Mesh Difference Visualization and Attribute Validation

David Doria Cody Phillips

Abstract

We propose a method for visualizing the differences between surface normals produced by two surface normal estimation algorithms. The two algorithms, Multi-Scale Tensor Voting (MSTV) and Prioritized IRLS Normal Estimation (PINE), operate over range images acquired by a ground-based single viewpoint LIDAR scanner. Both algorithms generate additional surface attributes as an intermediate step to computing surface normals. This intermediate information is useful in exploring any differences in the two technique's results. We visualize such intermediate information such that a direct comparison to areas of discord between the two estimation methods can be made. In addition to examining the relative error between the MSTV and PINE, we simulate the scanning process on computer model with known normals so that the performance of each method can be compared to a ground truth.

Keywords: Tensor Voting, Surface Reconstruction

1 Introduction

Modeling 3D objects and scenes from real world observations is an important task for many fields. This is especially true in computer vision, robotics and graphics, where many problems require methods for acquiring and interpreting raw three dimensional data. Most modeling techniques have been developed for use with small static indoor scenes or individual objects. One such example is Michelangelo's statue of David, which was digitized with a custom built Cyberware laser scanner as part of Standford University's Digital Michelangelo project [Levoy and Fulk 2000]. Advances within the last decade in active laser scanning equipment have enabled full 3D scans of outdoor environments to be acquired using a ground based single viewpoint range scanner. However, outdoor environments pose significant challenges that limit the success of directly applying previously existing techniques. Most of these challenges stem from the fact that is not as easy to exercise control over a large outdoor scene as it is with a small indoor object or scene . Several techniques have been developed to address these problems, but little research has been performed on comparing the results of such new methods. It is specifically of interest to determine structural characteristics that evade proper estimation, and the degree to which each technique is able to cope with these problematic surface features. To aid in this endeavor, we propose a method for visualizing the differences between surface normals produced by two surface normal estimation algorithms, Multi-Scale Tensor Voting (MSTV) and Prioritized IRLS Normal Estimation (PINE). Both algorithms operate over range images acquired by a ground-based single viewpoint LIDAR scanner and generate additional surface attributes as an intermediate step to computing surface normals. Visualizing such intermediate information is useful in finding intuitive explanations of areas of discord between the two estimation methods. In addition to examining the relative error between the MSTV and PINE, it is be fruitful to compare each algorithm's normals against a ground truth. The most accurate means of performing such a comparison is to simulate the scanning process on computer model with known normals, and run both methods on the simulated data. The main goals of this paper are to

- Visualize differences between calculated surface normals
- Visualize surface attributes to help explain such differences
- Generate ground truth normals using a simulated scanning process

2 Computing Normals

An example range image is shown in 1. The same range image with annoted normals is shown in 2.

Figure 1: *Stanford Bunny Model*

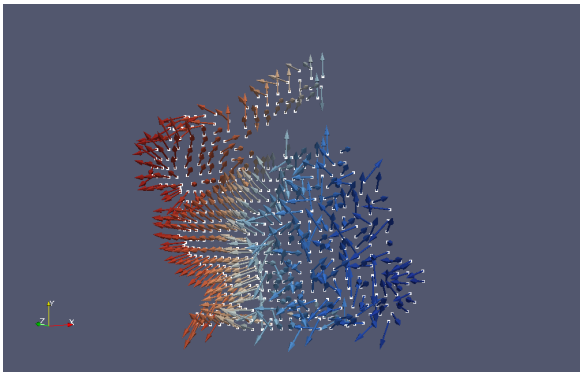

Figure 2: *Stanford Bunny with Normals*

2.1 Tensor Voting

The Tensor Voting Framework, pioneered by Medioni et al. [G. Medioni and Tang. 2000] [Guy and Medioni 1996], has been applied to numerous computer vision problems [Tang 2003] [E.-Y. Kang and Medioni 2002] [L. Reyes and Bayro 2007]. In its most general formulation, it takes as input points belonging to some N-dimensional space and encodes them as N-dimensional tensors. The point and its tensor are collectively called a token. Tensor voting propagates the information of each tensor into its local neighborhood by way of tensor voting, ultimately creating a dense tensor field from the originally sparse input. Each tensor can be decomposed into features that have some geometric meaning, and each feature has a corresponding saliency component.

In the 3-dimensional case, a tensor can be decomposed into three geometrically meaningful terms, the stick, plate and ball features.

A token's stick feature represents the surface normal at it's location while the plate feature represents a curve tangent vector. The saliency of the ball feature represents the degree to which the point appears to have no orientation, describing neither surface nor curve. If the saliency of a token's stick feature is sufficiently high, the point is likely to lie on a surface with a normal described by the stick direction. Surfaces and curves can be extracted from the tensor saliency field using non-maximal suppression.

Mordohai introduced a simplified voting scheme that allowed tensor voting to scale well to higher dimensions [Mordohai and Medioni 2006]. Tong introduced a multi-scale formulation of tensor voting that could automatically extract features across different scales [Tong. 2004]. A multi-scale approach to feature extraction is especially important when processing range data, as it inherently possesses features with large scale variations. King expanding on both the works of Mordohai and Tong creating a multi-scale tensor voting (MSTV) approach designed for the extraction of features from range data. He reduced the complexity of tensor voting by introducing some algebraic simplifications as well as devising smoother weighting profiles and a better curvature penalty [King 2008].

2.2 Prioritized IRLS Normal Estimation

The prioritized IRLS surface normal estimation (PINE) algorithm [Phillips 2009a] calculates surface normal and scale estimates by a robust fitting of local tangent planes to each point. The robust estimation of the tangent plane τ is achieved by Iteratively Reweighted Least Squares (IRLS) with an M-estimator using the Beaton-Tukey biweight influence function ω.

For each estimation to succeed a sufficient number of inliers must exist with respect to the estimated parameters $\tilde{\theta}$, with intercept $\tilde{\phi}$ and normal $\tilde{\eta}$.

The key aspect of PINE is the prioritization of the order in which each point is estimated. Intuitively, relatively flat regions are likely to produce stable estimations and are visited first. These estimates are then used to improve the estimation of adjacent regions that may contain discontinuities, noise or curved surfaces. The algorithm performs three passes over the data in three separate phases. In the first phase, points are assigned initial surface normal estimates if possible. They are assigned a priority value

$$1.0/\nabla\tau_{\tilde{\theta}} \times I \times \omega(\phi_0 - \tilde{\phi}, \rho) \quad (1)$$

where ϕ_0 is the measured distance and I is the fraction of inliers, and are placed in a priority queue. The priority metric is designed to capture the preference for stable flat regions when performing early estimates. The inverse gradient magnitude is a "flatness" term while the inlier fraction and m-estimator weighting are essentially stability terms.

During the prioritized estimation phase, points are estimated in the order they exist in the priority queue. If IRLS succeeds is this phase the point is classified as a either being an outlier or belonging to a cloud, curve or surface. If the point is classified as a surface, and therefore has a normal, its neighbors are re-evaluated as their priorities may be improved by this new estimate. If their priorities do improve, their estimated parameters are updated and their positions the queue are improved. After all the points have been visited in the prioritized estimation phase, a final pass over all the points is performed to calculate the final classifications and surface normals. For the final estimation the parameters are relaxed, and the neighborhood of a point is slightly increased to give it larger influence. Every point re-estimated is averaged with its neighbors weighted by ω parametrized by how much their normals differed in proportion to the estimated scale. If any point has failed all estimation attempts, it is labeled an outlier. The algorithm continues as in the prioritized phase, performing IRLS, classifying the points and updating the estimates of their neighbors.

3 Visualization

Eight different visualizations are implemented [Phillips 2009b], six of which can be represented either the form of a 2D image or as the color of a 3D mesh. The two that were not visualized on the 3D mesh are the RGB intensity image and scale estimates, as it was deemed unnecessary.

3.1 RGB Intensity Image

Each point in the range image that has a laser return has associated color data sampled from a color camera affixed to the scanner. An easy way to visualize the structure of the 3D data is to extract this color information and save it as a color image of the same dimensions of the range image. An example of an RGB Intensity image is shown in Figure 3.

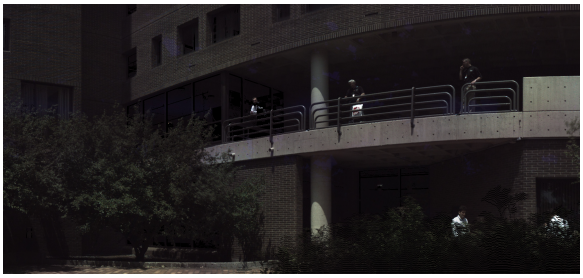

Figure 3: *Sample RGB Intensity Image*

3.2 Normal Comparison

It is useful to have a visualization of both algorithm's surface normals alone, as well as the difference between them.

We implemented the following visualizations:

- Normal Axis Colored (Figure 4)

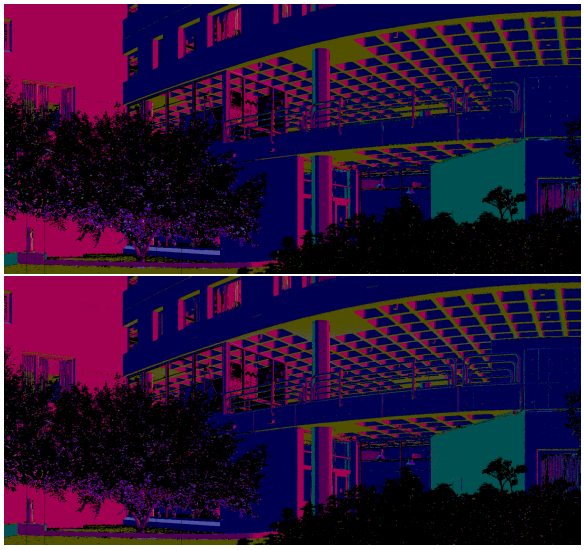

Figure 4: *Sample Axis Colored Images*

- Normal/LOS Angle Colored (Figure 5)

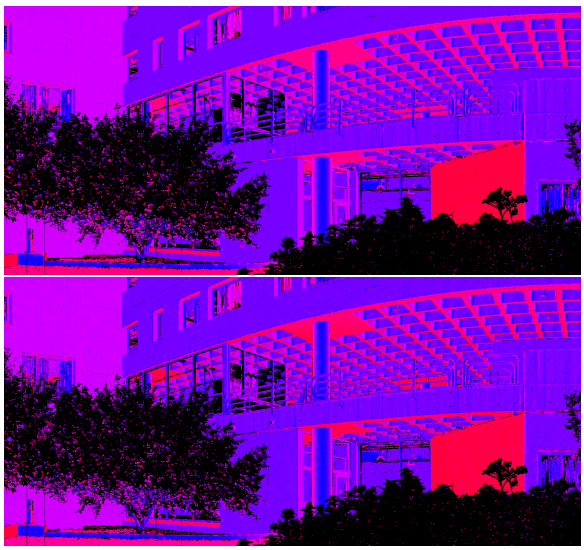

Figure 5: *Sample Dot Product Images*

- PINE/MSTV Thresholded Angle Colored (Figure 6)

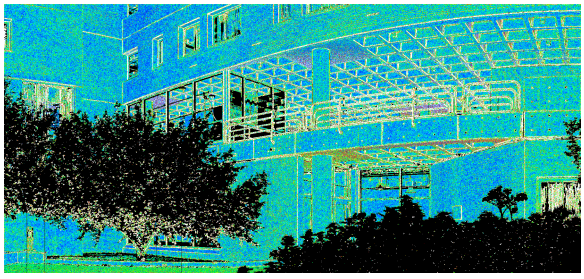

Figure 6: *Sample Angle Differences Image*

Two coloring methods were used to represent surface normals. The first method is coloring by axis components, where (x,y,z) of the normals correspond to (r,g,b). The second method is representing the angle between the normal and the line of sight for each point. The angle acts as a parameter to the hue component of the HSV color scale (with fixed S and V). To visualize the discrepancies between normals, it was decided to threshold the colors to specific ranges, as differences greater than a certain angle usually suggest the normal is conceptually undefined at the point (such as along thin structures like railings). For the angle difference, we used graded colors shown in Table 1.

Range (degrees)	Color
0 - 6	Blue
6 - 12	Green
12 - 18	Yellow
18 - 24	Red
> 24	White

Table 1: *Table of Colors*

3.3 Point Classifications

PINE produces 5 labelings, 4 of which are different types of non-surface points.

These labelings are summarized in Table 2.

Type	Color	Description
Undersampled	Yellow	Surfaces scanned at oblique angles
Outliers	Red	Points that don't appear to lie on the true surface
Cloud	Cyan	Unresolvably fine structures such as foliage/brush
Curve	Magenta	Very thin structures such as railings or power lines
Surface	Blue	Points with statistically stable normals

Table 2: *Classification Colors*

An example of a point classifications image is shown in Figure 7.

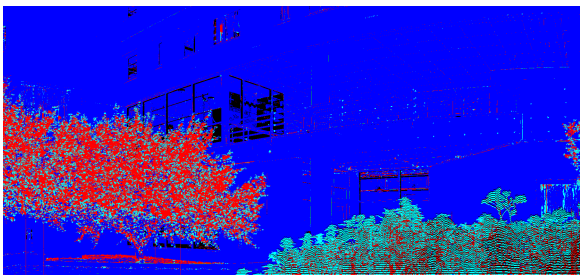

Figure 7: *Sample Point Classifications Image*

3.4 Scale Estimation

PINE produces scale estimates, which are a measure of how rough the surface is. Depth or angle discontinuities increase this value. An example of a scale image is shown in Figure 8.

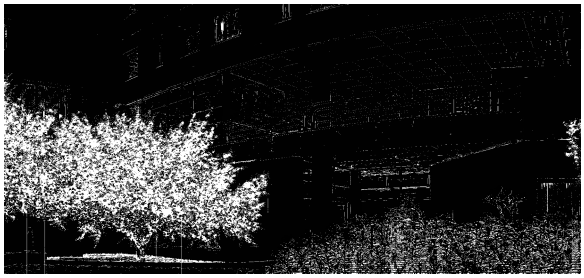

Figure 8: *Sample Scale Image*

3.5 Scale Normal Count

MSTV doesn't analyze all points across all scales, as this is computationally prohibitive. It instead selects points for each scale that are deemed to suitably describe the underlying structure. Therefore, some points are evaluated at multiple scales and are assigned multiple normals. This is a visualization of the number of normals assigned to each point. An example of a scale image is shown in Figure 9.

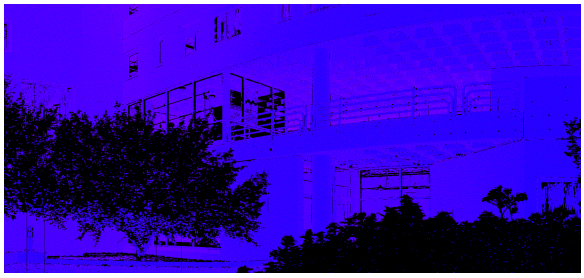

Figure 9: *Scale Normal Count Image*

3.6 Estimation Order

The prioritized nature of PINE is captured by an estimation order visualization. These visualizations show the order in which points were given their final estimations. A blue to white color scale was used, with blue points finishing first. An example image with points colored by order is shown in Figure 10

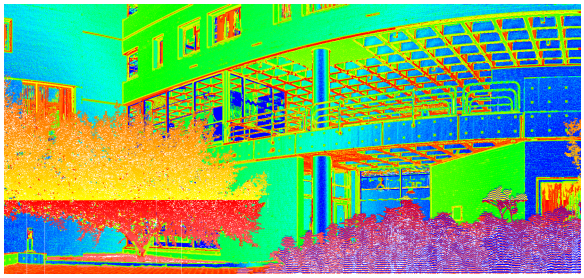

Figure 10: *Point Order Image*

3.7 Side-By-Side Comparison

To demonstrate the effectiveness of the visualizations, we provide a side-by-side comparison in Figure 11.

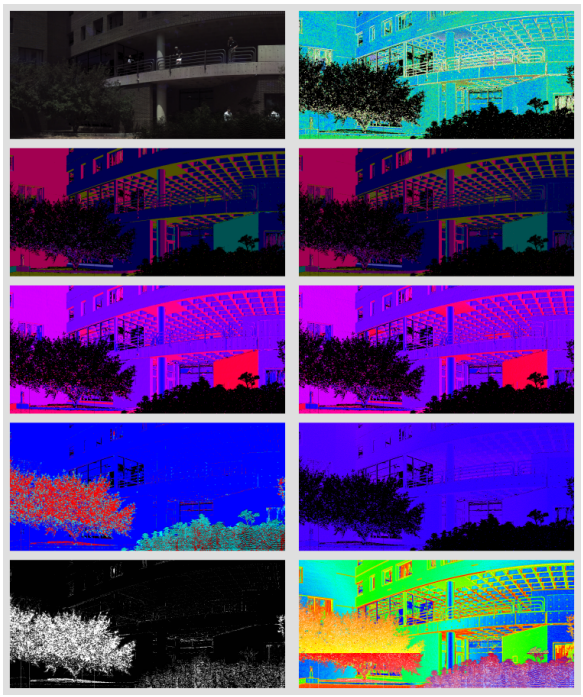

Figure 11: *Side-By-Side Comparison*

4 Iterative Development

Many challenges were encountered during the development of the visualization code, and the overall design and functionality of the code experienced many extreme alterations.

4.1 Non-STL Constructs

The base files for this project were imported from homework one. They employed many non-STL objects that used programming models that were slightly awkward in relation to the current Standard Template Library. For example, iteration through a collection structure required iterators to be explicitly created and destroyed by the collection class. The hash structures was replaced [Phillips 2009b] with STL sets or STL vectors, depending on their function. STL sets do not perform as well on lookup, however it was easy to create the required less-than operators to enable templated use, a feature lacking from the previous hash structure.

4.2 File Formats

The surface normal estimation code bases all accepted and produced different outputs. Of course, none of them matched the simplified obj file accepted by the homework one base files. The solution was to create a unified ModelFile class (Section 5) for handling the file conversions [Doria 2009]. The VTP file format (Section 5.2) was the common medium of exchange, as it is a very simple format and easily visualized using Paraview, an open source scientific visualization software. The input file for PINE is a PTX file (Section 5.1), and its output is a binary RPRG file containing the surface normals and many other surface attributes. Since the RPRG file is so special purpose, an application was written to store all required information in the ModelFile class for output to VTP as well as generating the RGB Intensity and scale estimation images [Phillips 2009b].

4.3 Naive OpenGL

The original design plan called for an interactive viewer of the different visualizations, based solely off the OpenGL base code provided with the homeworks. The visualization modes would change with a keypress. This design was successful for the initial visualization tests with small test objects, such as the 100k Stanford Bunny. However, this solution didn't scale well and crawled to a halt when the number of vertices increased to millions of points and even more polygons. Additional problems existed with the default camera setup. We needed to dynamically center the range scans, adjust the clipping plane and view frustrum to fit the larger scale building scans. [Doria 2009]. Ultimately, we decided to remove all OpenGL aspects from the code. We instead rewrote the code to create colored VTP files for visualization with Paraview. [Phillips 2009b]

4.4 Point Correspondences

It was originally thought to be a trivial task to take the difference between the values stored in the points of two meshes. Each point in the input data has a unique point location, which was going to be used to identify it for differencing. PINE however, smooths the input data, in doing so alters the point positions slightly. We needed to find another mechanism of aligning the data. We briefly considered implementing a nearest neighbor algorithm to establish correspondences, but then remembered that each point also has a unique grid index from the original range image. It was non-trivial to retain this information between format conversions and algorithm runs. It required changes to both the MSTV base code [Phillips 2009b] and the ModelFile class [Doria 2009].

4.5 Batch Visualizations

In order to fully understand the significance of the differences between the two methods results, many scans need to be examined. In the original OpenGL formulation, we had implemented a save buffer to file option to be used in connection with the auto centering camera [Doria 2009]. After we switched to VTP output, the coloring process could be scheduled in a batch, and the output visualized in Paraview later. Reviewing hundreds of 3D scans one by one in Paraview is still a very time consuming process. With the grid index information now in the code as a result of handling the point correspondence problem, we were able to write out color 2D images that directly corresponded to the original PTX range images [Phillips 2009b]. They images could then be quickly loaded, browsed and compared side-by-side using a standard image viewer.

4.6 Naive Ray Casting

The original Synthetic LIDAR range scanner (Section 6) was implemented using a naive raycaster tested for intersection with every triangle. This worked fine on the 100K Standford bunny, but could take hours on mesh's with millions of triangles. We implemented an Octree (Section 6.1) spatial data structure to cut down the time with dramatic effect [Doria 2009].

4.7 Normal Statistics

While visualizing the differences in normals is great for a qualitative analysis of performance, we needed to gather some statistics to make more quantitative claims. The visualize code was extended to gather the mean and standard deviation of the angular difference between the normals. We gathered histogram data for a finer representation of the distribution of error. [Phillips 2009b]

4.8 Memory Constraints

After running a batch job of over a hundred images, it wasn't until combining data that we realized that some visualizations didn't successfully complete. Reviewing the logs and performing some experiments revealed that the system was running out of memory for some of the larger scans. The 32 bit architecture of the machine used was only able to address between 3 and 4 GB of memory. Eight scans were unable to be visualized, the largest of which contained over 6 million points. After replacing the unused half edge data structure with the vertex-list representation [Phillips 2009b], only five scans were unable to be visualized.

5 ModelFile Class

We have developed a ModelFile class to easily work with several input and output file types.

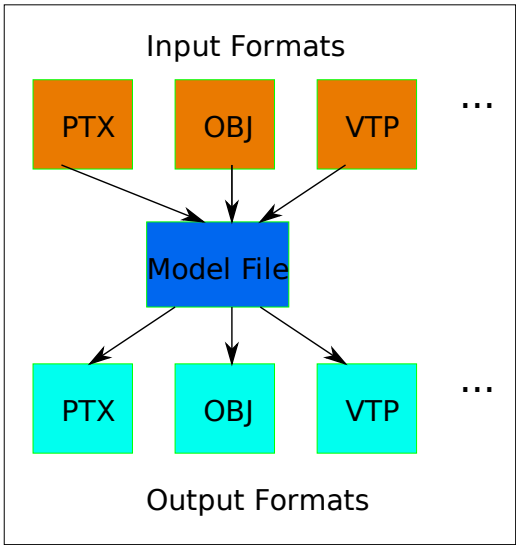

Figure 12: *ModelFile Class Ideology*

The class internally stores the points as a `vector<vgl_point_3d> Points`. VGL is part of the Vision X Library (VXL), and was used extensively for geometric calculations. The triangles are via list of triples indexing into the Points vector, that is, `vector<vector<unsigned int> > Triangles`. This provides a very simple framework for which there is a simple mapping to many standard 3D model formats.

5.1 PTX File Format

PTX is the format in which the Leica LiDAR scanner stores the range scans. It relies completely upon the inherent grid structure that is used to perform the scanning. The first two lines of the file indicate the number of rows (R) and columns (C) of points that compose the scan. The next eight lines indicate the scanners orientation to the world frame. However, in all cases that the authors have seen, the scanner is declared to be the world frame, hence these two matrices are identity. The remaining RC lines are as follows:

```
x y z intensity R G B
```

Where (x, y, z) are the coordinates of the return relative to the scanner, $intensity$ is the percent of light that was reflected, and

(R, G, B) is the color of the pixel that the scanner overlaid onto this point.

5.2 VTP File Format

The VTP format is part of the Visualization Toolkit (VTK) and stands for VTK Polydata. As opposed to the PTX format which is an *organized* set of points, the VTP file stores an *unorganized* set of points. It simply stores an array of point coordinates. A separate array can also be added to store a color corresponding to each point. Another array of the same length is used to store (if available) the normals of each point. If required, an array of three indices into the point array can be stored to indicate triangle connectivity.

6 Synthetic LiDAR Scanning Using Ray Tracing Techniques

To establish a ground truth for comparison, we will create a synthetic LiDAR scanner. Using techniques from ray tracing, a range scan of a 3d model can be produced. We will specify a scanner location (relative to the model) and a list of scan parameters (θ_{min}, θ_{max}, ϕ_{min}, ϕ_{max}, $\Delta\theta$, $\Delta\phi$). The scan is performed by tracing a spherical grid of rays through the world and determining their closest intersection with the model (ray-triangle intersections). The nature of the scanning process leads to organized point clouds, where we have information about the connectivity of the points. A flow diagram of the process is shown in Figure 13.

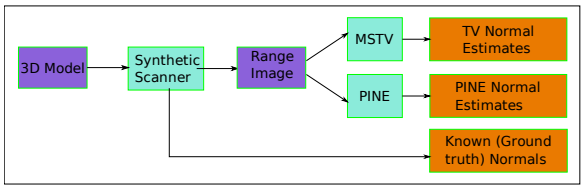

Figure 13: *Synthetic Scanner Motivation*

6.1 Octree

The running time of these ray-triangle intersections can be significantly reduced by storing the model in an octree. The model points are stored in the leaf nodes of an octree with a maximum depth of 5, which we determined to be sufficient experimentally. One could also specify a maximum number of points per leaf, and continue dividing the space until the threshold is broken. The model triangles are stored in the smallest node which contains all three vertices of the triangle. The fourth level of an octree containing the points of the Stanford Bunny is shown in Figure 14. The fourth level of an octree of the triangles of the same model is shown in Figure 15.

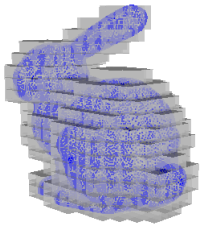

Figure 14: *Octree of Points*

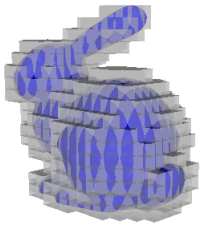

Figure 15: *Octree of Triangles*

6.1.1 Octree Ray-Triangle Intersection

The Ray-Triangle intersection with an octree is straight forward:

- Intersect ray with root box.
- If the box is a leaf, intersect the ray with all the triangles in the box.
- Else, intersect the ray with all of the sub boxes
- Recurse.

6.1.2 Octree Running Time Reduction

Without this spatial data structure, we must intersect each of the R rays with each of the T triangles in the model. We declare the closest intersection to be the intersection of the ray with the model. The running time is clearly RT. The analysis of the running time using the octree is not possible because it is completely dependent on the geometry of the model. Both the actual shape of the model and the density/clusteredness of the points play a large roll in determining the effectiveness of this data structure. An upperbound on the running time is $\frac{RT}{8}$, as a ray can intersect at most two of the eight subnodes of the root. However, in practice much better improvements are seen. In a 250,000 point scan of the Stanford Bunny, the running time was reduced from 20 min 30 sec to 0 min 24 sec, a factor of 51 . This example was run on a 3GHz Intel Pentium 4 computer with 2GB of RAM.

6.1.3 Other Octree Possibilities

There are several modifications to the octree data structure that could be implemented. If axis aligned boxes are used, the Ray-Box intersections become much faster. However, there could be excessive empty space in the tree with some model geometry. Spheres

can be used instead of boxes as the tree nodes. However, there are some implementation details that must be decided. Since non overlapping spheres do not fill a volume (the sphere packing problem), we must use overlapping spheres so points are not lost. This causes some triangles to be intersected twice. However, if the model is rotated, the tree does not have to be reconstructed, so spherical nodes are a viable option in very dynamic scenes.

6.2 Introducing Noise

[Doria 2009] The range scan produced by synthetically scanning a 3d model is completely noise-free. To make the data more realistic, we must add some noise. In a real LiDAR scanner, the noise is primarily along the direction of the ray that the point was sampled from. To model this noise, we add a vector in the direction of the ray with length drawn from a Gaussian distribution with variance σ_{LOS}. There is also slight angular noise. To model this, we add a vector orthogonal to the ray direction with orientation drawn from a uniform distribution $(0, 2\pi)$ and length drawn from a Gaussian distribution with variance σ_{orth}. Both of these types of noise are shown in Figure 16.

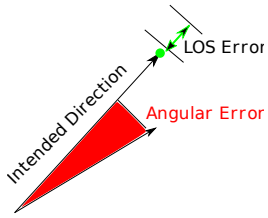

Figure 16: *Scanner Noise Model*

7 Differencing Algorithm

7.1 Preprocessing

The differencing visualization algorithm takes as base input file the results of the PINE algorithm. It takes for differencing a list of VTP files generated by MSTV, which performs estimates at many different scales, with one file per scale. PINE's output files must first be converted to a VTP, with only the surface points represented. This VTP file is then triangulated by first projecting the 3-D points along the line of sight onto a 2-D plane. The 2-D points are then triangulated using Delaunay triangulation, and the resulting connectivity is applied to the corresponding 3-D points. The MSTV VTP files do not need to be triangulated as their points will only be used if there is a corresponding point from the PINE VTP, which contain the geometry.

7.2 Differencing

The grid indices of each point from the original PTX file are used to align the data between the two input files. Since there are usually multiple normals assigned to each point in MSTV, we need to decide which normal to keep. There are two options, specified by a command line flag. One mode will keep the normal that differs least corresponding point from the base PINE file. Another mode is to keep the point that MSTV has decided has the highest saliency by comparing the "surfaceness" value stored in the VTP output from MSTV.

7.3 Colorization

To visually indicate the differences in scalar values, we construct a spectrum of colors using the HSV color space. We fix H and S, and map $(\Delta_{min}, \Delta_{max}) \rightarrow (blue, red) \rightarrow (V = 240, V = 360)$. The HSV color space is shown in Figure 17.

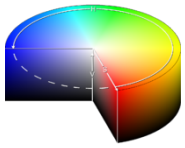

Figure 17: *HSV Color Space*

8 Observations

The synthetic data demonstrates that both techniques produce normals that closely match ground truth.

8.1 Comparison of MSTV To Ground Truth

Figure 18 shows the ground truth normals (top) compared to the MSTV normal estimations (bottom), both using axis coloring.

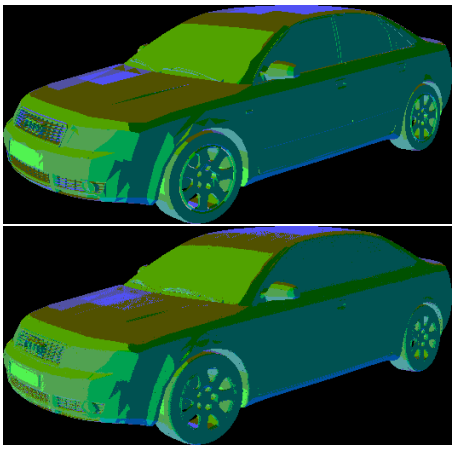

Figure 18: *Comparison of Ground Truth Normals to Normals Computed Using MSTV (Axis Coloring)*

Figure 19 shows the ground truth normals (top) compared to the MSTV normal estimations (bottom), both using dot product coloring.

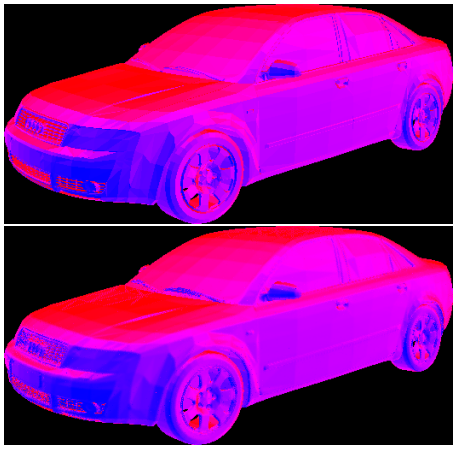

Figure 19: *Comparison of Ground Truth Normals to Normals Computed Using MSTV (Dot Product Coloring)*

Figure 20 shows the difference between ground truth normals and the MSTV normal estimations.

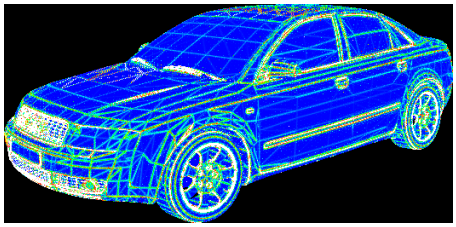

Figure 20: *Coloring of Differences Between MSTV and Ground Truth*

It is interesting to note that with MSTV, the normals behave very poorly at polygon boundaries. Clearly this is not an issue in real data, but there are many real world examples that behave similarly to polygon boundaries (ie. corners of objects).

8.2 Comparison of PINE To Ground Truth

Figure 21 shows the ground truth normals (top) compared to the PINE normal estimations (bottom), both using axis coloring.

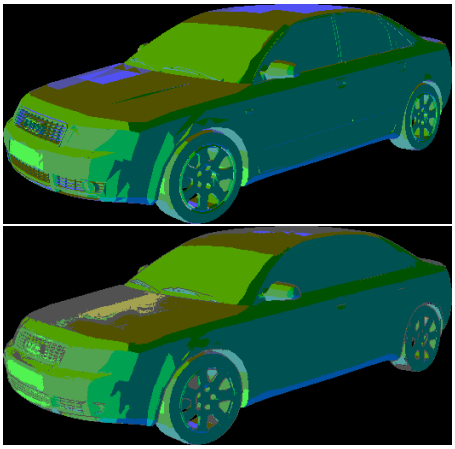

Figure 21: *Comparison of Ground Truth Normals to Normals Computed Using PINE (Dot Product Coloring)*

Figure 22 shows the ground truth normals (top) compared to the PINE normal estimations (bottom), both using dot product coloring.

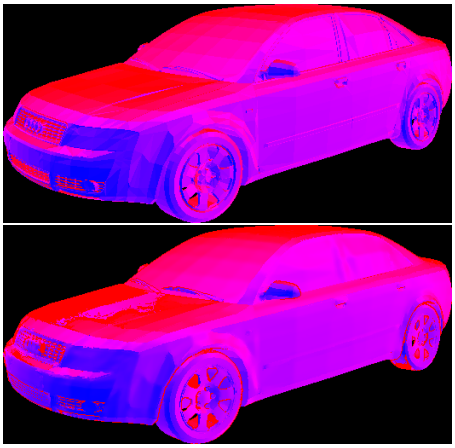

Figure 22: *Comparison of Ground Truth Normals to Normals Computed Using PINE (Dot Product Coloring)*

Figure 23 shows the difference between ground truth normals and PINE normal estimates.

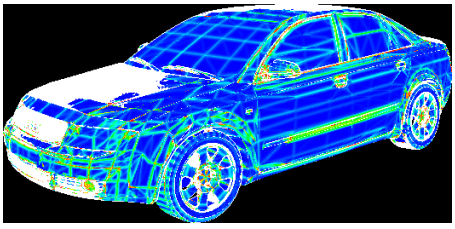

Figure 23: *Coloring of Differences Between PINE and Ground Truth*

We can see the PINE produces extremely accurate normal estimations on the synthetic data.

8.3 MSTV/PINE Algorithm Observations

MSTV can be seen (Figure 24) to exhibit Moire banding and aliasing effects in its surface normal estimates of high spatial frequency structures such as brick work, a result of the sampling process.

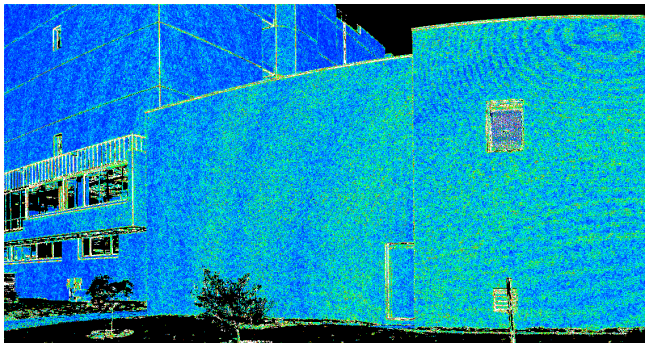

Figure 24: *Moire Banding of MSTV*

The strongest observation from examining the output of the differnce visualizations is that PINE produces much smoother normals than MSTV. Acting as a low-pass filter, it smooths away noise. However, it retains less fine surface detail than MSTV. The two algorithms also tend to differ in their estimation of very undersampled oblique surfaces. Visualizing the additional surface attributes calculated by the two methods has brought a great deal of insight about the failure of MSTV on such regions. MSTV appears to produce mostly incorrect normals, with "good" normals speckled in correspondance with points that were sampled at many scales. Figure 25 demonstrates this.

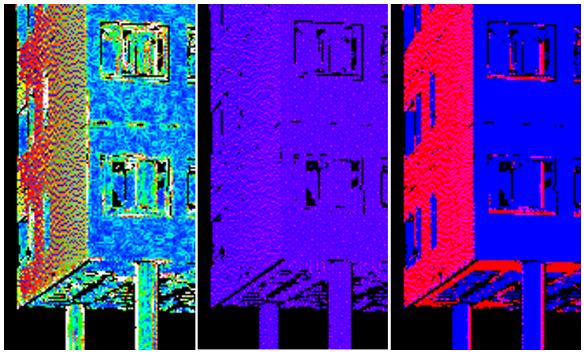

Figure 25: *Good Normals Speckled Where MSTV Sampled Points At Many Scales*

The oblique plane on the right image should be all pink. Instead it is red, with pink speckels. The pink speckels correspond to the points of high sampling (Red Points, Middle Image), that agreed with the results of PINE. (Blue/Green Points, Left image).

9 Conclusions

We have presented a method that visualizes the differences between the surface normal estimates generated by the PINE and MSTV estimation algorithms. We have compared the algorithms performance to ground truth normals obtained by producing synthetic range scan of a CAD model. We have also shown how visualizing other surface attributes can play an enormous role in understanding areas in which the algorithms perform well, or perform less well than desired.

References

DORIA, D., 2009. These elements of the project were accomplished by david, April.

E.-Y. KANG, I. C., AND MEDIONI, G. 2002. Robust affine motion estimation in joint image space using tensor voting. *International conference on pattern recognition*.

G. MEDIONI, M.-S. L., AND TANG., C.-K. 2000. A computational framework for segmentation and grouping. *Elsevier*.

GUY, G., AND MEDIONI, G. 1996. Inferring global perceptual contours from local features. *International Journal of Computer Vision*.

KING, B. 2008. *Range data analysis by free-space modeling and tensor voting*. PhD thesis, Rensselaer Polytechnic Institute.

L. REYES, G. M., AND BAYRO, E. 2007. Registration of 3d points using geometric algebra and tensor voting. *International Journal of Computer Vision*.

LEVOY, M., P. K. C. B. R. S. K. D. P. L. G. M. A. S. D. J. G. J. S. J., AND FULK, D. 2000. The digital michelangelo project: 3d scanning of large statues. *27th Annual Conference on Computer Graphics and interactive Techniques International Conference on Computer Graphics and Interactive Techniques*.

MORDOHAI, P., AND MEDIONI, G. 2006. Tensor voting: A perceptual organization approach to computer vision and machine learning. *Synthesis Lectures on Image, Video, and Multimedia Processing*.

PHILLIPS, C. 2009. *Prioritized IRLS Surface Normal Estimation and Point Classification*. Master's thesis, Rensselaer Polytechnic Institute.

PHILLIPS, C., 2009. These elements of the project were accomplished by cody, April.

TANG, J. J. C.-K. 2003. Image repairing: robust image synthesis by adaptive nd tensor voting. *Computer Vision and Pattern Recognition*.

TONG., W.-S. 2004. *A Complete Theory on 3D Tensor Voting for Computer Vision and Graphics Applications*. PhD thesis, Hong Kong University of Science and Technology.