounding pixels is created from the base image. Then, using the mate image, a strip consisting of the 7 rows centered on that of the point of interest is created. The strip is 7 pixels tall instead of 5 to allow for the possibility of error in parallax alignment.

The corresponding point in the mate image could lie anywhere along the strip, but only extremely tall objects would have a large parallax in aerial images containing as much overlap as these images share. Knowing this, the actual search area was limited to 101 columns surrounding the point of interest, 50 to either side.

The search is an iterative double loop in the MatLab code; one for the three searched rows, one for the columns. The kernel is slid across the strip and a score comparing it to the 5x5 pixels at that point in the strip is stored. At the end of the loop, the best unique score above a threshold is deemed to be the location of the matching point.

The actual comparison was attempted in two ways. The first was to take the dot product of the two 5x5 blocks. This would be similar to the Spectral Angle Mapper, which is a classification method used in multi and hyperspectral imaging. The resulting output for each comparison is the angular distance between the two blocks in 25-dimensional space. This was the preferred method, because it can make a comparison even if the images were taken with different illumination.

The less ideal method that was used in its place was a straight pixel-for-pixel difference between the two blocks. To help account for possible illumination differences, during the rectification step, the mate image was histogram matched to the base image. Despite its simplicity, this method resulted in a better set of matched lines. As can be seen in Figure 10 though, it still was not successful to a useful degree.

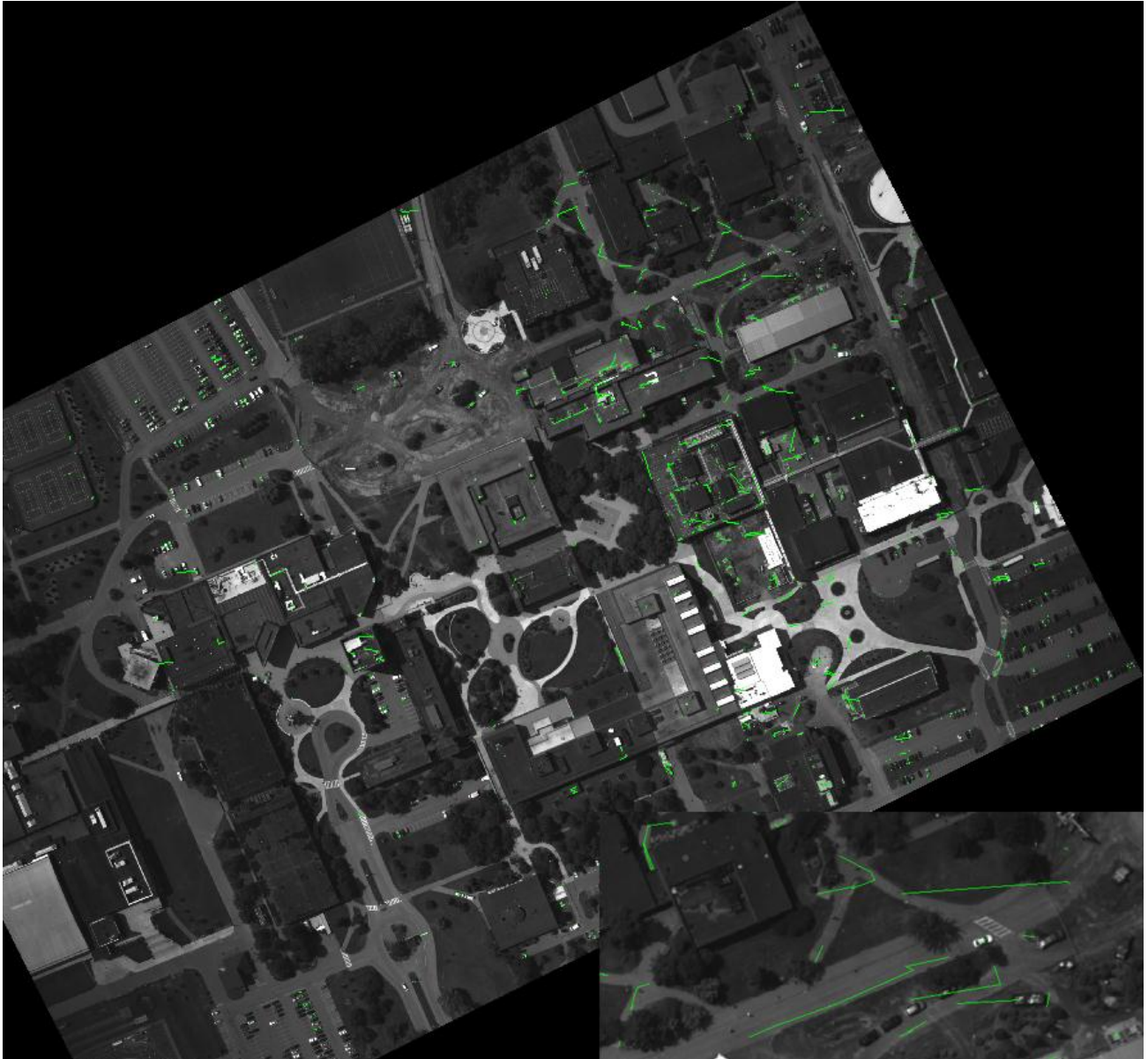


Figure 10 – Matched Lines Found in the Mate Image

Step 9 – Calculate Elevation

For the points that were found, the photogrammetric parallax equation was used to calculate the elevation at that pixel.

$$h_{pt} = \frac{H * P_d}{P_a + P_d}$$

Equation 1 – Height from Stereo Parallax

Where:

H is the flying height in meters above WGS84 datum. H is taken as the average of the two camera heights.

P_d is the differential parallax; the pixel difference between point locations in the image pair.

P_a is the absolute parallax; the pixel difference between the principal points of the image pair.

h_{pt} is the height in meters of the point of interest above the WGS84 datum.

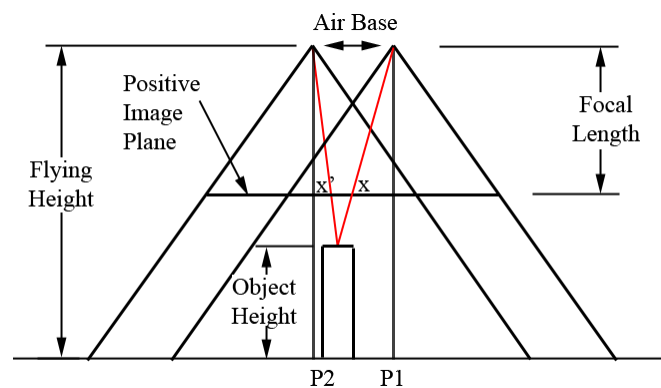


Figure 11 – Diagram explaining parallax components. P_a=P₂-P₁, and P_d=x'-x.

Step 10 – Reduce Elevation Error

Once the point elevation is known, the same steps used to calculate the image corner coordinates can be used to find the point latitude and longitude. Because of the high amount of overlap between images, each point of interest will be found in more than one pair of images. Since it is unlikely that the heights calculated in step 9 will all be exactly the same, they are averaged together to minimize the error.

Step 11 – Calculate Coordinates

Now that each point has only one height, the camera pointing vectors are created from the metadata and the row-column location of the pixels. The intersection of the pointing vector and a plane at the height of the pixel determines the latitude and longitude.

For any points that were not found in the previous steps, the line segment containing it is identified and all points from that line segment are removed.

All of the latitude, longitude, and elevations are stored grouped by their original segments in a matrix and saved to disk.

4. ANALYSIS

This workflow did not perform well. The resulting model contained only enough structure to possibly be recognized as the RIT campus when viewed straight down.

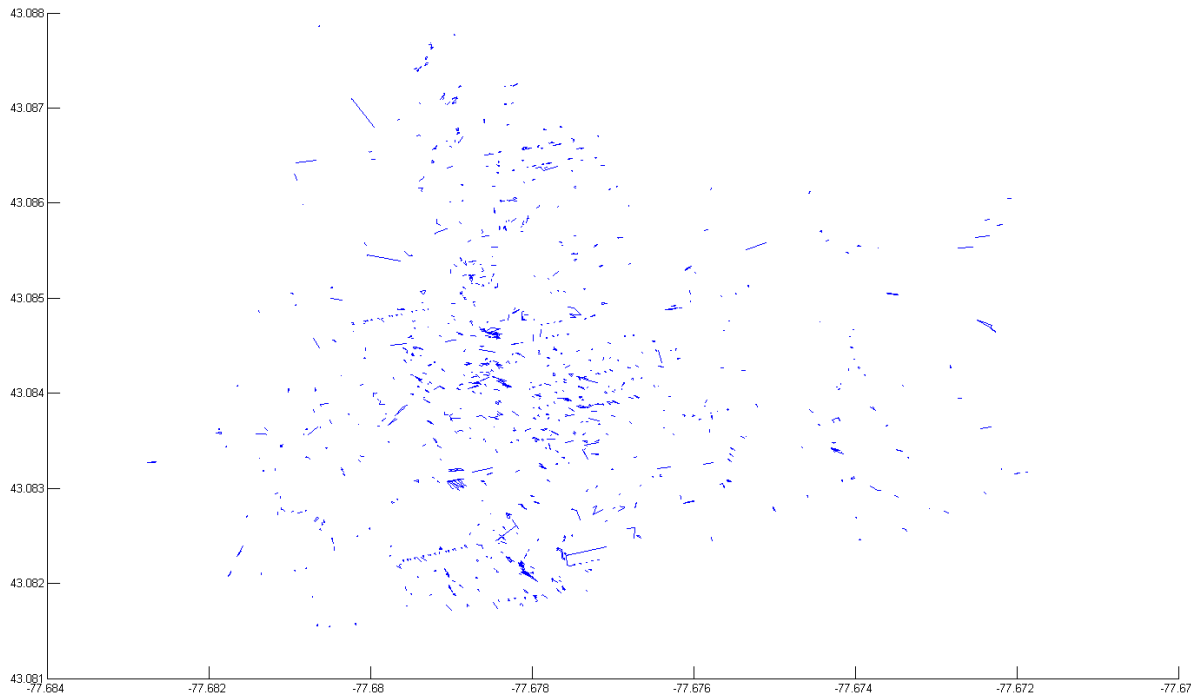


Figure 12 – Resulting Model Viewed Top-Down

When the model is rotated to be viewed from the side, it becomes clear that there is no meaningful structure. The lines all appear to point back at the cameras, when the majority should all be purely horizontal or vertical. This is an indication that either the parallax equation, or point correspondence steps result in large errors.

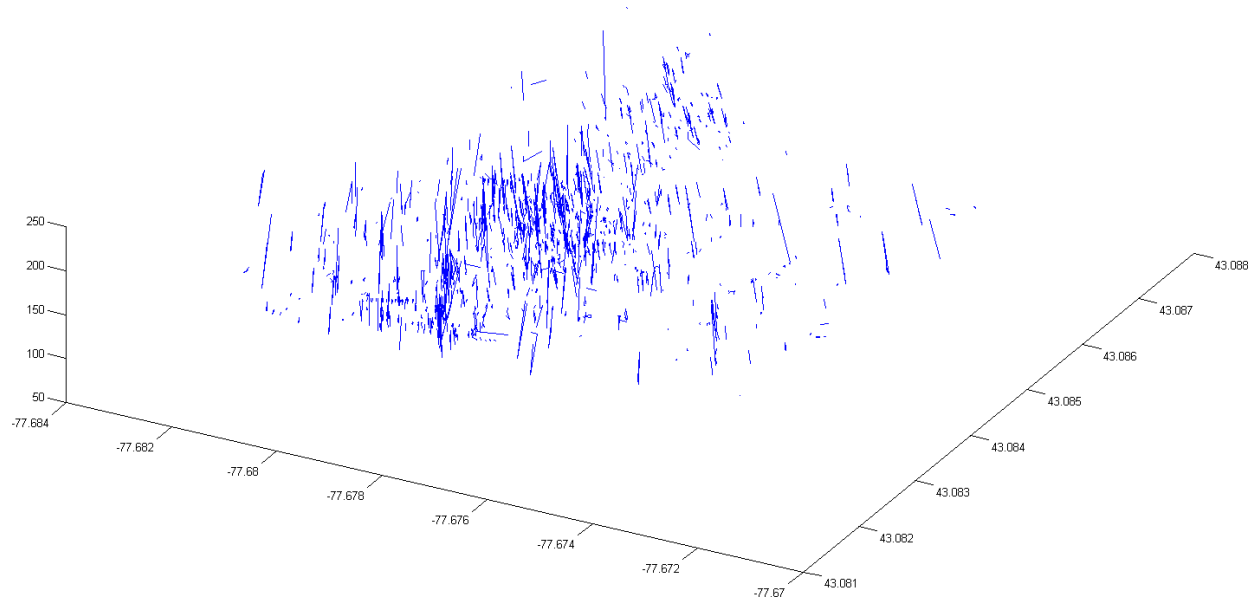


Figure 13 – Resulting Model, Rotated View

There is one major cause for the poor performance shown. The biggest source of error appears to be in the point correspondence of step 8. While Figure 10 clearly showed there were many points not matched between images, it may not be immediately obvious the points it did find were unreliable. Look at Figure 14 below. This is a histogram of parallax (in pixels) of all the points for which the workflow found matches.

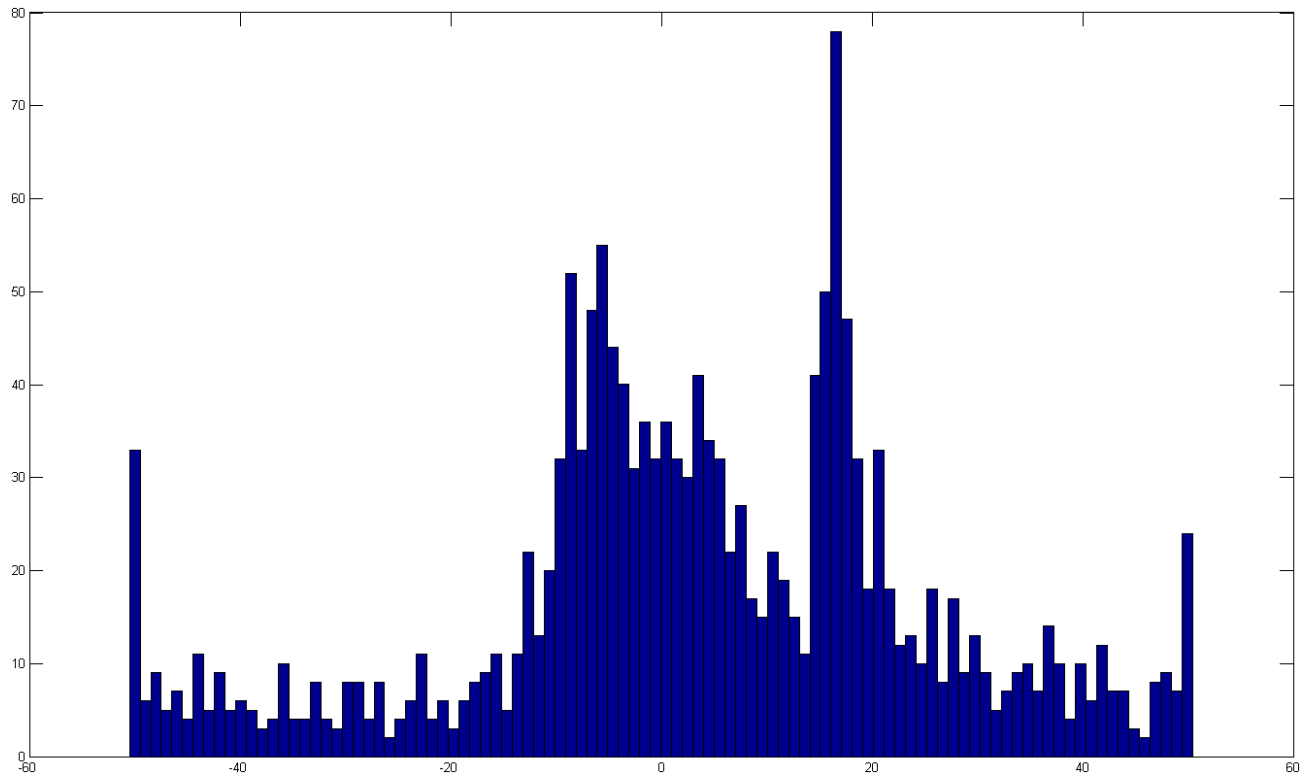


Figure 14 – Parallax for the Points Found

This is a clear indication that the point matching algorithm is insufficient. For the image pair studied in this paper, the maximum parallax that could be found manually was 36 pixels. Since the majority of lines would have been at ground level, it makes sense that there are so many near-zero parallaxes. It is also believable that the peak near 20 pixels could be for buildings. The peaks at ± 50 pixels and the number of counts above ± 40 are wrong. It would appear the correspondence algorithm has a tendency to favor the extremes of the range as the match.

5. CONCLUSION AND IMPROVEMENTS

This workflow needs improvement. As-is, it is not useful for creating models but it does have potential. Most other geo-modeling workflows will start the model creation at the point cloud, like in Figure 3. While CMVS/PMVS does an excellent job of creating a point cloud, turning a point cloud into a wireframe is no simple task. Even determining what points in the cloud make up the edge of an object is difficult because some points may be missing, or there may be slight errors.

This workflow attempted to save calculation time by changing perspective: pre-processing the images and only searching for the known vertices of edges. Perhaps a better method would be to save the pre-processed edges, and still use CMVS/PMVS. PMVS can save, as part of its output, a list of the original pixel coordinates used to create each point in the cloud. If those points can then be mapped back to the line segments found in the edge detection, a wireframe model could very easily be built. Given more time, that would be a good next step to try.

Whether or not CMVS/PMVS is run, a bundle adjustment algorithm such as Bundler should definitely be implemented. Bundle adjustment is a required step for CMVS/PMVS. For the workflow in this paper, the benefits would be: 1) The points found by SIFT would be refined, leading to better fundamental and transformation matrices, and 2) The camera positions and orientations would more accurately be represented in the photogrammetry equations.

More accuracy would also be gained by introducing digital elevation maps (DEM), such as from the Shuttle RADAR Topography Mission (SRTM). This workflow used a single ground elevation, chosen at the location of the CIS building, as the ground elevation for the entire scene. This is a shortcut that can work for small, flat areas but will not work for general use. To drop the sides of the building to ground, an accurate ground elevation must be known. Using the method of ray/plane intersection of the corner coordinate section would require an iterative search. Each new corner coordinate answer would also give a new elevation. The loop would end when the change in latitude/longitude was smaller than the post spacing of the DEM.

ACKNOWLEDGMENTS

Pat North was a big help with this project, as always.

REFERENCES

1. Hartley, Richard, and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge, UK: Cambridge UP, 2003. Print.
2. Schott, John R. "Section 2.2 Quantitative Analysis of Air Photos." *Remote Sensing: The Image Chain Approach*. New York: Oxford UP, 2007. Print.
3. Lengyel, Eric. *Mathematics for 3D Game Programming and Computer Graphics*. Hingham, MA: Charles River Media, 2004. Print.
4. Moffitt, Francis H. *Photogrammetry*. Scranton: International Textbook, 1967. Print.
5. Nilsen, Jan E. "Jan Even Nilsen's homepage." Evenside. 27 Mar. 2007. Nansen Environmental and Remote Sensing Center. 01 Aug. 2012 <<http://www-2.nersc.no/~even/>>.
6. Kovesi, Peter. "MATLAB and Octave Functions for Computer Vision and Image Processing." Peter's Functions for Computer Vision. 27 Sept. 2010. The Centre for Exploration Targeting at The University of Western Australia. 01 Aug. 2012 <<http://www.csse.uwa.edu.au/~pk/research/matlabfns/>>.
7. Mital, Parag K. "Siftgpu - Matlab (Mex) port of SiftGPU." Siftgpu - Matlab (Mex) port of SiftGPU. Oct. 2010. Google Project Hosting. 01 Aug. 2012 <<http://code.google.com/p/siftgpu/>>.
8. Lowe, David. "Distinctive Image Features from Scale-Invariant Keypoints." *International Journal of Computer Vision* 60 (2004): 91-110.