

Topic XX 3D都市モデルを使って全国で使えるシステムを作る

地理情報はさまざまな用途に使うことができます。しかし、大きな範囲を扱おうとすると現代のコンピューターでも簡単には扱うことができないほどの巨大なデータになります。そのため、地理情報を適切なアルゴリズムやデータ構造で扱うための多くの研究開発が行われてきました。PLATEAUの3D都市モデルは、三次元の詳細なデータになったことで、さらに巨大なデータの扱いに気を付けなければならなくなっています。さらに、それに加えて三次元コンピュータグラフィックス特有の最適化なども考慮しなければならなくなっています。そして、現代ではクラウドなどのオンラインリソースを上手に使ったオンラインのシステムをどのように構築するかという、サーバーやネットワーク特性まで考慮しなければならないケースが増えています。本トピックでは、これらの一般的な考え方を解説し、いくつか筆者がこれまでに作った事例を取り上げつつ、読者の皆さんのが課題解決のヒントになるように説明します。

巨大な地理情報の扱い方

まず、地理情報がどのくらい巨大なデータになるかを、実際に調べてみます。

たとえば、地球表面を1m²に区切った画像データを作るとしましょう。1ピクセルが1m²になる地球表面の画像データという想定です。衛星写真を思い浮かべるとよいでしょう。

地球の表面積はおおよそ、 $4 \times \pi \times 6,371 \text{ km}^2 = 510,049,428,806,000 \text{ m}^2$ になります。1ピクセルを3バイト(=RGB各色1バイト)として換算すると、 $510,049,428,806,000 \times 3 = 1,530,148,286,418,000 \text{ バイト} \approx 1.5 \text{ ペタバイト}$ のデータサイズになります。これが月ごと週ごとなどで蓄積されることを考えるとなかなかデータサイズです。

一方で、[オープンストリートマップ](#)というプロジェクトが、オープンな汎用地図のベクターデータを作成しています。このオープンストリートマップの[全球データ](#)が、現時点では圧縮されたものでおよそ70GBあります。ちなみに、PLATEAUの東京都のデータは圧縮ファイルで5.5GBです。PLATEAUの情報量で全球データを作ったとすると相当大きなデータになりそうです。ベクターデータの方が桁が少ないとといえば、手元のパソコンで処理するにはちょっと戻込みするレベルの大きさのデータです。そのため、さまざまなアルゴリズムやデータ構造の工夫で、高速に扱えるようにしていかなければ実用的なシステムを作れません。本章では、このような巨大な地理情報を扱うときに有効ないくつかの考え方を説明します。PLATEAUの地理情報を使った全国規模のシステムをつくる際のヒントになれば幸いです。

局所化をうまくつかう

巨大なデータを扱う時の基本は、データの局在性をうまく使い、そもそも一度に処理するデータの量を減らすことです。地理情報データの場合、一度に同時に全データを処理しなければならないことはまれです。注目しているデータ(たとえば今自分がいる場所の周辺)が、地球の裏側のデータを参照しないと処理できないことはほとんどないので、おおよそ注目点の周囲の限定的な範囲のデータを参照するだけで十分なケースがほとんどです。

このようなケースでは、ダウンロードしたりメモリに展開しなければならないデータ量は周辺の狭い範囲のデータだけとなるので、処理量、メモリ使用量などが全体を処理するのに比べて圧倒的に少なく済み、現実的に処理できます。

これは、データ処理だけでなく、表示などでも有効な考え方です。

二次元の地図を表示することを考えた場合、表示している枠の範囲外のデータは基本必要ありません。また、ゲームエンジンでもFarClipなどのレンダリングする最大距離が決められており、それ以遠のオブジェクトについては描画されません。であれば、全データを用意せずにレンダリングに必要な範囲のデータを用意するだけで問題ないということになります。

このように、処理しなければならないデータの範囲を狭めることは、現実に動作する仕組みを作る際に必須の思考と言えます。しかし、全体のデータから必要な範囲のデータを検索・抽出するのに時間がかかってしまうと本末転倒になります。

分割をうまくつかう

そこで次に考慮するのが、データの分割です。都度全データから検索・抽出するのではなく、あらかじめデータ全体を適切な範囲に分割しておいて、分割単位ごとに処理する考え方です。

もっとも簡単な分割はグリッドです。全データ範囲を格子状に区切り任意の座標からその所属する升目を計算できるようにする仕組みです。簡単な例を示します。

例として100km×100kmの地域を対象とするとします。これを1km四方のグリッドで区切り、処理単位とします。座標系は図の通りです。

TODO 図

ここで、任意の座標から所属するグリッドを求めるプログラムを書くと以下のようになります。

```
def grid(int x, int y):
    return ((int)(x/100),(int)(y/100))
```

TODO プログラム精査 Pythonでいい？C#とかにしようか？

また、この仕組みだと境界条件さえきちんと考慮すれば隣接グリッドを求めるのも簡単です。

この例では単純な平面上の分割を取り上げましたが、実際には地球が球体であることの考慮や基本とする座標系をどうするかなど、考え方や用途によってさまざまな分割が考えられます。矩形ではなく、六角形を用いた分割なども提案されています。

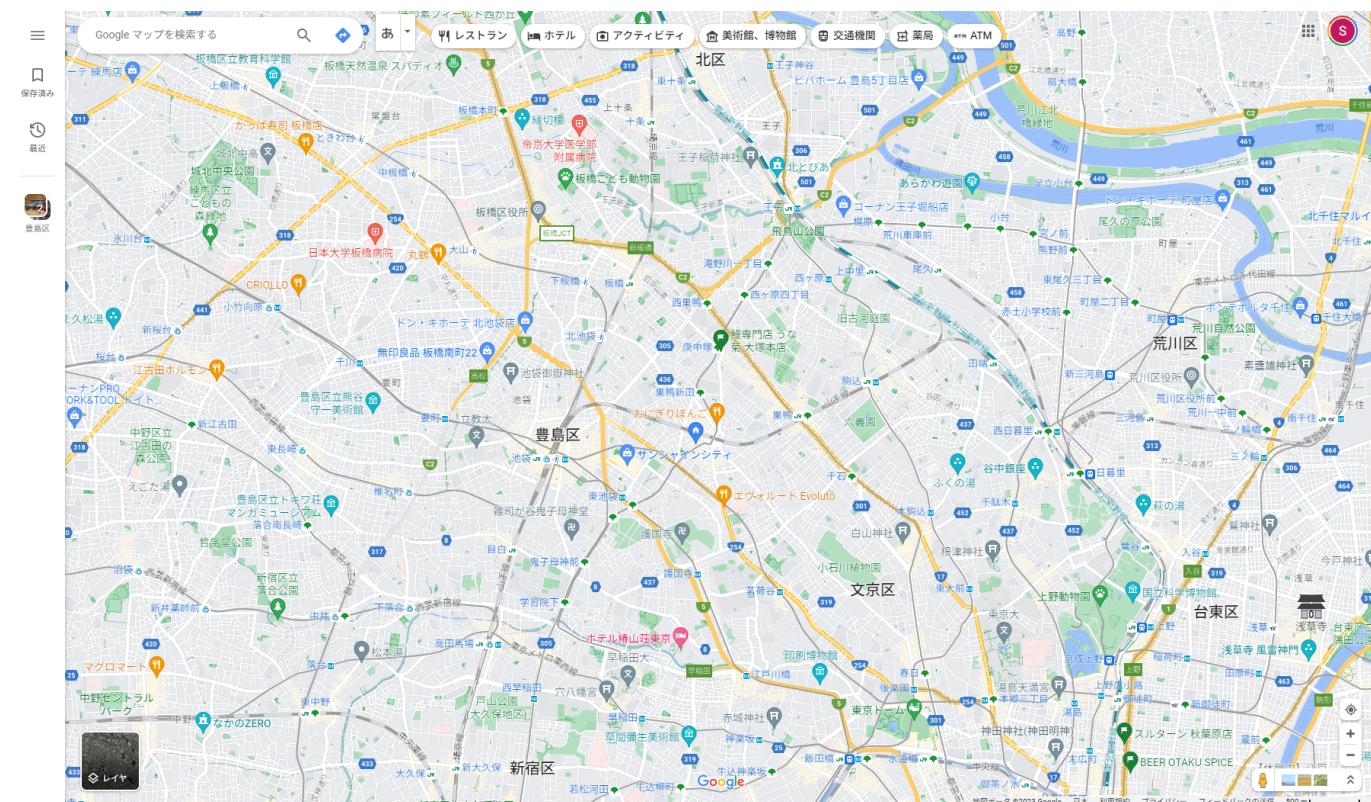
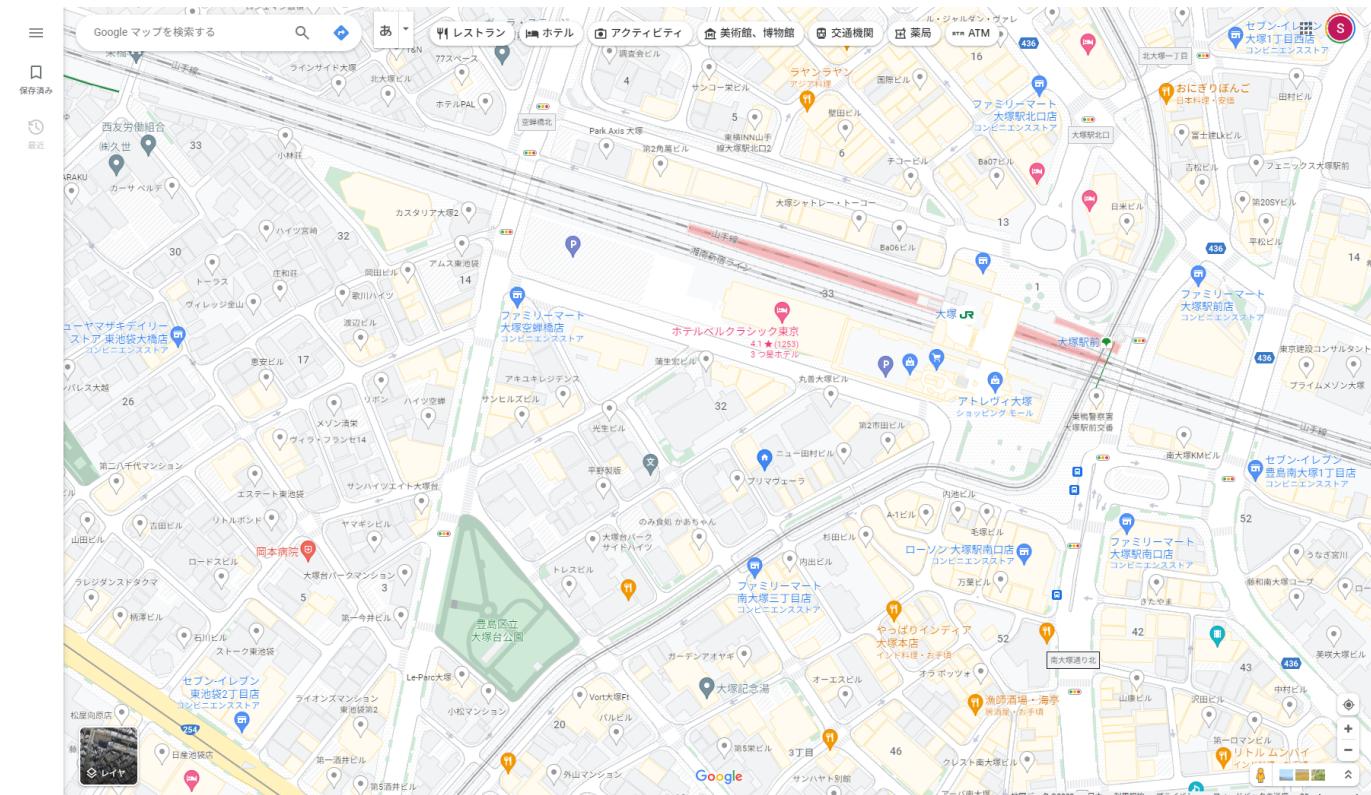
階層化をうまくつかう

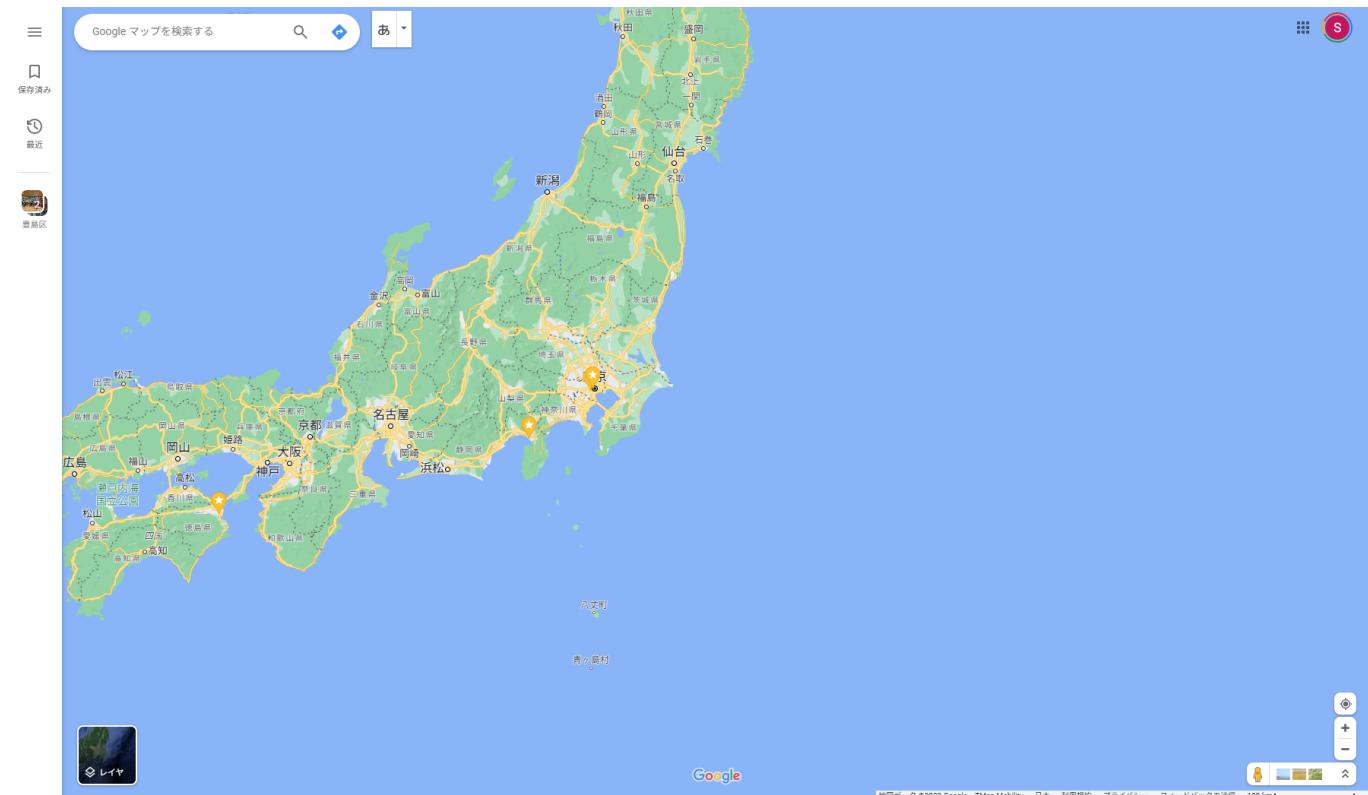
しかし、単純な分割では難しいユースケースがあります。地図がズームする場合はどうしたらよいでしょうか。

ズームするということはどんどん縮小（離れた視点に）していくと、結局広範囲を扱わなければならなくなり、結果大量のグリッドを処理しなければならなくなり、前節のように単純に分割しただけでは破綻します。

ここで使われるのが、階層化の考え方です。

Google Mapsなどで、拡大縮小していると地図の雰囲気が切り替わるのをご存じでしょうか。





TODO GoogleMapsじゃなくて地理院地図の方がいいか。要ライセンス確認

これは、ズームのレベルごとに異なるデータを表示しているからです。ピラミッドデータ構造というような名前でも呼ばれありますが、データを階層的に管理し、上の階層は下の階層のデータをまとめたデータを持つようなデータ構造になっていて、縮尺に合わせて適切な階層のデータを表示しています。また、各階層のデータはその階層に適した粒度で簡略化したりすることで、どのズームレベルのデータでもクライアントが扱えるデータ量に収まるように調整されています。

このように階層化することで、縮尺を変えた場合でも計算量をあまり変えずに処理や描画ができます。また、データ構造を工夫することでデータが存在しない場所のデータ量を削減できます。

TODO 図 データがないところを高い階層で止めてデータ量を節約する

Web地図でよく使われるXYZタイルは、Webメルカトル座標系で表した緯度経度を四分木で分割し、分割数に応じたズームレベルで階層化したものです。

また、3Dコンピュータグラフィックスにおいても階層化をLODの異なるデータととらえることで、視点に近い部分は詳細モデルを使い、遠い部分で簡略化したモデルを使うなど、有効に使うことができます。

※※※ LODという言葉について ※※※

LOD (Level of Detail) という言葉は近い意味としてPLATEAUのコンテキストとゲームエンジンのコンテキストの両方で使われています。

しかし、両者は明確に違うコンテキストで使われているので注意が必要です。

どちらも詳細度という基本の意味は同じですが、PLATEAUではデータが記述している内容の詳細度を表します。LODレベルが上がるごとにより詳細なデータが追加されていきます。

一方でゲームエンジンのコンテキストでは、見た目をなるべく維持しながら遠くのものは簡略化して表示するという考え方で使われます。LODレベルが上がるごとに、元のモデルから簡略化されたモデルとなります。しかし、PLATEAUのLODと違い、データは簡略化されるだけで要素は増減しません。

LODという言葉が出たときはどちらのコンテキストで使われているか注意して読んでください。

空間分割

ここまで分割と階層化の両方をうまく使って、空間全体を分割していくと処理の効率化ができます。もちろん、用途によって適するもの適さないものがあるので、性質をよく理解して使うことが重要です。

空間分割は、特性や用途によりさまざまな方法があります。考慮すべきポイントとして、データ自身の性質、分布、データ構造の更新の頻度や負荷要件、メモリ使用量、各種検索における計算量などになります。

以下に、よく使われる空間分割を簡単に解説します。

TODO 図を描いた方が分かりやすいか。

Quadtree・Octree

四分木、八分木とも呼ばれるデータ構造で、二次元の場合四分木を三次元の場合八分木を使います。各次元ごとに二分割ずつしながら空間分割するデータ構造で、Web地図で使用されているほか、点群の管理にも使われることがあります。

BSP

最初期のFPSゲームとして有名なDOOMで描画の高速化のために使われたことで有名なデータ構造です。空間内に存在するポリゴンを分割平面として再帰的に二分割して構築する二分木のデータ構造です。

kd-tree

BSPの特殊な形で、座標軸に沿った平面で分割するものです。

R-tree

ジオメトリそのものではなく、包括する矩形で木を構成するデータ構造です。最近傍や内包を矩形を基に探索します。多くの空間インデックスの実装で使われています。

空間インデックス

空間分割を使うと、地物の検索なども効率化できます。地理空間データベースなどでこのために構築されるデータ構造を空間インデックスと呼びます。一般的なリレーションナルデータベースシステムでは、データの検索クエリの高速化のためにインデックスというデータ構造が作られます。よく使われるものに、B-Treeやその変種のデータ構造があり、データベースシステムの利用に必須の仕組みとなっています。しかし、一般的な行と列で構成されるデータベースで使われるインデックスのアルゴリズムは地理情報の検索には向きません。そのため、地理情報検索に向けた空間インデックスが考案され実装されています。

PostgreSQLの地理情報拡張であるPostGISの場合、R-Treeの空間インデックスを使うことができます。

他にも、モートン符号やZ階数曲線と呼ばれる符号化の仕組みもあります。各次元の座標値をビット列として前方から交互に配置しなおしたビット列にすることで、範囲検索や内包の判定が前方一致のみで計算できます。四分木や八分木のデータ構造を含み、また数字として近いビット列が表す場所は近い場所に存在する

ことが保証されるなど、データの局所化の観点から見ても効率の良い空間インデックスとなります。一つの値のみの検索で範囲検索ができるので、Key Value Storeなどでの利用が有利です。

また、モートン符号を文字列化したジオハッシュというインデックスもあります。

空間ID

空間分割や空間インデックスのデータ構造をもとにして計算されたIDをつかった空間IDという考え方があります。空間IDの考え方を使うと、地理的な位置を一意のIDで表しデータと紐付けることができます。

空間IDは、作成するサービスやアプリケーションに適したものにすることが多いですが、これを標準化して地球上の場所を一意に表せる仕組みを作り、地理空間データの相互活用がしやすい環境を作ろうという試みもあります。デジタル庁の[空間IDの取り組み](#)は、政府があつかう地理空間情報を統一的なIDで取り扱う仕組みを検討しています。

また、[What3words](#)は、複雑な住所システムを簡易な3単語だけで表す新しい空間IDベースの住所システムを提案しています。

クラウドの活用

現代ではオンラインサービスにおいてクラウドを有効に活用することが一般的です。巨大な地理情報データを扱う場合、都度データをサーバー側のプログラムで計算して配信すると、処理負荷が大きくなります。そのため、あらかじめ処理してキャッシュしておいたファイル単位で配信することが多くなります。その際に選択肢にあがるのがオブジェクトストレージです。AWSのS3やGCPのCloud Storageなどがあります。また、CDN (Contents Delivery Network) の使用も重要です。近年、これらのクラウドを基盤とした配信システムに最適化したファイル形式も登場してきました。

Cesiumで利用されている3DTilesは、あらかじめ空間分割と階層化したデータをファイルの形でオブジェクトストレージに配置し配信することで、効率よく大規模な地図データを利用可能にしています。

最近では、Cloud Optimizeというキーワードが流行っていて、クラウドのオブジェクトストレージサービス (S3など) の特徴である、HTTP(S)での配信に特化したファイル形式が提案されています。オブジェクトストレージサービスでは、HTTPのレンジリクエストを使い、巨大なファイルの一部だけをリクエストすることができます。そのため、これまで複数ファイルに対して、複数のリクエストを行っていたのを、複数の範囲を指定したリクエスト一つで同じデータを取得できます。これにより、データ管理の容易さ、低コスト、効率の良い通信というメリットがあります。また、CDN(Contents Delivery Network)を有効活用してユーザーに近いところにキャッシュして高速化することも容易です。

こうした方向性のデータ形式として、COG(Cloud Optimized GeoTIFF)、COPC(Cloud Optimized Point Cloud)、PMTilesなどがあります。

データの圧縮

圧縮アルゴリズムは巨大なデータを扱うときに非常に強力な武器となります。公開されている地理情報データは、データの相互利用のために汎用的な読みやすいフォーマットで配布されていることが多く、それらのフォーマットは一般的にはデータ容量が大きくなりがちな冗長なフォーマットになっています。そのため、実際に使うデータをどう圧縮するかが重要になってきます。

たとえば、PLATEAUのCityGMLデータはテキストの非常に冗長なデータとなっているため、かなり大きな容量となっています。意味的に同じデータを、ProtocolBufferなどの事前に構造を定義したバイナリ形式で保

存すると大幅に容量を削減できます。また、データ形式をバイナリ形式にすることで、読み込み時のパースの処理などを高速化などの利点もあります。

他にも、属性データの部分をSQLiteなどのファイルベースのデータベースに変換すると、検索が容易になります。

一方で3Dモデルのジオメトリの圧縮に使えるのが、Dracoという圧縮フォーマットです。テキストをベースとしたOBJや、複雑なFBXと比べると、ファイルサイズと読み込み速度で利点があります。

また、テクスチャをゲームエンジンで扱う場合には、JPEGやPNGなどの一般的な画像圧縮形式ではなく、DXTCやASTCのようなGPU専用のテクスチャ圧縮形式を使った方が効率的です。しかし、GPUごとに扱うことのできる形式が異なっている場合が多いため、配信用の形式とするには課題があります。近年、BasisUという中間形式にすることで複数のテクスチャ圧縮形式にクライアント側で容易に変換可能なフォーマットが提案されており、利用が進むことが期待されます。

どちらにしろ、使う目的に合わせて最適化したデータ形式を使うことで、多くのメリットがあります。PLATEAUを使ったシステムを構築する際に、こうした最適化した専用データ構造を検討することは一考の価値があると思います。

無限PLATEAUの仕組み詳解

「無限PLATEAU」は著者が構築したPLATEAUの3D都市モデルデータのWeb配信システムです。公開されているPLATEAUのデータの中から建築物のCityGMLのデータをもとにして、LOD0の平面形状と高さを抽出し、軽量な3D形式に変換してWeb配信するシステムです。これにより、あらかじめモデルデータをアプリ内に入れておかなくても「PLATEAUのあるところならどこでもランタイムにオンラインでモデルが表示される」 = 「無限にPLATEAUが表示され続ける」ということで、「無限PLATEAU」と名付けています。あらかじめすべてのデータを変換しておくことで、ランタイムでの軽量高速な動作を実現しています。

本項では、この仕組みを簡単に解説します。なお、より詳細な仕組みについては、筆者の所属する株式会社ホロラボの有志で作成した[「ホロらぼん Vol.01」](#)の第1章をご参照ください。

無限PLATEAUのデータ処理と空間分割について

まず無限PLATEAUを作るにあたり採用した空間分割を説明します。無限PLATEAUでは、前項で説明した空間分割の中で一番シンプルなパターンである1レベルのグリッドを採用しました。これは、ARなどで実際にその場にいるときにその場の3D都市モデルを歩行者視点で表示することを目的にしていたため、複数のズームレベルに対応する階層化の必要がなかったことからの選択です。また検索も自分がいるまさにその地点のグリッドを計算できればよいため、範囲の検索など複雑な処理が必要ないため、空間インデックスなども必要としないという判断です。

データ圧縮

無限PLATEAUでは、Googleの開発したDracoという3Dデータ圧縮を使っています。Draco圧縮は点群や3Dモデルのデータを劇的に圧縮できるデータフォーマットです。非可逆圧縮で圧縮率もモデルに依存しますが、モデルによっては1/10以下のサイズに圧縮されます。Draco形式への変換は、Googleが公開している[コマンドラインツール](#)を使います。対応データ形式は、OBJ、STLとPLYです。

無限PLATEAUでは、CityGMLから建物ごとにジオメトリを切り出し、建物ごとの重心に基づいてグリッドに分割しています。その後、CityGMLの座標からグリッドの中心を原点とする座標系に変換したうえでジオメ

トリをOBJファイルに書き出します。それを、前述のDracoのコマンドラインツールを使い、Draco形式に保存して配信用のデータファイルとしています。

URL設計

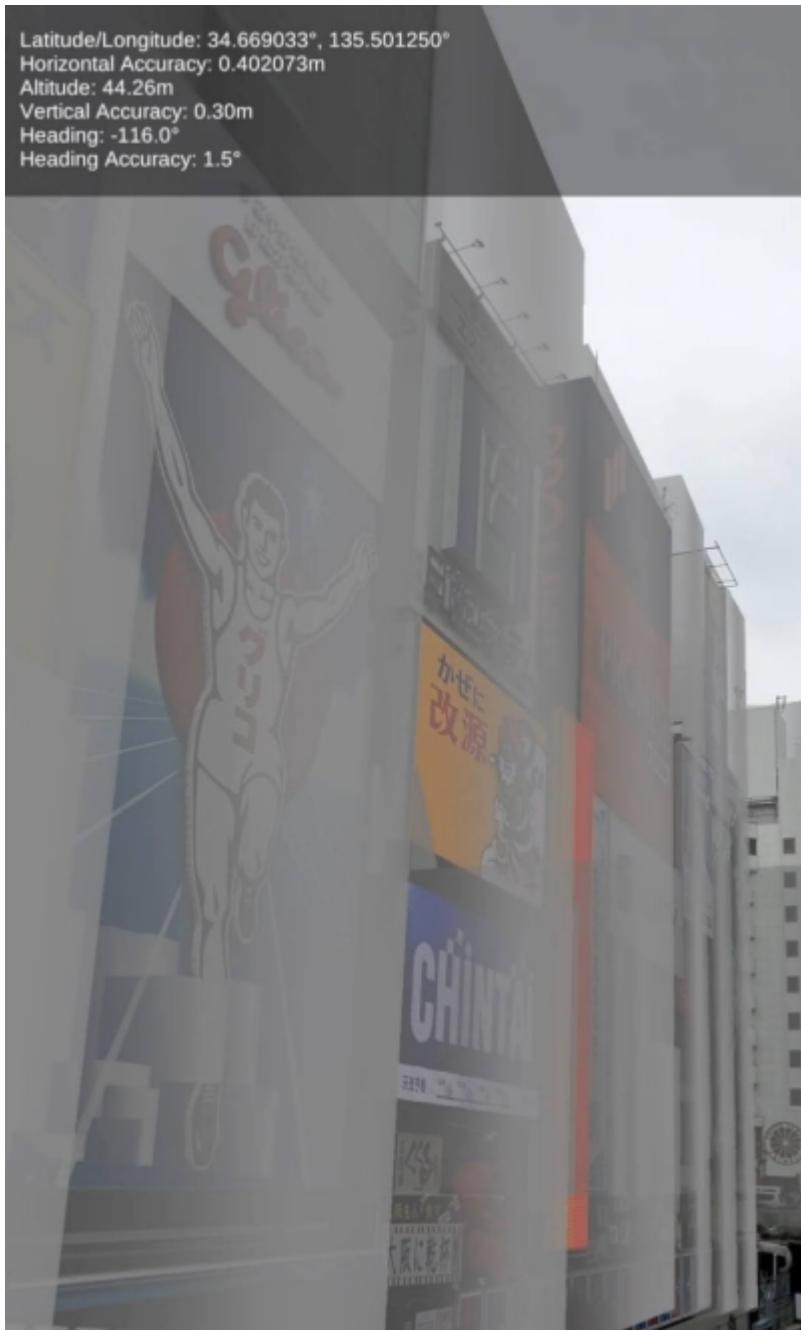
無限PLATEAUのフォルダ構造は、XとYのグリッド番号を上から2桁ずつ区切って交互に並べたものになっています。Webサーバーでファイル配信を運用することを考えると、一つのフォルダにあまり多くのファイルを置くことは性能を劣化させます。そのため、分割して深めのフォルダ階層にすることで一つのフォルダのファイル数をおさえました。また、クライアント側で与えられた地理座標から計算でURLが求まるようにすることで、サーバー側は単純なファイル配信Webサーバーとしますようにしました。

グリッド移動判定の工夫

クライアント側では、ユーザーの移動に伴って、動的に必要なデータをロードしなければなりません。そこでその判定のために、グリッドのXとYの数字を結合した空間コードというひとつの数字を作り、それとのイコール判定でグリッドの移動を判定しました。直前の空間コードと現在の空間コードを比較し、違っていたらロード処理を行います。

無限PLATEAUまとめ

簡単ですが、無限PLATEAUの設計や考え方を紹介しました。

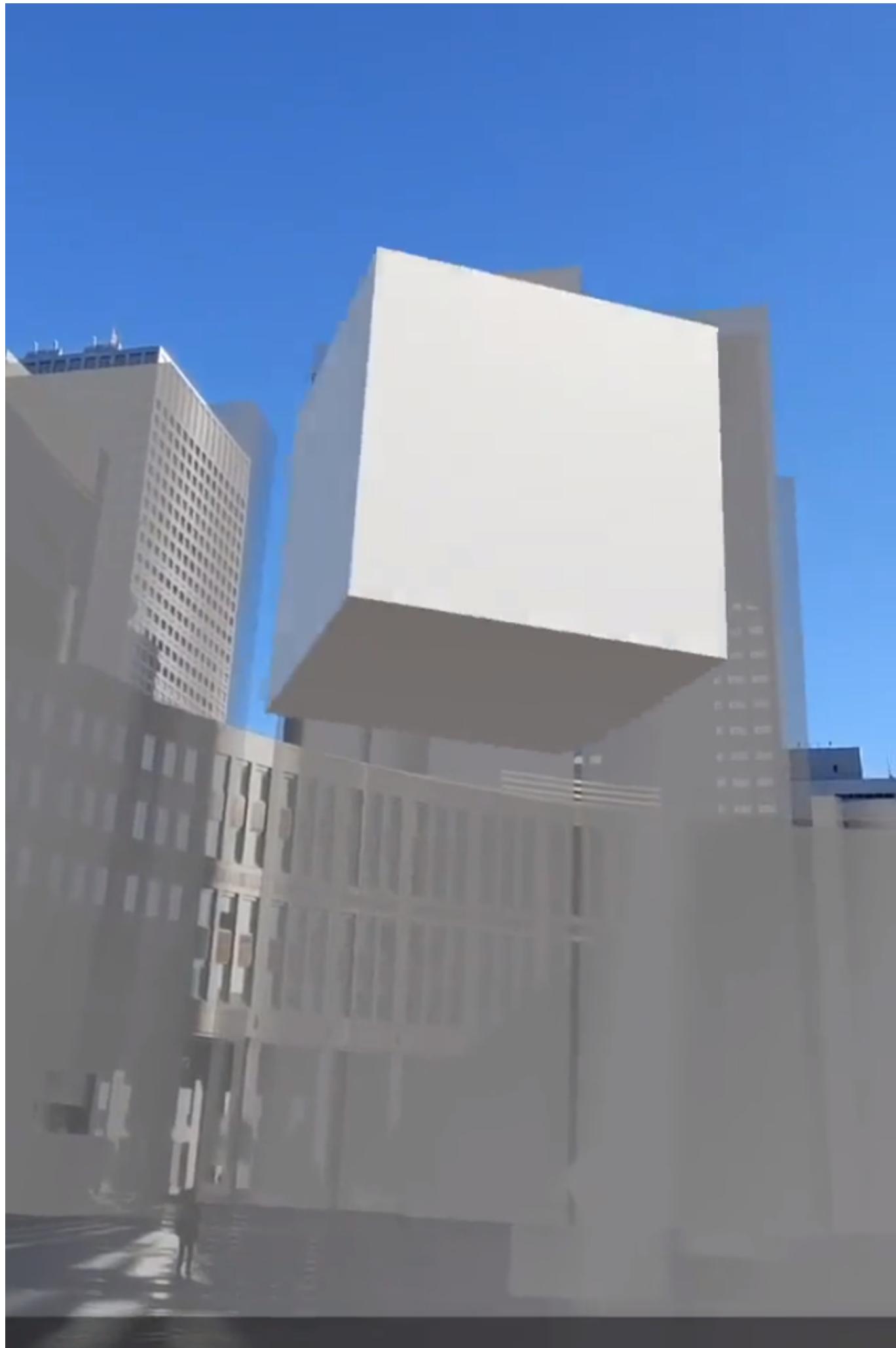


広範囲でスケールする3Dモデルの配信の仕組みとして、参考になれば幸いです。

スケールする地理情報システムでサービスを作る

Orthoverseは、PLATEAUデータを活用したARコンテンツの配信システムです。

Latitude/Longitude: 35.690096°, 139.692541°
Horizontal Accuracy: 0.550052m
Altitude: 73.02m
Vertical Accuracy: 0.36m
Heading: 155.0°
Heading Accuracy: 1.5°



Etheriumのコントラクト（いわゆるNFT）として空間IDとURLを紐付ける仕組みを構築し、場所にURLが紐付くような仕組みになっています。URLはそのままコンテンツへのポインタとなっており、クライアントアプリはGeospatialAPIにより特定した現在位置からその場所のコンテンツをロードし、ARで表示します。

本項では、この場所とコンテンツを紐付ける仕組みについて解説します。

空間IDの設計について

無限PLATEAUと同様に、Orthoverseでも緯度経度を所定のズームレベルのXYZタイル互換のグリッドの番号に変換しています。ズームレベルは20を採用し、日本周辺ではおよそ数10m四方の空間を最小単位としています。

KVSで空間情報を管理する仕組みについて

空間IDにより、IDと場所が一意に紐づくため、シンプルなKeyValueStoreで空間を管理することができます。Orthoverseでは、KVSとしてEtherium上にNFTとして構築したコントラクトに情報を保存しています。NFTのアドレスがそのまま空間IDの格納先になり、アドレスに紐付くデータにURLが入るようになっています。

こうすることで、非常にシンプルに、場所に紐付く情報の管理ができるようになっています。

コンテンツの配信の仕組みについて

URLの先のWebサーバーには、マークアップテキストとしてコンテンツが格納されています。マークアップテキストはMozillaのA-Frameを参考にした独自形式で、3Dコンテンツを表現することができます。

```
<!DOCTYPE html>
<html>
<head>
<meta charset='utf-8'>
<title>Orthoverse Test</title>
<meta name='description' content='Orthoverse Test'>
<script type='text/lua'>
i = 0;
function start()
return
end
function update()
i=i+1
document:GetEntityByID('test'):GetComponentBase('rotation'):Set('x','..i)
return
end
</script>
</head>
<body>
<a-scene scale="0.1 0.1 0.1">
<a-box color='red' depth='2' height='4' width='0.5'>
<a-entity geometry='primitive: cone; radiusBottom: 1; radiusTop: 0.1; height: 2;'>
</a-entity>
<a-entity geometry='primitive: cylinder; height: 3; radius: 2;' link='href:'>
</a-entity>
</a-scene>
</body>
</html>
```

```
https://oho-sugu.github.io/ovtestpages/Test2.howl; title: My Homepage; target: _self;'></a-entity>
<a-entity geometry='primitive: cylinder; openEnded: true;'></a-entity>
<a-entity geometry='primitive: torus; radius: 2; radiusTubular: 0.2; arc: 180;'>
</a-entity>
<a-box id='test' color='blue' depth='1' height='1' width='1' position='1 1 1' rotation='30 0 30' scale='1.5 1.5 1.5'>
<a-sphere color='yellow' radius='0.5' position='2 1 1' link='href: https://oho-sugu.github.io/ovtestpages/Test2.howl; title: My Homepage; target: _self;'></a-sphere></a-box>
<a-box animation='property: object3D.position.z; to: 2; dur: 1000; loop: true' animation_2='property: object3D.position.x; to: 2; dur: 100; loop: true' color='green' depth='0.5' height='0.5' width='0.5' position='-1 -1 -1'></a-box>
</a-box>
</a-scene>
</body>
```

Orthoverseまとめ

このような仕組みにすることで、地球上のどこにでもコンテンツを配置できるようになっています。もちろん、PLATEAUがないところでは建物の3Dモデルは表示されませんが、地理情報は広範囲を大規模に扱っていきことで大きな価値を生み出します。

現状のように一部の限定された地域でのアプリではなく、より広範囲のサービスという形でPLATEAUや地理情報を活用するものが出てくる一助に本稿がなれば幸いです。