Robust Edge Detection and GPU-Based Smoothing for Extracting Surface Primitives from Range Images

Kunihiko Ikeda[1], Chihiro Matsunuma[1] and Hiroshi Masuda[2]

[1]The University of Tokyo, ikeda@nakl.t.u-tokyo.ac.jp
[2]The University of Tokyo, masuda@sys.t.u-tokyo.ac.jp

ABSTRACT

It is important to construct 3D virtual models of man-made fields in which people work and live. Recent mid-range and long-range laser scanners can be used to acquire 3D shapes of cities, buildings, factories, heavy goods, transportation infrastructure, and so on. However, they tend to produce outliers and very noisy points near silhouettes and sharp edges of objects. This problem makes it difficult to reconstruct bounded faces. In addition, since enormous volumes of point-clouds are captured from a broad range of scenes, efficient processing methods are required. In this paper, we propose a robust edge detection method and an efficient GPU-based smoothing method for reconstructing primitive surfaces. We first calculate straight edge lines and silhouette lines from raw scanned data, and then eliminate noises and outliers by our GPU-based smoothing method for calculating surface equations. Then primitive surfaces are extracted using sharp edges, silhouette lines and surface equations. Our method is useful to robustly extract surface primitives from practical noisy point-clouds.

Keywords: point-cloud, GPU, edge detection, surface detection.
DOI: 10.3722/cadaps.2011.603-616

1 INTRODUCTION

The recent progress on mid/long-range laser scanners has made it possible to measure a broad range of scenes in a short time. These laser scanners can be used for acquiring 3D shapes of cities, buildings, factories, heavy goods, transportation infrastructure, etc. Two types of laser scanners are typically used in surveying large-scale objects. One is the time-of-flight scanner, which measures the round-trip travel time of the laser pulses. The other type is the phase-based laser scanner, which radiates continuous modulated laser pulses and obtains distances using the phase difference between the emitted and received signals. The both types of scanners can capture tens of millions points in a single scan.

Our goal is to reconstruct 3D virtual models of man-made fields in which people work and live. These 3D models are useful for supporting maintenance tasks, asset management, object detection, and the record of current situations. We can observe that many components in man-made fields consist of combinations of simple surfaces, such as planes, cylinders, cones, spheres, and tori, because

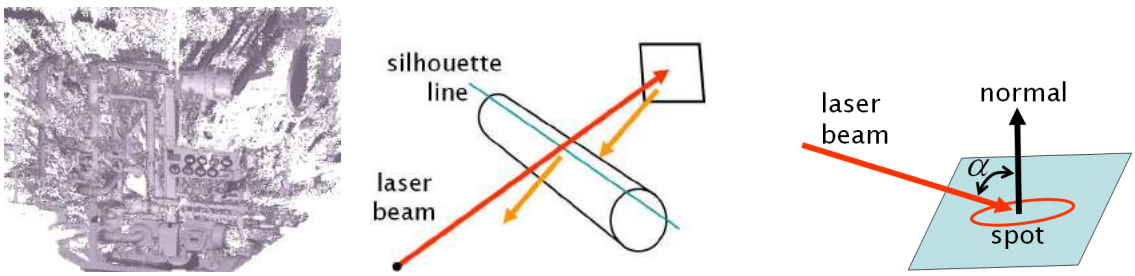
standardized parts are typically used in the field of plant construction, civil engineering, architecture, and transportation infrastructure. Therefore, we focus on extracting faces and edges for primitive surfaces.

Although many methods have been proposed for calculating primitive surfaces [4-6][8], it is still difficult to robustly detect edges and silhouette lines of surfaces. Fig. 1(a) shows an actual point-cloud, in which many noisy points and outliers are included. One of the key reasons of noises is the existence of mixels, which are caused by laser beams reflected from multiple surfaces. Fig. 1(b) shows mixels, which are caused at a silhouette line of a cylindrical object. Since wrong distance values are measured at mixels, they become outliers. Mixels are also caused at sharp edges. In addition, large noises and outliers are measured when laser beams are reflected from inclined surfaces. Fig. 1(c) shows that a laser spot is stretched on a surface when the angle α between the laser beam and the surface normal is almost 90 degree. Inclined surfaces are often caused near silhouette lines of curved surfaces such as cylinders. These problems make it difficult to robustly and precisely calculate edges and silhouettes of objects, because points captured from sharp edges and silhouettes are not reliable.

In this paper, we first propose a robust edge detection method. Since laser scanners tend to output wrong distances near sharp edges and silhouette lines, we generate a brightness image and a distance image using a point-cloud, and extract feature segments as candidates of edges and silhouettes. Feature segments are used to estimate candidate regions of primitive surfaces and to generate bounded faces.

Then we apply a region-growing method to candidate regions for detecting primitive surfaces. However, the region-growing method may fail to grow regions when adjacent points are very noisy. Although it is necessary to eliminate outliers and large noises, faithful smoothing methods, such as the moving least squares [1][3] and the moving robust estimate [5][6] are expensive for large-scale point-clouds. Therefore, we introduce a GPU-based robust estimate for obtaining smoothed points.

In the following Section, we will show an overview of our method. Section 3 presents our edge detection method. In Section 4, we will explain how to implement GPU-based smoothing based on robust estimate. Section 5 discusses surface extraction and Section 6 states conclusions.

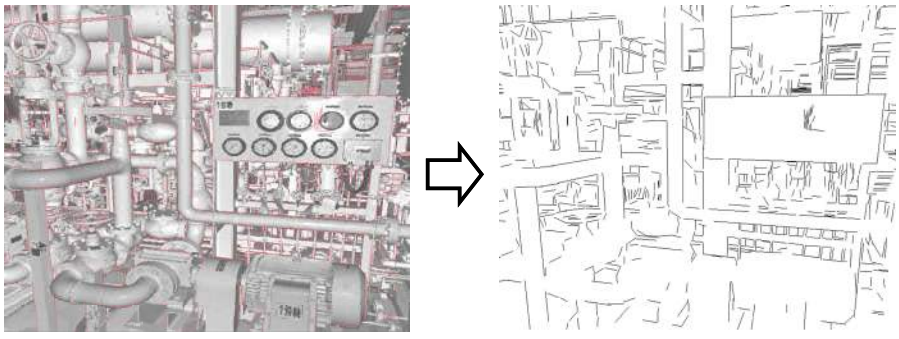


(a) Noisy point-cloud. (b) Mixels on a silhouette line. (c) Reflection from an inclined face.

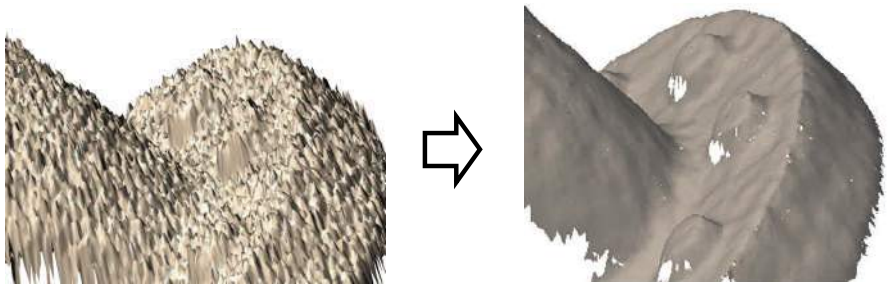
Fig. 1: Causes of outliers.

2 OVERVIEW

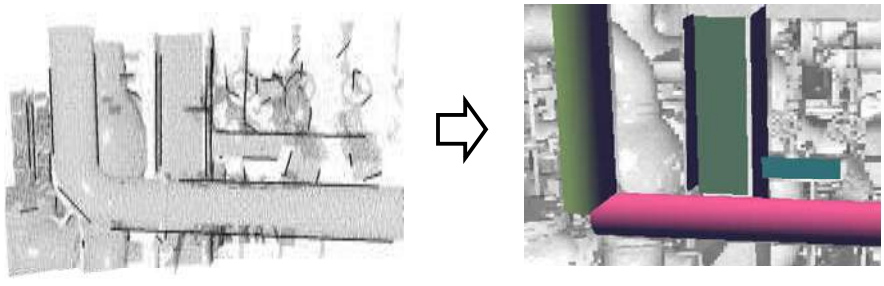
Fig. 2 shows an overview of our surface reconstruction method. First, we extract sharp edges and silhouette lines from raw scanned data. Time-of-flight and phase-based laser scanners output reflectance values at each direction of laser beams as well as coordinates. Therefore, we can generate a brightness image and a distance image using a point-cloud (Fig.2 (a)). While distance images are suitable for extracting feature points on silhouette lines, brightness images can be used to extract ones on sharp edges. We apply the Laplacian filter for distance images and Canny edge detection for brightness images. We combine feature points of the two images to detect continuous line segments (Fig.2 (b)). Line segments are candidates of sharp edges and silhouette lines. They are used to obtain candidate regions of primitive surfaces.



(a) Reflectance image and feature points. (b) Detection of feature segments.



(c) Noisy point-cloud. (d) Smoothing with GPU.



(e) Candidate regions of primitive surfaces. (f) Detection of primitive surfaces.

Fig. 2: A process overview.

Point-clouds captured by mid/long-range laser scanners are very noisy, as shown in Fig. 2(c). Although the region-growing method is useful to efficiently extract regions of primitive surfaces, it may fails to grow regions on very noisy data. In our previous work [5], we proposed a robust smoothing method based on moving robust estimates. This method is robust to outliers, but it is very time-consuming. To improve efficiency, we introduce a GPU-based robust estimate smoothing (Fig. 2(c, d)).

Primitive surfaces are detected using feature segments and a smoothed point-cloud. In our method, the region-growing method is only applied to regions between two feature segments (Fig. 2(e)). If a region successfully grows to a certain size, it is stored as a primitive surface. Primitive surfaces are converted to bounded faces using feature segments and grown regions (Fig. 2(f)).

3 ROBUST EDGE DETECTION METHOD

3.1 Generation of Reflectance Image and Distance Image

The direction of a laser beam is controlled by the azimuth angle θ and the zenith angle ϕ, as shown in Fig. 3(a). In principle, the coordinate of a point is calculated by the distance r and the direction (θ,ϕ). Coordinate (x,y,z) can be mapped on 2 dimensional space (θ,ϕ), as shown in Fig. 3(b). We call this rectangular image as a *Mercator image*. A Mercator image consists of 30M~50M pixels according to the number of points.

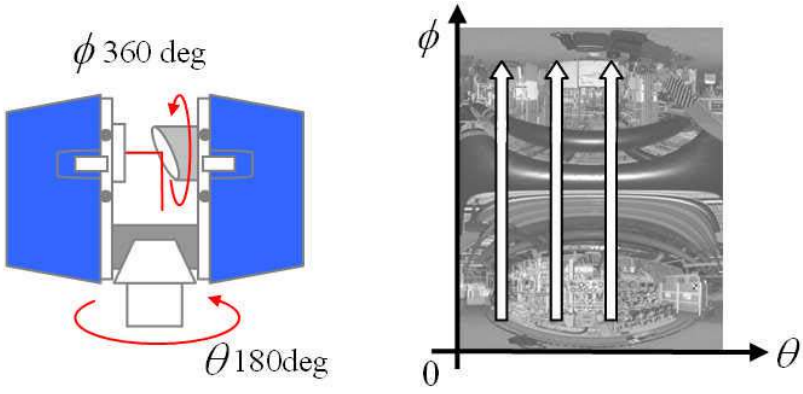


Fig. 3: Laser scanner and reflectance image: (a) Laser scanner, (b) 2D image by the azimuth and zenith angles.

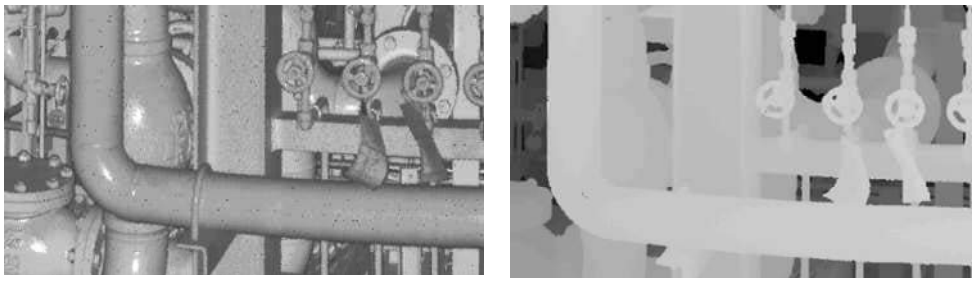


Fig. 4: Laser scanner and reflectance image: (a) Brightness image, (b) Distance image.

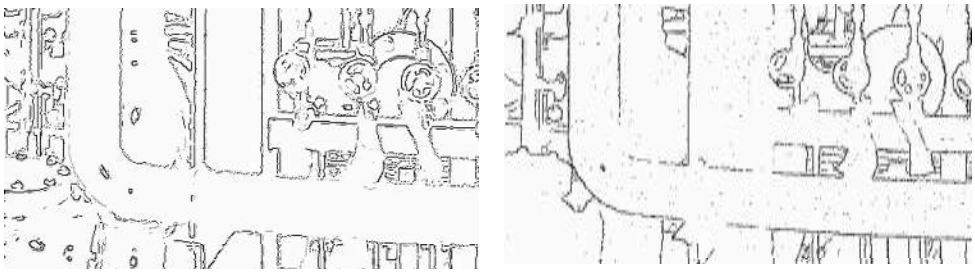


Fig. 5: Feature points: (a) Canny edge detection to brightness image, (b) Laplacian filtering to distance image.

When we set a reflectance value on each pixel of a Mercator image, we can generate a brightness image, as shown in Fig. 4(a). When we set a distance value instead, we can obtain a distance image. In Fig. 4(b),

nearer points in the distance image are displayed more brightly. When multiple scanned datasets are obtained and registered, their brightness image and distance image are maintained as well as transformation matrices.

3.2 Extraction of Feature Points

We extract feature points from distance and brightness images. Since distances change discontinuously on silhouette lines, feature points on silhouette lines can be extracted by applying differential filters, as shown in Fig. 6(b). Since brightness values discontinuously change on sharp edges, we also apply differential filters to detect internal sharp edges, as shown in Fig. 6(c).

Various types of filters have been proposed in the field of image processing [2]. We require filters that can robustly extract only feature points on sharp edges and silhouette lines. We investigated several image filters for distance images and brightness images.

For distance images, we compared results of the depth check, the Sobel filters, and the Laplacian filter. The depth check extracts feature points that have very different depth values at adjacent points. The Laplacian filter is defined by the mask of Fig. 7(a). The Sobel filters are defined by the two masks in Fig. 7(b), and the output is calculated as the root mean square of results of the two masks. In the both types of filters, feature points are extracted when the output values are more than certain thresholds. In our experiments, the Laplacian filter output the best feature points, because it tended to extract fewer points on inclined surfaces. In this paper, we use the Laplacian filter to extract silhouette lines.

For brightness images, we compared the Sobel filters, the Laplacian filter, and the Canny edge detection [2]. In our experiments, the Canny edge detection output the best feature points. In the implementation of this method, we used only 8 neighbors in hysteresis thresholding to efficiently process large-scale data. Fig. 5 shows feature points extracted from distance and brightness images.

3.3 Detection of Sharp Edges and Silhouette Lines

Feature points in a distance image and a reflectance image are merged and adjacent feature points are connected to form continuous lines. When the number of points of a continuous line exceeds a threshold, the line is registered as a *feature line*.

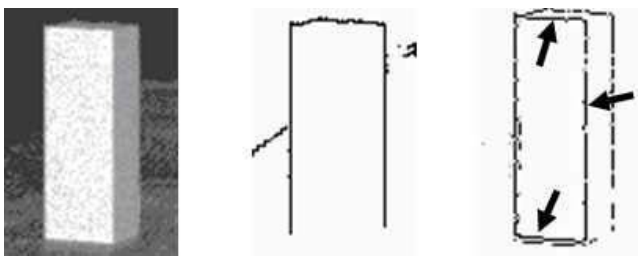


Fig. 6: Types of feature points: (a) Brightness image, (b) Silhouette lines, (c) Internal edges.

1	1	1
1	-8	1
1	1	1

-1	0	1
-2	0	2
-1	0	1

-1	-2	-1
0	0	0
1	2	1

Fig. 7: Filters for image: (a) Laplacian filter, (b) Sobel filters.

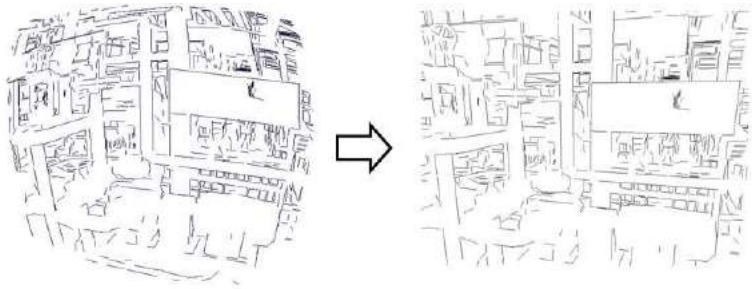


Fig. 8 Conversion from a Mercator image to a perspective image: (a) Feature lines on a Mercator image, (b) Feature segments on a perspective image.

However, straight lines are distorted on the Mercator image, as shown in Fig. 8(a), although the Mercator image is useful for representing all points in a single image. Since it is important to detect straight lines for extracting rectangular faces and ruled surfaces, we convert the Mercator image to perspective images, in which straight lines in 3D space are always mapped to straight 2D lines, as shown in Fig. 8(b). Fig. 9 shows the relationship between a Mercator image and a perspective image. Although (i, j) on the perspective image and (θ, ϕ) on the Mercator image can be mutually converted, a perspective image can cover a limited range of the Mercator image. Therefore, we cluster nearby feature lines and project them on the same perspective screen. The size of a screen is determined by the range of each group. When a feature line is too large to draw in a single screen, it is subdivided into appropriate sizes of lines.

For detecting straight lines on each perspective image, we first select seed points that fit to a straight line, and then extend the line by adding adjacent feature points. We call a detected line as a *feature segment*. Fig. 10 shows examples of feature segments.

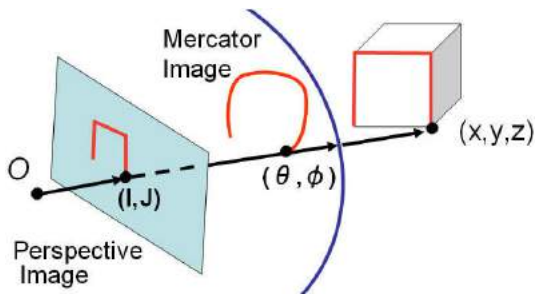


Fig. 9: Projection on perspective screens.

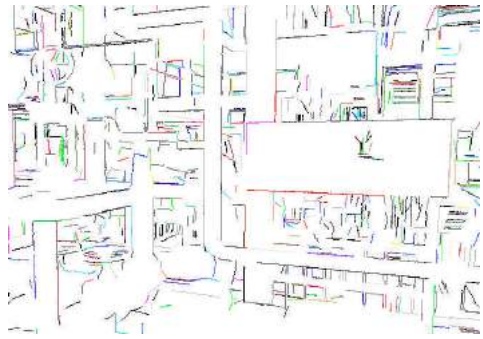


Fig. 10: Feature segments.

3.4 Detection of Pairs of Edges

Boundary edges of planar faces and silhouette lines of ruled surfaces appear as pairs of straight lines. We detect such pairs in feature segments. Suppose two lines $\mathbf{p}_1\mathbf{p}_2$ and $\mathbf{q}_1\mathbf{q}_2$. We project end points of each line to the other line, as shown in Fig. 11. If at least one of the projected points rides on the lines, the two lines are regarded as pair edges.

When pair edges are almost parallel, they can be a candidate of silhouette lines of a cylinder or boundary edges of a rectangle face (Fig. 12). Then the region between the pair edges is checked whether it fits to a plane or a cylinder. If pair edges are incorrectly selected, as shown in Fig. 12(b), the pair is discarded because no surfaces are fitted to the region. Elliptical lines are required to obtain candidate regions of other surface types, but the current implementation supports rectangle faces and cylinders that can be estimated by straight lines.

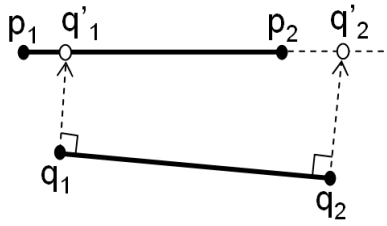


Fig. 11: Selection of a pair of lines.

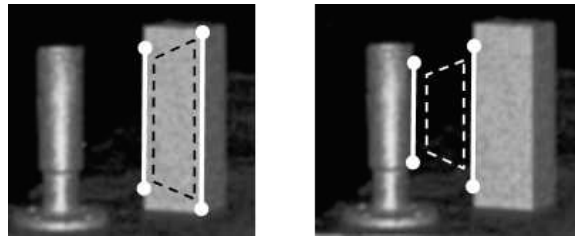


Fig. 12: Candidates of primitive surfaces: (a) Correct candidate region, (b) False candidate region.

4 SMOOTHING OF POINT-CLOUDS WITH GPU

4.1 Smoothing by Robust Estimate

When a point-cloud is very noisy, it is difficult to robustly calculate surface equations. Therefore, we smooth a point-cloud before calculating surface equations. In our previous work [5], we proposed a method for generating smoothed mesh models from very noisy and large-scale point clouds. In this work we introduced a robust smoothing operator based on the Lorentzian estimate. Since this operator requires nonlinear optimizations, it takes a lot of computation time when the operator is applied to large-scale point clouds.

In recent years, the performance of GPUs has significantly improved, and GPGPU (general-purpose computing on graphics processing units) has become common. In this section, we introduce a GPU-based smoothing method and accelerate the moving robust estimate for smoothing.

Our smoothing method is based on the robust estimate. In the first step of this method, referenced plane H is fitted to the neighborhoods of point p_i (Fig. 13(a)). Neighbor points are projected on the referenced plane and represented as a height field $\{(u_i, v_i, z_i)\}$. Then a quadratic surface $z = S(u,v)$ is fitted to the neighborhoods by calculating:

$$\min_{a} \sum_{j \in N_i} \log\left[1 + \frac{\left\{S\left(u_j, v_j\right) - z_j\right\}^2}{\sigma^2}\right] \theta\left(\left|c_i - q_j\right|\right), \tag{4.1}$$

where $\boldsymbol{a} = \{a_k\}(1 \le k \le M)$ denotes parameters of a quadric surface; M is the number of parameters; N_i is an index set of neighbor points; σ is the standard deviation; c_i is the referenced point; θ is a monotonically decreasing function. Finally, p_i is projected to the quadric surface and is moved to $p_i = \left(u_i, v_i, S\left(u_i, v_i\right)\right)$, as shown in Fig. 13(b).

Fig. 13: Smoothing by fitting: (a) Referenced plane H, (b) Quadric surface S.

4.2 GPU and CUDA

We implement this smoothing method on a GPU. Before we describe our method, we will briefly explain features of GPUs and CUDA.

In this paper, we use NVIDIA's GPU and CUDA [7]. A GPU has many processing units and can switch threads very quickly. Three types of management units are used in a GPU to control threads in parallelized calculations. Fig. 14 shows the three types of units. A thread is a minimum management unit and each thread has a unique index. A block is a set of threads and can manage them in a three-dimensional array. Threads in the same block can be synchronized. A grid has blocks in a two-dimensional array. Each grid can be called by a CPU as a kernel function.

Fig. 15 shows a simplified image of GPU architecture. A streaming processor (SP) is a minimum unit of processing and it controls calculations in a thread. A streaming multi processor (SM) has eight SPs and a shared memory, which can be accessed as fast as register and shared by SPs in the same SM. A block is controlled in each SM. Threads in the block can be synchronized and share data on the shared memory. The device memory has a very large capacity compared to shared memory and it can be accessed by all SPs in a GPU. However, the access to device memory is much slower than the access to shared memory and register. For improving efficiency, it is very important to allocate frequently-used data to shared memory or registers.

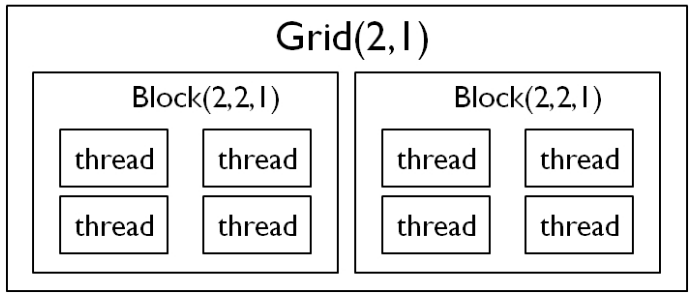


Fig. 14: Three types of units on GPU.

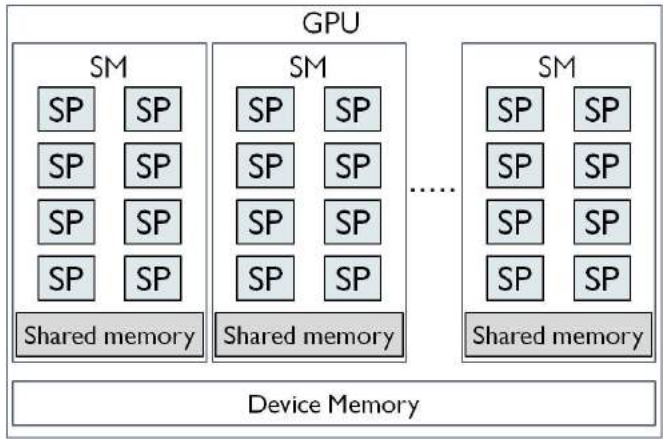


Fig 15: GPU architecture.

4.3 Allocation of GPU for Smoothing

Our GPU-based smoothing method is based on the robust estimate. Since points in a point-cloud are regularly aligned on the Mercator image as shown in Fig. 3(b), we can obtain neighbor points using adjacency relationships on the Mercator image.

This smoothing method can be independently applied to each point. Point p_i is smoothed using many neighbor points. In our experiments, 100~200 neighbor points were required to produce good

smoothing results. Fig. 16 shows how to calculate a smoothed point. In this figure, pixels represent points and gray pixels show neighbor points of p_i.

In our implementation, a block is assigned to the calculation of a smoothed point, and a thread is assigned to calculations related for each neighbor point. In each thread, the same calculation process is repeated in parallel. For reducing latency in calculation, it is important to store frequently-accessed data on shared memory. We allocate coordinate values of neighbor points to shared memory.

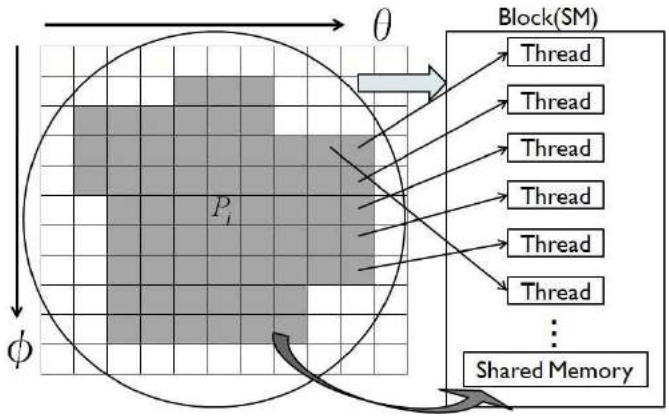


Fig.16: Smoothing of point p_i on GPU.

4.4 Smoothing with GPU

Our smoothing method consists of the following processes:

- retrieving neighboring points of p_i;
- eliminating invalid points and outliers;
- calculating referenced plane H and quadratic surface S;
- projecting p_i onto the quadratic surface S.

We implement these processes on a GPU.

Fig. 17 shows a flow of our smoothing method on a GPU.

In the step1, the number of blocks and threads are determined. As shown in Fig. 16, a block calculates a smoothed point, and a thread executes calculations for each neighbor point. We set the number of blocks as the one of points to be smoothed in a kernel function, and the number of threads as the one of neighbor points in each block.

In the step2, neighbor points are collected using adjacency relationships on a Mercator image, and we store the coordinates of neighbor points on shared memory. Coordinates are frequently accessed and shared by all threads in a block. Residual values for surface fitting are also allocated on shared memory.

In the step3, invalid points and outliers are eliminated from neighbor points, because undefined pixels and extremely far points are included in a Mercator image. When the number of neighbors is less than a threshold, the point is discarded as an outlier.

In the step4, data on shared memory are reordered, as shown in Fig. 18. In this figure, blue cells represent valid neighbors and red ones invalid. Since the combination of valid and invalid neighbor points makes many branches of instructions in threads, valid neighbors are moved to the top of shared memory for reducing braches of instructions.

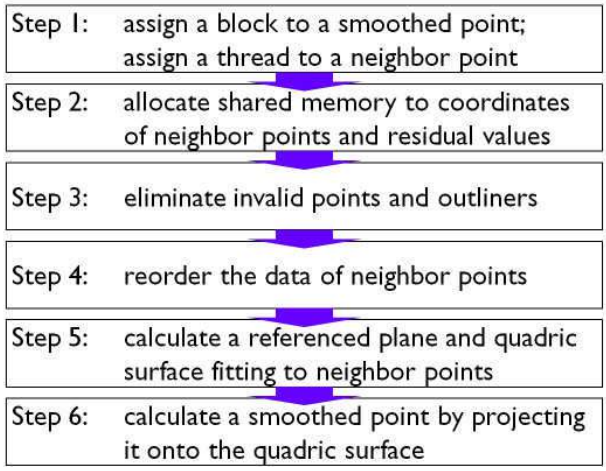


Fig. 17: Process flow of smoothing on a GPU.

Fig. 18: Reorder neighbor points in shared memory.

In the step5, a referenced plane and a quadric surface are calculated using neighbor points. They are calculated by the non-linear optimization [5]. In this process, liner equations $\boldsymbol{AX} = \boldsymbol{B}$ are frequently solved for obtaining initial values and updating solutions in the Gauss-Newton algorithm. Fig. 19 shows how to calculate elements in matrix $\boldsymbol{A}$ and $\boldsymbol{B}$. Each thread calculates a value related to a neighbor point, and then threads are used to calculate the sum of values. These calculations are synchronized using shared memory. Finally, thread1 solves $\boldsymbol{AX} = \boldsymbol{B}$, as shown in Fig. 20.

In the step6, p_i is projected on the quadric surface and the smoothed point of p_i is obtained. The result is transferred to a CPU.

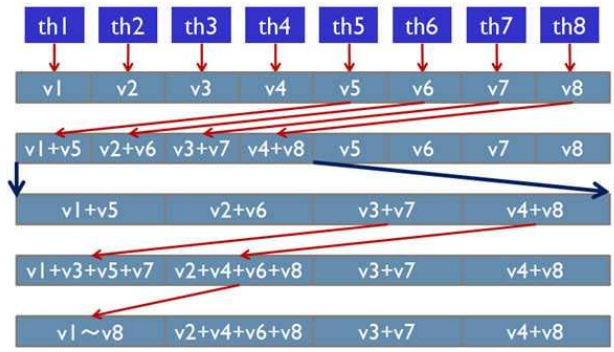


Fig. 19: Calculation of the sum of values on shared memory.

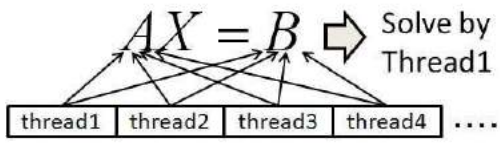


Fig. 20: Calculation of $\boldsymbol{AX} = \boldsymbol{B}$.

4.5 Experimental Results

Fig. 21 shows a noisy point-cloud. This data can be converted to 601×301 pixels on a Mercator image. Mesh models can be easily generated using the adjacency relationships on the Mercator image. Fig. 22 shows three mesh models generated using raw and smoothed points. Tab. 1 shows the execution time of smoothing on a CPU and a GPU. The result shows that our GPU-based method can significantly reduce the executive time of smoothing.

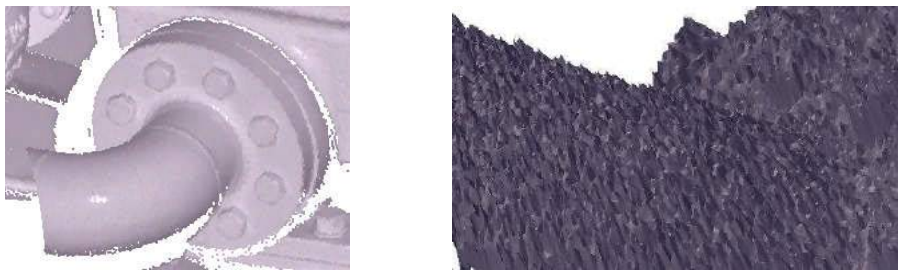


Fig. 21: Noisy point-cloud: (a) Brightness image, (b) Zoom-up of noisy mesh.

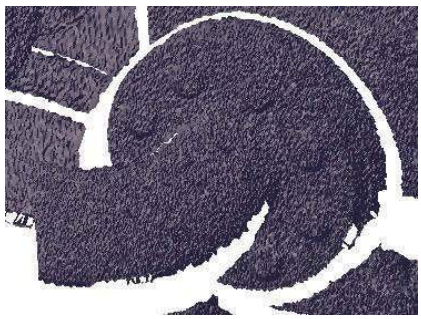
	CPU (Core2 2.66GHz)	GPU (GTX285)
Data1 (601×301)	54 sec.	3.8 sec.
Data2 (579×285)	52 sec.	3.2 sec.
Data3 (538×261)	44 sec.	2.7 sec.

Tab. 1: Timing of smoothing operations.

5 CALCULATION OF SURFACE PRIMITIVES

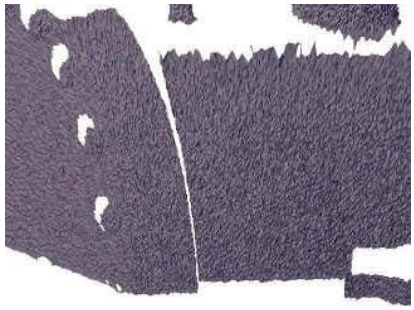
The extraction of primitive surfaces has been intensively studied so far. Lukacs, et al. [4] proposed non-linear fitting methods for primitive surfaces. For extremely large-scale point-clouds, we proposed the grid mesh format to apply the region-growing method in an out-of-core manner [6]. Schnabel, et al. [8] proposed a RANSAC-based approach for extracting primitive surfaces from noisy point-clouds. While RANSAC-based methods are more robust than the region-growing, they require a large amount of trials for calculating with a high degree of accuracy.

We use a grid mesh format [6] to manipulate a large-scale point-cloud. In this format, points in a point-cloud are projected on a Mercator image and they are subdivided into rectangle grids (Fig. 23). Each grid has coordinates and adjacency relationships of vertices. We maintain a point-cloud on a hard disk, and load only a necessary region in the main memory while processing.



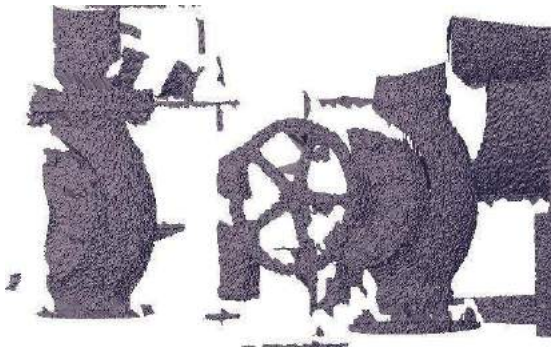
(a) Data1 Original mesh.

(b) Data1 Smoothed mesh.

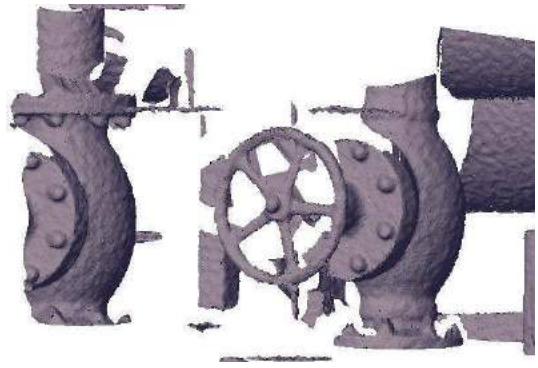


(c) Data2 Original mesh.

(d) Data2 Smoothed mesh.



(e) Data3 Original mesh.



(f) Data3 Smoothed mesh.

Fig. 22: Result of smoothing.

We calculate surface equations by using the region growing method. In the region growing method, a seed region is selected first, and then the surface region is detected by growing the initial seed region. We select seed regions using pair edges, as shown in Fig. 24(a). The surface type and initial values of a surface equation are estimated using the RANSAC method. Although the RANSAC method is computationally expensive, it is applied only to small candidate regions in our method. In addition, the number of trials can be significantly reduced in our method, because the result is only used as seed points of the region-growing method and does not have to be calculated in high precision.

When surfaces with the same equation are calculated from different candidate regions, the surface that has the largest number of points is selected. Fig. 24(c) shows an example of detected rectangles and cylinders.

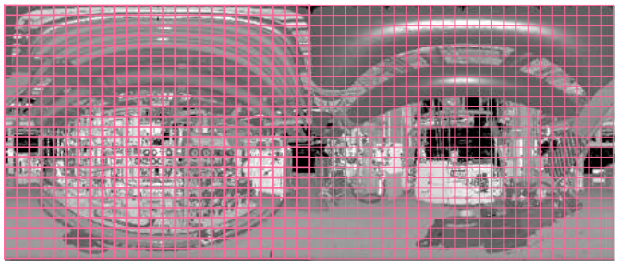
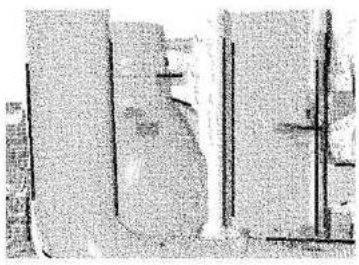
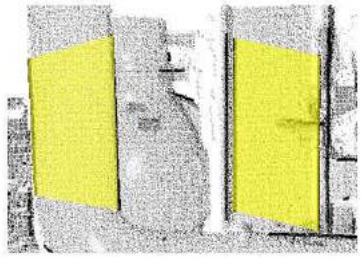


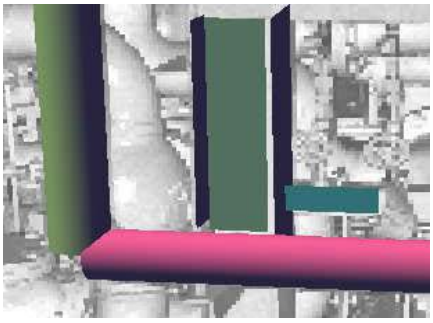
Fig. 23: Grid mesh format.



(a) Detected edge-pairs.



(b) Seed regions.



(c) Rectangles and cylinders.

Fig. 24: Primitive surfaces detected from edge-pairs.

6 CONCLUSION

In this paper, we proposed a method for extracting sharp edges and silhouette lines using a brightness image and a distance image, which were generated from a point-cloud. In our method, sharp edges and silhouette lines can be robustly detected even when their points are outliers. We showed how to estimate candidate regions of primitive surfaces by using feature segments. We also proposed a GPU-based smoothing method for efficiently detecting primitive surfaces. Experimental results of smoothing showed our GPU-based method was much faster than CPU-based calculation. Finally, we showed that primitive surfaces could be extracted using feature segments and smoothed points.

In the future works, we would like to extend our edge detection method, because our method is limited to straight lines in the current implementation. Other curved lines are required for silhouette lines of spheres and tori. We also would like to investigate image processing methods to detect feature points more robustly.

ACKNOLEDGEMENTS

This work was supported by Grant-in-Aid for Scientific Research (No. 21360069). Example point-clouds are courtesy of Shinsei-Giken and Leica Geosystems.

REFERENCES

[1] Alexa, M.; Behr, J.; Cohen-Or, D.; Fleishman, S.; Levin, D.; Silva, C.T.: Point set surfaces. IEEE Visualization, 2001, 21–28.
[2] Forsyth, D. A.; Ponce, J.: Computer Vision: A Modern Approach, (2002), Prentice Hall.
[3] Levin, D.: Mesh-independent surface interpolation. Geometric Modeling for Scientific Visualization, 2003, 37–49.
[4] Lukacs, G.; Marshall, A.D.; Martin, R.R.: Faithful least-squares fitting of spheres, cylinders, cones and tori for reliable segmentation. Proceedings, 5th European Conference on Computer Vision, 1998, 671–686.
[5] Masuda, H.; Tanaka, I.: Extraction of Surface Primitives from Noisy Large-Scale Point-Clouds, Computer-Aided Design & Applications, 6(3), 2009, 47-57. DOI: 10.3722/cadaps.2009.387-398
[6] Masuda, H.; Tanaka, I.: As-Built 3D Modeling of Large Facilities Based on Interactive Feature Editing, Computer-Aided Design & Applications, 7(3), 2010, 349-360. DOI: 10.3722/cadaps.2010.349-360
[7] NVIDIA CUDA Programming Guide Ver. 2.3, (2009), NVIDIA.
[8] Schnabel, R.; Degener, P.; Klein, R.: Completion and Reconstruction with Primitive Shapes, Computer Graphics Forum, 28(2), 2009, 503-512. DOI: 10.1111/j.1467-8659.2009.01389.x