Interactive Visual Analytics in Support of Image-Encoded LiDAR Analysis

Todd Eaglin, Xiaoyu Wang, and Bill Ribarsky, University of North Carolina at Charlotte

Abstract

LiDAR data is a significant resource for identifying similar geospatial features in urban planning, land use analysis, emergency response, and other applications. Traditionally LIDAR is analyzed through manual process, which is a very challenging task due to the need to identify similarities over a growing size and complexity of data. To alleviate this challenge, we designed and developed a GPU-powered visual image analytics system to handle this operation at large scale. Our system encodes human-freeform-LiDAR selection into 2D images through an autonomous image analysis process that matches selected areas of interest. To ensure the system's practicality in handling hundreds of stitched LiDAR patches, we have scaled up our algorithms through a series of parallelized GPU processing, analyzing, and encoding methods. We conducted informal user studies to assess the utility and usability of the system.

Introduction

LiDAR, a remote sensing technology, has become a crucial instrument for us to extract and monitor up-to-date, accurate geographic information about areas of interests. Professionals involved in a variety of industries and applications (like disaster management) are increasingly utilizing three-dimensional sources of information to create photorealistic 3D visualizations, extract 3D features and export products to geospatial tools to help understand geospatial features in both urban planning [5] and emergency responding [7].

Especially in emergency management and disaster response, LiDAR presents a more effective data source to locate weak points. For instance, Kwan et al. [12] used about 50 million LiDAR points to identify blockages in the transportation network caused by Hurricane Katrina. Whereas, Clasen et al. [6] further examined the potential areas where LiDAR data can be used for disaster and emergency planning, including understanding how slope of the terrain affects landslides, analyzing tree maps for fire disasters, and more importantly finding suitable areas for rescuer deployment and helicopter landings.

Zerger et al. [21] examined the current technical limitations with GIS in emergency situations. They discovered that existing systems were unable to provide details in real-time due to limited processing power and the size of data. Database queries were extremely time consuming and unsuitable for real-time planning.

While the utility of LiDAR can be significant, analyzing it effectively over a larger area still imposes a significant challenge. Based on our collaboration with North Carolina Department of Transportation, we observed the following challenges:

- *Scale* - Due to the nature of LiDAR it can grow exponentially when examining larger spatial regions, at higher resolutions, and at different points in time. The files can be hard to manage; even loading and viewing them can be a problem. Collecting different sources of data, processing them with a myriad of different tools, and converting LiDAR coordinate systems can also be quite time-consuming. Kwan et al. [12] used a data set of about 50 million LiDAR points, but they do not mention the scalability of their work and how quickly it takes to process.
- *Interactions* - There is a need for being able to analyze and manage LiDAR data in realtime. Otherwise, it's difficult to do comparative feature analysis or disseminate the results to responders. There is a need for interactive tools for analysis and searching to solve these problems.
- *The lack of searchable content*. To be able to fully and effectively analyze the LiDAR data, the system must support: finding areas of similar elevation; finding buildings of similar size or type; finding road networks and their structures (e.g., bridges, embankments, ditches, etc.). In addition one should be able to find special relationships such as roads near areas of high vegetation; roads near coastal areas.. . .

To alleviate these challenges and make LiDAR analysis more interactive and effective, we present a GPU-powered visual image analytics system that processes, analyzes, searches LiDAR data at scale. Our system encodes human-freeform selection into 2D images through an autonomous image analysis process that matches selected areas of interest. By leveraging the ability of users to create meaning and relationships in the data, our approach becomes the starting point for assisting them in understanding these datasets for a multitude of problems. Our work focuses on how a user can be helped to identify features and find similar ones via intelligent searching amongst millions and millions of LiDAR data points.

As an example use case a severe hurricane hits a coastal area causing significant damage as well as changing the coastal landscape. Any previous LiDAR data would not be accurate, therefore new LiDAR would have to be collected and analyzed. Rescue teams would need to find appropriate areas for operations and safely landing helicopters to provide relief. In order to find such areas in newly mapped LiDAR data it would take an exhaustive amount of time. Our process and tools aim to solve these issues by allowing users to search LiDAR data sets in a free-form manner to find features. First using our process we can analyze any new LiDAR data sets very quickly using our GPU architecture. They can then use our visual analytics tool to find the most suitable location for their needs.

The rest of the paper is structured as follow: in section 2 we discuss in depth why we choose to use image analysis and the benefits of using that approach. We cover the computational and

space complexity of an image analysis approach. We also discuss how it ties in with visual analytics and the overall goal of incorporating a user's knowledge to drive the analysis. In Section 3 we present our method for encoding details from LiDAR data into a 2D image. We discuss the algorithms that we used in the context of GPU computing as a means to accelerate user interactivity. Our Visual Analytics system is presented in Section 4; we present use cases an informal user study in Section 5. We wrap up with conclusions in Section 6.

Related Work

In this section we discuss our motivation for tackling this problem and why we have chosen image analysis and how that plays into our larger goal of allowing real time interaction and analysis of LiDAR data. Since our approach is multidisciplinary, we cover the recent work in both image analysis with relation to LiDAR visualizations. For example, Richter et al. [15] examined a novel use of improving LiDAR visualization using GPU acceleration and out of core rendering. Butkiewicz et al. [4] explored the combination of visual analytics and LiDAR to detect temporal changes in a city environment. The authors looked at finding the differences between different time periods of LiDAR data using the 3D geometry created from a LiDAR point cloud. Using the computed geometry provided more accuracy than a point cloud using a grid approach with nearest neighbor and outputting that as a 2D image. In our approach we create 3D geometry to provide more accuracy and details, but we return to using 2D images by encoding the geometry back into an image. We do this for several reasons that we discuss in Section 3.

Butkiewicz et al. [4] did not mention anything about scalability since they compared unique spatial regions in time and did not compare any deltas between regions. This resulted in a much smaller number of overall comparisons. Our approach allows a user to analyze two different regions regardless of time. The authors concluded that there is still much work to be done in the area of algorithms for handling issues of matching or issues with users not being able to define filter metrics. Blaschke et al. [2] surveyed the existing research in image analysis for remote sensing; they outlined some of the benefits to using image segmentation.

A lot of work has been done on classification in LiDAR data and extraction of specific features like buildings, as described by Hermosilla et al. [10], who utilized image thresholding using two specific values. The first threshold value is the minimum height of a particular building and the second value indicates the presence of vegetation. Their results were quite strong, using image thresholding, but this was only when the thresholding values were adjusted to the specific type of environment in the LiDAR data. We expand on this work in several ways. Firstly, we add more thresholding values by converting a LiDAR point cloud into a RGB image from a triangulated 3D mesh. During this process of creating a new image from the LiDAR point cloud, we encode high quality information about unique traits, but more importantly we encode them in a way that they are properly adjusted so thresholding can be done more efficiently and with a clearer understanding of the result. We discuss this in detail in Section 3.

Why Image Analysis?

Our data set consists of about fifty Digital Surface Model (DSM, ground elevation plus all features, such as buildings, on it) LiDAR patches along the coast around Wilmington, NC. Each patch is made up of X, Y, Z locations that denote the geographical position and the elevation of the terrain at that position. Each patch in the data set contains between 1 million and 4 million points. In total we have approximately 100 million points.

Computation Evaluation

Apprx 1 million points per patch	1 Patch	5 Patchs	10 Patchs	25 Patchs	50 Patchs
CPU - Single Thread	1m 28s	7m 20s	14m 50s	32m 30s	1h 6m
CPU - Multi Thread (12 threads)	1m 28s	2m 3s	3m 39s	8m 53s	19m 45s
GPU	18s	43s	1m 20s	3m 16s	6m 34s

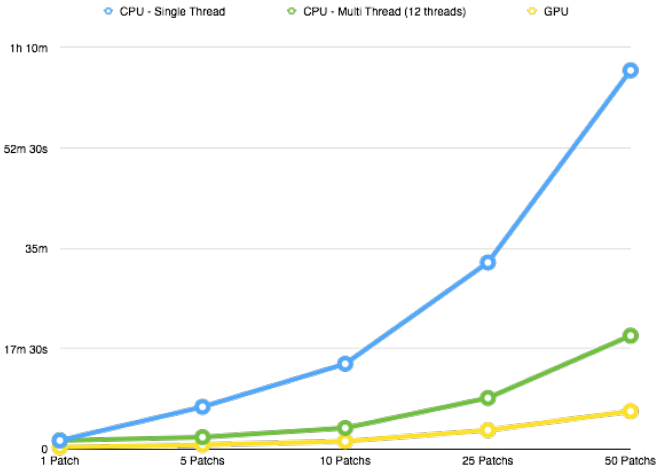

Figure 1. *Timing results using three different metrics. The first is a single thread CPU implementation. The second is a multi-thread CPU implementation. Lastly, is our implementation with a GPU. The benchmark was performed on a 3.5 GHz 6-Core Intel Xeon E5, and a GTX 960 Nvidia GPU.*

To ensure the real time interaction and analysis of LiDAR data, we turn to research in image analysis. Primarily, we utilize image analysis to handle both the space and time complexity of LiDAR data and automation. Image analysis allows us to search very quickly amongst massive data sets.

We are also using image analysis because the area of digital image processing/computer vision is well-established with good techniques for handling images and finding unique features. Since we compute the resulting image this allows us complete control over the output making image analysis an even better choice.

Space and Time Complexity

LiDAR data is complex and very large. So doing real-time interaction can be difficult. But, converting the data to an image and rendering it makes it easier to handle. Using images is also a great means of storing the resulting data after processing. In most cases the resulting image is dramatically smaller in memory footprint than the original LiDAR data used to produce it. We are essentially compressing the original data while adding even more information.

Using our data set we have multiple raw patches with four million points. The memory footprint of one of those patches is about 200MBs. Converting it to an image not only adds more information but dramatically decreases the size by almost a factor of one hundred to 2MBs through the use of lossless compression.

Since the image size is fixed that means it will scale very well.

We ran three benchmarks using different core combinations to evaluate the scalability and performance of the algorithm for use in interactive applications in figure 1. To tightly control the benchmark tests we used the same LiDAR patch, containing approximately one million points and had the application load it multiple times based on the number of patches being tested.

Automation

Automation is an important tool in analyzing large data sets because it helps a user focus on what matters. It delegates the complexities and challenges of the data to the computer, while letting a user focus on the decision making process. Semi-automated techniques while quite useful still require lots of existing data and training sets to get meaningful results. This can be time consuming not just in teaching a system, but collecting the data in the first place. Current work in deep learning used for recognition requires millions of tagged images in order to effectively train a model.

We achieve automation by intelligently selecting specific values so that we encode enough information at each pixel. The better the information we can encode into the resulting image the more accurate results we can achieve. We encode high quality sources of information, then we can use image analysis techniques to quickly compare different sections without requiring user training.

Algorithm and Methods

In this section we will detail the overall design and methods we used for our system. The first part covers how we encode unique details into an image, what those unique values are, and how we calculate those unique values. The second part covers the actual system implementation and the technologies we use to build the system.

Process and Encoding Values

In this section we detail the process of taking a LiDAR point cloud and transforming it into a custom image with specific RGB values in which we have encoded unique details. The first step of the process is taking the point cloud and generating a triangulate continuous mesh using Delaunay triangulation. We then compute specific values using these triangles. We used prior work done in visually encoding unique attributes from [17], Composite Density Maps for Multivariate Trajectories. This paper developed a flexible architecture for visually analyzing attributes that might reveal patterns. The authors were able to easily modify and calculate attributes and composite them into images in order to find unique patterns. In our work we propose three attributes that we believe were the most helpful in our analysis use cases, but the system is flexible enough that additional analysis techniques could be introduced or added to provide a new dimension of understanding or pattern analysis. Along a similar thread Kewei et al. [13] developed a statistical method for calculating a descriptor for stream lines and vector fields in order to do clustering and comparisons. For our work we propose a method by which we vectorize certain properties of LiDAR data in order to create an image while encoding more information. Another approach by Thompson et al. [19] developed a new representation called hixels that provided a compact and information rich format, but added scalar-value uncertainty. We use pixels as our storage medium to provide a more compact form and easy to use with image analysis techniques, but in doing so it does provide some uncertainty by utilizing a scalar values.

As part of our implementation we process and load each patch independently. To do this efficiently, we must manage the data in parallel fashion using OpenCL/OpenGL and other high performance technologies. We've placed all of our visual encoding operations onto the GPU using a GPU quad tree with a mixed approach using spatial hashing for our specific problem. As we discuss later we do this as part of the analysis step to save memory and computation. We begin by constructing the quad tree and spatial hash of all the triangles within the mesh. After we have completed this operation we pass these parameters into the GPU: triangle indices, vertex positions, vertex color, quad tree, hashed cells, and visualization parameters. Due to the synergy between GPU computation and rendering technologies we send all this information together because we can then instantly render the triangle data with the computed color data since it already exists on the GPU. This saves a significant amount of time due to the slow nature of copying data to and from system memory to GPU memory.

Quad Tree/Spatial Hashing Construction

We use a mixed approach combining spatial hashing with a quad tree. We start off as usual recursively subdividing triangles in 4 quadrants storing each quad in a heap array for direct ingestion by the GPU. Once we subdivide to a point where a quadrant contains 20 or less triangles we stop. Since GPU languages are C based we had to be very careful with how we represented data on the GPU. Memory is limited and there are limits to the size of buffers that can be allocated, so large amounts of data can't be crudely stored. [9]

At the 20 triangles or lower point a spatial hash cell is created that contains an index to each of those triangles from the global list of triangles. Then this cell is added to a global array and the index of its location is stored in that leaf quadrant. This produces an in order heap of just the triangles collected in the cells based on the leaves in the quad tree. As the recursion cycles back from the leaves to the root, a cell index range for each quadrant is built. This is possible because the leaves are in order. So at any given quadrant in the tree all the triangles can be accessed in roughly in roughly O(n) time because everything is indexed into an array. We use 20 triangles as the cut off point for a spatial hash cell because there is some memory overhead to maintain each cell. Since GPUs are limited in memory we chose this cut off point specifically for our setup. Lowering the cut off point would decrease the computation time since each triangulated patch is on the order of a few million triangles, but it would increase the memory overhead.

Local Normalized Elevation

We compute the normalized height of each vertex of each triangle based on it's surrounding triangles. This allows for very easy comparison of a specific region of the resulting image with any other area in the image. Computing the normalized height is the hardest part of all the values we encode. Since we are normalizing the elevation to values between [0.0,1.0] it is very sensitive and difficult to quantify if there is a gradual slope in the terrain. For example suppose there are two buildings with the same height

at two opposite ends on a LiDAR tile and there is a gradual slope in the terrain between them. Normalizing the elevation will result in one building being drastically higher then the other when both buildings are the same height.

Our work focuses on using just DSM LiDAR. We do not directly calculate the DTM (that is, the ground terrain surface without buildings or other features) from the LiDAR data to create the nDSM (normalized digital surface model, or the elevation of all objects on a flat surface) It is calculated by finding the difference between DTM and DSM. Doing so would require another computational step and require more memory. Instead we do normalization as part of the analysis process. We looked at several approaches. The first was the normalizing approach described in the previous paragraph, but this produced undesirable results. As mentioned buildings of the same height were being miscalculated due to variances in overall elevation. The second approach was a brute force comparison using a defined radius to find nearby triangles, but due the massive size of the data this operation was extremely slow and far from ideal. The third approach was sampling a large set of random triangles. This approach actually produced decent results and was quite fast, but it did not always work.

We then explored using a quad tree and using the minimum and maximum of the leaf nodes in the tree. This worked, it was fast, but it did not handle cases of very flat areas. For example our data covers a coastal region with a wide mix of elevation types from water to downtown cities. The normalized elevation for the water areas was heavily miscalculated. To fix this issue we include one more step. In order to account for variances in gradual terrain slope we use a statistical approach to measure the variance compared to the minimum and maximum elevation of the entire LiDAR patch. We do this by first selecting a specific triangle and iterating to the lowest quadrant containing it in our quad tree. If the variance in elevation does not meet a specific deviation from the elevation of the entire LiDAR patch then we step up in the quad tree to the parent and do the calculation again until we eventually get to a quadrant that contains enough variance. This iterative approach is similar to the method described in [8], but instead of iteratively using smaller and smaller windows we use the quad tree to find the best defined area for normalization.

The normalized height is the single most crucial value that we encode for doing any analysis. It also drives the rest of the values we encode. Since we select a region with enough statistical variance in elevation we use that region to calculate any comparison analysis that we encode. This value will eventually be stored as the red channel in our final rendered image.

Surface Normal and Slope

The surface normal is the vector of the plane created by the triangle. We have to encode this three dimensional vector to a one dimensional value. We do this by taking a uniform directional vector and compute the dot product of it and the surface normal. This creates a one dimensional value between 0.0 and 1.0 that gives us the slope of the surface. This calculation is the same as computing the light intensity on a given surface using a directional light. The idea is straight forward any surface with a steep slope with relation to the directional light will have lower intensity. Computing the slope of the surface allows us to very easily distinguish steep surfaces like buildings and trees. This value will eventually be stored as the green channel in our final rendered image.

Navigation Mesh and Drive-ability

During our initial work for determining values, we found it difficult to detect a road network using all the previous methods. Our initial analysis was for LiDAR patches in coastal regions. We found that due to the nature of the elevation of the ground, very minimal building placement, and vegetation it can be difficult to reliably detect roads from some types of LiDAR.

One option is to use existing road vector maps; while this would produce the best results it is not always feasible. The points in time when the LiDAR was collected and when the road vector maps were created can be different. Analyzing the differences between different periods in time, especially in rural and underdeveloped areas would require collecting older road vector maps and might not be possible in all cases. We wanted a consistent solution based on the LiDAR itself.

The solution we explored was generating a navigation mesh based on the geometry of the resulting triangulated LiDAR data [11]. We explored navigation mesh creation using an image-based approach with LiDAR in [1]. But, the work they presented was done in city landscapes and there is no mention of rural environments or any such results. In addition, most navigation mesh research aims to produce a simplified graph network for performing path planning. Our goal is not to produce an optimized graph, but to create an estimation metric for determining a road network. Therefore, we are not concerned about the complexity of the graph since we have no plans to perform path planning.

We use our GPU computing workflow to complete this process. The first step operates on all the triangles independently in the 3D LiDAR data. It finds the best node size in the quad tree to perform the local normalized height. It calculates the local normalized height for that triangle, the dot product of the surface normal with a directional vector, the area of the triangle. Lastly it flags the triangle with a single bit value if it fits the criteria for being drive-able. As part of the navigation mesh creation process we discard triangles with surface normals too steep for a road. We also discard triangles that are too small for a car to pass on.

We use the quad tree nodes to estimate a navigation mesh. Areas of high elevation change will most likely not have suitable geometry for connecting together into a navigation mesh, therefore we can minimize our search within those nodes and limit the amount of computation that has to be done. In order to compute an estimated road network we use a modified kernel density estimation using the quad tree as the range in which to do the local search and each triangle as a cell. In the local search we determine how many driveable triangles are connected with a particular triangle to form a continuous path. This process runs on each triangle in the mesh in parallel on the GPU. Since most of the calculations performed are vector operations the GPU is suited well for this task.

This results in an estimated network that a car can physically drive on. While not completely accurate, it estimates a reliable road network that can be easily encoded into an image that is based on the LiDAR data itself.

Image Analysis + Search

After processing a LiDAR patch and calculating the encoded values, we store each attribute as a color for each vertex of each

triangle in the triangulated mesh. We then use this color data and render the triangulated mesh to an image. The encoded values represent each color channel in the image. The red channel contains the calculated normalized height of the 3D geometry. The green channel contains the dot product of the normal for the 3D surface and a static directional vector. Lastly the blue channel contains an approximated road network. As part of the rendering process we use a fixed orthographic projection and scale across all the LiDAR patches to maintain a constant pixel to meter ratio. For the purpose of our work we constrain the image size to 2048x2048 pixels using 72 DPI and a bit depth of 24 RGB. In this case we chose a bit depth of 24 while this gives us up to 16 million different color combinations in some cases this might not be enough detail. This flexibility allows us to control many aspects of the image and the memory footprint. If we are trying to process and analyze large numbers of LiDAR patches to where storing all the resulting images would require too much space, we can adaptively reduce the size of the image through either the bit depth or image size. If we want to compare just a few patches and there is ample space, we can increase the image size and bit depth to give more accuracy.

Once a set of LiDAR tiles has been converted to images they can be used to search using image analysis. We use techniques that have already proven very effective and they are extremely fast. The first step of our process is using image segmentation through thresholding. This initial phase can be done extremely quickly on a GPU and it instantly isolates regions that share similar values with a region a user is looking for. The entire goal of how we encoded attributes into images was for this step of thresholding. The process of encoding values into the RGB spectrum had to be meticulous because thresholding is very sensitive. The next step after thresholding is boundary analysis and blob extraction we also perform this on the GPU in order to achieve the up-most speed in searching. We perform blob extraction through marching squares [14]. The marching squares algorithm is embarrassingly parallel and fits well into our GPU workflow. Lastly we calculate an image histogram for each blob to further analyze later. Each histogram for each blob can also be calculated on the GPU depending on the size of the blob. We combine all of these steps into a continuous process whereby we keep as much information as possible on the GPU to minimize data transfer from host to device and vice versa. This allows us to very quickly find results from searches on very large datasets in realtime. Using a similar hardware setup for the encoding process and 1024 x 1024 image sizes we are able to search approximately 52 processed LiDAR patches per second.

Application

In this section we describe the visual analytics tool that we have developed using our techniques. In order to do searching and finding we use thresholding and image segmentation. A user first makes a selection for some type of feature (for example a building). We determine the most frequent color in that user's selection. Using this RGB value we then threshold all existing processed LiDAR images. Since we have encoded meaningful information into the processed images, using thresholding is a highly effective technique for finding areas of similar value based on the user's selection.

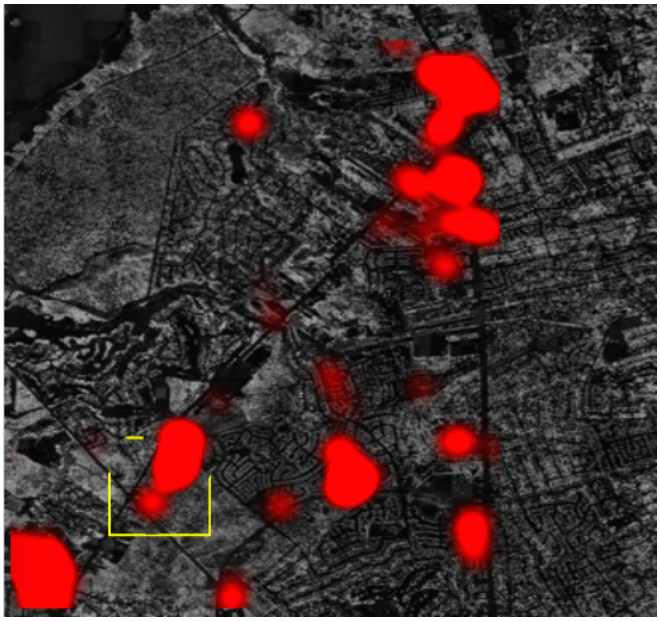

Figure 2. *One of the coordinated views. This view is the global view that shows a heatmap weighted by the similarity of the results. It also shows the bounding box for the the zoomed in detail view.*

Figure 3. *One of the coordinated views. This view is the zoomed in detail view that allows users to make selections. The blue rectangle is a user's selection over a building structure to search for similar features. Higher intensities of red indicate areas of higher elevations. Higher intensities of green indicate steep slopes. Higher intensities of blue indicate areas less likely to be roads.*

Interface

We present to the user two coordinated views. We followed the guide lines for Multiple View Coordination in [16]. One view is a global fixed view that provides a complete overview of all the LiDAR tiles in figure 2. The second is a zoom in detail view that allows a user to navigate around and zoom into specific areas in figure 3. Together the views are connected. In the global view a bounding box represents the view bounds and position of the zoomed in view. Below the coordinated views is a scatter plot that displays all the similar feature results calculated from a selection in figure 6.

Interaction and Analytics

We provide the user with two selection tools. The first is a bounding box selection and the second is a lasso tool that is similar to what is in most image editing software applications. Once a user makes a selection we isolate that part of the image and run modified k-means clustering on the selection to extract the four most dominate colors as well as the radius of the clusters that we use to control the range of thresholding. Due to its nature k-means does not produce the same results each time by choosing random seeds. This is significant because thresholding is very sensitive to small changes; different clusters from the k-means could result in quite different results. We explored solutions presented in [20].

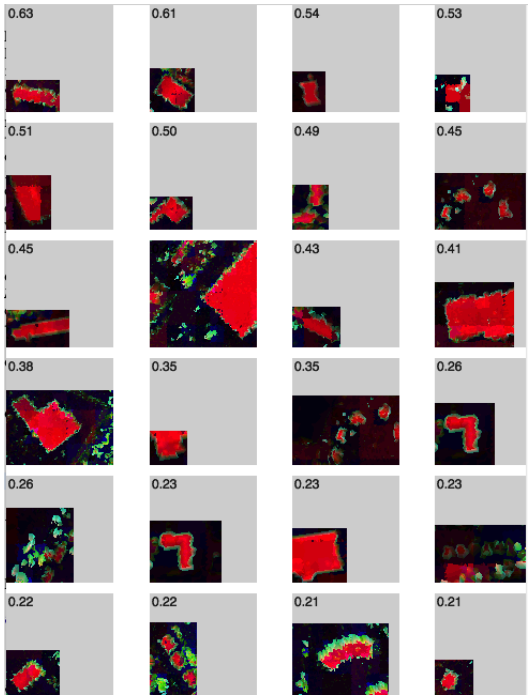

Figure 4. *The detailed list view, which shows the sorted results of a search. Each cell contains a ranking and an image of the feature that was found.*

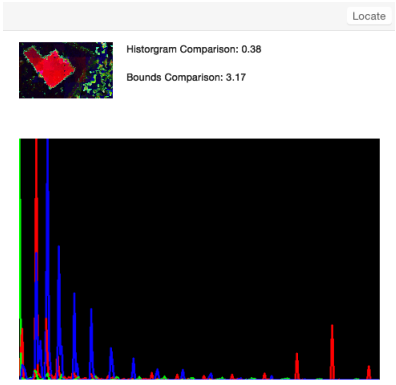

Figure 5. *The detailed feature view, which shows the histogram comparison and the bounds comparison. Below it shows a detailed break down of the histogram for that feature. At the top is a locate button that allows a user to locate the feature directly in the LiDAR patches.*

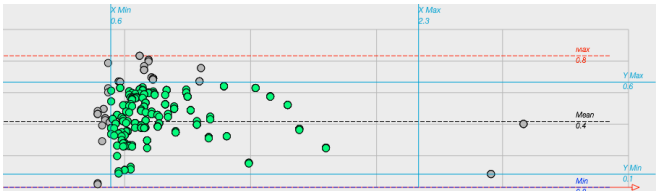

Figure 6. *An example scatter plot showing a selected subset of data from a search. The subset of data selected is represented by the highlight green points. The points in grey are unselected and filtered out.*

$$d(H_1, H_2) = \frac{\sum_I (H_1(I) - \bar{H_1})(H_2(I) - \bar{H_2})}{\sqrt{\sum_I (H_1(I) - \bar{H_1})^2 \sum_I (H_2(I) - \bar{H_2})^2}}$$

Figure 7. *Histogram comparison function.*

Ultimately to fix this problem we ended up seeding each cluster using the same values each time based on the RGB channels. This produces the same results each time for the same selection.

Using each color we threshold the image using the GPU to speed up the process. Then we run boundary analysis and blob extraction. As previously mentioned we perform these operations on the GPU as well using marching squares. Each blob is then further analyzed and compared to the original selection through a histogram comparison and a bounds comparison. We plot each feature that is found in a scatter plot.

Along the x axis we plot how similar in width and height a specific feature is. A value of 1.0 would mean the feature is approximately the same size. A value of 2.0 would mean the feature is approximately twice as large as the user's selection and a value of 0.5 would mean the feature is about half the size of original selection. Along the y axis we plot the histogram comparison. We compute this value by creating a color histogram for the original selection and all of the results found. We then use the function in Figure 7 derived from OpenCV [3] to compute the correlation between two histograms. This metric ranges from 0.0 to 1.0 based on correlation. where 1.0 means the feature and selection have strongly similar histograms.

Within the scatter plot we display the max, mean, and minimum values from the resulting analysis. We also allow users to filter results further by sliding adjustable clipping lines that narrow results.

Any selection made in the scatter plot is then reflected in the detailed list view in Figure 4 and the global view in Figure 2. We used prior techniques to visualize spatial uncertainty that we have found effective in [7]. We incorporated glyphs to represent how similar a feature is to a user's selection. The glyph is represented as a circular ring that is added on top of the global view. The more complete the ring is the higher the similarity exists between that feature and the selection. A user can then further click on this glyph in the global view to reveal a detailed view of the feature in Figure 5.

We also incorporated a heatmap to quickly visualize hotspots as seen in Figure 2. We use a kernel density estimation that is weighted using the histogram similarity. In this way users can quickly focus on regions with the highest similarity of results. Lastly, we provide a concentrated list of the narrowed results from the scatter plot ranked by similarity in Figure 4. From this list users can select a specific feature that was found and visually examine it in Figure 5. They can locate it directly from the global view to find its origin. Also a user can create a new search using that feature to further refine what they are looking for.

Case Study and Results

In this section we discuss the case study we ran and details of the three different tasks performed by participants. As part of our case study we followed previous methods described in [18].

We selected 10 participants for an informal evaluation of the tool having them complete 3 basic tasks that focus on the visual analytics portion of our application. The participants in our evaluation were equally balanced between male and female. Their ages ranged from college age to senior citizen age. The professional background of our participants also covered a wide range from engineering to medical professional.

We set up three tasks ordered as to how a prospective user would use the application. In the first task we focused on discovery, exploration and selection. In the second task we took the prior selection from task 1 and built upon it by asking the participant to analyze the results of their selection. This included evaluating the spatial analysis results and detailed results. In the last task we asked participants to use the interactive tools to further drive the analysis by narrowing their original selection and analyzing the results from those interactions.

At the end of the session we followed up with participants with a questionnaire of 12 questions, 3 for each task and 3 detailing the overall usage of the tool. Each task had 3 questions with numerical scales to rank a particular sub-task. The final 3 questions ranked the overall learning experience and ease of use. The questionnaire also included a descriptive question asking which feature or function was most helpful to accomplish a task.

Task 1

In the first task we covered navigation and zooming as well as the use of the box and lasso tool. Initially we asked participants to navigate and focus on a particular building. We located for them on the global view a particular area of interest and asked them to navigate and focus on that region. We wanted to measure how easy it was for a participant to pick up the application and navigate to an area of interest. Secondly, we had them use both the box and lasso tool to make a selection around the area of interest. These two operations are the core of initiating the analysis of our tool so being able to very quickly navigate to areas of interest and make selections is vital to the analysis. All of the participants highly ranked the ease of use for navigating and focusing on a particular region.

Task one result averages, ranked from 1 to 7. 1 being very difficult and 7 being very easy.

How easy was it to use navigating and zooming	6.5
How easy was it to use the box selection tool.	6.625
How easy was it to use the lasso selection tool.	6.125

Task 2

In the second task we had participants analyze the results of their prior selections from task 1. We first started by asking them to read from the scatter plot of results and detail for us the overall metrics of the results. This included identifying the complete range from the smallest result returned to the largest, as well as the least similar result to the most similar. We also asked them to tell us what the mean value was for each metric. The goal of this task was to get participants to analyze the results of their selection by evaluating the results returned in the scatter plot and doing visual analysis. As part of the visual analysis we had participants use the heatmap to locate regions and then had them perform basic interaction with the annotations to find results with spatial context.

We asked participants to use the heatmap overlay to tell us where there are areas of similar features. We then had the participants navigate to those areas. Next, we asked participants to switch from the heatmap overlay to the annotation overlay. We then asked participants to select one of the annotations in the global view in the region they had just navigated to and examine the result.

As part of our evaluation we had participants rank the ease of use in using the scatter plot, understanding the scatter plot results, using the heatmap and interacting with the annotations. All of these subtasks were ranked highly for ease of use and understanding. The heatmap function was also listed as the most helpful function for accomplishing tasks.

Task two result averages, ranked from 1 to 7. 1 being the least helpful/intuitive and 7 being the most helpful/intuitive.

How intuitive was the scatter plot and understanding the results.	5.875
How helpful was the annotation overlay in examining results.	5.625
How helpful was the heatmap in locating results.	5.75

Task three result averages, ranked from 1 to 7.

How easy was it to locate buildings of similar size	6.0
How helpful is the detailed list of ranked results.	5.625
How easy was it to locate a result on the map.	6.25

Task 3

In task 3 we asked participants to use their prior selection from task 1 to narrow down the search results to find buildings of similar size with the most similarity to their selection. We then asked participants to locate the top three similar features for us on the map using the detailed results table. This final task built upon the two previous tasks by first requiring navigating and selecting a region of interest. Then, from task 2, the participant analyzed the overall results of their selection in both the scatter plot and spatially.

We started by telling participants that we wanted to find features within a specific size range with the most similarity. This required the participants to interact with the scatter plot results by filtering out all the results below and above certain values, in this case to find features of a certain size and with the most similarity.

Once an interaction occurred in the scatter plot, it is updated to the detailed results table that sorts all the results based on their similarity metric. Participants then used this table to select the top three features that were most similar. Using the detailed view they were able to then locate and point out where these features were on the LiDAR patches.

Future work and Conclusion

We will start first by discussing the shortcomings. During our research we very quickly discovered that there is no single method for solving our problem; it is multi-faceted. One of the key areas of our method is encoding values. Since our encoded values are based on a triangulated mesh any artifacts or errors produced by the triangulation will percolate into the analysis. If the values are too ambiguous and do not provide enough detail, then trying to do further analysis or extracting other features is extremely difficult. We discovered this in trying to extract a road network. The encoded values we were using were too ambiguous; we had to come up with another metric and therefore we chose to use a navigation mesh. We also recognize that our solution is not a complete solution for all possible use cases for analyzing LiDAR data, but we believe that our approach is flexible enough to incorporate new analysis techniques that can be embedded into the final encoded image.

Overall for all three tasks, we received strong positive rankings on the follow-up questions we asked. At the end of the questionnaire we asked three general questions about the overall system. The first question asked participants to rank the ease of use for the application. The second question asked participants how similar they thought the results were to what they had originally selected. Last, we provided an open-ended question asking participants what they thought was the most helpful feature. We received high ranks from our participants for the overall ease of use in using the interface, carrying out the tasks, and understanding the analysis results. The participants also thought they results they found were very similar to what they had originally selected.

Final question result averages, ranked from 1 to 7. 1 being the worst and 7 being the best.

How would you rate the overall ease of use of the interface.	6.0
How similar do you think the results are from a selection.	5.625
What was the most helpful function.	Heatmap

As we mentioned in our prior work section, previous work done in this area found thresholding to be extremely accurate and efficient for these types of tasks. But it was prone to user input error, it required fine tuning the thresholding values, and only operated on two thresholding values. We have improved upon these issues by first removing user input error. The range that is selected for thresholding comes directly from the user. We allow them to extract this information from the processed LiDAR tiles themselves using a modified k-means algorithm that we discuss later in the application section. By using the processed data as input to search any existing processed tiles we can control for any user input errors in searching. This also solves the second issue of fine tuning the thresholding values. Using a k-means algorithm we are able to select the best range to use for thresholding. We have added another dimension of detail to do thresholding, but more importantly we have meticulously performed pre-processing and analysis on the LiDAR tiles to generate high quality information that has been encoded into the image. Lastly our process is flexible in that many variables can be changed to account for specific use cases. The image size, bit depth, and the encoding processes of mapping information to the RGB channels in the processed images can all be modified to fit certain criteria.

We explored the possibilities of using machine learning for classification and matching, but ultimately that required an extensive training set of data. The goal of our effort was to find a scalable, unsupervised approach that leveraged human analysis in an interactive tool for large data sets. Our solution is not bullet proof and is not immune to false positive results being generated, but that is where we bring the user in for understanding.

As future work we see this being a great addition to the tool. Allowing the user to remove bad matches and refine a data set based on his or her selection. We see potential in machine learning to utilize these selections made using the application that would allow for easy tagging and filtering. As the system is used it can learn more about what a particular user is looking for and continually build up a repository of knowledge that can be applied to later analysis or future predictions. In this way the tool could be a means by which a data set is created that could be used for deep learning.

Due to the nature of LiDAR it can grow exponentially when examining larger spatial regions and different points in time. The size of LiDAR datasets keeps growing. Sensors are collecting more and more data. The files can be hard to manage; even loading and viewing them can be a problem. It can be time consuming collecting different sources of data, processing them with a myriad of different tools, and converting LiDAR coordinate systems.

In this paper, we present and discuss a concept and the implementation of a system that is capable of analyzing and visualizing LiDAR data in real-time. We also tackled three major issues: scalability, interaction, and the ability to search effectively. LiDAR data by nature can be massive, hard to manage, and visualize. We utilized GPU computing and image analysis to achieve our goal of real-time interaction and analysis. As part of our GPU processing step we independently analyze each patch of LiDAR data to encode values into a 2D image.

We implemented our concept into an interactive visualization tool that allows users to easily and efficiently explore, find areas of interest, and analyze those areas. We incorporated several interactive tools for managing and organizing the data. Using the tool we ran an informal case study evaluation our system with 10 participants. Each participant performed a series of tasks that examined how easy the system was to use in navigating, exploring, and selecting areas of interest. Next we had each participant analyze the results of their selection through the use of a scatter plot and spatial analysis. In the last task we had participants isolate the data looking for specific features. Overall the participants for each tasked ranked the system very well. They found it responsive, easy to use and understand. They were also confident in the ability of the system to return similar results. Through the results and efforts of our work we believe we have accomplished what we set out to originally do.

This work was supported in part by the U.S. Department of Homeland Security VACCINE Center under award no. 2009-ST-061-CI0002.

References

[1] A. Akaydın and U. Güdükbay. Adaptive Grids: An Image-based Approach to Generate Navigation Meshes. *Optical*

Engineering, 52(2):Article No. 027002, 12 pages, February 2013.

[2] T. Blaschke. Object based image analysis for remote sensing. {*ISPRS*} *Journal of Photogrammetry and Remote Sensing*, 65(1):2 – 16, 2010.

[3] G. Bradski. *Dr. Dobb's Journal of Software Tools*, 2000.

[4] T. Butkiewicz, R. Chang, Z. Wartell, and W. Ribarsky. Visual analysis and semantic exploration of urban lidar change detection. In *Proceedings of the 10th Joint Eurographics / IEEE - VGTC Conference on Visualization*, EuroVis'08, pages 903–910, Aire-la-Ville, Switzerland, Switzerland, 2008. Eurographics Association.

[5] R. Chang, T. Butkiewicz, C. Ziemkiewicz, Z. Wartell, N. S. Pollard, and W. Ribarsky. Legible simplification of textured urban models. *IEEE Computer Graphics and Applications*, 28(3):27–36, 2008.

[6] C. Clasen, F. A. Kruse, and A. Kim. Analysis of lidar data for emergency management and disaster response. In *Imaging and Applied Optics Technical Papers*, page RTu2E.2. Optical Society of America, 2012.

[7] T. Eaglin, X. Wang, W. Ribarsky, and W. Tolone. Ensemble visual analysis architecture with high mobility for large-scale critical infrastructure simulations, 2015.

[8] J. Estornell, L. A. Ruiz, B. Velzquez-Mart, and T. Hermosilla. Analysis of the factors affecting lidar dtm accuracy in a steep shrub area. *Int. J. Digital Earth*, 4(6):521–538, 2011.

[9] E. J. Hastings, J. Mesit, and R. K. Guha. Optimization of large-scale, real-time simulations by spatial hashing.

[10] T. Hermosilla, L. A. Ruiz, J. A. Recio, and J. Estornell. Evaluation of automatic building detection approaches combining high resolution images and lidar data. *Remote Sensing*, 3(6):1188–1210, 2011.

[11] M. Kallmann and M. Kapadia. Navigation meshes and real-time dynamic planning for virtual worlds. In *ACM SIGGRAPH 2014 Courses*, SIGGRAPH '14, pages 3:1–3:81, New York, NY, USA, 2014. ACM.

[12] M.-P. Kwan and D. M. Ransberger. Lidar assisted emergency response: Detection of transport network obstructions caused by major disasters. *Computers, Environment and Urban Systems*, 34(3):179 – 188, 2010.

[13] K. Lu, A. Chaudhuri, T.-Y. Lee, H.-W. Shen, and P. C. Wong. Exploring vector fields with distribution-based streamline analysis. In *Visualization Symposium (PacificVis), 2013 IEEE Pacific*, pages 257–264, Feb 2013.

[14] H. Mantz, K. Jacobs, and K. Mecke. Utilizing minkowski functionals for image analysis: a marching square algorithm. *Journal of Statistical Mechanics: Theory and Experiment*, 2008(12):P12015, 2008.

[15] R. Richter and J. Dllner. Concepts and techniques for integration, analysis and visualization of massive 3d point clouds. *Computers, Environment and Urban Systems*, 45(0):114 – 124, 2014.

[16] J. C. Roberts. State of the art: Coordinated and multiple views in exploratory visualization. In *Fifth International Conference on Coordinated and Multiple Views in Exploratory Visualization*, pages 61–71, 2007.

[17] R. Scheepens, N. Willems, H. van de Wetering, G. Andrienko, N. Andrienko, and J. van Wijk. Composite density maps for multivariate trajectories. *Visualization and Computer Graphics, IEEE Transactions on*, 17(12):2518–2527, Dec 2011.

[18] M. Sedlmair, M. Meyer, and T. Munzner. Design Study Methodology: Reflections from the Trenches and the Stacks. *IEEE Trans. Visualization and Computer Graphics (Proc. InfoVis)*, 18(12):2431–2440, 2012.

[19] D. Thompson, J. Levine, J. Bennett, P.-T. Bremer, A. Gyulassy, V. Pascucci, and P. Pebay. Analysis of large-scale scalar data using hixels. In *Large Data Analysis and Visualization (LDAV), 2011 IEEE Symposium on*, pages 23–30, Oct 2011.

[20] K. Wagstaff, C. Cardie, S. Rogers, and S. Schroedl. Constrained k-means clustering with background knowledge. In *In ICML*, pages 577–584. Morgan Kaufmann, 2001.

[21] A. Zerger and D. I. Smith. Impediments to using {GIS} for real-time disaster decision support. *Computers, Environment and Urban Systems*, 27(2):123 – 141, 2003.