TIN generation and point-cloud compression for vehicle-based mobile mapping systems

Hiroshi Masuda[1*] and Jun He[2]

1) The University of Electro-Communications
2) The University of Tokyo

* Coresponding author.
 Name: Hiroshi Masuda
 Tel: +81-42-443-5408.
 Email: h.masuda@uec.ac.jp
 Address: 1-5-1 Chofugaoka, Chofu, Tokyo 182-8585 Japan

TIN generation and point-cloud compression
for vehicle-based mobile mapping systems

Abstract:
The vehicle-based mobile mapping system (MMS) is effective for capturing dense 3D data of roads, roadside objects and buildings. Since discrete points are not convenient for many application systems, a triangulated irregular network (TIN) is often generated from point-clouds. However, TIN data require two or three times larger storage than point-clouds. If TIN models can be promptly generated while loading point-clouds, it would not be necessary to store huge TIN models on a hard disk. In this paper, we propose two efficient TIN generation methods according to types of laser scanners. One is the line-by-line TIN generation method, and the other is the GPS-time based method. These methods can quickly generate TIN models based on scan lines of laser scanners. In addition, we introduce a new compression method to reduce the loading time of point-clouds. Our compression method is also based on the scan lines of laser scanners. Since points captured by a MMS tend to be positioned on nearly straight lines, their data size can be significantly reduced by coding the second order differences.

Keywords: Mobile mapping system; Point-cloud; Laser scanning; Triangulated irregular network; Mesh generation; Geometry compression.

1. INTRODUCTION

The vehicle-based mobile mapping system (MMS) is effective for capturing 3D data of roads, roadside objects and buildings. A MMS is a vehicle on which laser scanners, cameras, IMU and GPS navigation systems are mounted. When high-performance laser scanners are mounted on a MMS, hundreds of thousands of 3D coordinates are measured every second. High-density point-clouds of real world environments are useful for many fields, such as civil engineering, architecture, and infrastructure management.

Many researchers have studied on the detection of roads, buildings, and roadside objects from MMS data [1-9]. Since discrete points are not convenient for many application systems, a triangulated irregular network (TIN) is often generated from point-clouds [10-13]. TIN models are useful for rendering, detection of bumps and potholes on roads, object recognition, and so on. TIN models are a set of triangles that share common vertices. Since each triangle maintains connectivity with adjacent triangles and vertices, geometric properties such as gradient and curvature can be calculated using neighbor vertices.

When a MMS captures point-clouds in city-scale driving, the data volume is often more than a terabyte. A huge volume of MMS data leads to a serious problem in TIN generation. Since the data size of TIN models is twice or three times larger than that of point-clouds, it is not desirable to store both TIN models and point-clouds. If we can promptly generate TIN models while loading point-clouds, we do not have to store huge TIN models in a hard disk.

In this paper, we discuss efficient TIN generation methods for huge point-clouds. Our goal is to promptly generate TIN models from point-clouds without storing additional data. Since the computation cost of general-purpose TIN generation is expensive, we develop special-purpose TIN generation methods for MMS data.

Even when only point-clouds are maintained on a hard disk, their data size is still large for storing and loading. Therefore we also discuss compression methods for point-clouds captured by a MMS. To reduce loading time, simple and fast decompression schemes are strongly required as well as high compression rates.

In the following sections, we discuss the characteristics of point-clouds captured by typical laser scanners. In Section 3, we propose a TIN generation method for the line-by-line type of laser scanners. In Section 4, we also propose a different TIN generation method for the spiral type of laser scanners. In Section 5, we propose a compression method that is suitable for point-clouds captured by MMSs. In Section 6, we show the experimental results for our methods. Finally, we conclude our research in Section 7.

2. Types of laser scanners on MMS

2.1. Laser scanners on MMS

Figure 1 shows an example of MMSs. In this MMS, laser scanners, CCD cameras, GPSs, and an IMU are mounted on the roof of a vehicle. These devices are synchronized, and their outputs have common time stamps. GPSs and an IMU are used to identify positions and attitudes of the vehicle based on the global coordinate system. Laser scanners capture 3D coordinates on the scanner-centered coordinate system. Since the relative positions of laser scanners are known with respect to the vehicle, local coordinates of points can be converted into global coordinates. Digital images from CCD cameras are used to add colors to point-clouds.

Laser scanners capture a wide range of point-clouds by changing the direction of laser beams. We can classify laser scanners into two types according to scanning angles. One type of scanner emits laser beams in limited ranges line-by-line, while the other type of scanner continuously emits laser beams in a spiral manner with a 360-degree rotation. In this paper, we call these two types of scanners the line-by-line type and the spiral type. While the line-by-line type is typically much lower in cost than the spiral type, the spiral type can output a higher resolution of point-clouds. The users of MMSs typically select one or the both types based on budgets, objects to be measured, and requirements on the resolution and accuracy.

In our experiments, we used two different laser scanners mounted on MMSs. Table 1 shows their specifications. SICK LMS 291 is the line-by-line type, which emits laser beams within 180 degrees. RIEGL VQ-250 is the spiral type, which emits laser beams continuously. The both scanners are widely used by survey companies in Japan.

Points captured by a MMS are stored in files in the order of measurement. Each point has a 3D coordinate, GPS time, and intensity values. GPS time indicates the time when points are measured. RGB colors may be optionally added in post-processing phases by projecting point-clouds on calibrated digital images.

Figure 1. Mobile mapping system [14]

Table 1: Specification of laser scanners [15,16]

Laser Scanners	SICK LMS291	RIEGL VQ-250
Sampling Speed [points/sec]	54,000	300,000
Rotation Frequency [Hz]	75	100
Scanning Type	Line-by-Line	Spiral

2.2. Line-by-line type of scanners

The line-by-line laser scanner captures point-clouds in limited angles, as shown in Figure 2(a). In this type of scanner, point-clouds are separated into scan lines within the specified angle, as

shown in Figure 2(b). When point-clouds are captured using the line-by-line laser scanner, they can be sequentially processed line by line.
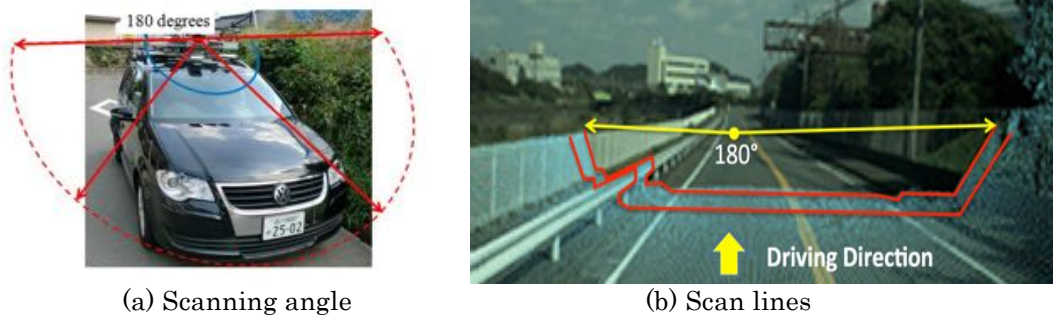

(a) Scanning angle   (b) Scan lines

Figure 2. Line-by-line scanning

2.3. Spiral type of scanners

The other type of laser scanner captures point-clouds in a spiral manner within 360 degrees, as shown in Figure 3. Since a MMS moves forwards while laser beams are rotating, the trajectory of laser beams is like a spiral. This type of scanner outputs a sequence of points that have no explicit boundaries of scan lines.
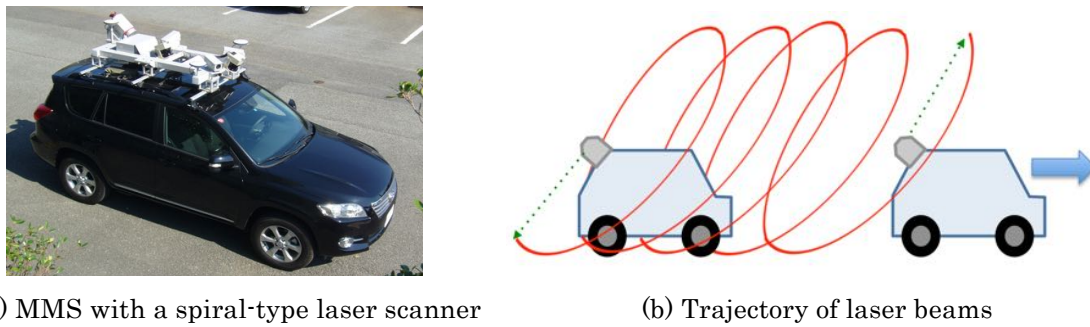

(a) MMS with a spiral-type laser scanner   (b) Trajectory of laser beams

Figure 3. Scanning in a spiral manner

3. TIN generation for line-by-line laser scanners

Since general-purpose TIN generation is not very efficient for huge point-clouds, we consider new methods that utilize the characteristics of point-clouds captured by MMSs. First, we consider point-clouds captured by the line-by-line laser scanner. Point-clouds from this type of laser scanner can be quickly triangulated by creating triangles between scan lines.

Since points are stored in the order of measurement, scan lines can be generated by connecting points sequentially. Figure 4 shows two scan lines: *L1* and *L2*. In our algorithm, a point is selected from *L1*, and the nearest point in *L2* is then searched. When the distance of the nearest point is smaller than threshold *d*, an edge is generated between two points. Once an edge is generated, the next nearest point is searched for only within the next *n* points in *L2* to improve efficiency. In this paper, we set the value of *n* to 5. The threshold value *d* is determined depending on the speed *s* (m/sec) of a vehicle and the rotation speed *f* (Hz) of a laser scanner. Since we can estimate the interval of scan lines as s/f, we specify the threshold value *d* as ks/f, where *k* is a constant value. In our experiments, we set the value of *k* to 1.2.

Then, polygons are subdivided into triangles. An example of generated TIN models is shown in Figure 5. The left figure is a wireframe of a road surface, and the right figure is a rendered image. Points on scan lines are not ordered at equal distances when laser beams are not returned because of a certain reason, such as potholes on roads. As shown in circles Figure 5, our method can correctly generate triangles even when some points are missing on scan lines.
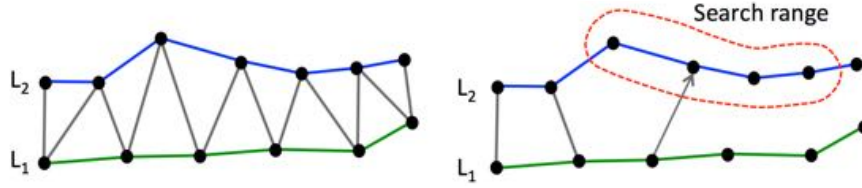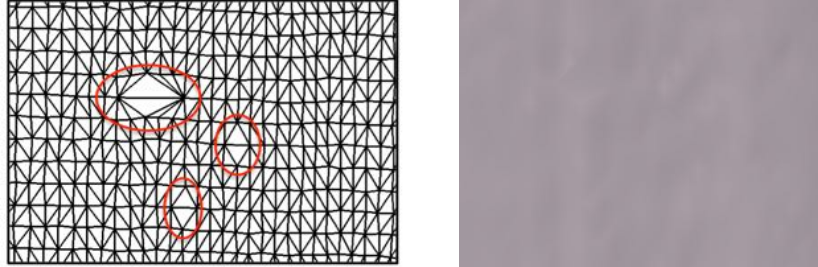
Figure 4. Search range for the next neighbor point
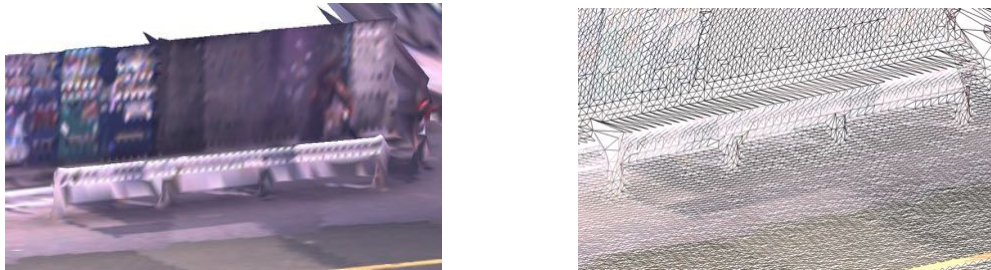

Figure 5. TIN model of a road surface

Figure 6. TIN models generated using the line-by-line method

Figure 6 shows TIN models generated using line-by-line triangulation. The point-cloud was captured with SICK LMS 291 with 37.5 Hz. We set the parameter s to 11.1 m/sec, which is the typical legal speed in Japan. In this example, 9.8 thousand irregular points were detected and repaired. Since our method searches only local neighbors, TIN models can be very efficiently generated.

4. TIN generation for the spiral type of laser scanner

4.1. Rotation frequency and GPS Time

Since point-clouds captured by spiral-type laser scanners do not have explicit scan line boundaries, line-by-line triangulation is not applicable. We introduce a different triangulation method for spiral-type laser scanners. We describe sequential points as $\{\mathbf{p_i}\}$ $(i = 1,..,n)$, where $\mathbf{p_i} = (x_i, y_i, z_i, t_i)$ is a coordinate, and t_i is GPS time.

Since spiral-type laser scanners tend to output huge point-clouds, quickly generating millions of triangles is necessary. Scan lines are also generated by connecting points in the temporal order, as shown in Figure 8. In this paper, the distance between points on a scan line is described as a *pitch*, and the distance between adjacent scan-lines as an *interval*. To generate a triangle using points $\mathbf{p_i}$ and $\mathbf{p_{i+1}}$ in Figure 7, the third vertex of the triangle has to be detected in the adjacent scan line. If disconnected neighbor points can be quickly detected, TIN models can be efficiently generated. Therefore we consider restricting search areas based on the cycle of laser beams.
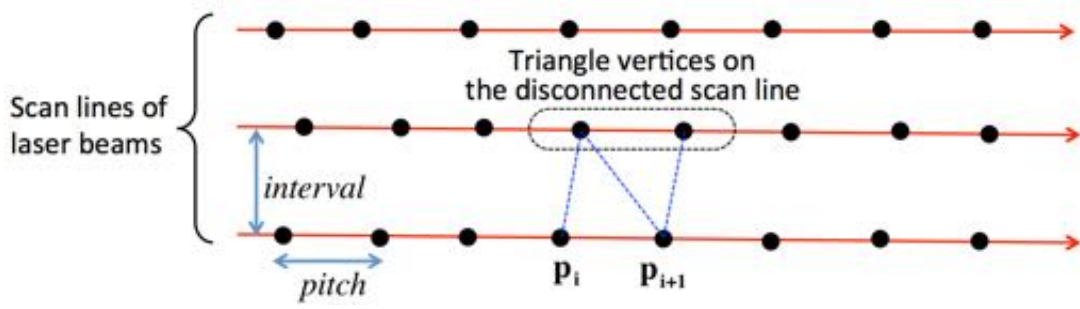
Figure 7. Scan lines of points captured by spiral-type laser scanners

Laser beams from a laser scanner rotate within 360 degrees in a cycle of the constant frequency f Hz, which is given as the specification of laser scanners. Then disconnected neighbors are expected to be found around $1/f$ second later. We denote $1/f$ as T.

We verified the accuracy of the frequency using point-clouds captured with 100 Hz (T=1/100 sec). Figure 8 shows that the frequency can be a good marker to restrict the search ranges of the nearest points. In this figure, points are sequentially connected with lines, and the colors of the lines are changed to green or magenta every T second. Circles show where colors are changed. Since each circle position is close to the nearest points of the previous circle, the search range for the nearest point can be restricted based on the GPS time and the rotation frequency.

Even when the frequency of a laser scanner is unknown, it can be estimated by detecting the nearest points. We calculated differences of GPS time between each point and its nearest disconnected point, and estimated rotation frequencies as the mode values. We evaluated 100 point-clouds. Then errors of rotational frequency were less than 0.2% in all cases.
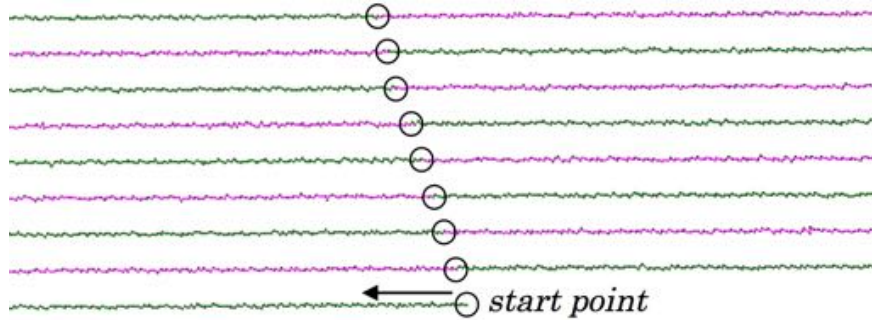

Figure 8. Points every 1/f second

4.2. TIN generation algorithm for spiral-type laser scanners

Figure 9 shows the process of our TIN generation algorithm. We explain our algorithm using a point-cloud in Figure 9(a). First, points are connected in the temporal order when pitches are less than threshold d_{pitch}. Then scan lines are generated, as shown in Figure 9(b).

The nearest neighbor of each point is searched in the adjacent scan line. In our method, the candidates of the nearest point are selected using the rotation frequency. For each point $\mathbf{p_i}$, a point in T second later is selected using the following formula. We denote the selected point as $\mathbf{p_j}$.

$$\underset{j}{\mathrm{argmin}}\left|(t_i + T) - t_j\right| \qquad (j > i) \tag{1}$$

The candidate points are selected from neighbor points of $\mathbf{p_j}$. We define the candidates as $\{\mathbf{p_k}\}$ $(j - w \leq k \leq j + w)$, where w is the size of search ranges. The value of w is determined based on the accuracy of rotational frequency. In this paper, we set the value of w to 5.

Then the nearest point of each point is selected from the candidates and connected by an edge, as shown in Figure 9(c). We generate polygonal faces by traversing edges, and finally, we subdivide polygonal faces into triangles using the Delaunay triangulation, as shown in Figure 9(d).

When scan lines include outliers, irregular scan lines may be generated. Scan lines are

intersecting in the left of Figure 10(a), and a scan line turns back in the right figure. In such cases, our method produces reversed and overlapping triangles, as shown in Figure 10(b). We repair such cases by locally applying the Taubin operator, which is a well-known smoothing operator for TIN models [17]. The Taubin operator is a low-pass filter that selectively smoothens sharp vertices. We apply the Taubin operator only to points that are shared by triangle faces with opposite normal vectors. Figure 10(c) shows triangles repaired using the Taubin operator. In this figure, modified points are shown in circles. Since our method only moves a small number of irregular points, it can repair distorted triangles very quickly.

Figure 11 shows a TIN model generated from 2.7 million points. This TIN model consists of 5.0 million triangles. The computation time for triangulation was 5.0 seconds, while loading time was 7.6 seconds. Since point density is very high, dense triangles are generated. Figure 12 shows a TIN model of roadside objects. The result shows that that our method can convert dense point-clouds into TIN models without down sampling.

Although the computation time of triangulation was faster than the file loading time, triangulation of large point-clouds is time-consuming. Therefore, we improve efficiency by introducing adaptive triangulation.
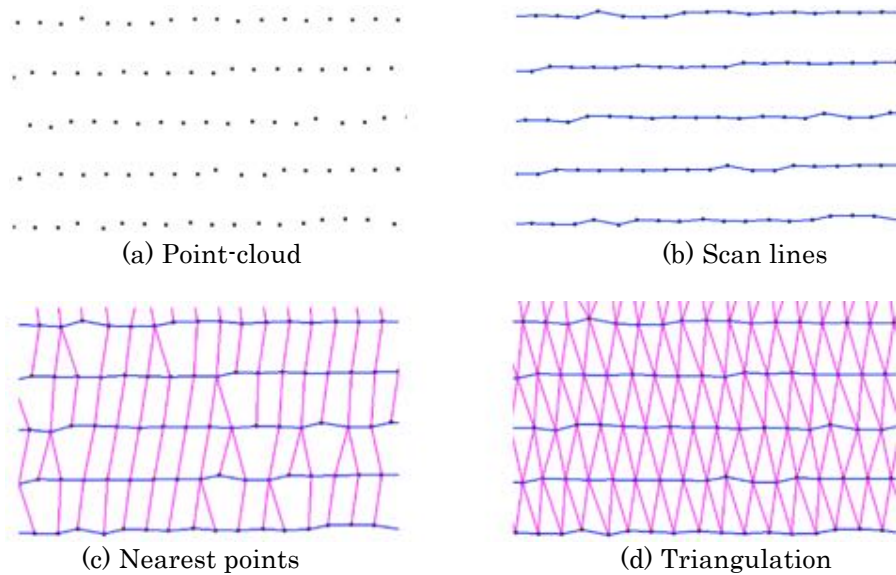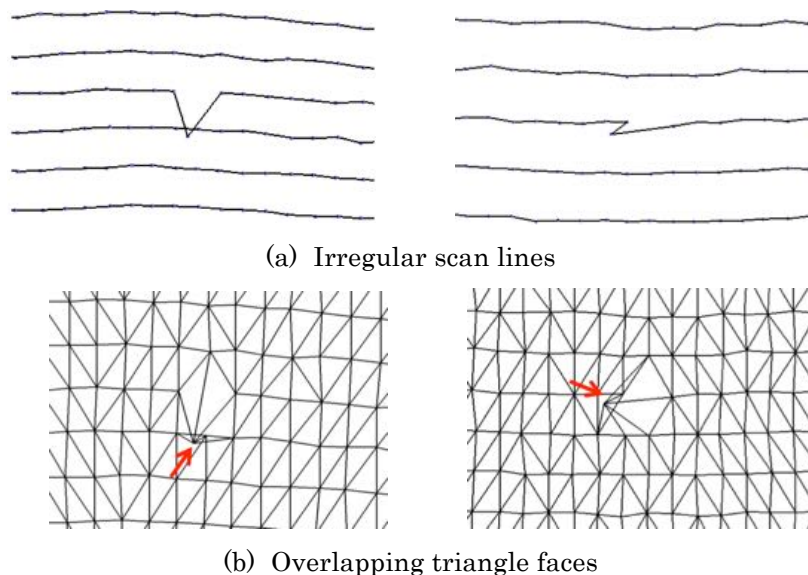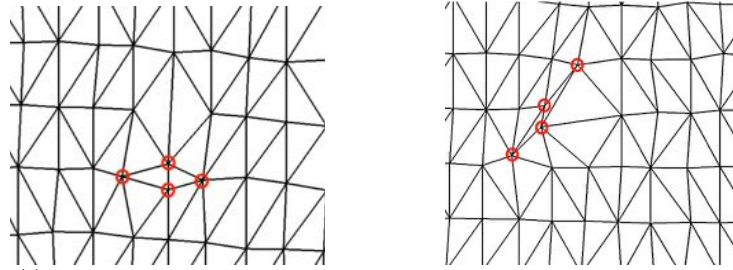

(a) Point-cloud

(b) Scan lines

(c) Nearest points

(d) Triangulation

Figure 9. TIN generation for spiral-type laser scanners

(a) Irregular scan lines

(b) Overlapping triangle faces

(c) Triangles repaired by moving points in circles

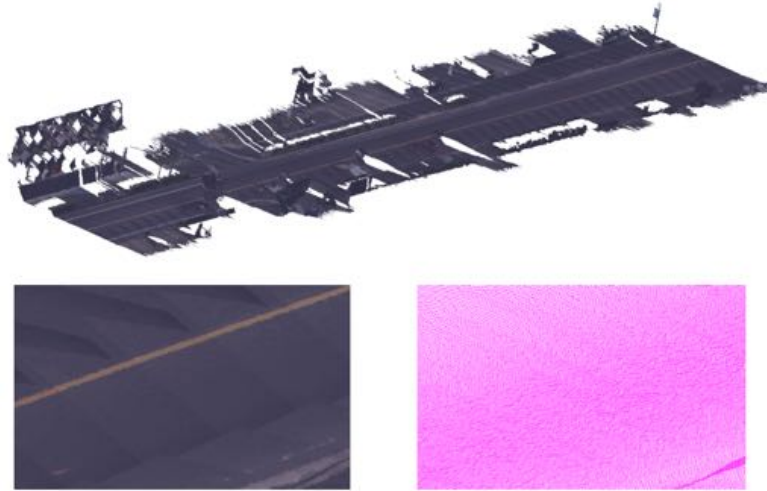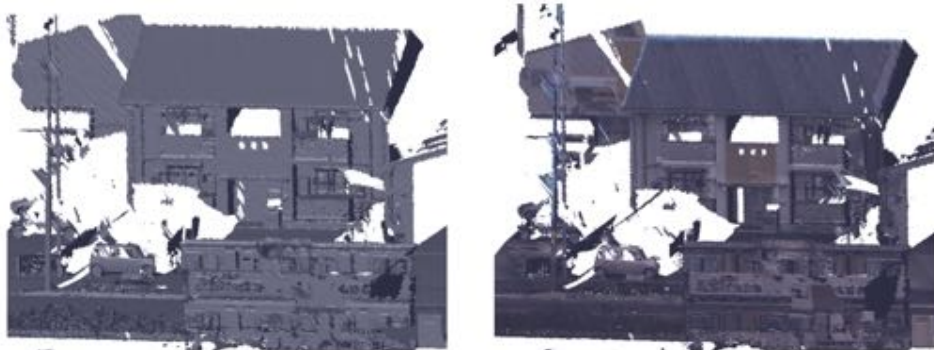Figure 10. Repair of distorted triangles caused by outliers

Figure 11. TIN model generated from 2.7 million points

(a) Rendering using Gouraud shading   (b) Rendering using RGB colors

Figure 12. TIN model of roadside objects

4.3. Adaptive TIN generation

In the case of most spiral-type laser scanners, the lengths of scan-line intervals are much larger than the lengths of spacing, as shown in Figure 13(a). When all points in scan lines are used for triangulation, a lot of thin triangles are generated, as shown in Figure 13(b). An unnecessarily large number of thin triangles cause distorted rendering and long computation time.

To avoid thin triangles and improve performance, we introduce adaptive triangulation that adaptively eliminates points on scan lines.

We explain our algorithm using Figure 14. We suppose that the start point of triangulation is $\mathbf{p_i}$ on the scan line 1. Then the nearest point is searched from points on the adjacent scan-line 2 using the method described in Section 4.2. We denote the nearest point of $\mathbf{p_i}$ as $\mathbf{q_i}$ and the distance between $\mathbf{p_i}$ and $\mathbf{q_i}$ as d.

Then the distance $q \cdot d$ is calculated. We define q as the quality parameter, which is determined by the user to control the density of triangles. Points on each scan line are reduced so

that the distances between adjacent points become approximately $q \cdot d$. In Figure 14, $\mathbf{p_j}$ and $\mathbf{p_k}$ are selected on scan line 1, and $\mathbf{q_j}$ and $\mathbf{q_k}$ are selected on scan line 2.

The nearest points are searched from the reduced points, as shown by arrows in Figure 14. Finally, polygonal faces are subdivided into triangles by adding dashed lines.

The distance d is calculated each time when two continuous scan-lines are processed. In our method, points on each scan line are reduced once. In Figure 14, if the next scan-line 3 is processed, points only on the scan-line 3 are reduced because the scan-line 2 is previously reduced in triangulation between the scan-lines 1 and 2.

We applied adaptive triangulation to scan-lines in Figure 13. As shown in Figure 15(a), the original scan lines include dense points. Figure 15(b) shows reduced points by setting the quality parameter to 1. The result in Figure 15(c) shows that thin triangles in Figure 13(b) are extensively improved using the adaptive triangulation. The performance of triangulation can also be improved using the adaptive triangulation. In our experiments, the computation time of triangulation of the point-cloud in Figure 11 was reduced from 5.0 seconds to 0.24 seconds by using the adaptive triangulation with $q = 1$.

(a) Points on scan lines   (b) Thin triangles
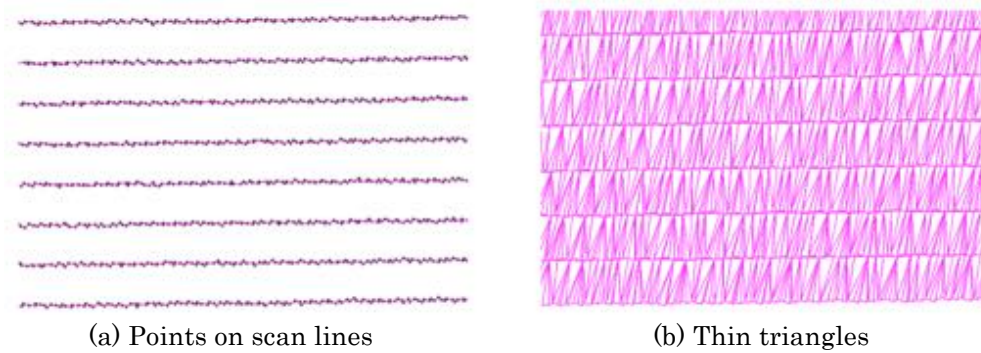
Figure 13. Thin triangles in TIN model

Figure 14. Adaptive triangulation method

(a) Original scan lines   (b) Reduction of points   (c) Result of triangulation
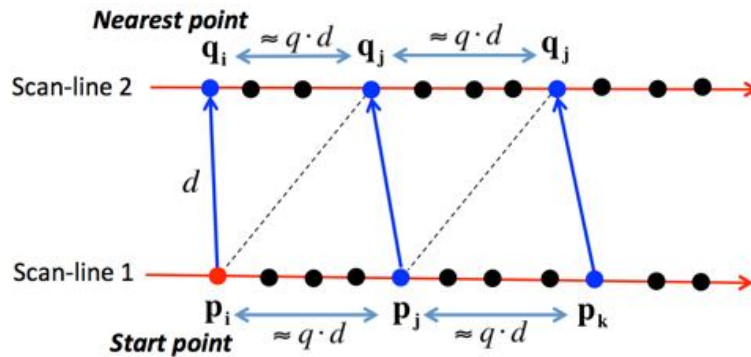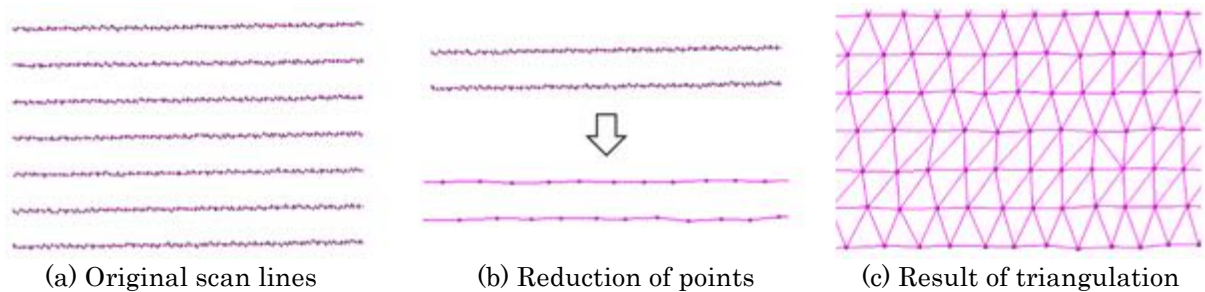
Figure 15. TIN model generated using adaptive triangulation

5. Compression of point-clouds captured by MMS

5.1. Coherency in point sequences

Even when TIN models can be promptly reconstructed from point-clouds, the data sizes of point-clouds are still too large for storing and loading. It is desirable that the data size of a huge point-cloud is reduced and the loading time is shortened.

A variety of compression methods have been proposed for TIN models [18-23]. Although these methods achieve excellent compression rates, they need a lot of computation time to decode large compressed data. To promptly generate TIN models from point-clouds, compressed data have to be quickly decompressed.

In this paper, we propose a new compression method based on scan lines in MMS data. Since most points are captured densely on smooth surfaces, a sequence of points tends to be nearly straight lines. Such linear configurations of points commonly appear on man-made objects, such as roads, walls, buildings, traffic signs, and so on.

In our compression scheme, we suppose that most points are equidistantly distributed on nearly straight lines. If this assumption is satisfied, the coordinates of points can be approximately predicted using previous points. When three points are on a straight line equidistantly, the following second-order difference δ_i becomes 0.

$$\delta_i = \mathbf{p}_i - 2\mathbf{p}_{i-1} + \mathbf{p}_{i-2} \tag{2}$$

Table 2 shows the sequence of coordinates and GPS time in an actual point-cloud. In this table, second-order differences are quantized using their significant digits. While original values are represented using many digits, their second-order differences become very small.

When values are represented as second-order differences, they can be quickly restored as:

$$\mathbf{p}_i = \delta_i + 2\mathbf{p}_{i-1} - \mathbf{p}_{i-2} \tag{3}$$

Since this calculation is extremely simple and efficient, the encoded data can be promptly decoded.

Table 2. Values of a point-cloud and their second-order differences

X	Y	Z	GPS Time	δx	δy	δz	δt
-92951.141	-11432.156	81.777	276887.6793997				
-92951.133	-11432.147	81.777	276887.6794066				
-92951.125	-11432.145	81.779	276887.6794100	0	-2	4	-1
-92951.125	-11432.141	81.781	276887.6794134	-8	1	-2	0
-92951.117	-11432.136	81.779	276887.6794168	8	1	-2	0
-92951.117	-11432.13	81.776	276887.6794202	-8	1	-2	0
-92951.109	-11432.127	81.780	276887.6794237	8	-3	6	1
-92951.102	-11432.122	81.778	276887.6794271	0	2	-5	-1
-92951.102	-11432.116	81.776	276887.6794305	-8	1	-1	0
-92951.094	-11432.113	81.778	276887.6794340	8	-3	4	1
-92951.094	-11432.109	81.778	276887.6794374	-8	1	-1	-1
-92951.086	-11432.104	81.778	276887.6794408	8	1	0	0
-92951.078	-11432.101	81.779	276887.6794442	0	-1	1	0

5.2. Compression of point-clouds

Point-cloud data mainly consist of 3D coordinates and GPS time, which are represented in double precision. Therefore, we consider a compressed representation for coordinates and GPS time.

In our compression scheme, points are connected in the temporal order when pitches are less than the threshold d_{pitch}, and each connected scan line is separately encoded. Suppose that a scan line is represented as $\{\mathbf{p}_k\}$ $(r \le k \le s)$. Then these points can be compactly represented using $\mathbf{p}_r$, $\mathbf{p}_{r+1}$, and $\{\delta_k\}$ $(r + 2 \le k \le s)$, where $\delta_k = (\delta x_k, \delta y_k, \delta z_k, \delta t_k)$.

We encode the second order difference of coordinate $(\delta x_k, \delta y_k, \delta z_k)$ using multiples of M bits, as shown in Figure 16. The M-th bit indicates whether there are following bits. Signs of positive or negative are also represented by a string of bits for each scan line. We used $M=4$ in our experiments, because the compression ratio was the highest at this setting. We encode each scan line separately. When a sequence of points in a scan line terminates, the delimiter bits [0,0,0,1] is described in a file to indicate the end of a scan line.

As shown in Table 1, most second-order differences of GPS times are 0, 1, or -1. This is because points are sampled at the equal time interval. Other values are generated only when laser beams are not returned and points are missing. The values 1 and -1 are caused because of round-off errors of GPS time. In our evaluation using 50 million points, 94.6 % of δt were 0, 1, or -1. Based on these observations, we encode δt using 2 bits, as shown in Table 3. When the code is '10' (others), the value of δt is written following the 2-bit code using the number of bits in which the maximum signed-integer can be represented.
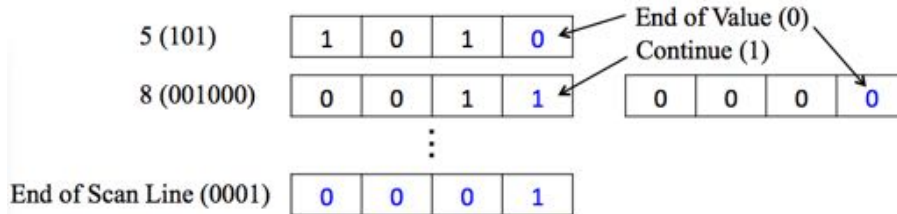

Figure 16: Representation of second order differences

Table 3. Encoding of GPS time

Code	00	01	11	10
δt	0	1	-1	Others

6. Experimental Results

6.1. Performance evaluation for line-by-line TIN generation

We evaluated the line-by-line triangulation using point-clouds captured by SICK LMS 291 with the 180-degree scanning angle. Point-clouds in our experiments were captured for a highway and a residential area in Japan. We measured CPU times to generate TIN models. Since loading time strongly depends on storages, we measured the processing time after points were loaded on RAM. We used a PC with 3 GHz Intel Core i7 CPU and 12 GB RAM.

We compared our method to one of the best 2D Delaunay triangulation codes, which was developed by Shewchuk [24,25]. When we applied 2D Delaunay triangulation to point-clouds, we projected all points on the x-y plane.

Table 4 shows the processing time of line-by-line TIN generation. The result in Table 4 shows that our method was 3.9 times faster than Delaunay triangulation. This experimental result indicates that our TIN generation scheme is very efficient for practical MMS data.

Table 4. Timing of TIN generation by the line-by-line method

	Points	Timing	Points/Sec
Delaunay Triangulation	300,000	0.66 sec	0.45 million
Line-by-Line Method	300,000	0.17 sec	1.8 million

6.2. Performance evaluation of TIN generation for the spiral-type laser scanner

We evaluated TIN generation for spiral-type laser scanners. We used point-clouds captured by RIEGL VQ 250 with the 360-degree scanning angle. Since Delaunay triangulation is very time-consuming for huge point-clouds, we did not compare our method to Delaunay triangulation. Streaming Delaunay triangulation [26] is known for the triangulation of huge point-clouds, but this method is not suitable for promptly triangulating point-clouds because it is designed for out-of-core processing and loads files twice for triangulation.

Our algorithm consists of the nearest point search based on GPS time and the triangle generation. We measured CPU time for these two processes. Table 5 shows the computation results. In these examples, TIN models were generated without reducing points. The result shows that our method could generate very large TIN models in a reasonable time.

Table 5. Timing of TIN generation for spiral-type laser scanners

Number of Points	TIN Size		CPU Time (sec)			Points/Sec
	Vertices	Triangles	Neighbor Search	Triangle Generation	Total Time	
897,457	897,457	1,693,340	0.12	1.46	1.58	0.57 million
1,795,316	1,795,316	3,360,296	0.30	2.99	3.29	0.55 million
2,691,974	2,691,974	4,997,336	0.45	4.55	5.00	0.54 million
3,587,535	3,418,403	6,621,428	0.62	5.81	6.43	0.56 million
4,484,626	4,271,762	8,258,613	0.76	7.55	8.31	0.54 million

6.3. Performance evaluation of adaptive TIN generation

We also evaluated the timing of adaptive TIN generation by setting the quality parameter to 1. Since adaptive triangulation significantly reduced the number of triangles, it was able to triangulate very large point-cloud files in a very short time. The results are shown in Table 6.

Table 6. Timing of adaptive TIN generation ($q = 1$)

Number of Points	TIN Size		CPU Time (sec)		
	Vertices	Triangles	Neighbor Search	Triangle Generation	Total Time
897,457	87,175	162836	0.02	0.06	0.08
1,795,316	173,177	321917	0.03	0.13	0.15
2,691,974	259,622	480004	0.05	0.19	0.24
3,587,535	345,247	635427	0.07	0.25	0.32
4,484,626	432,495	793143	0.08	0.32	0.40
5,382,313	516,691	940771	0.10	0.38	0.48
6,279,047	602,611	1098461	0.12	0.46	0.57
7,177,817	692,522	1261706	0.13	0.51	0.64

6.4. Quality and performance of adaptive TIN generation

In adaptive triangulation, the quality of TIN models is controlled by the quality parameter q. We evaluated trade-offs between the quality and the performance using different quality parameters. Figure 17 shows rendering results of TIN models. In this figure, the resolutions of TIN models are degraded when large quality parameters are specified. We estimated the resolution of each TIN model by calculating the average pitches of reduced points on the road surface. While the average pitch is 1.4cm in the case of $q = 0.2$, it is 5.8 cm in the case of $q = 0.8$. The larger the pitch is, the more degraded the TIN model is. In Figure 17, grooves on the wall and a thin pole are shown in circles. They are gradually vanishing when the quality parameter is larger and therefore the average pitch is larger.

Table 7 shows CPU time when the different quality parameters are used. This result shows that the performance is related to the number of triangles and there is a trade-off between the quality and the performance of TIN generation.
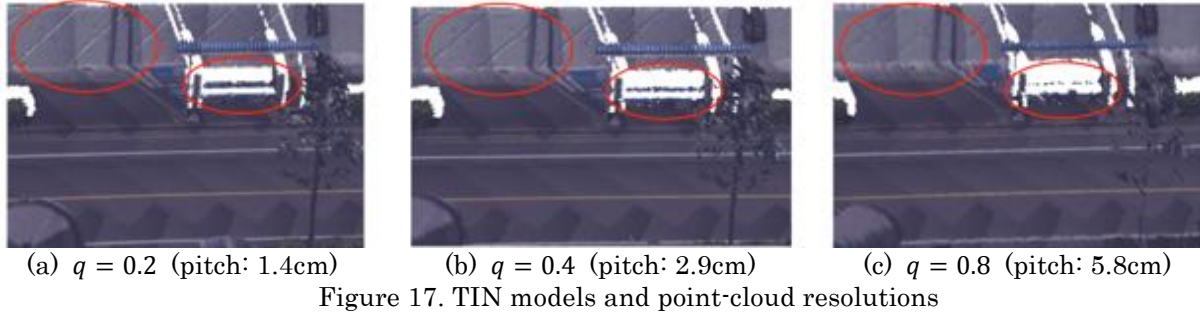

(a) $q = 0.2$ (pitch: 1.4cm)　　　　(b) $q = 0.4$ (pitch: 2.9cm)　　　　(c) $q = 0.8$ (pitch: 5.8cm)

Figure 17. TIN models and point-cloud resolutions

Table 7. Timing of adaptive TIN generation with different quality parameters

Quality Parameter	TIN Model			CPU Time (sec)		
	Vertex	Face	Pitch of Points (cm)	Neighbor Search	Triangle Generation	Total
(a) $q = 0.2$	575,736	1,111,086	1.4	0.08	0.93	1.01
(b) $q = 0.4$	279,890	533,375	2.9	0.04	0.28	0.32
(c) $q = 0.8$	132,376	247,886	5.8	0.03	0.11	0.14

6.5. Compression ratios

We applied our compression method to practical MMS data. We compared data sizes with ASCII data and binary files. Original values are represented using double precision, but they can be simply reduced to single precision values by calculating the differences from a reference point. In this evaluation, we suppose that each point is represented using 4 single precision values in a binary file. In the ASCII format, each point (x_i, y_i, z_i, t_i) is represented using 44 characters including delimiters, as shown in Figure 15.

In our experimental data, point-clouds are separately stored in files every 300,000 points. We applied our compression method to 200 files and evaluated the average, the best, and the worst data sizes. In this experiment, we set the value of d_{pitch} to 0.3 m. The result is shown in Table 8. Our experimental result shows that our compressed data were much smaller than the binary files.

Table 8. Compression results of point-clouds

	Bytes per point		
	Average	Best Case	Worst Case
ASCII file	44.0	44.0	44.0
Binary	16.0	16.0	16.0
Our Method	5.4	3.7	7.2

7. Conclusions

In this paper, we introduced efficient TIN generation methods for point-clouds captured by MMSs. We proposed two methods according to types of laser scanners; namely, the line-by-line method, and the GPS-time based method. We also proposed adaptive triangulation to reduce thin triangles and improve efficiency. Our experimental results show that our methods can very

quickly generate TIN models. We also introduced a compressed representation for point-clouds. Since points tend to be placed on nearly straight lines, their data size can be significantly reduced by coding second order differences. In our experiments, our method significantly reduced the data sizes of point-clouds.

For future work, we would like to develop TIN generation for overlapping scan lines, because our methods generates distorted TIN models when the scan lines are overlapping. We would also like to investigate a trade-off between quality and performance in more detail and estimate adequate quality parameters.

Acknowledgement

References

[1] W. Rong-ben, G. Bai-yuan, J. Li-sheng, Y. Tian-hong, G. Lie, Study on curb detection method based on 3D range image by laser radar, IEEE Intelligent Vehicles Symposium (2005) 845–848.

[2] C. Yu, D. Zhang, Road curbs detection based on laser radar, IEEE Signal Processing, (2006) 8–11.

[3] K. Kodagoda, W. Wijesoma, A. Balasuriya, CuTE: Curb Tracking and Estimation, IEEE Transactions on Control Systems Technology 14(5) (2006) 951–957.

[4] C.P. Mc Elhinney, P. Kumar, C. Cahalane, T. McCarthy, Initial results from European road safety inspection mobile mapping project, ISPRS Commission V Technical Symposium (2010) 440-445.

[5] J. He, H. Masuda, H. Reconstruction of roadways and walkways using point-clouds from mobile mapping system, Asian Conference on Design and Digital Engineering (2012) Article 100099.

[6] S. El-Halawany, A. Moussa, D.D. Lichti, N. El-Sheimy, Detection of road curb from mobile terrestrial laser scanner point cloud, ISPRS Workshop on Laserscanning (2011).

[7] H. Masuda, Shape reconstruction of poles and plates from vehicle-based laser scanning data, The International Symposium on Mobile Mapping Technology (2013).

[8] H. Lin, J. Gao Y. Zhou, G. Lu, M. Ye, C. Zhang, L. Liu, R. Yang, Semantic decomposition and reconstruction of residential scenes from LiDAR data. ACM Transactions on Graphics, 32(4) (2013).

[9] L. Nan, A. Sharf, H. Zhang, D. Cohen-Or, B. Chen, SmartBoxes for interactive urban reconstruction, Transactions on Graphics, 29(4) (2010).

[10] V.J.D. Tsai, Delaunay triangulations in TIN creation: an overview and a linear-time algorithm, International Journal of Geographical Information Systems 7(6) (1993) 501-524.

[11] A.N. Akel, K. Kremeike, S. Filin, M. Sester, Y. Doytsher, Dense DTM generalization aided by roads extracted from LiDAR data. ISPRS WG III/3, III, 4 (2005) 54-59.

[12] F. Goulette, F. Nashashibi, I. Abuhadrous, S. Ammoun, C. Laurgeau, C. An integrated on-board laser range sensing system for on-the-way city and road modelling. ISPRS Commission I Symposium (2006) 43:1-6.

[13] A. Jaakkola, J. Hyyppä, H. Hyyppä, A. Kukko, Retrieval algorithms for road surface modelling using laser based mobile mapping. Sensors 8(9) (2008) 5238–5249.

[14] Mitsubishi Electric Coop., Mobile mapping system high-precision GPS mobile measurement equipment, http://www.mitsubishielectric.com/bu/mms/index.html.

[15] SICK, LMS200/211/221/291 Laser measurement systems, Technical Description (2006).

[16] RIEGL, Data sheet RIEGL-250 (2012).

[17] G. Taubin, A signal processing approach to fair surface design, ACM SIGGRAPH 95 (1995) 351-358.

[18] M. Isenburg, P. Lindstrom, Streaming meshes, Visualization 2005 (2005) 231-238.

[19] M. Isenbueg, J. Snoeyink, Face fixer: compressing polygon meshes with properties, SIGGRAPH (2000) 263-270.

[20] M. Deering, Geometry compression, ACM SIGGRAPH 95 (1995) 13-22.

[21] G.Taubin, J.Rossignac, Geometry compression through topological surgery, ACM Transactions

on Graphics, 17(2) (1998) 84-115.

[22] C. Touma, C. Gotsman, Triangle mesh compression, Graphics Interface, Vol. 98 (1998) 26-34.

[23] H. Masuda, R. Ohbuchi, Coding topological structure of 3D CAD models, Computer-Aided Design, 32(5-6) (2000) 367-375.

[24] J. R. Shewchuk, Delaunay refinement algorithms for triangular mesh generation, computational geometry, 22(1) (2002) 21–74.

[25] J. R. Shewchuk, Triangle: Engineering a 2D quality mesh generator and Delaunay triangulator, Applied computational geometry towards geometric engineering, Springer Berlin Heidelberg (1996) 203-222.

[26] M. Isenburg, Y. Liu, J. Shewchuk, J. Snoeyink, Streaming computation of Delaunay triangulations, ACM Transactions on Graphics, 25(3) (2006) 1049-1056.