DEIM Forum 2022 A41-4

動的点群のデータベースを用いた管理手法

松本　佳大† 　杉浦　健人† 　石川　佳治† 　陸 　可鏡†

† 東海国立機構名古屋大学大学院情報学研究科　〒 464–8603 愛知県名古屋市千種区不老町
Email: matsumoto.yoshihiro.k3@s.mail.nagoya-u.ac.jp, {sugiura,lu}@db.is.i.nagoya-u.ac.jp,
ishikawa@i.nagoya-u.ac.jp

あらまし　3 次元空間上の点群データに対するリアルタイムでの取得および活用は，点群データの取得に使用される LiDAR センサの普及もあり進んでいる．しかしこのような点群データの取り扱いにおける既存の技術は，ある特定の時点のスナップショットを表す点群を主な処理対象としており，時系列に基づいて膨大な量の点群データを蓄積する用途においてはその効果を十分に発揮できない．そのため本研究では，既存のデータベース技術を応用した時系列点群データの効率的な管理方法について検討する．格納済みの点群データで差分抽出を行い近似した点群を削除することによる点群データの圧縮手法や，データベース内の時間軸の範囲に基づいた点群の検索手法について提案する．

キーワード　動的点群データ，LiDAR データ，時系列データ管理．

1 はじめに

1.1 背　　景

光の照射で物体との距離を計測する *LiDAR*（*light detection and ranging*）センサの普及により，点群データのリアルタイムでの取得およびその活用が可能となった．LiDAR はレーザー光を周囲に照射し，物体に反射して跳ね返るまでの時間を計測することで物体との距離や方向を測定する．具体的な活用例としては飛行機から点群データを観測し，地形調査を実施した事例が挙げられる．また，GPS のデータを正確に得られない屋内でも点群データを獲得し，物体認識や自己位置推定が可能となる利点から自動運転の分野でも研究が進んでいる [1].

一方，機器の普及に伴い異なる観点からの活用も期待され始めている．例として LiDAR センサを用いた混雑状況の予測が考えられており，街頭に LiDAR センサを設置したとき得られる点群データを解析することで点群の変動に基づく混雑状況の推定が行える．

しかし，点群データのリアルタイムな管理および処理を考えた場合，その性能やそもそものデータ保持方法に関して課題が発生する．LiDAR センサは周辺に存在する物体の表面情報を点群データとして取得するため，継続して周辺状況を監視する場合，データの取得間隔に応じて膨大な量の点群データが蓄積される．ある特定のスナップショットを表す点群が処理対象となる技術では，このような**動的点群データ**を効率的に扱うことができない．そのため，時間軸に対応した点群データの管理手法を構築する必要がある．位置情報以外の属性を持つ点群データの管理手法に対しては現在，MPEG-PCC をはじめとしたプロジェクトが研究を進めており，そこで提案された Geometry based Point Cloud Compression（G-PCC）と呼ばれる圧縮手法は八分木を用いて 3 次元空間を分割することで，点群の位置情報に加え RGB といった色の情報を含む点群を圧縮する [2].

1.2 目的と貢献

本研究では G-PCC をはじめとした既存技術を応用し，LiDAR センサから得られる動的点群データのデータベースにおける管理手法について提案する．特に本稿では，既存の点群データの管理手法についてまとめ，動的点群データの効率的な管理方法について検討する．

本稿の貢献を以下に示す.

- 動的点群データをデータベースで管理するためのアーキテクチャおよびデータフローの提案
- 事前に取得した同じ地点での点群データを活用する，動的点群データの圧縮機能の提案
- データベースの機能を活用した時空間データとしての動的点群の検索機能の実装

1.3 構　　成

本稿の構成は次の通りである．まず 2 章で本稿の関連研究を述べる．次に，3 章では本研究で提案する動的点群データのシステムについて記述し 4 章でシステムの管理，検索，圧縮における具体的な機能をそれぞれ提示する．5 章で実験により提案システムの性能を検証し，最後に本研究の結論を 6 章で述べる．

2 関連研究

本章では関連研究として，ファイルに基づく点群管理とデータベースを用いた点群管理について述べる．そしてその関連研究を踏まえ，本研究の貢献について提示する．

2.1 ファイルに基づく点群管理

ファイルを用いた管理方法について述べる．PLY や OBJ など，様々な企業や団体によって 3 次元オブジェクトを表すためのファイルフォーマットが開発されており，点群データもそれらのフォーマットで管理できる [3]．ファイルに基づくデータ管理は伝統的な方法であり，使用できるソフトウェアやライブラ

リが多い，フォーマットに応じた圧縮方式が存在するなどの利点がある．ただし，対応するソフトウェアさえあればデータの読み込みを簡単に行える一方，フォーマットが異なる点群データの管理が煩雑となるなどの問題もある．検索効率の向上のために，一部のファイルフォーマットではファイル内に別ファイルへのリンクを埋め込むことによって，ファイル上で点群データの索引を構築できる．また，ファイル内に別ファイルのパスを保持することにより参照が可能となる．ファイルの例としては，LiDAR センサで得られる点群データの標準的なフォーマットである LAS ファイルが空間充填曲線と八分木の 2 つの索引を利用できる [4]．一方で格納効率の向上のために，一部のファイルフォーマットでは効率的な圧縮手法も開発されている．例えば，LAS の圧縮フォーマットとして Rapidlasso の LAZ（LASzip）や ESRI ZLAS が存在する．ただし，こうした圧縮は処理するファイルの数やサイズの増加によるスケーラビリティの問題を持つ．

2.2 データベースでの点群管理

データベースを用いた点群管理として，pgpointcloud を使用した点群管理について述べる [5]．pgpointcloud は OSS（open source software）データベースの 1 つである PostgreSQL の拡張モジュールであり，PcPoint と呼ばれるデータ型を定義する．PcPoint 型は複数の次元，または各次元に対して複数の属性を設定可能であり，LiDAR センサの持つ様々な変数を保持できる [6]．例えば各点の RGB カラーモデルなどの定義が可能であり，他にも検出時間といった LiDAR センサから得られる位置情報以外のデータも効率的に格納できる．更に，点群データをテーブル上で効率的に保持するために，PcPoint 型をまとめた PcPatch と呼ばれるデータ型も使用できる．データベースのテーブル上では 1 つのデータを 1 つのタプルで表すが，点群管理において点データを個別に扱うのは非効率的である．そのため，PcPatch 型は PcPoint 型を集約した点群ないし部分点群を 1 つのデータとして格納し，タプルとしての管理を容易にする．これにより，膨大な量のタプルを保持することで生じるオーバーヘッドの問題も回避できる．点群データを操作する際は，この PcPatch 型に対してフィルタリングや索引の作成といった処理を施す．

2.2.1 pgpointcloud を使用した点群の格納

pgpointcloud を用いて点群を保存する際には，1 つの点単体を 1 つのタプルに逐一保存するのではなく，PcPatch によって点の集合として保存する．これにより数十億個の点群データも数百万程度の PcPatch 型として扱えるようになり，データベース内の行数を数桁分減少させることが可能となる．B^+ 木や R 木といった PostgreSQL に元々備わっている索引を利用する際にも，削減されたタプルに対して効率的に索引を構築できる．また，共通のフレームワークの提供や索引サイズの削減といったメリットもある．PcPatch における点群のグループの作成においては，点群は空間的な位置によって検索される傾向があるため空間の範囲に基づいて点群として抽出する．しかし，建造物や自然物をそれぞれ構成する点群といった意味付けできる情報があれば，その制約を基にした点群の作成も可能である [7].

2.3 点群の圧縮手法

ここでは，3 次元空間上の点群データの圧縮手法として 1 章で取り上げた G-PCC について述べる．これは多視点カメラから再構築された色情報を有する密な点群データや LiDAR から得られる大規模空間の疎な点群データを対象に，誤差の発生が許容されない状況に対応することを目的とした圧縮方式である．3 次元座標上の位置情報と色の情報を示す属性が対象データとなる．最初に点群の座標を圧縮し，次にその座標における属性を圧縮する．

この点群の座標における圧縮であるが，G-PCC において座標の圧縮には 2 通りの手法が存在する．1 つ目の手法である Octree geometry coding は 3 次元空間上の立方体を再帰的に八分木で分割し，分割された小立方体内部の点の有無を参照することにより各点の位置情報を圧縮する．2 つ目の手法である Trisoup geometry coding は物体表面を構成する点群を三角形メッシュの集合として表現し，オブジェクトの表面を近似することで点群を圧縮する．

点群の属性の圧縮は Regional Adaptive Hierarchical Transform（RAHT）と Predicting Transform，Lifting Transform の 3 種類の手法が存在する．RAHT は八分木で分割した空間を空間全体から再帰的にたどり，八分木の下位のレベルにあるノードの色を基に次のレベルにあるノードの色を予測することで圧縮する．Predicting Transform では最初に与えられた点群を粗い粒度から細かい粒度へ便宜的に分類し，細かい粒度の属性は粗い粒度の属性から平均値か近傍値という予測方法を適用することで細かい粒度から粗い粒度の順に属性を圧縮する [8]．Lifting Transform は Predicting Transform を基にしているが細かい粒度から粗い粒度に圧縮する中で，近傍点との距離などを変数に粒度ごとの属性値を更新する手順が追加されている．

2.4 提案手法の貢献

本研究においては 2.2.1 章で記載した pgpointcloud を活用し，動的点群のデータを管理するシステムを提案する．G-PCC では色の情報も含めた点群を圧縮していたが，本研究では 3 次元空間上の位置情報に加え時間軸の情報を含めた点群データを管理する．また，動的点群データをデータベース内で効率的に活用する機能として時空間データとしての点群の検索機能や圧縮手法について述べる．

3 動的点群の管理システム

本章では本研究で実装する動的点群データの管理システムについて，その構成の概要と実装方針について述べる．

3.1 システムの実装方針

動的点群の管理システムに必要な機能として，本稿では点群の圧縮機能と検索機能を扱う．

まず圧縮機能だが，管理システムにおいては 1 章でも述べたように LiDAR センサが継続して周辺の点群データを獲得する．

取得したデータをデータベース内で管理する場合，管理する点群データの量は非常に膨大なものとなる．一般に動的点群データのデータ量は大きく，時間軸に対して細かい粒度で点群データが取得されるため，現実的に時系列で取得される点群に対してはデータの圧縮が必要である．この膨大な量のデータを圧縮することでより多くの範囲の点群の処理が可能となり，点群の検索などのデータに対する処理も効率化される．そのため動的点群データの圧縮をシステムの実装における最優先の目標と定め，より多くのデータの管理が可能なシステムを構築する．

次に検索機能であるが提案するシステムにおいては時系列に沿った膨大な量の点群データを管理しなければならない．そのため実際に点群データを活用する際にはある時刻の点群データや一定の時間軸の範囲に基づく特定の地点の点群を抽出することが考えられる．このような場合において全ての点群データを逐一抽出していては効率的な処理を実行できない．そのため点群内の各点の位置だけでなく，時系列情報にも対応可能な点群データの検索機能を提供する．点群の検索機能においてはそれぞれの 3 次元座標上の各座標軸の最大値と最小値，そして時間の範囲を提示することで該当する点群を検出する．

そしてこのような圧縮・検索機能を実現するための実装方針として，本研究では既存 OSS の活用を前提としたシステムを設計する．動的点群は近年その利活用が議論され始めたデータであり，MPEG によってその標準化は進められているが，G-PCC を始めとする具体的なライブラリの整備はまだ進行中である．つまり，圧縮など動的点群を操作するライブラリについては今後様々な展開が予想される．そのためある特定のライブラリに依存した形式でデータベース中に格納するのではなく，なるべく汎用的な形で動的点群を整理および格納し，今後の点群に関するライブラリの動向に対して柔軟に対応可能なアーキテクチャを考える．

3.2 システムの構成

ここでは，提案するシステムにおける大まかなアーキテクチャとデータベース内部における点群データの一連の流れについて述べる．なお，アーキテクチャと点群データの流れは図 1 のようになる．

3.2.1 システムのアーキテクチャ

提案システムの大まかなアーキテクチャについて述べる．まず，動的点群データを格納するデータベースにおいては，DBMS（Data Base Management System）である PostgreSQL を用いてデータを管理する．PostgreSQL は SQL 言語を用いてデータベースの作成や編集，データの読み書きが可能なほか，内部の索引やデータ型などをユーザ定義で実装できる高い拡張性を備えている．提案システムにおけるデータベースでは点群の 3 次元上の座標データと，点群を観測した時間軸データを格納する．このとき未加工点群に対して，データベースまでの格納処理を一意に把握するための ID を個別に定め，観測したデバイスの情報と共にデータベース内に格納する．データベース内では点群の座標データの集合を PcPatch に集約することで内部のテーブルに格納するため pgpointcloud を拡張モジュールとして導入する．そしてデータベース内への点群データの入力またはデータベース中のデータの出力ではオープンソースのライブラリである PDAL（Point Data Abstraction Library）を内部のライブラリとして使用する．PDAL を使用することで点群データの変換をはじめとした様々な処理が実行可能となる．提案システムではデータの入出力でのインタフェースとなり LAS や PLY といった点群データの形式を PcPatch に変換して入力，または PcPatch を別の形式に変換して出力する．API と変換された点群データでやり取りを交わしユーザに活用可能な形で提供する．

3.2.2 データベース中の点群の流れ

データベース内においては以下の図 1 のような流れのもとで，点群を取り扱う．図 1 のデータベース中のテーブルと，点群データの入力から出力における一連の流れについて述べる．

a） データベース中のテーブル

ここでは図 1 に記されたデータベース中のテーブルについて述べる．まず，テーブル dirty_points であるがこちらはパイプラインより得られた点群データをデータ型 PcPatch に変換して格納するためのテーブルである．データベース内では点群を効率的に活用するため，新たにデータベースに加わった点群を一定の範囲ごとに区分けするが，その整形の過程では dirty_points の未加工点群データを他のテーブルの値と照らし合わせ点群データを整形する．次にテーブル store_histories はパイプラインより得られた点群の時間軸データを格納する．点群の整形ではこの時間軸データを dirty_points と同じように参照する．グリッド範囲の座標区分を示す spatial_grid は点群の整形において，点群をどの範囲で区分けするか 3 次元座標上の最大値と最小値をそれぞれの次元でデータを格納することで定める．pc_snapshots はデバイスが点群データを観測する地点において事前に獲得した点群データをスナップショット点群として格納している．未加工点群の整形や圧縮においてはこのスナップショット点群を使用することでより効率的な処理を実行できる．最後にテーブル cleaned_points と pc_entities であるが pc_entities には整形後の点群データが格納され cleaned_points には pc_entities 内で各タプルを識別する主キーである pc_id を参照用の情報として格納する．なお pc_entities と cleaned_points に格納されている情報はほぼ同様であるが，cleaned_points は点群を間接参照するためのテーブルであり，圧縮機能を実現するための役割を果たす．

b） 点群を取り扱う一連の流れ

図 1 で提示されたテーブルにおける dirty_points への挿入から pc_entities までの一連の流れを述べる．最初にパイプラインを通じてローダから挿入された点群データをテーブル dirty_points に格納する．この時，時間軸データとして格納時の時刻もテーブル store_histories に挿入する．そして事前に格納されていた spatial_grid のグリッド範囲の情報と pc_snapshots のスナップショットとなる点群データをデータの区分けと圧縮に使用することで pc_entities と cleaned_points に整形後のデータを格納する．データベース内の点群データを活用するときは pc_entities と cleaned_points よりデータを参照しパイプラインを通じてユーザに出力される．

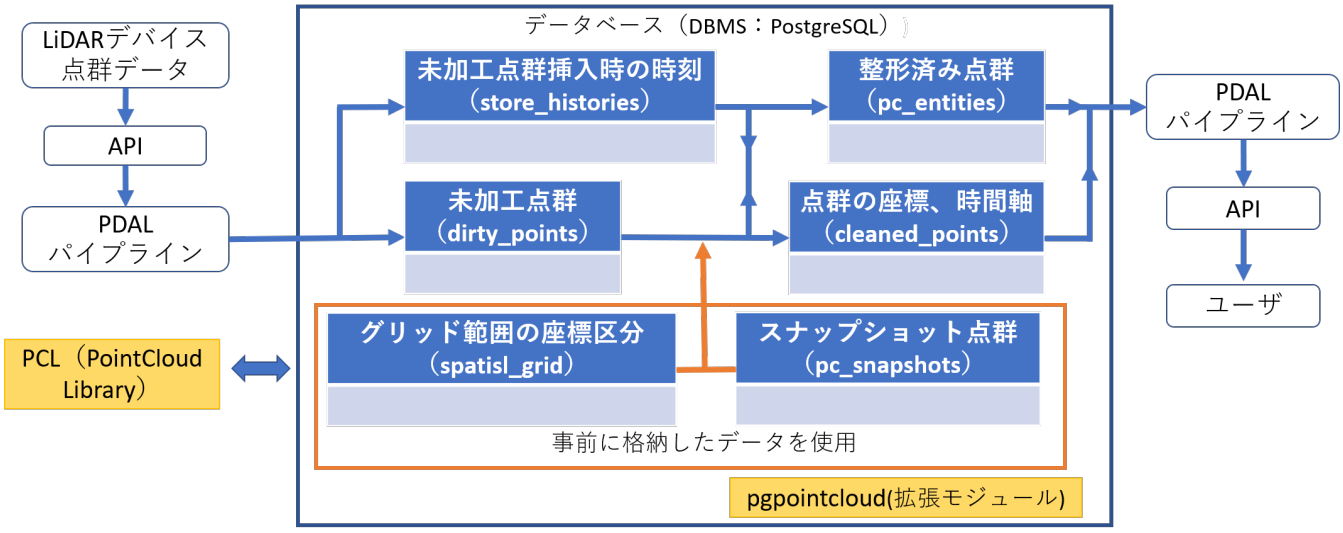

図 1　システムのアーキテクチャとデータの流れ

4　管理システムの実装

本章では LiDAR で得られた動的点群のデータをデータベース上で扱うにあたり，点群データの格納と圧縮，検索の機能を持つシステムについて提案する．このシステムではまず LiDAR デバイスからローダを通じてデータベースに未加工の動的点群データを挿入し，3 次元座標上の範囲ごとにデータを分割することでデータベースに点群データを格納する．点群データはテーブル pc_entities 内にそれぞれ points として PcPatch のデータが 3 次元座標上の範囲に従い分割して格納されている．この範囲に基づき，共に格納された挿入時の時間軸データと併せて点群を効率的に検索する．点群データの圧縮は地形をモニタリングするための点群とは別に獲得したスナップショットの点群を活用する．同じ地点のスナップショット点群との差分を抽出し，近似した点群を削除することでデータが圧縮される．

4.1　点群データの格納

点群データの格納においては関連研究で述べた pgpointcloud を使用する．LiDAR デバイスで得られた点群データの集合をデータ型 PcPatch としてデータベースに挿入し，検索機能および圧縮機能を効率的に行うための前処理も実行する．以下では，動的点群の挿入を未加工点群の挿入とその整形の 2 段階に分けて述べる．なお，点群の挿入は圧縮処理の基準となるスナップショット点群の挿入とモニタリング時の挿入との 2 つに分かれるが，登録したスナップショットの更新処理を除き基本的な処理の流れは同一であるためまとめて説明する．

4.1.1　未加工点群の挿入

まず，LiDAR から取得した動的点群データをデータベースへそのまま挿入する．これは LiDAR によるデータの取り込みと後述する整形処理を分離するためであり，取り込みと整形を非同期に行うことで処理の並列度を高める．

以下点群挿入の手順を述べる．まずデバイスから観測した点群データ，使用した LiDAR デバイスの UUID，および各デバイスで管理するトランザクション ID を送信する．UUID とは識別子を生成する標準規格であり同一のアルゴリズムを使用しても既知の世界上の他の誰かが同一識別子を生成する可能性がほとんどないように選択されたアルゴリズムで作成される．そのため，未加工点群に対して生成アルゴリズムにより重複の可能性が極めて低い一意な ID を高速に割り当てられる．ここでは UUID と 2.2.1 章で述べた格納処理を一意に識別するトランザクション ID を点群挿入処理の管理，各 LiDAR デバイスの観測範囲の取得や処理失敗時の再処理の補助などに使用する．次に OSS のライブラリである PDAL（Point Data Abstraction Library）が提供するパイプラインローダ [9] を通じて図 2 のようにデータ型を PcPatch に変換しつつデータベースへ取り込む．PDAL には点群のフィルタリングや結合などの機能が実装されており，それぞれの細かな処理の集合をパイプラインとして組み上げることで，点群データへの複雑な処理を一連の操作として動的に作成できる．図 2 では特定の形式の点群データを読み込む Reader の処理と，読み込んだデータをデータベースやテーブルを指定することで PcPatch として出力する Writer の処理で構成されたパイプラインを実行している．

また，提案するシステムで実際に点群を取り込む際には PDAL パイプラインのオプションである SQL の事前・事後実行を活用する．この事前・事後実行を施すことでデータの挿入前と挿入後において，テーブルの作成をはじめとした新たな処理を命令できる．点群データを取り込むにあたり，まず SQL の事前実行でデータベースに一時テーブルを作成し，PDAL パイプラインでは点群データのみをこの一時テーブルに書き込む．そして SQL の事後実行により，一時テーブル上の点群データに管理用のデバイス UUID およびトランザクション ID を付与しつつ未加工点群の格納に使用するテーブル dirty_points へと転記する．同時に点群の書込み時刻をテーブル store_histories に記録し，この書込み時刻を点群の時間情報として扱う．このとき，格納する点群に色情報といった座標以外の情報も含まれていた場合，3 次元座標の位置情報と共に PcPatch 内部に含まれる情報としてまとめて dirty_point に格納する．

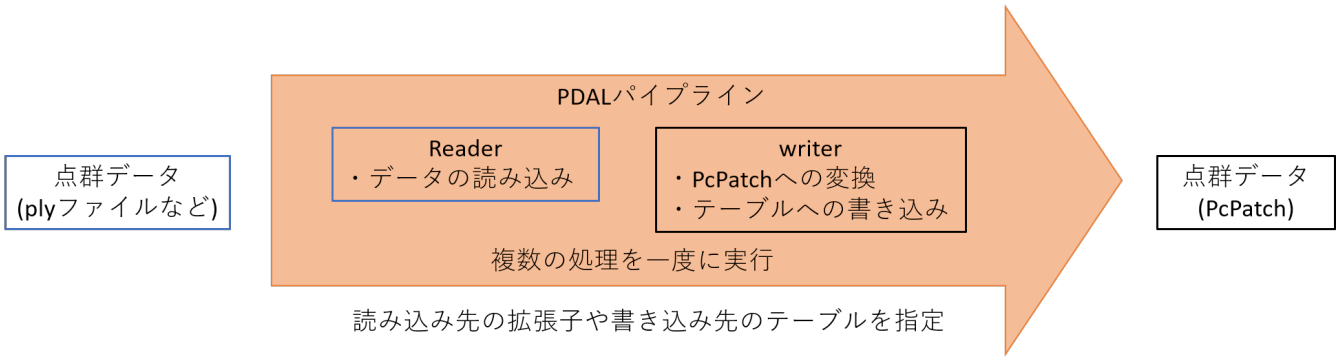

図 2　点群データを PcPatch に変換するパイプライン

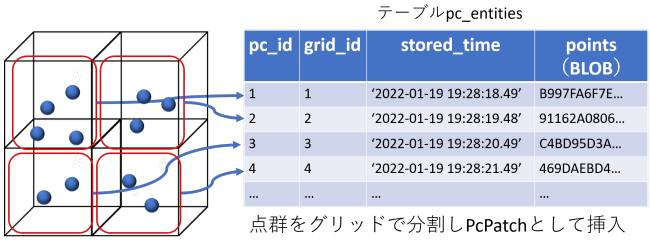

図 3　グリッドによる未加工点群の分割

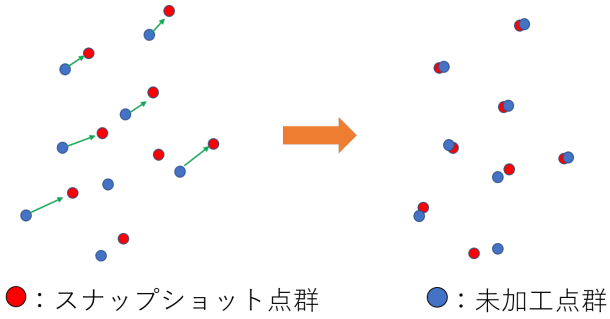

図 4　未加工点群の位置合わせ

4.1.2　点群の整形

テーブル dirty_points に未加工点群のデータを挿入した後，点群の効率的な検索や圧縮を補助するために点群を整形する．例えば，点群のデータを検索する上でも dirty_points 内の PcPatch データから抽出すれば膨大な量の点群データを参照しなければならず，検索において膨大な時間がかかってしまう．そのため取得した点群を複数のグループに区分けすることで指定の点群をより短時間で参照することが可能となり処理も容易となる．このグループ分けは 3 次元空間上のそれぞれの座標軸を指定の範囲で区切り，その境界線で構成されたグリッドにしたがって図 3 のように分ける．

整形処理の大まかな流れとしては最初に未加工点群とスナップショット点群との位置合わせを行い，LiDAR デバイスでの撮影時に発生しうる微妙な位置ずれを修正する．その後，点群を 3 次元空間でグリッド上に分割し，点群の実体を表す PcPatch のデータと検索・圧縮用の参照 ID をそれぞれ対応するテーブルへ挿入する．以下，位置合わせとグリッド分割の処理についてそれぞれ具体的に説明する．

最初に dirty_points の未加工点群を区分けするにあたり，点群の取得における微妙なズレを補正する．点群データの獲得にあたっては，デバイスの設置場所の僅かなぐらつきといった原因から点群の取得時に位置ずれを起こす可能性がある．そのままでは観測した物体が動かなかった場合でも，その位置を変えたと判断されてしまう．それにより後述する点群の圧縮処理が上手く機能しない事態も引き起こす．そのため，図 4 のように点群の位置合わせの処理を実行する．位置合わせではまずデバイスの情報を基にそのデバイスが点群を観測したグリッドの範囲を特定する．そこからテーブル pc_snapshots より位置合わせの基準とするスナップショット点群の点群データを取得する．スナップショット点群には事前に獲得しておいた各グリッドごとの点群データを使用する．そして 3 次元空間上の点群処理に活用されるオープンソースのソフトウェアライブラリである PCL の registration 関数を未加工点群とスナップショット点群に対して実行する [10]．pcl は数々の処理モジュールから構成されており，点群データに対してフィルタリングや特徴推定などの処理を実行できる．その中の registration 関数は ICP アルゴリズムと呼ばれる手法を使用し，対象となる各点群間の最近接点との距離の総和を最小化するよう未加工点群に剛体変換の処理を施す．これにより 2 つの 3 次元点群データ間の重なり部分を検出し，取得したスナップショット点群と未加工点群の座標の位置を調整する．

点群の位置合わせを行った後，グリッドの範囲に基づき点群データを格納する．格納までの手順においてはまず，未加工点群のデータで構成された PcPatch に対して内部の点群の値を出力する pgpointcloud の関数 PC_Get を使用し点群の座標の値を取得する．この取得した点の座標と spatial_grid に記されたグリッドの領域を照らし合わせ，それぞれのグリッドにどの点が格納されるかを判別する．格納されるグリッドごとに点の集合を PcPatch を作成する pgpointcloud の関数 PC_MakePatch を使用し，1 つの PcPatch としてまとめる．これをそれぞれのグリッドで実行することで，内部の点群がグリッドごとに定められた PcPatch を取得する．このように獲得した各 PcPatch とあわせてトランザクション ID とデバイスの情報より store_histories に保存された時刻情報を取得し，テーブル pc_entities に格納する．最後に，pc_entities に格納された点群へ対応する pc_id を参照用の情報として取得し，テーブル cleaned_points へと格納する．

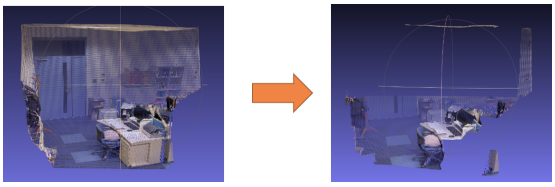

図 5　圧縮後の点群データ

4.2　点群データの圧縮

点群データの圧縮はスナップショット点群を活用する．グリッド内の点群がその時刻のスナップショット点群と十分に類似する場合，その点群を削除することで行われる．削除後の点群データをユーザから要求された場合は，削除した点群の代わりにスナップショット点群を返すことで対応する．これより，データベース内に格納する点群のデータ量を減らし，異なる時刻であっても実際に読み取られる点群が同じになる．また，スナップショット点群が読み出されることによるキャッシュヒット率の向上も期待できる．

具体的な手順としては，まずデバイスの情報と特定の時刻を参照し pc_entities 中に PcPatch 型として格納されている点群データを取得する．次に取得した点群データとグリッド範囲が等しいスナップショット点群をテーブル pc_snapshots より取得し，各点群を結合し 1 つのタプルにまとめる．そして，位置合わせの際と同様に PCL を用いて 2 つの点群の差分を基に検出した類似度を抽出する．点群の座標間の距離を 2 つの点群の間で逐一比較し，他方の点群と近接している点がないか，最終的には点群全体が近似しているかを確認する．各グリッドごとにスナップショット点群と比較し，近似していると判断されたグリッドでは図 5 のようにスナップショット点群のみを残し比較した点群を削除する．また，削除と同時に cleaned_points 内の対応する参照用 ID（pc_id）をスナップショット点群のものに更新する．この一連の処理を指定したデバイスが観測したグリッドごとにそれぞれ実行することで圧縮処理を終える．

4.3　時間軸に基づく点群データの検索

点群の検索ではユーザからデータベースを使用する API に対して，検索する時間と 3 次元空間上の座標の範囲を提示することで一連の処理を実行する．命令を受け取ったデータベースはテーブル spatial_grid における 3 次元空間のグリッドの範囲と，cleaned_points に格納された時間軸の情報を参照する．グリッドの探索においては PostgreSQL の拡張モジュールである PostGIS の関数と，spatial_grid に格納された PostGIS のジオメトリ変数である geom を使用することで図 6 のように提示した座標範囲が含まれるグリッドを検出する [11]．これより検索条件に該当する pc_entities 内の PcPatch を抽出する．提案するシステムにおけるデータベース上の検索機能は PcPatch の抽出に留まり，点群データとしての表示においては API のユースケースに合わせて新たに処理を施す．

5　提案システムの実験

提案システムの性能について実験した．一定の地点で観測した点群データを提案システムに導入し，PcPatch としての点群の格納，dirty_points に格納してからの点群の整形，近似した点群の削除による圧縮にかかる時間をそれぞれ計測した．圧縮については圧縮処理の前後におけるデータ量の変化を測定した．また圧縮後の点群データに対して特定の範囲を定めて検索処理を実行し，検索結果の点群データを ply ファイルとして出力した．以下の項目では実験を執り行う上での実験環境と，実験手法，実験結果，結果から得られた考察についてそれぞれ述べる．

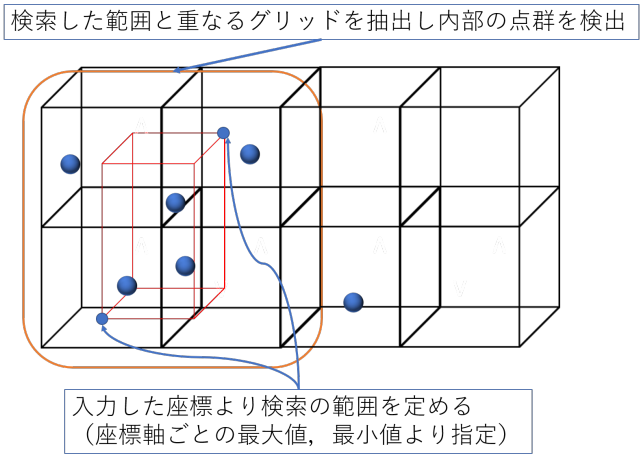

図 6　グリッドの参照による点群データの検索

表 1　実験におけるサーバー環境

Item	Value
CPU	AMD EPYC 7702P 64-Core Processor
RAM	DIMM DDR4 Synchronous Registered （Buffered）3200 MHz （32GB×4）
OS	Ubuntu 20.04.3 LTS
Compiler	(Ubuntu 9.3.0-17ubuntu1 20.04) 9.3.0

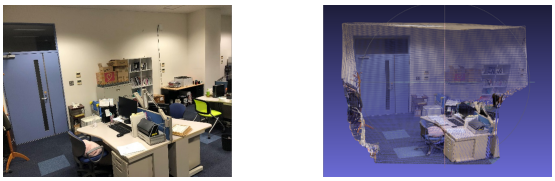

図 7　実験での観測場所と LiDAR で観測した際の点群データ

5.1　実 験 環 境

ここでは実験で使用したサーバの環境とデータセットについて述べる．まずサーバの環境であるが下記の表 1 の環境のもとで実験をおこなった．

次に計測の過程で処理したデータセットだが，ここでは第 3 世代の iPad の LiDAR センサで獲得した点群データを使用した．iPad には LiDAR センサが備え付けられており，これを用いた iPad で使用可能な点群獲得のアプリケーションを作成した．このアプリケーションを用いて一定の地点から約 2 時間 40 分に渡り，毎秒点群を獲得する．LiDAR で計測した場所は図 7 の写真で示された場所となる．この外観を LiDAR で観測し図 7 のような点群を保持する ply ファイルとしてサーバ上に格納した．

秒間隔で観測した点群データはそれぞれ別のファイルとして保存し，ファイル数は合計 9714 となる．この点群データは属性として 3 次元座標上の位置データの他，RGB の色情報のデータも保持している．そしてファイルの容量は 1 つ当たり約 2MB であり合計の容量は約 18.68 GB となった．また，ply ファイルはテキスト形式とバイナリ形式の 2 通りの書式での保存が可能である．獲得した ply ファイルはテキストファイルとして保存しているが，保存の形式をバイナリ形式に変換すると合計の容量は約 12.48 GB となる．これらのファイルを全てデータベースに格納し，点群データに対して整形処理と圧縮処理をそれぞれ実行した．

5.2 実験手法

この節では実験におけるそれぞれの項目の具体的な計測手法について述べる．

実験ではまずデータベースに新たな点群データを格納し，その作業時間を計測する．計測する作業では指定のディレクトリ内に ply ファイルとして存在する点群データを PcPatch としてテーブル dirty_points へ格納する．このとき未加工点群の挿入では，個々の ply ファイルに対して挿入のためのパイプラインを構築する必要がある．パイプラインでは特定のデータを挿入するにあたり読み込むファイル名を指定しなければならないが，ここではそれぞれの ply ファイルを基にパイプラインを作成するシェルスクリプトを製作した．シェルスクリプト内では繰り返しの操作に for 文を使用し，各操作ではそれぞれディレクトリ内のファイルを 1 つずつ参照する．パイプラインのフォーマットにファイル名を逐一当てはめることで，それぞれのファイルを PcPatch として書き込むパイプラインを構築し，実行する．これによりシェルスクリプトを使用することでディレクトリ内の点群データを一度に挿入することが可能となる．実験においてはこのシェルスクリプトの実行時間を計測することで，処理全体にかかった合計時間を測定した．なお，実行時間の記録はシェルスクリプトの処理と同時に最終的な時間をテキストファイル runtime.txt に書き込むようコマンドラインを構築することで記録した．

次に，挿入された未加工点群に対して，グリッドと時間軸データを基に pc_entities や cleaned_points にデータを整形する時間を計測する．まず，dirty_points に格納した中で最も過去の時間となる点群に対して整形処理を行う．この点群は位置合わせや圧縮で使用するスナップショット点群として pc_snapshot に格納する．そして dirty_points に残っている全ての未加工点群データに対してストアドプロシージャLoop_InsertCleanedPoints を使用する．この Loop_InsertCleanedPoints は dirty_poits 内の点群データ全てに整形処理を実行するプロシージャである．今回の実験では Loop_InsertCleanedPoints による整形処理の時間を計ることで実行時間を記録する．なお合計時間の記録は，PostgreSQL 内において問合せの実行計画を表示する EXPLAIN コマンドのパラメータである timing を使用する．この timing により PostgreSQL で実行した問合せの時間を取得できるため，このプロシージャの実行の際に timing を用いて整形処理の実行時間を把握する．

表 2　点群データの検索範囲

	min	max
x	-3000	-1000
y	1000	5000
z	-9000	-4000
stored_time	2022-01-22 14:13:24	2022-01-22 14:13:25

表 3　各処理ごとの処理時間

処理名	処理時間（s）
点群データの格納	2,396
点群データの整形	14,073
点群データの圧縮	5,673

表 4　点群の圧縮による pc_entities 内の PcPatch のメモリ容量の変化

	メモリ容量（MByte）
圧縮前	4,972
圧縮後	2,376

そして整形済み点群の圧縮処理だが実行時間については整形処理と同様に，pc_entities の点群全体に対して実行した圧縮処理のストアドプロシージャLoop_RemoveSimilarPointClouds の実行時間を timing で測り，取得した．このとき，圧縮によるデータ量の変化を測定するが，そこでは pgpointcloud の関数 PC_MemSize を使用する．PC_MemSize を使用することで各 PcPatch に対してデータベース内のメモリ容量を検出できる．圧縮の前後で pc_entities の PcPatch を格納した points の欄に対して PC_MemSize の合計値を求めバイトの単位を計算し直すことでデータ量の変化を記録した．

最後に点群データの検索処理であるが表 2 の範囲をもとに，データベース内の点群データを検索しパイプラインを用いて ply ファイルとして出力した．時間の計測についてはデータベース内の SQL 文による検索時間と，検索後のデータを点群データと出力するまでにかかった時間をそれぞれ想定した．pc_entities 内の値の出力時間を timing で計測し，ply ファイルで出力する時間については点群データの格納と同様に runtime.txt に書き込むコマンドラインを構築することで検証した．

5.3 実験結果

ここでは実験より得られた結果について述べる．まず格納，整形，圧縮処理のそれぞれの実行時間について表 3 に示す．このとき，圧縮の前後における使用データ量の変化は表 4 のようになり，点群データの検索結果は図 8 のような点群が出力された．点群のデータベース内での検索時間は 1.580 ミリ秒であり，パイプラインでの出力には 0.13 秒の時間がかかった．

5.4 考　　察

まず，動的点群を挿入する際のスループットについて述べる．表 3 に示すとおり，提案システムは約 18.68 GB の動的点群データをおよそ 2396 秒でデータベースへ格納しており，そのスルー

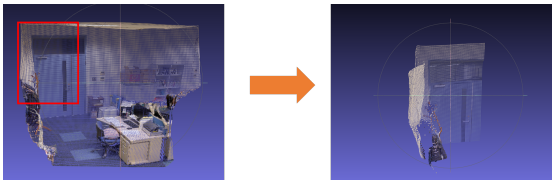

図 8　点群データの検索結果

プットは約 8.0 MB である．今回使用したファイルは 1 ファイルあたり約 2MB であるため，単体の LiDAR デバイスを用いたモニタリングであれば 3 から 4fps でのモニタリングが可能となる．これは例えば人の移動などを観測する程度であれば十分な粒度である．一方で今回のシステムは全ての処理をシングルスレッドで逐次的に行っているため，ストレージ I/O が許す限り，モニタリング対象のデバイスの追加は可能だと考えられる．また，サーバと LiDAR デバイスとをつなぐネットワークは現実的に考えれば無線通信となるため，点群の挿入に関してはネットワークのスループットが観測対象の上限数を決める上でのボトルネックとなると考える．

次に，点群を整形および圧縮する際のスループットについて述べる．表 3 に示すとおり，点群の挿入に比べ整形および圧縮には合計でおよそ 8.2 倍の処理時間が必要となっている．これは，整形もしくは圧縮の各ステップで ICP アルゴリズムにより点群中の各点をマッチングする必要があり，これがボトルネックになっていると考えられる．一方で，点群の整形および圧縮処理はサーバ内で完結する処理であり，処理の並列化が容易である．つまり，異なる時刻ないし異なるデバイスから取得された動的点群は並列で整形および圧縮処理が可能であるため，例えば今回の実験設定であれば理想的にはおよそ 8 スレッドでの並列処理を行えば点群の挿入と同程度のスループットが達成できると考えられる．また，整形処理と圧縮処理はそれぞれ別のトランザクションで扱うため，非同期処理による処理のスケジューリングも柔軟に行える．以上より，提案システムによる点群の整形および圧縮処理は実現可能であると考える．

最後に，点群の圧縮効率について述べる．表 4 に示すとおり，圧縮により点群の総量はおよそ 5 割程度まで削減されている．期待ほど小さくないこの減少率は，今回の実験で設定したグリッドサイズの大きさが原因として挙げられる．グリッドサイズが大きい場合，グリッド中の一部の変化が検知された際にその専有面積が小さくともグリッド全体がデータベース中に残されるため，圧縮効率が下がってしまったと考えられる．また，差分を検出するための指標や差分の有無を判定するしきい値など，より効率的な圧縮を達成するための改善が必要だと考える．

6　おわりに

6.1　まとめ

本稿では LiDAR センサから得られる点群データとして動的点群データを想定し，その管理手法について検討した．関連研究として pgpointcloud やそれを用いたデータベースでの管理，ファイルに基づく管理方法を述べた．そして pgpointcloud のデータ型 PcPatch による時間軸データを使用した点群の検索や圧縮の機能を持つシステムを提案し点群の格納，整形，圧縮や検索において処理時間やデータの変化量の実験結果を提示した．

6.2　今後の課題

点群データの圧縮において，どのようにして観測した物体に動きがみられる点群データを保持するかが今後の課題として挙げられる．5 章では点群の圧縮処理により，テーブル pc_entities のデータ量の減少が確認できた．しかし座標に動きがみられる部分を圧縮の過程で削除されずにいるかや，圧縮した全てのデータを時間軸や座標を指定した際に元の点群の位置や並びを保持したまま出力できるかを検証しなければ点群の管理には使用できない．このような動的点群全体における圧縮の解析が今後の研究で必要となる．

また 5.4 節でデバイスでの稼働について述べたが，実際にシステムとして使用するのならばデバイスの追加に対して問題なく処理できるかを確認する必要があり，複数デバイスによるシステムの稼働の実装も課題となる．

謝　　辞

本研究は JSPS 科研費（16H01722，20K19804, 21H03555）の助成，および国立研究開発法人新エネルギー・産業技術総合開発機構（NEDO）の委託業務（JPNP16007）の結果得られたものである．

文　　献

[1] 伊藤 敏夫, 自動運転のための LiDAR 技術の原理と活用法. 科学情報出版, 2020.

[2] D. Graziosi, O. Nakagami, S. Kuma, A. Zaghetto, T. Suzuki, and A. Tabatabai, “An overview of ongoing point cloud compression standardization activities: Video-based (V-PCC) and geometry-based (G-PCC),” *APSIPA Transactions on Signal and Information Processing*, vol. 9, no. 1, 2020.

[3] S. Psomadaki, “Using a database for dynamic point cloud data management.” Graduation Plan, Delft University of Technology,2016.

[4] F. Alvanaki, R. Goncalves, M. Ivanova, M. Kersten, and K. Kyzirakos, “GIS navigation boosted by column stores,” *PVLDB*, vol. 8, pp. 1956–1959, 2015.

[5] R. Cura, J. Perret, and N. Paparoditis, “Point cloud server (PCS) : Point clouds in-base management and processing,” *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. II-3/W5, pp. 531–539, 2015.

[6] “GitHub - pgpointcloud/pointcloud.” `https://github.com/pgpointcloud/pointcloud`.

[7] R. Cura, J. Perret, and N. Paparoditis, “A scalable and multi-purpose point cloud server (PCS) for easier and faster point cloud data management and processing,” *ISPRS Journal of Photogrammetry and Remote Sensing*, vol. 127, 2016.

[8] 田川 憲男, 田良島 周平, 吉田 周平, 河村 圭, 多田 昌裕, 新井啓之, メディア工学の研究動向. 映像情報メディア学会, 2020.

[9] “GitHub - pdal/pdal.” `https://github.com/PDAL/PDAL`.

[10] “GitHub - pointcloudlibrary/pcl.” `https://github.com/PointCloudLibrary/pcl`.

[11] “PostGIS — spatial and geographic objects for postgresql.” `https://postgis.net/`.