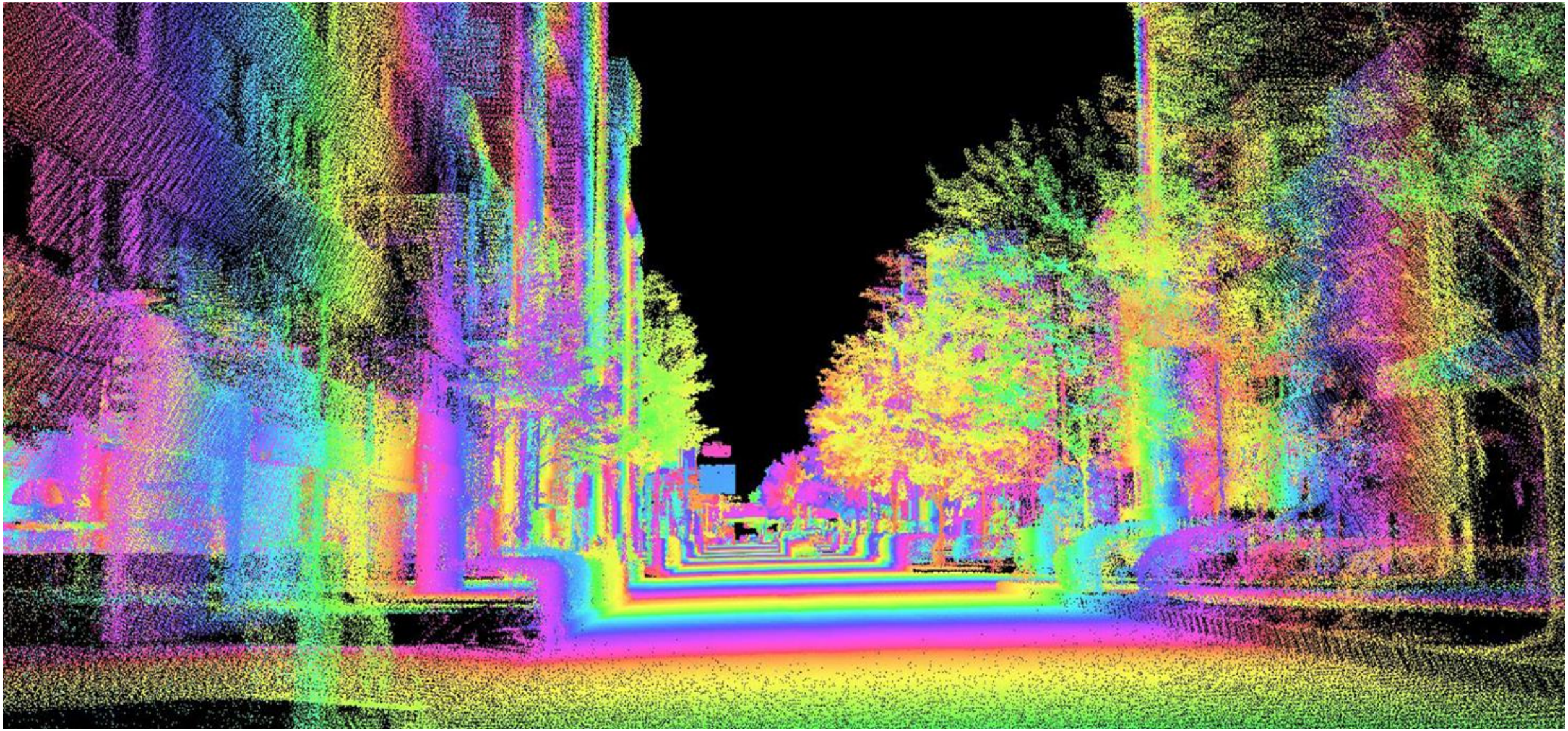
A.I.を用いた3D都市モデルの自動更新手法 技術検証レポート

Technical Report for Methodology of Automatic Updating of 3D City Models using A.I.

PLATEAU by MLIT



目次

I. 実証概要
1. 全体概要 4
2. 実施体制 6
3. 実証エリア 7
4. スケジュール 8

II. 実証技術の概要
1. 活用技術 １０
2. City Generator １１
3. Blender １２
4. CloudCompare １３
5. Pytorch １４
6. Open3D １５
7. 簡易MMS １６

III. 技術調査
1. 調査内容及び方針・課題 １８
2. 自動モデリングツール開発のための手法 １９
3. 実証システムでの技術選定 ２５
4. 点群データの取得 ２６
5. 参考論文 ３０

IV. 実証システム
1. システムアーキテクチャ図 ３４
2. データアーキテクチャ図 ３５
3. システム機能 ３６
4. データ ３７
5. トレーニングデータ作成 ４１
6. ノイズ除去 ４７
7. 建物抽出 ５５
8. 点群合成 ６２
9. 更新箇所特定 ７２
10. メッシュ再構築 ７８
11. 窓検出 ９４
12. アルゴリズム １０３ １０３

V. 実証システムの検証
1. 実証フロー １１７
2. 3次元点群データ取得及び作成 １１８
3. トレーニングデータの作成 １２５
4. ノイズ除去 １２６
5. 更新箇所特定 １２９
6. LOD3建築物モデリング １３９

VI. 成果と課題
1. 今年度の実証で得られた成果 １７０
2. 今後の取り組みに向けた課題 １７４

用語集 １７５

Ⅰ．実証概要

Ⅱ．実証技術の概要

Ⅲ．技術調査

Ⅳ．実証システム

Ⅴ．実証システムの検証

Ⅵ．成果と課題

全体概要（1/2）

本実証の全体概要は下表のとおり。

業務名	デジタルツイン構築に向けた3D都市モデルの更新に関する調査研究業務
実施場所	宮城県仙台市
目標・課題 ・創出価値	• 3D 都市モデルの更なる活用を進めるためには、日々変化する都市空間の3次元的な情報をリアルタイムに取得し3D都市モデルに反映することが求められる。 • 従来、3D都市モデルの更新は航空測量データを用いて都市全体を一気に更新する方法が標準的であった。この方法は、広い範囲を面的に更新できる利点がある反面、多大な時間や費用が必要であり、コストや短周期の更新といった面では課題がある。都市は都心部など一部エリアで頻繁に変化していく一方、あまり変化しないエリアもあるため、変化点のみをピンポイントで抽出してデータ取得と更新を行うことができれば、3D都市モデルの鮮度を効率的に維持することができる。 • 本業務では、バス等のモビリティに搭載されたLiDAR等で定常的に取得される点群データや、スマートフォン等で市民が日常的に取得できるデータを活用することで、3D都市モデルのデータソースを取得。これに基づき都市の変化点の検出するA.I.及び3D都市モデルを生成する自動モデリングツールを開発することで、高頻度かつ低コストの3D都市モデル更新を目指す。
調査研究業務の概要	• 宮城県仙台市において、公共交通機関であるバスや一般車両に三次元測量を行うためのLiDARを設置し、定常的に高精度の点群データを取得する仕組みを構築。これに加え、LiDAR付きスマートフォンを活用し、市民参加により点群データを収集するためのクラウドソーシング型データ取得を行う。 • 取得した点群データをもとに、都市の変化点を抽出するため、都市の変化パターンを深層学習させたA.I.を開発し、既存の3D都市モデルをインプットデータとした変化点の自動検出を可能とするシステムを構築。さらに、このシステムが変化点として検出した箇所の点群データから、新たなLOD3建築物モデルを構築するため、既存のLOD2建築物モデルをベースとしたサーフェス再構築、属性情報付与、CityGML符号化等を行う3D都市モデル自動生成システムを構築する。 • これらの一連のシステムの開発及びフィジビリティスタディを行うことで、3D都市モデルの高頻度かつ低コストの更新スキームを確立し、デジタルツインの社会実装を加速する。

全体概要（2/2）

本実証の全体概要は下表のとおり

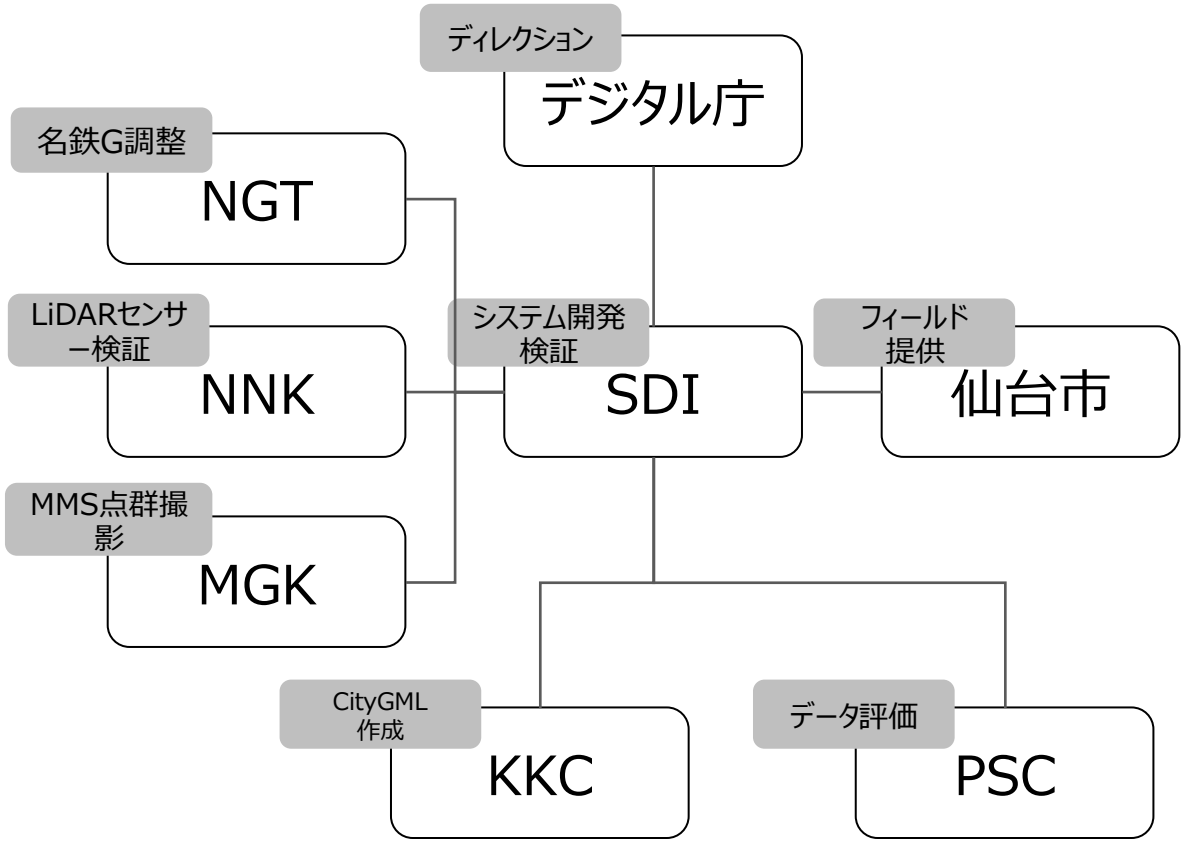
実証仮説	• バスやタクシーなどの公共交通へ取り付けられたLiDARセンサーや、iPhone LiDAR等のクラウドソーシング型のデータを活用することで、高頻度かつ低コストな3D都市モデルの更新を可能とする。 • 取得した点群データを活用した建築物の新築や滅失を判定する技術を開発することで、3D都市モデルの更新が必要な箇所をリアルタイムで把握する。 • 様々なデバイスから取得された断片的な点群データを統合して3D都市モデルを作成する技術を開発することで、多様なデータを活用した高頻度かつ低コストな3D都市モデルの更新を可能とする。
検証ポイント	• 一般車両へ後付け可能なLiDARを用いた簡易MMSによる点群データ取得方法の検証 • ノイズ除去、位置補正による断片化されたMMS点群、iPhone点群の合成精度の検証 • 自動異動判読による新築・滅失・滅失からの新築・増改築の検出精度検証 • 点群データからのサーフェス再構築及びメッシュ化精度の検証 • 生成されたLOD3モデルの精度検証、省力化の検証

実施体制

本実証の実施体制は下表及び右図のとおり

表　各主体の役割

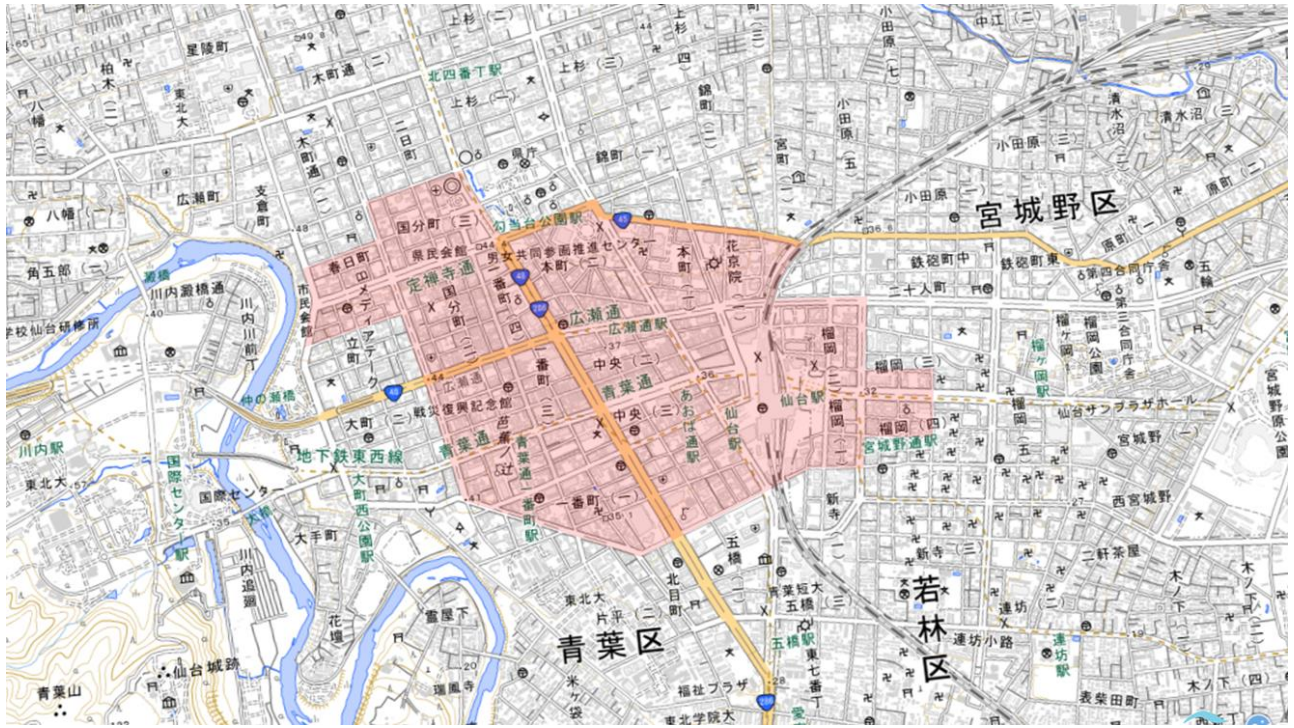
主体	役割
Symmetry Dimensions Inc.(SDI)	・全体統括 ・ニューラルネットワーク構築
名古屋鉄道株式会社(NGT)	・名鉄グループ各社調整
中日本航空株式会社(NNK)	・公共交通/一般車両のLiDARセンサー取付及びデータ作成
宮城交通株式会社(MGK)	・計測車両の運行及び測量業務
国際航業株式会社(KKC)	・LOD2範囲の実験用データ作成 ・CityGML作成及び省力化評価
株式会社パスコ(PSC)	・CityGML及び異動判読品質評価



実証エリア

本実証の実施対象エリアは下表の通り

宮城県　仙台市　都市再生緊急整備地域（約186ha内）



・実証実験エリア：宮城県仙台市都市再生緊急整備地域（赤枠内）

スケジュール

本実証は、下表のスケジュール沿って実施した

実施事項	令和4年							令和5年		
	6月	7月	8月	9月	10月	11月	12月	1月	2月	3月
1. 作成手法に関する検討	←	→								
2. 技術動向の調査	←	→	→							
3. 公共交通へのLiDAR取付手法検討	←	→	→							
4. MMS点群データ（バス・タクシー）撮影			←	→						
5. A.I.モデル構築		←	→	→	→	→	→			
6. MMS点群データからのサーフェス再構築					←	→	→	→	→	
7. 作成された3D都市モデルの評価・検証								←	→	
8. テスト／実証実験								←	→	
9. 報告書作成								←	→	

Ⅰ．実証概要

Ⅱ．実証技術の概要

Ⅲ．技術調査

Ⅳ．実証システム

Ⅴ．実証システムの検証

Ⅵ．成果と課題

活用技術一覧

本実証に関わる技術は以下の通り

項目	内容
City Generator	Symmetry Dimensions社が開発した、仮想空間におけるトレーニングデータ自動作成ツール - 都市空間の特徴あるオブジェクトを自動配置し仮想の都市を作成する - 仮想の都市内で簡易MMS（Velodyne VLP-16 LiDAR Puck）及びiPhone LiDARをシミュレートしてトレーニング用点群データを作成する
Blender	3DCG制作を行うオープンソースのソフトウェア。Pythonによるさまざまなプログラミング、自動化が可能。
CloudCompare	3D点群編集を行うオープンソースのソフトウェア。LAS、PLY、OBJなど28の入力形式に対応。
PyTorch	深層学習モデル開発に利用される主要フレームワークの一つ。Meta社の人工知能研究開発グループが開発。
Open3D	3Dデータを扱うオープンソースのPythonライブラリ。Intel Labsが開発。
簡易MMS	一般車両やタクシーなどの車両に後付けできる可搬型MMSで、センサは3kg（令和４年11現在）と軽く、マグネットや軽微な器具で車両に搭載できる。従来のシステムで必要だったオドメーターなどの付属機器、電源などの配線工事が不要。中日本航空社が開発。

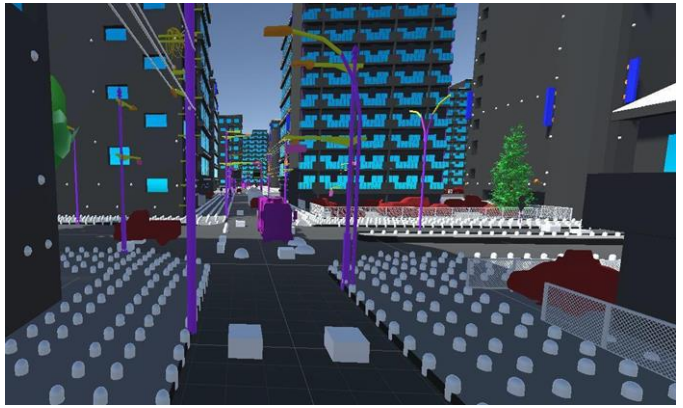
City Generatorについて

仮想空間におけるトレーニングデータ自動作成ツール

概要

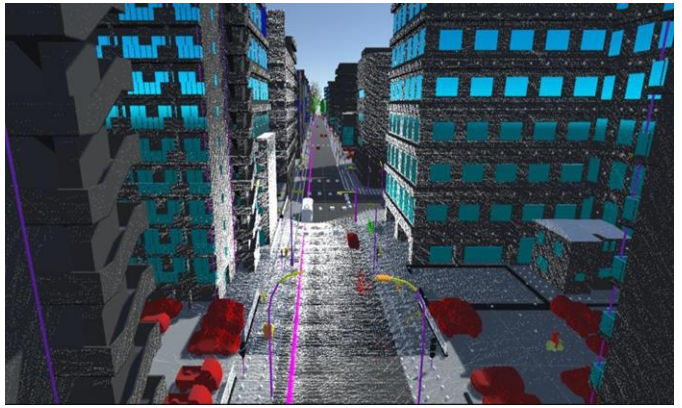
項目	詳細
名称	City Generator
概要	• A.I.モデルを構築するための大量のトレーニングデータを自動で作成する • 都市空間の特徴あるオブジェクトを自動配置し仮想の都市を作成する • 仮想の都市内で簡易MMS（Velodyne VLP-16 LiDAR Puck）及びiPhone LiDARをシミュレートしてトレーニング用点群データを作成する
主な機能	• 仮想都市作成機能 • MMS/iPhone/Drone LiDARシミュレート
本ユースケースで利用する機能	• 仮想都市作成機能 • MMS/iPhone/Drone LiDARシミュレート

使用イメージ



都市オブジェクトを配置して仮想の都市を自動作成

MMS/iPhoneをシミュレートして、点群を自動作成



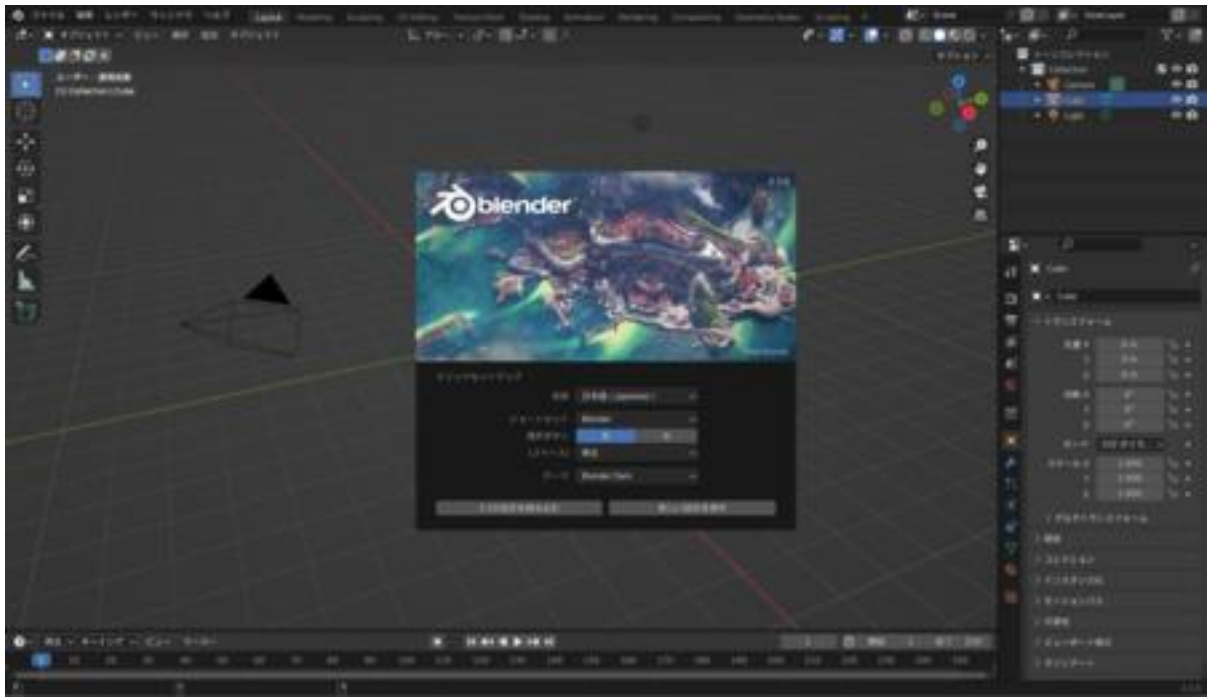
Blenderについて

3DCG制作を行うオープンソースのソフトウェア

概要

項目	詳細
名称	Blender
概要	• OSSの3D制作ツール • モデリング、アニメーション制作などをワンストップで可能。
主な機能	• 3DCG、アニメーション作成 • 各種フォーマット変換 • Blender Python（BPY）
本ユースケースで利用する機能	• フォーマット変換（FBX→OBJ） • Python言語による自動処理

使用イメージ



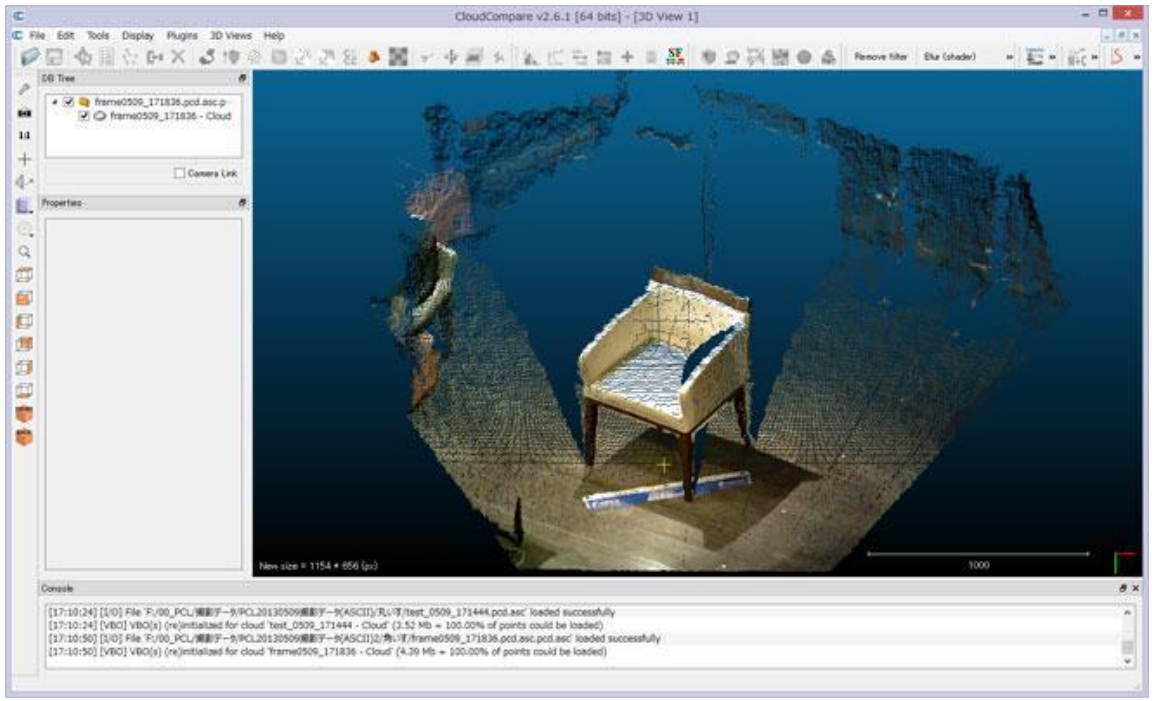
CloudCompareについて

3D点群編集を行うオープンソースのソフトウェア。

概要

項目	詳細
名称	Cloud Compare
概要	• 3D点群データとメッシュデータの処理のためのオープンソースソフトウェア
主な機能	• 点群データの読み込みと表示 • 点群データの切り取り • 点群データの移動 • 点群データのフィルタリング
本ユースケースで利用する機能	• 3Dモデル(OBJ)の点群データへの変換

使用イメージ



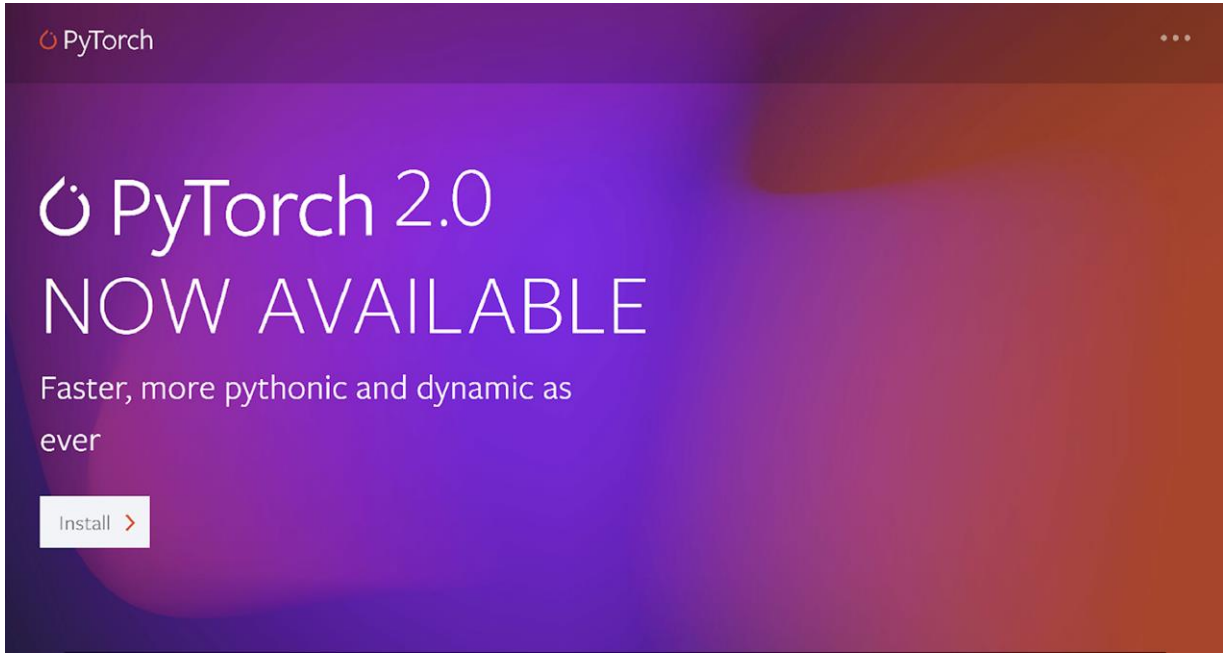
Pytorchについて

深層学習モデル開発に利用される主要フレームワークのひとつ

概要

項目	詳細
名称	PyTorch
概要	• 深層学習モデル開発に利用される主要フレームワークの一つであり、Meta社の人工知能研究開発グループによって開発。 • モデリング、アニメーション制作などをワンストップで可能。
主な機能	• A.I.モデルの開発フレームワーク
本ユースケースで利用する機能	• A.I.モデルの開発フレームワーク

PyTorch公式HP



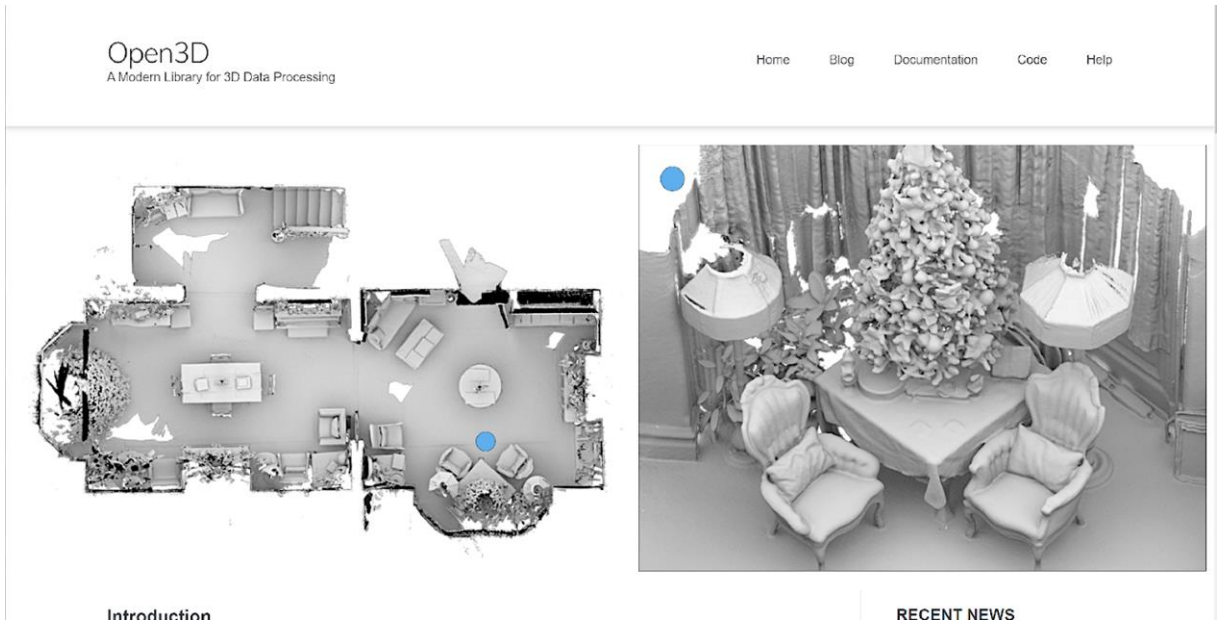
Open3Dについて

3Dデータを扱うオープンソースのPythonライブラリ

概要

項目	詳細
名称	Open3D
概要	• 3Dデータの可視化、前処理、解析のライブラリ
主な機能	• 基本的な3Dデータ構造及び処理アルゴリズ • シーンの再構築 • 3Dビジュアライゼーション
本ユースケースで利用する機能	• 3Dビジュアライゼーション

Open3D公式HP



簡易MMS

Mobile Measurement System(MMSは車両等に装着したLiDARセンサー等によって周囲を測量する装置

概要

項目	詳細
名称	MMS N-QUICK
概要	中日本航空(株)が開発した簡易MMSは、一般車両やタクシーなどの車両に後付けできる可搬型MMS
主な機能	• LiDAR点群の取得 • 自己位置の推定
本ユースケースで利用する機能	• LiDAR点群の取得 • 自己位置の推定

イメージ図

Ⅰ．実証概要

Ⅱ．実証技術の概要

Ⅲ．技術調査

Ⅳ．実証システム

Ⅴ．実証システムの検証

Ⅵ．成果と課題

調査内容及び調査方針

本実証に関わる技術調査内容及び方針は下表の通り

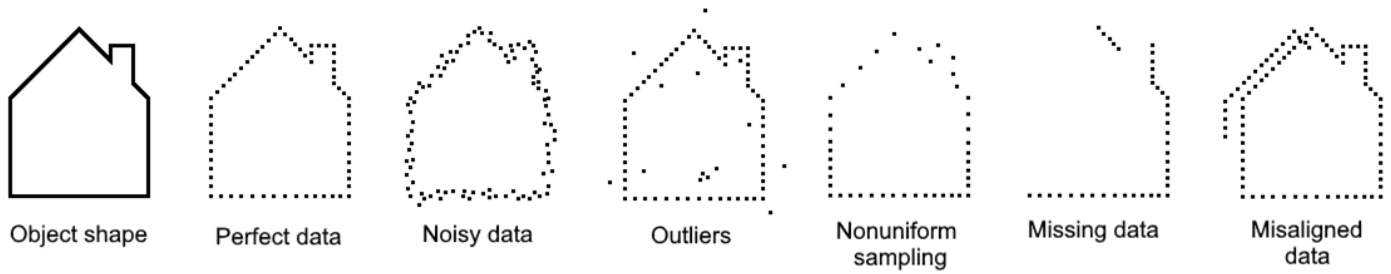
調査内容	• 3D 都市モデルの作成検証及び自動モデリングツールの開発に関する調査 - 国内外の企業、大学等における点群・3Dモデリングに関連した技術開発、A.I.手法の論文、社会実装の状況について調査を実施し、本実証で採用する技術選定を行う。
調査方針・課題	• 点群データの取得 - モビリティ運行基盤としての3次元地理空間情報の取得方法 - 公共交通機関の通常運行時に利用可能な機器・取付方法・撮影手法 - 市民からの情報提供として、スマートフォン（iPhone）のLiDARセンサーを活用したデータ取得 • 点群データのノイズ除去 - 移動物やセンサーに起因するノイズのA.I.による自動化された除去 - A.I.学習のためのトレーニングデータ作成手法 • 点群データの合成 - 日時や環境条件、取得ソースの異なる断片化した点群データから建物単位の点群データを合成する手法 • 都市の更新箇所検出 - 取得された点群データから、都市の更新箇所（新築・滅失・滅失から新築・増改築）を検出する手法 • サーフェス再構築 - 3D都市モデル（CityGML LOD1/2/3）の作成を前提とした点群データからのサーフェス再構築手法

自動モデリングツール開発のための手法①

点群データからサーフェスを再構築する手法・アルゴリズムについて、調査を実施した。

- 点群データからサーフェスメッシュを再構築する場合、複数の潜在的な誤差の処理が必要になる。本技術調査では、車両に搭載されたセンサーの利用による広域な都市部のスキャンでの問題点を下表に示す

ノイズ種別	概要
ノイズの多いデータ（Noisy Data）	測定ノイズには、センサーノイズ等のセンサー側に起因するものと対象物の反射率など表面特性で引き起こされるものがある。ノイズ強度と点密度の粗さは、一般的にセンサーからの距離によって増加する。ノイズの前処理手法として、ノイズの平滑化や、平滑面の予測など新しい手法も導入されている [4, 17].
はずれ値（Outliers）	物理的な空間に存在しない"幽霊"のようなデータ。対象物の表面反射やデータ取得プロセスの不完全さが主な原因である。はずれ値はサーフェス再構築プロセスでは扱いにくいエラー類型になり、局所的な点密度測定、シーン内を移動するセンサーの時間的な持続性を観察することで、検出・フィルタリングできる場合が多くある。
データの欠落（Missing Data）	対象物とセンサー間にある別の物体による遮蔽や、対象物の大きさ・距離・角度の影響で取得範囲に収まらない場合に発生する可能性がある。データ欠損箇所をどのように処理するかは、それぞれの再構築手法で決定される。一般的には対象物が「密閉型」であると仮定し、データ欠損箇所を一方に延長して補完する。本実証のような屋外の広範囲スキャンの際は、極めて不正確な物体形状につながる。
不均一な取得（Nonuniform sampling）	測定時の何らかの原因で、時間的に等間隔で撮影されず、点密度の不均一なデータ。点密度の修正や補正が困難な場合がある。
位置がずれたデータ（Misaligned Data）	スキャン時の個々の測定値の誤差等で正しくレジストレーションされない場合に発生する。データ処理のパイプライン初期に発生する問題で、後の修正や補正が困難な場合がある。



自動モデリングツール開発のための手法②

点群データからサーフェスを再構築する手法・アルゴリズムについて、調査を実施した。

- 点群データからのサーフェス再構築、物体検出の研究開発の多くは数十年前に発表された手法の改良が中心である。近年、自動運転技術の大幅な進化に伴い、都市等の大規模なユースケースに対する研究へ移行が進んでいる。本技術調査では、サーフェス再構築についての網羅的な研究論文を下表に示す

アルゴリズム・手法	概要
SDF Representation dynamics node（SDF）	入力点群である未組織点の集合との間の（符号付き）距離に基づいて暗黙的サーフェスを記述するアルゴリズムである[5]。1992年に導入され、すぐにこのタスクのための基本的アルゴリズムの1つとなり、暗黙的な表現から明示的なメッシュサーフェスを再構築し、さらに昔ながらのマーチングキューブアルゴリズムなどの他のツールとも組み合わされている [6]。過去 20年間，Surfels(Surface Element)は複雑な形状を素早くレンダリングするための一般的な方法として使用されている。
Geometric Objects	計算負荷の多くを前処理工程に移し，ソースデータの高速トラバース可能なOctreeライクな表現を作成，ポリゴンメッシュなど他のアプローチと比較して非常に高速なレンダリングを可能にする [7]．点群データからのメッシュ再構築に利用された新しい研究論文が現在も定期的に発表されている [8]
Moving least squares (MLS)	集合内の各点に対して加重最小二乗法を計算することにより，未整理の点群から滑らかな一続きのサーフェスの再構築する長い歴史を持つツールである [10]。MLSは、ノイズの多い入力の処理、ローカルフィッティングによる高品質な表面法線の近似検出に優れており、アルゴリズムの新しいアップデートにより、特徴の正確なモデリングなど、以前は欠けていたものも可能にっている [12, 13]。
Truncated Signed Distance Function (TSDF)	Microsoft Kinect のような低コストの深度カメラによって生成されたデータを扱うために特別に作成されたボリューム点群処理アルゴリズムであり，点群をボクセル表現に変換するためにSDFの亜種を使用する。このアルゴリズムは、ノイズの多いデータを統合して、リアルタイムでインタラクティブに変更できる高品質のシーンモデルを作成することが可能だが、メモリ消費量が多いため、一部のアプリケーションでの使用が制限される [9, 10, 12]。
Probabilistic Volumetric Reconstruction（PVR）	この手法は、特に草や木の葉のような曖昧な領域を扱う際に有用なモデルである [16]。また、SDFや Surfelsなどの他のアプローチと組み合わせることで、局所的な不確実性に応じて異なる符号化方法を動的に切り替えることができる表面表現を作成することが可能である [17]。
Poisson Surface Reconstruction（PSR）	SDFと同じ研究者らによる新しい方法であり、サーフェス再構築問題に対して異なるアプローチをとり、分割や局在化を行わず、入力点群全体を一度に考慮する。従来の手法よりも詳細なサーフェスの再構築することが可能だが、高速な処理と引き換えに細かい部分が失われてしまうことがある。また、建物の部分的なスキャンしか得られない場合，未視認領域の形状がアルゴリズムの出力と大きく異なることがあるため，このアルゴリズムのもう一つの特徴である密なメッシュ作成が逆に問題となる場合がある [12, 14, 15]。

自動モデリングツール開発のための手法③

点群データからサーフェスを再構築する手法・アルゴリズムについて、調査を実施した。

- 2010年代までは、入力点群が数百万点に及ぶ可能性がある場合でも、小規模スキャンからの再構築に焦点を当てたメッシュ再構築手法が主であった。近年では、屋外での点群からメッシュへの変換アルゴリズムを利用し、高い詳細度を維持しながら、オブジェクトを正確にセグメントするユースケースが劇的に増加している

アルゴリズム・手法	概要
屋外などの大規模な点群に対する近年のアプローチ	”3D Surface Reconstruction from Voxel-based Lidar Data”は、車載LiDARで取得した点群をマルチスケールボクセル化し、さらにTSDF計算によりスキャン領域の連続メッシュサーフェスを再構築するメモリ使用量最適化システムを実装するものである。単眼2DカメラとLiDARセンサーなどの深度計測の両方を利用した同時定位マッピング（SLAM）は複数存在するが、一般的にはどちらかの手法が他方をサポートするためだけに利用されている（例：2Dカメラデータは3Dセンサー由来のモデルのテクスチャリングにのみの利用等） [18] 。
	“Lidar-Monocular Surface Reconstruction Using Line Segments”では、単眼2DカメラとLiDARの両方を併用し、2D画像で検出された線を3D点群にマッピングして環境のきれいなサーフェス再構築を作成するシステムを提案し、従来のSLAM実装と比較したこのアプローチの利点について論じている。広範囲のエリアをカバーするデータセットの場合、航空機に取り付けられたLiDARセンサーを見下ろす角度で測量し、建物の屋根構造を正確に表現する航空点群データを使用することが可能である。このデータをサーフェス再構築に利用することで、LOD2レベルの建物表現が可能となるが、屋根下の3次元情報がないため、より高いレベルのディテールを生成することができないという問題がある。また、建物底面の形状と上から見た屋上の形状・大きさが必ずしも一致しないため、リアルな建物モデルを自動的に作成することは困難である [19]。
	“Strategies for 3D Modelling of Buildings from Airborne Laser Scanner and Photogrammetric Data Based on Free-Form and Model-Driven Methods: The Case Study of the Old Town Centre of Bordeaux”では,複数のメッシュ構築方法を試した結果、どの方法も人間の入力ステップが含まれるシステムには勝てないことを認めている [20]。
	大規模な都市モデルを自動的に生成する最近の例として、Huangらによる2022年の論文“City3D: Large-scale Urban Reconstruction from Airborne Point Clouds”があり、彼らのアルゴリズムは、数万の建物を含む点群に対して最適な3Dメッシュを作成することが可能である [21]。
	航空LiDARスキャンを地上計測と組み合わせることを論じた論文は驚くほど少なく、“Urban Flood Modelling Combining Top-view LiDAR Data with Ground-view SfM Observations”のように、洪水のモデリングや予測など、より専門的な用途に焦点が当てられている [22]。
	空中のLiDARと地上のLiDARデータを自動的に合成するシステムの研究の1つとして、バークレー大学から2009年のテクニカルレポート“Fast Surface Reconstruction and Segmentation with Ground-Based and Airborne LiDAR Range Data”に記載されている。主な内容は、両方のデータソースからのメッシュ再構築と地面のセグメンテーションであり、地上からのスキャンでは見えない領域（屋根）を検索し、生成した高さマップを使って空中データからその領域を抽出し、その境界エッジで2つのメッシュを「縫い合わせる」ことによって、2つのメッシュを結合するアルゴリズムについて記述されている [23]。

自動モデリングツール開発のための手法④

点群データからサーフェスを再構築する手法・アルゴリズムについて、調査を実施した。

アルゴリズム・手法	概要
屋外などの大規模な点群に対する近年のアプローチ	[23]と同時期に東京大学で行われた別の研究“Three-Dimensional Modeling of an Urban Park and Trees by Combined Airborne and Portable On-Ground Scanning LiDAR Remote Sensing”では、航空LiDARスキャンと地上スキャンを組み合わせて、樹木の3Dモデルの作成と各種データの計測が検討された。しかし、彼らの研究は、自動化されたパイプラインや新しいアルゴリズムの実装ではなく、一般的なアイデアの探求に焦点を当てたものである [24]。
	SLAM技術に代わるアプローチとして、PSR（Poisson surface reconstruction）アルゴリズムを用いて、車両に搭載したLiDARによって短時間に取得した多数の点群からメッシュを作成し（単一スキャンによる誤差を最小限に抑える）、これを正確な車両位置決めとグローバルマップの作成に利用する方法を“Poisson Surface Reconstruction for LiDAR Odometry and Mapping”で紹介している。彼らの実装の興味深い点は、作成された各メッシュの三角形の密度スコアで、これを用いて、十分な点サンプルを含まないメッシュの領域を消去することができる点である。これにより、PSRの一般的な問題である、結果として得られるメッシュが完全に密閉されてしまうという問題を回避することが可能である。彼らのテスト結果は、TSDFのような他の一般的なサーフェス再構築手法よりもかなり良い結果に到達できることを示唆しており、彼らのdensity filtering enhanced Poisson surface reconstructionアルゴリズムが、SLAM的ローカリゼーションという大きな文脈から切り離しても有用なツールになる可能性があることを示唆している [25]。
	“Dense 3D Surface Reconstruction of Large-Scale Streetscape from Vehicle-Borne Imagery and LiDAR”はSLAMの実行や置き換えではなく、SLAMの位置特定を利用するシステムである。大規模な街並みの高密度・高品質の3D再構築は、まず、ターゲットエリアを通過する車両の走行全体から選択したキーフレームから粗い表現を作成することによって達成される。それぞれの新しいキーフレーム（2DカメラとLiDARデータ）は、既存の表現を更新してノイズや冗長性を取り除き、高密度再構築に向けて段階的に進むのに使用される [26]。

自動モデリングツール開発のための手法⑤

点群データからサーフェスを再構築する手法・アルゴリズムについて、調査を実施した。

- 深層学習によるアプローチも登場しており、前処理を行わない点群データによる三次元形状の学習によりサーフェス再構築を可能にする研究や、安価なRGB-Dカメラからノイズの影響を効果的に軽減しサーフェス再構築の可能性が示されている。

アルゴリズム	概要
Deep Learning Approach	最近では、深層学習ベースのサーフェス再構築モデルも登場しており、“Implicit Geometric Regularization for Learning Shapes”や、“SAL: Sign Agnostic Learning of Shaped from Raw Data”では、符号非依存学習（SAL: Sign Agnostic Learning）では、未ラベリングのRawデータを用いて、点群からの直接メッシュを再構築するアプローチである [27,28]。
	“Deep Implicit Moving Least-Squares Functions for 3D Reconstruction”は、ニューラルネットプラットフォームに古典的手法を実装する方法の提案である。これらの深層学習ベースのシステムは，車両スケールまでの点群からメッシュを再構築する単一オブジェクトのユースケースに依然として焦点を当てており，都市全体を取り扱う規模のメッシュ再構築に拡張されている例は存在しない。これは[2]でも裏付けられており、著者らは、現在の深層学習ベースの手法は、古典的なサーフェス再構築手法と比較して、学習データセットの内容外のオブジェクトにあまり汎化しないことを明らかにしている [29]。

自動モデリングツール開発のための手法⑥

点群データからノイズを除去する手法・アルゴリズムについて、調査を実施した。

- 点群データからノイズを除去するための前処理としてのセマンティックセグメンテーション手法は、多くが2D画像ベースのものであるが、3Dデータソースを入力した研究もLiDARセンサーやRGB-Dセンサーの低価格化により増加している

アルゴリズム・手法	概要
RandLA-Net	シンプルなアイデアで高速・高精度な三次元点群データのセマンティックセグメンテーションを可能にする“RandLA-Net: Efficient Semantic Segmentation of Large-Scale Point Clouds”は、局所点群に対して注目点からの相対座標に基づくAttentionを適用してから特徴量を計算し，これとResidual Connectionを組み合わせたブロックで点群を処理する。Residual Connectionが注目する局所領域の拡大に対応しており，このブロックを重ねるだけで自然に対応する局所領域が拡大される [30]。
PointNet	LiDARなどから取得される点群データに対し、2D空間へのマルチビュー投影や3Dボクセル化などを行わず点群データからロバストな特徴表現を直接学習する深層学習手法である [31]。
Cylinder3D	発表当時、SemanticKIITTIやその他のテストデータセットで世界1位の成績を収め注目された“Cylindrical and Asymmetrical 3D Convolution Networks for LiDAR Segmentation”は、スパースデータを編kなさせて性能向上を図り、他の畳み込み式点群セグメンテーションモデルの性能を超えようとしている。既存のモデルは弱い点群密集部分と点が少ない箇所の両方で上手く機能するこを目指している [32]。

- 更新された建物の検出や、点群間での差異を検出し、位置の補正を行うロバスト推定手法は、古典的手法から新規のものまで多く存在する。本調査では特に都市で取得される広範囲・大容量の点群利用を前提に調査を行った。

アルゴリズム・手法	概要
Ransac	“Usac: A universal framework for random sample consensus”は、ロバスト推定の古典的なアルゴリズムで、ランダムサンプリングを繰り返す事により、外れ値を含まない”あたり“を引き当てる手法である。RANSACのネックは処理の高速化であり、様々な手法が検討されている。[33]。
ICP 及びその亜種	2つのターゲット点群間の姿勢や位置を調整するICP“Least-Squares Fitting of Two 3-D Point Sets”は、古典的な手法であり“Efficient Variants of the ICP Algorithm”等の多くの亜種が存在している。初期値に大きく依存するため、あらかじめ適度に近い位置が必要であり、しばしば他のアルゴリズムと併用して使用され、繰り返しの計算により精度をあげていく [34][35]。
DCPCR	“DCPCR: Deep Compressed Point Cloud Registration in Large-Scale Outdoor Environments”の特徴は、圧縮された点群に対しての精度が高いことにあり、屋外の広範囲での利用を前提とした都市空間での利用に最適である。ノイズの大きい入力データや書記と大きく異なるターゲット点群に対し良好なパフォーマンスを発揮する。[36]。

実証システムでの技術選定

本実証システムで採択する手法・アルゴリズムは下表のとおり

アルゴリズム・手法	概要
RandLA-Net	点群データの領域分類（セマンティックセグメンテーション）において、「SphereTransformer」や「Cylinder3D」など最新モデルが登場しており、採択を検討したが仕様前の作業が膨大であったこと、最新モデルではないが、非常に高速で他のプロジェクトで多く利用されており実績のあるMITライセンスであるOpen3Dライブラリに含まれる「RandLA-Net」を採択した。RandLA-Netが発表された2020年ではその識別性能だけはなく、速度においても最先端であったが、2022年時点でもトップに近い性能を持ち、一般的なCUDA対応GPUを搭載したPCで、数万点の点群をほぼリアルタイムに処理することが可能である。
DCPCR/GICP	都市の変位箇所検出おいて、「RANSAC (Random SAMpling Concensus)」 や「ICP」及びその亜種である「Point to Point ICP」、「Point to plane ICP」、「汎用ICP」などが有力なアルゴリズムとなる。本実証では、屋外での広域な点群データを取り扱うことが前提のため、競合と比較して大容量の点群を効率的に取り扱え流転、近似点検出やレジストレーションが可能な点、非常にノイズの多い入力や大きな回転差のある点群データ入力でも良好なパフォーマンスを発揮する「DCPCR」及び「GICP」の組み合わせを採択した。
ICP	点群データの合成・位置合わせにおいては、利用実績が多数であり、機能性が高く、高速で、点群を移動させる際の最大量を指定可能な「Point to Pont ICP」を採択した。
PSR	点群データのサーフェス再構築において、「SDF Representation dynamics node（SDF）」、「Probabilistic Volumetric Reconstruction（PVR）」、「Truncated Signed Distance Function (TSDF)」、「 Moving least squares (MLS)」、「Geometric Objects」など様々なモデルが存在している。一方で、これらのアルゴリズムは古くから使われているモデルの改良であり、小型のセンサーによる局所的なオブジェクトの表面再構築での利用が主であり、本実証における都市空間の地物、建物などの巨大なオブジェクトには適合しないと判断した。本実証では、技術調査時点で点群データから水密な（穴などの抜けのない）3Dモデルを出力できる唯一のモデルであった「PSR」を採択した。
City Generator	「RandLA-NET」で学習を行うためには対象となるオブジェクトに手動でラベリング処理を行った大量のトレーニングデータが必要となる。手作業で付与されたラベルは常に不正確さを含み、必要なトレーニングデータセットは数万点に及び、ラベリング作業も時間がかかるため、本実証では、自動でラベリング処理及びトレーニングデータ生成を可能にするソフトウェア「City Generator」を採択した。

点群データの取得①

点群データの取得について、調査を実施した。

- 公共交通への取り付け及び運行中のデータ取得が可能な簡易MMSについて、本技術調査で行った項目を下表に示す

項目	概要
LiDARセンサーに関する調査	 構成・スペック **センサー部** 重さ：3kg（ヘッド部） レーザスキャナ：Velodyne VLP16×1台 GNSS/IMU：Septentrio AsteRx-i3 S Pro+ ×1台 カメラ：USB2.0カメラを車載カメラとして提供 **処理部** コンピュータ：Intel NUC (Windows OS) 電源：車両シガーソケットとDC/ACインバータ ソフトウェア：中日本航空　収録・演算ソフト **その他** 通信：車載PCのインターネット接続が必要 基準局サービス：Ntrip形式の配信サービスが必要 **レーザスキャナ** レーザ：Laser Class 1 eye safe 波長903nm 取り付け角：約45度（車体に対して） 視野角：VLP16: 水平360度全周、垂直±15度 反射強度：あり 発射点数：VLP16: 約30万点/秒 最大到達距離：VLP16: 100m（実質の有効距離は25m程度） 測距精度：±3cm（1σ@25m） **GNSS/IMU** 位置測位：■上空視界の良い場所（RTK-FIX） 高さ：1cm＋1ppm（2cm@10km相当） 水平：0.6cm＋0.5ppm（1cm@10km相当） ■橋や樹木の下を通過する場合（非FIX時間が短い） 高さ：約5cm、水平：約30cm ■長時間GNSS環境が悪い場合(非FIX時間が30秒以上) 高さ：約20cm、水平：約300cm 姿勢角：Roll/Pitch 0.02度 Heading 0.15度 **総合的な性能目安** 絶対精度：約10cm （GNSS測位が良い場合） 相対位置：約5cm （GNSS測位が良い場合）

点群データの取得②

点群データの取得について、調査を実施した。

- 公共交通への取り付け及び運行中のデータ取得が可能な簡易MMSについて、本技術調査で行った項目を下表に示す

項目	概要
運行ルート及び地物に関する調査	【調査結果】 • 本実証のエリアである仙台市駅周辺は、ビルに囲まれた地域であり、ガラス張りの高層ビル（マルチパス）、細い路地（アーバンキャニオン）、街路樹の立ち並ぶ道路など、GNSS測位環境が悪い状況であった。実際の作業に当たっては、都市部における精度向上及び、後続作業（点群マッチング、クリーニング処理）が重要であることが確認できた。 • 本実証では、宮城交通株式会社のバス路線によるデータ取得を行うため、運行ルートの調査を行った。バス路線だけでは建物正面しかデータが取得できないことが判明した。ビルに囲まれた細い路地等では、位置情報を取得するためのGNSS測位環境が悪いことがわかった。また、日中は交通渋滞や駐車車両による問題が確認できた。深夜の計測を行ったところ、不要な車両などが少なく、車線変更も容易なためGNSS受信状況に配慮した計測作業を実施できることがわかった。 • 本実証のエリア全域を車両に設置したカメラ撮影を行った。バス路線に関しては、360度カメラによる映像撮影を行った。 【課題対応】 • バス路線だけでは取得できない建物は、タクシーを想定した一般車両でのデータ取得を実施することとした。システムはバスに取り付けるものと同じ簡易MMSの取り付けを行う。 • 対象地域はビルに囲まれた地域であり、ガラス張りの高層ビル（マルチパス）、細い路地（アーバンキャニオン）、街路樹の立ち並ぶ道路など、GNSS測位環境が悪い状況をプレ計測で確認した。その後システムに改良を加えて、8月の一般車両計測では近傍でのGNSS基準点設置（My基準点）、受信機アップグレード、夜間の計測等により品質を向上させた。

点群データの取得③

点群データの取得について、調査を実施した。

- 公共交通への取り付け及び運行中のデータ取得が可能な簡易MMSについて、本技術調査で行った項目を下表に示す

項目	概要
バスでのLiDARセンサー取付手法及び取得データの調査	【調査結果】 • バスへの取り付けにおいては、車体の改造を行わずに実装が可能なマグネット方式の陸運局への確認を行った。 • 電源管理においては、安定した電力供給を可能にするため10時間の連続供給可能なモバイルバッテリーの利用を行った。 • 本実証での運用においては、システムの改良（自動化、安定化）や実際の運航者である宮城交通と協議を行い、データ収集のために作業員がバス運行所に常駐しなくとも良い体制を構築した。実際には、宮城交通の運転手が出発前にデータ保存用SSDとモバイル電源を交換して、後は通常運行によりデータ収集できるように設計を行った。また、作業員はインターネット経由で愛知県の事務所からデータ収集状況を確認可能とした。 <車外側：センサー取付> ・車体改造等を行わない ・強力マグネット式 （陸運局確認済み）   <車内側：電源と管理> ・安定するモバイル電源とした ・10時間の連続供給 （自動終了機能）  

点群データの取得④

点群データの取得について、調査を実施した。

- 公共交通への取り付け及び運行中のデータ取得が可能な簡易MMSについて、本技術調査で行った項目を下表に示す

項目	概要
バスでのLiDARセンサー取付手法及び取得データの調査	【調査結果】 • 公共交通（バス）でのLiDARセンサーデータ取得は、名古屋鉄道及びグループ企業である宮城交通と共同で実施した。バスに機材を取り付けるための法令や技術的な課題、実施スケジュールの調査・確認を行った。 • システムは既存の簡易MMSをベースに、バスの整備や管理者と協議を重ねて、確実に搭載できる方法を確認した。 ＜簡便な運用法＞ 宮城交通の運転手が運用できるように、できるだけシンプルな操作にした。また、手順や操作ラベルと目につくようにした。  ＜遠隔監視可能な環境整備＞ ・遠隔での対処を可能にする環境を構築 

Ⅲ. 技術調査 > 5. 参考論文

参考論文①

参考論文を下表に示す

参照論文	[1] Berger M., Tagliasacchi A., Seversky L.M., Alliez P., Guennebaud G., Levine J.A., Sharf A., and Silva C.T., “A Survey of Surface Reconstruction from Point Clouds”, in Computer Graphics Forum, Volume 36, Issue 1, pp. 301-329, January 2017. [2] Huang Z., Wen Y., Wang Z., Ren J., and Jia K. “Surface Reconstruction from Point Clouds: A Survey and a Benchmark”, May 2022. [3] Zeng Xin, ”A Fast Wall Surface Point Cloud Extraction Method for Generating 3D Models of Buildings using MMS Point Cloud Data”, Ph.D. dissertation, Kyushu Institute of Technology, 2022. http://hdl.handle.net/10228/00007580 [4] Craciun D., Deschaud J.E., and Goulette F., “Automatic Ground Surface Reconstruction from mobile laser systems for driving simulation engines”. SIMULATION, SAGE Publications, 2017, [5] Hoppe H., DeRose T., Duchamp T., McDonald J., and Stuetzle W., “Surface Reconstruction from Unorganized Points”, ACM SIGGRAPH 1992. [6] Lorensen W.E, and Cline H.E., “Marching Cubes: A High Resolution 3D Surface Construction Algorithm”, in Computer Graphics, Volume 21, Number 4, July 1987. [7] Pfister H., Zwicker M., van Baar J., and Gross M., “Surfels: Surface Elements as Rendering Primitives”, SIGGRAPH 2000. [8] Schöps T., Sattler T., and Pollefeys M., “SurfelMeshing: Online Surfel-Based Mesh Reconstruction”, in IEEE Transactions on Pattern Analysis and Machine Intelligence, September 2018. [9] Izadi S., Kim D., Hilliges O., Molyneaux D., Newcombe R., Kohli P., Shotton J., Hodges S., Freeman D., Davison A., and Fitzgibbon A., “KinectFusion: Real-Time 3D Reconstruction and Interaction Using a Moving Depth Camera”, in Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology UIST’11, pp.559-568, October 2011. [10] Splietker M and Behnke S., “Directional TSDF: Modeling Surface Orientation for Coherent Meshes”, in IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), November 2019. [11] Alexa M., Behr J., Cohen-Or D., Fleishman S., Levin D., and Silva C.T., “Point Set Surfaces”, in IEEE Visualization 2001, October 2001. [12] Meerits S., Nozick V., and Saito H., “Real-time Scene Reconstruction and Triangle Mesh Generation using Multiple RGB-D Cameras”, in Journal of Real-Time Image Processing 16, pp. 2247-2259, 2019.

Ⅲ. 技術調査 > 5. 参考論文

参考論文②

参考論文を下表に示す

参照論文	[13] Fleishman S., Cohen-Or D., and Silva C.T., “Robust Moving Least-Squares Fitting with Sharp Features”, in ACM Transactions on Graphics, Volume 24, Issue 3, pp.544-552, July 2005. [14] Kazhdan M., Bolitho M., and Hoppe H., “Poisson Surface Reconstruction”, in Symposium of Geometry Processing 2006, June 2006. [15] Kazhdan M., Chuang M., Rusinkiewicz S., and Hoppe H., “Poisson Surface Reconstruction with Envelope Constraints”, in Eurographics Symposium on Geometry Processing 2020, volume 39, number 5, July 2020. [16] Ulusoy A.O., Geiger A., and Black M.J., “Towards Probabilistic Volumetric Reconstruction using Ray Potentials”, in 2015 International Conference on 3D Vision, 2015. [17] Dong W., Wang Q., Wang X., and Zha H., “PSDF Fusion: Probabilistic Signed Distance Function for On-the-fly 3D Data Fusion and Scene Reconstruction”, in European Conference on Computer Vision, ECCV 2018, October 2018. [18] Roldão L., Charette R., and Verroust-Blondet A., “3D Surface Reconstruction from Voxel-based Lidar Data”, in 2019 IEEE Intelligent Transportation Systems Conference (ITSC), June 2019. [19] Amblard V., Osedach T.P., Croux A., Speck A., and Leonard J.J., “Lidar-Monocular Surface Reconstruction Using Line Segments”, in 2021 IEEE International Conference on Robotics and Automation (ICRA), June 2021. [20] Constantino D., Vozza G., Alfio V.S., and Pepe M., “Strategies for 3D Modelling of Buildings from Airborne Laser Scanner and Photogrammetric Data Based on Free-Form and Model-Driven Methods: The Case Study of the Old Town Centre of Bordeaux (France)”, in Applied Sciences 2021. [21] Huang J., Stoter J., Peters R., and Nan L., “City3D: Large-scale Urban Reconstruction from Airborne Point Clouds”, January 2022. [22] Meesuk V., Vojinovic Z., Mynett A.E., and Abdullah A.F., “Urban Flood Modelling Combining Top-view LiDAR Data with Ground-view SfM Observations”, in Advances in Water Resources 75, January 2015. [23] Carlberg M., Andrews J., Gao P., and Zakhor A., “Fast Surface Reconstruction and Segmentation with Ground-Based and Airborne LiDAR Range Data”, Berkeley University Technical Report No. UCB/EECS-2009-5, January 2009. [24] Omasa K., Hosoi F., Uenishi T.M., Shimizu Y., and Akiyama Y., “Three-Dimensional Modeling of an Urban Park and Trees by Combined Airborne and Portable On-Ground Scanning LiDAR Remote Sensing”, in Environmental Modeling & Assessment 13, pp. 473-481, November 2008.

参考論文③

参考論文を下表に示す

参照論文	[25] Vizzo I., Chen X., Chebrolu N., Behley J., and Stachniss C, “Poisson Surface Reconstruction for LiDAR Odometry and Mapping”, in 2021 IEEE International Conference on Robotics and Automation (ICRA), June 2021. [26] Lin X., Yang B., Want F., Li J., and Wang X., “Dense 3D Surface Reconstruction of Large-Scale Streetscape from Vehicle-Borne Imagery and LiDAR”, in International Journal of Digital Earth 2021, Volume 14., Number 5, pp.619-639. 2021. [27] Gropp A., Yariv L., Haim N., Atzmon M., and Lipman Y., “Implicit Geometric Regularization for Learning Shapes”, in 37th International Conference on Machine Learning,July 2020. [28] Atzmon M. and Lipman Y., “SAL: Sign Agnostic Learning of Shaped from Raw Data”, in 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), June 2020. [29] Liu S., Guo H., Pan H., Wang P., Tong X., and Liu Y., “Deep Implicit Moving Least-Squares Functions for 3D Reconstruction”, in Conference on Computer Vision and Pattern Recognition CVPR 2021, June 2021 [30] Qingyong Hu, Bo Yang, Linhai Xie, Stefano Rosa, Yulan Guo, Zhihua Wang, Niki Trigoni, Andrew Markham “RandLA-Net: Efficient Semantic Segmentation of Large-Scale Point Clouds”, om Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2020 [31] Yulan Guo, Hanyun Wang, Qingyong Hu, Hao Liu, Li Liu, and Mohammed Bennamoun “Deep Learning for 3D Point Clouds: A Survey” [32] Zhu X., Zhou H., Wang T., Hong F., Ma Y., Li W., Li H., and Lin D., “Cylindrical and Asymmetrical 3D Convolution Networks for LiDAR Segmentation”, in 2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2021. [33]R.Raguram, O.Chum, M.Pollefeys, J.Matas, and J.Frahm, "Usac: A universal framework for random sample consensus", IEEE Transactions on Pattern Analysis and Machine Intelligence(PAMI), 2013 [34] Szymon.R and Marc.L, “Efficient Variants of the ICP Algorithm”, Proceedings Third International Conference on 3-D Digital Imaging and Modeling, 2001, pp. 145–152. [35] K.S.Arun; T.S.Huang; S.D.Blostein, “Least-Squares Fitting of Two 3-D Point Sets” in IEEE Transactions on Pattern Analysis and Machine Intelligence, PAMI-9, Issue:5, Sep.1987 [36] Louis Wiesmann Tiziano Guadagnino Ignacio Vizzo Giorgio Grisetti Jens Behley Cyrill Stachniss, “DCPCR: Deep Compressed Point Cloud Registration in Large-Scale Outdoor Environments” IEEE Robotics and Automation Letters 7 (3), 6327-6334

Ⅰ．実証概要

Ⅱ．実証技術の概要

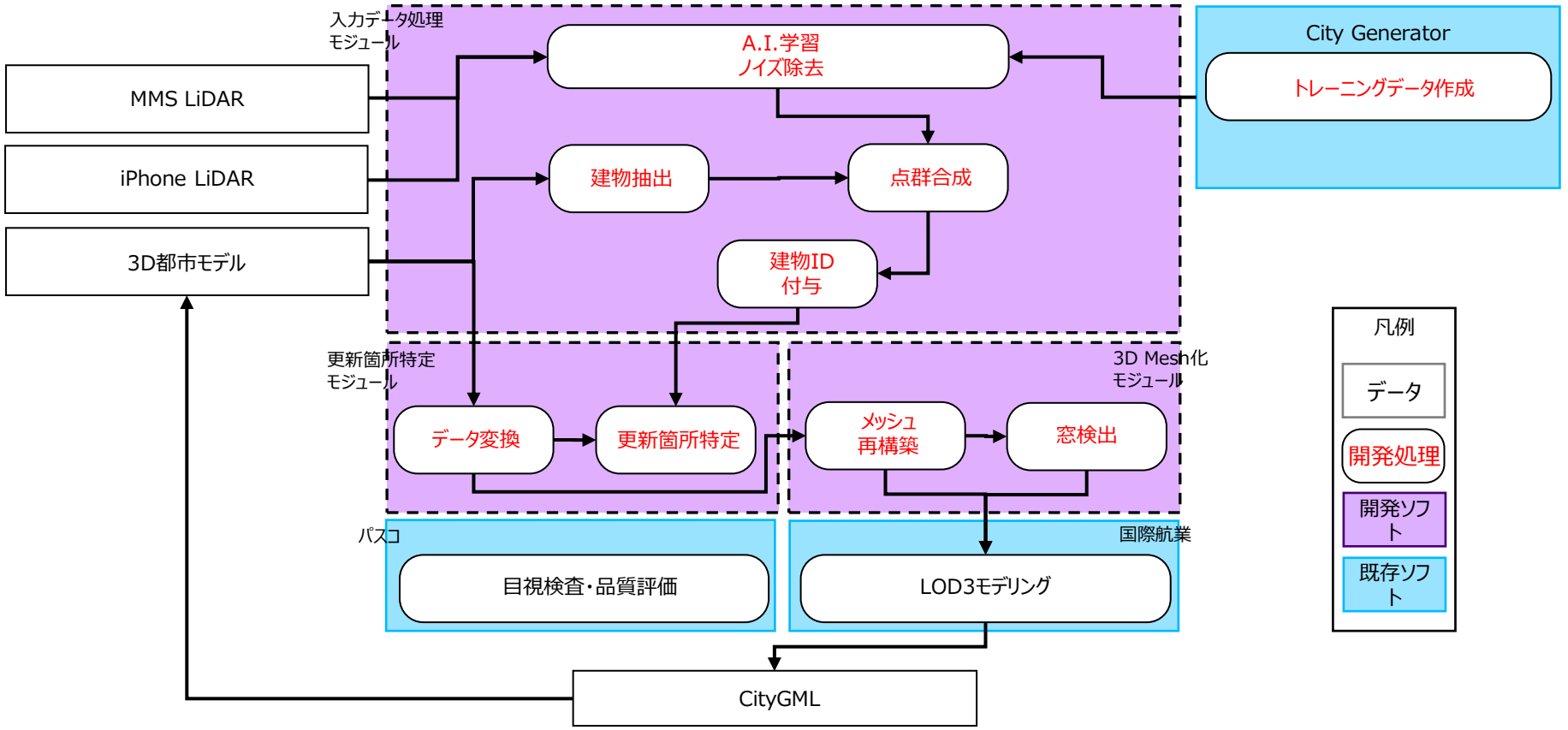
Ⅲ．技術調査

Ⅳ．実証システム

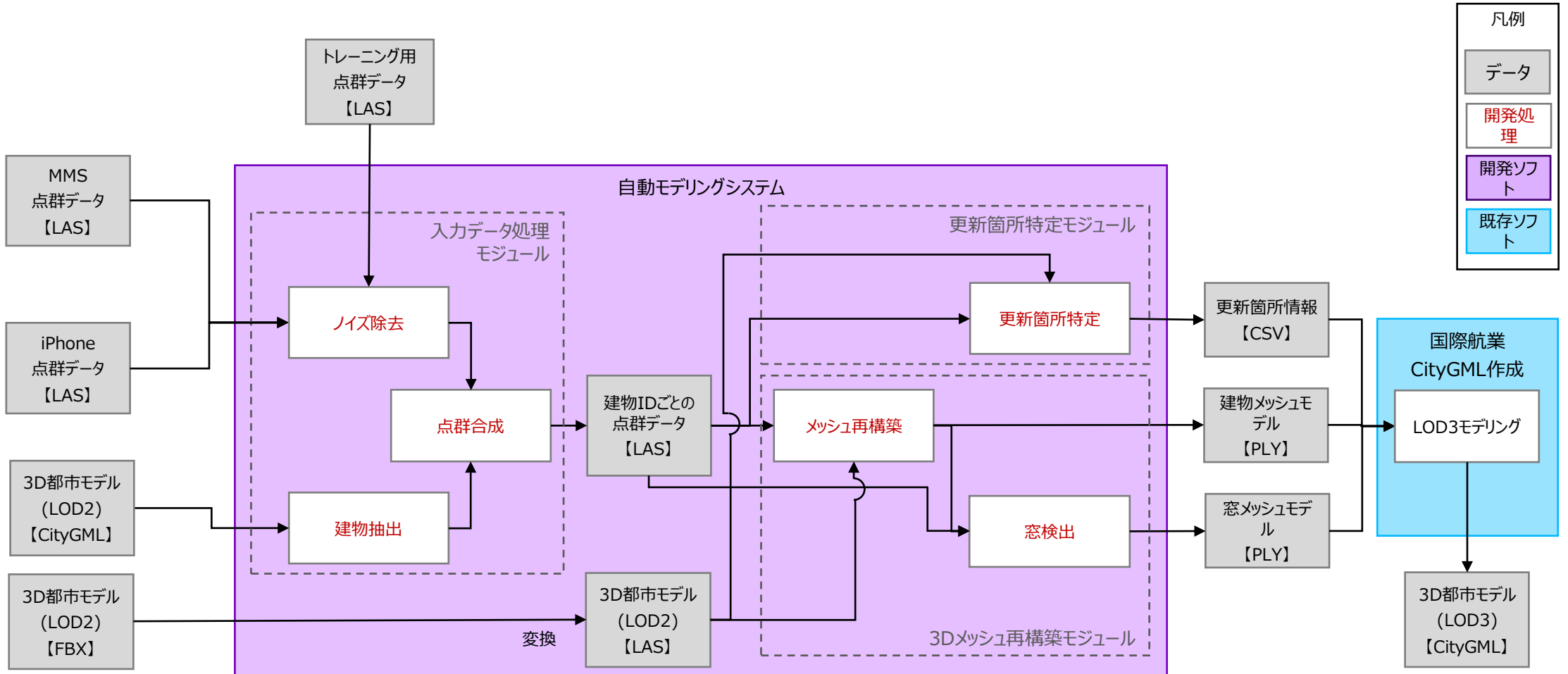
Ⅴ．実証システムの検証

Ⅵ．成果と課題

システムアーキテクチャ図



データアーキテクチャ図



システム機能

本調査研究で実証した機能を下表に示す

＜凡例＞**赤太字：新規開発要素**

#	機能名	説明
1	**トレーニングデータ作成**	City Generatorを用いて、ノイズリダクションのセマンティックセグメンテーションで用いるトレーニング用点群データを作成する。
2	**ノイズ除去**	トレーニング用点群データを用いてセマンティックセグメンテーションを行い、点群データから植生・移動物体・路側構造物・その他センサーに起因するノイズの除去を行う。
3	**建物抽出**	3D都市モデル(LOD2)の2Dフットプリントポリゴンを生成する。
4	**点群合成**	複数の点群データの位置合わせおよび合成処理を行い、建物ID単位の点群データを出力する。
5	**更新箇所特定**	点群データと過年度3D都市モデルを比較し、建物更新箇所の検出を行う。
6	**メッシュ再構築**	点群データからメッシュを再構築する。必要に応じて、点群データの欠損部を3D都市モデルLOD2で補間する。
7	**窓検出**	点群データおよびメッシュデータから窓の位置を検出する。
8	CityGML作成	自動モデリングシステムから出力されたデータを用いて、CityGMLの作成を行う。

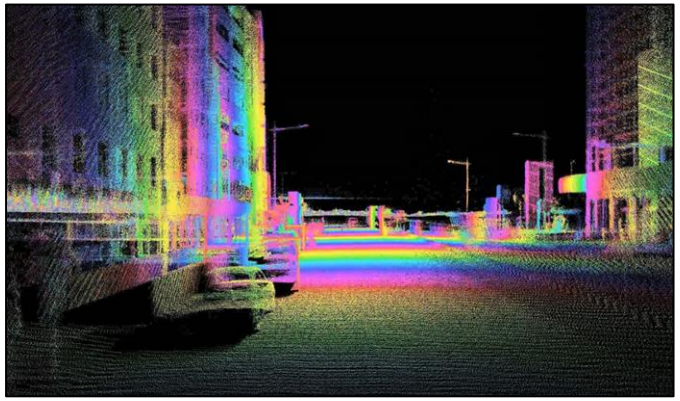
測量成果

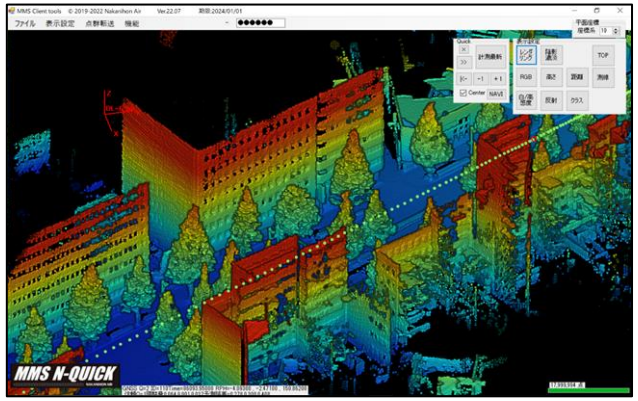
本調査研究で取得した測量成果を下表に示す

測量成果 整備対象	一般車両 簡易MMS計測 （点群データ/参考カメラ） 撮影：2022/6/19-6/21 点密度：200-400点/m2 作業機関：中日本航空	一般車両 簡易MMS計測 （点群データ/参考カメラ） 撮影：2022/8/17-8/19 点密度：200-400点/m2 作業機関：中日本航空	路線バス 簡易MMS計測 （点群データ/参考カメラ） 撮影：2022/9/21-10/5 点密度：200-400点/m2 作業機関：中日本航空
測定範囲	仙台駅周辺道路	仙台駅周辺道路	宮城交通走行路線
地図情報レベル	レベル2500相当※	レベル1000相当　※	レベル1000相当　※

※地図情報レベル
簡易MMSは現時点では公共測量作業規程に準拠した手法ではないため、レベルに相当と記す。自社検証においてはGNSS受信環境が良い場所（RTK-GNSSがFIX解、もしくは30秒以内にFIX解に戻るFloat解）ではレベル500を満足する結果を得ている。本調査研究の参考として記す。

※仙台市駅周辺の測位環境
仙台市駅周辺はビルに囲まれた都心部であり、ガラス張りの高層ビル（マルチパス）、細い路地（アーバンキャニオン）、街路樹の立ち並ぶ道路など、GNSS測位環境が悪い状況であった。そのため、一部の点群では測位精度が悪い点群も含まれているが、本調査検討では、後続作業（点群マッチング、クリーニング処理）で点群間の相対位置の整合性を高めている。





点群データ

本調査研究で取得した点群データを下表に示す

測定エリア	iPhone LiDAR計測①	iPhone LiDAR計測②	使用ソフトウェア
仙台駅周辺	撮影：2022/10/21-10/22 使用機材：iPhone12 Pro 点密度：非公開 作業機関：Symmetry Dimensions	撮影：2022/10/21-10/22 使用機材：iPhone14 Pro 点密度：非公開 作業機関：Symmetry Dimensions	• SiteScape • 独自計測iOSアプリケーション(Symmetry Dimensions開発)

※iPhone LiDARの性能
iPhone12 Pro及びiPhone 14 Proに搭載されているLiDAR性能は非公開であり、ソフトウェア側の設定により可変となっている。本調査研究の参考として記す。
※仙台市駅周辺の測位エリア
簡易MMSを設置した、バス、タクシーを想定した一般車両で撮影が困難で合ったら、車両通行不可能な細い路地、奥まった入口を持つ地物等を優先的に撮影を行った。

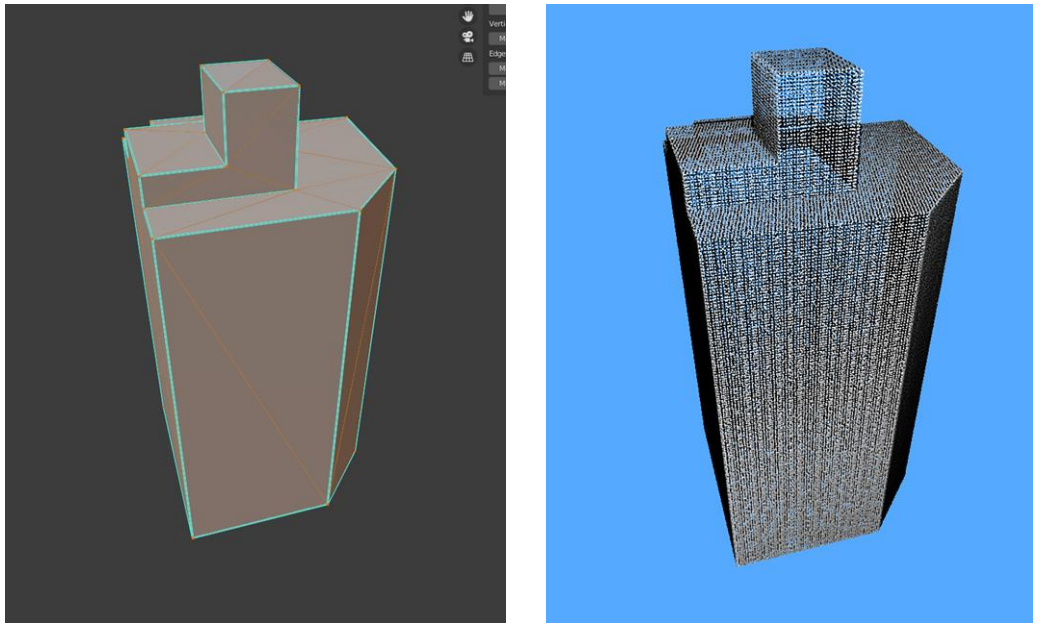
データ処理一覧

本調査研究で行ったデータ処理を下表に示す

システムに入力するデータ（データ形式）	用途	処理内容	データ処理ソフトウェア	活用データ（データ形式）
建物モデル（LAS形式）	変更箇所特定およびメッシュ再構築における点群データ補間	仙台市緊急整備地域のFBX形式の3D都市モデル(LOD2)から、各建物毎のLASファイルに変換	Blender CloudCompare	3D都市モデル（FBX形式）

建物モデル（LAS形式）

本調査研究で行ったデータ処理を下表に示す

オペレーションのフロー		イメージ
FBX→OBJ変換	仙台市緊急整備地域の3D都市モデル（FBX形式）データをBlenderに読み込み、建物毎の個別のOBJファイルに分割して出力する。	
OBJ→LAS変換	建物毎のOBJファイルをCloud Compareに読み込み、LAS形式に変換して出力する。	

FBX形式の建物をLAS形式に変換した例

トレーニングデータ作成

City Generatorを用いてオブジェクトをランダムに配置して仮想の都市を作成。仮想の都市の中でMMS・iPhoneをシミュレートして、ラベリングされた点群データの撮影を行い、大量のトレーニング用データを作成する。

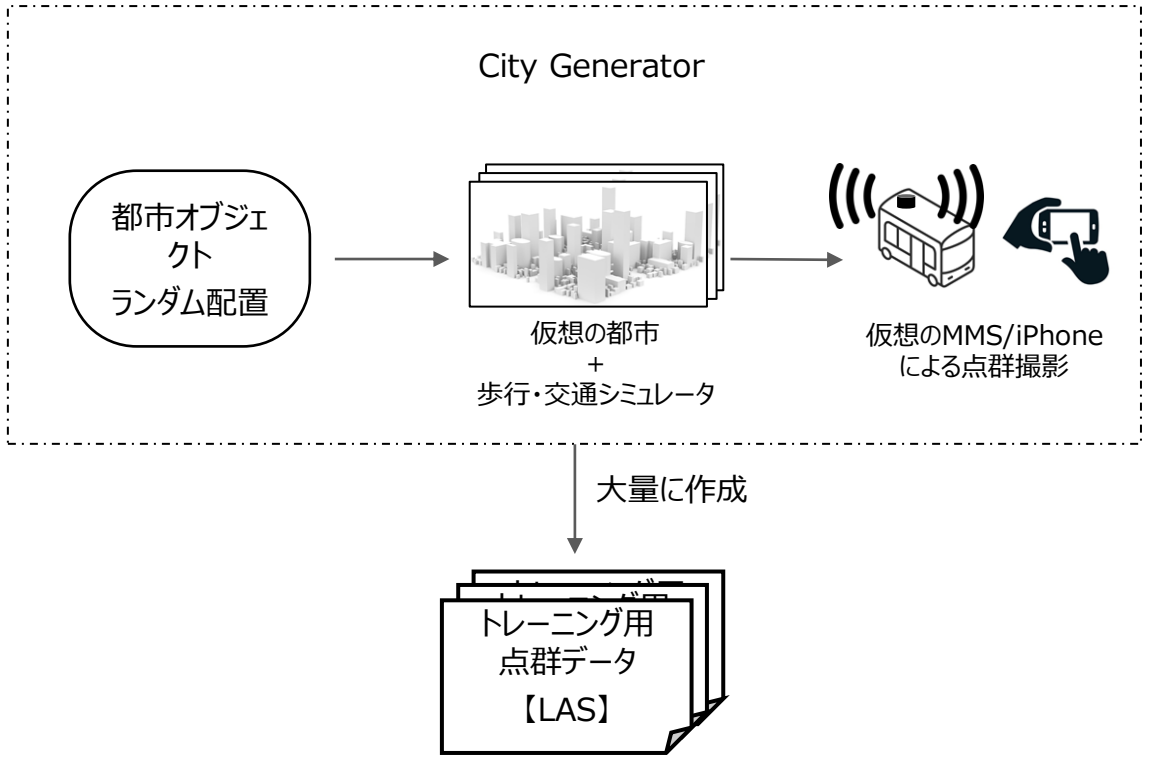
操作／処理フロー

ステップ①
City Generatorで都市の地物オブジェクトをランダムに配置し、仮想の都市を作成する。

ステップ②
仮想の都市の中で移動物シミュレータ（歩行者・自転車・車）を実行する。

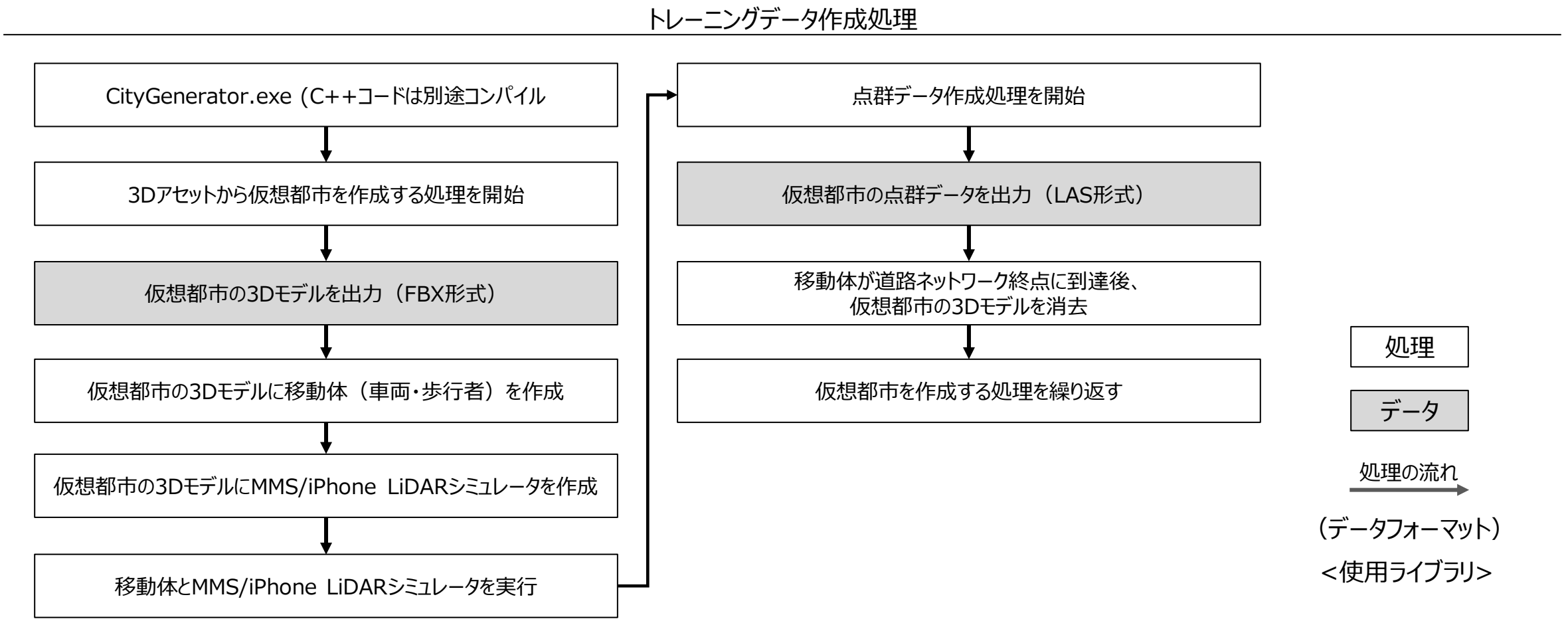
ステップ③
MMS、iPhoneのLiDARをシミュレートし、仮想の都市の中で点群データを撮影し、ラベリング済のトレーニング用点群データを大量に作成する。

システム図



トレーニングデータ作成｜処理フロー

学習用トレーニングデータ作成方法は以下の通り



トレーニングデータ作成｜概要

仮想都市の自動作成

セグメンテーションモデルに必要な十分な多様性を持つ大量のトレーニングデータを作成するため、完全にランダムな仮想の都市環境を作成可能なCity Generatorを活用した。それぞれのシーンには、さまざまな建物や都市のオブジェクトの組み合わせが含まれるように自動で作成され、車両は道路沿いに決められたパスに従う移動、歩行者のシーン内ランダム移動が可能である。

作成した仮想世界と歩行者・車両

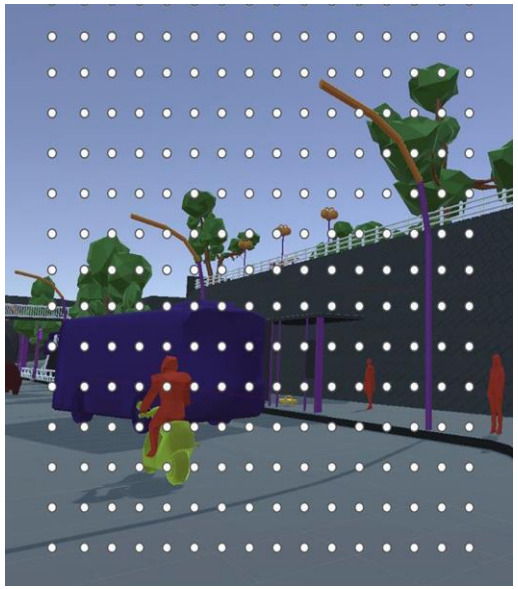
トレーニングデータ作成｜概要

iPhone LiDARのシミュレーション

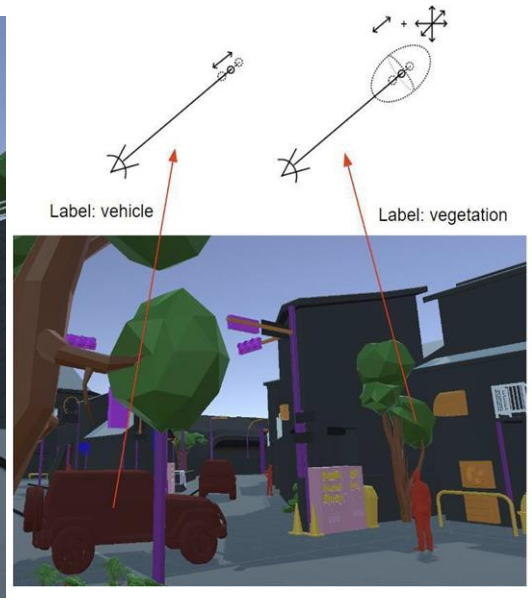
現実の世界で行うiPhone LiDARスキャナによる点群計測を模倣するため、仮想の都市環境の中で、iPhone LiDARスキャナを持たせた歩行者のシミュレートを実装した。

現実の世界でiPhone LiDARスキャナを使用して点群計測を行った場合の移動体のノイズや点密度、視野範囲、約6メートルのスキャン距離、推定ノイズプロファイル等をシミュレートすることで、実際に撮影されたデータに近いトレーニングデータを作成することが可能になった。

また、植生など、 不確かな表面を持つ地物に対する振る舞いをより現実的にするために、仮想LiDARビームが空間内で命中した地物のターゲットラベルに応じて追加ノイズを加えた。窓に仮想LiDARビームが命中した場合は、垂直に近いLiDARビームのみ反射光を受光でき、垂直から遠ざかると反射光の受光確率が急速にゼロに向かって減少する。これは現実の世界での振る舞いをシミュレートしている。



仮想iPhone LiDARレイキャストグリッド

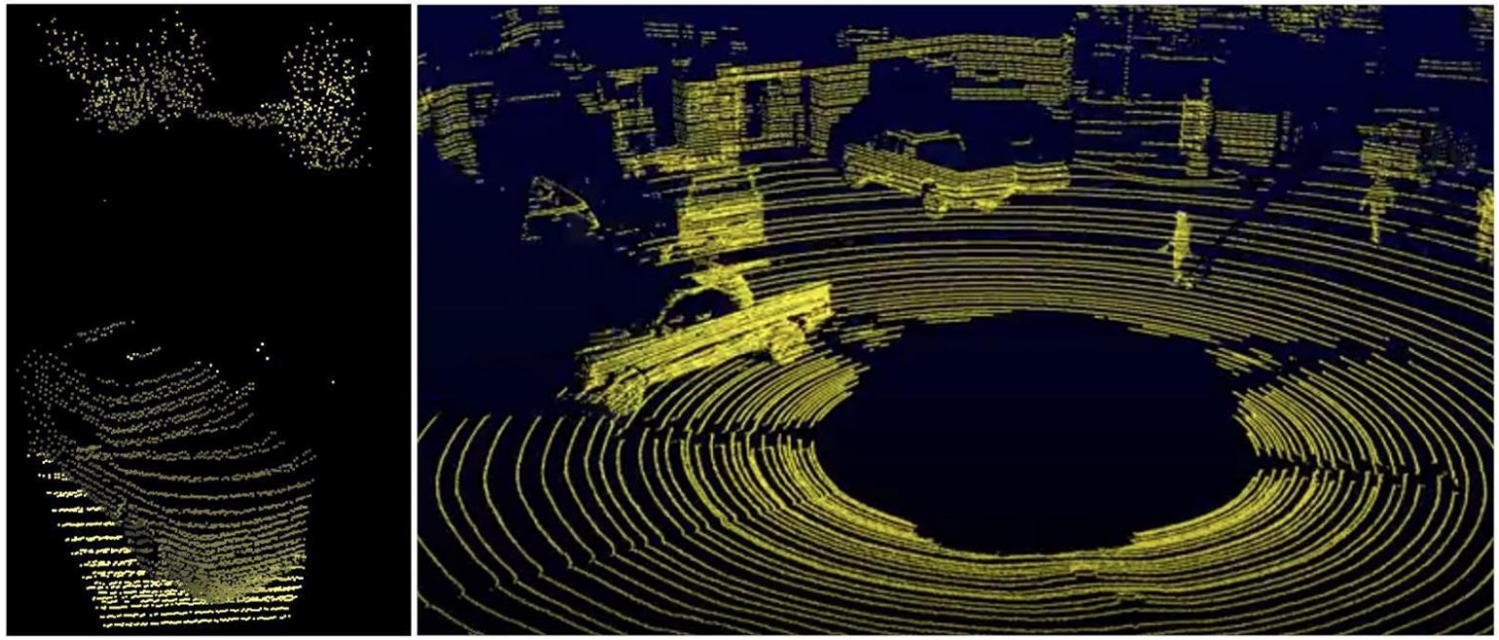


ビームが当たる物体の種類に応じて変化するノイズを付加

トレーニングデータ作成｜概要

MMSのシミュレーション

iPhone LiDARスキャナと簡易MMSでは性能に大きな差異があり、本実証のセグメンテーションモデルのためのトレーニングデータに適さないため、iPhone LiDARスキャナのシミュレートと併せて、本実証で使用する簡易MMSに実装されているセンサー「Velodyne VLP-16 LiDAR Puck」の持つ360度の視野角と約100メートルのスキャン範囲の性能をシミュレートするためのシステムを実装した。



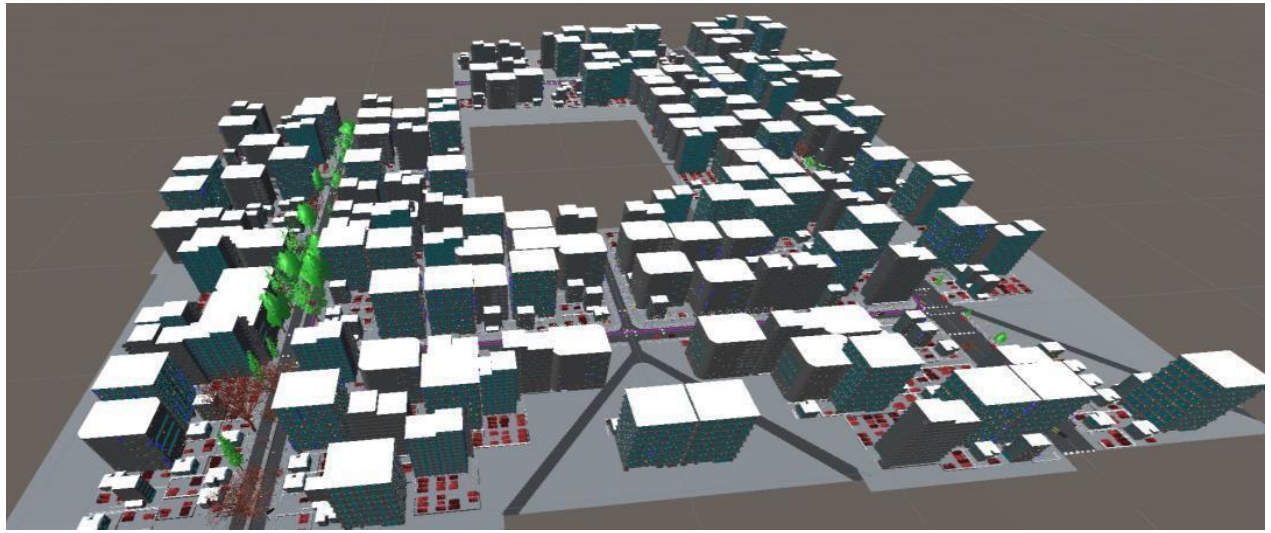
iPhone用LiDARと長距離360度LiDARの比較（右画像出典：Velodyne）

トレーニングデータ作成｜概要

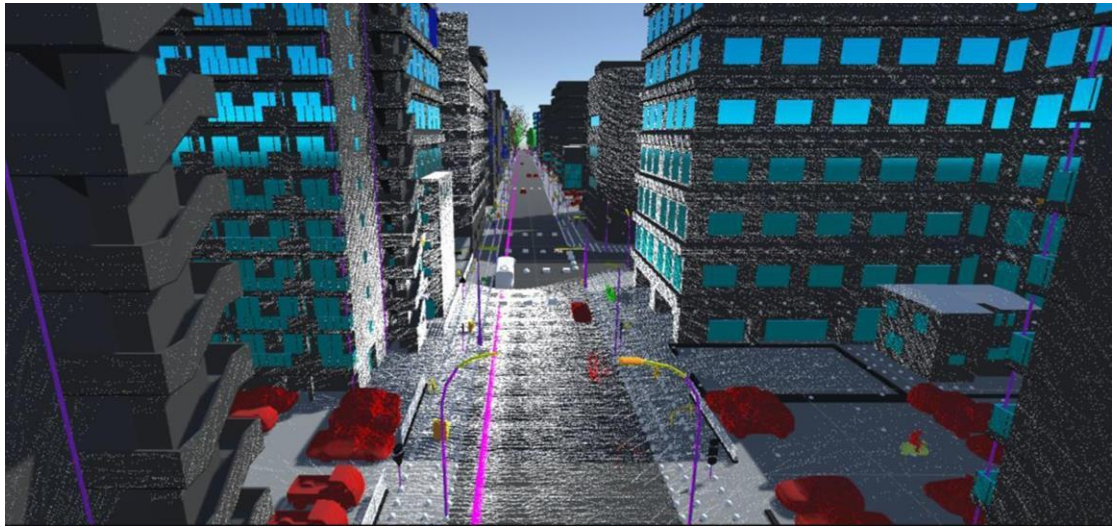
City Generatorの実行

City Generatorは、道路の左右100メートルの土地を持つランダムな道路ネットワークを作成し、3D建物やその他のオブジェクトの3Dモデルアセットからランダムに配置を行い、道路上の交通、歩道の歩行者をシミュレートする。

「Velodyne VLP-16 LiDAR Puck」をシミュレートする仮想簡易MMSを、シーン内の仮想車両の屋根に45度の角度で取り付け、走行しながら都市の景観をスキャンする。車両が道路ネットワークの終わりに達すると、シーンは破棄され、完全に新しいランダムな都市の景観が生成され、そのプロセスが繰り返される。



インルートの長さが約1kmの街並みを生成したもの



仮想車載LiDARでスキャンした点群
（紫色の線はセンサーを取り付けた車のルート）

ノイズ除去（領域分類A.I.）

トレーニング用点群データを用いてセマンティックセグメンテーションを行い、点群データから植生・移動物体・路側構造物・その他センサーに起因するノイズの除去を行う。

操作／処理フロー

ステップ①

City Generatorで生成されたトレーニング用点群データを用いて、RandLA-Netを訓練する。

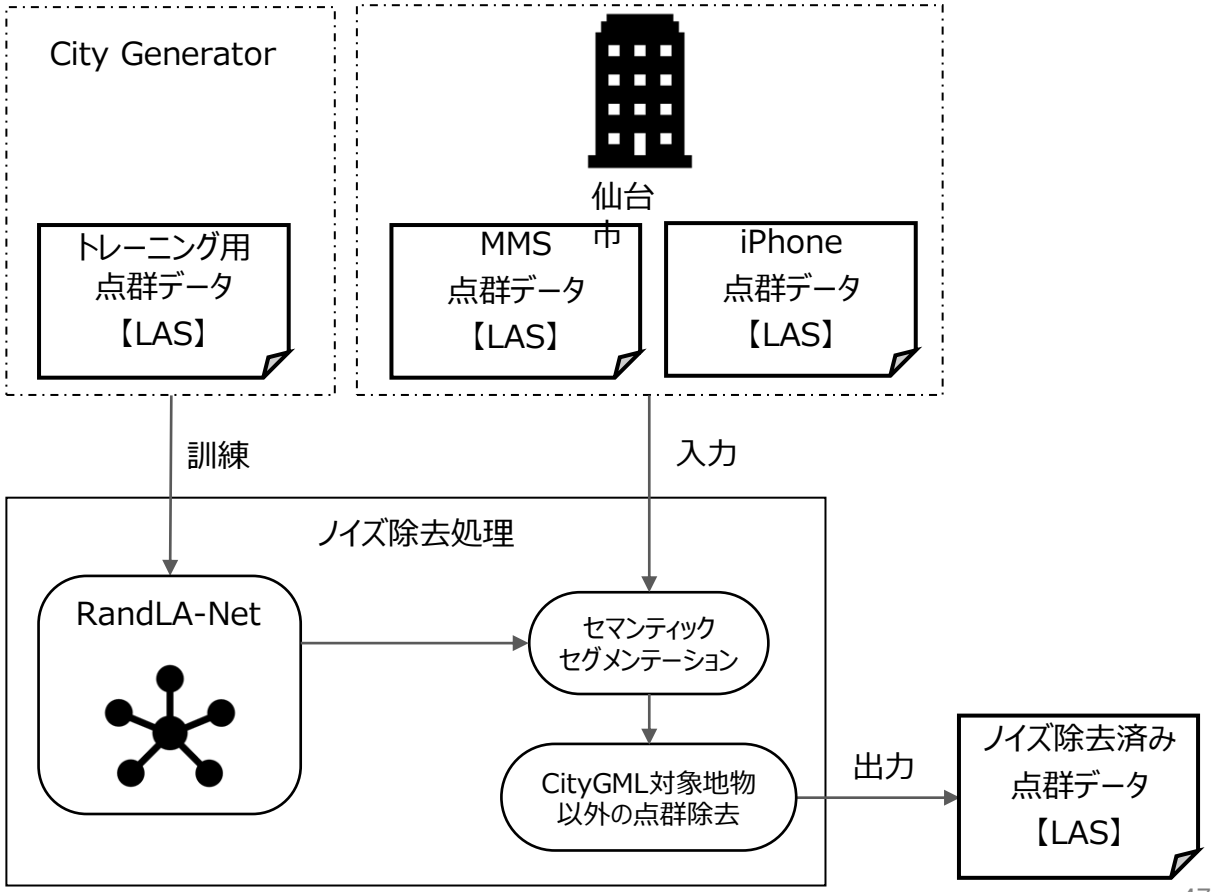
ステップ②

MMS点群データおよびiPhone点群データをRandLA-Netに入力してセマンティックセグメンテーション処理行い、地物を分類する。

ステップ③

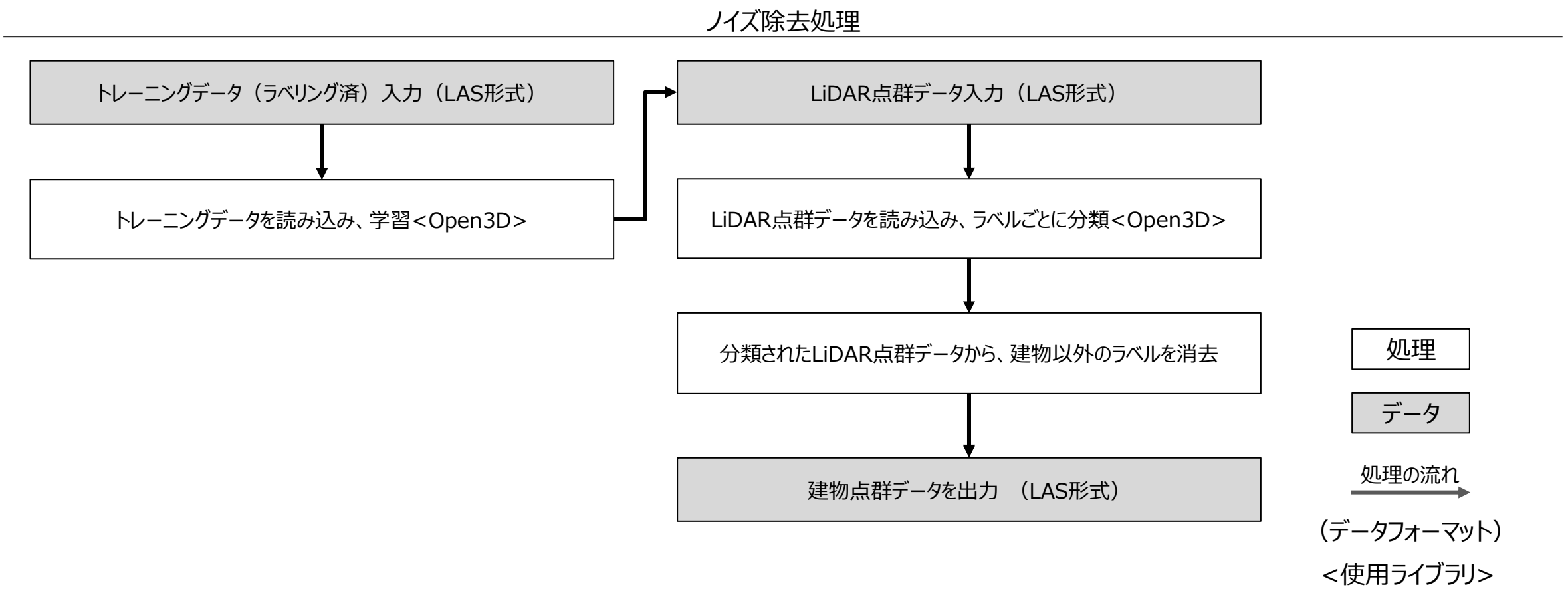
点群データからCityGML対象地物以外に分類された点群を削除する。

システム連携図



ノイズ除去（領域分類A.I.）｜処理フロー

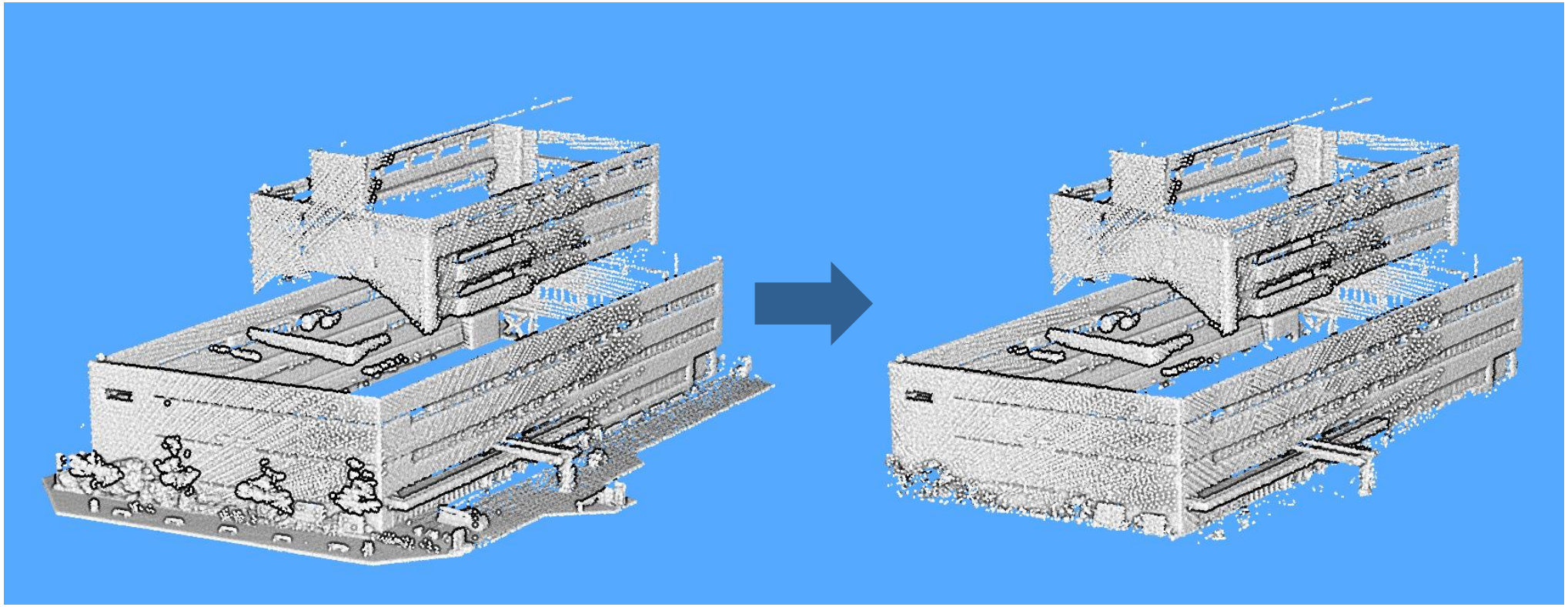
点群データのノイズ除去方法は以下の通り



ノイズ除去（領域分類A.I.）｜処理フロー

点群データのノイズ除去方法のイメージは以下の通り

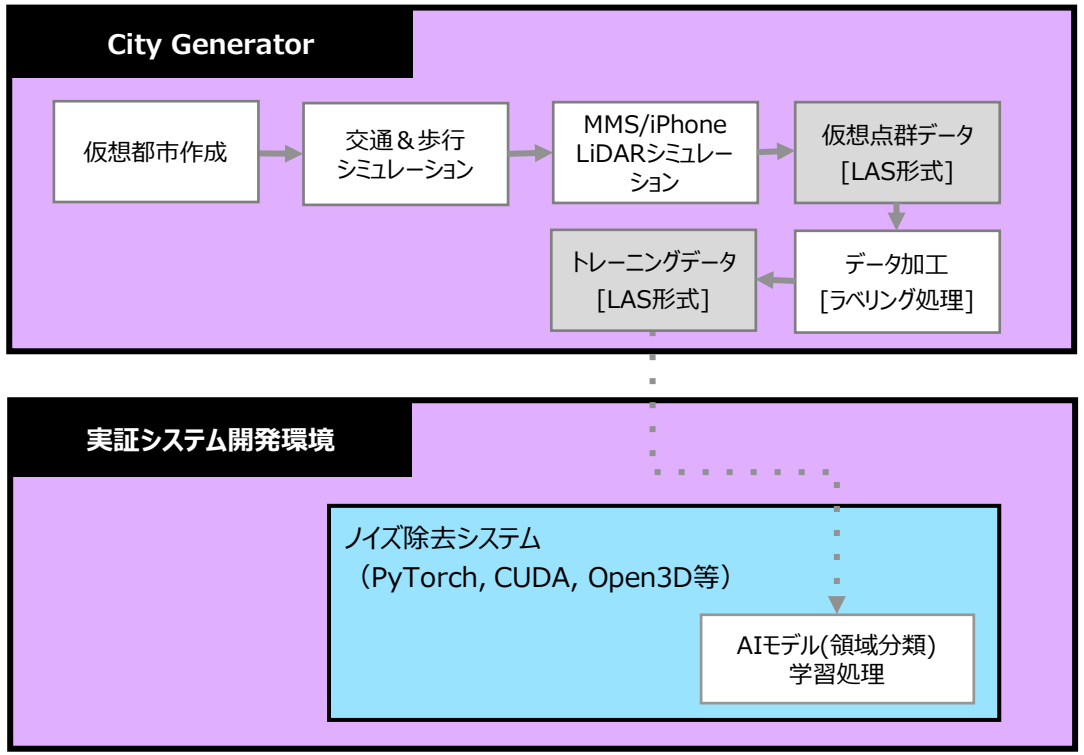
ノイズ除去処理イメージ



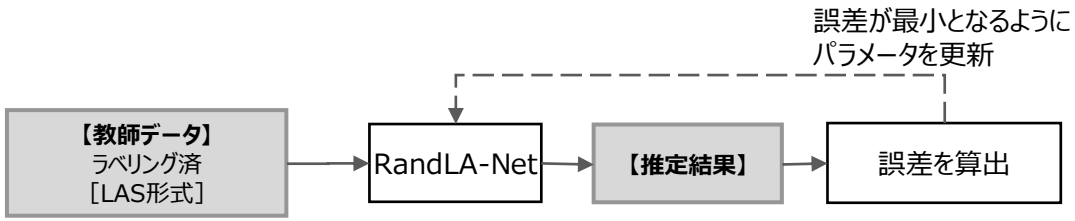
セマンティックセグメンテーションによりLOD3モデル化に必要な地物以外を消去している

ノイズ除去（領域分類A.I.）｜学習方法

点群データから分類検出された結果が、トレーニングデータに近付くよう学習



仮想都市で作成された点群データにラベリングを行い、教師データとして入力し、A.I.モデル（RandLA-Net）推定結果と教師データの誤差が最小になるようにモデルのパラメータを更新 → 分類推定を教師データに近づけることが目的



ノイズ除去（領域分類A.I.）｜使用方法

点群データのノイズ除去を行うためのスクリプトの設定と使い方

実行コマンドに関する仕様

項目	内容
実行するコマンド名	python segmentation_pipeline_sendai.py
コンフィギュレーション	スクリプトはPythonファイルで直接設定します
結果	LAS形式の点群、設定によりセグメント化、フィルタリングされた点群

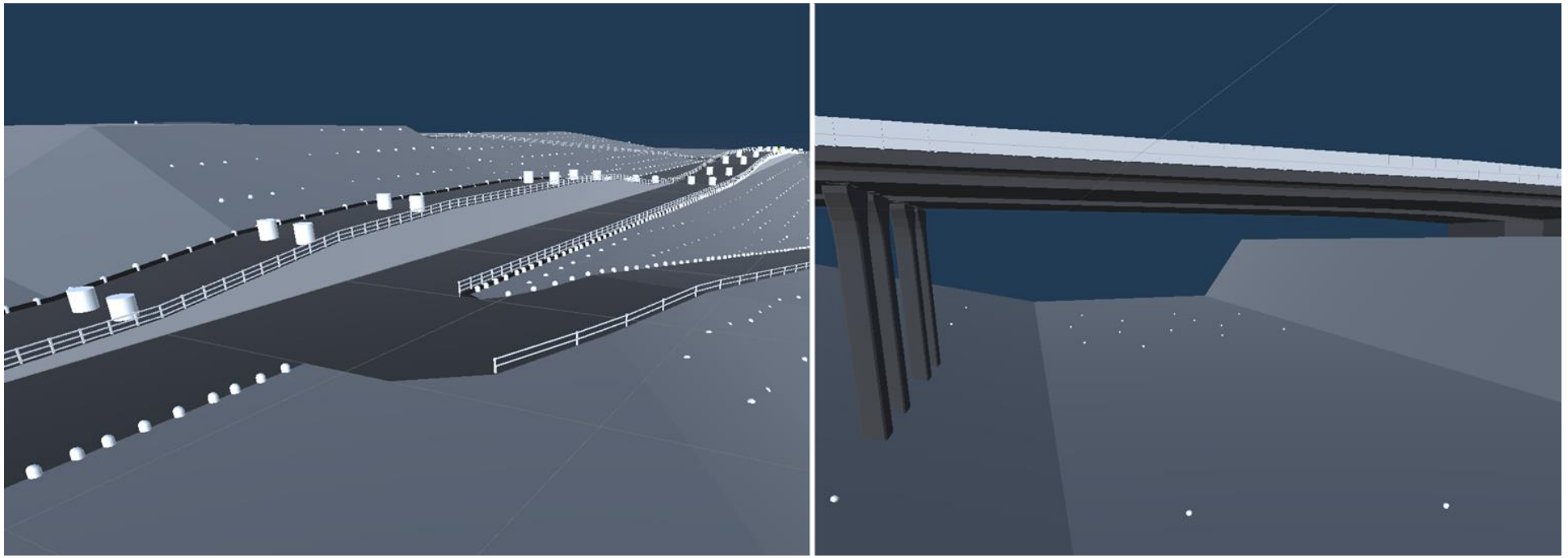
コンフィグレーション

キー	内容
chosen_dataset	セグメンテーションモデルのためのデータパーサーと前処理ルーチンをセットアップ。初期値は、*'symmetry_testset'*を使用。
cfg_folder	モデル設定.ymlファイルが置かれているフォルダ
cfg_name	モデル設定.ymlファイルの名前
ckpt_name	ロードするモデルチェックポイントファイルの名前。初期値は、'ckpt_00029.pth'が使用される
input_dataset_path	セグメント化するLAS点群のあるソースディレクトリ
output_dataset_path	セグメンテーション結果が格納される出力ディレクトリ
crop_size	セグメンテーションで切り取る入力点群サイズ。crop_size が大きいと処理速度が遅くなる。初期値は「250000」。
output_only_buildings	「True」設定時、出力点群に「Building」ラベルを持つ点のみが含まれる
maximum_output	「True」設定時、出力点群に、LASの各種データフィールドに格納されたラベル確率情報が含まれる

ノイズ除去（領域分類A.I.）｜問題点①

LOD2モデルの2Dフットプリントポリゴンを拡張するためのスクリプトの設定と使い方

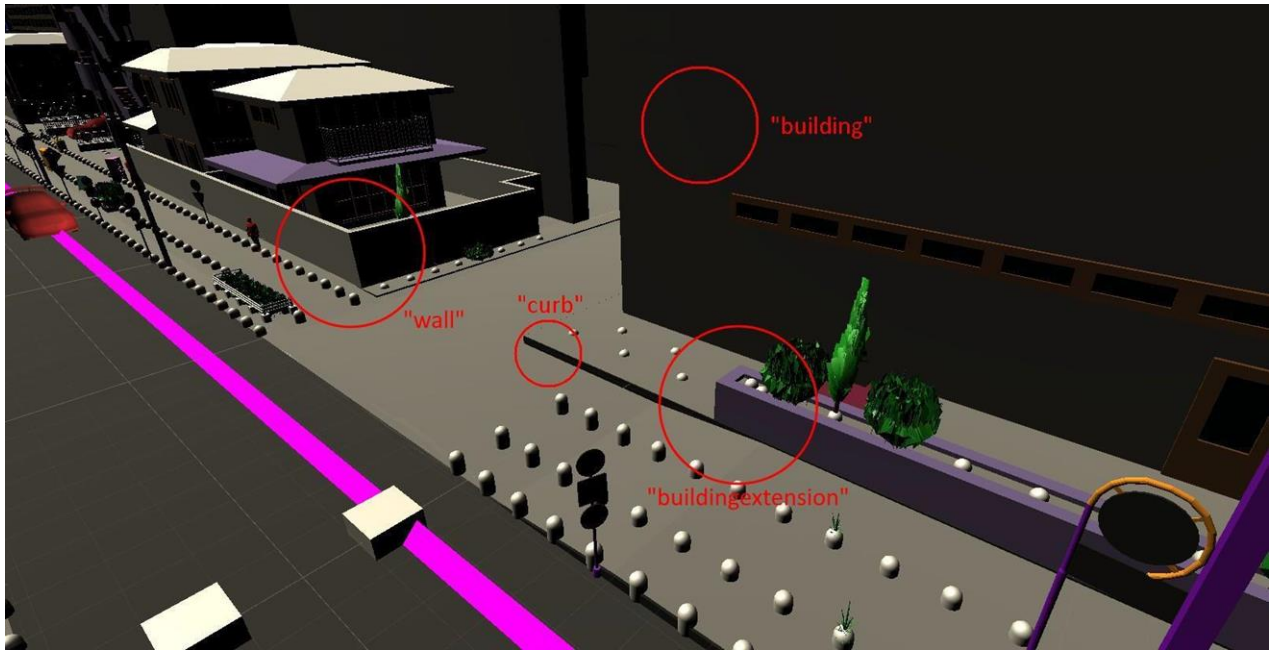
新しい地物・道路を追加したときに、性能低下が現れ始めた。仙台市街は水平ではない地形が含まれているため、勾配や橋のある地形の道路や地面のモデルを追加したが、これにより建物の接地面付近でモデルの性能低下が生じた。勾配や橋を含む地形を扱うのはより困難であることが判明した。



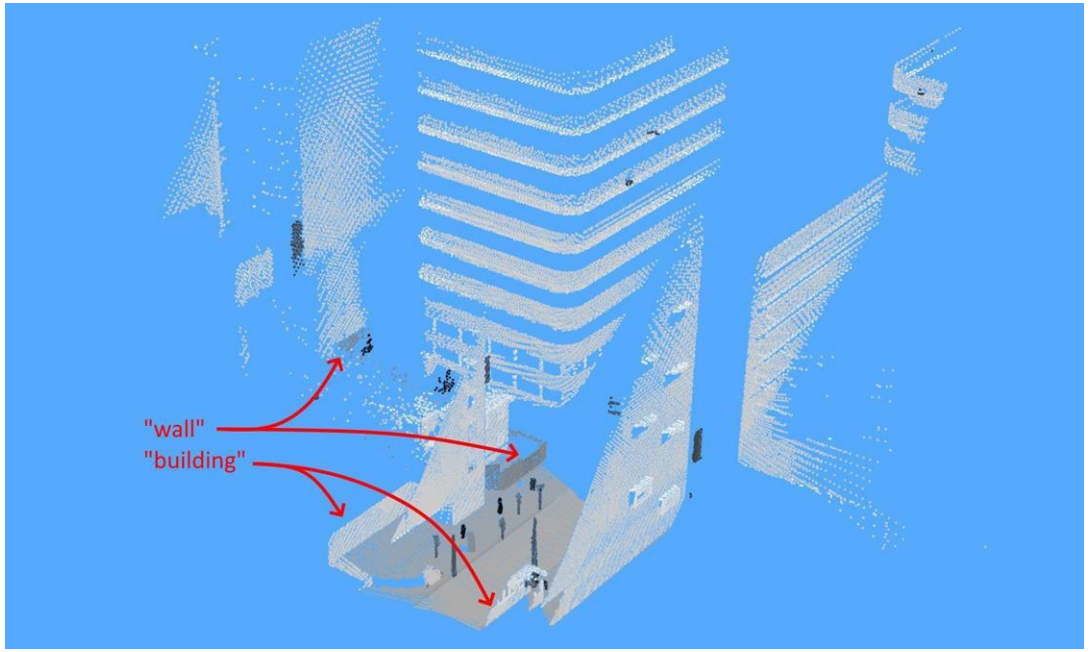
様々な勾配の道路や橋を含む仮想環境

ノイズ除去（領域分類A.I.）｜問題点②

トレーニングデータの点に付与されるラベルのカテゴリには、「building」、「wall」、「curb」、「buildingextension」などの垂直平面によって構成される複数のラベルカテゴリがある。これらのカテゴリは、主に高さ（場合によっては位置や他の地物との関係）によって区別されるため、適切なセグメンテーションが不可能な場合がある。具体的には矩形に切り取ったセグメンテーション入力点群数が10万から50万点時、入力点群の境界上で建造物の一部が薄い壁のように部分的にしか見えない場合に顕著であった。このような場合、一部しか見えていない建造物は「building」と「wall」の間での誤検出を引き起こす結果となった。



垂直な平面を持つラベルカテゴリ

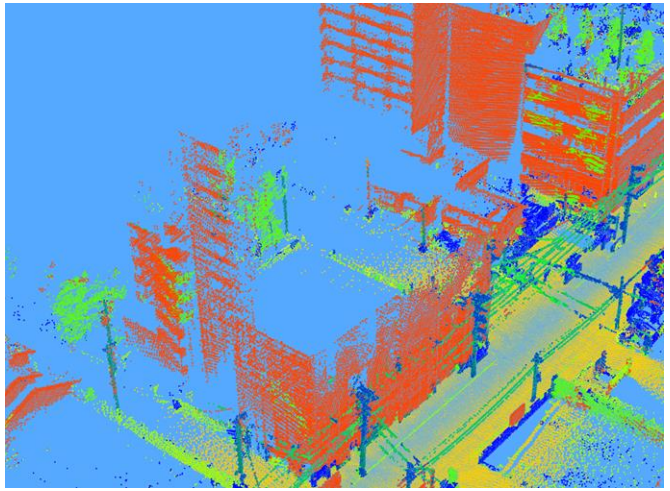


垂直な平面を持つカテゴリを見分けるのが困難なケース

ノイズ除去（領域分類A.I.）｜問題点③

勾配や橋を含む地形の追加によってこの現象は悪化し、3つ目の競合カテゴリである「ground」とも混同するようになった。モデルは建造物の最下部1mを「ground」と判断し、建造物の1階以上の高さでは本来の「building」というラベルに収束した。このため一つの建物に複数のラベルが混在するようになった。

この問題は、最新のセグメンテーションモデルでは、高さとモデルによって出力される基礎ラベルの確率に基づいて複数のラベルを統合することによって一部回避されたが、セグメンテーションされた建物の底部が最適に捉えられていない結果となった。この課題を解決するため、City Generator自体の修正も行い、平坦でない地形の除去、テストエリアの建物周辺に見つかる形状に重点を置いてトレーニングなどを行った。しかし、1回のイテレーションには、新しいトレーニングデータの準備、モデルのトレーニング、および実際のデータのセグメンテーション等の作業が発生し、この作業に最大2週間を要するため、特徴的な地形の性能改善のためのイテレーション高速化には課題が残されている。



建物抽出

3D都市モデル(LOD2)の建物外形による2Dフットプリントポリゴンの作成及び拡張

操作／処理フロー

ステップ①

実証実験エリアの最新の3D都市モデル(LOD2)から水平方向の2Dフットプリントポリゴンを取得する。

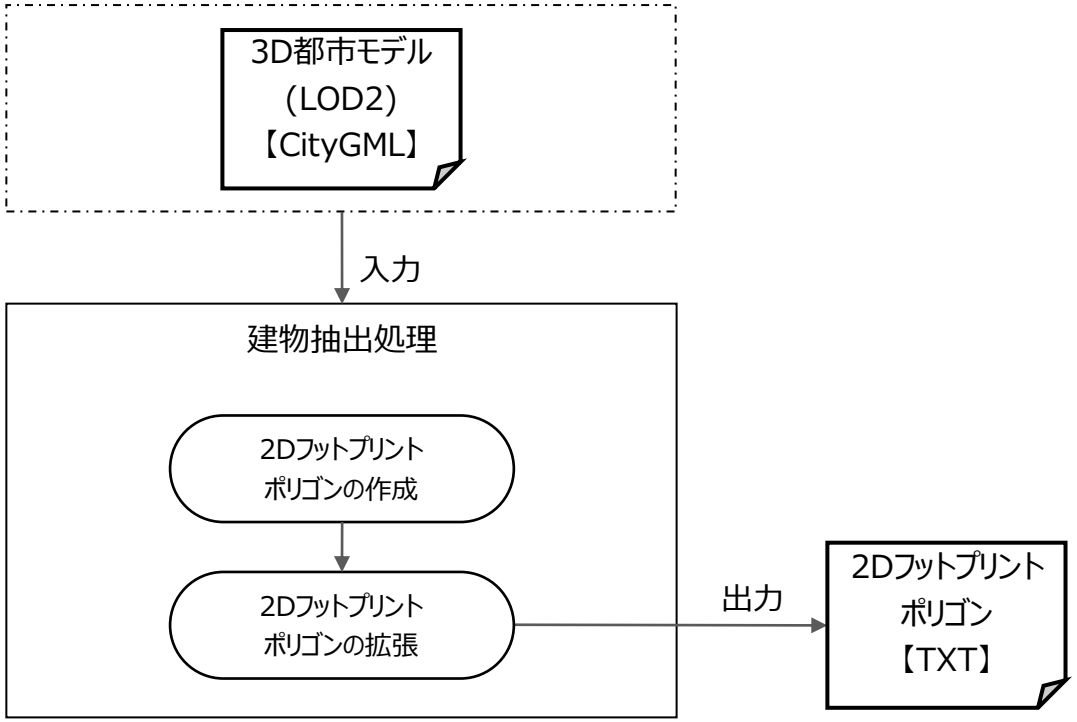
ステップ②

2Dフットプリントポリゴンを拡張する。その際、隣接する建物境界に被らない様に、自動調整を行う。

ステップ③

拡張した2Dフットプリントポリゴンをテキスト形式で出力する。

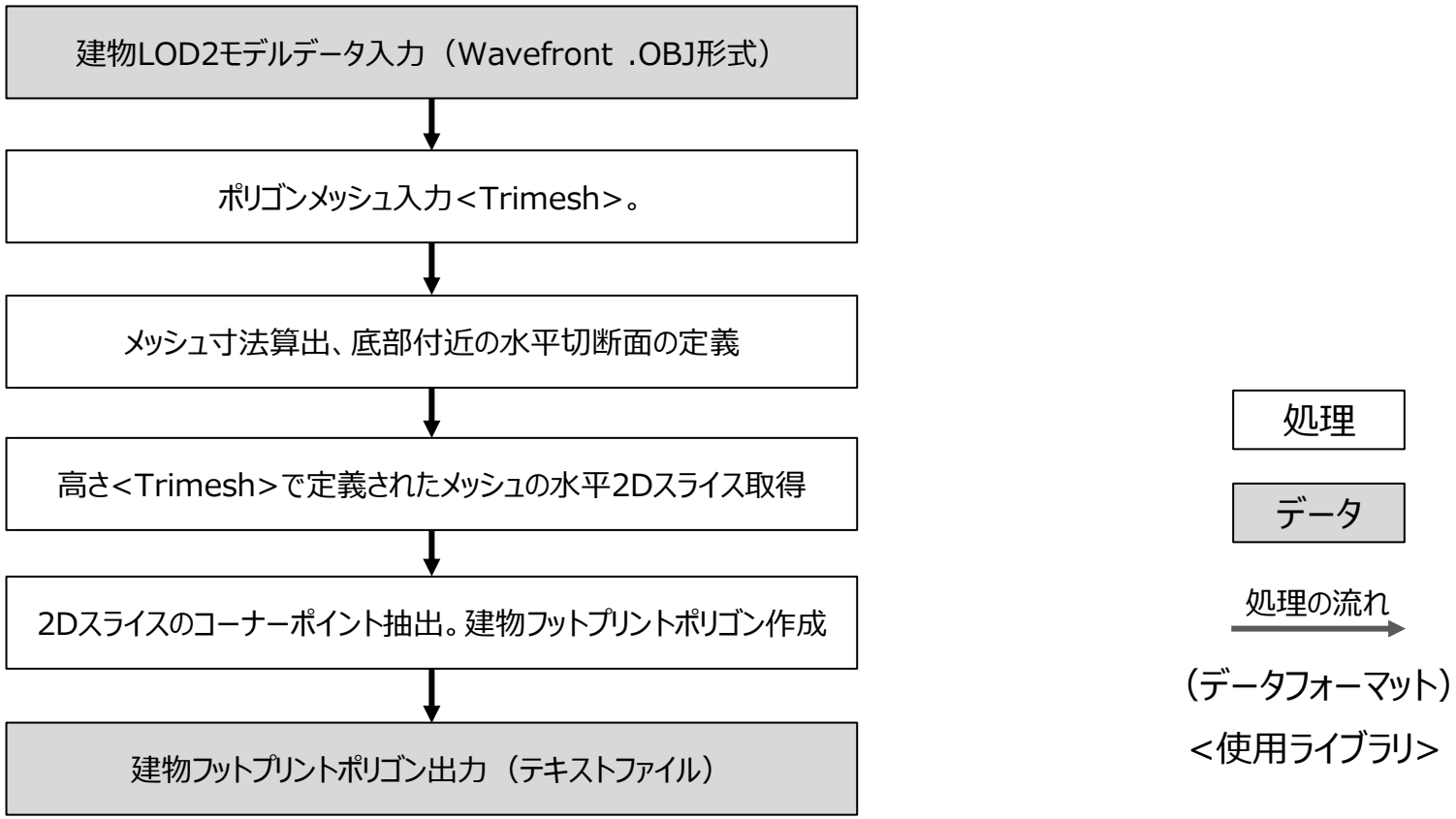
システム連携図



2Dフットプリントの作成｜処理フロー

LOD2モデルの2Dフットプリントポリゴンの作成方法は以下の通り

2Dフットプリントポリゴン作成処理



2Dフットプリントの作成｜使用方法

LOD2モデルの2Dフットプリントポリゴンを作成するためのスクリプトの設定と使い方

実行コマンドに関する仕様

項目	内容
実行するコマンド名	python 01_create_footprint_polygons.py
コンフィギュレーション	スクリプトはPythonファイルで設定
結果	各行に建物LOD2オブジェクトの名前と、2Dフットプリントポリゴンを定義するX/Y座標ペアのセットが含まれるテキストファイルを出力

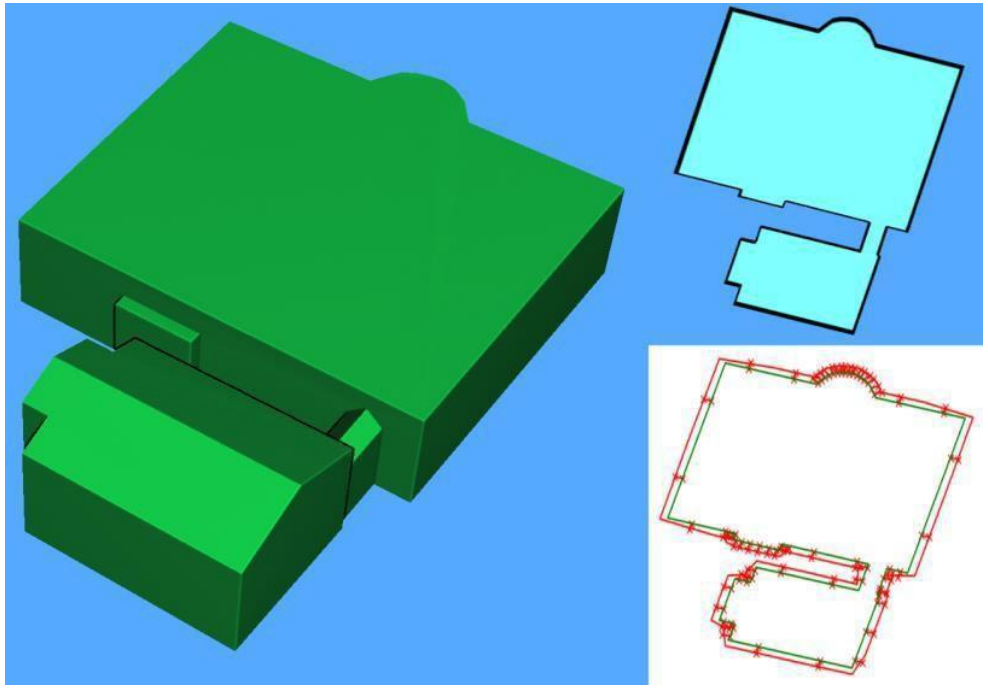
コンフィグレーション

キー	内容
input_path	Wavefrontの.obj形式のターゲット建物LOD2モデルを含むディレクトリ
output_path	結果のテキストファイルの出力先
dilation_amount	ポリゴンは作成時に直接拡張可能だが近傍建物は考慮されない。0.0に設定しておき、フットプリントのポリゴンサイズを大きくするには別のスクリプトを使用
visualize_results	ポリゴン画像の出力時 'True'、それ以外は 'False' に設定

2Dフットプリントの作成｜概要

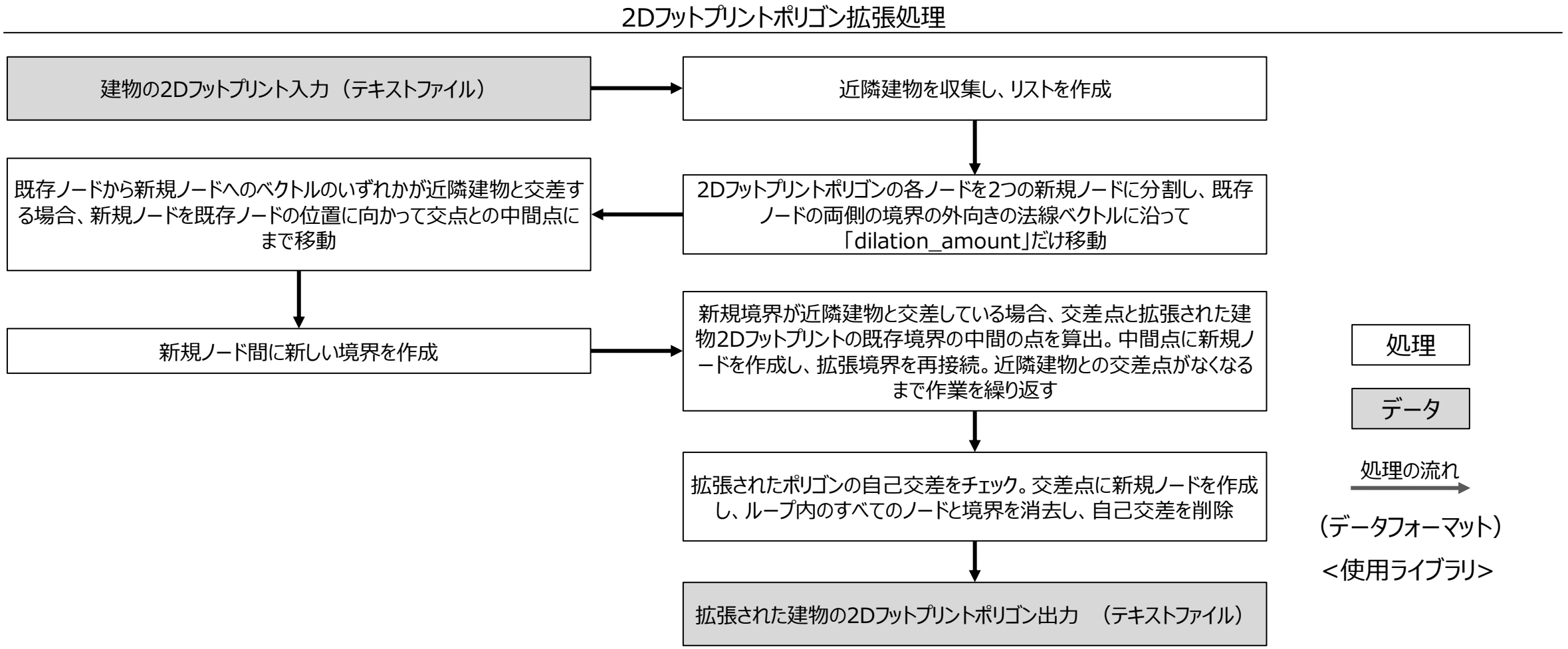
簡易MMS及びiPhone LiDARで取得された点群データには、広範囲に複数の建物が含まれるため、ターゲットとなる建物を個別に抽出する方法が必要となる。本実証では、既存の3D都市モデル（LOD2モデル）を利用した。

LOD2モデルは、窓や廊下、軒などの詳細な形状のない3D建造物モデルであり、上から見た場合には地面に投影された建物のフットプリントを得ることができる。LOD2モデルを接地面よりも僅かに上の高さで水平にスライスすると、建造物のフットプリントを表現する2Dポリゴンが得られる。これらのポリゴンをLiDARデータセットから建造物を抽出するために使用した。



LOD2モデルを取得し水平2Dスライスを生成

2Dフットプリントの拡張｜処理フロー



2Dフットプリントの拡張｜使用方法

LOD2モデルの2Dフットプリントポリゴンを拡張するためのスクリプトの設定と使い方

実行コマンドに関する仕様

項目	内容
実行するコマンド名	python 02_polygon_expansion.py
コンフィギュレーション	スクリプトはPythonファイルで設定
結果	建物LOD2オブジェクトの名前と、拡張された2Dフットプリントポリゴンを定義するX/Y座標ペアのセットが含まれるテキストファイルを出力

コンフィグレーション

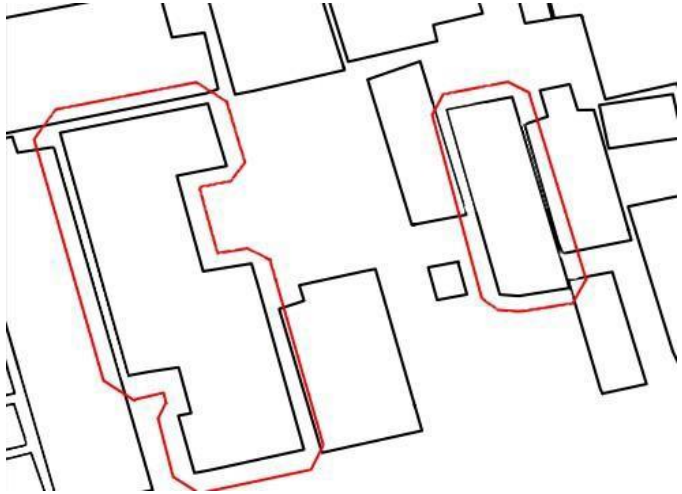
キー	内容
process_all_buildings	true」に設定後「input_filename」で定義されたすべてのビルを処理 Target_building'で定義された単一の建物のみを処理する場合は'False'に設定
target_building	input_filename」に含まれる1棟の建物のLOD2モデル名
input_filename	01_create_footprint_polygons.py‘スクリプトで生成される、建物フットプリント定義を含むファイル
output_filename	拡張された2Dフットプリントポリゴンの定義を含む結果のテキストファイル
dilation_amount	2Dフットプリントポリゴンを拡張する量（メートル単位）
visualize_results	2Dフットプリントポリゴン画像の出力時 'True'、それ以外は 'False' に設定
output_results_to_file	2Dフットプリントポリゴンのファイル出力時「True」、それ以外は「False」に設定

2Dフットプリントの拡張｜概要

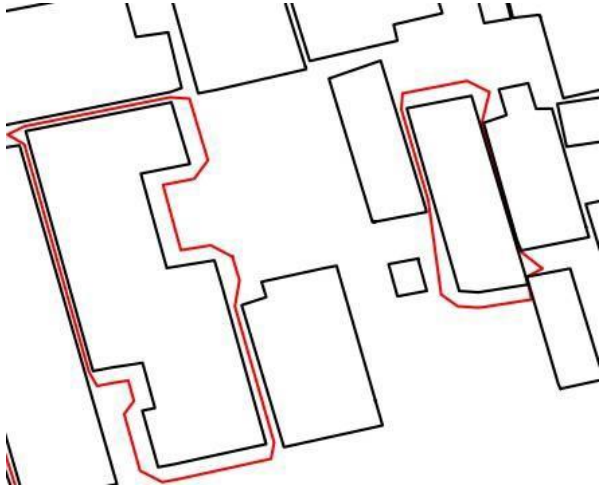
2Dフットプリントの作成処理では、①バルコニーのような付属物がフットプリントから突き出している場合、②点群データが持つ位置の誤差によりフットプリントから建造物の一部が突き出している場合に問題がある。このため2Dフットプリントの拡張が必要になる。現実の都市環境では建造物は高い密度で建設されるため、建造物間の境界を拡張する余裕がないことも多い。

この問題に対応するため、フットプリント拡張アルゴリズムを周囲の地物を考慮するように修正した。各頂点は他の近隣建造物のフットプリントと交差しないように拡張され、拡張した頂点間にエッジを追加する際にも他の建造物のフットプリントとの交差の可能性を検出し、拡張したポリゴンのエッジに追加の頂点を追加することで近隣建造物との接触を回避した。

このアルゴリズムを用いてLOD2モデル全体(2,406棟)を処理し各建造物毎にどの程度の距離を拡張するかを記録したテーブルを作成した。



（左）元のフットプリントポリゴン



（右）隣接する建物を考慮したフットプリントポリゴン

点群合成

複数の点群データの位置合わせおよび合成処理を行い、建物ID単位の点群データを出力する

操作／処理フロー

ステップ①

複数のノイズ除去済み点群データ、2Dフットプリントポリゴンを入力する。

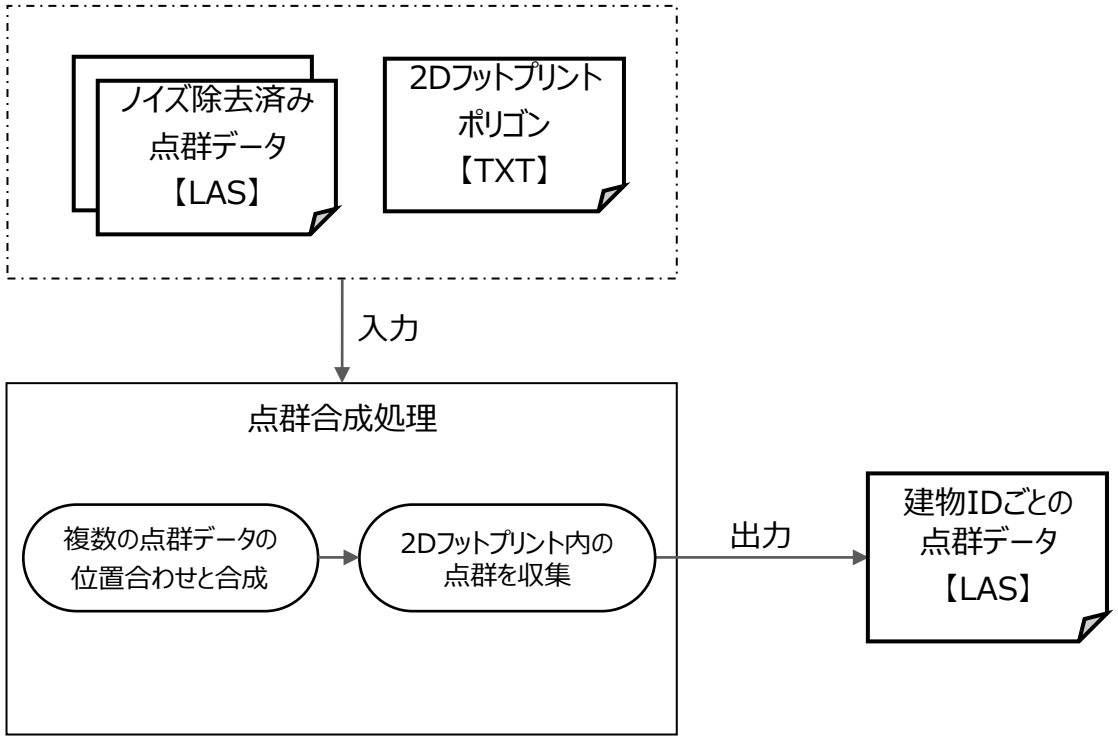
ステップ②

Open3Dのレジストレーション機能を用いて、複数の点群データの位置合わせと合成を行う。

ステップ③

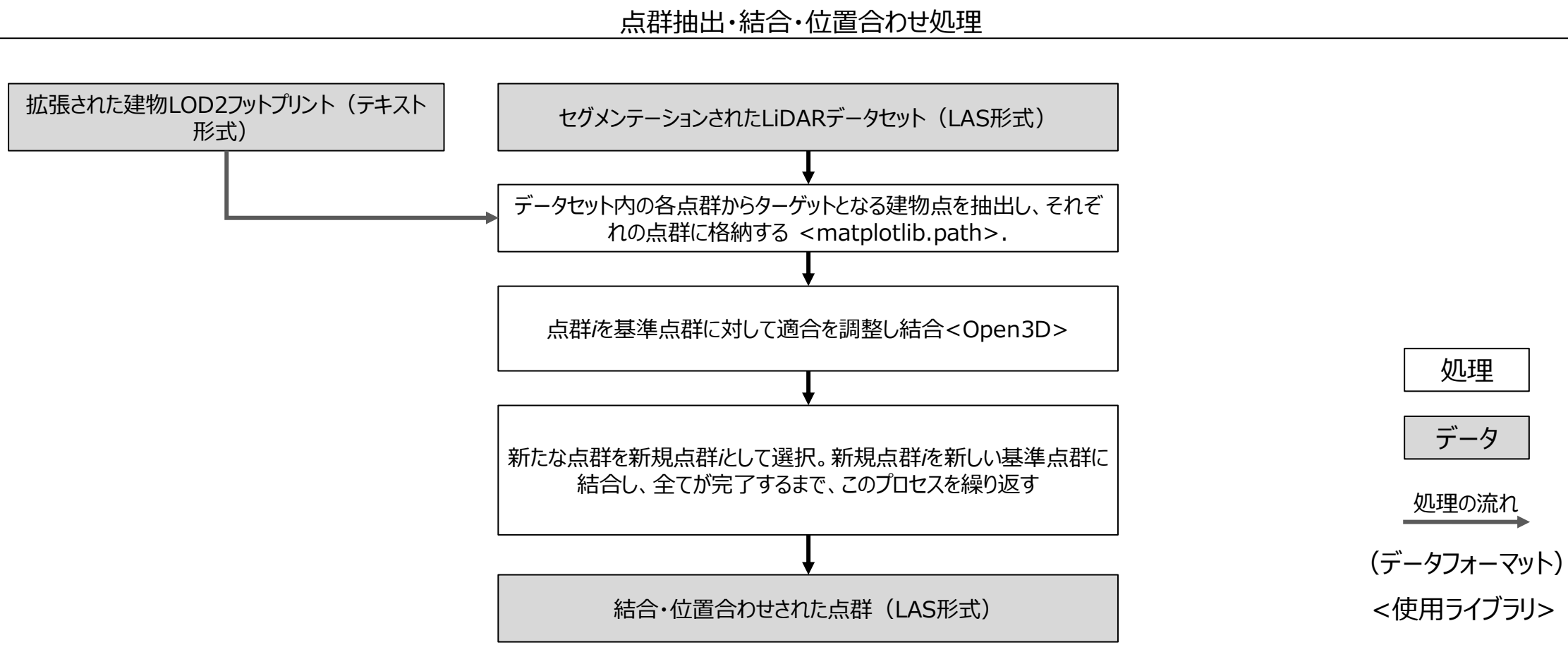
合成された点群データに対し、2Dフットプリントポリゴンを適用し、建物ID単位の点群データを出力する。

システム連携図



点群合成｜処理フロー

複数の点群データの位置合わせおよび合成処理を行い、建物ID単位の点群データを出力する



点群合成｜使用方法

点群抽出・結合・位置合わせ用スクリプトの構成と使用方法

実行コマンドに関する仕様

項目	内容
実行するコマンド名	python 04_split_dataset_points_to_buildings_with_realignment
コンフィギュレーション	スクリプトはPythonファイルで設定
結果	各建物の点群を、点群合成されたものと点群合成されていないものの2つのフォルダに格納

コンフィグレーション

キー	内容
input_building_list	ターゲットビルディングのLOD2 .obj名リスト
building_polygons_file	拡張された建物のフットプリントポリゴンの定義（LOD2 .obj名でID付与）を持つテキストファイル
bounds_file	LiDARデータセットの各点群のバウンディングボックスを生成して処理を高速化（'create_dataset_bounding_box_list.py'で生成）
input_dataset_path	セグメント化されたLiDARデータセットの位置 LASデータ
aligned_output_path	アライン面ンとされた結合建物点群の出力へのパス
unaligned_output_path	点群合成されていない結合建物点群を出力するためのパス

点群合成｜使用方法

点群抽出・結合・位置合わせ用スクリプトの構成と使用方法

実行コマンドに関する仕様

項目	内容
実行するコマンド名	python 04_split_dataset_points_to_buildings_with_realignment
コンフィギュレーション	スクリプトはPythonファイルで設定
結果	各建物の点群を、点群合成されたものと点群合成されていないものの2つのフォルダに格納

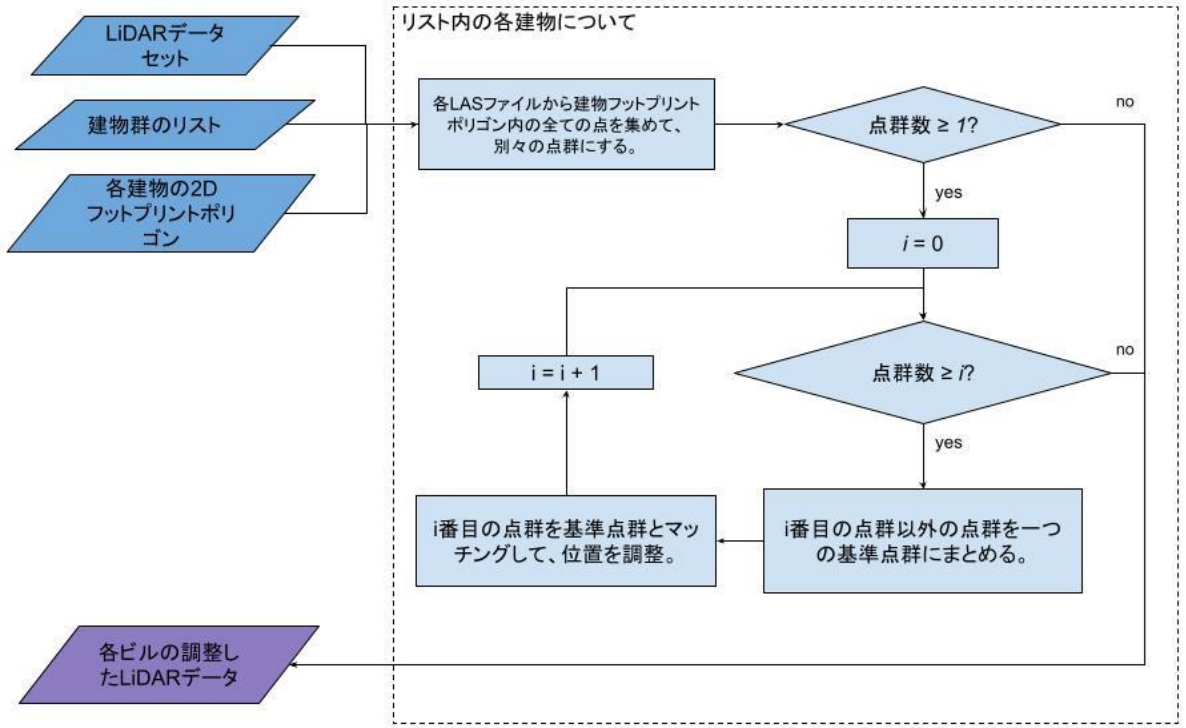
コンフィグレーション

キー	内容
input_building_list	ターゲットビルディングのLOD2 .obj名リスト
building_polygons_file	拡張された建物のフットプリントポリゴンの定義（LOD2 .obj名でID付与）を持つテキストファイル
bounds_file	LiDARデータセットの各点群のバウンディングボックスを生成して処理を高速化（'create_dataset_bounding_box_list.py'で生成）
input_dataset_path	セグメント化されたLiDARデータセットの位置 LASデータ
aligned_output_path	アライン面ンとされた結合建物点群の出力へのパス
unaligned_output_path	点群合成されていない結合建物点群を出力するためのパス

点群合成｜概要

点群合成の処理①

点群の点群合成は、MMS点群から各建物の点を取得し、合成によって建物一つごとの点の塊として出力する処理に統合されている。この処理を可視化したものが以下のフローチャートである。



大規模LASデータセットにおける建物の点群合成処理

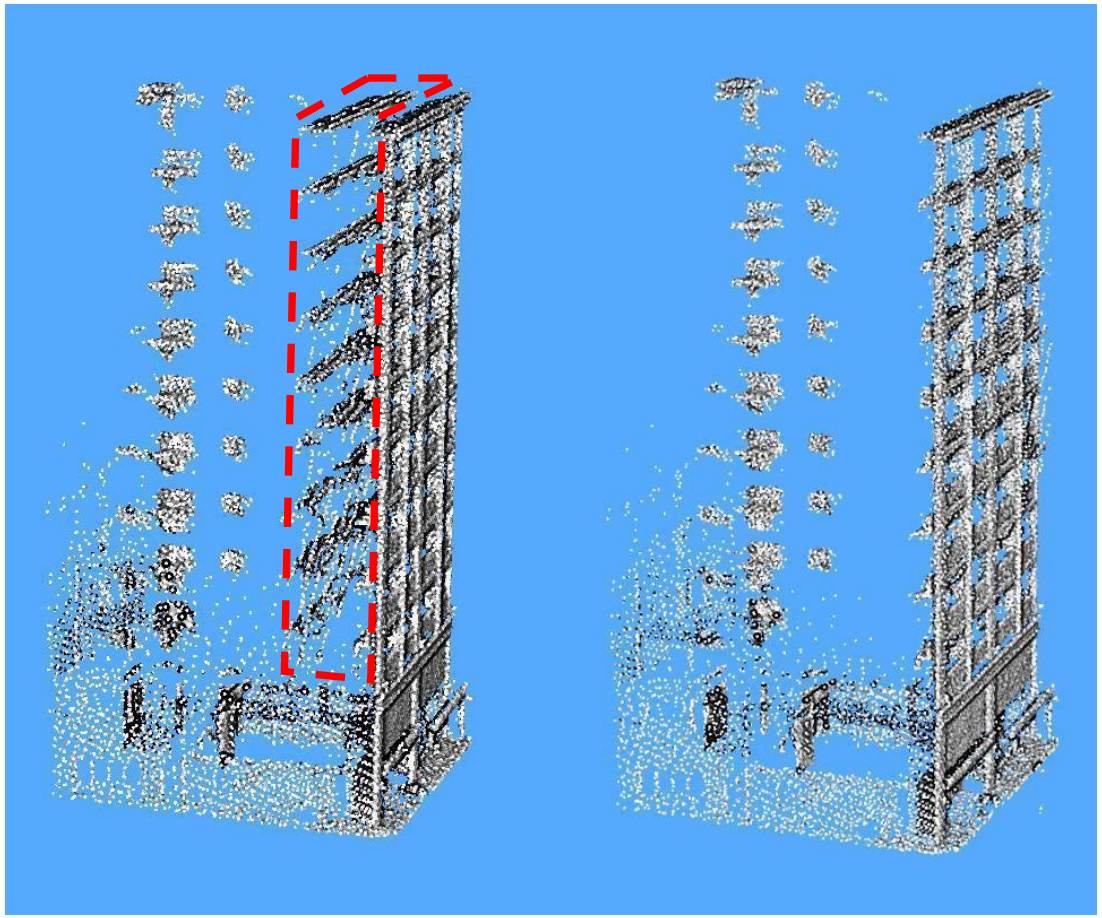
点群合成｜概要

点群合成の処理②

本実証の当初では、全てのMMS点群に対して、LOD2モデルに点群合成を行う複雑なアプローチを試みた。しかし、この方法では前章で示したLOD2データと実際の建造物の形状や大きさの違いから深刻なズレが発生した。このアルゴリズムは本実証の終盤で破棄され、より単純なアルゴリズムに置き換えることで、LiDARスキャン間の点群合成を大きく改善することに成功した。

最新の点群合成手順はシンプルで高速に実行できるものである。最初に対象となる建造物を含む点群をすべて集め、それらの点群同士のすべての組み合わせで点群合成を行う。この過程を繰り返すことで点群全体が徐々に最適な位置へ収束する。

この手法では、ある時点でいかなる点群とも重ならない点群や位置合わせの誤差が大きい点群は、入力された点群をそのまま受け入れる。また、全ての点群同士での点群合成を試みることから、スキャン範囲が重ならない建物の北と南の側面からスキャンしたそれぞれの点群の位置合わせを行うことが可能である。



（左右）点群合成前後の合成点群の比較。LiDARスキャンの1つが他のデータと比較して3mの位置誤差があるが、点群合成処理により修正されていることが分かる

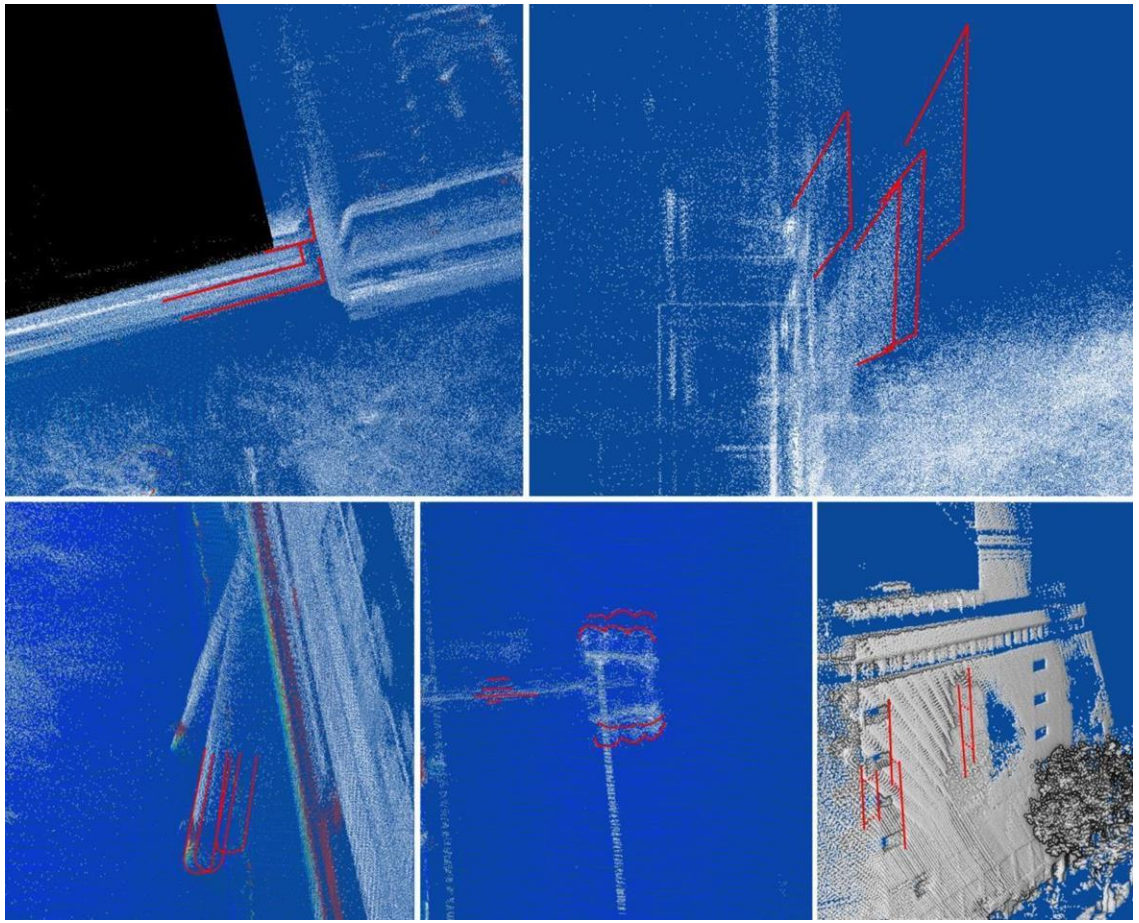
点群合成｜概要

点群合成に必要な条件①

簡易MMSを取り付けた車両によって取得された点群の多くは非常に品質がよく、そのまま組み合わせても見た目には十分に点群合成されているかのように見える。しかし、実際にアルゴリズムで処理をしようとすると問題が多く、追加の位置補正が必要である。

ほとんどの建物で起きる問題は、同じ箇所を別のタイミングに測量したときの点群の僅かな位置のズレである。同じスキャニングセッションで僅かな時間差しかない場合は誤差が小さく、深刻な問題にはならないが、誤差は時間とともに増加しつづける傾向にあり、特に同じルートで異なる日に撮影したバス搭載の簡易MMSデータ同士を結合した場合に顕著な誤差が発生した。

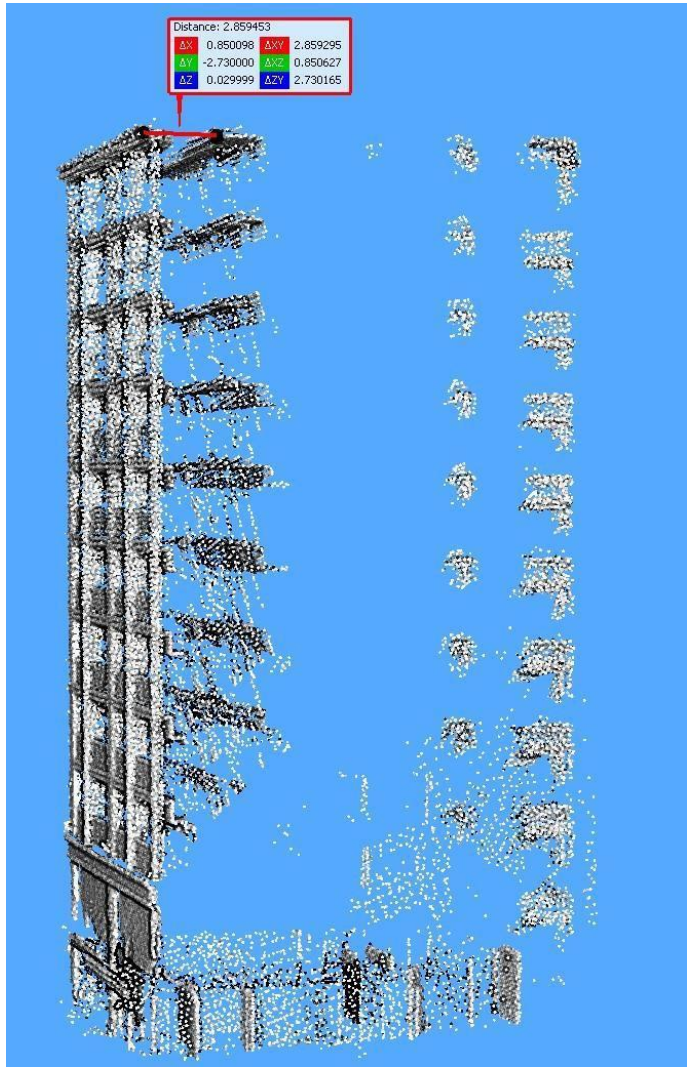
データ取得に使用した「VLP-16 LiDAR Puck」の最大測量距離は約100mであり、仮にセンサーの回転が1度ズレていた場合、センサーから25m離れた地点では0.5m近く位置誤差が生じる。このような現象は合成された点群データのうちセンサーから離れた位置にある脇道や高台などで確認される。



（上）位置が不正確な複数の点群。センサーから遠ざかるにつれて位置誤差が大きくなっている

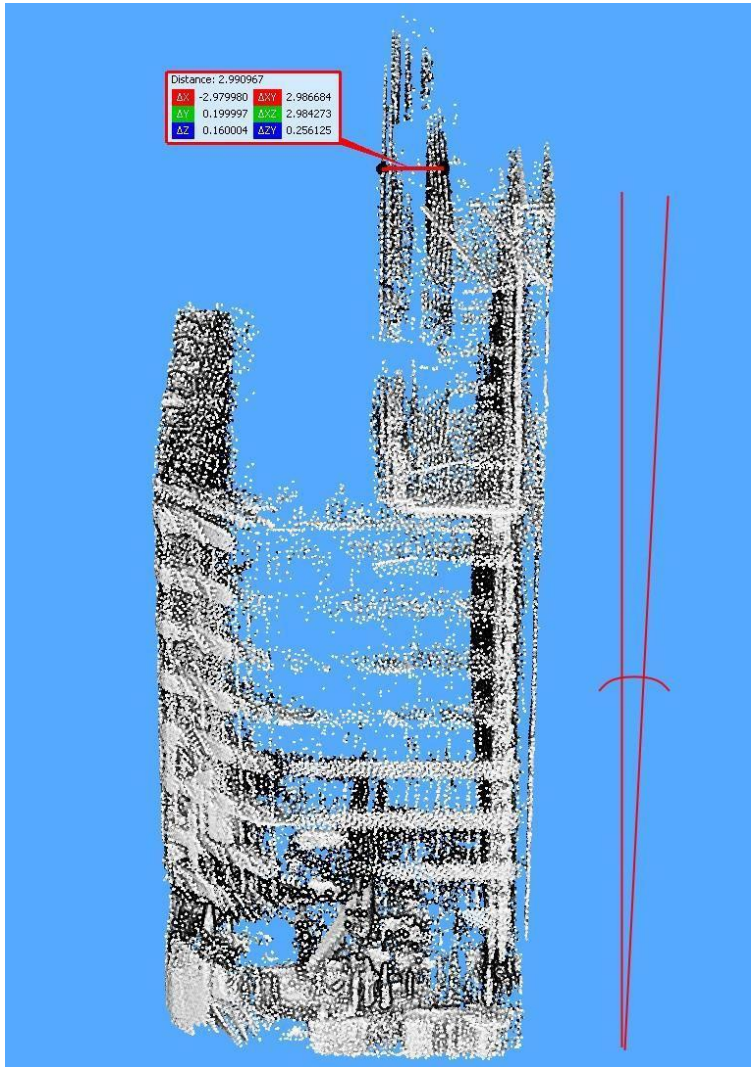
点群合成｜概要

点群合成に必要な条件②



（左）6つのMMS点群の組み合わせ
そのうちの1つは他と比べほぼ3mの誤差がある

（右）ビルの一部のスキャンが5～10度の角度を
持ち、最上階で3mの位置誤差が発生している

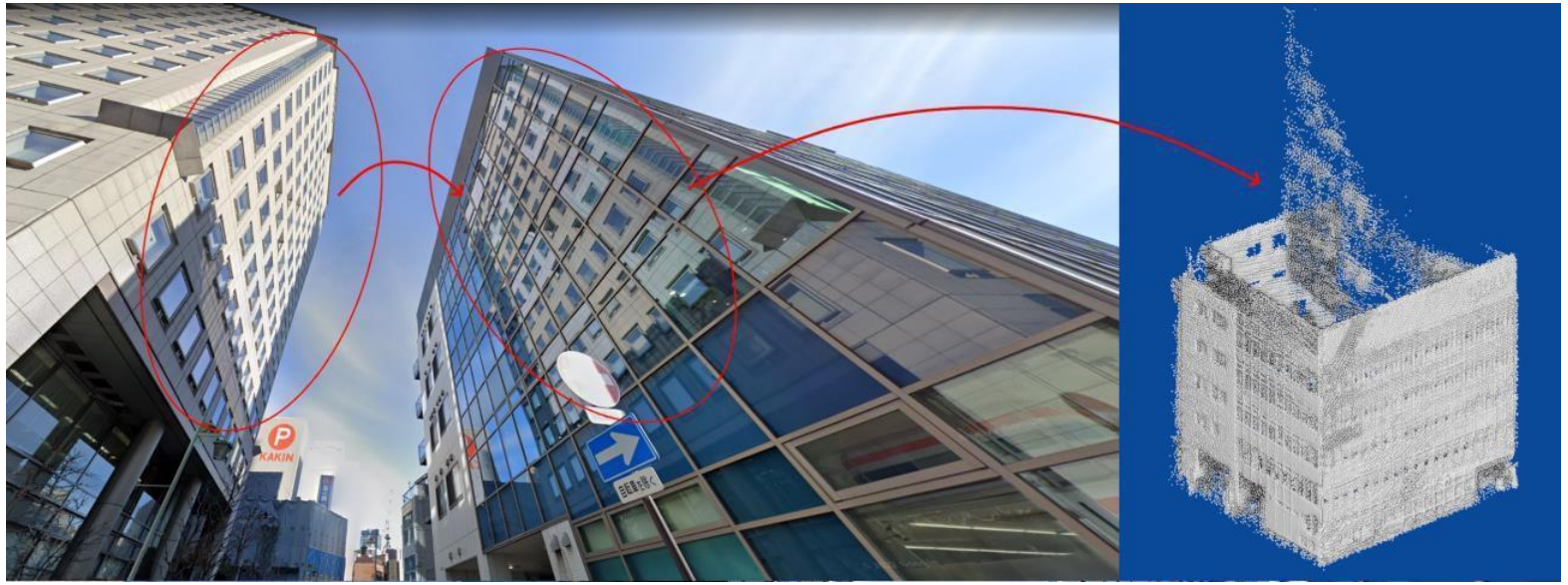


点群合成｜概要

点群合成が困難なケース①

本実証では、LiDARセンサーの位置が、車両の移動では説明できない速さで突然大きくズレるような現象が1つのLASファイル内で発生することがあった。現象が発生時、車両が僅かな回転を繰り返すと若干異なる位置に位置が補正され、その後に正しい位置に補正された。このようなズレの頻度は非常に低く、また対処には別途特殊なアルゴリズムの実装が必要であったため本実証の範囲では非対処とした。

また、より頻度の高い問題として、LiDARの赤外線レーザーの反射によって引き起こされるものがあり、都市部に多く存在するガラス面が多いビルなどが、誤差の原因となる。下記はその一例で、LiDARレーザーの反射する角度によって、ガラス張りの建造物の反対側に位置する建物が建物内部に存在する蜃気楼のように見えている。

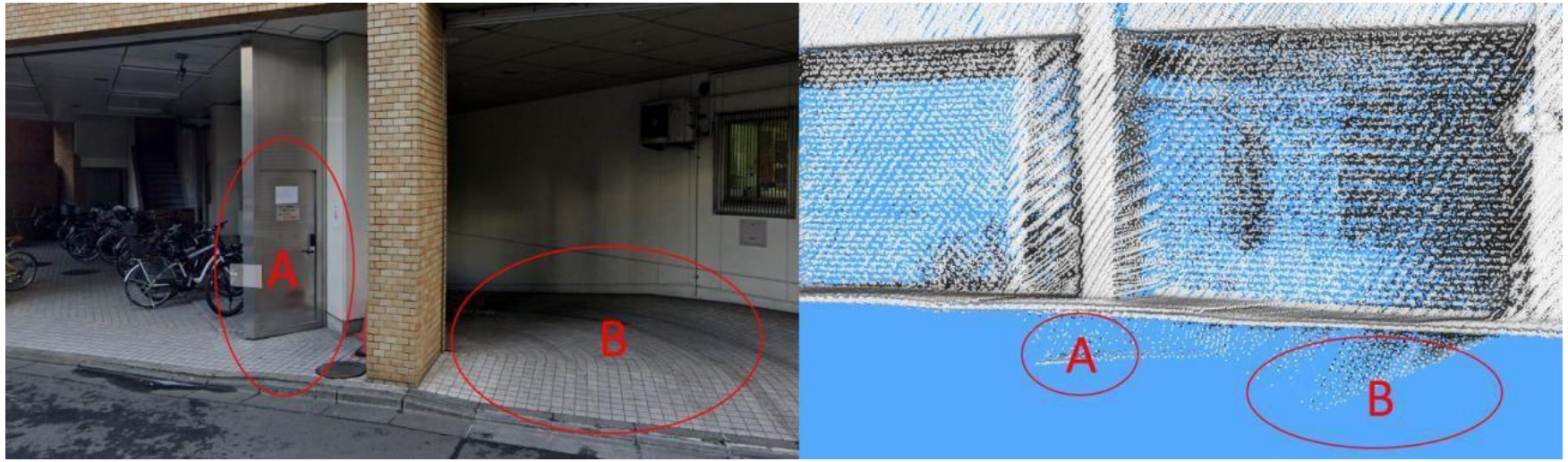


ガラス壁に映り込んだ建造物が、建造物内部に存在する蜃気楼のように見えている

点群合成｜概要

点群合成が困難なケース②

このケースでは、金属の壁にレーザーが強く反射することによって地面よりも下に存在しない床が見えている。



（上）反射で問題が発生した別の例。Bと書かれたスロープは正しくキャプチャされているが、Aと書かれた金属製の壁が反射して地下構造物のように見えている。実際には水平である

これらの問題はメッシュ再構築時に、LOD2から生成した点群と対応する近傍点をフィルタリングすることである程度解決が可能である。しかし、本項68頁「点群合成に必要な条件①」で示すようなLiDARとLOD2データの選択という、より困難な問題を生み出すことになる。

更新箇所特定

点群データと過年度3D都市モデルを比較し、建物更新箇所の検出を行う

操作／処理フロー

ステップ①

ノイズ除去済み点群データおよび過年度3D都市モデル(LOD2)から変換された点群データをDCPCRで大容量の点群データの圧縮、近似点を比較する。

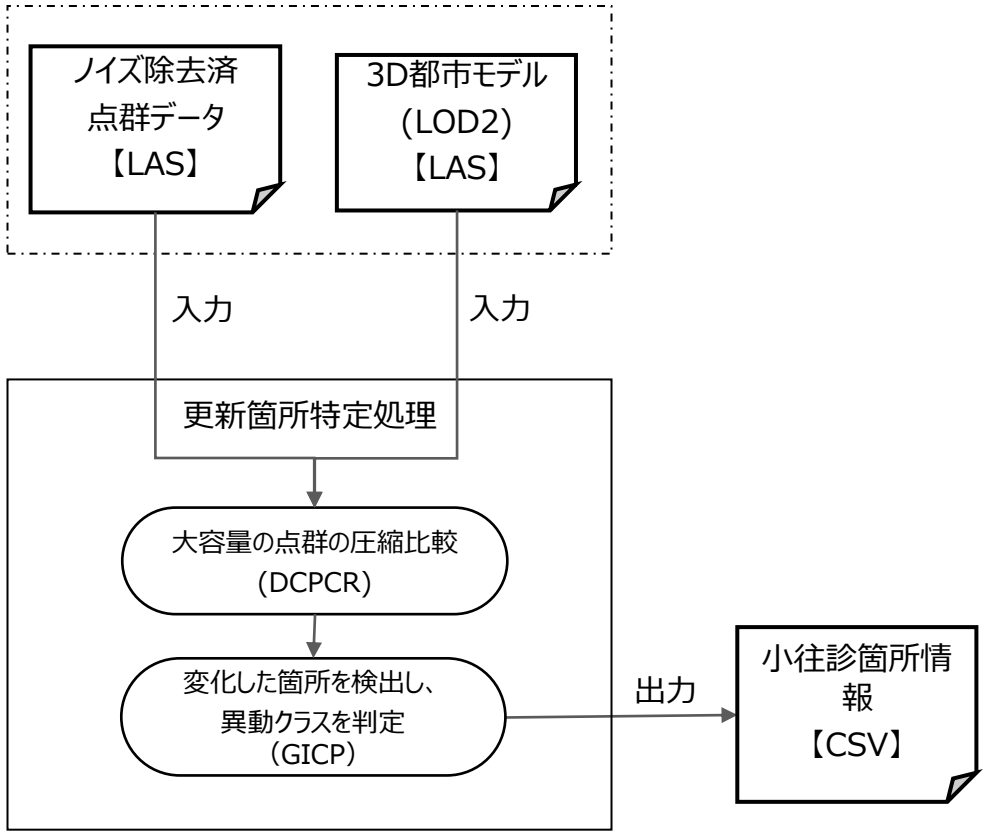
ステップ②

DCPCRで比較された点群をGICPで詳細比較し、変化した箇所を特定する。

ステップ③

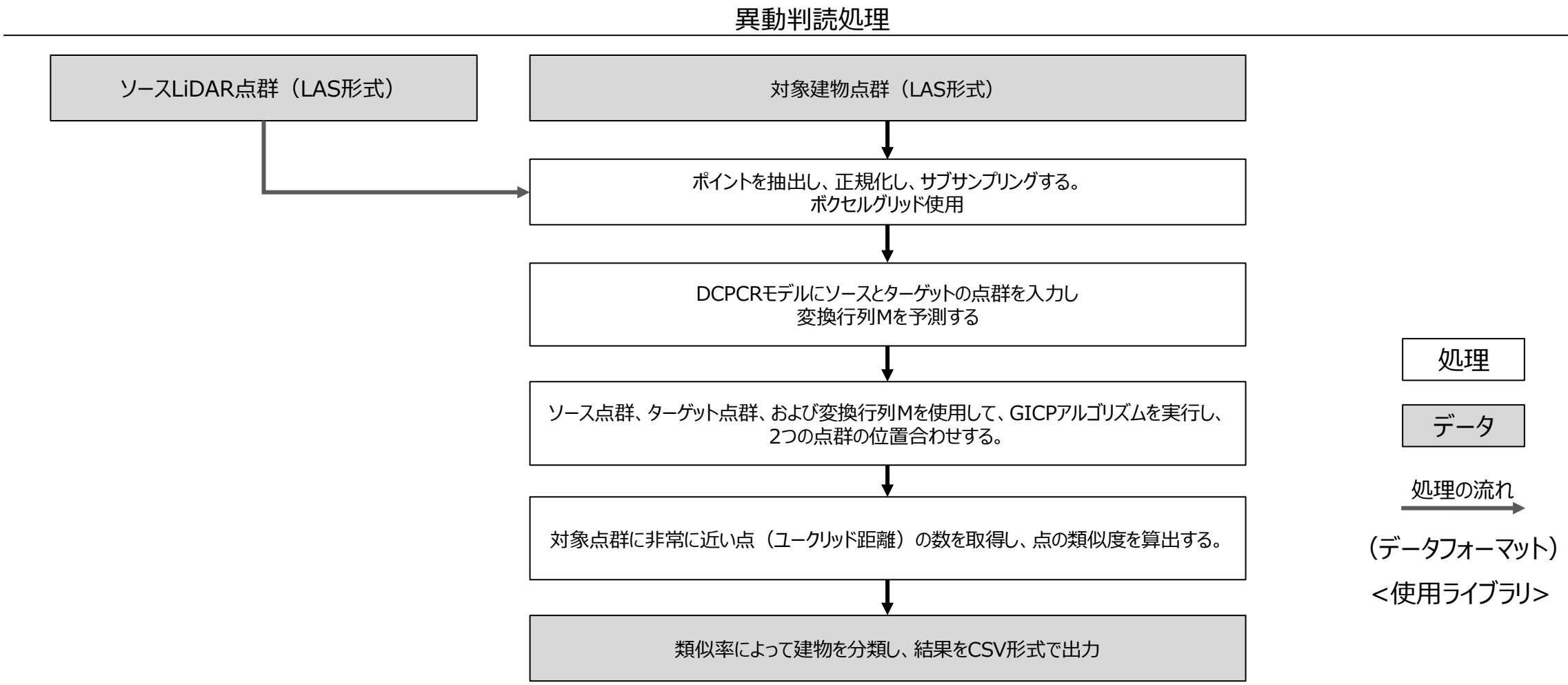
建物IDと異動クラス(新築・滅失・増改築・滅失後新築)を変更箇所情報として出力する。

システム連携図



DCPCR/GICPによる異動判読｜処理フロー

DCPCR/GICPによる異動判読処理の流れは以下の通り



DCPCR/GICPによる異動判読｜使用方法

DCPCR/GICPによる異動判読処理の設定と使用方法

実行コマンドに関する仕様

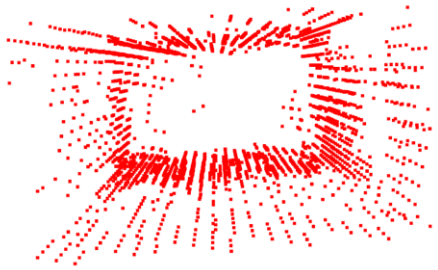
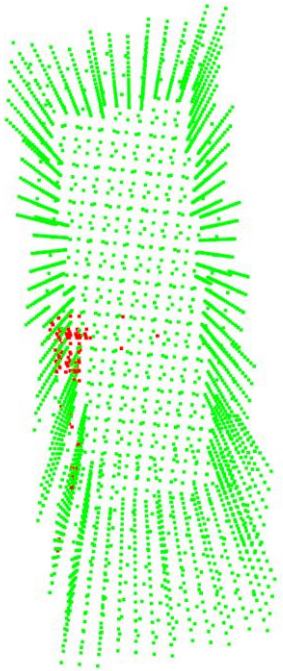
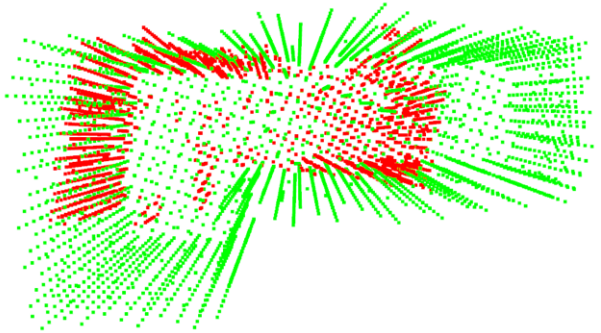
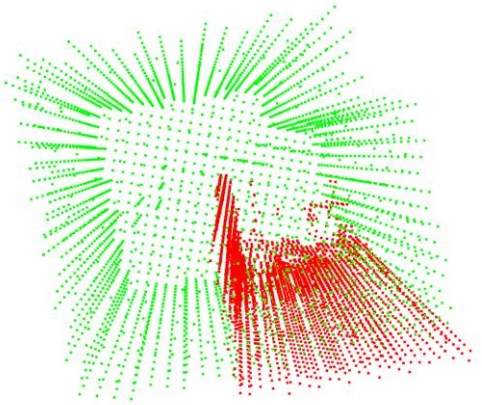
項目	内容
実行するコマンド名	python pointcloud_similarity.py -c [path_to_configfile] [オプション：他のフラグを追加]
コンフィギュレーション	スクリプトで設定。コマンドラインでの設定も可能
結果	各建物のクラス（状態）を格納したCSVファイル

コンフィグレーション

キー	内容
Config (-c)	設定用yamlファイルへのパス
Fine_tune (-ft)	GICPでの微調整
Voxel_size (-vs)	ダウンサンプリングのためのボクセルサイズ
Similarity_ratio (-sr)	2つの建物の類似性を定義する比率
Point_threshold (-t)	建物が破壊されたとみなされる1スキャンあたりの最小ポイント数

DCPCR/GICPによる異動判読｜使用方法

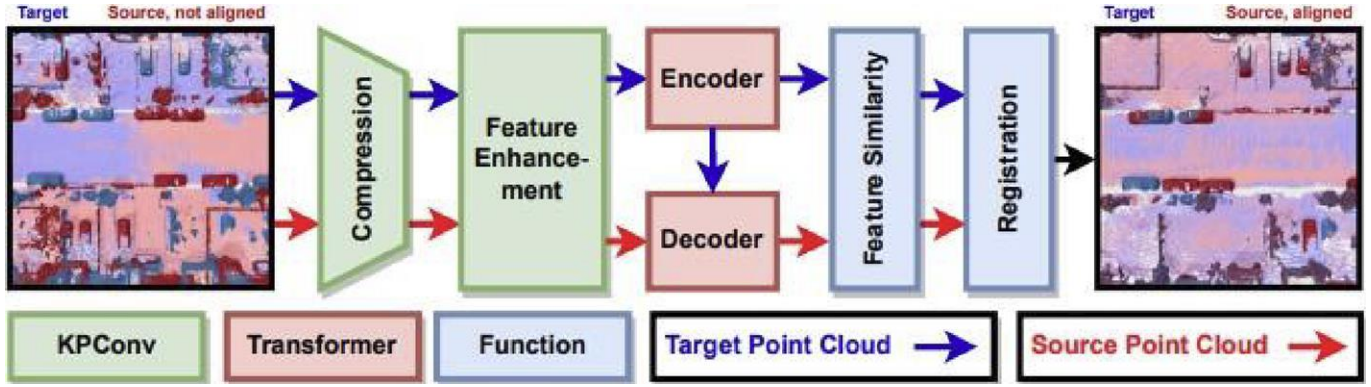
異動判読処理の可視化例である。赤色はLiDARで取得した点群、緑色はLOD2から作成した点群であり、4つの異動クラスの判定方法を可視化した。

新築の例（LOD2なし）	滅失の例	増改築の例	滅失後新築の例
			

DCPCR｜概要

DCPCRは、大規模な屋外環境で取得された点群データを特徴ベースで効率的な圧縮を行う自動運転領域で活用されているアルゴリズムである。

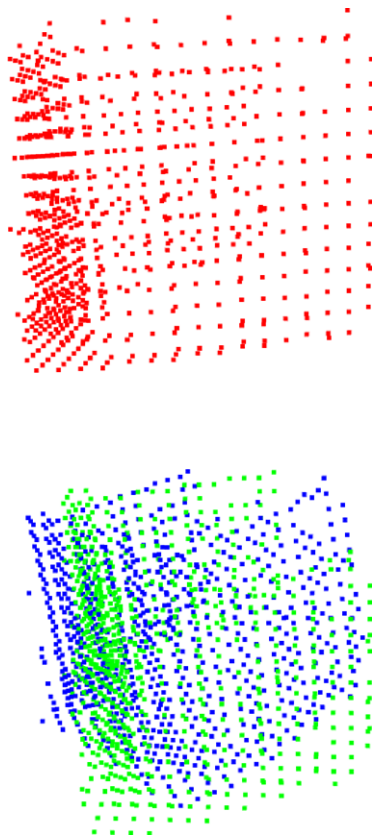
DCPCRは、2つの点群間において、近似点の対応関係を検出し、相対変換を推定するという古典的なパラダイムを踏襲している。古典的なICPでは、対応関係は幾何学的近傍によって決定されるが、DCPCRは特徴ベースのアプローチに従う。回転と並進からなる変換は、図に示すようなニューラルネットワークを用いて推定される。



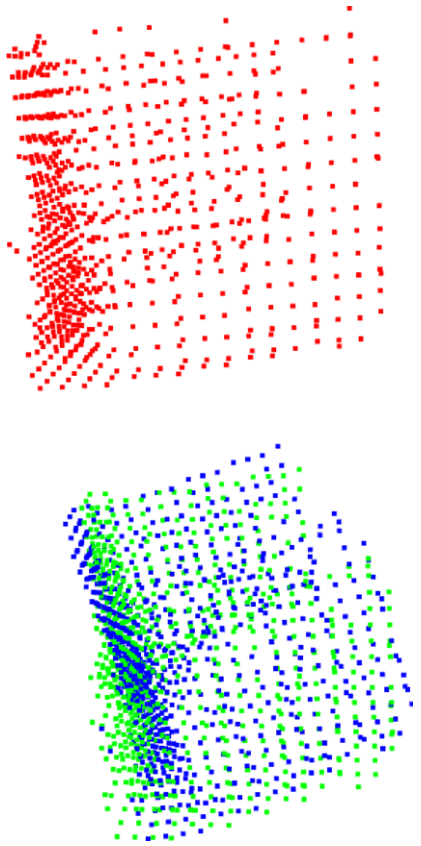
DCPCRの詳細な図解（出典：https://github.com/XuyangBA.I./awesome-point-cloud-registration）

GICP｜概要

「Generalized-ICP（GICP）」は、２つの点群データの再近接点同士をマッチングさせるアルゴリズムである。都市などの点群では前述のDCPCRと併用することが多い。従来手法である「ICP（Iterative Closest Point）」の拡張版である。



GICPを用いない場合
赤：ソース点群
緑：ターゲット点群
青：マッチングした位置



GICPを用いた場合
赤：ソース点群
緑：ターゲット点群
青：マッチングした位置

メッシュ再構築

点群データからメッシュを再構築する。必要に応じて、点群データの欠損部を3D都市モデルLOD2で補間する

操作／処理フロー

ステップ①

建物IDごとの点群データを入力する。

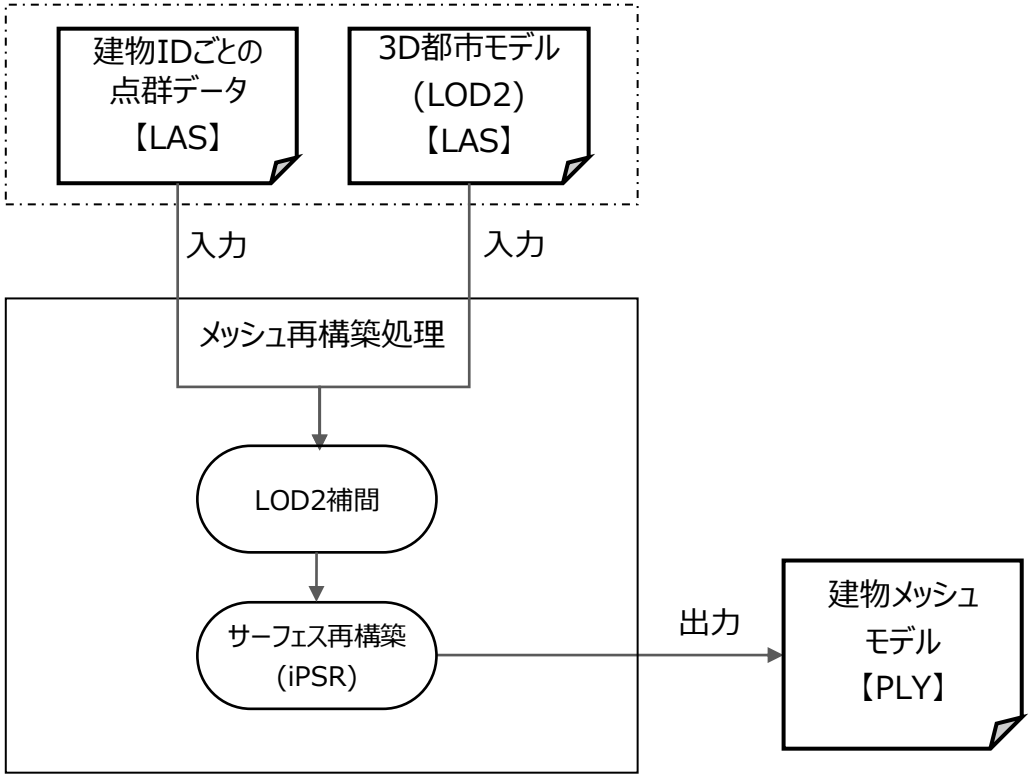
ステップ②

点群データの欠損部分について、3D都市モデルLOD2の点群データで補間する。

ステップ③

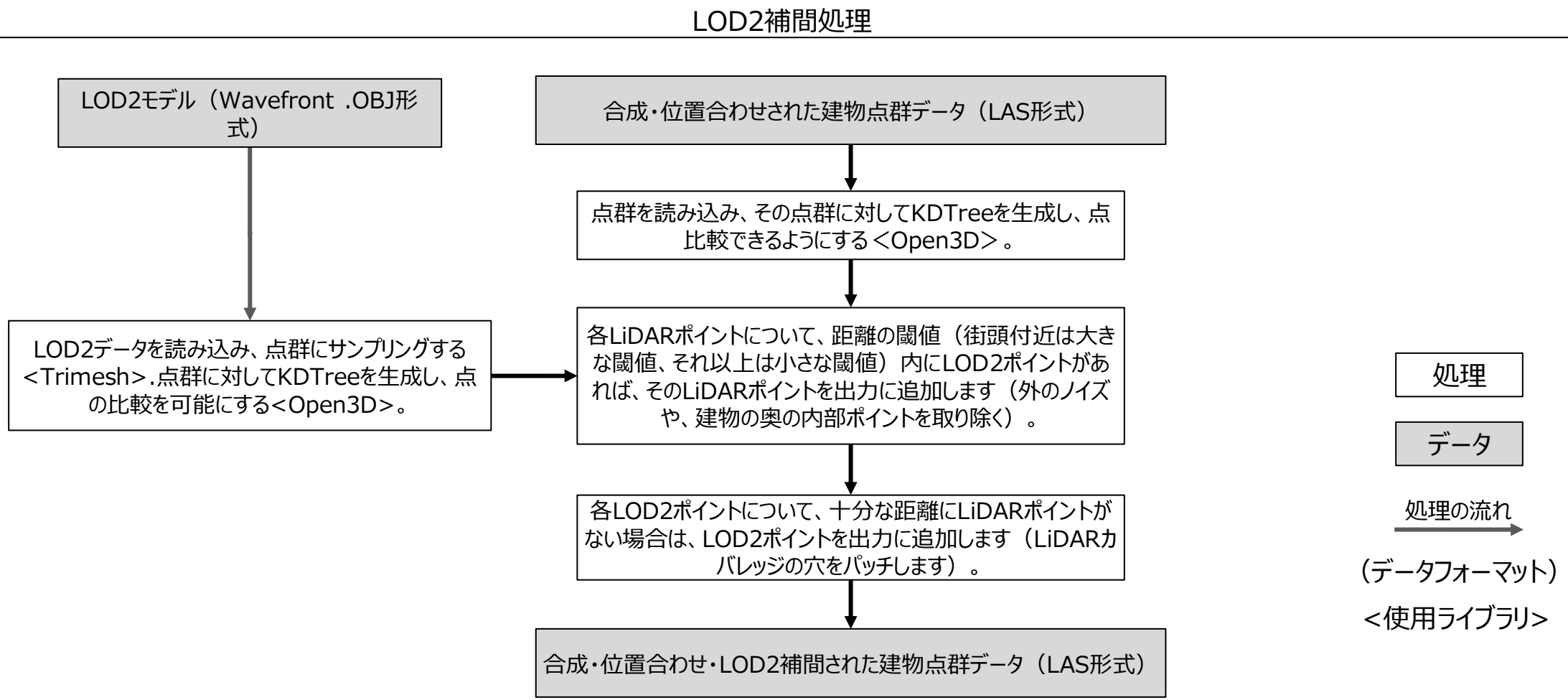
iPSRによりサーフェス再構築処理を行う。

システム連携図



点群データのLOD2補間｜処理フロー

点群データのLOD2データによる補間処理は以下のとおり



点群データのLOD2補間｜使用方法

点群データのLOD2データによる補間処理の設定と使用方法

実行コマンドに関する仕様

項目	内容
実行するコマンド名	python 05_combine_lod2_and_point_cloud.py
コンフィギュレーション	スクリプトはPythonファイルで設定
結果	メッシュ再構築に対応したPLY形式の点群

コンフィグレーション

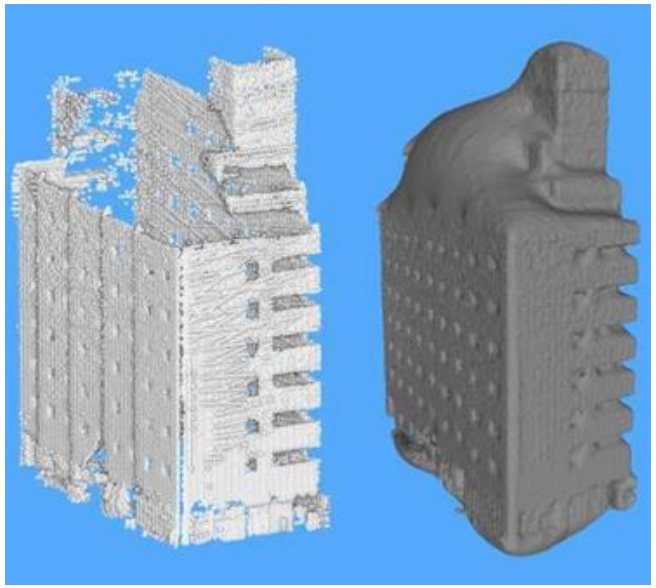
キー	内容
input_point_cloud_path	(‘04_split_dataset_points_to_buildings_with_realignment.py’)で出力されるLAS形式の点群の格納ディレクトリ
input_lod2_obj_path	Wavefrontの.OBJ形式のLOD2建物モデルの格納ディレクトリ
output_point_cloud_path	生成された点群の格納ディレクトリ
process_entire_directory	「True」設定時、入力パスの全点群を処理（点群に一致するLOD2モデルが必要）。それ例外は、‘building_number’で定義された1つの建物だけを処理
building_number	「process_entire_directory」を「False」に設定時、この番号を持つ建物のみを処理

点群データのLOD2補間｜概要

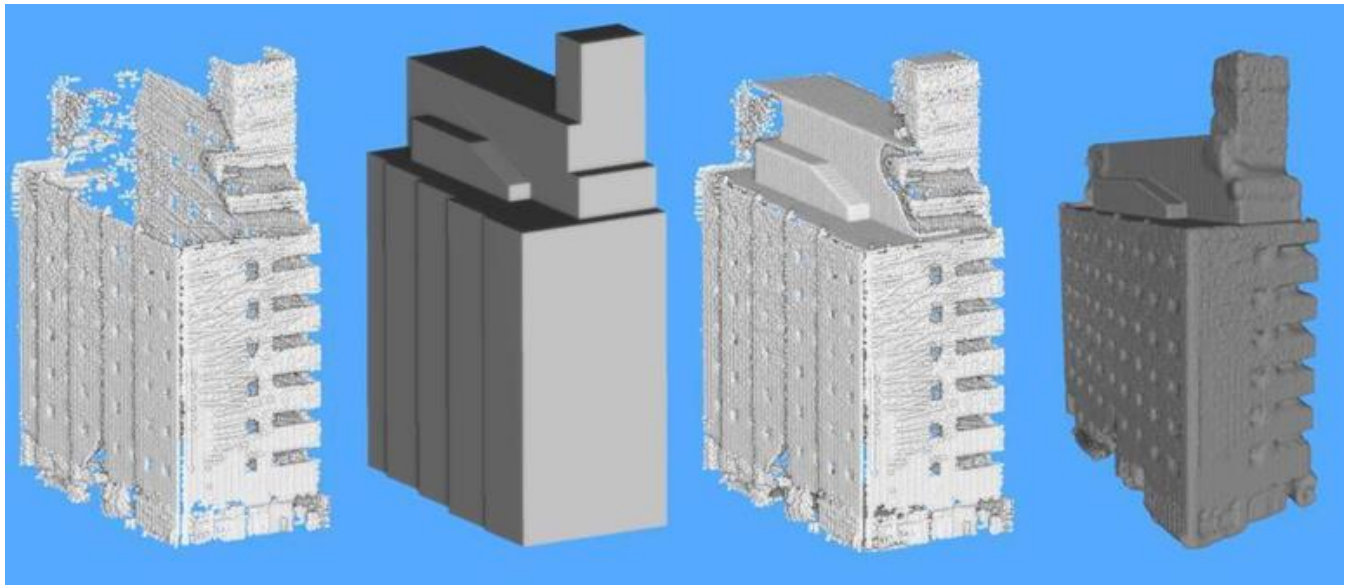
点群データ撮影時に取得できなかった欠落部分を補う

建造物が密集している都市部では、車両搭載の簡易MMSで建物を四方から正確にスキャンできるケースは少なく、またスキャン角度と撮影距離の問題で屋上部はスキャンできない。

しかし、点群からメッシュを正確に作成するためには、建物の全面を捉える必要がある。特にポアソン表面再構築アルゴリズムは、密閉されたメッシュを作成することを目的としているため、データが欠落した箇所に対し不適切な補間を行う可能性がある。このため、既存のLOD2建築物モデルを使用して、点群の欠落部分を補間した。下図は、この手法の効果を視覚化したものである。



建物のLiDARスキャンをポアソン面再構成した例



建物のLiDARスキャンとLOD2モデルを点群化したものを結合し、ポアソン面再構成した例

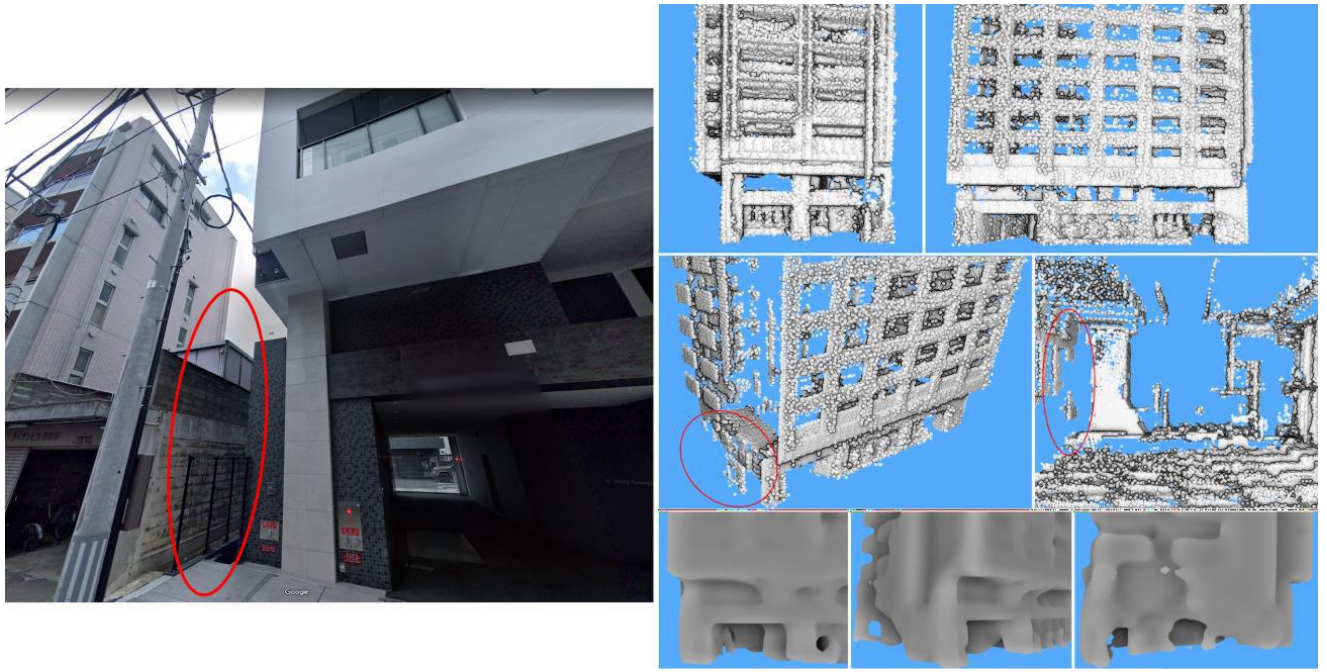
点群データのLOD2補間｜概要

点群データ撮影時に取得できなかった欠落部分を補う

MMS点群とLOD2モデルの自動合成が困難な場合も多い。右の画像は、隣の建物（赤丸）がMMSの視界から対象建物の1階奥の壁を遮る難易度の高いケースの一つである。

このケースでは、建物を貫く内部の通路の側の壁がMMSで捉えられない一方で、その壁からわずか1メートルしか離れていない地点にMMSで捉えられる壁が存在し、建物の境界を見分けることを困難にしている。また、建造物の地上付近は凹んだ形状をしており、ビルの側面の通路と内部の通路を誤認しやすい形状となる。

このような環境で正しいメッシュを得るためには、iPhone LiDAR等で隠れた箇所のデータを追加取得するか、より現実的な方法としてメッシュ作成プロセスの中で矛盾するデータの手作業による修正が必要になる。



（図）MMS点群とLOD2の結合時の自動化が難しい例
建物を通る廊下は正しくキャプチャされてるが、欠落した外壁に近すぎるため、LOD2データによるパッチが適用されず、メッシュが破損している

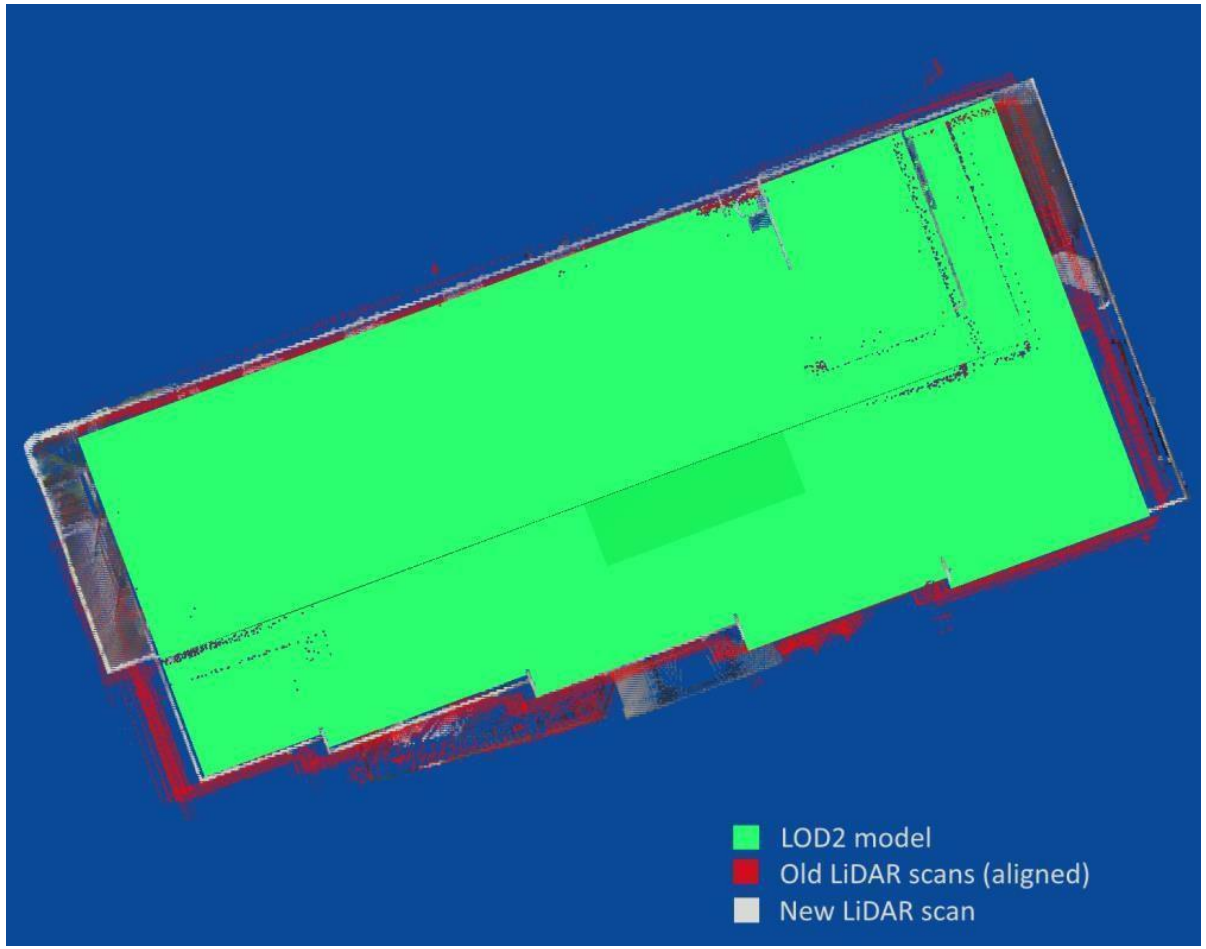
点群データのLOD2補間｜問題

LOD2モデルとの差異の解決①

本実証では、MMS点群データのスキャン時の欠落部位を補うため、LOD2モデルによりMMS点群データの補完を行った。

1つ目の問題は、MMSで撮影された実際の建造物の点群データとLOD2モデルの間には少なくない差異が存在することである。

右の画像に見られるように、手動で作成されたLOD2モデルは、バルコニーなどの特徴が（LOD2の標準製品仕様に含まれないため）除外されている。また、実際の建造物のかなりの範囲がLOD2モデルと比較した場合、外壁範囲等が異なる場合があり問題を引き起こす要因となりえる。



（上）LOD2モデルとMMS点群データが一致しない例。手動で位置合わせしたMMS点群データは、LOD2モデルより2mほど特定領域が広くなっている。

点群データのLOD2補間｜問題

LOD2モデルとの差異の解決②

この問題を解決するため、LOD2モデルのフットプリント生成を活用した。LOD2フットプリントからはみ出る可能性のあるバルコニーなどの付属物や、MMS自己位置推定の誤差を考慮し、「7. 建物抽出」の章と同様に、LOD2フットプリントポリゴンに対し、設定した量の建物領域の拡張を行った。しかし、LOD2モデルがMMS点群データよりもかなり小さい場合、フットプリントポリゴンの拡張量が、建物を完全に囲むのに十分でない場合がある。

右の画像は、同じ建物を使ってこれを説明しているものである。LOD2フットプリントの初期値の拡張量は1.5メートルだが、この建物の場合、MMSで建物全方位が完全に取得されていた場合でも、すべてのバルコニーデータを網羅するためには4メートルの拡張が必要であった。そのため、初期値では左側のバルコニーの点群データの一部が出力時に欠落を起こしていた。この問題を解決するためには、より柔軟で内容を考慮したLOD2フットプリント拡張アルゴリズムを開発する必要があった。

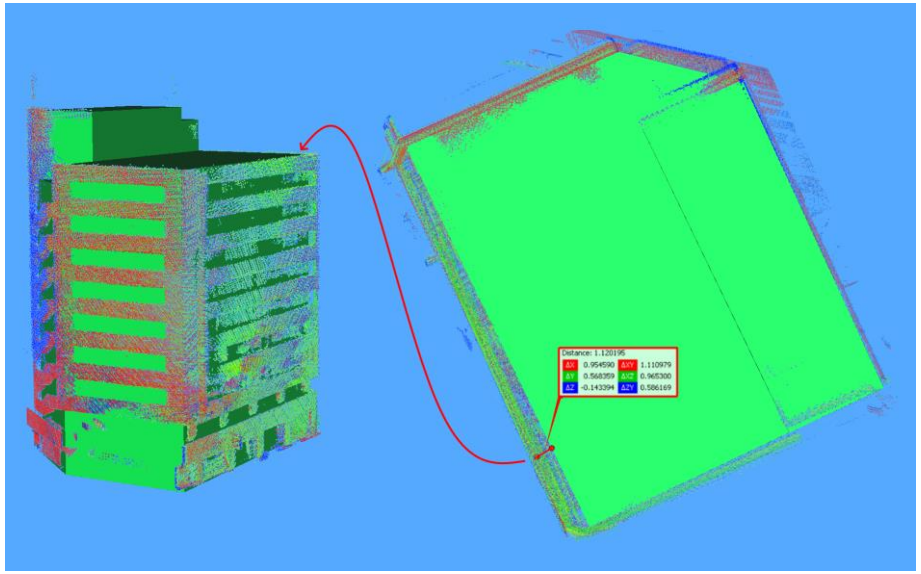
（左）LOD2のフットプリントを規定値の1.5mで拡大したもので、完璧にアラインしても実際の建物を捉えるには十分でない。（右）LOD2フットプリントを規定値から4m拡大したもので、拡大量が十分であることがわかる

点群データのLOD2補間｜問題

LOD2モデルとの差異の解決③

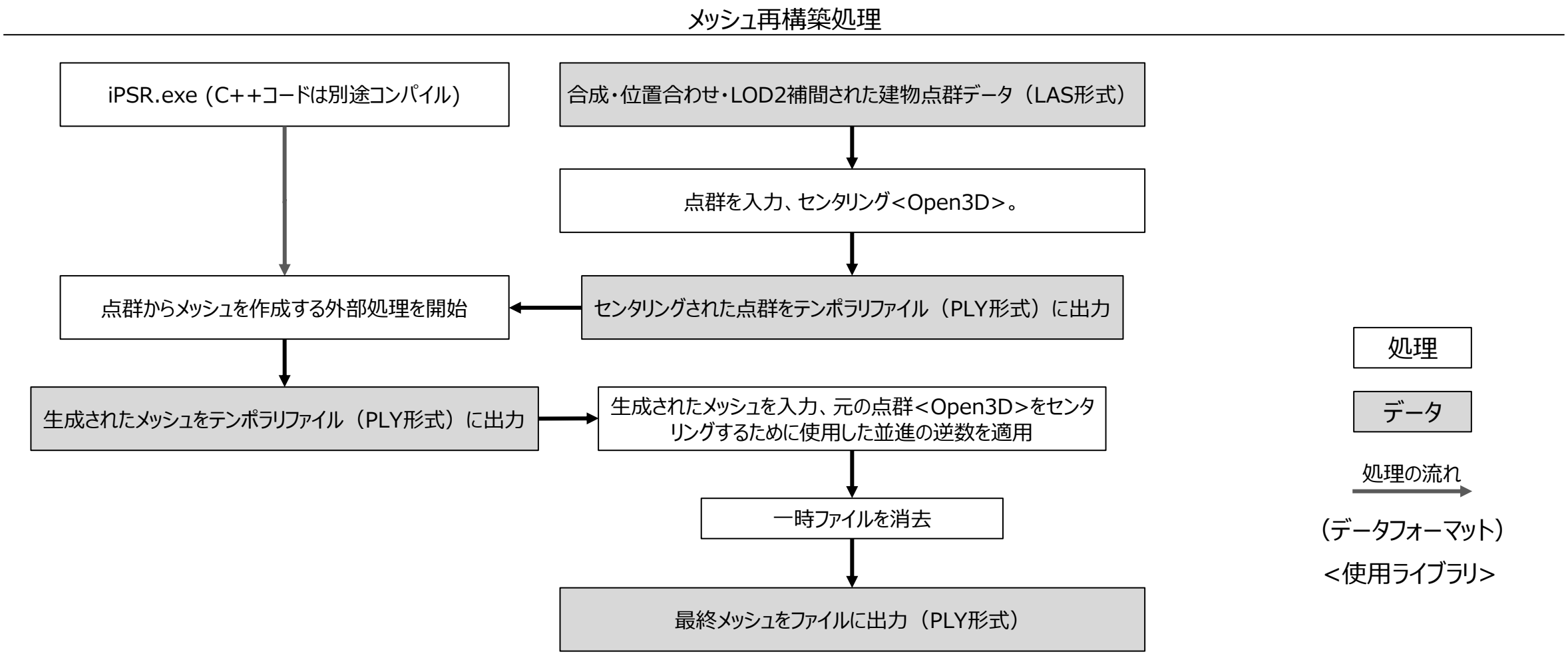
2つ目の問題は、メッシュ再構築に使用するデータソースを決定する際に発生した。同じ位置を表すはずのMMS点群とLOD2モデルが離れている場合、廊下や窓、深い軒下などの特徴を検出することが難しくなるため、LOD2モデルを使用することでむしろ性能低下を招いた。この課題は今後も継続的な解決が必要である。

3つ目の問題は、LOD2モデルをMMS点群の点群合成のアンカーとして使用する際に発生した。これは前述のようなLOD2モデルと実際の建造物のズレに起因している。このような問題が発生したためにLOD2モデルは点群のクロッピングのみに使用し、新たにLOD2モデルを全く利用しない点群点群合成アルゴリズムを実装した。



iPSRによるメッシュ再構築｜処理フロー

iPSRによるメッシュ再構築処理の流れは以下の通り



iPSRによるメッシュ再構築｜使用方法

iPSRによるメッシュ再構築処理の設定と使用方法

実行コマンドに関する仕様

項目	内容
実行するコマンド名	python 06_run_iPSR.py
コンフィギュレーション	スクリプトはPythonファイルで設定
結果	生成メッシュは出力フォルダに格納

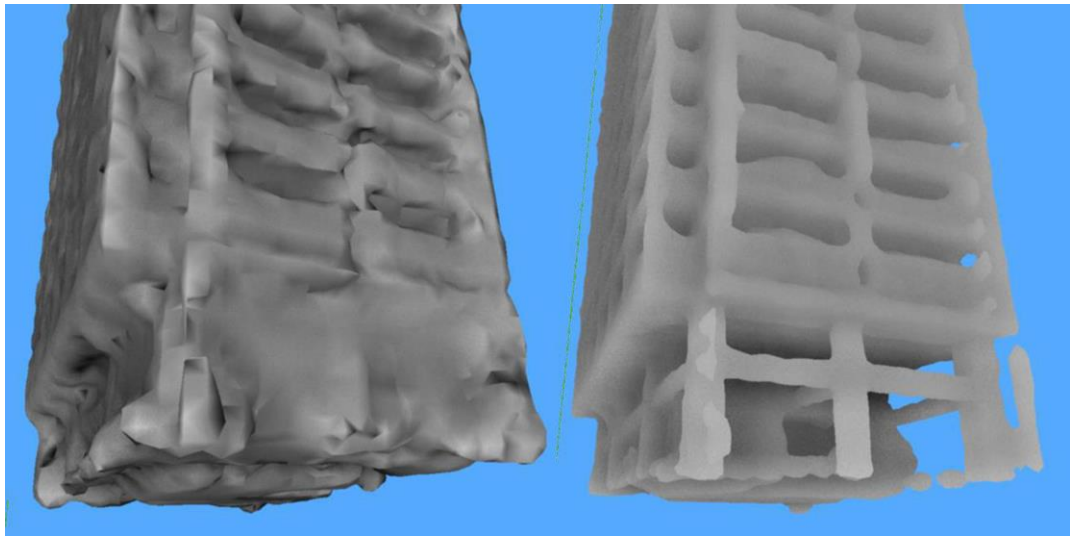
コンフィグレーション

キー	内容
process_entire_folder	「True」設定時、入力フォルダ内の全点群を処理する 「False」設定時、1つの点群を定義する
input_path	LAS形式で保存された点群（要点群合成処理）を含むフォルダ
output_path	最終メッシュの出力先となるフォルダ
input_filename	process_entire_folder‘ が ’False‘ に設定時の入力点群ファイル名
output_filename	process_entire_folder‘ が ’False‘ に設定時の出力点群ファイル名

改良型ポアソン表面再構成アルゴリズム(iPSR)①

2022年秋、「iPSR（iterative Poisson surface reconstruction for unoriented points）」と呼ばれるポアソン表面再構成アルゴリズムの改良バージョンが発表された。（ https://arxiv.org/abs/2209.09510, https://github.com/houfei0801/ipsr）。本実証では、オリジナルの「PSR（ポアソン表面再構成アルゴリズム）」を用いていたが、2022年12月に「iPSR」を取り入れた実証を行った。

標準的なポアソン表面再構成アルゴリズムは、処理するポイントクラウドの法線ベクトルが必要であり、より複雑でノイズの多いポイントクラウドでは生成が難しい場合がある。ノイズが多く法線が安定しない場面では適切な表面を生成できず、建物のすべての内部構造が無視されることになる。



建物正面は非常に複雑な形状をしており、点群に対して直接法線を推定すると失敗する。

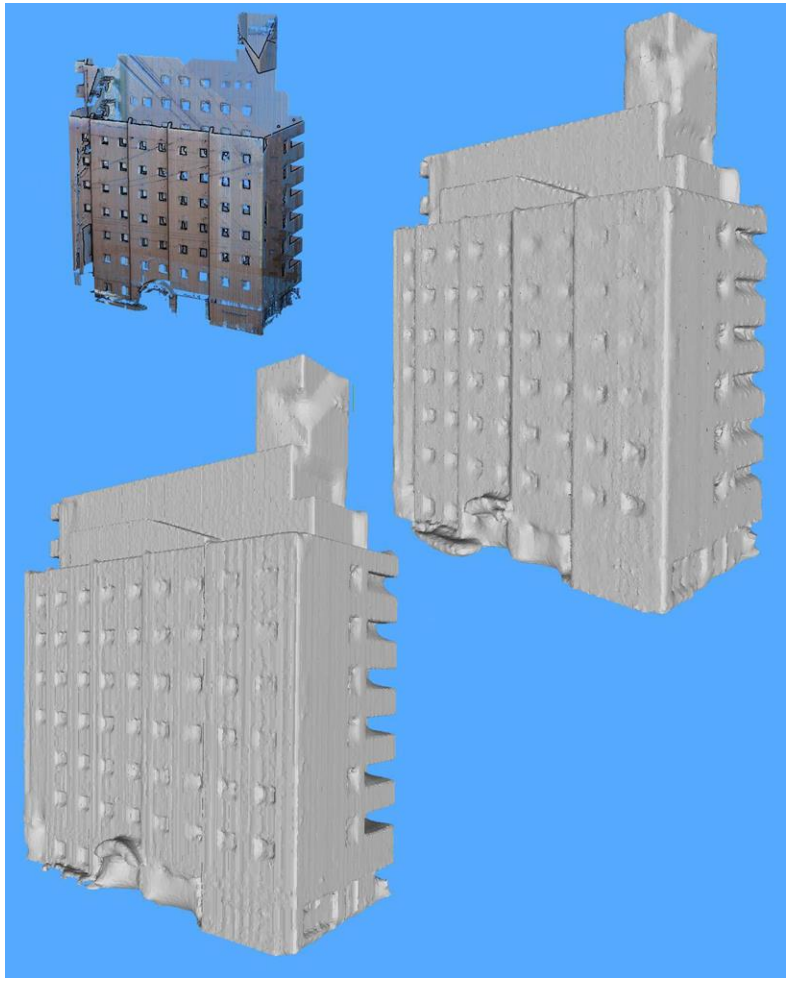
（左）従来のポアソン表面再構成アルゴリズムでは、法線ベクトル情報が必要なため、建物の再現に失敗している。

（右）改良型iPSRアルゴリズムでは、メッシュ再構築中に法線ベクトルを繰り返し生成するため、より良い結果が得られる。

改良型ポアソン表面再構成アルゴリズム(iPSR)②

新規iPSRでは、事前に法線ベクトルを計算する必要がなくなり、各点に対する法線ベクトルを推定し、それに基づいてポワゾン表面を生成し、その後、再度2段階のプロセスを複数回繰り返し、最適解に収束するまで法線とメッシュ品質を改善する。アルゴリズム作成者は仮に初期法線ベクトルがランダムに初期化されたとしても、ほとんどの場合5～30回の反復で正しい解が見つかると述べている。

新規iPSRアルゴリズムの反復的な性質は、困難なメッシュ形状を推定可能なだけでなく、ノイズがあっても従来より点群の直線エッジを直線的なメッシュに変換することを可能にする。右の画像は、ポアソンとiPSRアルゴリズムの両方で生成されたメッシュである。iPSRではポアソンと比較してよりエッジが協調され平滑化が行われている。

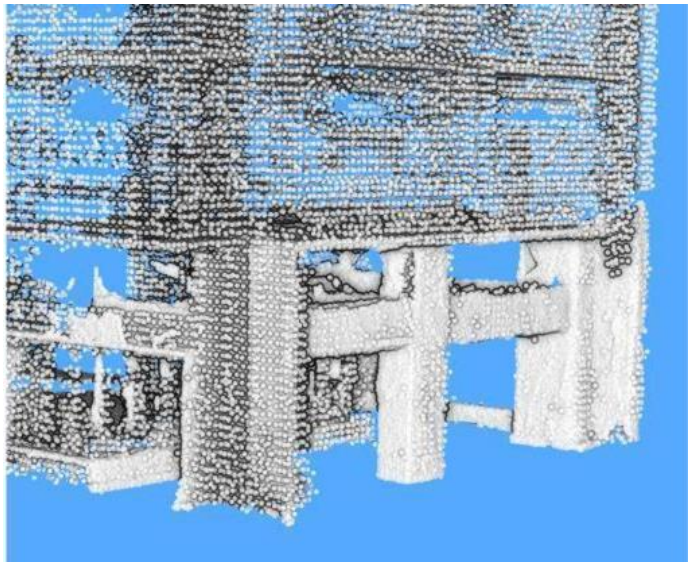


車両に搭載されたLiDARスキャンを組み合わせたメッシュの作成例。（上）PSR、（下）iPSRの比較

点群データの追加スキャンによる補間｜概要

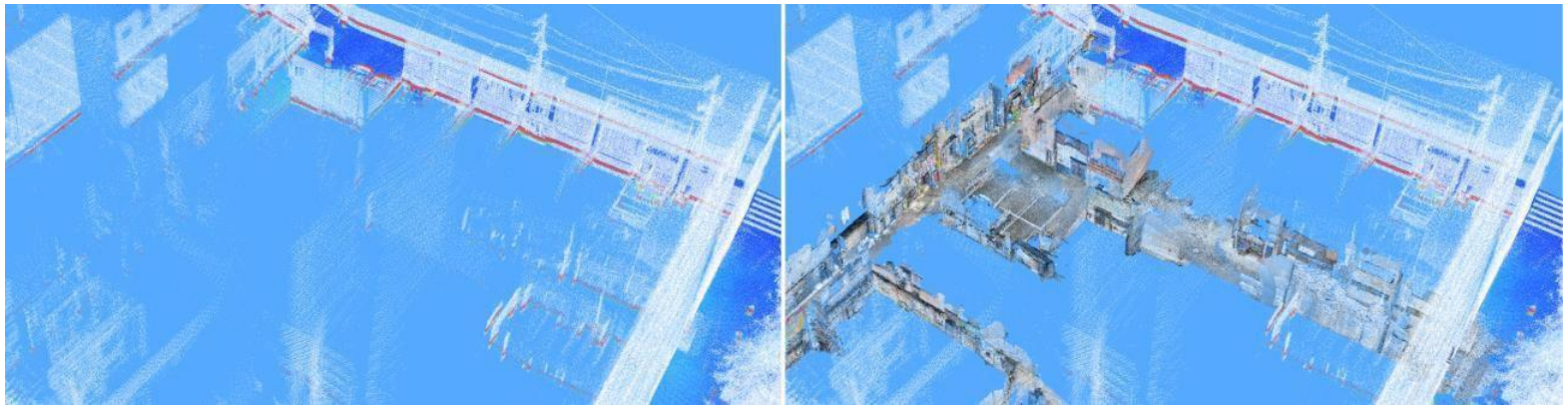
可能な限り高品質な結果を得るために、複数の種類のセンサーでスキャンしたデータも利用可能である。LiDARを搭載したiPhoneなどのモバイルスキャナーを使用し、車両に搭載したセンサーが見通せない場所からのデータ取得により、建物全方向から高品質のスキャンを生成することが可能である。

（左）バス搭載センサーのLASファイルを複数組み合わせて作成した点群に、iPhone LiDARでスキャンしたデータを追加し、死角をカバーしたもの

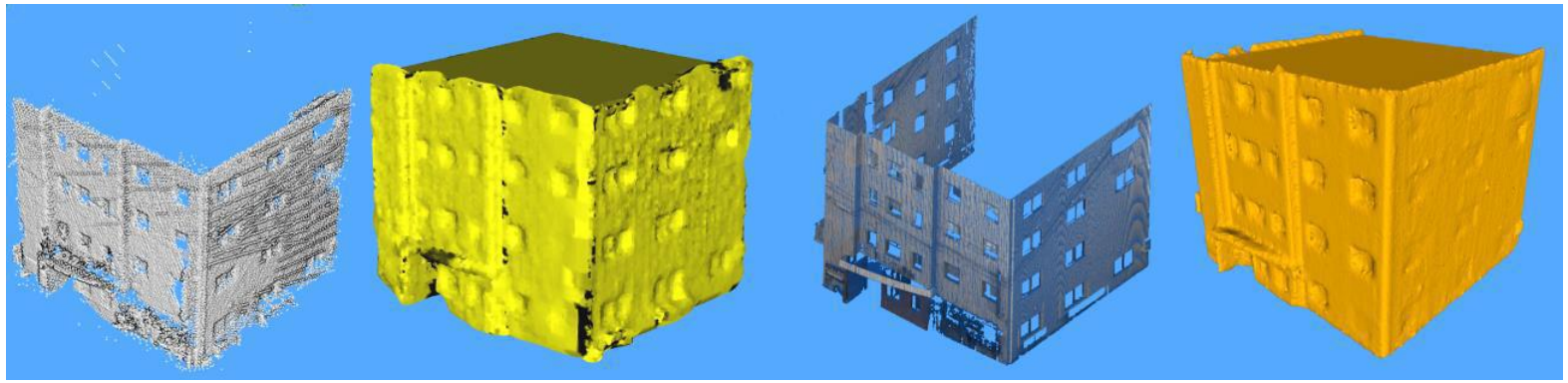


（右）iPhone LiDARでスキャンしたデータ

点群データの追加スキャンによる補間｜概要



車載型LiDARが設置されていない路地（左）を、iPhoneで撮影した点群データ（右）で補間している



（左）車両でスキャンした複数の点群から作成した未処理のポアソンメッシュ（右）iPhoneで撮影した点群データで補間したポアソンメッシュ。両メッシュともLOD2データで欠損部分を埋めている

メッシュの後処理｜概要

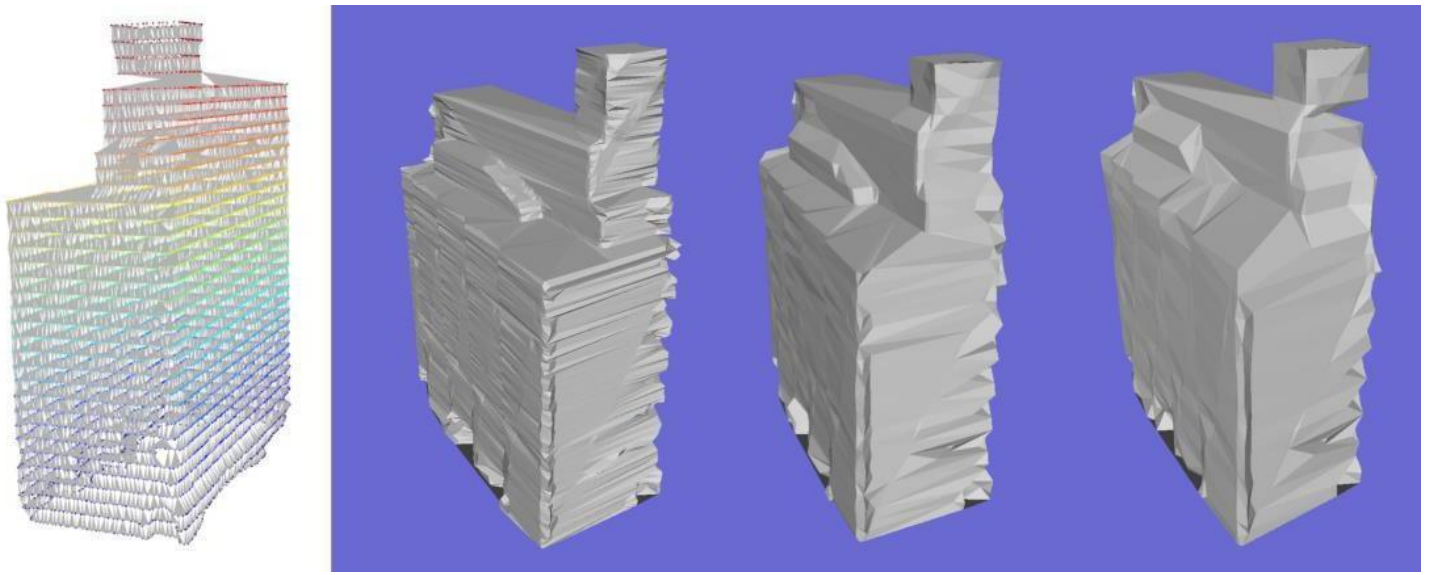
ポアソンメッシュを単純化する方法の一つとして検討したのが、建造物を水平に複数スライスし、各スライスのエッジを接続して立体物を作る手法である。LOD2モデルから建物のフットプリントポリゴンを作成する方法を流用し、建物のポアゾンメッシュを均等・水平にスライスし、それぞれのスライスを垂直方向のエッジで接続して立体を作成した。

この手法のでは、LOD2モデルも同様にスライスを行い、LOD2フットプリントスライスの各頂点に近似のポリゴンスライス頂点を選択し固定することで、ポリゴンスライスの形状がLOD2フットプリントの形状からの乖離を防ぐ点であった。その後、「Ramer-Douglas-Peuckerアルゴリズム（Iterative end-point fitアルゴリズム）」を使用して、頂点間の多角線を統合することで簡素化し、LOD2フットプリントスライスより詳細な形状に近づくよう調整を行った。

等間隔で水平方向のスライスを複数作成し、全体の形状を保ちながら簡略化・積層化して建物の形を再現

メッシュの後処理｜概要

各水平スライス一つ一つについての簡略化は成功したが、積層化の段階で問題が発生した。点群からポアソンメッシュに変換されたときに残るノイズは頂点削減数を100倍化した場合にも部分的に残り、画像のようにビニールで包まれたようなメッシュになってしまった。また、建物の水平部分が正確にメッシュ化されないという問題があり、現実の都市空間に存在するさまざまな建物形状に対応させるには複数の要素の組み合わせが必要であり、本実証ではポリゴンスライスの手法を断念した。



スライスした建物を様々なスライス間隔でリメッシュする最初のテストでは、壁の上のノイズの伸び、ディテールの損失、屋根や垂直でない部分の処理の困難さなど、多くの新しい問題が浮き彫りとなった

窓検出

操作／処理フロー

ステップ①

建物IDごとの点群データを入力し、点密度による窓検出を行う。

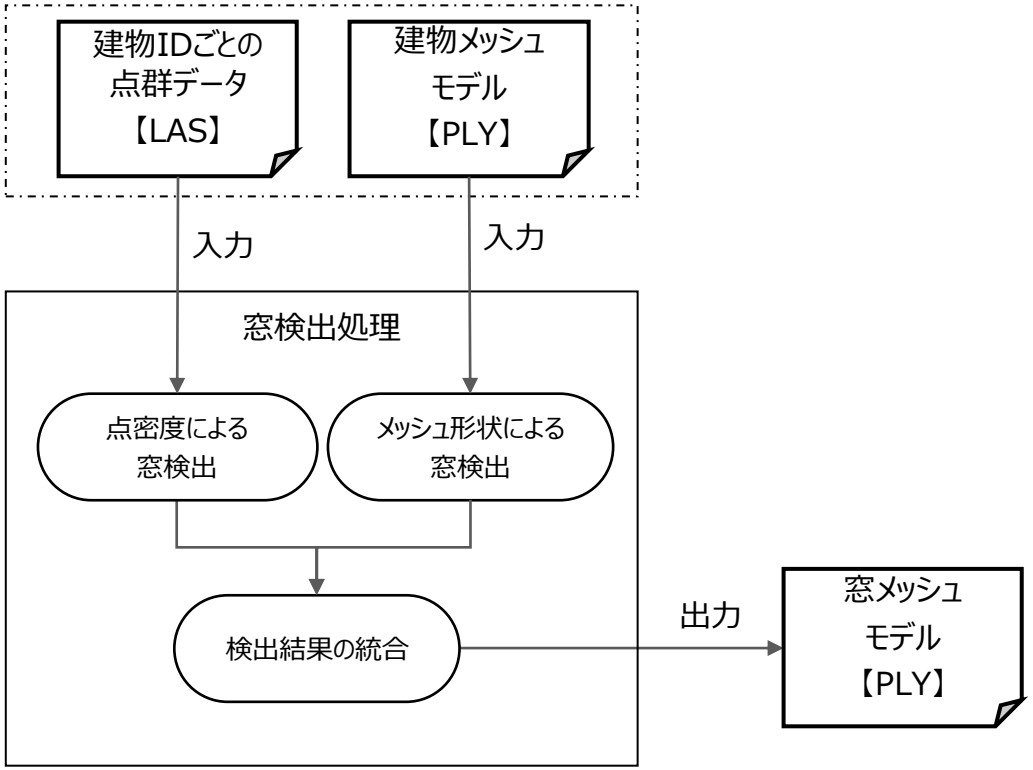
ステップ②

建物メッシュモデルを入力し、メッシュ形状による窓検出を行う。

ステップ③

点密度による窓検出結果と、メッシュ形状による窓検出結果を統合し、窓メッシュモデルを出力する。

システム連携図



マスク処理

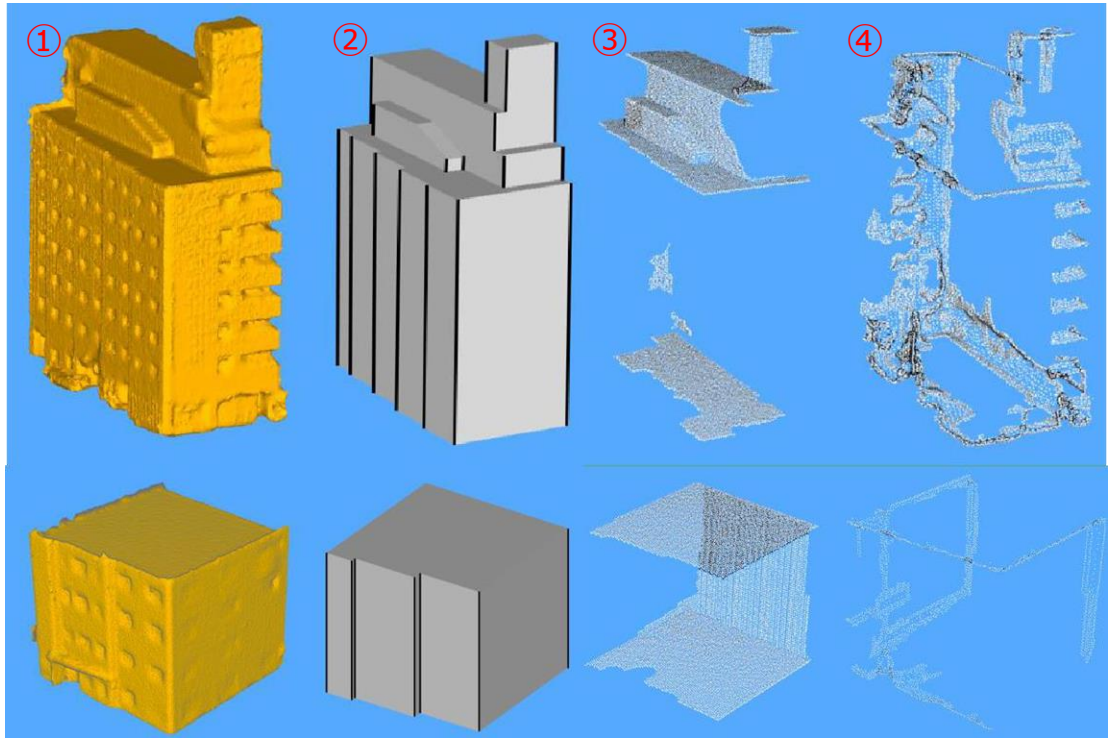
建築物LOD3モデル化に必要な建物の窓検出のための算出用マスク設定

窓の位置・寸法を自動で算出する際、誤認識及び算出処理速度を向上させるため算出範囲から除外するマスクの設定を行った。

第一のマスクは、LOD2モデルの垂直エッジである。LiDAR点群データには様々な量のノイズが含まれており、建物のコーナー付近は窓と似た形状になる場合があり、これを除外する必要があったためである。

第二のマスクは、LOD2モデルを使って点群の欠落を補完しているエリアで、実世界の窓情報を含んでおらず除外の必要があった。

第三のマスクは、取得された点群データの低密度領域である。LiDAR点群の密度に基づき建物メッシュの頂点をクラスタリングすると、低密度の大きなクラスタが1つのマスクに集まることとなる。このため、建物の形状、スキャン角度、オクルージョンによってデータが不十分または不正確に収集されるエリアでは、窓の検出ができなくなる可能性があったためである。



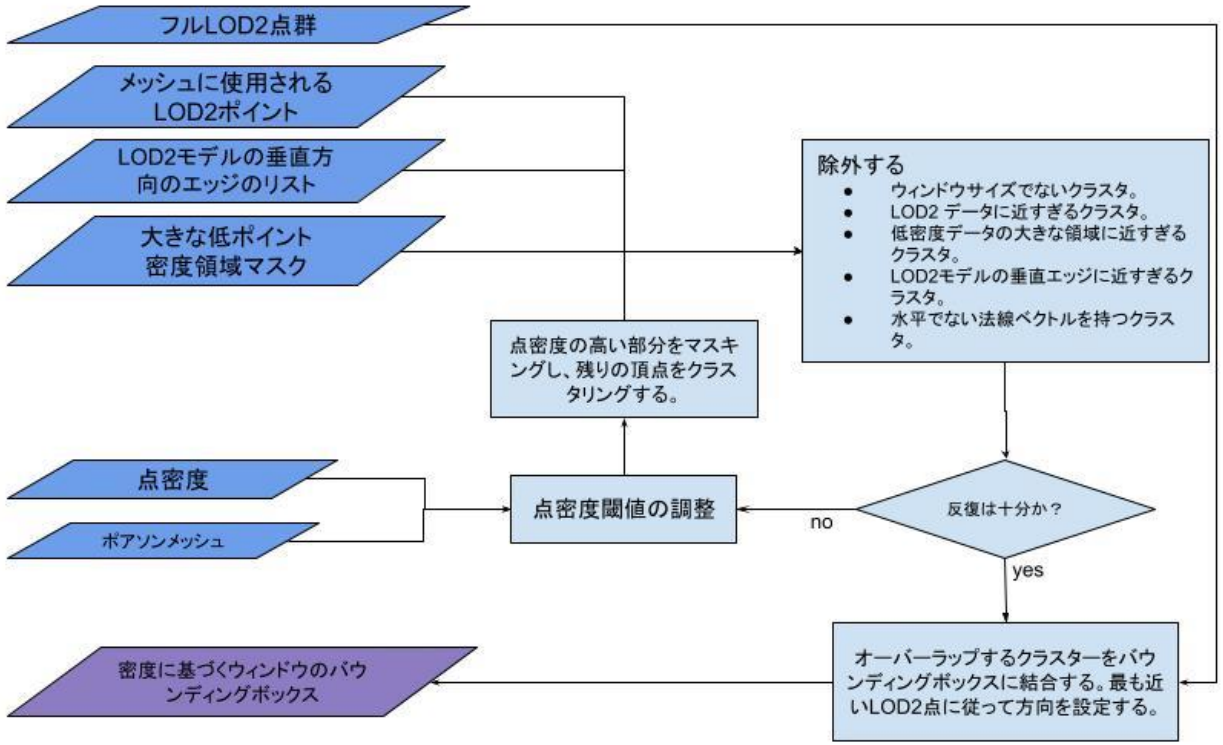
（上）LiDAR点群例。左から右へ、①ポアソンメッシュ②LOD2モデル③LOD2モデルから生成されたエッジ強調を行ったポアソンメッシュ領域④ポアソンメッシュの中で点密度の低い領域

点密度に基づく窓検出①

ポアソン表面再構成（PSR）アルゴリズムを使ってメッシュを作成する場合、出力の1つは密度リストで、局所近傍にどれだけの点があるのかを頂点ごとに記録したものである。

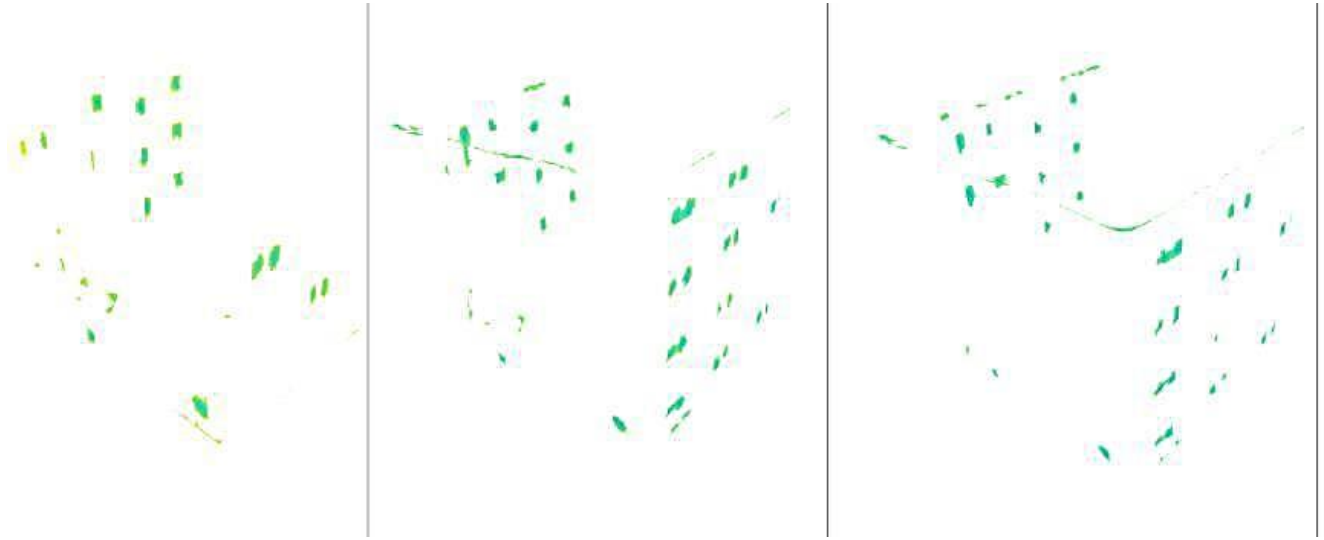
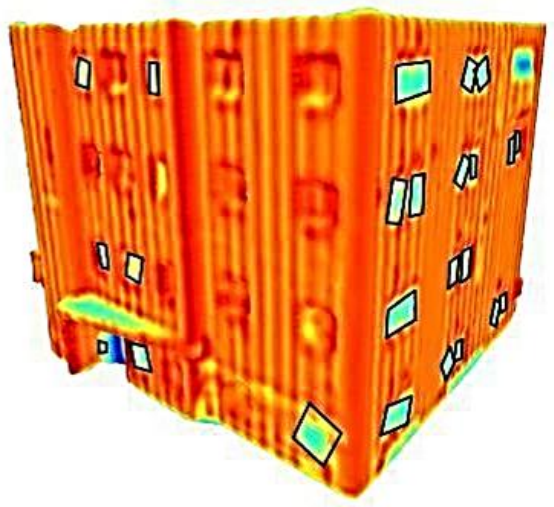
この密度値は、使用したLiDARセンサーの種類、センサーから建物までの距離、オクルージョン、メッシュに変換する前に点群をどの程度ダウンサンプリングしたかによって変化する可能性がある。そのため、密度閾値を複数使用してメッシュをフィルタリングし、点密度が低い窓寸法の群をすべて抽出する必要があった。

前章で説明した事前計算されたマスクに加えて、非水平法線ベクトルを持つ点での潜在的な密度ベースの窓はすべて破棄を行った。



点密度データを用いた窓検出アルゴリズム

点密度に基づく窓検出②

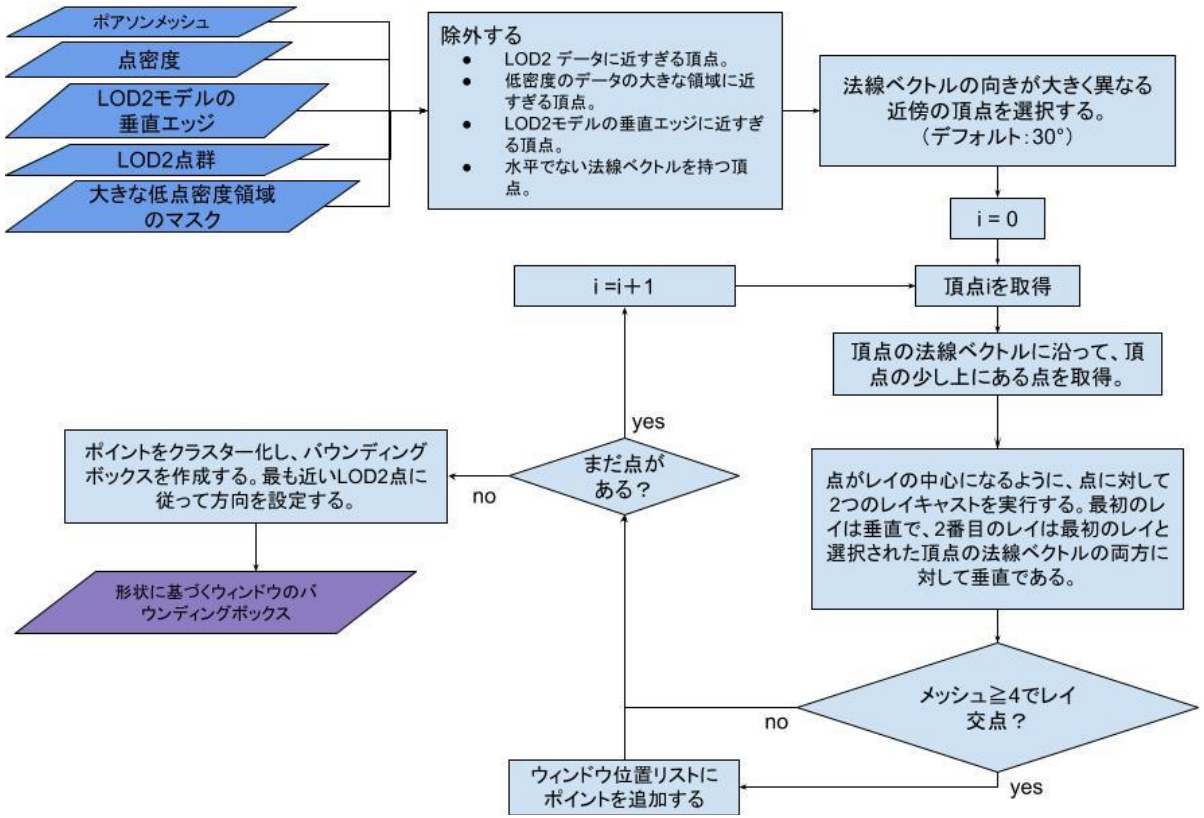
	
ポアソンメッシュの点密度情報を複数の密度閾値と併用し、繰り返しフィルタリングすることで、窓のような大きさと形状を持つ低密度領域を顕在化させることが可能である	検出された窓の形状を強調表示したもの

メッシュ形状に基づく窓検出①

窓の裏側でLiDARのビームが反射し、点密度の低下が見られない場合、窓の後ろからの反射により、メッシュ表面に生じた小さな凹みを検出することで窓を特定することが可能である。

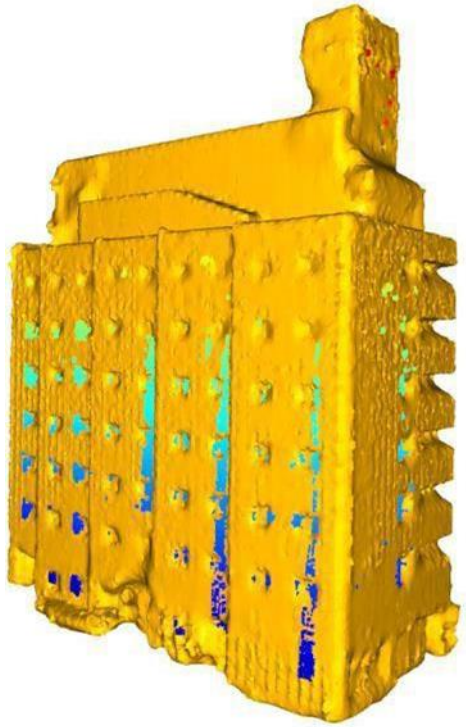
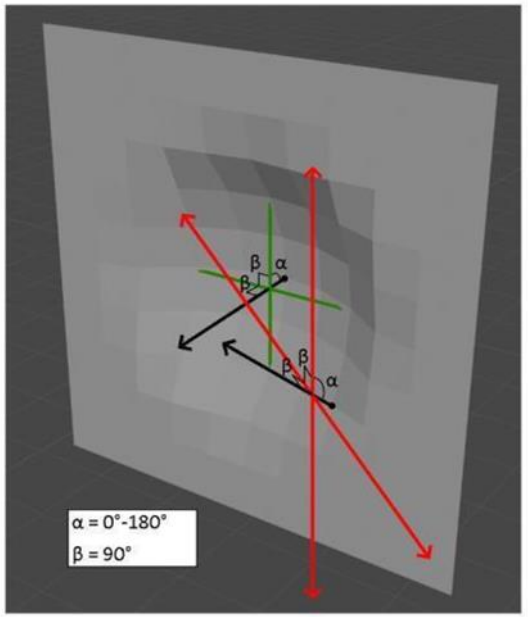
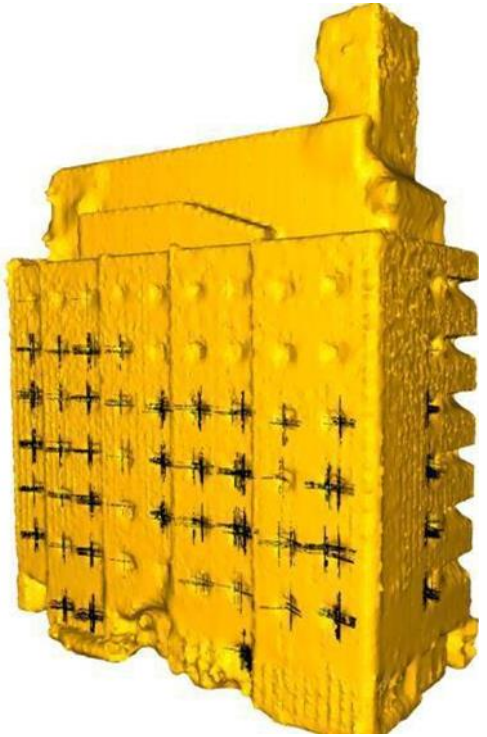
本来、頂点の法線ベクトルの方向を近傍の比較によりメッシュ表面の凹みを検出可能であるが、 LiDARの点群にはノイズが含まれているため、結果として得られる法線ベクトルのフィールドも同様にノイズが多く、より詳細なアプローチが必要となった。

窓の形状の候補は非常に粗い法線ベクトルフィルターを使用して選択され、次に2つの垂直なレイキャストを選択された候補法線にフィッティングする。レイキャストがメッシュ表面を両方向に横切る場合、その位置は窓のような窪みの底にある可能性が高い。



メッシュ形状データを用いた窓検出アルゴリズム

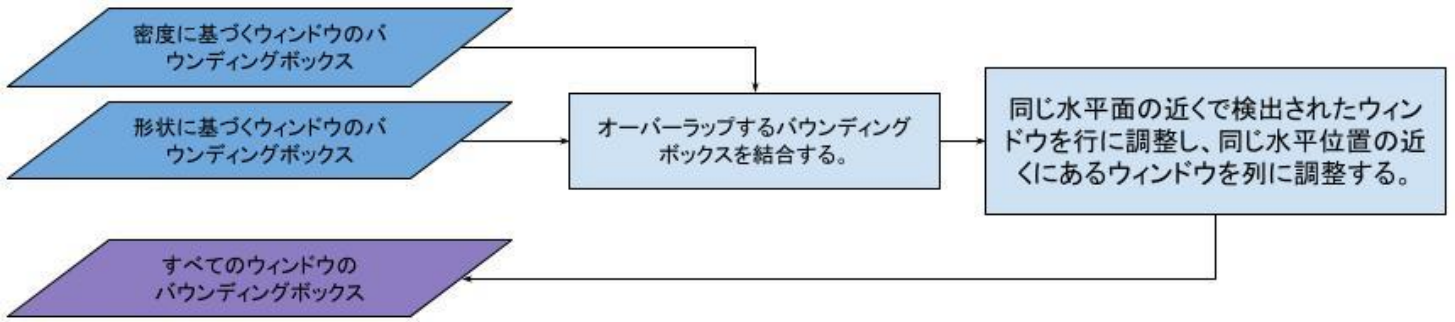
メッシュ形状に基づく窓検出②

		
初期マスキング後の潜在的な形状ベースのウィンドウ位置	1つの受け入れられた テストポイントと1つの拒否されたテストポイントによるレイキャストテストの説明	クラスタリング前のレイキャストチェックにパスした位置

検出結果の統合①

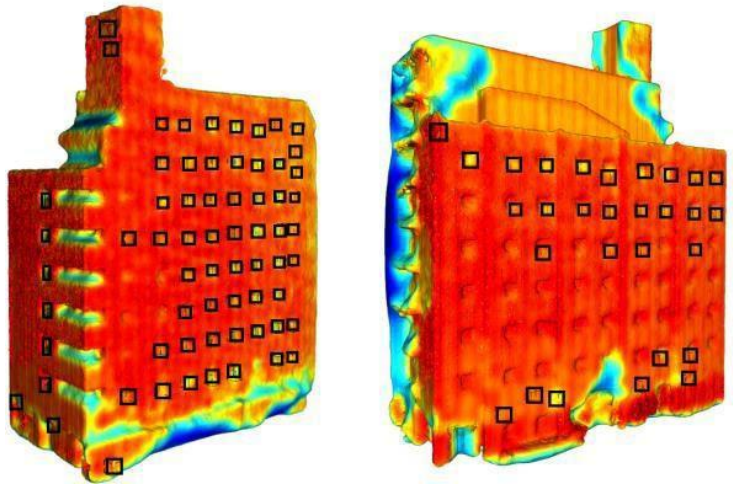
密度ベースで検出された窓と、形状ベースで検出された窓について、その平均的な位置で重なり合うバウンディングボックスを合成した。密度ベースで検出された窓は、LiDARセンサーからの距離によってポイントの密度が大きく変化し、また、窓の後ろにカーテンなどが存在する可能性があるため、正確な位置と寸法にならないことに対処する必要があった。同様に、形状ベースで検出された窓は、メッシュ表面のノイズにより、検出された窓の中心が領域のどこにでも存在する可能性があるため、完全に信頼できるものではないことに留意する必要がある。

検出された窓を単純に結合した場合、一般的にかなり不均一な外観となり、窓の位置、形状、大きさを大まかに推定することしかできない。このため、結合された窓に対し規則化フィルタリングを行った。高さ寸法の近い窓はZ座標の中央値に移動し、縦列の窓はX/Y座標の中央値に移動する。縦方向にも横方向にも他の窓と対にならなかった場合は、誤検出として削除を行った。

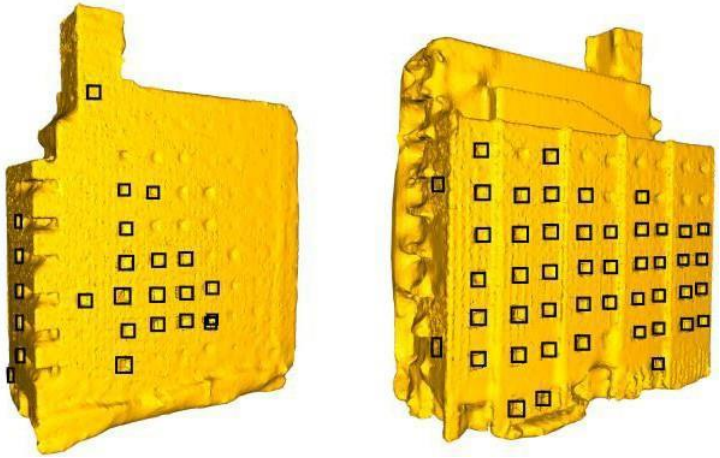


異なるアプローチで検出された窓の組み合わせアルゴリズム

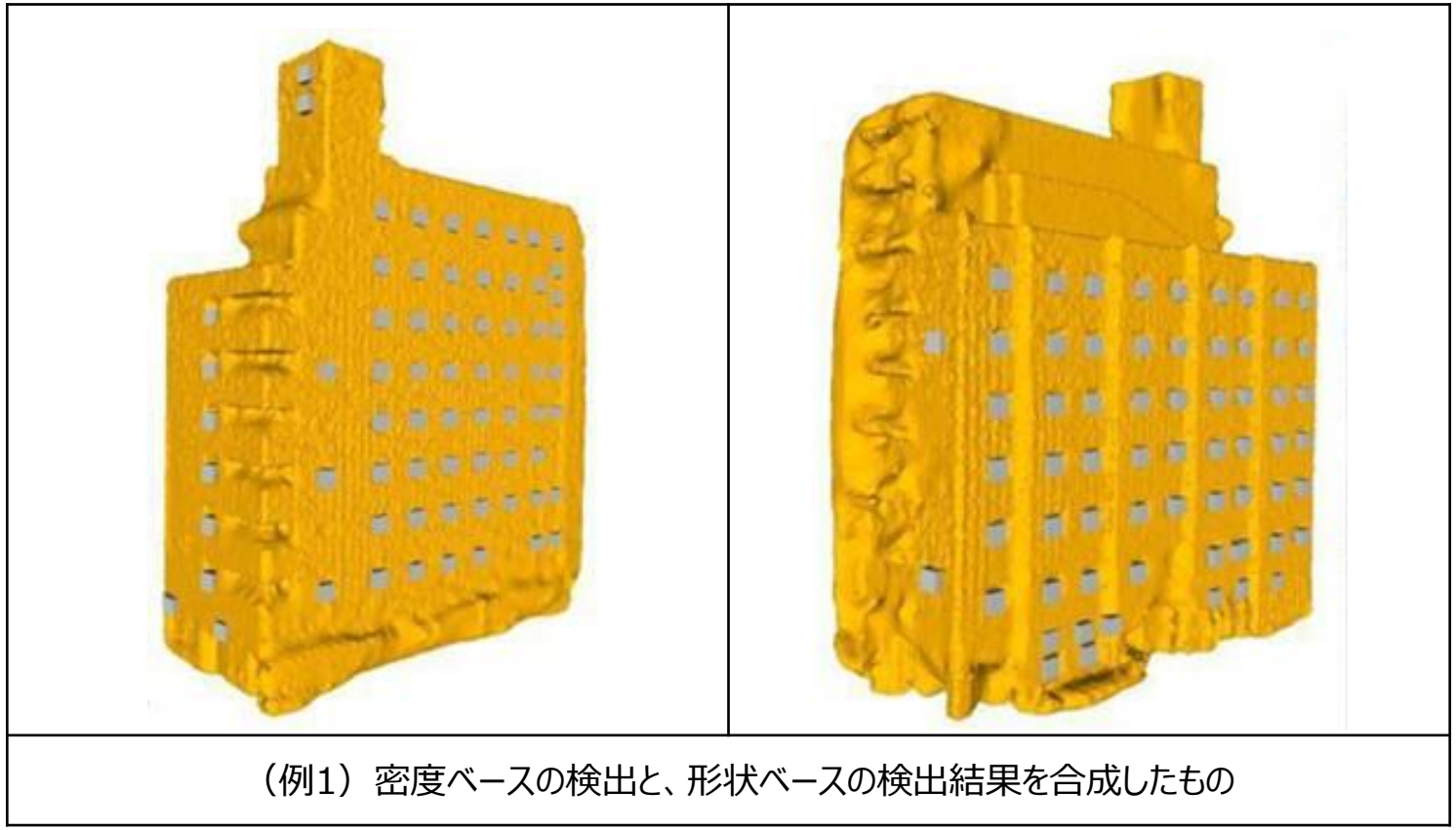
検出結果の統合②



（例1）密度ベースで検出された窓の結果

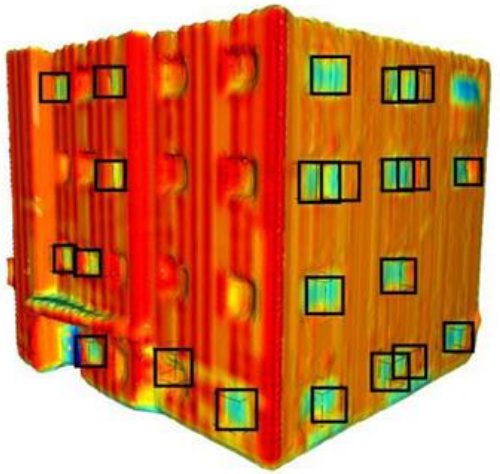


（例1）形状にベースで検出された窓の結果

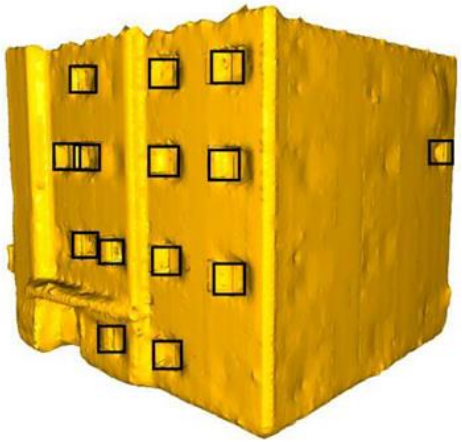


（例1）密度ベースの検出と、形状ベースの検出結果を合成したもの

検出結果の統合③



（例２）密度ベースで検出された窓の結果



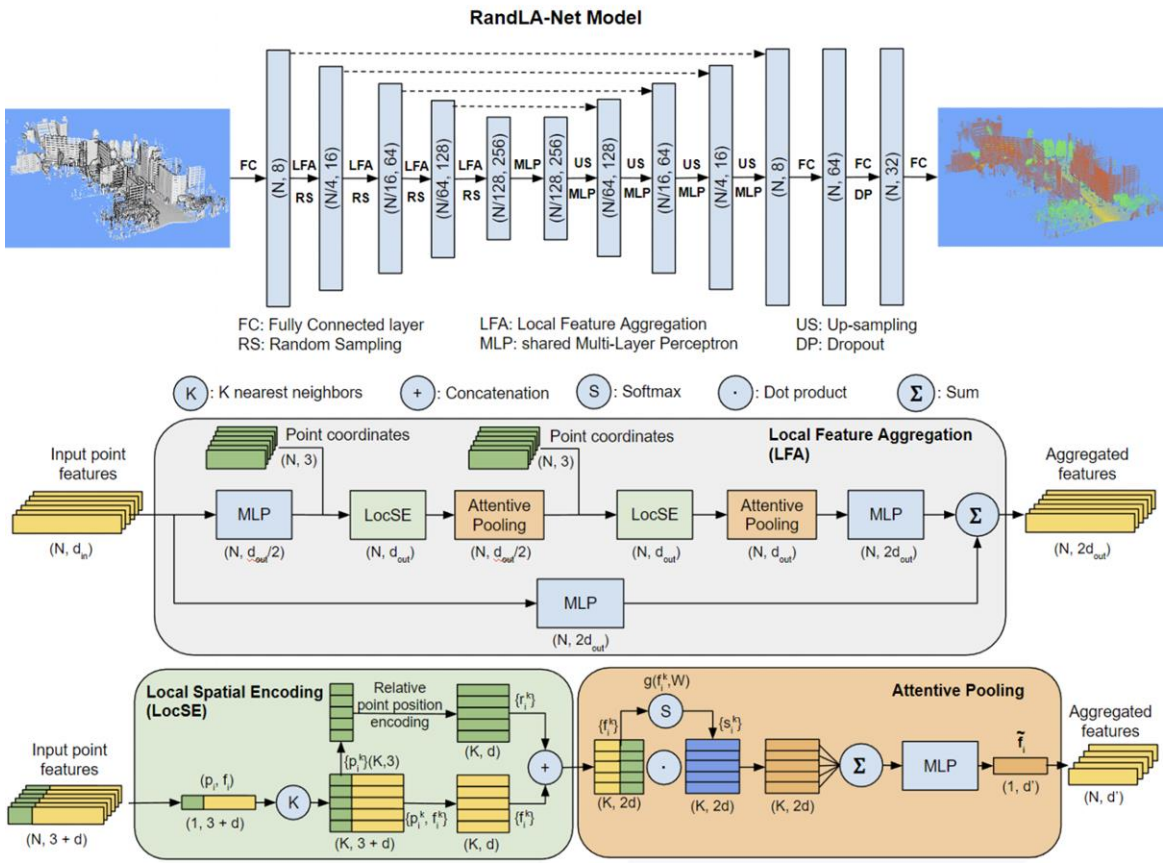
（例２）形状ベースで検出された窓の結果

（例２）密度ベースの検出と、形状ベースの検出結果を合成したもの

RandLA-Netモデル｜主要コード

ディープラーニングによる点群セグメンテーションモデルの実装

```
class RandLANet(BaseModel):
    def __init__(
        cfg = self.cfg

        dim_feature = cfg.dim_feature
        self.fc0 = nn.Linear(cfg.dim_input, dim_feature)
        self.batch_normalization = nn.BatchNorm2d(dim_feature,eps=1e-6,momentum=0.01)
        d_encoder_list = []

        # Encoder
        for i in range(cfg.num_layers):
            name = 'Encoder_layer_' + str(i)
            self.init_dilated_res_block(dim_feature, cfg.dim_output[i], name)
            dim_feature = cfg.dim_output[i] * 2
            if i == 0:
                d_encoder_list.append(dim_feature)
            d_encoder_list.append(dim_feature)
        feature = helper_torch.conv2d(True, dim_feature, dim_feature)
        setattr(self, 'decoder_0', feature)

        # Decoder
        for j in range(cfg.num_layers):
            name = 'Decoder_layer_' + str(j)
            dim_input = d_encoder_list[-j - 2] + dim_feature
            dim_output = d_encoder_list[-j - 2]

            f_decoder_i = helper_torch.conv2d_transpose(True, dim_input,dim_output)
            setattr(self, name, f_decoder_i)
            dim_feature = d_encoder_list[-j - 2]


        f_layer_fc1 = helper_torch.conv2d(True, dim_feature, 64)
        setattr(self, 'fc1', f_layer_fc1)
        f_layer_fc2 = helper_torch.conv2d(True, 64, 32)
        setattr(self, 'fc2', f_layer_fc2)
        f_layer_fc3 = helper_torch.conv2d(False,32,cfg.num_classes,activation=False)
        setattr(self, 'fc', f_layer_fc3)
        self.m_dropout = nn.Dropout(0.5)
```



RandLA-Netモデルの詳細な図解（出典：https://github.com/QingyongHu/RandLA-Net）

RandLA-Netモデル｜主要コード

ディープラーニングによる点群セグメンテーションモデルの実装

```
def init_dilated_res_block(self, dim_input, dim_output, name):
    f_pc = helper_torch.conv2d(True, dim_input, dim_output // 2)
    setattr(self, name + 'mlp1', f_pc)

    self.init_building_block(dim_output // 2, dim_output, name + 'LFA')

    f_pc = helper_torch.conv2d(True,dim_output,dim_output * 2,activation=False)
    setattr(self, name + 'mlp2', f_pc)
    shortcut = helper_torch.conv2d(True,dim_input,dim_output * 2,activation=False)
    setattr(self, name + 'shortcut', shortcut)


def init_building_block(self, dim_input, dim_output, name):
    f_pc = helper_torch.conv2d(True, 10, dim_input)
    setattr(self, name + 'mlp1', f_pc)

    self.init_att_pooling(dim_input * 2, dim_output // 2,name + 'att_pooling_1')

    f_xyz = helper_torch.conv2d(True, dim_input, dim_output // 2)
    setattr(self, name + 'mlp2', f_xyz)

    self.init_att_pooling(dim_input * 2, dim_output, name + 'att_pooling_2')


def init_att_pooling(self, d, dim_output, name):
    att_activation = nn.Linear(d, d)
    setattr(self, name + 'fc', att_activation)

    f_agg = helper_torch.conv2d(True, d, dim_output)
    setattr(self, name + 'mlp', f_agg)
```

基本的な構成要素

- 一連の局所特徴集約ブロックは、計算された局所特徴の畳み込みによって、入力点周辺のより大きな領域を徐々にカバーするように作成されます。

RandLA-Netモデル｜主要コード

ディープラーニングによる点群セグメンテーションモデルの実装

```
class conv2d(nn.Module):
    def __init__(self,
                 batchNorm,
                 in_planes,
                 out_planes,
                 kernel_size=1,
                 stride=1,
                 activation=True):
        super(conv2d, self).__init__()
        self.conv = nn.Conv2d(in_planes,out_planes,kernel_size=kernel_size,stride=stride,padding=(kernel_size - 1) // 2)
        self.biases = self.conv.bias
        self.weights = self.conv.weight
        self.batchNorm = batchNorm
        if self.batchNorm:
            self.batch_normalization = nn.BatchNorm2d(out_planes,momentum=0.01,eps=1e-6)
        if activation:
            self.activation_fn = nn.LeakyReLU(0.2, inplace=True)
        else:
            self.activation_fn = nn.Identity()

    def forward(self, x):
        x = self.conv(x)
        if self.batchNorm:
            x = self.batch_normalization(x)
        x = self.activation_fn(x)
        return x


class conv2d_transpose(nn.Module):
    def __init__(self,
                 batchNorm,
                 in_planes,
                 out_planes,
                 kernel_size=1,
                 stride=1,
                 activation=True):
        super(conv2d_transpose, self).__init__()
        self.conv = nn.ConvTranspose2d(in_planes,out_planes,kernel_size=kernel_size,stride=stride,padding=(kernel_size - 1) // 2)
        self.biases = self.conv.bias
        self.weights = self.conv.weight
        self.batchNorm = batchNorm
        self.batch_normalization = nn.BatchNorm2d(out_planes,momentum=0.01,eps=1e-6)
        if activation:
            self.activation_fn = nn.LeakyReLU(0.2)
        else:
            self.activation_fn = nn.Identity()

    def forward(self, x):
        x = self.conv(x)
        if self.batchNorm:
            x = self.batch_normalization(x)
        x = self.activation_fn(x)
        return x
```

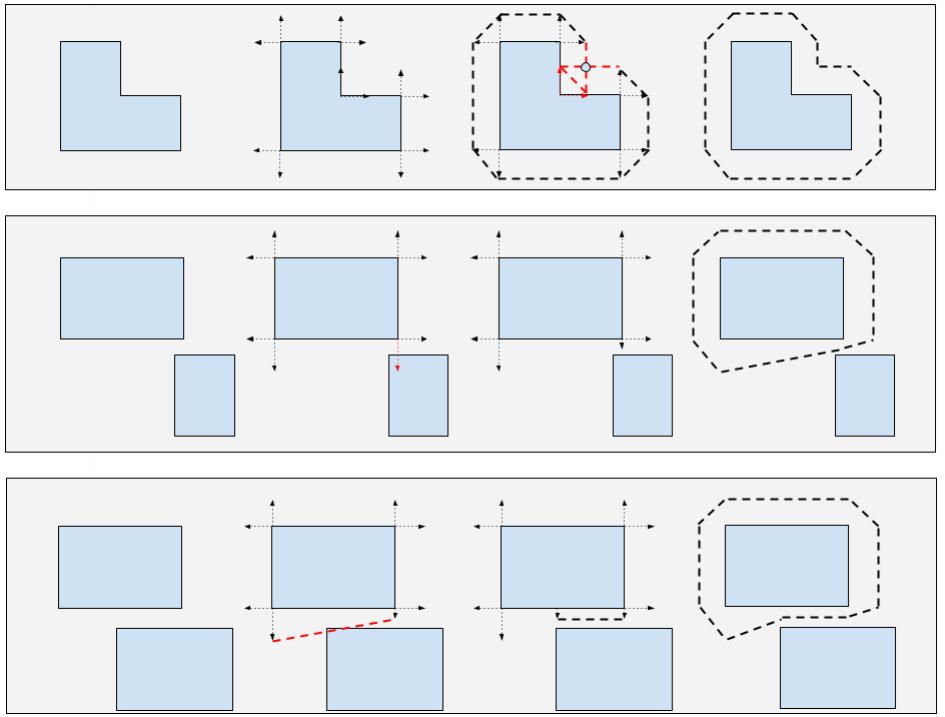
コンボリーションズ

- 下位レベルでは、レイヤー間の特徴伝播と、デコーダ側での分割された点群の再作成に、標準的な2Dコンボリューションが使用される。

2Dフットプリントの作成｜主要コード

LOD2から2Dフットプリントポリゴンへのスライス処理（実装コード、動作部分のみ抜粋）

```
    ...
    model_file = os.path.join(input_path, filename)
    mesh = trimesh.load(model_file, process=False, force="mesh", skip_materials=True, skip_texture=True)

    # Calculate cross section of the mesh that will represent the building footprint
    slice_2D, to_3D = cross_section(mesh)

    # Extract the point values that define the perimeter of the polygon
    poly = slice_2D.polygons_full[0]
    x_coordinates, y_coordinates = poly.exterior.coords.xy
    ...

def cross_section(mesh):
    plane_normal = [0,0,1]
    bounding_box = mesh.bounds
    min_x = bounding_box[0, 0]
    max_x = bounding_box[1, 0]
    min_y = bounding_box[0, 1]
    max_y = bounding_box[1, 1]
    min_z = bounding_box[0, 2]
    middle_x = min_x + ((max_x - min_x)/2.0)
    middle_y = min_y + ((max_y - min_y)/2.0)
    cut_height = min_z + 0.6

    slice = mesh.section(plane_origin=[middle_x, middle_y, cut_height], plane_normal=plane_normal)
    to_2D = trimesh.geometry.align_vectors(plane_normal, [0,0,1])
    slice_2D, to_3D = slice.to_planar(to_2D = to_2D)

    return slice_2D, to_3D
```



（上）拡張した元のフットプリントポリゴン
（中・下）近隣建物を考慮した境界再算出

ICPによる点群合成｜主要コード

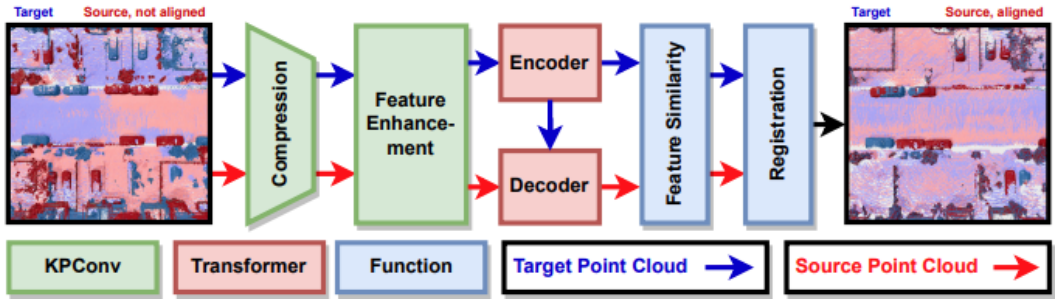
点群点群合成処理（実装コード、動作部分のみ抜粋）

```
def align_point_clouds( inside_points_list ):
    unaligned_lidar_point_clouds = []
    aligned_lidar_point_clouds = []
    downsampled_lidar_point_clouds = []

    for point_cloud_num in range(len(inside_points_list)):
        point_cloud_data = inside_points_list[point_cloud_num]
        points = point_cloud_data[0]
        point_cloud = o3d.geometry.PointCloud()
        point_cloud.points = o3d.utility.Vector3dVector(points)
        downsampled_point_cloud = o3d.geometry.PointCloud.voxel_down_sample(point_cloud, 0.2)
        aligned_lidar_point_clouds.append(point_cloud)
        unaligned_lidar_point_clouds.append( copy.deepcopy(point_cloud) )
        downsampled_lidar_point_clouds.append(downsampled_point_cloud)

    lidar_matching_max_distance = 5.0 # Very large matching distance allows for fixing large position errors, pretty safe to use, when combined with low maximum error for accepting the resul
    transformation_init_guess = np.identity(4)

    # Next, combine every point cloud other than one, and adjust that one cloud towards the whole of the rest, if the error of adjusted point cloud is small enough
    for moving_num in range(len(downsampled_lidar_point_clouds)):
        downsampled_moving_point_cloud = downsampled_lidar_point_clouds[moving_num]
        static_point_cloud = None
        for static_num in range(len(downsampled_lidar_point_clouds)):
            if static_num == moving_num:
                continue
            if static_point_cloud == None:
                static_point_cloud = downsampled_lidar_point_clouds[static_num]
            else:
                static_point_cloud = static_point_cloud + downsampled_lidar_point_clouds[static_num]
        alignment_result = refine_registration_point_to_point(downsampled_moving_point_cloud, static_point_cloud, transformation_init_guess, lidar_matching_max_distance, 200)
        if alignment_result.inlier_rmse < 0.25:
            print("  Point cloud #{num1}, error={num2}, adjusting pose".format(num1=moving_num, num2=alignment_result.inlier_rmse))
            aligned_lidar_point_clouds[moving_num].transform(alignment_result.transformation)
            downsampled_lidar_point_clouds[moving_num].transform(alignment_result.transformation)
        else:
            print("  Point cloud #{num1}, error={num2}, skipping.".format(num1=moving_num, num2=alignment_result.inlier_rmse))
    print("Matching process completed.")
    return aligned_lidar_point_clouds, unaligned_lidar_point_clouds

def refine_registration_point_to_point(moving, reference, initial_transformation, distance_threshold=1.0, max_iteration=100):
    result = o3d.pipelines.registration.registration_icp( moving, reference, distance_threshold, initial_transformation,
                                                          o3d.pipelines.registration.TransformationEstimationPointToPoint(),
                                                          o3d.pipelines.registration.ICPConvergenceCriteria(max_iteration=max_iteration))
    return result
```

DCPCRモデル｜主要コード

DCPCRによる点群点群合成処理（実装コード、動作部分のみ抜粋）

```
class RegisterNet(nn.Module):
    def __init__(self, tau=1,
                 weighting='information_gain',
                 input_transform=False,
                 nr_attn_blocks=0,
                 attention_normalization='softmax',
                 kp_radius=0.05,
                 nr_kp_blocks=0,
                 radial=False
                 ):
        super().__init__()
        self.point_net = blocks.PointNetFeat(
            in_dim=6, out_dim=256, input_transform=input_transform, norm=True)

        self.conf = ConvNet(in_channels=256, out_channels=256,
                            radius=kp_radius, num_layer=nr_kp_blocks, radial=radial)

        self.transformer = Transformer(
            d_model=256,
            num_layer=nr_attn_blocks,
            dim_feedforward=512,
            dropout=0)

        self.attention = blocks.Attention(
            tau, attention_normalization=attention_normalization)
        self.weighting = blocks.CorrespondenceWeighter(
            weighting=weighting)
        self.registration = blocks.SVDRegistration()
```



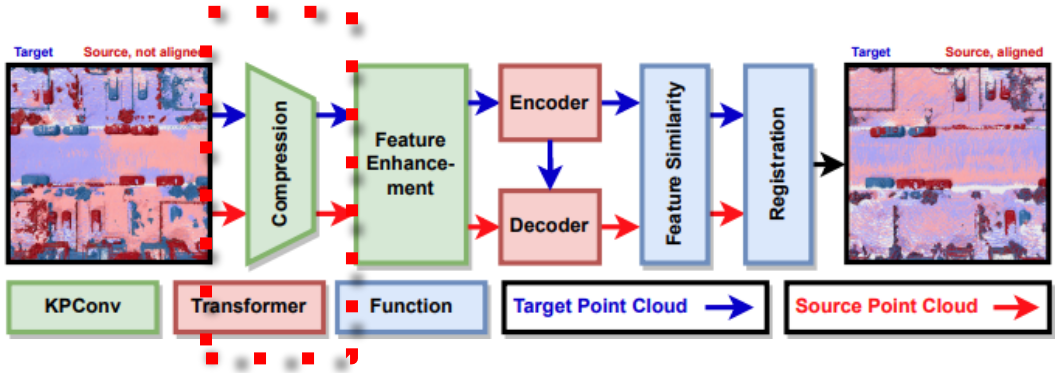
DCPCRモデル｜主要コード

DCPCRによる点群点群合成処理（実装コード、動作部分のみ抜粋）

```python
class PointNetFeat(nn.Module):
    def __init__(self, in_dim=3, out_dim=1024, input_transform=True, feature_transform=False, norm=True):
        super(PointNetFeat, self).__init__()
        self.conv1 = nn.Linear(in_dim, 64)
        self.conv2 = nn.Linear(64, 128)
        self.conv3 = nn.Linear(128, out_dim)
        self.bn1 = nn.LayerNorm(64) if norm else nn.Identity()
        self.bn2 = nn.LayerNorm(128) if norm else nn.Identity()
        self.bn3 = nn.LayerNorm(out_dim) if norm else nn.Identity()
        self.feature_transform = feature_transform
        self.input_transform = input_transform
        if self.feature_transform:
            self.fstn = STNkd(k=64, norm=norm)
        if self.input_transform:
            self.stn = STNkd(k=in_dim, norm=norm)
```

圧縮ネットワーク

- PointNetは、圧縮された点表現を、マッチングに適したローカライゼーション固有の特徴空間に変換するために使用されます。



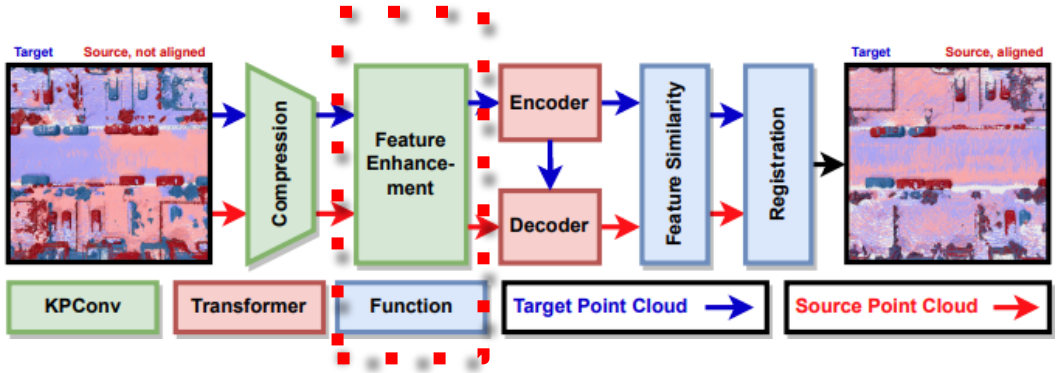
DCPCRモデル｜主要コード

DCPCRによる点群点群合成処理（実装コード、動作部分のみ抜粋）

```
class ResnetKPConv(nn.Module):
    def __init__(self, in_channels, out_channels, radius, kernel_size=3, KP_extent=None, p_dim=3, radial=False, f_dscale=2):
        super().__init__()

        self.ln1 = nn.LayerNorm(in_channels)
        self.relu = nn.LeakyReLU()

        self.kpconv = KPConv(in_channels=in_channels,
                             out_channels=out_channels//f_dscale,
                             radius=radius,
                             kernel_size=kernel_size,
                             KP_extent=KP_extent,
                             p_dim=p_dim,
                             radial=radial)

        self.ln2 = nn.LayerNorm(out_channels//f_dscale)
        self.lin = nn.Linear(out_channels//f_dscale, out_channels)

        self.in_projection = nn.Identity() if in_channels == out_channels else nn.Linear(
            in_channels, out_channels)
```

フィーチャーエンハンスメントネットワーク

- 圧縮ネットワークからの特徴量は局所的な情報しか含まないため、KPConv（スパースポイントコンボリューション演算子）を用いて半径*r*以内の点データを集約している。



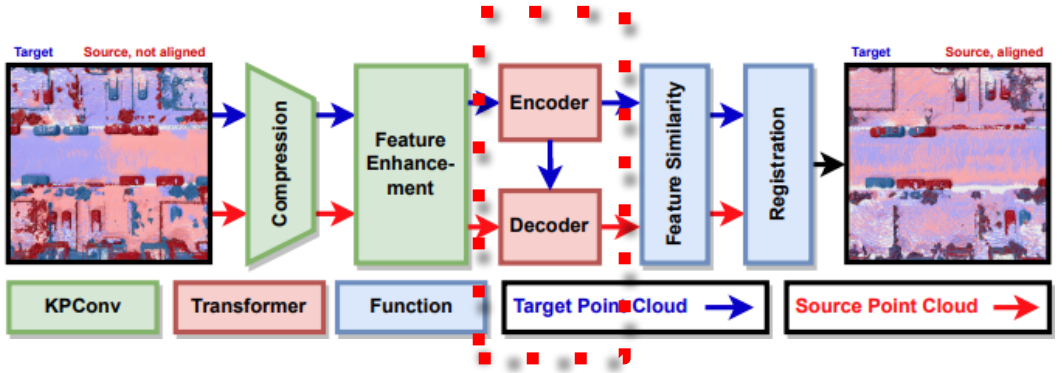
DCPCRモデル｜主要コード

DCPCRによる点群点群合成処理（実装コード、動作部分のみ抜粋）

```
class Transformer(nn.Module):
    def __init__(self,
                 d_model,
                 num_layer=2,
                 nhead=8,
                 dim_feedforward=512,
                 dropout=0
                 ):
        super().__init__()
        enc = nn.TransformerEncoderLayer(d_model=d_model,
                                         nhead=nhead,
                                         dim_feedforward=dim_feedforward,
                                         dropout=dropout,
                                         batch_first=True,
                                         norm_first=True)
        self.enc = nn.TransformerEncoder(enc, num_layers=num_layer)

        dec = nn.TransformerDecoderLayer(d_model=d_model,
                                         nhead=nhead,
                                         dim_feedforward=dim_feedforward,
                                         dropout=dropout,
                                         batch_first=True,
                                         norm_first=True)
        self.dec = nn.TransformerDecoder(dec, num_layers=num_layer)
        self.num_layer = num_layer
```

トランスフォーマー

- グローバルな特徴集約のためのマルチヘッド自己アテンション機構にトランスフォーマーが使用されています。



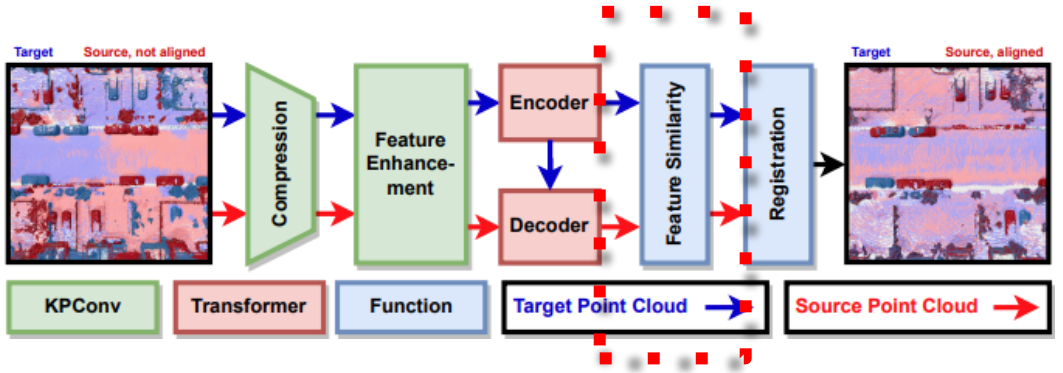
DCPCRモデル｜主要コード

DCPCRによる点群点群合成処理（実装コード、動作部分のみ抜粋）

```python
class CorrespondenceWeighter(nn.Module):
    def __init__(self, weighting='max'):
        super().__init__()
        self.weighting = weighting
        if weighting == 'topk':
            self.top_k = 20  # FIXME: make parameter
            self.model = nn.Sequential(
                nn.Linear(20, 64),
                nn.LeakyReLU(),
                nn.Linear(64, 64),
                nn.LeakyReLU(),
                nn.Linear(64, 1),
                nn.Sigmoid()
            )

        self.weight_fkt = eval('self.'+weighting)  # function name
```

特徴の類似性

- 登録に適したポイントに集中するために、対応関係の品質を推定することを学習します。



DCPCR/GICPによる異動判読｜主要コード

グローバル（DCPCR）と精緻化（GICP）レジストレーション処理（実装コード、動作部分のみ抜粋）

```
    ...
    # Get input point clouds and store them in tensors
    data_source  = torch.tensor(data_source, device=dev).float()
    data_target  = torch.tensor(data_target, device=dev).float()

    with torch.no_grad():
        # Input the source and target pointcloud to DCPCR and get the estimated pose
        est_pose, _, _, _ = model(data_target, data_source)
        if fine_tune:
            # Use GICP for pose refinement
            init_guess = est_pose.detach().cpu().squeeze().numpy()
            est_pose = fine_tuner.refine_registration( geom_source, geom_target, init_guess, distance_threshold=voxel_size*5)
            est_pose = torch.tensor(est_pose, device=dev, dtype=torch.float)

        # Transform the source pointcloud with the predicted transformation matrix (rotation and translation)
        ps_t = transform(data_source, est_pose, device=dev)
    ...


def refine_registration(source, target, initial_guess, distance_threshold=0.8):
    # Run GICP with a distance threshold defining registration convergence
    kernel_threshold = 0.3 * distance_threshold
    robust_kernel = o3d.pipelines.registration.GMLoss(kernel_threshold)
    gcp = o3d.pipelines.registration.TransformationEstimationForGeneralizedICP(robust_kernel)
    result = o3d.pipelines.registration.registration_generalized_icp(source, target, distance_threshold, initial_guess, gcp)
    return result.transformation
```

iPSRによるメッシュ再構築｜主要コード

スクリプトからiPSR実行ファイルを呼び出す（実装コード、動作部分のみ抜粋）

```python
# Read data
input_file = os.path.join( input_path, filename )
las = laspy.read(input_file)
xyz = np.vstack([las.x, las.y, las.z]).transpose()
point_cloud = o3d.geometry.PointCloud()
point_cloud.points = o3d.utility.Vector3dVector(xyz)

# Center data
center = point_cloud.get_center()
print("  offset:", center)
point_cloud.translate(-center)
o3d.io.write_point_cloud("tmp_ipsr_input.ply", point_cloud, write_ascii=False, compressed=False, print_progress=True)

# Run iPSR algorithm
print("Launching iPSR algorithm in a separate process...")
process = subprocess.run(["ipsr.exe", "--in", "tmp_ipsr_input.ply", "--out", "tmp_ipsr_output.ply", "--depth", "8", "--iters", "25", "--pointWeight", "6", "--neighbors", "16"])
print("running...")
print("Process returned:", process.returncode)

# Read the output, and move it back to the original position, then save
print("Preparing the output...")
mesh = o3d.io.read_triangle_mesh("tmp_ipsr_output.ply")
mesh.translate(center)
o3d.io.write_triangle_mesh(output_file, mesh)

# Delete the temporary files
print("Cleaning up...")
os.remove("tmp_ipsr_input.ply")
os.remove("tmp_ipsr_output.ply")
```

窓検出アルゴリズムの一般化

本実証での窓検出アルゴリズムには大きな課題が存在する。前章で使用した2つの例を比較すると、下に示した図のように最高のパフォーマンスを達成するために必要なパラメータは両者でかなり異なっていた。使用法をシンプルにするための工夫も必要であり、最終的には、建物の種類、センサー、スキャン距離、ノイズの状況などに応じていくつかの設定のテンプレートを用意し、状況に合わせて選択することが必要になる。

```
# MMS scan of Hotel Central Sendai
window_detection_use_radius_search                                = False
window_detection_large_low_density_area_threshold                 = 0.1
window_detection_density_threshold                                = 0.3
window_detection_density_minimum_distance_to_lod2_areas           = 1.0
window_detection_density_minimum_distance_to_large_low_density_areas = 1.0
window_detection_density_cluster_together_distance                = 1.5
window_detection_shape_depth                                      = 0.2
window_detection_shape_cluster_together_distance                  = 1.5
window_detection_shape_minimum_distance_to_lod2_vertical_edges    = 1.0
window_detection_shape_minimum_distance_to_lod2_areas             = 1.5
window_detection_shape_minimum_distance_to_large_low_density_areas = 1.5
```

```
# High-quality scan of Kitakawa Clinic
window_detection_use_radius_search                                = True
window_detection_large_low_density_area_threshold                 = 0.15
window_detection_density_threshold                                = 0.2
window_detection_density_minimum_distance_to_lod2_areas           = 0.5
window_detection_density_minimum_distance_to_large_low_density_areas = 0.5
window_detection_density_cluster_together_distance                = 0.5
window_detection_shape_depth                                      = 0.17
window_detection_shape_cluster_together_distance                  = 1.5
window_detection_shape_minimum_distance_to_lod2_vertical_edges    = 0.5
window_detection_shape_minimum_distance_to_lod2_areas             = 1.0
window_detection_shape_minimum_distance_to_low_density_areas      = 1.0
```

2つの例で設定したパラメータの違い

Ⅰ．実証概要

Ⅱ．実証技術の概要

Ⅲ．技術調査

Ⅳ．実証システム

Ⅴ．実証システムの検証

Ⅵ．成果と課題

実証フロー

点群データを取得し、実証システムの自動更新処理の精度の検証を行う実証フローを以下に示す

実証フロー

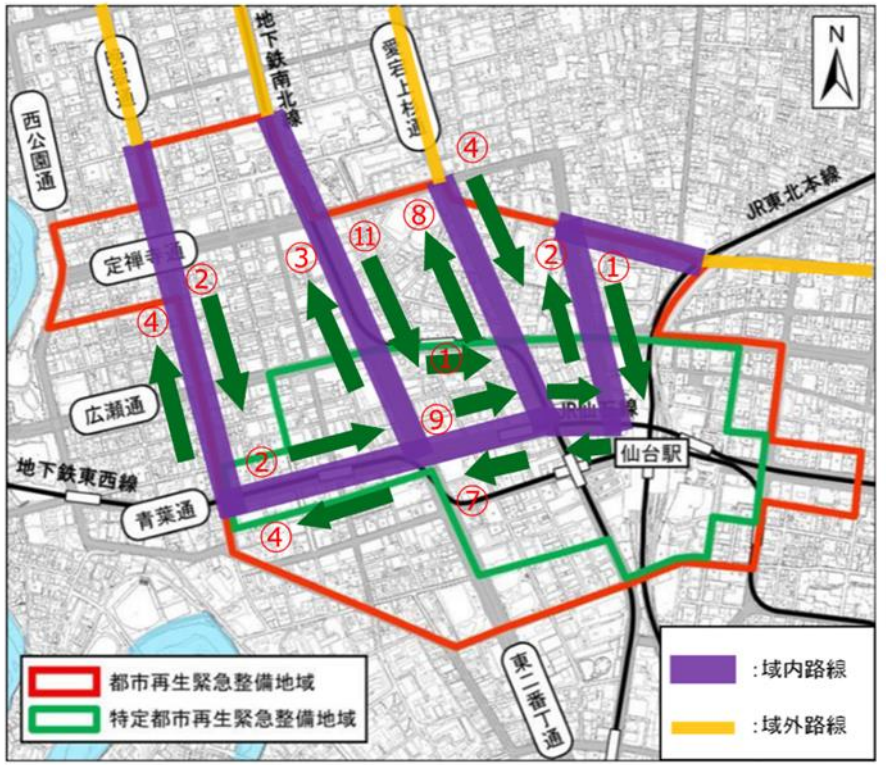
3次元点群データ取得及び作成	• 公共交通車両への簡易MMS（LiDARセンサー）取付 • バス及びタクシーを想定した一般車両による3次元点群データの取得・作成 • iPhone LiDARセンサーを用いた、運行ルート外の点群データの取得・作成
3D都市モデルの自動更新検証	• MMS/LiDAR点群データを用いた、自動更新処理の精度の検証 • 精度向上、不具合対応のイテレーション開発 • ・3D都市モデルの自動更新の検証デモンストレーション開催
評価検証	• 作成された3D都市モデル（LOD3）の評価 • A.I.を用いた3D都市モデルの自動更新手法の有用性を検証する

簡易MMSによる3次元点群データの取得検証①

宮城県仙台市の都市再生緊急整備地域内の宮城交通バス運行ルートを対象として3次元点群データ取得及び作成を実施した

期間中、仙台駅を目的地とした運行があり、当該ルートを複数回データを取得した（計測期間：2022年9月21日～10月5日）

システム取り付け確認の様子



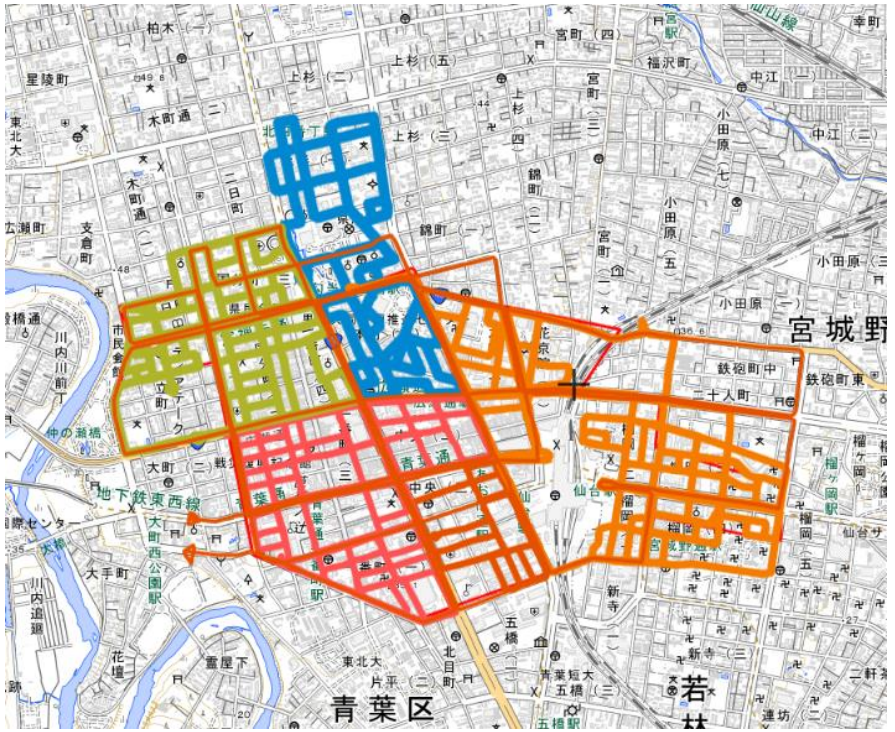
バス走行ルート

簡易MMSによる3次元点群データの取得検証②

宮城県仙台市の都市再生緊急整備地域内でタクシーを想定した一般車両による3次元点群データ取得及び作成を実施した

期間中、仙台駅を目的地とした運行があり、当該ルートを複数回データを取得した（計測期間：2022年8月17日）

タクシーを想定した使用車両

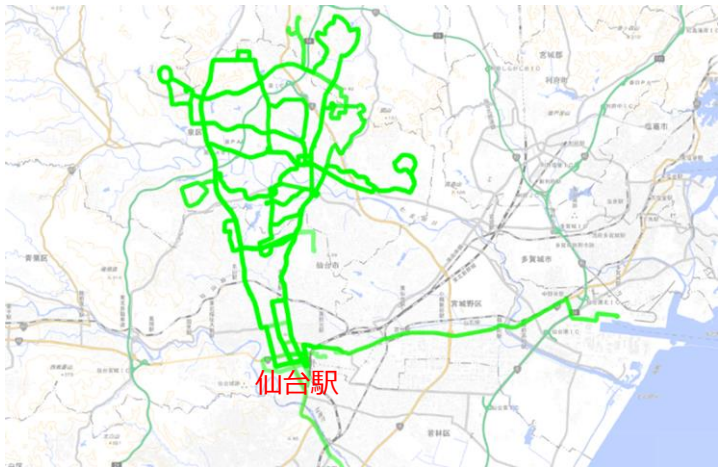


車両走行ルート（色：1回の走行で走る単位）

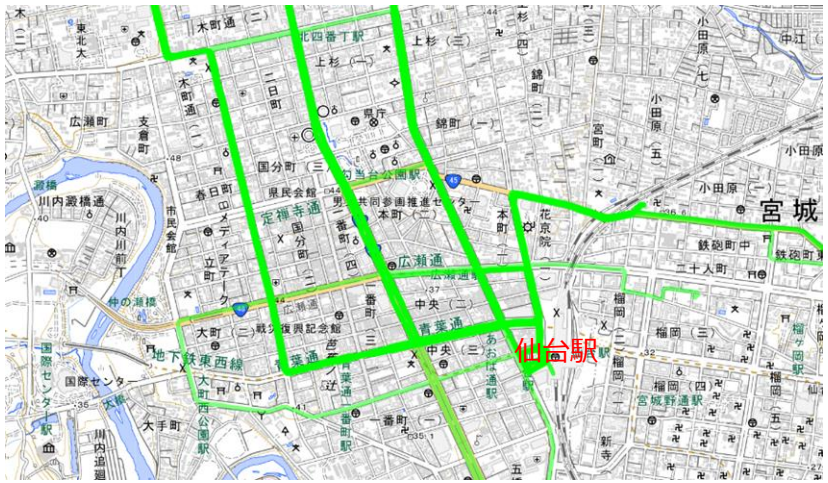
簡易MMSによる3次元点群データの取得検証③

測定した走行ルート：拠点となる仙台駅周辺と泉区の経路

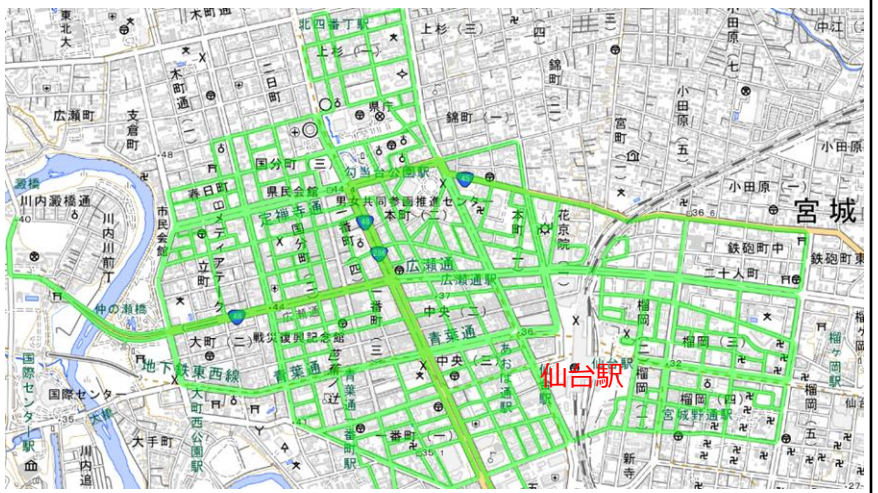
宮城県　仙台市　　バス及び一般車両の走行ルート（約10km、186ヘクタール内）



バスの走行ルート



バスの走行ルート（拡大）

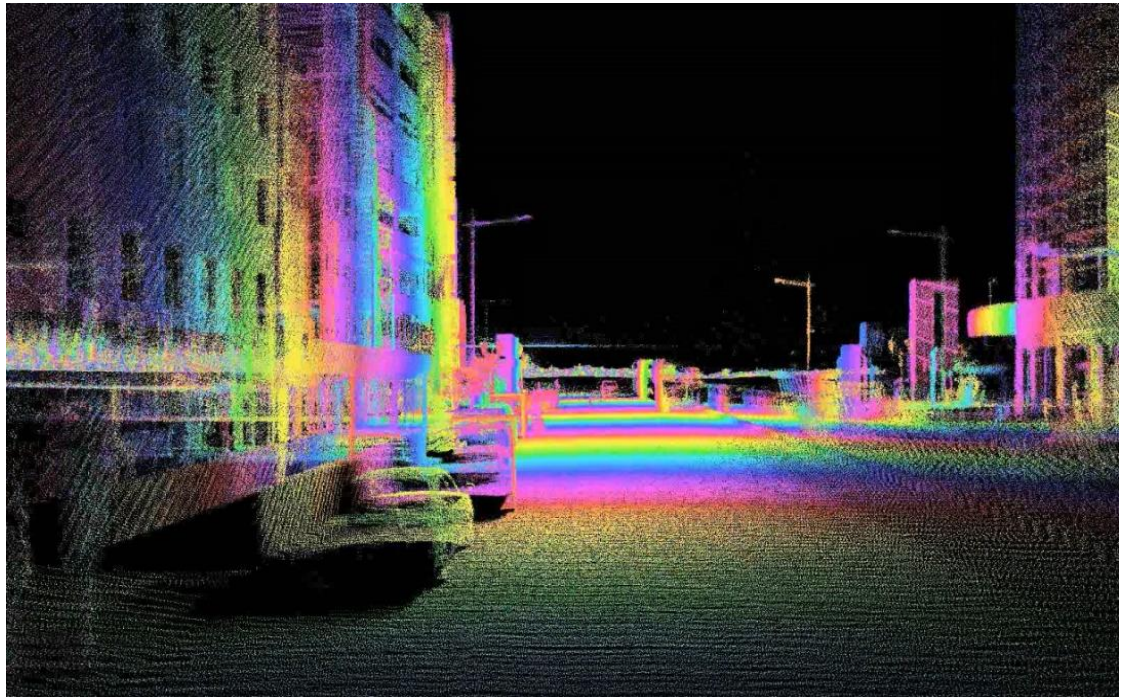


一般車両の走行ルート

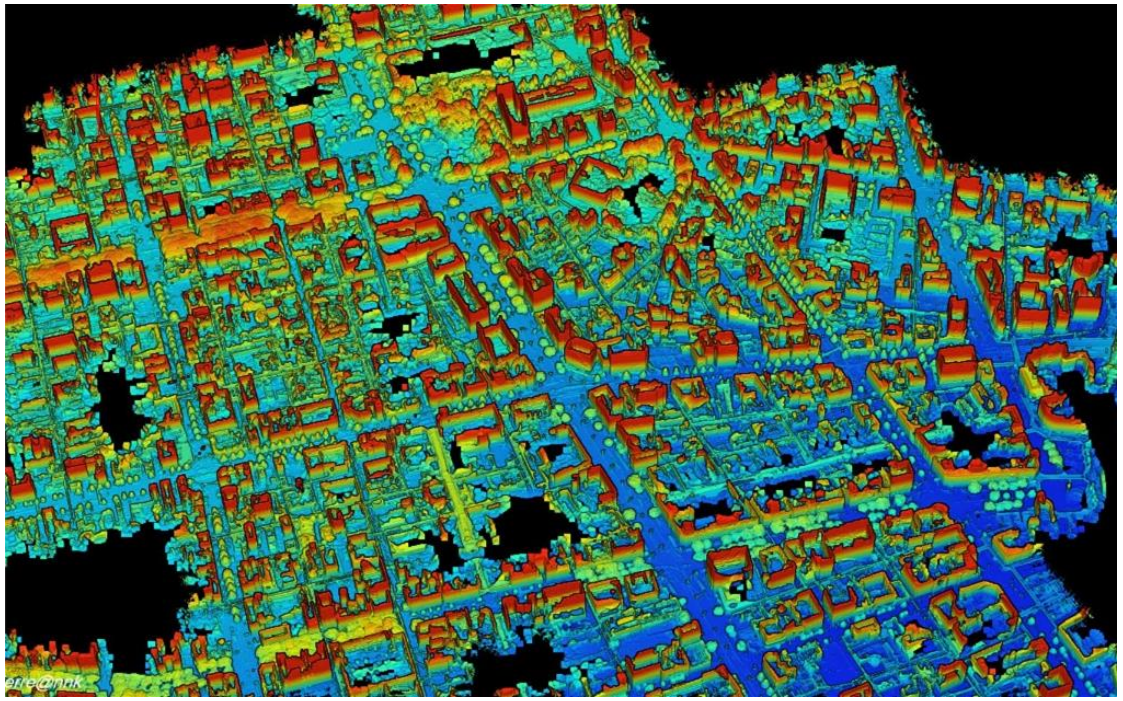
簡易MMSによる3次元点群データの取得検証④

撮影された点群

バス及びタクシーを想定した一般車両に取り付けたLiDARセンサーで撮影された点群データ



バスで取得したデータ

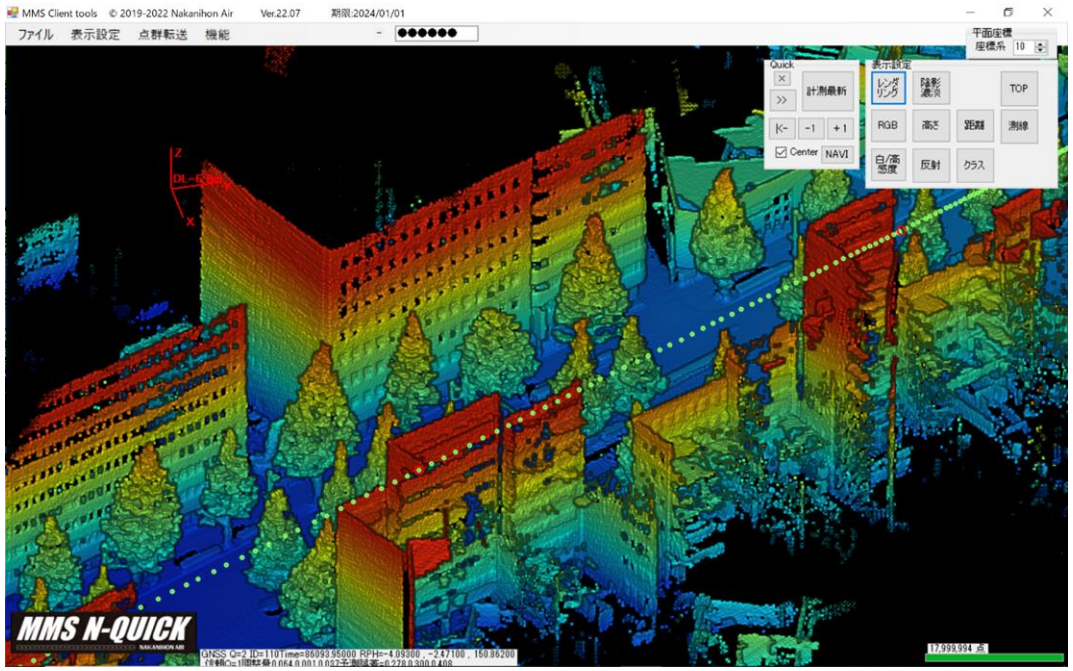


タクシーを想定した一般車両で取得したデータ

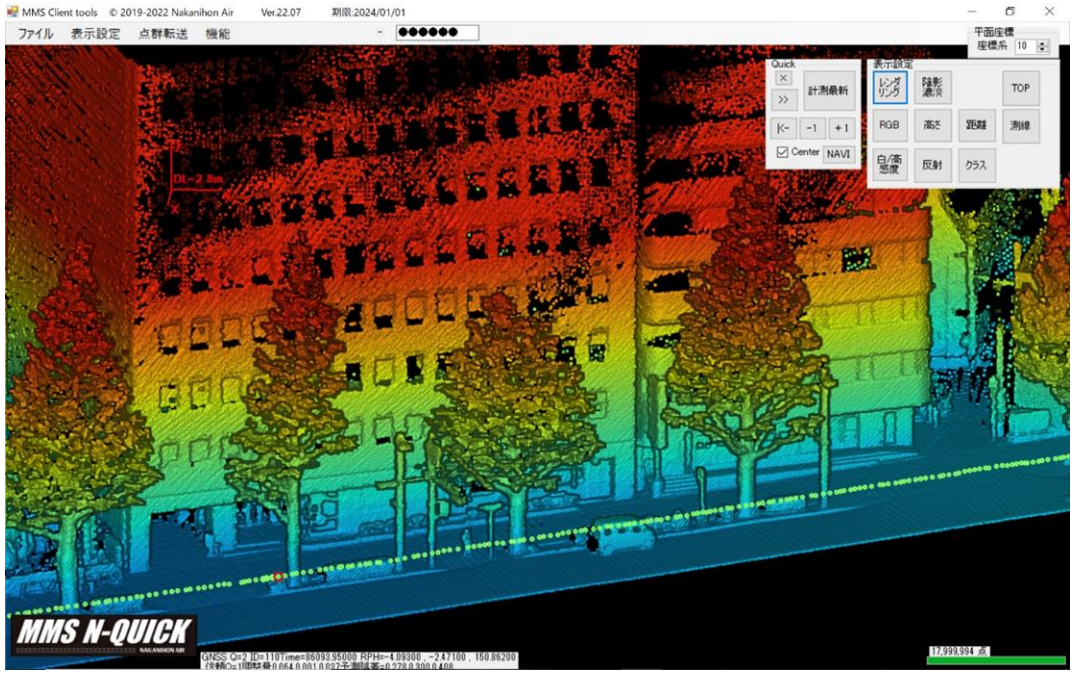
簡易MMSによる3次元点群データの取得検証⑤

撮影内容及び基準点について

バスの計測では、民間のGNSS補正サービス（SoftBank社ichimill）を基準として、測地座標の点群を作成した



バスに取り付けたセンサーで取得した点群データ（鳥瞰表示）



バスに取り付けたセンサーで取得した点群データ（拡大表示）

簡易MMSによる3次元点群データの作成検証

取得データの品質（位置精度や密度）について

本プロジェクトでは、実際の検証点測量を行っていないため、他地区での検証結果を参考する。他地区での検証では、計測時のGNSS測位品質がRTK FIXまたは30秒以内の非FIX環境ではレベル500相当であった。

基準点	検証点数[点]	水平位置較差 [m] dH=sqrt(dx²+dy²)			標高較差[m] dz			
		最大格差	平均	標準偏差	最小	最大	平均	標準偏差
Ichimill	107	0.279	0.097	0.114	-0.113	0.171	-0.004	0.039
My基準点	112	0.446	0.116	0.135	-0.150	0.033	-0.042	0.049

$= \sqrt{\frac{\Sigma(\text{残差})^2}{n}}$ $= \sqrt{\frac{\Sigma(\text{残差})^2}{n}}$

一方で、RTK-GNSSの測位品質が長期間にわたって悪い場合は、水平位置や標高精度は1mを越えることがあり、その場合はレベル2500相当となる。

地図情報レベル	水平位置の標準偏差	標高点の標準偏差	等高線の標準偏差
250	0.12m以内	0.25m以内	0.5m以内
500	0.25m以内	0.25m以内	0.5m以内
1000	0.70m以内	0.33m以内	0.5m以内
2500	1.75m以内	0.66m以内	1.0m以内
5000	3.50m以内	1.66m以内	2.5m以内
10000	7.00m以内	3.33m以内	5.0m以内

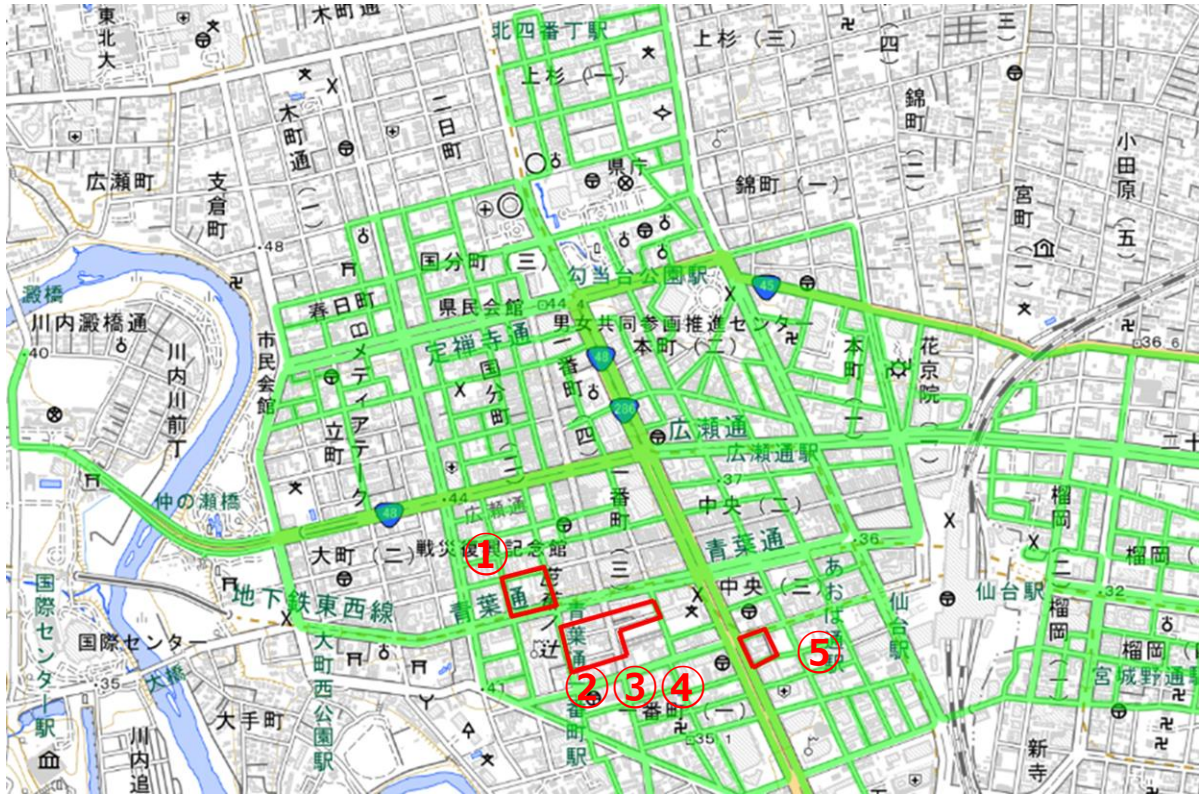
対象地域はGNSS測位が困難な都市部であり、すべての成果をレベル500相当とすることは困難であった。一部含まれるレベル2500相当のデータを鑑みて、レベル1000相当と表現した。

密度は車両速度やセンサーからの距離に起因するため一概には記載できないが、車両付近で密度が高く、約4cmに一点（500-1000点/m2）程度であった。一方で、歩道や建物付近では密度が下がり200-400点/m2であった。

iPhone LiDARによる点群データの取得・作成検証

撮影箇所について

- 主にバス路線及びタクシーを想定した一般車両が進入できない路地裏等を撮影箇所として選定した。



【iPhone LiDAR撮影箇所】

① 宮城県仙台市青葉区1丁目エリア
② 宮城県仙台市青葉区一番町2丁目3番地エリア
③ 宮城県仙台市青葉区一番町2丁目4番地エリア
④ 宮城県仙台市青葉区一番町2丁目5番地エリア
⑤ 宮城県仙台市青葉区中央4丁目5番地エリア

トレーニングデータの作成検証

City Generatorで作成されたトレーニングデータは、各点に対応するオブジェクトクラスラベルが付与された大規模点群データで、24,000のトレーニングサンプルと2,000の検証サンプル、合計で340GBのデータを出力した。

一般的に、モデルが認識する必要がある各オブジェクトクラス毎には少なくとも1,000トレーニングデータサンプルが必要である。トレーニングデータには合計33種類の異なるオブジェクトクラスが含まれており、今回のモデルのトレーニングには、密接に関連するラベルを組み合わせて、12のラベルセットに統合した。

0	undefined
1	person
2	bbicycle
3	vehicle
4	truck
5	static
6	clutter
7	advert
8	buildingclutter
9	wallsign
10	sunshade
11	pole

12	hbar
13	powerline
14	sign
15	trafficlight
16	fence
17	wakk
18	vending
19	stA.I.rs
20	curb
21	ground
22	road
23	tree

24	treebranch
25	vegetation
26	building
27	window
28	windowframe
29	roof
30	ceiling
31	buildingextension
32	extensionceiling

0	undefined	clear
1	vehicle	vehicle + truck
2	clutter	person + bicycle + static + clutter + advert + hbar + sign + trafficlight + vending
3	pole	pole + tree
4	powerline	powerline
5	treebranch	treebranch
6	vegetation	vegetation
7	wall	wall + curb
8	fence	fence
9	stA.I.rs	stA.I.rs
10	ground	ground + road
11	wallclutter	buildingclutter + wallsign + sunshade
12	building	building + window + windowframe + roof + ceiling + buildingextension + extensionceiling

ノイズ除去（領域分類A.I.）｜学習検証①

セグメンテーションモデルのトレーニングは、作成されたトレーニングデータセットを使用して、12GBのVRAMを持つRTX3060Ti GPU、高速のM.2 SSD、64GBのRAMを搭載したPCを使用し、1回のトレーニングセッションに約1週間要し、合計4千時間以上に亘った

トレーニングに利用したPC

セグメンテーションモデルトレーニング用PC

【スペック】
CPU：i7-11700K @3.70GHz
GPU：RTX360Ti（12GB）
RAM：64GB DDR4
SSD：1TB + 2TB M.2 NVMe

ノイズ除去（領域分類A.I.）｜学習検証②

仮想トレーニングデータセットを用いてセグメンテーションモデルを反復的に訓練した。精度検証として、学習時間と分類精度に分けて検証を行った

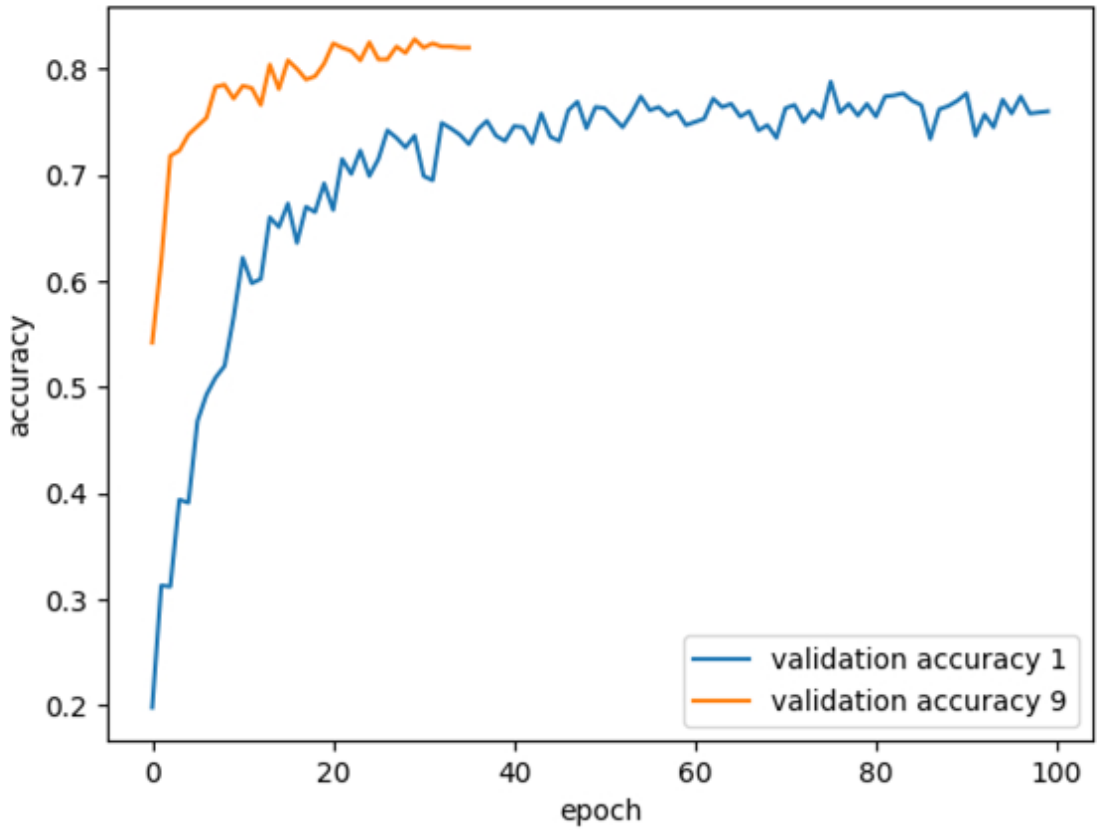
検証の概要

検証目的
- トレーニングデータによる学習精度を検証する

検証内容及び検証結果
① セマンティックセグメンテーションにおける分類結果の精度
- トレーニングデータを用いた学習初期と後期における分類精度と速度評価
  - **Epoch：学習時間**
    - 学習に要したイテレーションの時間
  - **Accuracy: 分類精度**
    - セマンティックセグメンテーション実行時の分類精度。1.0を100%とする

② 学習時間と精度
- 青が学習初期（Version.1）の結果である。Accuracyが0.75に収束するまでに要した時間は約58時間である
- 黄が学習後期（Version.9）の結果である。Accuracyが0.85に収束するまでに要した士官は約2時間である
- 学習時間及び分類精度が向上している

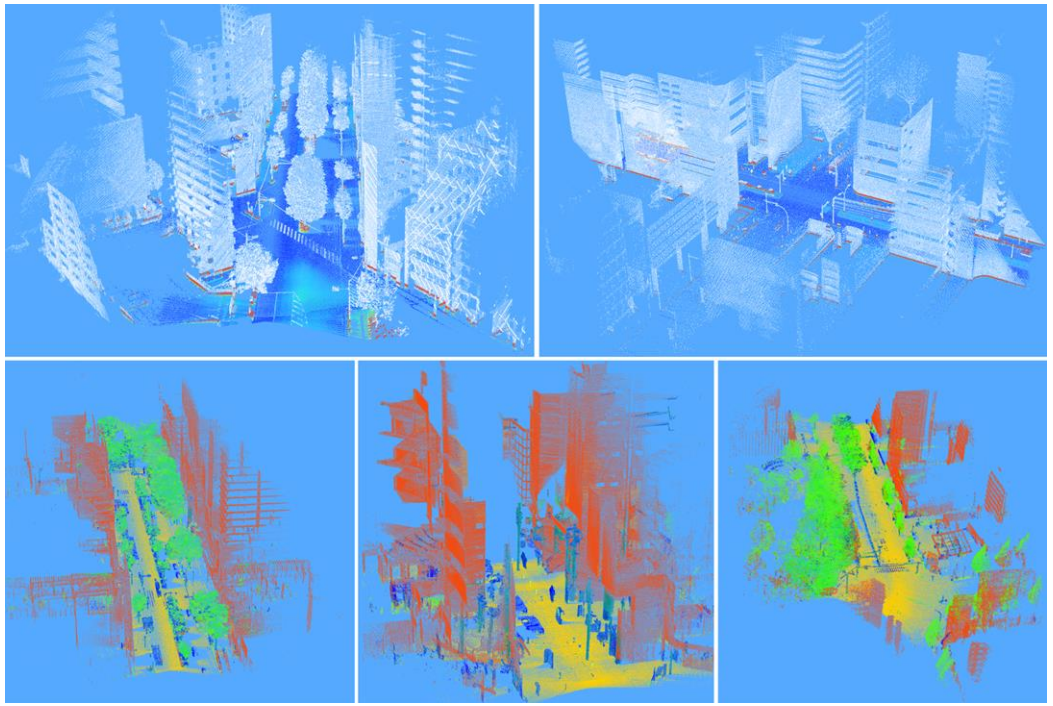
結果



ノイズ除去（領域分類A.I.）｜学習検証③

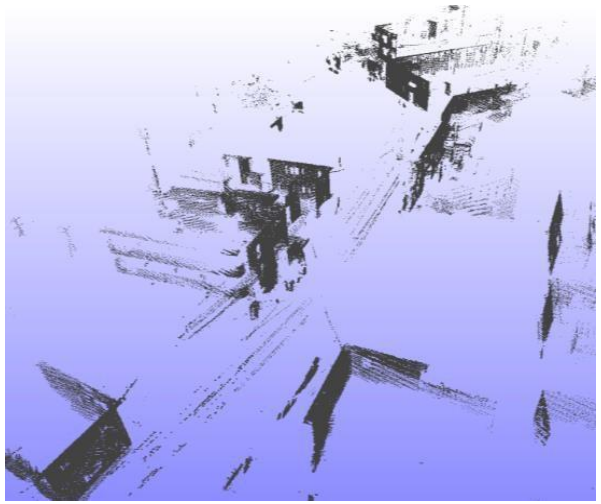
仮想トレーニングデータセットを用いて反復的な訓練プロセスを進めることで、LiDARデータによるセグメンテーション結果が向上した

領域分類

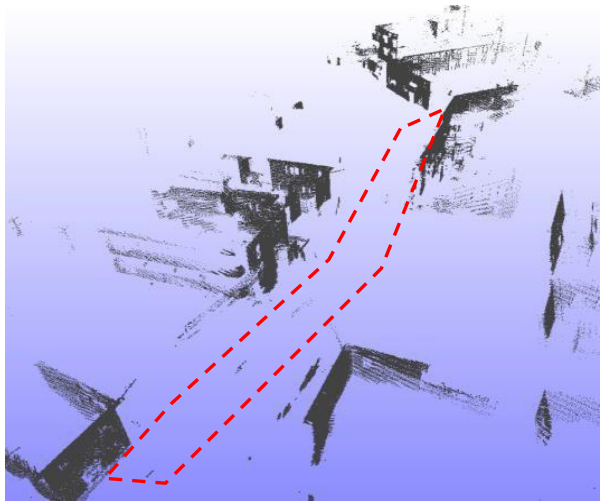


（左上）仙台のLiDARサンプル　（右上）仮想トレーニングデータ
（下3点）LiDARデータがセグメンテーションで地面と建物が綺麗に分離されている

ノイズ除去



学習初期（Version.1）



学習後期（Version.9）

同じシーンにある「建物」のポイントを、2つの異なるバージョンの
トレーニングデータセットでトレーニング後、同一のLiDARデータでセグメンテーションしたもの

2Dフットプリント作成検証

LOD2モデルから抽出した2Dフットプリントポリゴンを検証した

検証の概要

検証目的
- 建物の抽出精度検証する

検証内容及び検証結果
① 建物抽出結果の精度
- **総建物数**
  - 仙台市都市再生緊急整備地域に含まれる2,604棟を対象とする
- **抽出建物数**
  - 2、604棟、100%の抽出が行われた

結果

2Dフットプリント拡張検証

近隣住居の境界を考慮して、2Dフットプリントの境界を最大限拡張を行い検証した

検証の概要

検証目的
- 建物の境界拡張精度を検証する

検証内容及び検証結果
① 境界検出結果の精度
- **総建物数**
  - 仙台市都市再生緊急整備地域に含まれる2,604棟を対象とする
- **抽出建物数**
  - 2、604棟、全てにおいて近傍建物との境界の交差点を生じずに境界の拡張が100%行われた

結果

Ⅴ. 実証システムの検証 > 5. 更新箇所特定

異動判読評価検証｜評価尺度

航空写真を用いた目視による異動判読※1をもとに、A.I.による異動判読の精度を評価する。

概要

航空写真（オルソ画像）を用いて目視で異動判読を行った正データに対し、A.I.がどれほどの精度で異動を判読できたか評価する。

（※1）A.I.判読棟数：276棟

【評価対象】

（1）全体、（2）新築、（3）滅失、（4）増改築、（5）滅失後新築の各ケース

【評価尺度】

（1）正解率：A.I.の判読結果がどれほど正データと一致するか

（2）適合率：A.I.が正と判読した中で実際に正データだった割合

（3）再現率：正データの中でA.I.が正と判読できた割合

（4）F値：適合率と再現率の調和平均

※目視による異動判読
H27年とR4年の航空写真オルソ画像を比較し、変化があった箇所について新築／滅失／増改築／滅失後新築の分類を実施

評価尺度の詳細

項目	目視：正	目視：負
A.I.：正	TP（真陽性）	FP（偽陽性）
A.I.：負	FN（偽陰性）	TN（真陰性）

正解率：$\frac{TP+TN}{TP+TN+FP+FN}$

適合率：$\frac{TP}{TP+FP}$

再現率：$\frac{TP}{TP+FN}$

F値：$\frac{2\times適合率\times再現率}{適合率+再現率}$

異動判読評価検証｜異動クラス

更新箇所特定は、A.I.によって異動を検出する処理であり、MMS点群データのうち異動のあった建物周辺の点群のみを抽出し次の、3Dモデル化処理への入力データを絞り込むために使われる。異動検出アルゴリズムは「DCPCR（大規模な屋外環境の点群深層圧縮）」及び「GICP（反復計算による再近接点適合）」を改良して使用した。

建造物の異動とは、3D都市モデルの更新作業において、過去と現在の建物に生じた変化を指す。異動には、新築、滅失、増改築、滅失後新築の4種の異動クラスがある。(右表)

異動検出処理モジュールには更新されていない3D都市モデルとその地域を網羅する点群データが与えられ、3D都市モデルと点群の差異に基づいて建物IDごとに異動の有無・種別を分類する。

本調査ではH27年度の航空写真を元に作成された3D都市モデルと、2022年6～8月にMMSでスキャンした点群データに後処理を加えたものを比較し異動検出を行った。

異動クラスの一覧

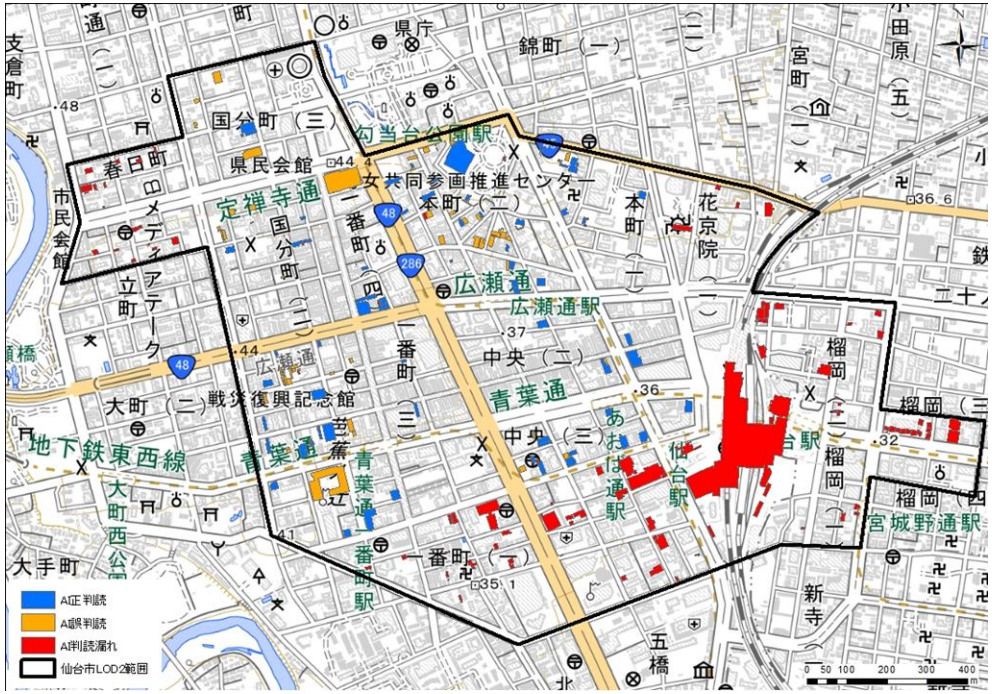
異動クラス	詳細
新築	更地であった場所に新たな建造物が建設されたことを示す。
滅失	存在していた建造物が解体されたことを示す。
増改築	既存の建物の一部の構造が増改築されたことを示す。
滅失後新築	存在していた建造物が解体され、その土地に別の建造物が建設されたことを示す。滅失した建物と新築した建物の数は必ずしも一致しない。

Ⅴ. 実証システムの検証 > 5. 更新箇所特定

異動判読評価検証｜判読全体

航空写真を用いた目視による異動判読をもとに、A.I.による異動判読（全体）の精度を評価する。

比較結果



項目	目視：変化あり	目視：変化なし
A.I.：変化あり	151	0
A.I.：変化なし	126	2481

※A.I.／目視ともに変化なしの棟数：R4LOD2建物棟数をもとに算出

単位：棟

精度評価

	正解率	適合率	再現率	F値
全体	0.954	1.000	0.545	0.706

・エリア内の建物棟数を母数とした場合の正解率は9割以上であり、非常に高いものであった。

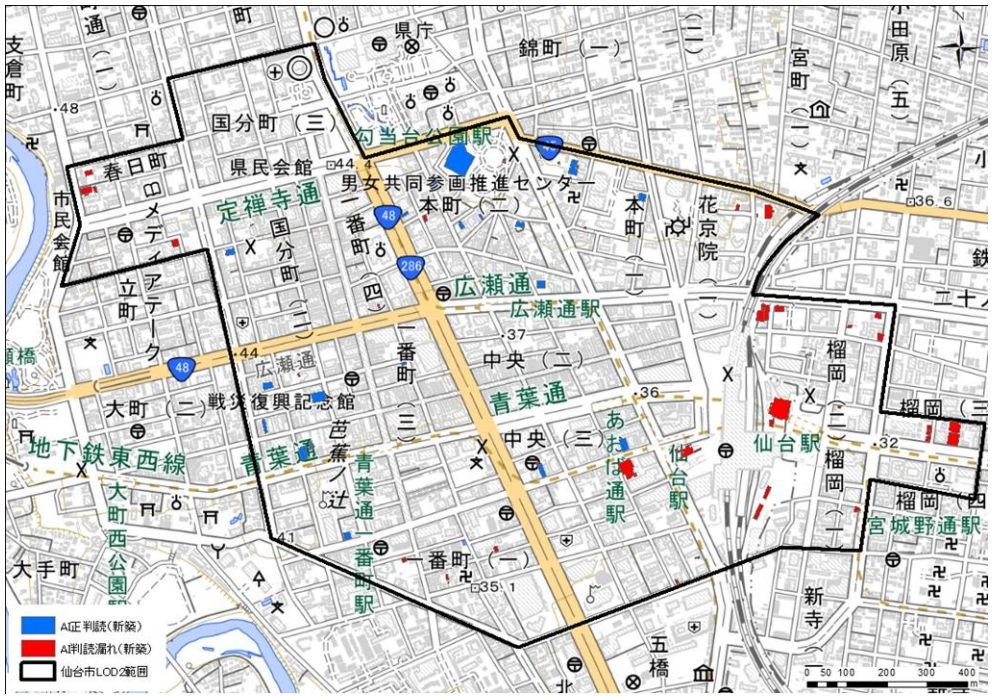
・適合率が100%であり、A.I.で変化を判読できた地物については、全て実際に新築／滅失／増改築／滅失後新築のいずれかの変化が発生していた。

・再現率は55%であり、判読すべき対象の建築物のうち、実際に判読できたものは約半数に留まった。

異動判読評価検証｜新築

航空写真を用いた目視による異動判読をもとに、A.I.による異動判読（新築）の精度を評価する。

比較結果



項目	目視：新築	目視：新築以外
A.I.：新築	30	0（0）
A.I.：新築以外	51（51）	195

単位：棟（カッコ内数値は判読漏れ数）

精度評価

	正解率	適合率	再現率	F値
新築	0.815	1.000	0.370	0.541

・判読対象を母数とした正解率は、8割程度と比較的高い傾向にあった。

・適合率が100%と高く、A.I.により新築と判定された箇所は、全て正しく判読されていた。

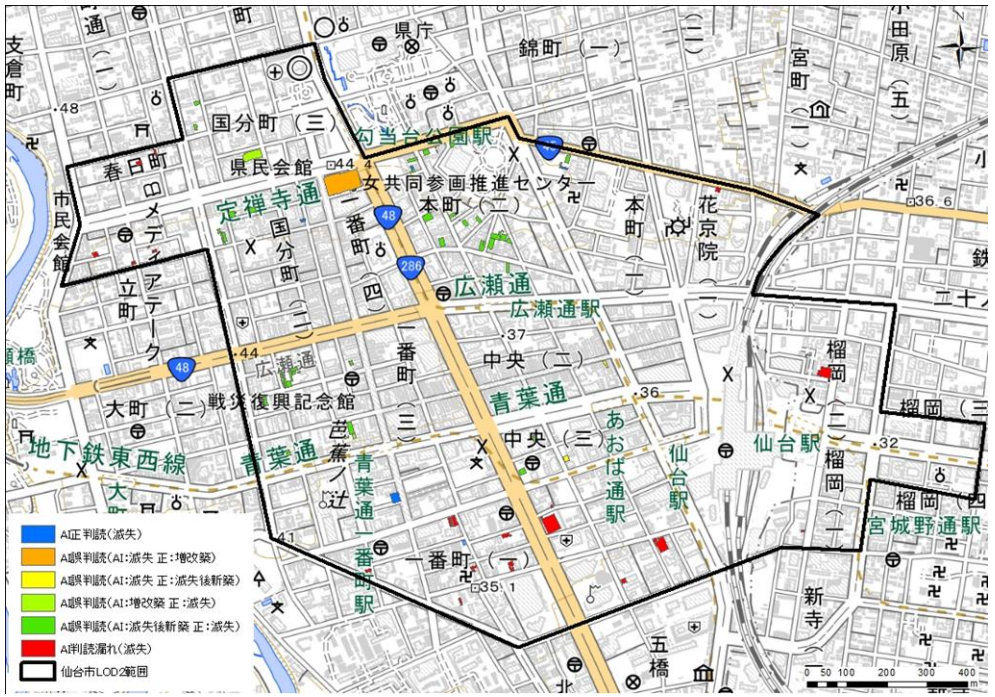
・全ての目視データのうち、A.I.で判読された新築建築物は37%に留まった。他の分類への誤判読は無かったものの、判読漏れが多く発生した。

Ⅴ. 実証システムの検証 > 5. 更新箇所特定

異動判読評価検証｜滅失

航空写真を用いた目視による異動判読をもとに、A.I.による異動判読（滅失）の精度を評価する。

比較結果



項目	目視：滅失	目視：滅失以外
A.I.：滅失	7	4（0）
A.I.：滅失以外	88（33）	177

単位：棟（カッコ内数値は判読漏れ数）

精度評価

	正解率	適合率	再現率	F値
滅失	0.667	0.636	0.074	0.132

・判読対象を母数とした際の正解率は約2/3に留まった。

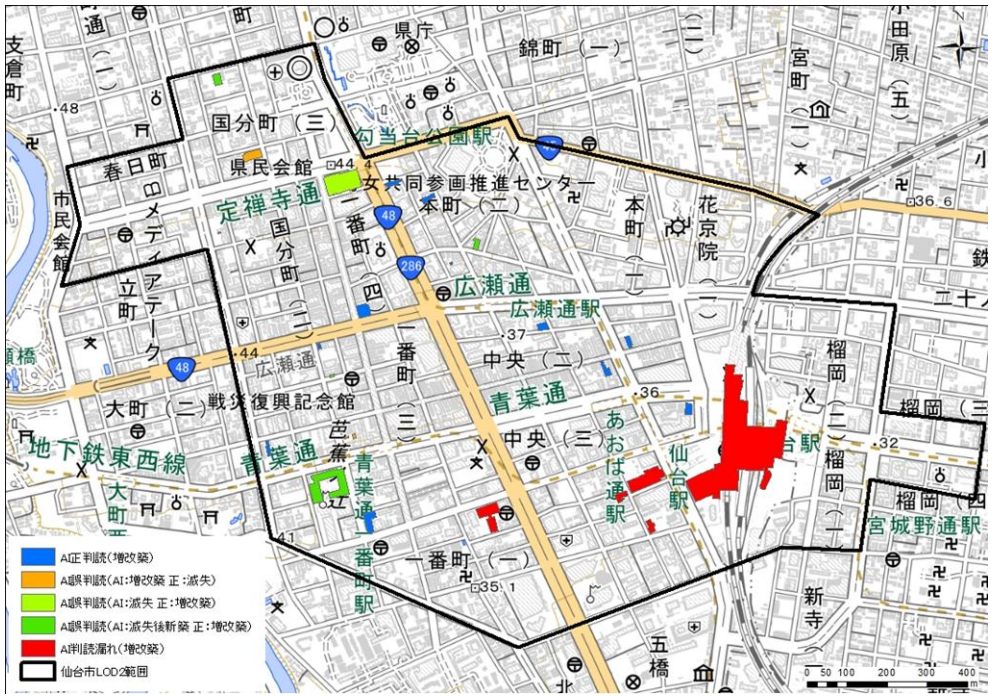
・A.I.で滅失と判定された建築物のうち、正しいものは64%に留まり、3割程度は他の分類へ誤判読された。

・再現率が低く、目視で滅失とされた建築物のうち、実際に滅失として判定されたものは7%である。ほとんどが他の分類へと誤判読されたほか、うち4割程度で判読漏れが発生した。誤判読の多くは滅失後新築である。

異動判読評価検証｜増改築

航空写真を用いた目視による異動判読をもとに、A.I.による異動判読（増改築）の精度を評価する。

比較結果



項目	目視：増改築	目視：増改築以外
A.I.：増改築	11	4（0）
A.I.：増改築以外	12（6）	249

単位：棟（カッコ内数値は判読漏れ数）

精度評価

	正解率	適合率	再現率	F値
増改築	0.942	0.733	0.478	0.579

・判読対象を母数とした際の正解率は、94%と高い傾向にあった。

・A.I.により増改築と判読された建築物のうち、73%が正しく判読されていた。

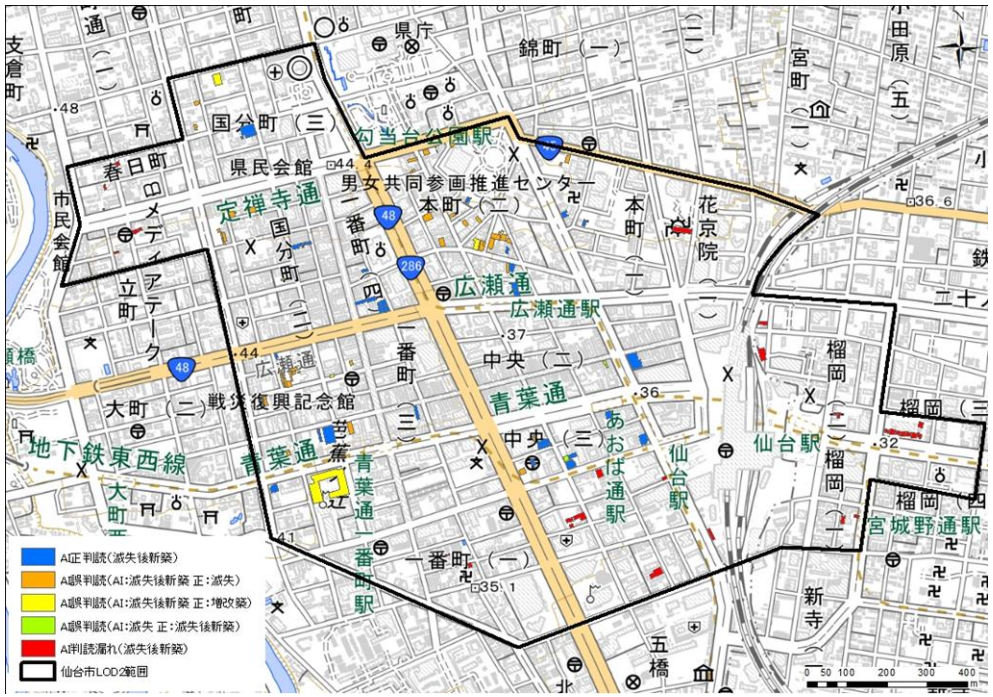
・実際に増改築があった建築物のうち、A.I.で正しく判読された建築物は5割前後に留まった。正しく判読されなかった増改築建築物のうち、誤判読と判読漏れが半数ずつを占めた。

Ⅴ. 実証システムの検証 > 5. 更新箇所特定

異動判読評価検証｜滅失後新築

航空写真を用いた目視による異動判読をもとに、A.I.による異動判読（滅失後新築）の精度を評価する。

比較結果



項目	目視：滅失後新築	目視：滅失後新築以外
A.I.：滅失後新築	36	58（0）
A.I.：滅失後新築以外	39（36）	143

単位：棟（カッコ内数値は判読漏れ数）

精度評価

	正解率	適合率	再現率	F値
滅失後新築	0.649	0.383	0.48	0.426

・判読対象を母数とした際の正解率は、65%と低い結果となった。

・A.I.で滅失後新築と判定された建築物のうち、実際に正しく判定されたものは38%程度に留まった。

・滅失後新築の建築物のうち、正しくA.I.で判定されたものは48%であった。判読されなかったもののほとんどは、判読漏れによるものであった。

異動判読評価検証｜まとめ

航空写真を用いた目視による異動判読をもとに、A.I.による異動判読の精度を評価する

異動判読評価
• 目視で判読できた地物のうち、A.I.で判読できたものは全体の55%であった。一方、A.I.で判読できた地物のうち、目視で判読できなかったものはなかった。エリア内の棟数を母数とした場合、全体の9割以上の地物を正しく判読することができた。適合率は100%であり、A.I.で変化を判読できた地物については、全て実際に新築／滅失／増改築／滅失後新築のいずれかの変化が発生していた。再現率は5割程度に留まり、判読すべき対象の建築物のうち、実際に判読できたものは約半数に留まった。
• 新築では適合率が高く、A.I.で新築と判定された建築物は正しく判定されており、信頼度が高い。一方、滅失では誤判読や判読漏れが多く発生した。異動判読の精度が低く、すべてをA.I.による処理で行うには現状難しい。誤判読は「滅失」と「滅失後新築」との間で多く発生しており、これら分類に対する精度向上が必要と考えられる。
• 建築物面積が狭小な地物については、新築での判読漏れが目立った。狭小な建物は他建物の影となりやすく、点群の取得ができなかったことで判読精度が低下した。
• A.I.の活用により、新築では一部ではあるが高精度で抽出ができることから、目視による判読作業内の一定の省力化に寄与できる可能性がある。しかしながら、さらなる作業の効率化／省力化へは、判読漏れや誤判読を改善する必要がある。
• 様々なデバイス今回のA.I.判読では、タクシーを想定した一般車両のみから点群を取得したエリア（仙台駅の南・東側エリア、LOD2エリア西端部）で、判読精度が低下する傾向が見られた。この要因として、バス車両と比較して一般車両の走行回数が大幅に少なかったことが挙げられる。点群取得を行う車両の走行回数が増加した場合、A.I.の判読精度が向上する可能性がある。

LOD3建築物モデリング評価検証

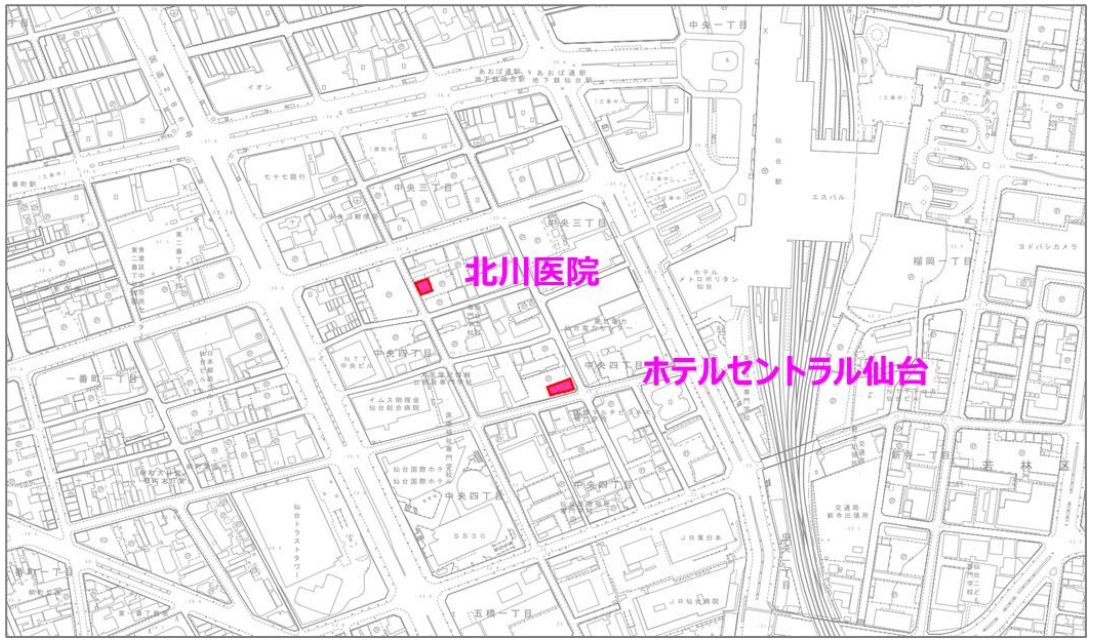
出力されたサーフェスモデル、自動処理済み点群データ、自動処理無し点群データを用いた目視及びLOD3モデル図化作業をもとに本システムの有用性を評価する

概要

- 自動モデリングツールで生成されたLOD3建築物モデル（サーフェスモデル）の精度を検証するため、比較として以下のモデルを作成した。
  ①入力データ処理モジュールによって自動処理された点群データから手動で作成したモデル
  ②通常の点群データから手動で作成したモデル

- それぞれに作成されたLOD3建築物モデルの相違点から、モデルとしての完全性や作業効率等の観点から評価をおこなった。
- 対象とした建築物は、宮城県仙台市内の「ホテルセントラル仙台」と「北川医院」とした。

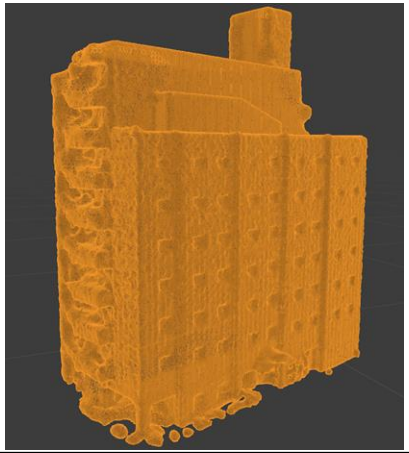
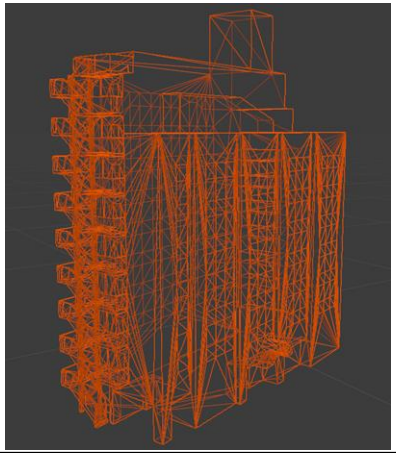
対象建築物



サーフェスモデル検証

自動生成されたサーフェスモデルとLOD3モデルの比較評価検証

自動生成されたサーフェスモデルとサーフェスモデルから作成したLOD3モデルのメッシュを比較し、自動生成モデルの完成度を評価した。

比較項目	自動生成されたサーフェスモデル	サーフェスモデルから作成したLOD3モデル	評価
メッシュの外観			自動生成されたサーフェスモデルは、全体としての形状は表現できているものの、エッジが不鮮明であり、入り口や階段など奥まった部分や複雑な形状の部分の再現度が低い。
頂点の数	179,365	10,591	94%の削減
辺の数	538,029	10,591	98%の削減
面の数	358,686	3,518	99%の削減
PLY形式のファイルサイズ	17,402 KB	679 KB	96%の削減

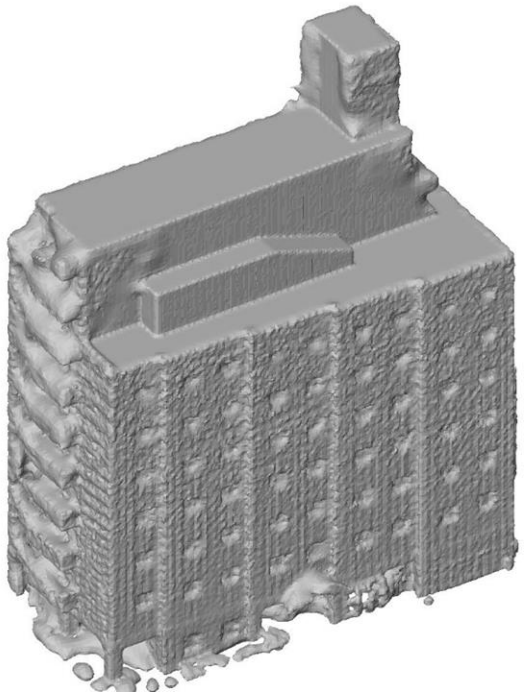
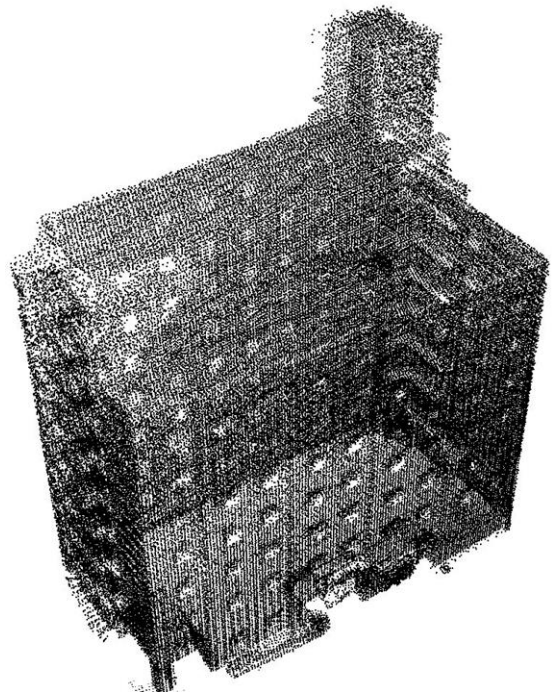
作成したLOD3モデルは、良好な形状を保ちつつ、メッシュ構造が大幅に簡略化されていることが見て取れる。メッシュをPLY(ASCII) 形式で保存した際のファイルサイズは約96%縮小されており、自動生成されたサーフェスモデルに対して大幅に軽量化されていると言える。

したがって、サーフェスモデルのLOD3レベルの自動生成には、まだ大幅な改善の余地があることが分かった。

LOD3モデル作成の省力化検証

サーフェスモデルから作成したLOD3モデルと自動化処理後の点群から作成したLOD3モデルによる比較評価検証

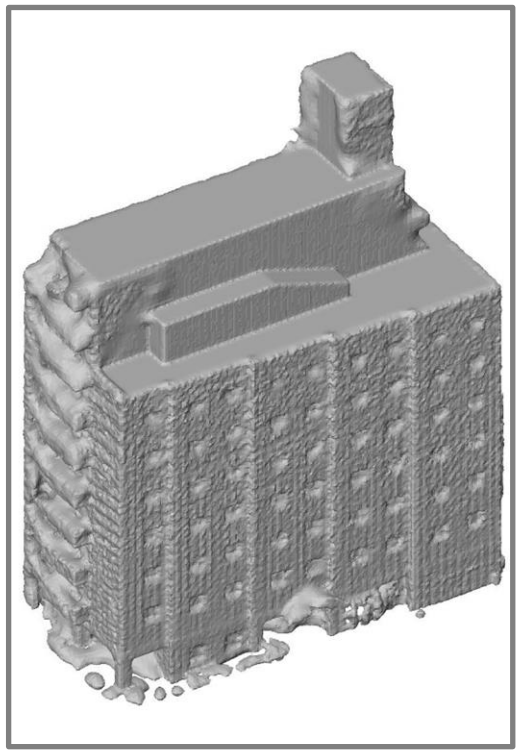
LOD3モデル作成の省力化評価として、自動化処理後（点群合成・ノイズ除去前と除去後）の点群と、その点群から自動生成されたサーフェスモデルをもとにLOD3建築物のモデリングを作成し比較を行った。

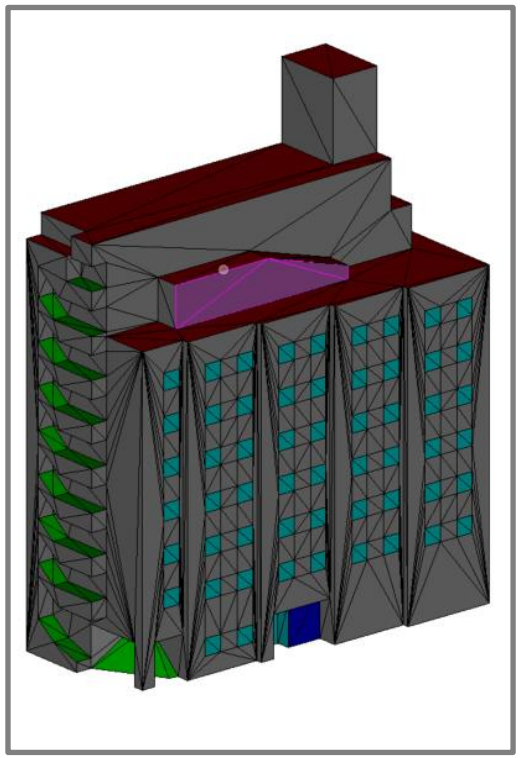
サーフェスモデル	点群
	
点群から自動生成されたサーフェスモデル	自動化処理後：点群合成・ノイズ除去後の点群

LOD3モデル作成の省力化検証

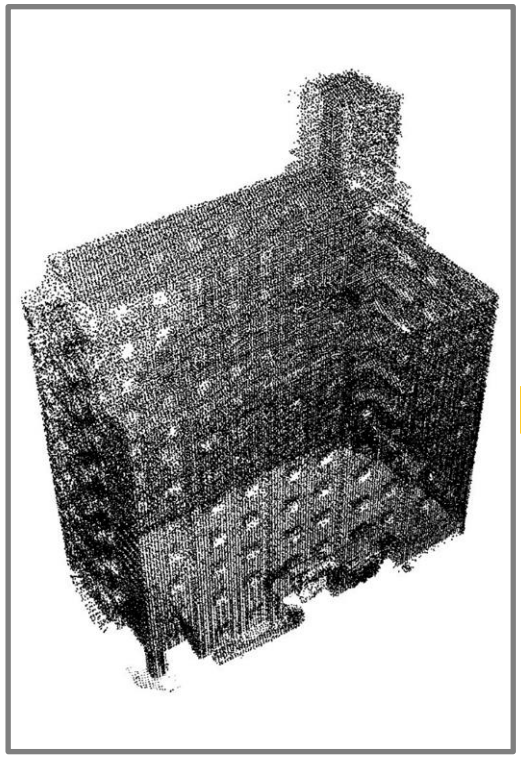
サーフェスモデルから作成したLOD3モデルと自動化処理後の点群から作成したLOD3モデルによる比較評価検証

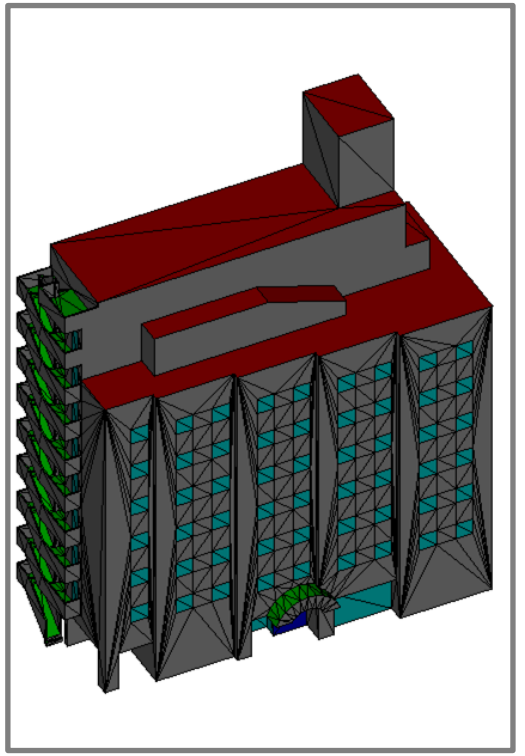
サーフェスモデルと点群データからLOD3建築物のモデリングをおこなった。





サーフェスモデルから
作成したLOD3



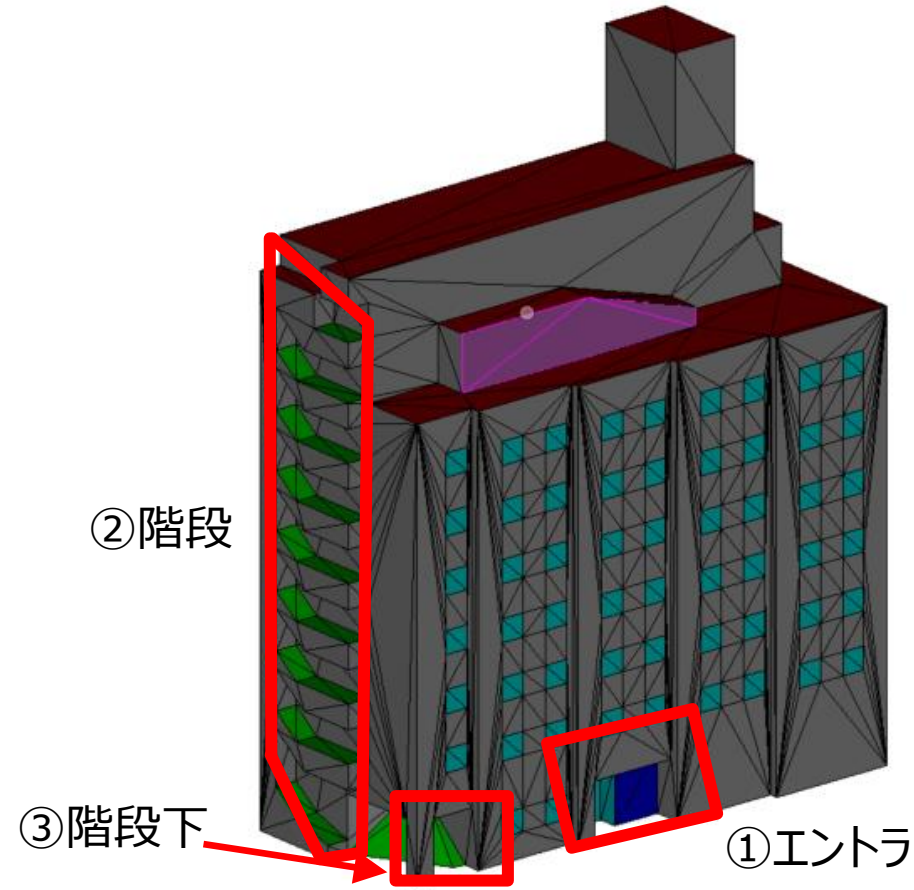


点群から作成した
LOD3

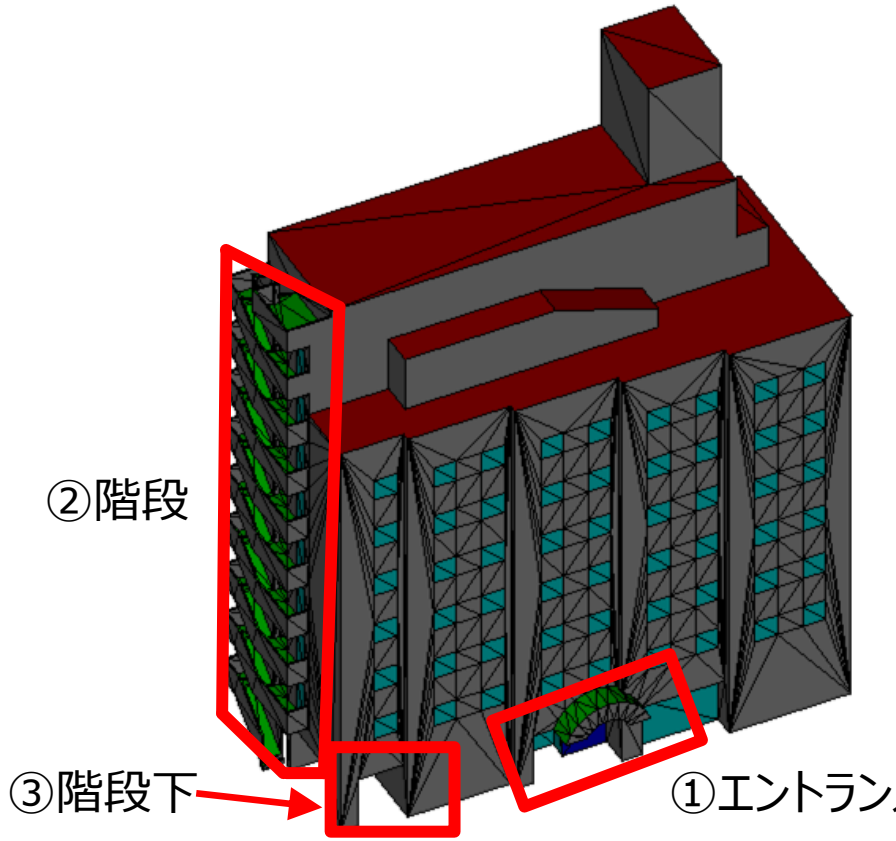
LOD3モデル作成の省力化検証

サーフェスモデルから作成したLOD3モデルと自動化処理後の点群から作成したLOD3モデルによる比較評価検証

サーフェスと点群データを使用してモデリングしたLOD3建築物で、両者を比較して、相違があった箇所の評価をおこなった。



サーフェスモデルでモデリングしたLOD3建築物と相違箇所

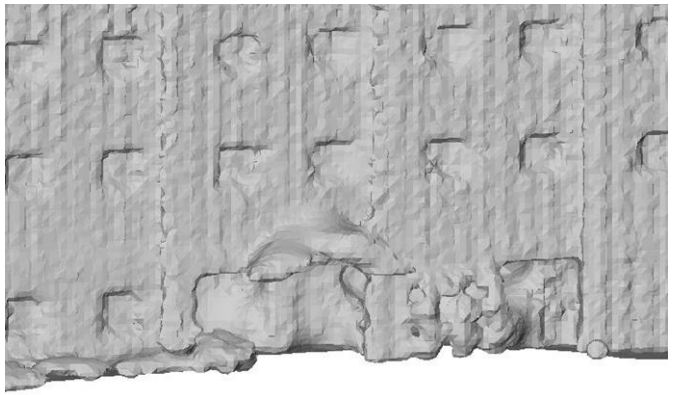
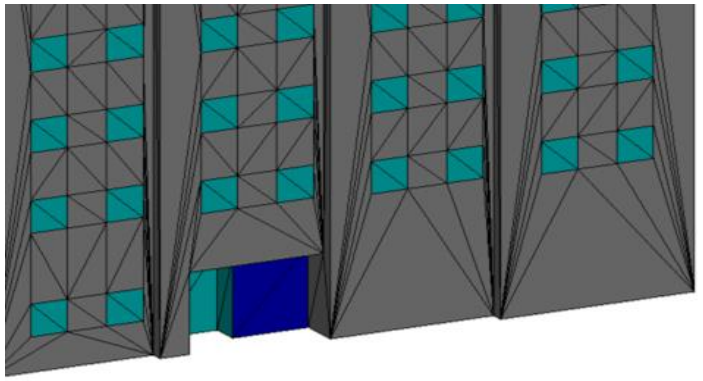
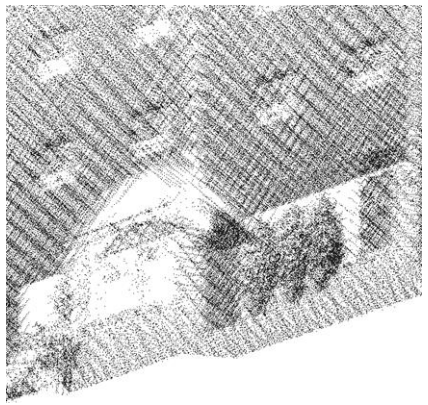
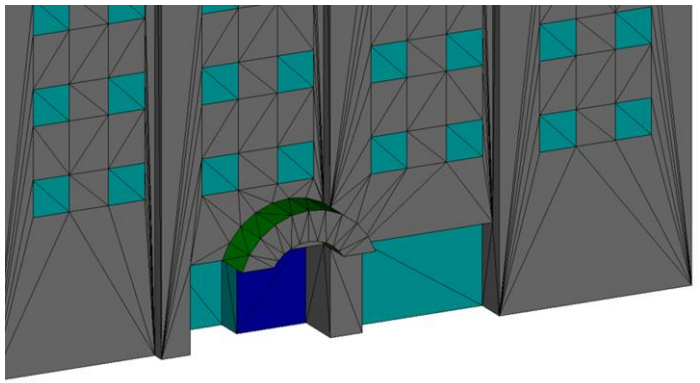


点群でモデリングしたLOD3建築物と相違箇所

LOD3モデル作成の省力化検証

サーフェスモデルから作成したLOD3モデルと自動化処理後の点群から作成したLOD3モデルによる比較評価検証

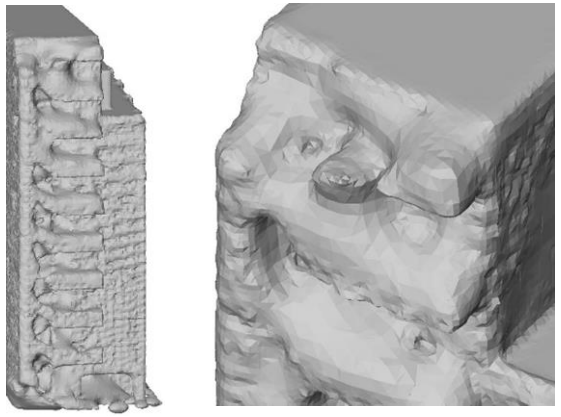
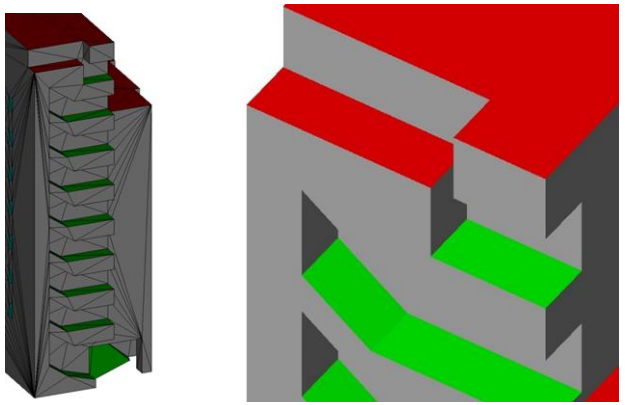
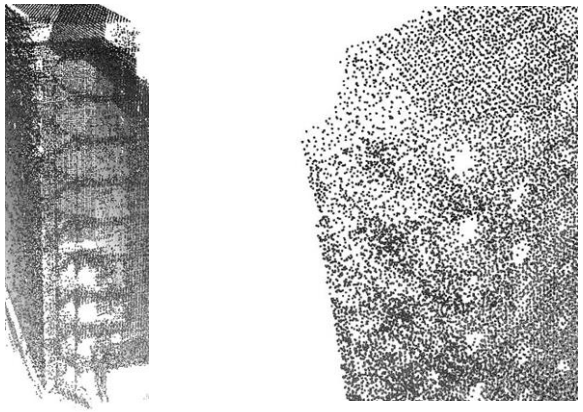
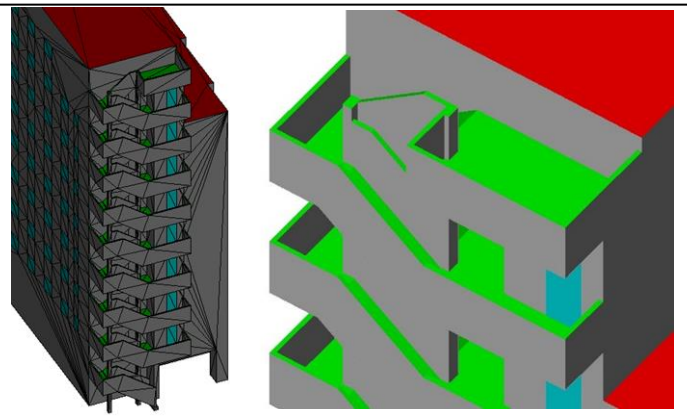
①エントランス

	元データ	作成したLOD3	評価
サーフェスモデル			入り口部が不鮮明で、入り口を完全に表現することができない。
点 群			入り口の凹凸が確認でき、写真を参考に、入り口を表現することができる。

LOD3モデル作成の省力化検証

サーフェスモデルから作成したLOD3モデルと自動化処理後の点群から作成したLOD3モデルによる比較評価検証

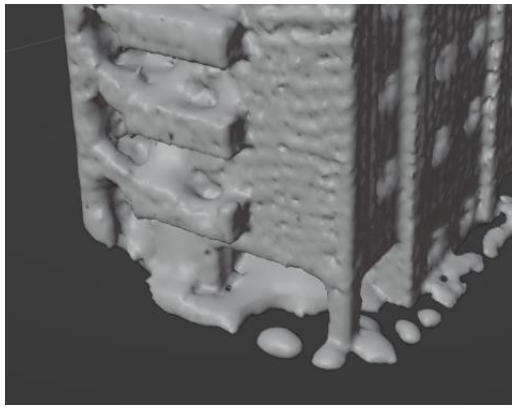
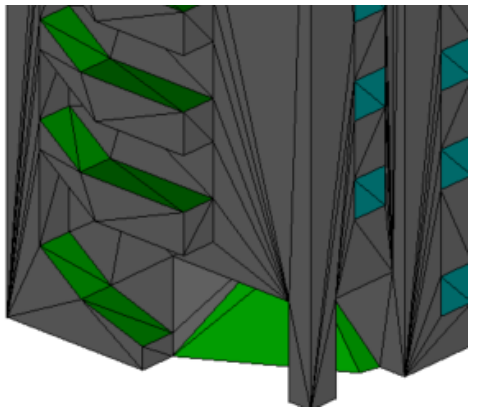
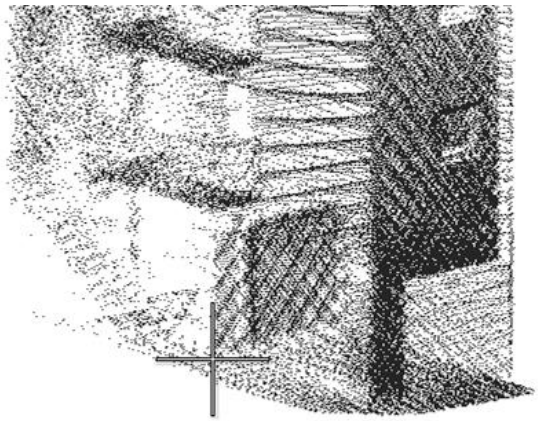
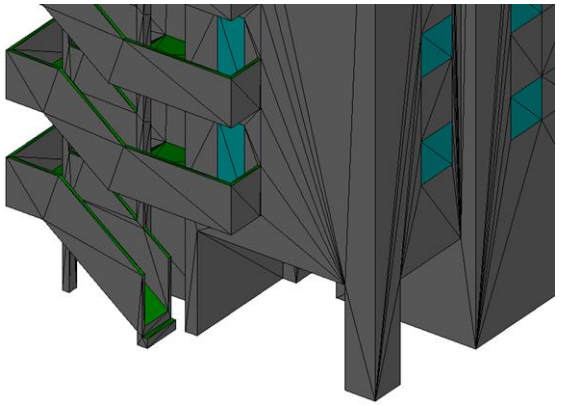
②階段

	元データ	作成したLOD3	評価
サーフェスモデル			階段の表面部分しか取得されておらず、階段を完全表現することができない。
点 群			階段や柱の形状や奥行を確認することができ、写真を参考に階段や柱を表現することができる。

LOD3モデル作成の省力化検証

サーフェスモデルから作成したLOD3モデルと自動化処理後の点群から作成したLOD3モデルによる比較評価検証

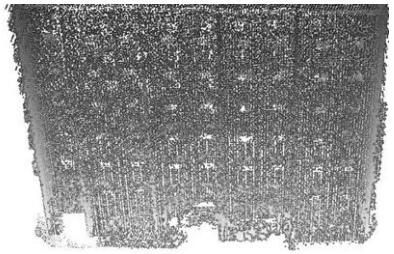
③階段下

	元データ	作成したLOD3	評価
サーフェスモデル			階段下部分が正しく取得されておらず、階段下を完全表現することができない。
点群			階段下の凹凸が確認でき、写真を参考に、階段下のへこみを表現することができる。

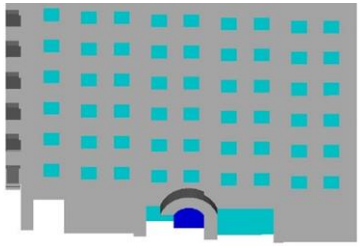
LOD3モデル作成の省力化検証

自動化処理後の点群と従来点群によるLOD3モデル図化手法検討

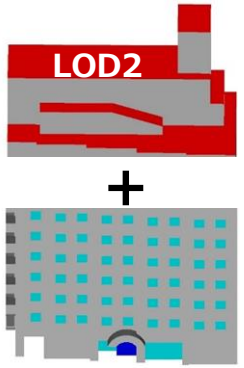
従来の点群データを使用したモデリング方法



点群データを使用してモデリングをする

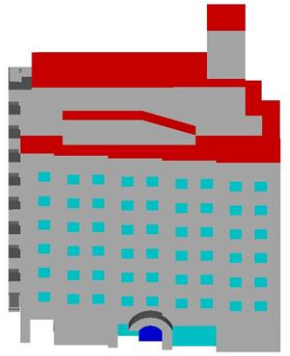


MMS点群データの当たっている面のみをモデリングする

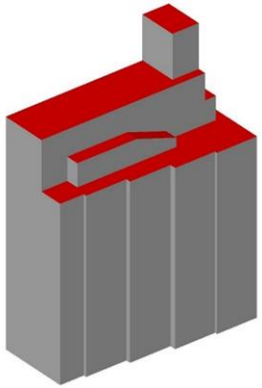


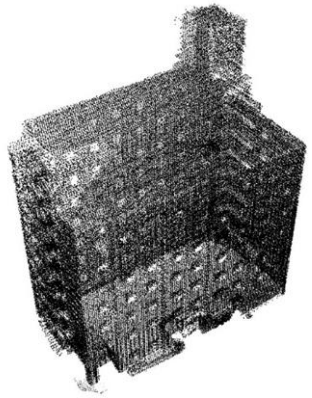
・点群データが当たっていない面は、既存のLOD2をコピーして接合させる。
・接合が出来ない場合は、LOD2の形状を参考にしてモデリングし直す。

完成



自動化処理後（点群合成・ノイズ除去後）の点群を使用したモデリング手法

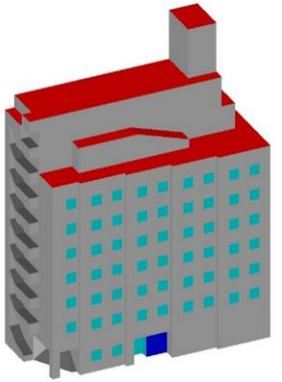




点群データは、既存のLOD2に合わせて作成されているので、既存のLOD2の形状をベースにして、窓や入口等のLOD3に必要な開口部について、点群を使ってモデリングをおこなう。

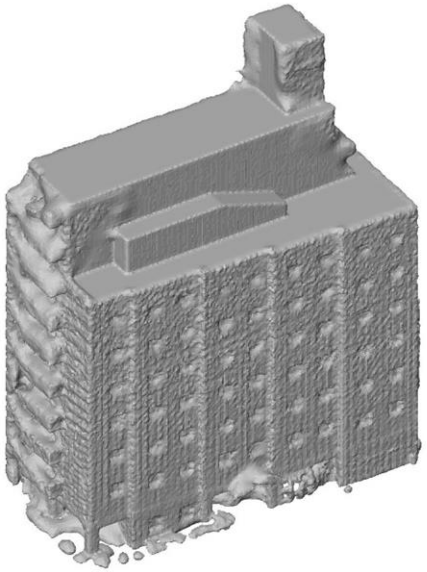
MMS点群データが当たっていない部分も、LOD2から点群を自動作成しているので、その点群を使ってモデリングをおこなう。

完成



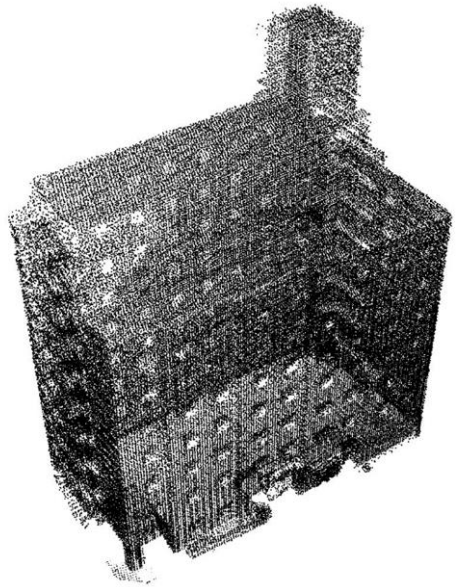
LOD3モデル作成の省力化検証

サーフェスモデルから作成するLOD3モデルと自動化処理後の点群から作成するLOD3モデルの省力化比較検証



サーフェスモデルを使用したモデリング方法

【モデリングの効率】
このサーフェスモデルの形状を修正してLOD3を作成すると効率が悪いが、サーフェスモデル上の必要な点の座標を取得して新たに図化することで、効率は非常に良くなる。サーフェスモデルは、建物の形状やエッジの位置がはっきり見えるため、直感的にも取得する位置や面を把握し易い。



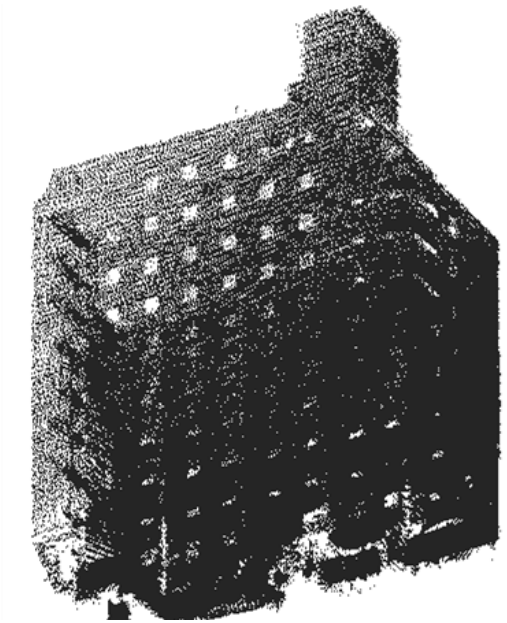
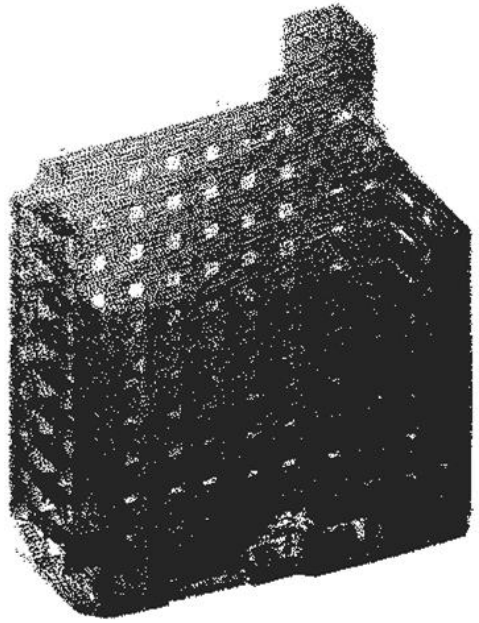
点群データを使用したモデリング方法

【モデリングの効率】
点群は折り重なって表示され、建物も透けて見えるので、建物のエッジの位置が非常に把握し難い。
CAD上のWindowを真上・横・斜め方向など複数開いて見て、建物の取得位置やエッジの位置を確認した上で図化する必要があり、モデリング作業の効率が良くない。

LOD3モデル作成の省力化検証

自動化処理前、自動化処理後の点群データ比較

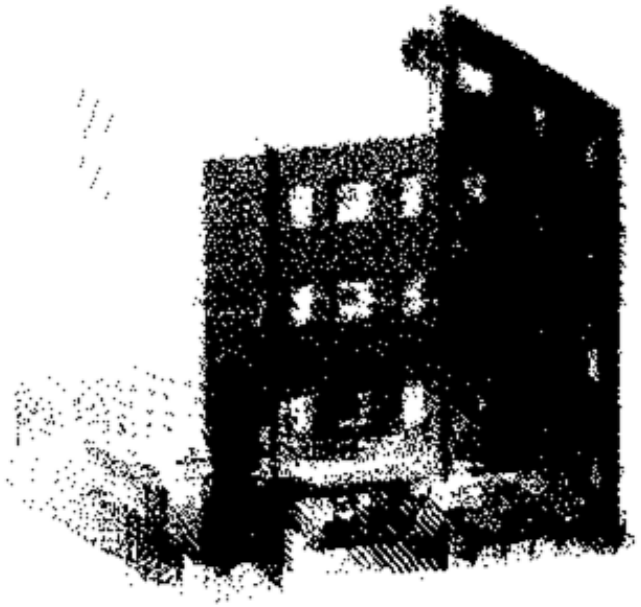
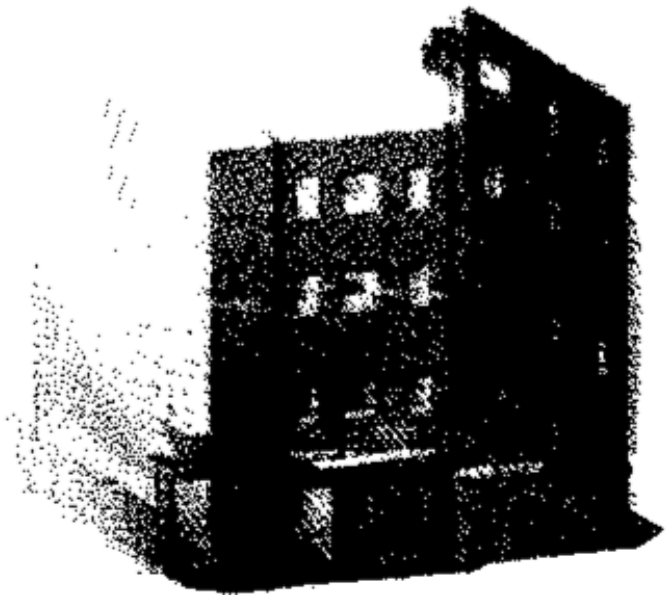
自動化処理前と自動化処理後（点群合成・ノイズ除去前と除去後）の点群で評価をおこなった。
(1) ホテルセントラル仙台

点群A	点群B
	
自動化処理前：点群合成・ノイズ除去前の点群	自動化処理後：点群合成・ノイズ除去後の点群

LOD3モデル作成の省力化検証

自動化処理前、自動化処理後の点群データ比較

評価は、自動化処理前（点群A）と点群合成・ノイズ除去を行った自動化処理後（点群B）の点群でおこなった。
(2) 北川医院

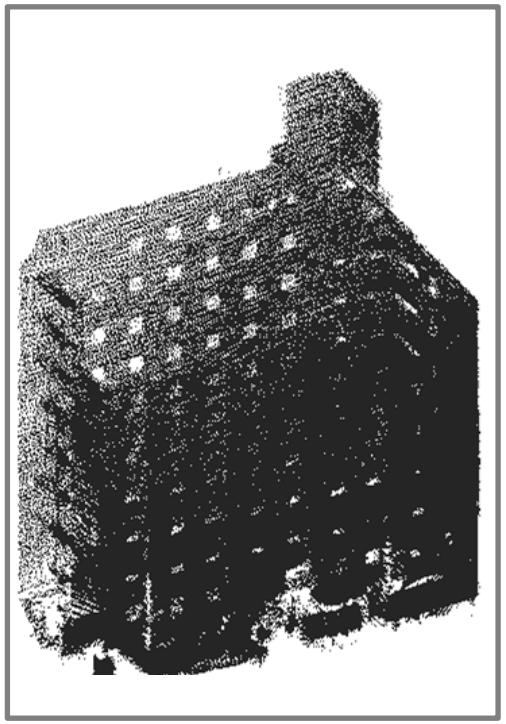
点群A	点群B
	
自動化処理前：点群合成・ノイズ除去前の点群	自動化処理後：点群合成・ノイズ除去後の点群

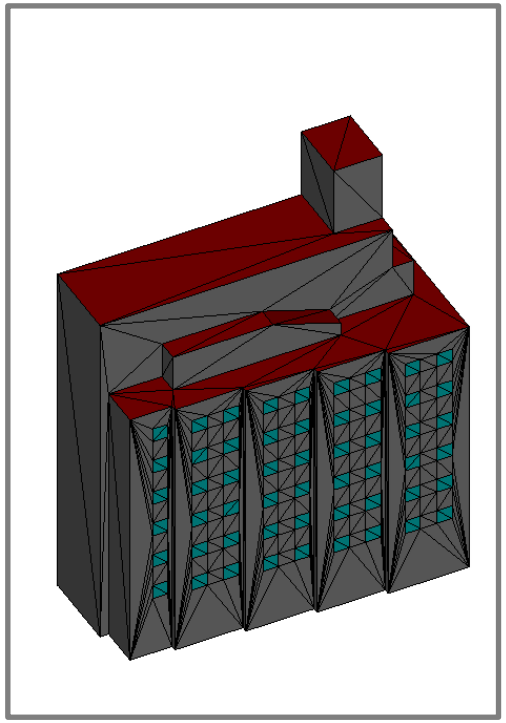
LOD3モデル作成の省力化検証

自動化処理前、自動化処理後の点群データから作成したLOD3建築物の比較検証

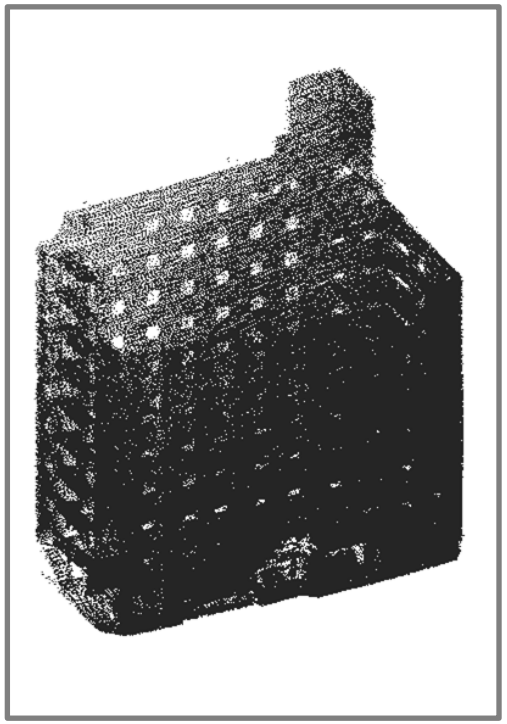
- データからLOD3建築物のモデリングをおこなった。

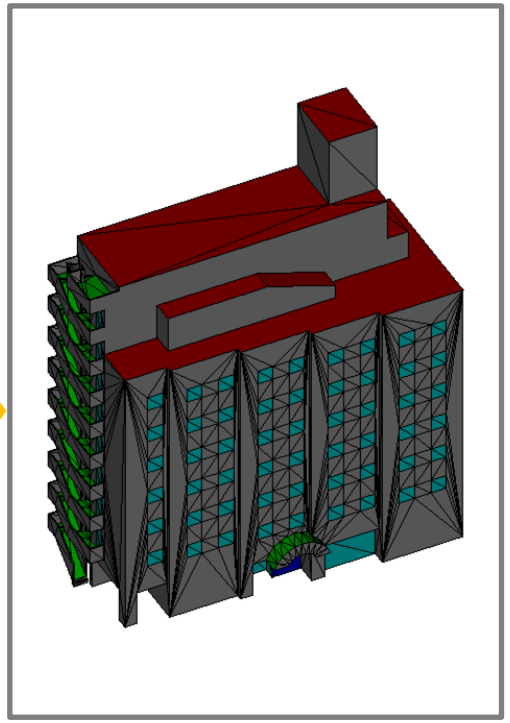
(1) ホテルセントラル仙台





点群Aから作成した
LOD3（A）





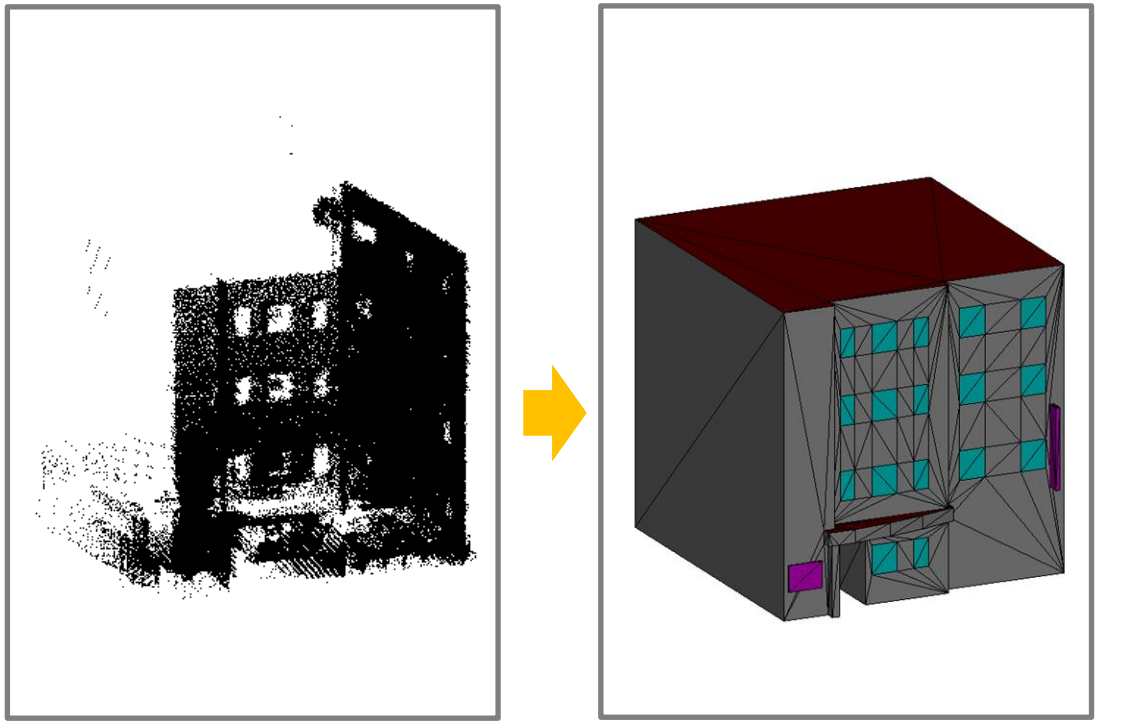
点群Bから作成した
LOD3（B）

LOD3モデル作成の省力化検証

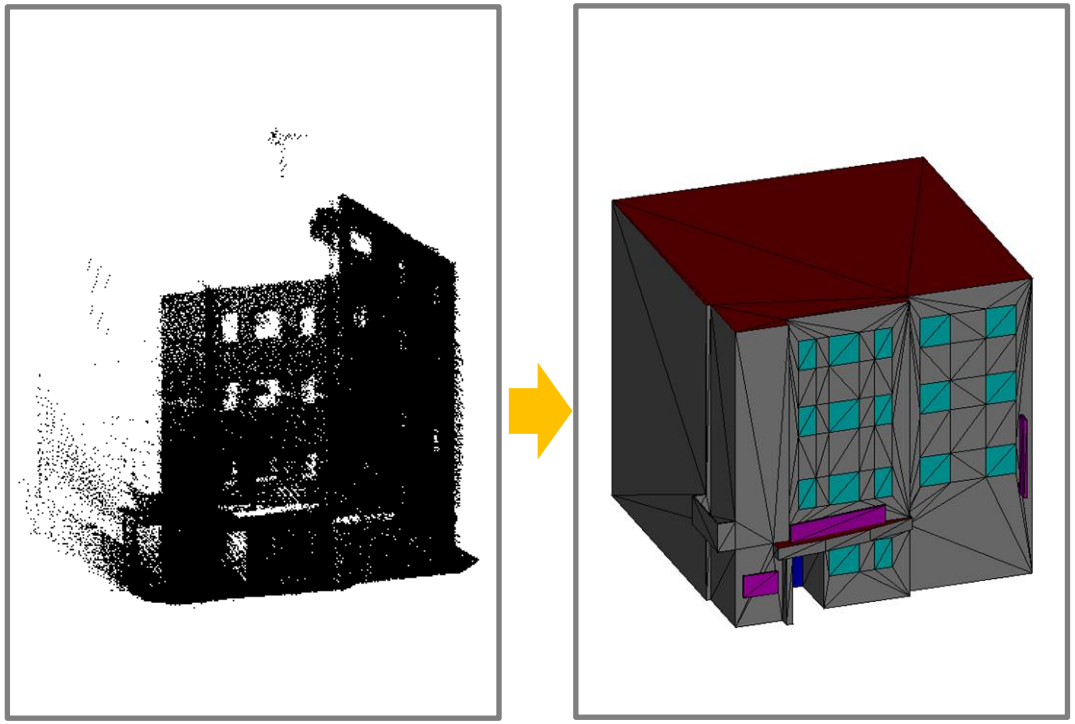
自動化処理前、自動化処理後の点群データから作成したLOD3建築物の比較検証

- 点群データからLOD3建築物のモデリングをおこなった。

(2) 北川医院



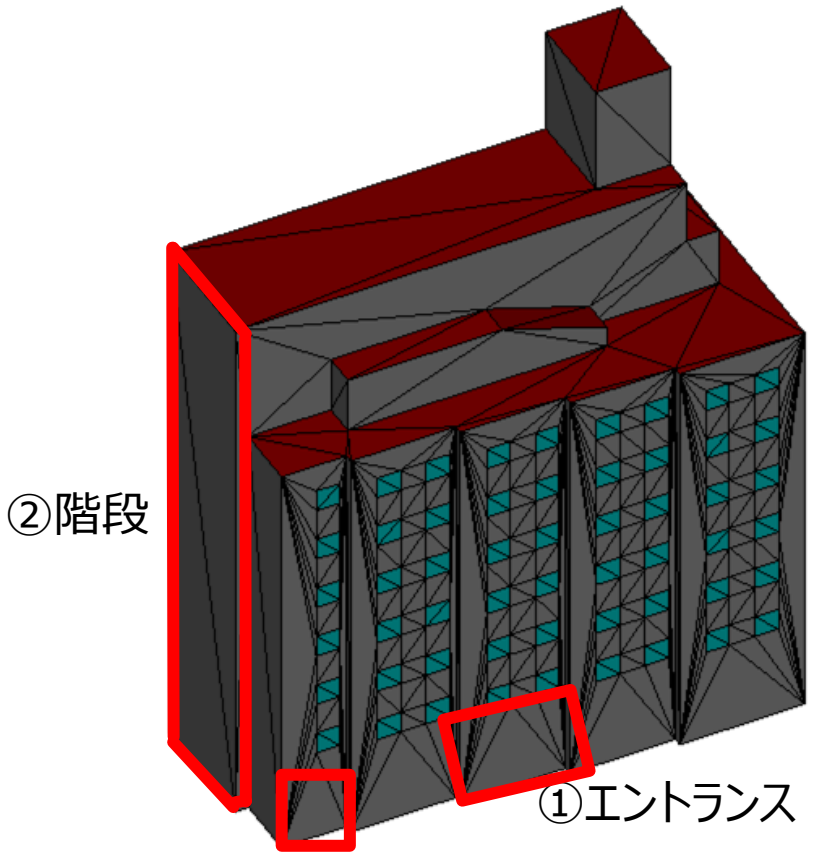
点群Aから作成した
LOD3（A）



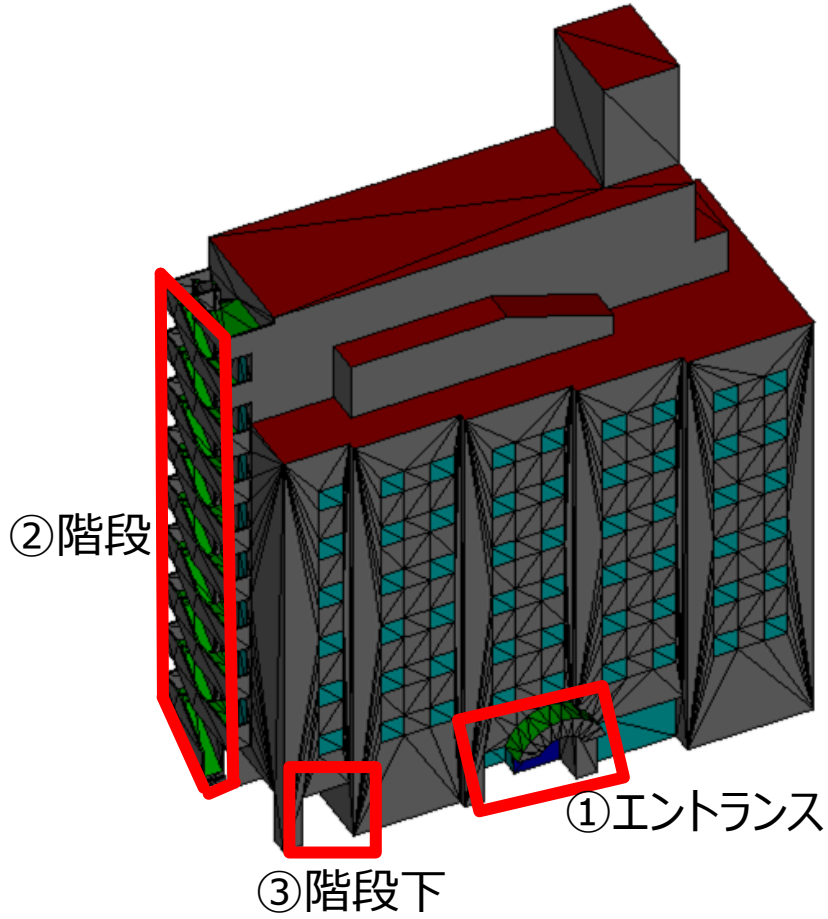
点群Bから作成した
LOD3（B）

LOD3モデル作成の省力化検証

自動化処理前、自動化処理後の点群データから作成したLOD3建築物の比較検証



点群Aでモデリングした LOD3建築物と相違箇所

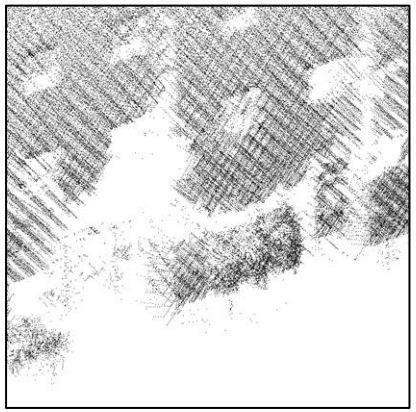
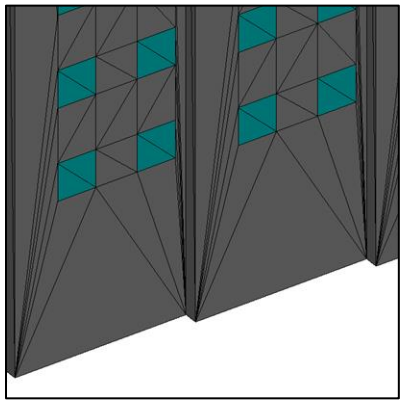
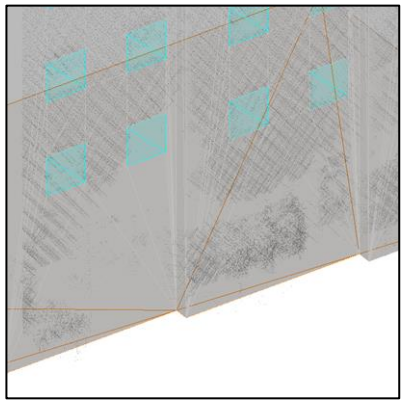
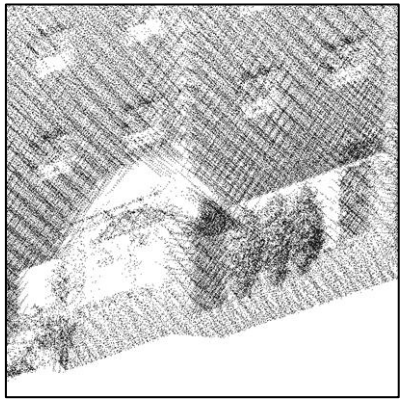
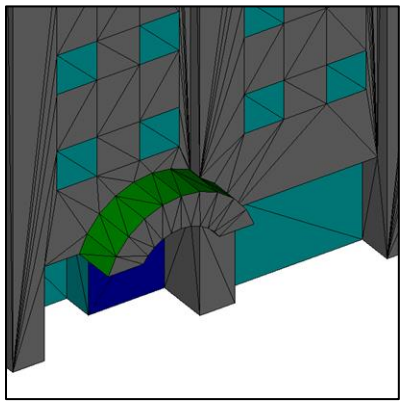
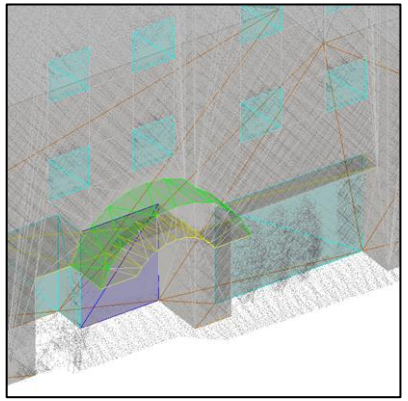


点群Bでモデリングした LOD3建築物と相違箇所

LOD3モデル作成の省力化検証

自動化処理前、自動化処理後の点群データから作成したLOD3建築物の比較検証

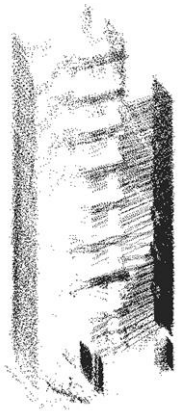
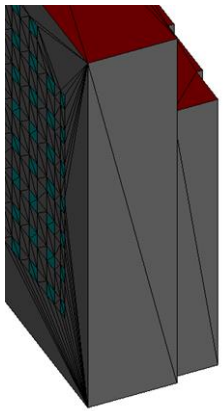
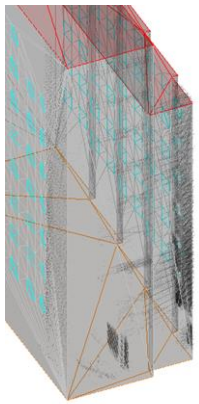
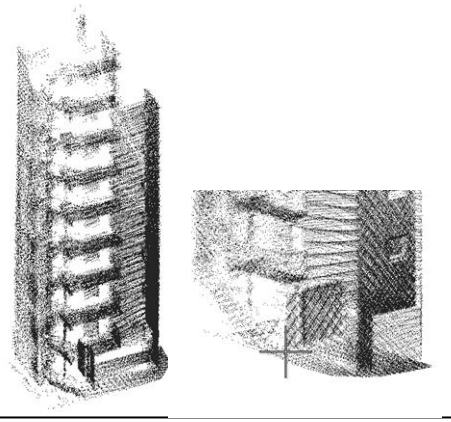
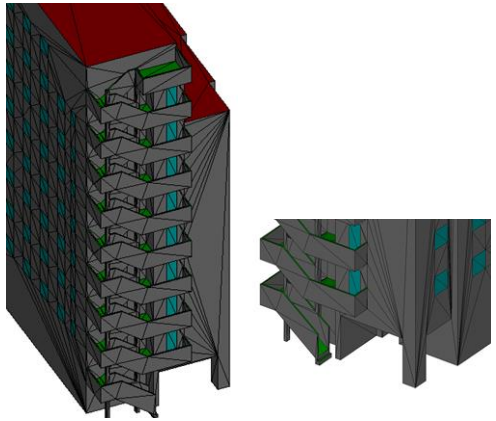
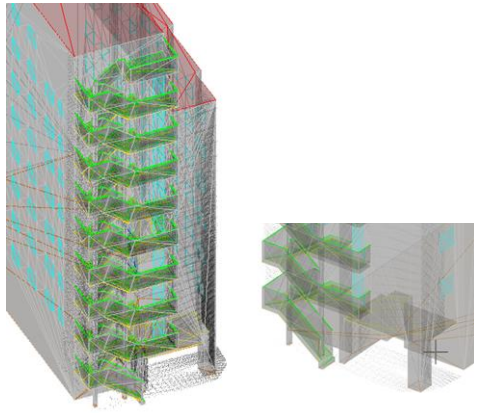
(1) ホテルセントラル仙台－①エントランス

	点群	作成したLOD3	点群+LOD3	評価
自動化処理前点群A				入り口の点群が不鮮明で、入り口を表現することができない。
自動化処理後点群B				入り口の凹凸が確認でき、写真を参考に、入り口を表現することができる。

LOD3モデル作成の省力化検証

自動化処理前、自動化処理後の点群データから作成したLOD3建築物の比較検証

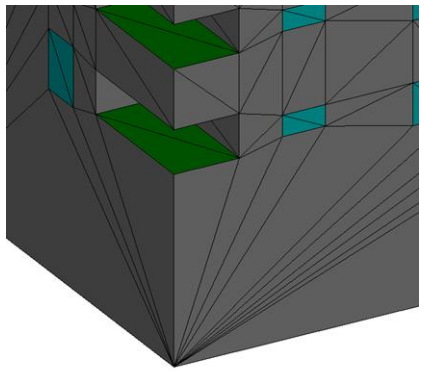
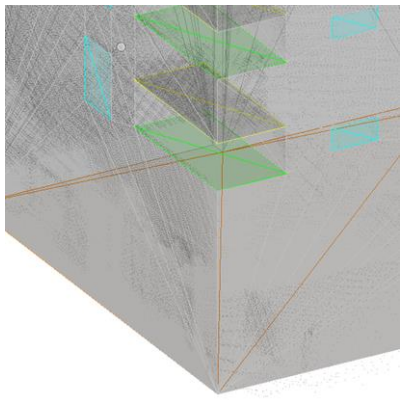
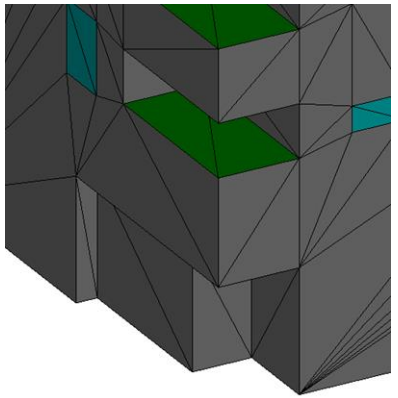
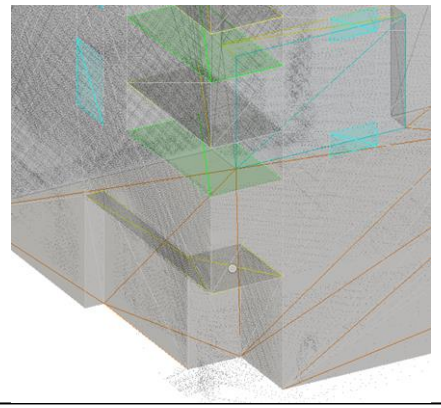
（1）ホテルセントラル仙台－②階段

	点群	作成したLOD3	点群+LOD3	評価
自動化処理前点群A				階段の表面部分しか写ってなく、奥行が分からないため、階段を表現することができない。
自動化処理後点群B				階段や柱の形状や奥行を確認することができ、写真を参考に階段や柱を表現することができる。

LOD3モデル作成の省力化検証

自動化処理前、自動化処理後の点群データから作成したLOD3建築物の比較検証

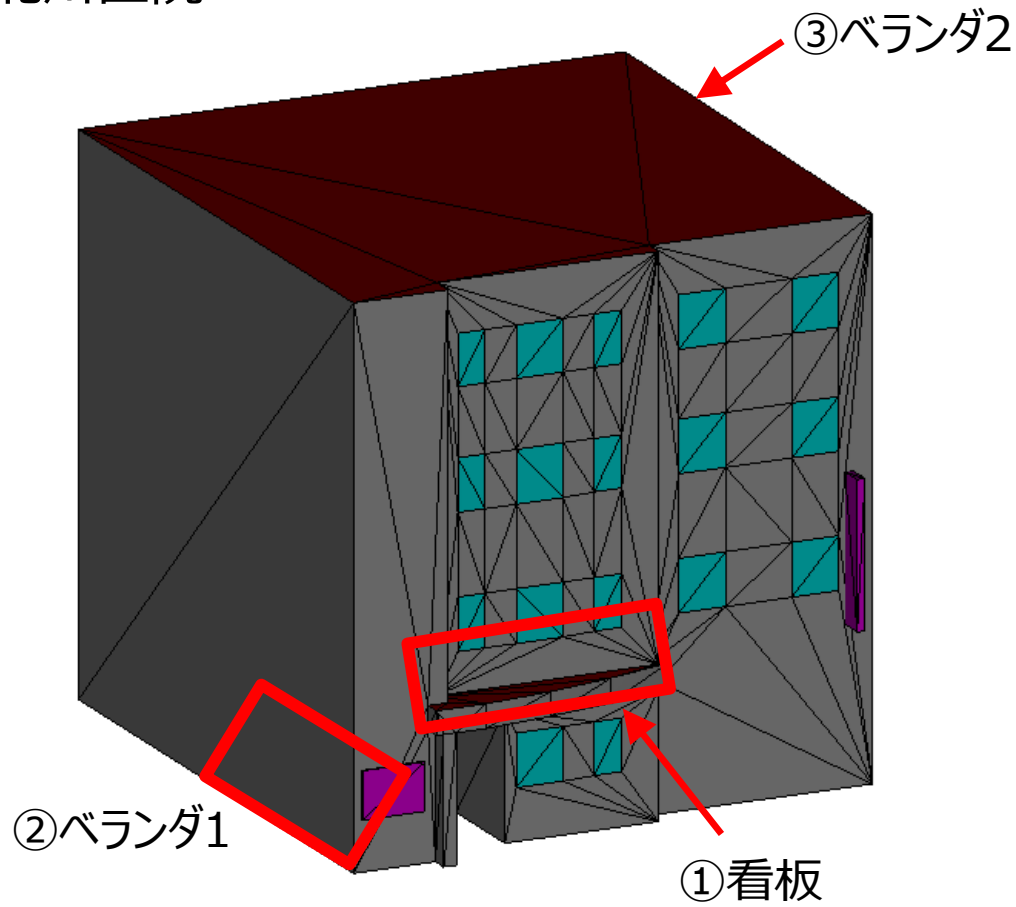
(1) ホテルセントラル仙台－③階段下

	点群	作成したLOD3	点群+LOD3	評価
自動化処理前点群A				奥行が判断出来ず、階段下のへこみを表現することができない。
自動化処理後点群B				階段下の凹凸が確認でき、写真を参考に、階段下のへこみを表現することができる。

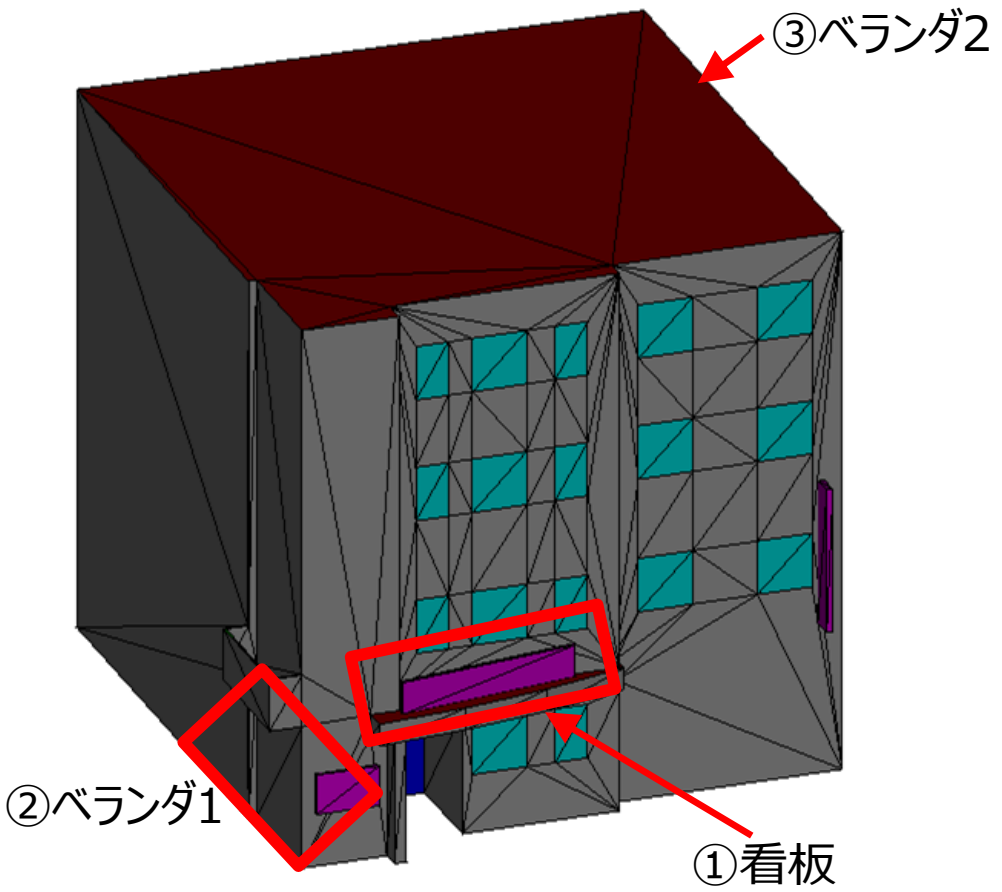
LOD3モデル作成の省力化検証

自動化処理前、自動化処理後の点群データから作成したLOD3建築物の比較検証

(2) 北川医院



点群Aでモデリングした LOD3建築物と相違箇所

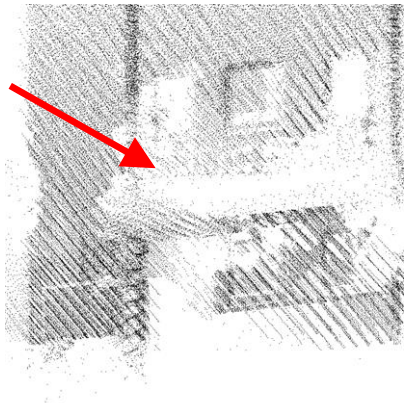
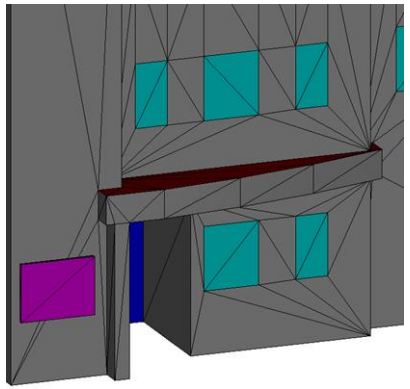
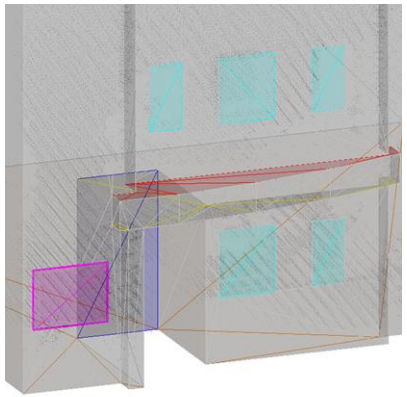
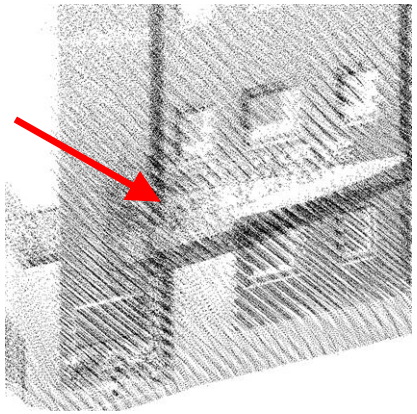
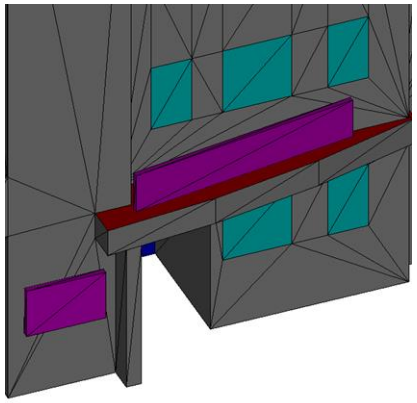
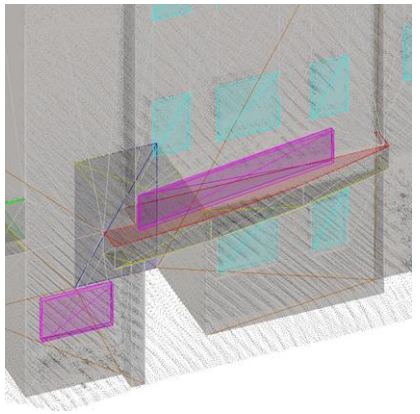


点群Bでモデリングした LOD3建築物と相違箇所

LOD3モデル作成の省力化検証

自動化処理前、自動化処理後の点群データから作成したLOD3建築物の比較検証

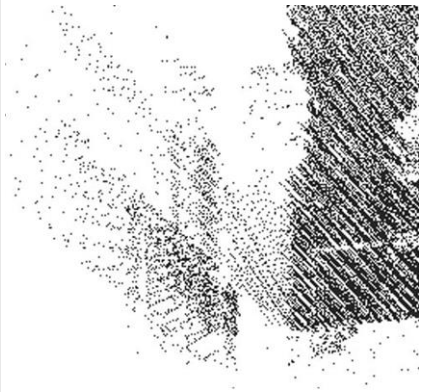
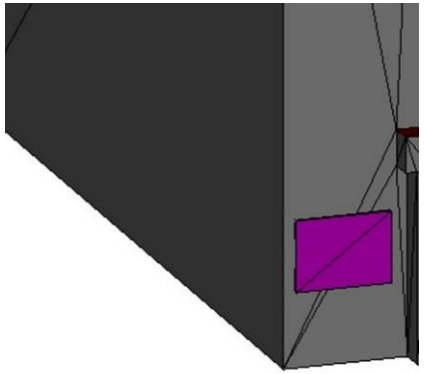
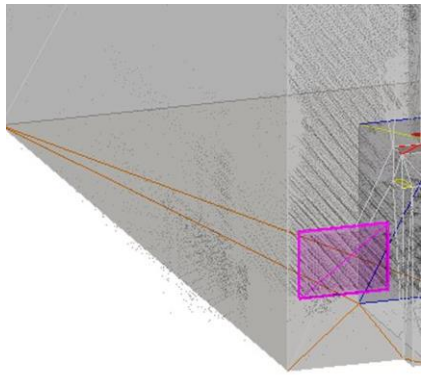
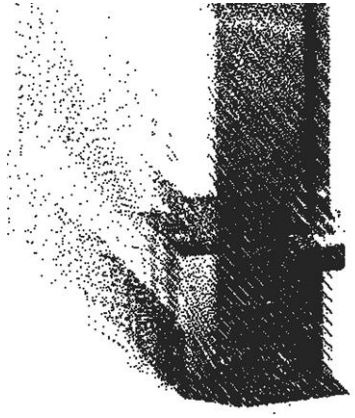
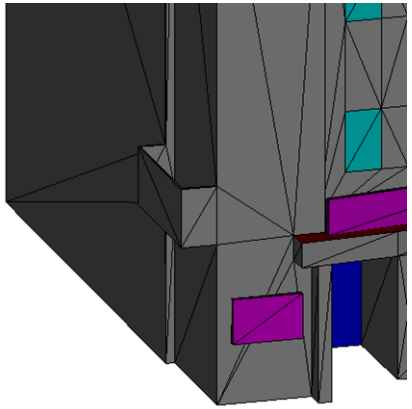
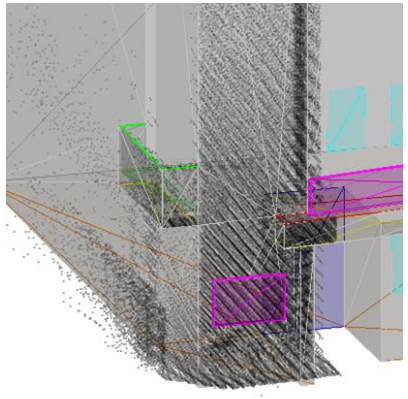
(2) 北川医院

	点群	作成したLOD3	点群+LOD3	評価
自動化処理前点群A				看板の点群が不鮮明で、看板を表現することができない
自動化処理後点群B				看板のおおよその形状が確認でき、写真を参考に、看板を表現することができる。

LOD3モデル作成の省力化検証

自動化処理前、自動化処理後の点群データから作成したLOD3建築物の比較検証

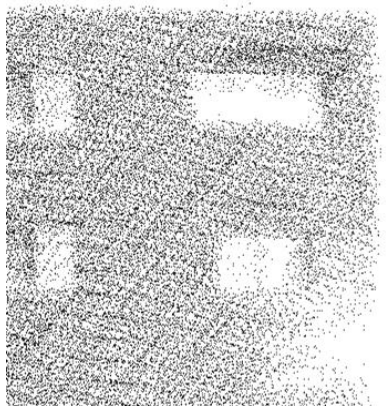
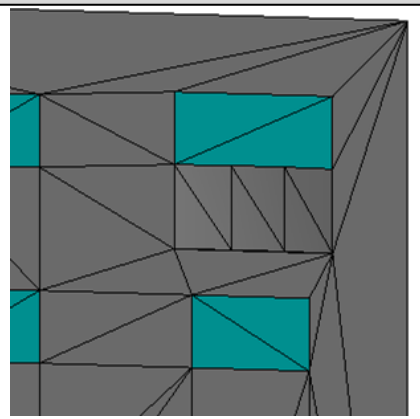
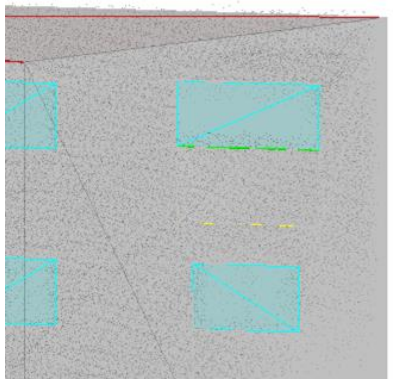
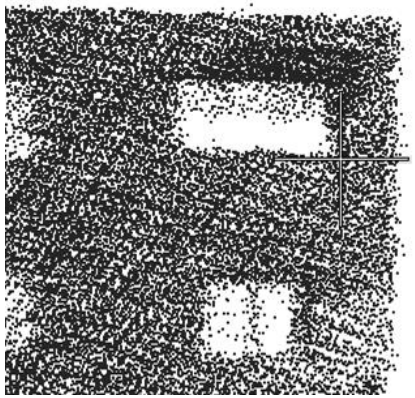
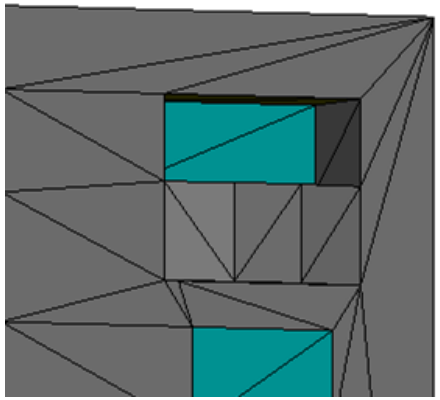
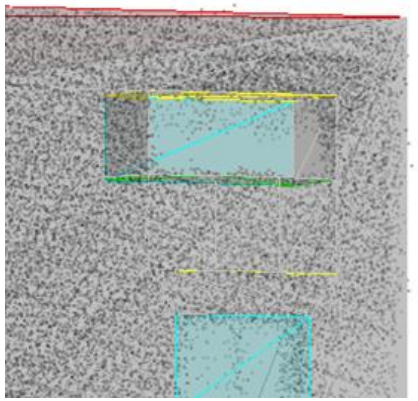
(2) 北川医院－②ベランダ1

	点群	作成したLOD3	点群+LOD3	評価
自動化処理前点群A				ベランダが写ってなく、ベランダを表現することができない。
自動化処理後点群B				ベランダの形状が確認でき、写真を参考にベランダを表現することができる。

LOD3モデル作成の省力化検証

自動化処理前、自動化処理後の点群データから作成したLOD3建築物の比較検証

(2) 北川医院－③ベランダ2

	点群	作成したLOD3	点群+LOD3	評価
自動化処理前点群A				ベランダの奥行きが判断できず、ベランダの奥行きを表現することができない。
自動化処理後点群B				ベランダの奥行きが確認でき、写真を参考にベランダの奥行きを表現することができる。

LOD3モデル作成の省力化検証｜まとめ

本調査研究の実証で得られた成果を下表に示す

- 従来どおりのMMS点群からのモデリングと比べて、本システムから自動生成されたサーフェスからのLOD3モデル作成業務では約32%、自動処理済の点群からでは約10.1%の省力化が図れた
- 一方、自動生成されたサーフェスからのLOD3モデルの自動生成は現状では平滑化が困難であり、本実証では自動化が図れなかった
- 公共測量成果に満たない簡易な点群からのLOD作成や、LOD2からLOD3を安価に作成するアプローチに期待が寄せられた

LOD3 建築物の作業能率（概算）

単位：時間

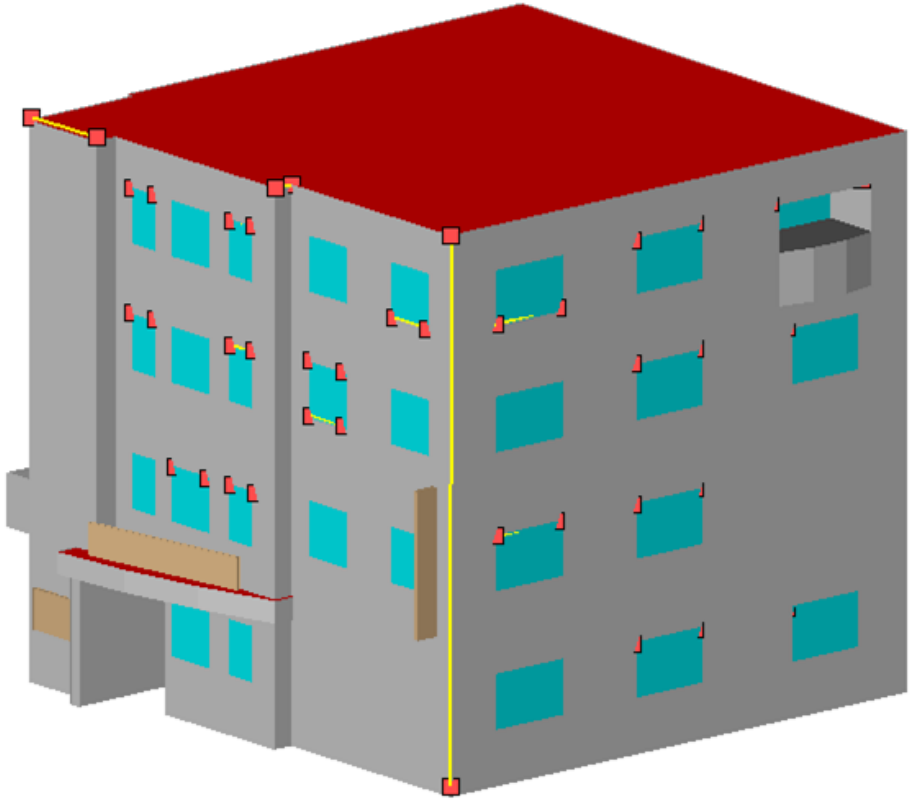
		A.I.による自動化処理の有無	ホテルセントラル仙台	北川医院	合計	省力化率	
1	従来のMMS点銀でモデリング	×	13.5	8.5	22		
2	点群データからのモデリング	○	12	8	20	90.90%	
3	サーフェスモデルでモデリング	○	9	6	15	68.20%	75%

本実証に寄せられた意見（2社：国際航業株式会社、株式会社パスコ）
• サーフェスモデルの形状を修正してLOD3を作成すると効率が悪いが、サーフェスモデル上の必要な点の座標を取得して新たに図化することで、効率は非常に良くなる。 • サーフェスモデルは、建物の形状やエッジの位置が点群と違い重なったり透けたりせずはっきり見える為、直感的にも取得する位置や面を把握し易い。 • CAD上でWindowを複数個開いて見る必要が無く、モデリング作業の効率が点群に比べ非常に良い。
• 骨格となる形状は、LOD2をそのまま使用する、もしくはLOD2を修正する事ができるので、点群を一から図化する必要がない。点群の当たっていない面がなくなり、LOD2との接合や、LOD2を参考にして新規モデリングする必要がなくなる。 • この手法では、MMSで取得されていない部分については、ドローンによる自動航行システムの運用、またiPhone等により簡易に取得された点群を自動処理をするデータに加えて補間することが検討できる。

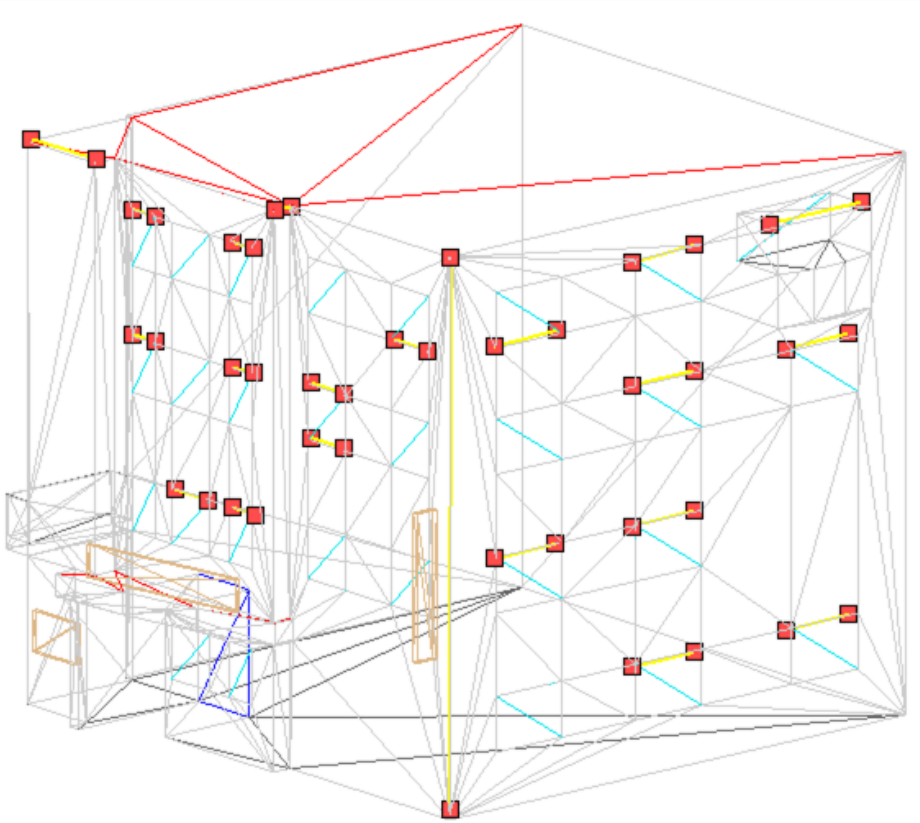
LOD3モデルの品質評価検証

サーフェスモデルから作成したLOD3モデルの品質評価

比較結果



精度評価

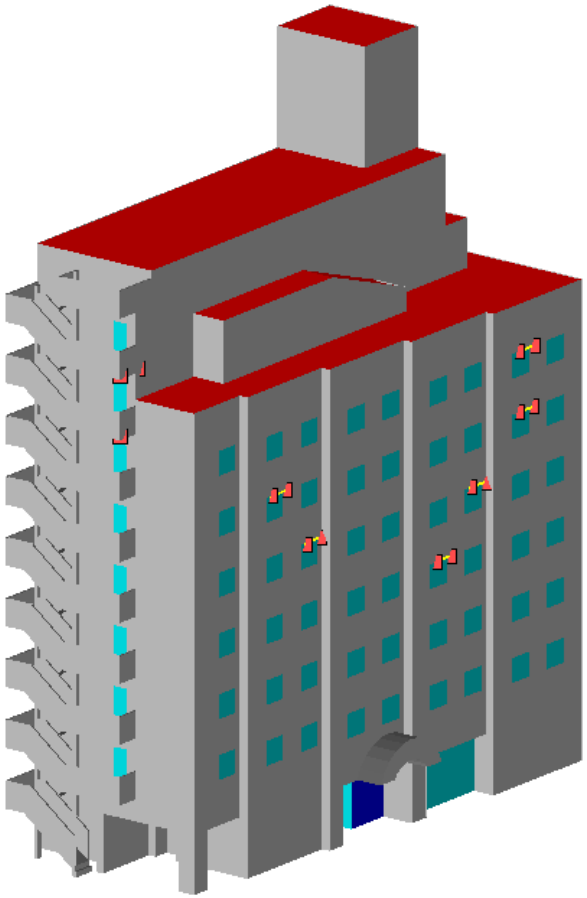


検証用に図化した辺

LOD3モデルの品質評価検証

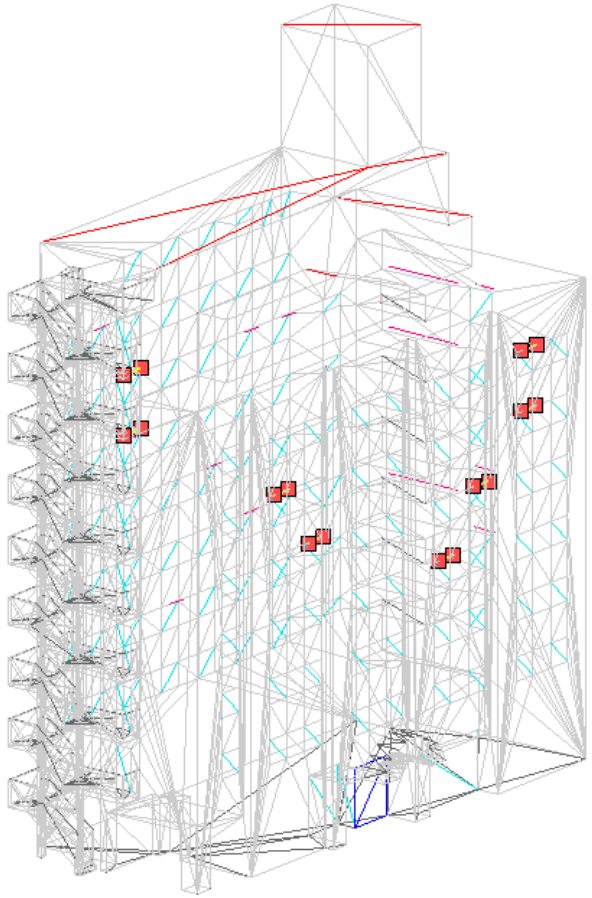
サーフェスモデルから作成したLOD3モデルの品質評価

比較結果



検証用に図化した辺

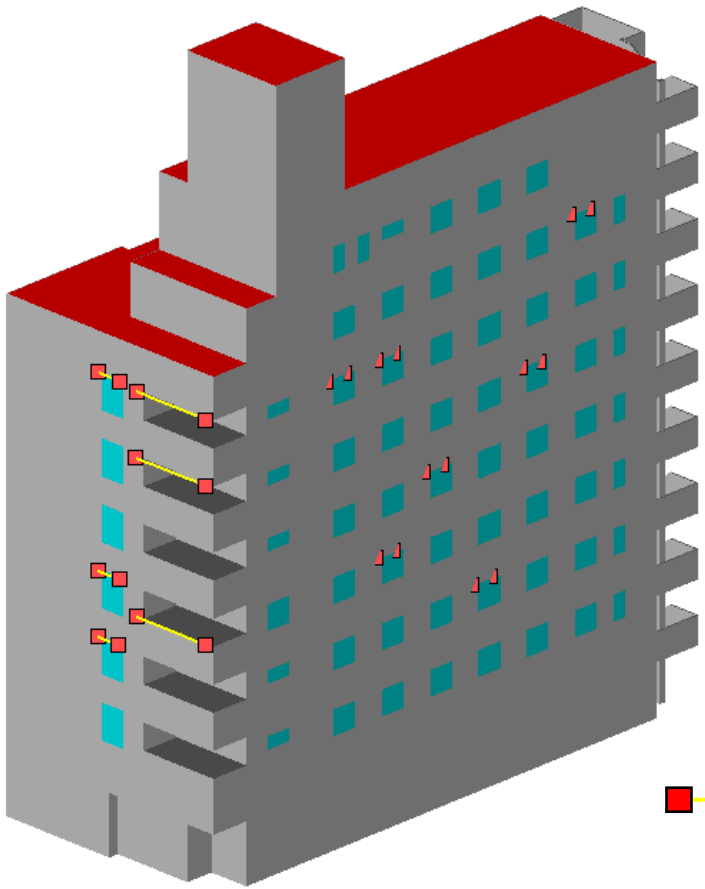
精度評価



LOD3モデルの品質評価検証

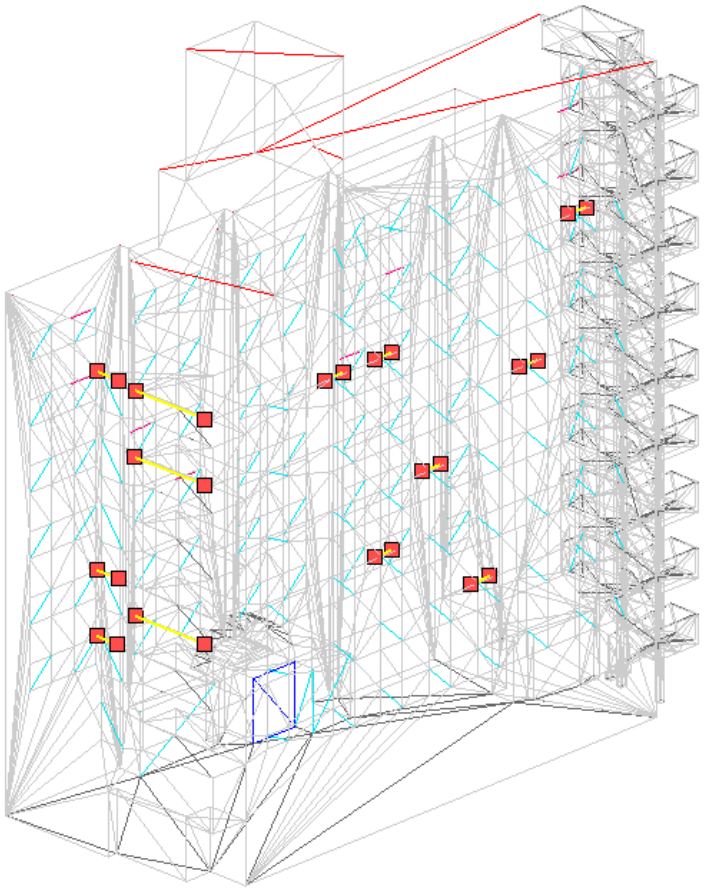
サーフェスモデルから作成したLOD3モデルの品質評価
ホテルセントラル仙台（モデルB - 裏面）

比較結果



検証用に図化した辺

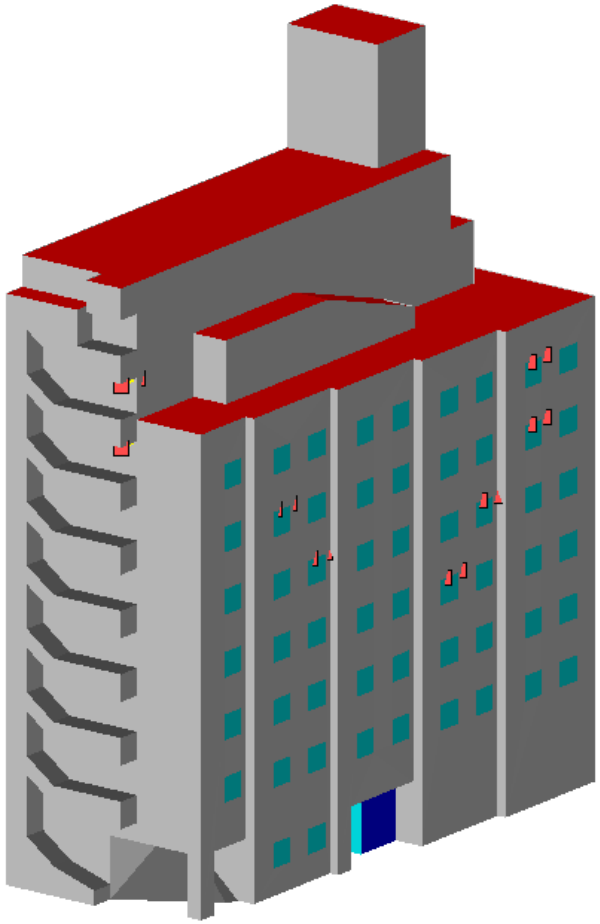
精度評価



LOD3モデルの品質評価検証

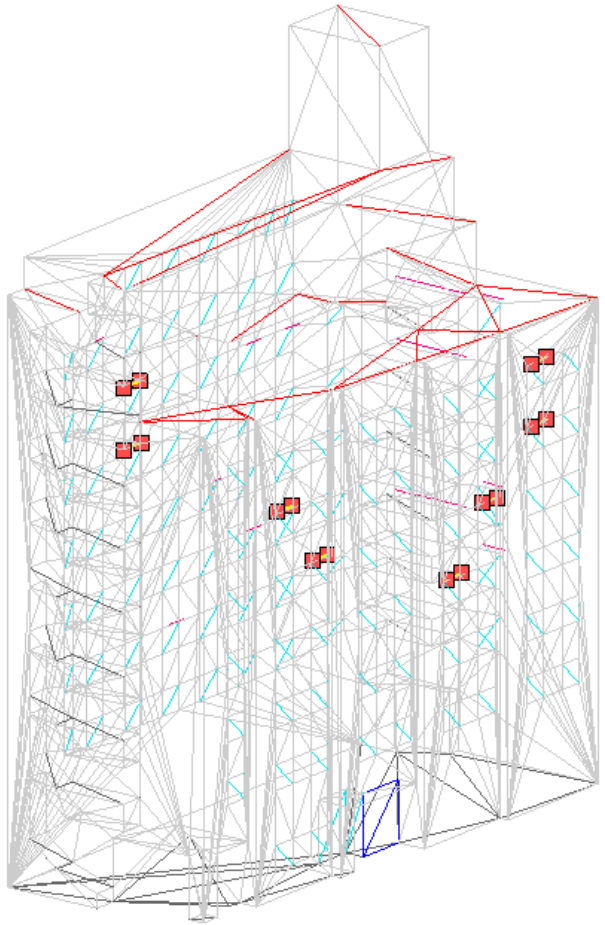
サーフェスモデルから作成したLOD3モデルの品質評価
ホテルセントラル仙台（モデルD - 表面）

比較結果



検証用に図化した辺

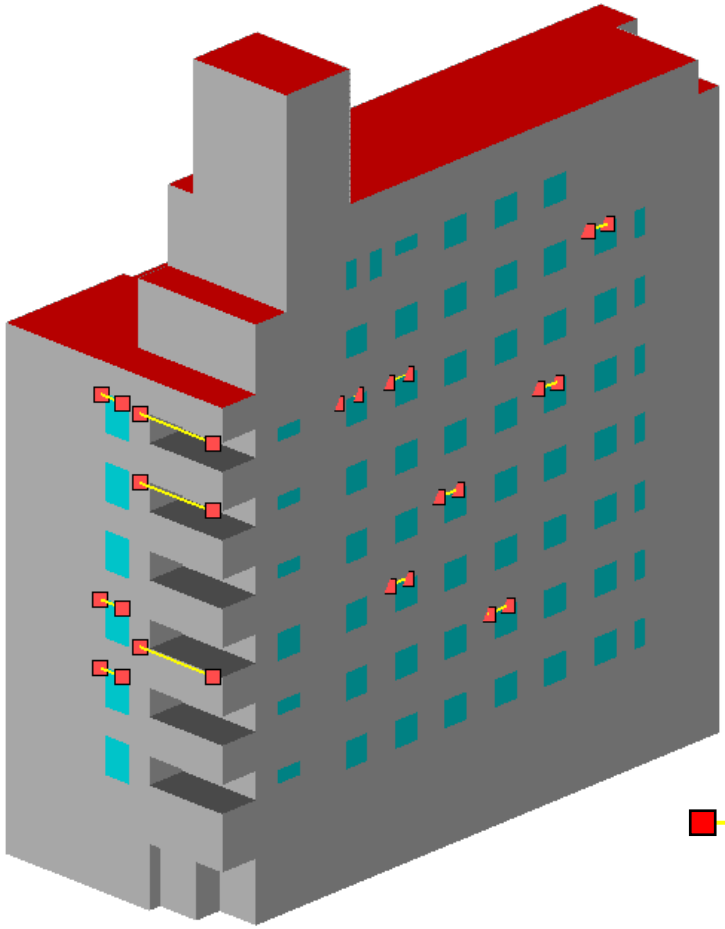
精度評価



LOD3モデルの品質評価検証

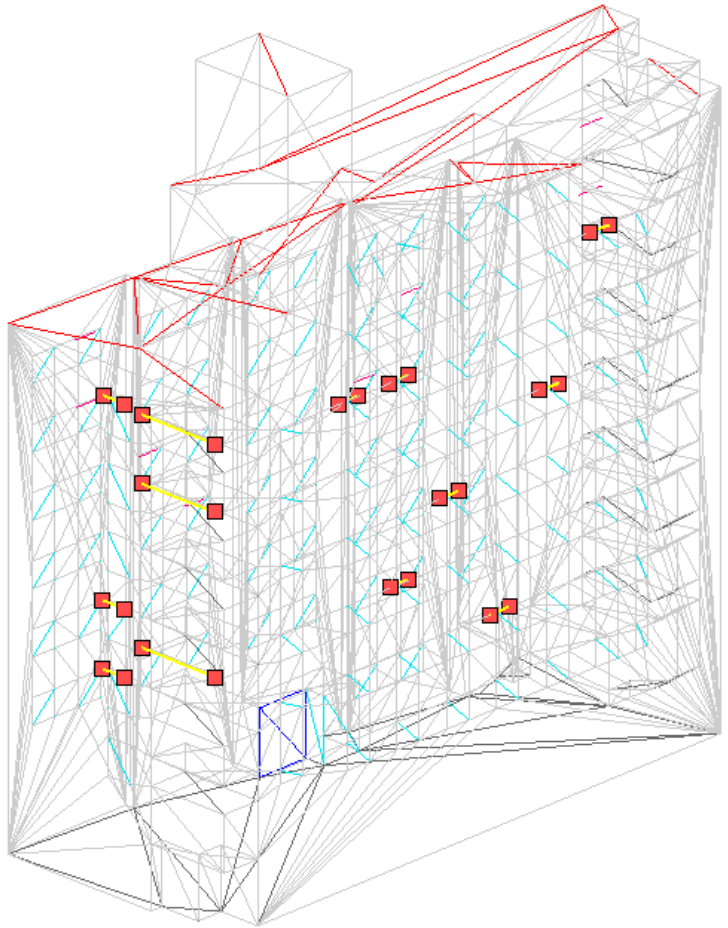
サーフェスモデルから作成したLOD3モデルの品質評価
ホテルセントラル仙台（モデルD – 裏面）

比較結果



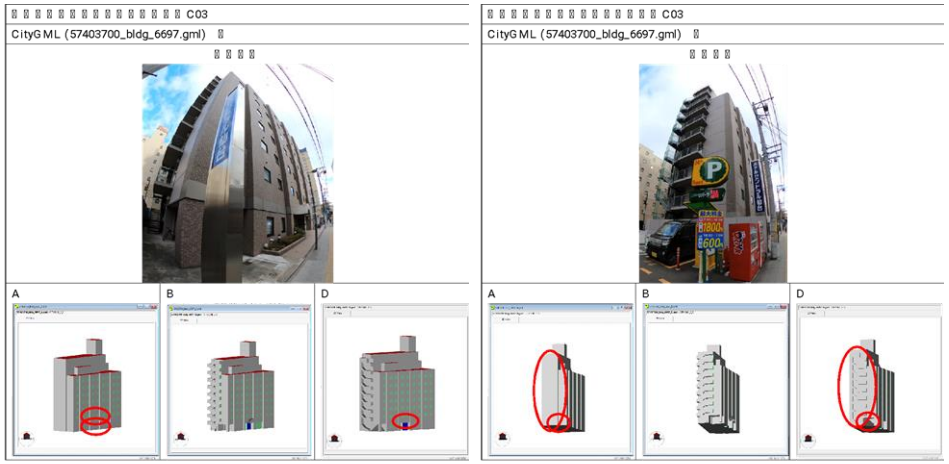
検証用に図化した辺

精度評価

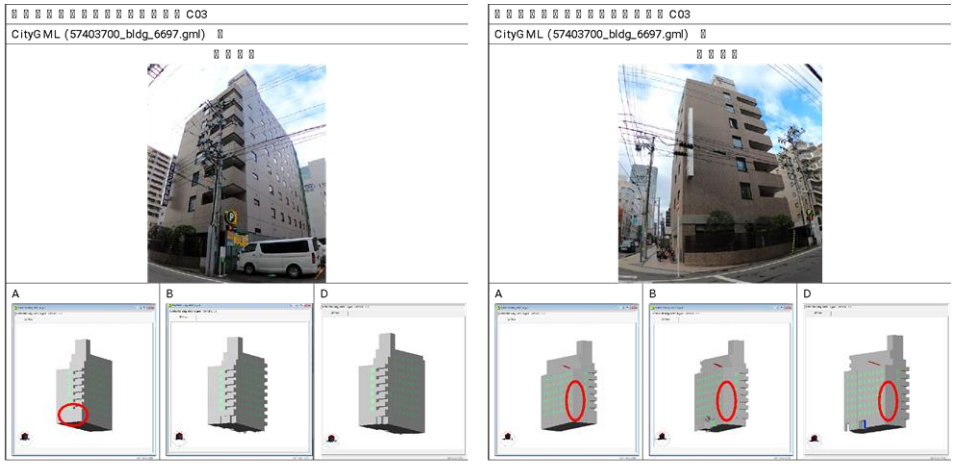


LOD3モデルの品質評価検証

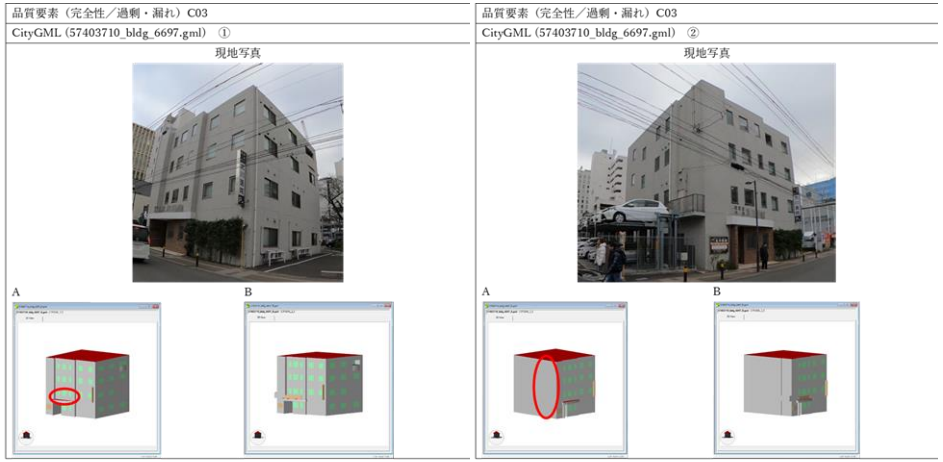
サーフェスモデルから作成したLOD3モデルの品質評価



ホテルセントラル仙台（モデルA/B/D）



ホテルセントラル仙台（モデルB/D）



北川医院（モデルA/B）

北川医院（モデルA/B）

LOD3モデルの品質評価検証｜まとめ

サーフェスモデルから作成したLOD3モデルの品質評価

3D都市モデルデータの品質評価結果

データ品質適用範囲：【地物型】建築物（LOD3）		CityGML (57403700_bldg_6697.gml) ホテルセントラル				CityGML (57403710_bldg_6697.gml) 北川医院			
品質要素		No.	モデル別（エラー数／標準偏差）			No.	モデル別（エラー数／標準偏差）		
			A	B	D		A	B	-
完全性	過剰・漏れ	C01	0	0	0	C01	0	0	-
		C03	6	1	4	C03	4	0	-
		C-bldg-01	0	0	0	C-bldg-01	0	0	-
論理一貫性	書式一貫性	L01	0	0	0	L01	0	0	-
	概念一貫性	L02	0	0	0	L02	0	0	-
		L03	0	0	0	L03	0	0	-
	定義域一貫性	L04	0	0	0	L04	0	0	-
		L05	0	0	0	L05	0	0	-
		L06	0	0	0	L06	0	0	-
	位相一貫性	L07	0	0	0	L07	0	0	-
		L08	0	0	0	L08	0	0	-
		L09	0	0	0	L09	0	0	-
		L10	0	0	0	L10	0	0	-
		L11	0	0	0	L11	0	0	-
		L12	0	0	0	L12	0	0	-
		L13	0	0	0	L13	0	0	-
		L14	0	0	0	L14	0	0	-
		L-bldg-01	0	0	0	L-bldg-01	0	0	-
		L-bldg-02	0	0	0	L-bldg-02	0	0	-
		L-bldg-03	0	0	0	L-bldg-03	0	0	-
		L-bldg-06	0	0	0	L-bldg-06	0	0	-
位置正確度 2500	絶対正確度	P05	対象外	許容範囲内 0.37m	許容範囲内 0.43m	P05	対象外	許容範囲内 0.12m	-
	外部正確度	P06	対象外	許容範囲内 0.07m	許容範囲内 0.08m	P06	対象外	許容範囲内 0.06m	-
主題正確度	非定量的主題属性の正しさ	T01	0	0	0	T01	0	0	-
	定量的主題属性の正しさ	T02	0	0	0	T02	0	0	-
		T03	0	0	0	T03	0	0	-
	分類の正しさ	T-bldg-01	0	0	0	T-bldg-01	0	0	-
		T-bldg-02	0	0	0	T-bldg-02	0	0	-

位置正確度（座標値による比較）

モデル名	地図情報レベル	水平位置（標準偏差）	標高（標準偏差）	判定
		1.75 m	0.66 m	
北川医院B	2500	0.12 m	0.06 m	許容値内
ホテルセントラル仙台B	2500	0.37 m	0.07 m	許容値内
ホテルセントラル仙台D	2500	0.43 m	0.08 m	許容値内

【品質評価】
今回の実証に適用した品質評価手法を示す。

(1)完全性
完全性の評価は、Google Erath及びGoogle Street Viewm p 全方位画像と作成されたLOD3都市モデルを目視で照合し、ジオメトリの過不足を検査した。本実証ではテクスチャ作成を行っていないため、テクスチャについては対象外とした。

(2)論理一貫性
Project PLATEAUで開発された位相一貫性の論理検査ツール(CityGML-geometry-validator)により検査した。エラー箇所の目視での確認は、FKZViewerを使用した。

(3)位置正確度
測量器による点群データを真値として、外壁のエッジ部分、ドア、窓などの特徴点で座標値を比較検査した。いずれも地図情報レベル2500の要求品質を準用する。

Ⅰ．実証概要

Ⅱ．実証技術の概要

Ⅲ．技術調査

Ⅳ．実証システム

Ⅴ．実証技術の検証

Ⅵ．成果と課題

今年度の実証で得られた成果①

本調査研究の実証で得られた成果を下表に示す

項目	実証で得られた成果
ノイズ除去（セマンティックセグメンテーション）	• 屋外での広範囲に渡る地理空間情報の取得、そのデータを活用した都市レベルの大規模なA.I.モデルの構築は世界的に見ても実証数が少なく、新規性の高い取り組みである。本実証では、Open3Dライブラリに含まれる「RandLA-Net」をベースに、仮想空間におけるトレーニングデータ自動生成ツールを活用し、物理空間のデータを一切使用せずに、仮想的な物理シミュレーションから生み出された点群データのみを用いたA.I.モデルの構築を行った結果、人・車・植生・壁・フェンス等の13の有効なラベルによる物理空間データへのセグメンテーション精度が大きく向上した。これは、A.I.モデルでの学習が演算処理やネットワークの高速化により指数関数的に学習速度・精度の向上が期待できることを意味している。 • 本実証でのセマンティックセグメンテーションは、対象の建物以外の地物の除去が目的であったが、対象の地物に対しての属性付与も可能であり、今後は建築物だけではなく、道路・植生・都市設備等、屋外でのスキャンに対する属性付与、3Dモデル化に期待ができる。 • 一方で、LOD3作成に必要な地物の構造（3D都市モデル標準製品仕様第2.x版に記載）など、必要となる開口部、付属物、屋根等のラベリングが行われておらず、早急な対応が必要であり、3Dモデル自動化への大きな課題がある。この問題はやや複雑で、A.I.モデルの構築と併せて、取得時に欠落した点群データの補間、新たなデータソースの検討が必要である。 • 自動化に向けては、新たなラベリングによるトレーニングデータ作成及び学習とあわせて、本実証では採択しなかった、トレーニングデータを必要としない、符号非依存学習（SAL:Sign Agnostic Learning）等の手法も検証の必要がある。
建物抽出（2Dフットプリントポリゴン生成・拡張）	• 建物の個別抽出のためのLOD2建築物モデルを用いたフットプリント生成及び建物境界の拡張では、汎用的な技術が存在しなかったためフルスクラッチでの実装を行った。本実証エリアの宮城県仙台市都市再生緊急整備地域の全2,604棟の100%の個別出力及び建物境界の拡張を可能にした。 • 日本の都市部では物理空間に存在する建物の多くが隣接して存在しているため、建物近傍に位置する点群データのノイズを自動除去する場合などでも有用性が高い技術である。

今年度の実証で得られた成果②

本調査研究の実証で得られた成果を下表に示す

項目	実証で得られた成果
点群合成（アラインメント）	• 日時や環境条件、取得デバイスの異なる断片化した点群データや、公共測量成果に寄らない安価な点群データから個別の建物の点群データを出力する点群合成は、本実証システムで目指す3D都市モデルの自動更新の重要なモジュールの１つである。本実証では、バス、タクシーを想定した一般車両、iPhone LiDARなど複数の異なる環境で取得された複数の点群を用いて、3D都市モデルに準拠した建物IDごとに、建物1棟単位で合成された点群データを自動出力した。また、撮影距離、LiDARセンサー取付角度、高層階、隣接して車両や人が通行できない建物間など、取得データに欠損が生じている点群データに対し、LOD2建築物モデルを自動合成し補間を行っている。これは、3D都市モデルデータをオープンデータとして公開している日本ならではの手法であり、今後、より詳細度の高いLOD3、LOD4が整備されることで利用可能地域や利用用途の広がりに期待できる。
更新箇所特定	• DCPCR（大規模な屋外環境での深層圧縮点群登録）とGICP（反復計算による近接点適合）アルゴリズムを用いた異動判読処理によって過年度の3D都市モデルと新規に取得した点群データの差異検出を行い、新たに更新された建築物（新築・滅失・滅失から新築・増改築）の検出と建物IDの抽出では、実証エリア内の建物棟数を母数とした正解率で95.4%以上と、非常に高い精度となり、A.I.で新築と判定された建築物は高い適合率で正しく判定することができた。 • しかし、バスに実装されたMMSにより複数回、繰り返しデータが取れている地域の算出率は高かった一方で、タクシーを想定した一般車両に実装されたMMSで撮影された地域では、新築・、滅失共に著しく精度が落ちる結果となった。これは、タクシーを想定した一般車両での撮影回数が低かったことに起因していた。点群データによる更新箇所特定や、3Dモデル化のためのデータソースとして検討する場合、いかに多くの回数、網羅的なデータを取得するかがA.I.モデル活用時の大きな課題である。

今年度の実証で得られた成果③

本調査研究の実証で得られた成果を下表に示す

項目	想定される技術面での優位性
可搬型簡易MMS	本実証で活用した簡易MMSは、今後の公共交通での地理空間情報取得や、車両等に後付けできる安価なLiDARセンサーとして期待できるものである。1台ごとの単一性能では、据え置き型の測量器が勝るが、本実証でアプローチした複数の断片化したデータの合成による3Dモデル化には力を発揮すると考える。今後、公共交通、物流、一般車両も含めて、広域のデータを低コスト・高頻度で大量に取得を行い、市民起点でデータを生み出すためのビジネスモデルが必要であると考える。
メッシュ再構築	• 点群データからの3Dメッシュ再構築処理は、2022年末に発表された「iPSR（改良型ポアソン表面再構成）」の登場により、従来の「PSR」では困難だった法線ベクトルの反復生成及びメッシュ形状の推定による点群データの直線エッジから直線メッシュへの変換が実現し、安価なLiDARスキャナからでも精度の高いサーフェスの再構築が可能となった。しかし、本実証ではLOD3モデルの完全な自動モデリングまでは至らない成果となった。 • 3D図化の観点では、本実証で作成したサーフェスモデルからLOD3建築物モデルへのモデリング化作業において一定の成果があった。従来のMMS点群データでは建物の幾何形状が層となり、3Dモデル作成時の面やエッジの把握に作業時間が割かれていた。本実証で点群から自動生成されたサーフェスモデルでは、建物形状やエッジの位置がはっきりと視認できるため、必要な点座標を取得しやすく、従来のMMS点群からのモデリングと比較して1棟につき13.5時間から9.0時間へ、44.4%の大幅な作業間の短縮を行うことができた。これは、サーフェス再構築の技術単体での精度向上ではなく、点群からのノイズ除去・点群合成・建物抽出、といった本実証の個別モジュールのシステム連携により実現した成果である。
窓検出	• 窓の形状・位置の検出について、点密度及び形状ベースでの検出アプローチを行い、一定値の窓の検出及び3Dモデル化を実現した。しかし実用レベルでの3Dモデル化には至らない成果となった。LOD3モデル化作業において、窓の形状と位置の取得は大きな省力化を生み出すと考えられるが、本実証における点群データの形状情報のみでは検出精度に限界があり、特に建物に対してのLiDARセンサーの角度と距離により誤検出、非検出が増加した。色情報及び反射率や反射強度情報を取得することで、更なる検出精度の向上が検討できる。

3D都市モデルによる技術面での優位性

項目	想定される技術面での優位性
セマンティックセグメンテーションの分類精度の向上	• 本年度の実証では3D都市モデルの自動更新のためのA.I.モデルのトレーニングを行うために、自動生成された仮想空間を活用した。今後3D都市モデルの整備自治体が増加し、セマンティクスな属性データを持つ3D都市モデルを活用することで、仮想空間内に現実の都市空間を詳細に再現し、詳細にラベリングされたトレーニングデータの生成を行うことが期待できる。物理シミュレーションなどと連携することで、物理空間では再現が困難な事象のA.I.モデルの効率的な学習が可能となる。
補正精度の向上	• LOD3、LOD4などの3D都市モデルの整備自治体が増加する事で、低コスト、高頻度なLiDARによる点群データではスキャンが難しい路地裏や、建物と建物の間などデータの欠落することが多い箇所を補間し、建物の再現性を高め、LOD2からLOD3への詳細度更新、サーフェス再構築、3Dモデル作成の自動化等に期待ができる。
A.I.モデルのトレーニングデータ制作時のコスト削減	• 従来、A.I.を活用したトレーニングでは膨大なトレーニングデータが必要であり、人の手によるラベリング作業などトレーニングデータの製作コストが課題であった。PLATEAU の3D都市モデルを活用することにより、低コストで実現可能なトレーニングデータの自動生成が加速し、都市空間解析や地理空間情報ビジネスを加速することが期待できる。

活用にあたっての課題

今後の民間・自治体双方での活用に向け、3D都市モデルデータ整備更新業務に対応可能なソリューション開発が必要

項目	活用にあたっての課題
LOD3作成の自動化	**LOD3作成の自動化に向けて、さらに多くのデータソースやドローン等との連携が必要** • 本実証では、公共交通（バス及びタクシーを想定した一般車両）とiPhne LiDARによるデータソースの取得を行ったが、撮影ルートやセンサー角度、センサー撮影可能距離、隣接した建物同士の隙間など、車両進入が困難な場所や、LiDARセンサーの撮影可能範囲外の高層階等のデータ欠落が多く発生した。建物の全方位の情報を取得するためにはさらに多くのデータソースや、ドローンの自動航行システムとの連携等が必要である • LOD2以上の3Dモデル生成には、建築物の「壁」「屋根」「床」などの幾何形状の点群データへのラベリングが必須である。ラベリング済トレーニングデータに寄らない、符号非依存学習のような深層学習のアプローチが期待される。 • 3D都市モデルの更新には、定常的な都市データの取得・更新が必要である。そのためには国や自治体が取得・提供するデータだけでは人的コストや費用が追いつかないため、民間同士のデータ連携を図っていくことが期待される
更新業務全体におけるソリューション提供	**ノイズ除去や複数のデータソースによる点群合成等、細かな業務・条件へ対応するためのソリューション提供が必要** • 完全自動化に至るためには研究開発にまだ時間がかかると考えられるが、ステップごとのデータクリーニングや最適化においても個別に提供可能で省力化が図れるソリューションや提案を提供する必要がある。ユーザーニーズや更新業務全体のワークフローを正確に理解し、ソリューションや提案をする必要がある
精度向上	**最新のA.I.モデルの検証や、精度向上のための取り組み等、長期的な施策の実施が必要** • 最新のA.I.モデルの検証や、精度を継続的に高めていく取り組みが必要である • 3D都市モデルLOD3/LOD4を追加することで、仮想空間でのトレーニングデータ作成・精度向上が見込まれる。3D都市モデルを整備する更なる自治体の増加が期待される。

用語
用語

用語	内容
GNSS	• Global Navigation Satellite System（衛星航行システム）衛星から発射される信号を用いて位置測定、時刻配信を行う
サーフェス	• 3DCGでの表面を指す
再現率	• 見逃しをしていない程度を把握するための指標で、「正しくAと予測できたデータ数」÷「実際にAであるデータ数」で計算される • 数値が高い方が、見逃しが少ないことを意味する
深層学習	• 第3次人工知能ブームの中心的な技術 • 画像認識・テキスト解析・音声認識等の複数の領域で、研究開発・実務利用が進んでいる
セマンティックセグメンテーション	• 2D/3D上で、特定のオブジェクトや地物などをラベリングしたがって分類する処理を指す
適合率	• 誤検出をしていない程度を把握するための指標で、「正しくAと予測できたデータ数」÷「Aと予測したデータ数」で計算される • 数値が高い方が、誤検出が少ないことを示す
トレーニングデータ	• A.Iによる機械学習アルゴリズムを適切にトレーニング（学習）を行う際に使用する、A.I.の判断の指標となるデータを指す • 画像を用いた教師データの場合は、画像データと画像データに対応したラベル（画像内の領域を分類したものや画像に名称を付けたり種類がある）を付与したペアになったデータセットが必要になる
メッシュ	• 国土地理院が中心になって整備をしている電子地図 • 国・地方公共団体・民間事業者等が地図整備をする際に基準とする地図
反射率	• レーザースキャンを行う際の、入射するレーザー光に対する反射光の強度であり、対象表面の状態も反映する
フットプリント	• 本実証実験では3D都市モデルのLOD2から生成した建物外形を指す
法線ベクトル	• ある平面に対する垂直な線を指す
メッシュ	• 3DCGで立体を表すデータ形式であり、ポリゴン（多角形）が集合した構造の計上表現を指す
LiDAR	• Light Detection And Ranging（光による検知と測距）。光を用いて対象物までの距離、位置、形状を検知できるセンサー機器
ラスターデータ	• グリッド上に構成されたデータを指す

デジタルツイン構築に向けた3D都市モデルの更新に関する調査研究業務
技術資料

令和5年3月　発行
委託者：デジタル庁
受託者：3D 都市モデルの更新に関する共同提案体

本報告書は、Symmetry Dimensions Inc.、名古屋鉄道株式会社、中日本航空株式会社、宮城交通株式会社、国際航業株式会社、株式会社パスコが、デジタル庁との間で締結した業務委託契約書に基づき作成したものです。受託者の作業は、本報告書に記載された特定の手続や分析に限定されており、令和5年3月までに入手した情報にのみ基づいて実施しております。従って、令和5年4月以降に環境や状況の変化があったとしても、本報告書に記載されている内容には反映されておりません。