大規模点群データの平滑化手法に関する研究（第 3 報）*

－GPU を用いた平滑化処理の高速化－

池 田　邦 彦***　　増 田　宏**

A Study on Smooth Surface Reconstruction from Large-Scale Noisy Point-Clouds (3rd report)
- GPU based Smoothing method for Large-Scale Point-Clouds -

Kunihiko IKEDA and Hiroshi MASUDA

The recent progress of mid/ long-range laser scanners enables to capture large-scale point-clouds in short time. However, point-clouds typically contain many large noises and outliers, and therefore, reliable smoothing methods are very important. Although the moving least-squares or the moving robust estimate is powerful smoothing tools, they are very time-consuming for large-scale point-clouds. In this paper, we propose a GPU-based method to accelerate the moving robust estimate. We implemented it on a GPU and evaluated its performance. The result shows that our GPU-based method is much faster than CPU-based smoothing, and it can be applied to large-scale point-clouds.

Key words: geometric modeling, point-cloud, smoothing, streaming, reverse engineering, GPGPU

1. 緒　　言

近年，中・長距離を計測できるレーザスキャナが急速に進歩し，工場などの大規模設備を短時間かつ高密度で計測できるようになってきた．特に位相差方式のレーザスキャナを用いれば，数千万から数億個の座標を短時間で取得することができる．そうした高密度の点群を用いて現実に則した 3D モデルを作成し，シミュレーションを行うことができれば，老朽化した生産設備やプラント施設の改修・メンテナンス作業での手戻りを減少させることができるので，作業時間やコストの削減が期待できる．

しかしながら，中・長距離レーザスキャナで計測した場合，計測点のノイズが大きく，大量の異常値が含まれることが多い．そのため，点群を用いたモデリングにおいては，その前処理として，ノイズや異常値を除去することが必要である．本研究では，そのための処理を平滑化と呼ぶ．

我々は，これまでの研究において，異常値があっても安定的に平滑化が行える移動ロバスト推定法を提案した[1]．また，メモリ容量を超える大規模点群を平滑化するために，データを逐次的にハードディスクから読み込み，ストリーミング形式で点群を平滑化する手法を示した[2]．

しかし，これらの平滑化で用いられている移動最小２乗法や移動ロバスト推定では，各点の平滑化に非線形最適化が必要となるため，非常に多くの計算時間がかかるという問題点がある．

平滑化計算の高速化のためには，GPU を用いた並列計算が有効である[3]．本研究では，メモリ容量を超える大規模点群にも適用できるようにするために，GPU による平滑化をストリーミング形式で行える手法を提案する．また，近傍点を距離に応じて適応的に制御することで品質と計算効率を向上できることを示す．

以下，第 2 章では移動ロバスト推定による点群の平滑化と GPU の構成について説明する．第 3 章では GPU による平滑化手法を示し，第 4 章ではストリーミング処理について示す．第 5 章で評価実験の結果を示し，第 6 章で結論を述べる．

2. 点群の平滑化

2.1 移動ロバスト推定による平滑化

中・長距離レーザスキャナで計測された点群データにノイズレベルが大きく，また多くの異常値が存在する．図 1 はプラントを計測した点群の一部を拡大したもので，ノイズレベルが大きいことがわかる．

点群の平滑化手法としては，移動最小 2 乗法を用いた手法が広く知られている[4]．最小２乗法は異常値に脆弱であるため，前処理で異常値を除去する必要がある．しかし，本研究で扱うような点群には大量の異常値が含まれるため，すべてを除去することは難しい．そのような場合には，移動ロバスト推定が有効である．ここでは，第 1 報で提案した移動ロバスト推定による平滑化[1]について説明する．平滑化は以下に示す手順で行われる．

1. 平滑化したい点を p_i とする．p_i に関して，近傍点 $\{\mathrm{p}_j\}$ $(j = 1,..N_i)$ を得る．
2. 近傍点を平面で近似する．この平面を参照平面 H とする（図 2(a)）．
3. すべての近傍点を参照平面 H 上に投影し，各点を平面上の座標系を用いて記述する．また，点 p_i を平面 H 上に投影した点を基準点 p_c とする．
4. 近傍点 $\{\mathrm{p}_i\}$ を平面 H 上のハイトフィールド (u,v,z) と見做し，2 次多項式曲面 $z = S(u,v)$ で近似する．
5. p_i を曲面 S に投影し，その点を平滑化された座標とする．(図 2 (b))

ここで，2 次多項式曲面 $z = S(u,v)$ は，以下の式(1) を解いて計算する．σ は標準偏差，また θ は距離によって減衰する関数で，関数 θ にはガウス関数を用いることが多い．

$$\min \sum_{1\le j\le N_i} \log\left[1+\frac{\left\{z_j - S\left(u_j,v_j\right)\right\}^2}{\sigma^2}\right]\theta\left(\left|p_c - q_j\right|\right) \quad (1)$$

---

*　原稿受付 平成 21 年 4 月 30 日
**　正 会 員　東京大学大学院（東京都文京区本郷 7-3-1）
***　学生会員　東京大学大学院（同上）

この式では，値が大きくはずれた異常値が存在しても，その値は相対的に重要視されず，解に与える影響が小さいことが知られている[1]．したがって，本研究で対象とする中・長距離スキャナのための点群処理に適している．

2.2 GPU による並列計算

平滑化処理では，各点の近傍点が取得できていれば，並列計算が可能である．本研究では，移動ロバスト推定による平滑化を GPU を用いて実現することを考える．

GPU は本来描画のためのハードウェアであるが，通常の計算においても高い並列性による高速な演算が可能である．本研究では NVIDIA 社の GPU を用い，開発のための言語は CUDA を用いて平滑化処理を実現する．ここでは，まず GPU の基本的な構成について説明しておく．

GPU はその内部に多くの演算ユニットを持ち，大量のスレッドを瞬時に切り替えることで高い並列性を実現し，高速な演算処理を可能としている．CUDA プログラミングでは，図 3 に示すように，スレッドを以下の 3 つのレベルで管理する．

(1) スレッド：計算の最小単位で，各スレッドは固有の ID 番号を持つ．
(2) ブロック：スレッドの集合を管理する．集合は 3 次元配列で記述できる．同一ブロック内のスレッド同士は同期を取ることができる．
(3) グリッド：ブロックの集合を管理する．集合は 2 次元配列で記述できる．異なるブロックのスレッドは同期が取ることができない．

図 4 は，GPU のハードウェア構成を示したものである．GPU において，ストリーミングプロセッサ (SP) は演算ユニットの最小単位であり，ストリーミングマルチプロセッサ (SM) は SP 8 個をグループ化したものである．GPU 内には，SM が多数存在する．SM 内部にはレジスタ並みに高速にアクセスが可能なシェアードメモリが存在し，SM 内の SP で共有される．

CUDA プログラミングでは，ブロックは SM に割り当てられ，ブロック内のスレッドは SP に割り当てられる．同じブロック内のスレッドはシェアードメモリを通じてデータを共有することができる．

高速化で重要となるのがメモリの管理である．GPU はすべての SP からアクセル可能なデバイスメモリを持つ．デバイスメモリは大容量であるが，シェアードメモリやレジスタに比べてアクセス速度が著しく遅い．一方で，シェアードメモリやレジスタは高速であるが容量が小さく，大きなデータは保持できない．そのため，レジスタやシェアードメモリをどう用いるかが計算の高速化において非常に重要となる．

3. GPU による点群の平滑化

3.1 GPU による平滑化

本研究では，移動ロバスト推定による平滑化を GPU を用いて実装する．そのためには，近傍点を得る処理と，参照平面 H と 2 次曲面 S を求めるために式(1)の非線形の最適化問題を解く処理が必要となる．また，大規模点群においては，すべてのデータをメモリ上に保持するには大容量のメモリが必要となるため，省メモリで処理できるストリーミング処理が必要である．本章では GPU による平滑化処理について示す．

本研究においては，図 5 に示すように，一つの点の平滑化処理をブロックに割り当て，平滑化で用いる個々の近傍点に対する計算をスレッドに割り当てる．また，平滑化の処理にあたり，

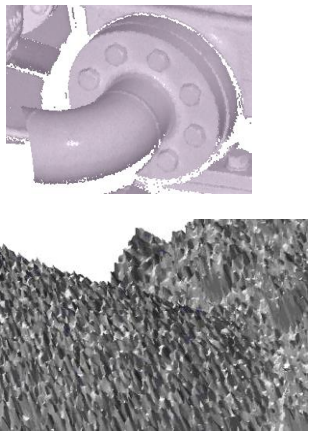
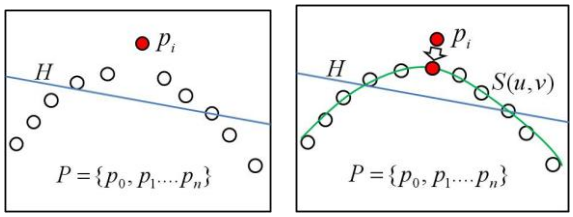


Fig. 1 Noisy point data



(a) reference plane: H (b) quadric surface: S

Fig. 2 Smoothing by fitting to a quadratic surface

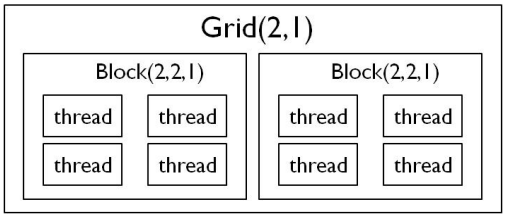


Fig.3 Three levels for GPU programming

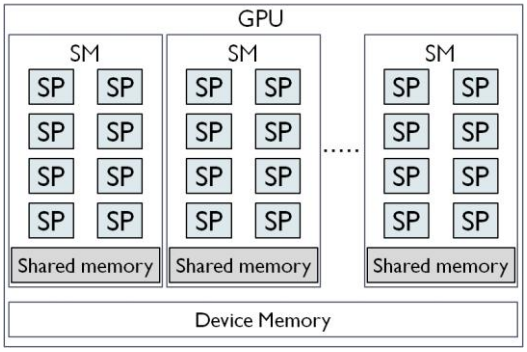


Fig.4 GPU architecture

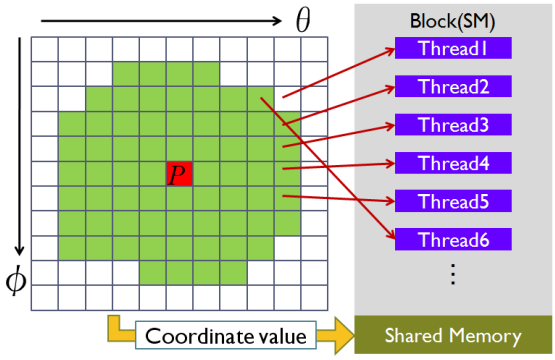


Fig.5 Smoothing of point p on GPU

近傍点の座標値は頻繁にアクセスするので，シェアードメモリで保持する．

本研究では，これらの座標を用いた処理を GPU 上で計算するとともに，ストリーミング形式で，ハードディスク，CPU，GPU を連携させて処理する．そのためには，近傍の点の座標を効率的に取得できる仕組みが必要である．まず，以下でそのための手法を示す．

3.2　ストリーミング処理による近傍点取得

平滑化処理は，近傍の点を用いて計算される．近傍の取得には，オクトツリーや kd ツリーなどを用いることが多い．しかし，k-近傍を得るには，近傍点を距離が近い順にソートすることが必要であるが，ソート処理を GPU で高速化することは難しい．

そこで，本研究では，PTX フォーマットで記述された点群データに基づいて，点群処理を行うことを考える．PTX フォーマットは点群を記述するための標準的なフォーマットの一つであり，点群を 2 次元配列によって記述した形式である．中・長距離のレーザスキャナでは，図 6 に示すように，レーザビームを垂直方向に 360 度回転させながら，水平方向に 180 度回転させることで全方位の点群を取得する．垂直方法に一周するごとに h 個のデータを取得し，垂直方向に w 周回転したとすると，点群は w×h の 2 次元配列として得ることができる．ここでは，この 2 次元配列を点群格子と呼ぶ．

PTX フォーマットでは，点群格子を以下のフォーマットで記述する．

(1)　2 次元配列の大きさ(w と h)
(2)　変換行列（レジストレーションデータ）
(3)　w×h 個の座標

座標は計測装置を原点とする座標系で記述され，計測不能な欠落した点は，座標 (0,0,0) で表す．また，複数の計測データをレジストレーションによって位置合わせしたときは，座標を直接書き換えるのではなく，ヘッダの変換行列を書き換える．

このフォーマットで記述された点群において近傍点を取得するためには，必要な個数の列のみをメモリ上に保持しておけばよいので，ストリーミング形式での処理が可能である．

ここで，平滑化したい点を p_0 とする．近傍処理においては，図 5 に示す点群格子上で，p_0 の近傍点の候補を $\{p_i \mid |I_i - I_0| \le D_0\}$ を満たす点として取得する．ここで，$I_i = (i_i, j_i)$は，点 $p_i = (x_i, y_i, z_i)$ における点群格子の座標である．D_0 は点群格子上での距離の閾値であり，この決め方については口述する．

ただし，点群格子上で近傍にある点が，3 次元空間でも近傍になるとは限らない．そこで，p_0 の近傍候補点 $\{p_i\}$ に関して，同一面上に乗っていると見做せるかどうかの判定を行う．ここでは，点群格子上での p_i の 4 隣接点を用いて，既に同一面上に乗っていると判定されている隣接点との空間距離が L_0 より大きい点を除外する．この判定は，すべての近傍点で同じ L_0 値を用い，p_0 からの点群格子上での距離が近い順に行う．また，計測できなかった無効な点の座標は (0, 0, 0) となっているので，その点は除外する．

ここで，閾値 L_0 は，計測の際のピッチ角 Δφ から計算することができる．図 7 に示すように，点群格子上で隣り合う点の位

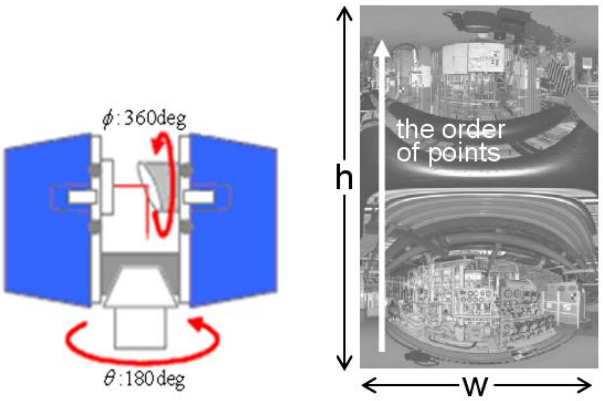


Fig.6　Measured data by a laser scanner

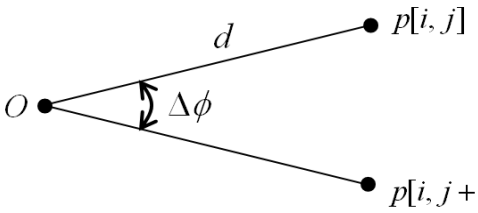


Fig. 7　Estimated distance between neighbor points.

(a)　Distance: 2m

(b) Distance: 9m

Fig.8　Smoothing by sizes 9, 13, and 17.

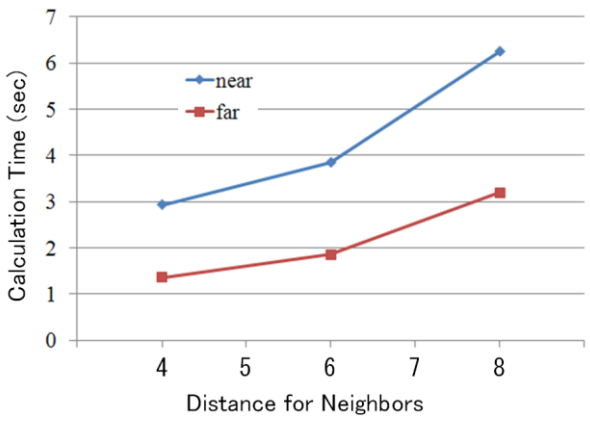


Fig.9　Calculation time for different sizes of neighbor points

置ベクトルが成す角度を $\Delta\phi$ ラジアンとする．このとき，隣り合う点が同一面上に乗っており，この面の法線とレーザ光との角度が φ であると仮定すると，$\Delta\phi$ は微小なので，二点間の距離は $d(\Delta\phi)/\cos\varphi$ と近似できる．ここで，d は p_0 と原点との距離である．ここでは，$\varphi \leq 70°$ として，距離の閾値 L_0 を $3dD(\Delta\phi)$ とする．

次に，近傍点取得を行うための点群格子距離 D_0 について考える．レーザスキャナは，同じ角度ピッチで座標を計測するので，点群格子上の隣接点の空間距離は，計測原点からの距離に比例して大きくなる．したがって，遠方の面が過度に平滑化されるのを防ぐために，遠方の点においては，平滑化に用いる点群格子の範囲を小さくする必要がある．

図 8 は計測原点から近い点群と遠い点群に関して，近傍点を得るための点群格子距離 D_0 を変化させて平滑化を行った結果である．この例では，D_0 の値を 4, 6, 8 の 3 通りで比較した．この図より，近い点群では不十分な近傍数であっても，遠距離の点群では十分に平滑化が行われていることが分かる．また，平滑化の実行時間を図 9 に示す．点の個数は D_0 の 2 乗に比例するので，D_0 を小さくするほど計算時間は短くなる．

ただし，D_0 の値を距離に応じてどのように変えるかは，計測装置の性能にも依存する．一般に，計測原点からの距離が大きくなるほど，誤差レベルも増大するからである．本研究では，Z+F 5003 を用いて計測した点群を用いて実験を行った結果，以下のように D_0 を決めることで良好な結果が得られることを確認した．

$$D_0 = \begin{cases} M & (d \leq d_s) \\ \max[\mathrm{int}(D_{\max}\sqrt{d_{\min}/d}), D_{\min}] & (d > d_s) \end{cases} \quad (2)$$

ここで，$\mathrm{int}(x)$ は x に最も近い整数を返す関数，d は p_0 の計測原点からの距離，d_s は基準距離，$D_{\max}$ は基準距離での点群格子距離，$D_{\min}$ は平滑化計算で必要となる最小の点群格子距離である．これらのパラメータは計測装置の精度に依存する量である．本研究の例題で示す点群では，図 8 の結果などに基づいて，$d_s = 2[m]$，$M = 8$，$N = 3$ と設定する．

3.3 GPU による平滑化

GPU の平滑化において，CPU から必要なデータを転送後，点群を平滑化するまでの流れを順に説明する．

3.3.1 シェアードメモリへのデータ転送

CPU から転送されてきた点群の座標データは，データ量が多いため，デバイスメモリに保持されている．平滑化計算においては，近傍点の座標に頻繁にアクセスするため，処理速度向上のために，各ブロックが担当する平滑化に必要な近傍点のデータをシェアードメモリにコピーする．

3.3.2 近傍点の絞り込み

次に 3.2 で述べた手法を用いて，シェアードメモリ上の近傍点から実際に平滑化に使用する近傍点を選択する．こうして近傍点が絞られた結果，平滑化に使用する近傍点としない近傍点のデータがシェアードメモリ上に混在する．GPU では，条件分岐により処理フローが分かれると処理速度が落ちるので，スレッド間の処理分岐を避けるために使用する近傍点のデータをシェアードメモリの先頭に寄せておく．また，この作業により近傍点の数が著しく減った点に関しては，その点を異常値であると見做して処理を終了する．

3.3.3 参照平面と 2 次曲面の計算

平滑化で用いる重み関数や座標変換などの計算は，各近傍点を担当するスレッドが並列して計算する．参照平面や二次曲面の算出では非線形最適化の計算を行うので，初期解の算出と解の更新が必要である．本研究では，ガウスニュートン法を用いて計算を行う．この計算では，線形方程式 $Ax = b$ を頻繁に解く必要がある．GPGPU による線形方程式の計算はよく知られた処理なので，ここでは A, b の計算法について説明する．

A, b の要素は暫定解と近傍点座標を元に算出されるが，GPU での計算過程は二通りに分けられる．一つは，ブロック固有の値から求められる要素で，並列計算する必要がないのでスレッド 1 で計算し，シェアードメモリに保存する．暫定解とその残差などを元に計算される値がこれに相当する．もう一つは近傍点ごとに座標値を用いて計算し，その値の総和により求められる要素である．この場合，各スレッドで近傍点に関する計算を並列で行い，その計算結果をシェアードメモリ上でスレッドを同期させながら総和を求める．

A,b が算出されたら，$Ax = b$ をスレッド 1 で解いて，参照平面と 2 次曲面を計算する．最終的に，すべての平滑化計算が完了したら CPU へ結果を転送する．

4. ストリーミング処理による大規模点群の平滑化

位相差方式で計測された点群データは非常に大容量である．GPU で大規模点群を処理する場合，カーネルの実行時間やメモリサイズの制約が問題になってくる．そこで，平滑化処理をストリーミング処理にすることで大規模点群に対応する．本研究では，PTX フォーマットで記述された点群座標を逐次読み込みながら，平滑化を行う．

ストリーミング処理の流れを図 10 に示す．この処理は以下の手順で行われる．以下の手順を繰り返すことで，大規模な点群を平滑化する際も，一度に扱う点群の量は少なくて済み，メモリ容量の制約された GPU でも安定した実行が可能となる．

(1) 逐次的な入力データの読み込み

ハードディスク上のデータを逐次的に読み込んでいく．その際，入力データは列ごとにデータが格納されているので，列単位でデータを読み込む．メモリに保持する列数は，一度のカーネルで実行する矩形に対して近傍点が取得できる範囲を最低限含むものとする．

(2) ブロック数とスレッド数

GPU で平滑化を行うカーネルにおいて，ブロック数は一度のカーネルで平滑化する点の数に対応し，スレッド数は 1 点の平滑化に使用される近傍点の個数に対応する．

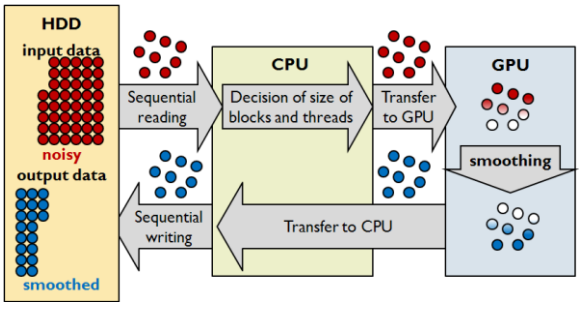


Fig. 10 streaming process for smoothing of large-scale point-cloud

ブロック数に関しては，メモリの制約やカーネルの実行時間の制約を満たすように設定する．特に，ディスプレイの表示も担っている GPU を GPGPU に用いる場合, OS に依存してカーネルの実行時間が制限されており，その時間を超えた処理は認められない．本研究では，Windows OS 上で処理を行っているので，カーネルの一回の処理がこの制限時間内に終わるようにブロック数を決定している．

(3) カーネル関数の実行

一回のカーネルの実行に必要な点群データを GPU に転送し，カーネル関数を CPU から呼び出す．GPU では， 3.3 に示した処理を実行する．GPU で平滑化が終了したら，その結果を CPU に転送する．

(4) 逐次的な出力データを書き込み

平滑化が終了した点群データから順に，逐次ハードディスクに書き込んでいく．

5. 評 価 実 験

本論文で提案した方法を用いて，点群の平滑化の評価実験を行った．本研究で用いた計測装置は，Z+F 社の Imager5003 である．この装置は計測距離が 1.0～53.5m で水平方向に 360 度，垂直方向に 310 度計測できる．計測は，図 6 において垂直方向に 360 度回転する毎に 10000 点計測するモードで計測を行った. 取得される点は約 5000 万点である．例題には，実際のプラント設備を計測したデータを用いた．

計算機環境は，CPU が Intel Corei7 (2.93GHz) で，GPU は NVIDIA GeForce GTX285 である． この GPU は 240 コアで，プロセッササイクルは 1.5GHz，デバイスメモリは 1GB である．

本手法の計算速度を評価するために，CPU のみを用いた計算と比較した．CPU での計算はシングルコアで行った．また，どちらの計算もストリーミング形式で行い，ハードディスクから逐次データを読み込みながら計算を行った．ストリーミング処理では，ファイルを読み込む時間も計算時間に含まれる．PTX フォーマットを ASCII 形式で読み込むとデータ読み込み時間が長くなるため，評価実験では同じ形式のバイナリファイルに変換し，データ読み込みのオーバーヘッドを少なくして計算時間を比較した．

計算時間は，表 1 に示すとおりである．GPU を用いた平滑化処理では，CPU のみを用いた場合と比較して，計算効率が約 9 倍に向上した．また，平滑化した点群から生成されたメッシュを図 11 と図 12 に示す．これらの結果より，本手法により CPU による計算と同等の十分良好な平滑化ができていることが検証できた．

Table. 1 Timing of smoothing operations

	CPU (Corei7 2.93GHz)	GPU (GTX285)
実行時間	115.4 min.	12.5 min.

6. 結　　言

本研究では，メモリ容量を超える大規模点群の平滑化を GPU を用いて高速に行う手法を示した．また評価実験の結果，CPU のみでの平滑化に比べ大きく実行時間を短縮するができることを示した．また，GPU のカーネルの実行時間の制限やメモリ容量の制限などがある中，平滑化の処理をストリーミング処理にすることで大規模な点群に対しても有効であることを示した．

今後の課題としては，誤差レベルの異なる複数の計測装置での検証が挙げられる．計測装置は現在でも年々進歩しており，速度や精度が向上している．今後，様々な計測装置での検証を通じて，平滑化で用いるパラメータの設定などについて検討していきたいと考えている．

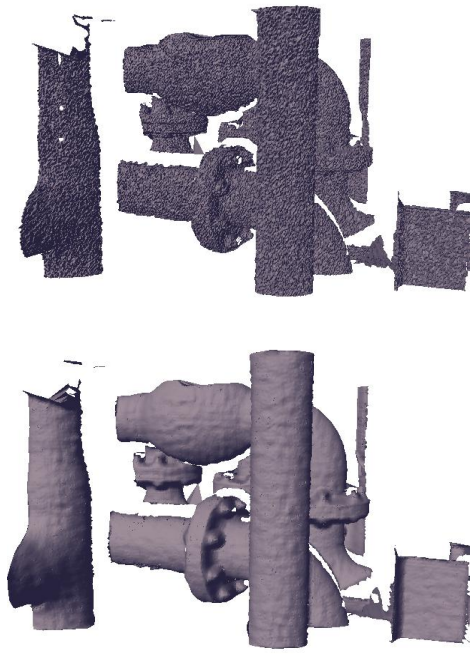


Fig.11 Smoothing result (original and smoothed meshes)

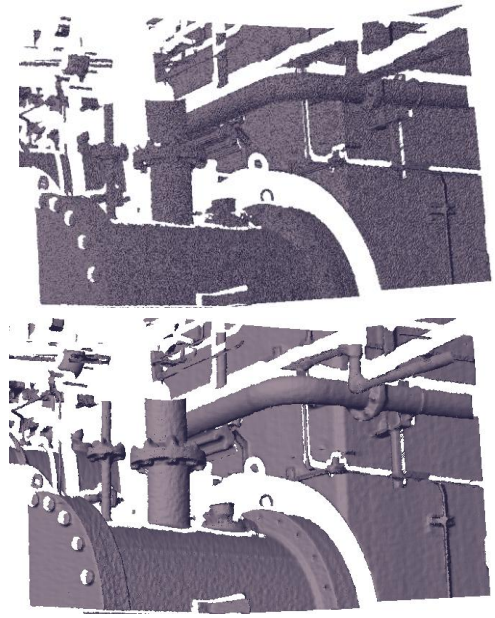


Fig.12 Smoothing result (original and smoothed meshes)

参 考 文 献

1) 増田宏，村上健治：大規模点群データの平滑化手法に関する研究（第 1 報）－ロバスト推定に基づく平滑化手法－，精密工学会誌，76(5), pp.582-586, 2010.
2) 増田宏，村上健治：大規模点群データの平滑化手法に関する研究（第 2 報）－大規模点群平滑化のためのストリーミング処理－，精密工学会論文誌, 76(6), pp.689-693, 2010.
3) K. Ikeda, C. Matsunuma, H. Masuda : Robust Edge Detection and GPU-Based Smoothing for Extracting Surface Primitives from Range Images, Computer-Aided Design and Applications, 8(4), pp. 603-616, 2011.
4) D. Levin.: Mesh-Independent Surface Interpolation, Geometric Modeling for Scientific Visualization (2003) 37