WebGL を用いた広域 GIS 道路データの可視化による駅の有効なカバー範囲の検討と設計

張付新[†1]　川合康央[†1]

概要：現代社会において，特に都市部では，多くの人々が電車や地下鉄などの公共交通機関を利用している．そのため，住居を探すなどの際に，駅からの距離は特に重要な判断材料になることが多い．また，都市計画の観点からも，道路や駅の位置は，その施設を利用可能な住民の数や，特定の駅の利用頻度といった問題を考える上で重要な指標となる．等時性線は，空間ネットワークデータベースの新しいクエリタイプであり，到達可能性分析を行うための有用な手段である．本研究では，駅周辺のカバー範囲を表現するために，従来の円形のカバー半径に代わりに，実際の道路距離に基づいてカバー率を計算し，可視化するものである．

1. はじめに

現代社会における交通ネットワークは複雑化している．特に都市部において，多くの人々は電車や地下鉄などの公共交通を利用している．例えば，部屋を探す場合，駅からの距離や歩く時間は,特に重要な選択条件となる.さらに，都市計画の視点から考えると，施設にどれだけの範囲内の住民が使用できるのか，設置した駅の使用頻度はどうなるのか，などといった問題は，道路と駅の位置関係が重要な指標となる．

都市計画では，到達可能性を分析するなどのタスクを処理する必要がある．都市の戦略的オブジェクトの計画に際しては，これらのオブジェクトを最適な位置に配置しなければならない.たとえば,地下鉄の駅や学校を建設した際，多くの人が快適な時間でその場所に到達できるために最適な場所はどこかを検討する必要がある[1].

また，都市構造の変化において，最も基本的かつ他の要素への影響が大きい都市構成要素の一つとして，道路網が挙げられる．例えば，郊外に新しいバイパス道路が建設されると，その沿道に流通施設や商業施設が集積し，周辺の土地利用形態は大きく変化する．反対に，旧道沿いの市街地では商店街が衰退し，都市の空洞化を招くこともある．区画整理によって道路網が整備されると，周辺の土地の利用価値は向上し，都市化・宅地化が加速する[2]．このようなことから，道路ネットワークによる分析は極めて重要な課題であると考えられる．

さらに，地理や交通ルートのようなデータの分析を行う際，研究者や専門家らは，主として QGIS などのような地理情報システムの専門的なソフトウェアを利用することが多い．しかし，専門的な知識を持っていない場合，ソフトウェアを学習する時間が必要な上，複雑な操作によって，なかなか思うような結果が得られないことが多い．また，地理的なデータ,分析に使用するソフトウェアやデータは，関連組織，もしくは政府組織からのライセンスが必要なこともある．

また，日本の都市部では，商業施設は常に駅から近い場所に集積しているという特徴がある．そこで，道路ネットワークに基づいて，等時線という手法で，駅のカバー範囲を計算し，可視化することによって都市計画に対して参考になるデータが得られると考えられる．この等時線は，空間ネットワークデータベースの新しいクエリタイプであり,到達可能性分析を実行するための便利な手段である[3].

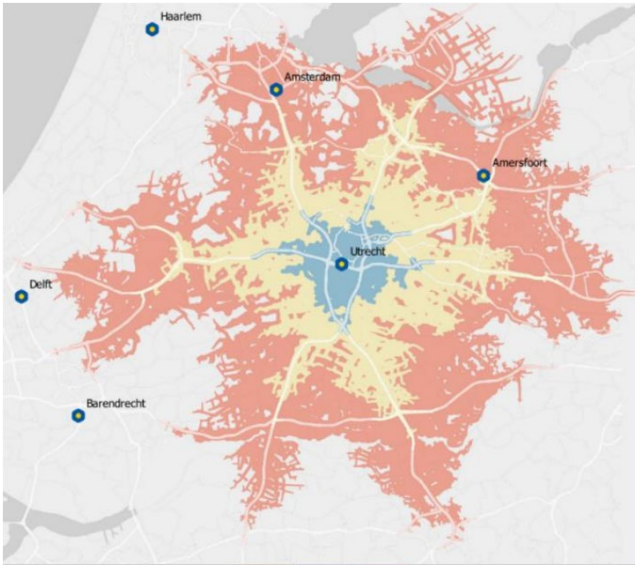

図 1　等時線による可視化の例[4]

都市の分析には，各種の制限があるため，多くの人が使いやすい，オープン化された可視化分析ツールが必要とされていると考えられる．そこで，本研究では，これまでの円形のカバー半径を利用して駅周辺のカバー範囲を表すのではなく，GPS 点で道路と連結して，駅周辺の道路ネットワークを全て線状に表すこととした．また，上述した可視化ツールを利用する際の利便性やライセンスのような課題を解決することも目的とする．分析したデータを表現する

†1 文教大学大学院

ため，一つのデータをレンダリングするエンジンを設計し，構築することとした．

2. 研究の方法

本研究では，利便性を考慮して Web 関連技術を採用した．また，システムを構築するため，オープンソース技術も活用する．必要な性能を考え，主に WebGL（Web Graphics Library）に関する技術を選択した．ブラウザ上で，百万レベルのデータをレンダリングするためには，GPU の力を使うべきだと考えた．背景用マップを表すため，ここでは，mapbox という技術を選択した．

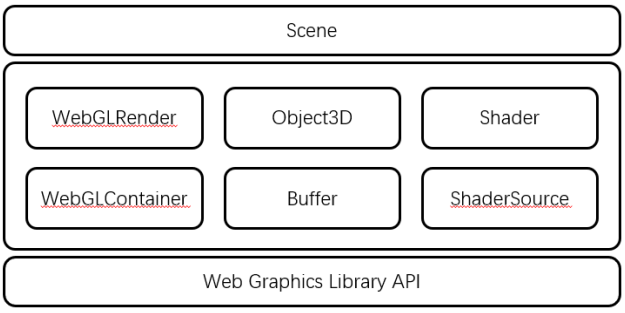

図 2　WebGL 技術を基づいて開発した
レンダリングエンジンの構成設計図

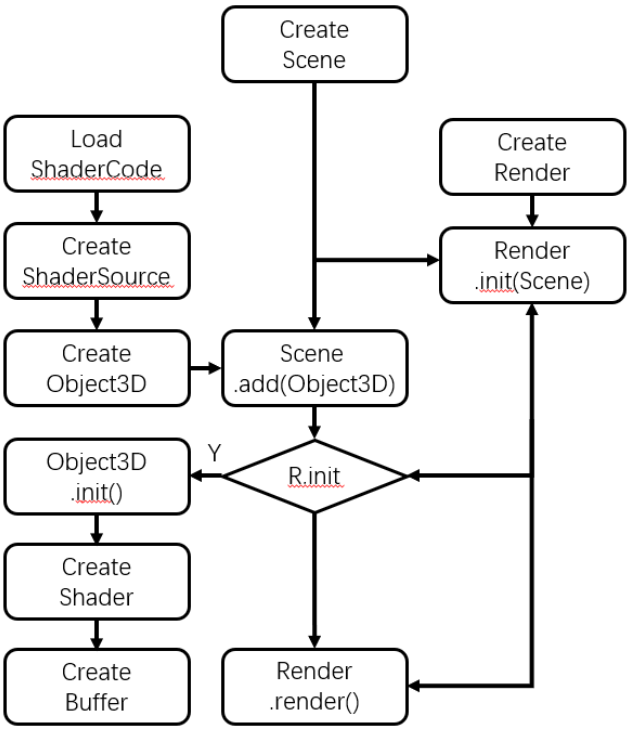

図 3　WebGL 技術を基づいて開発した
レンダリングエンジンのフローチャート

今回開発したレンダリングエンジンは，図 2 で示すように，主として七つの部分で組み立てた．コアとして，真ん中六つのクラスはすべて WebGL(Web Graphics Library) API[5]に基づいて開発した．また，システムの流れを図 3 で示す．Scene は List のような存在であり，すべての Object3D を Scene に入れる．Shader は，glsl コードを処理するためのクラスである．WebGL は，OpenGLES2.0 から派生した API であり，OpenGL ES Shading Language（GLSL ES）即ち glsl コードの記述が必要である．ここでは，図形色，光効果，陰影などの効果は，全て glsl コードによって実装した．Object3D を処理した後，WebGLRender を作り，初期化する．初期化する際，Object3D の初期化も行う．Shader 以外に初期化段階で処理するもう一つの部分は Buffer である．Buffer は，GPU にデータを転送する機能である．初期化完了後，render 関数を実行し，オブジェクトをレンダリングする．

カバー範囲を分析するための道路ネットワークデータは，オープンストリートマップ（OpenStreetMap）[6]を利用した．オープンストリートマップはオープンデータベースライセンス（ODbL 1.0）であり，利用において制約されることもない．さらに，マップデータを更新する場合でも，即座に対応できる．

上述したオープンストリートマップによる道路ネットワークデータは，道路の名前，類別，道路の関連性，経緯度の点座標のデータが揃っている．今回の研究では，経緯度の点座標データを取得し，道路ネットワークを構築し，その上で，分析を行う．

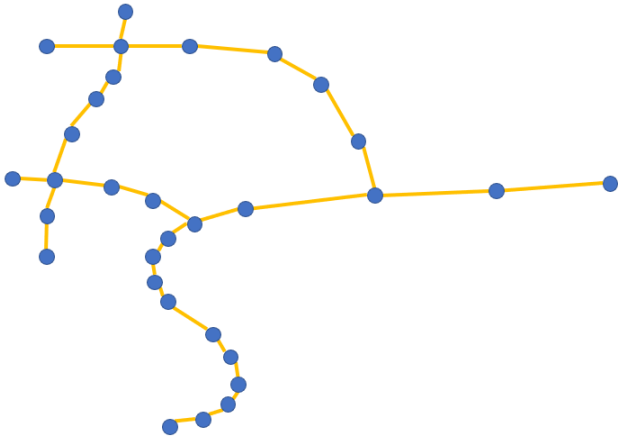

図 4　オープンストリートマップ経緯度点座標の例

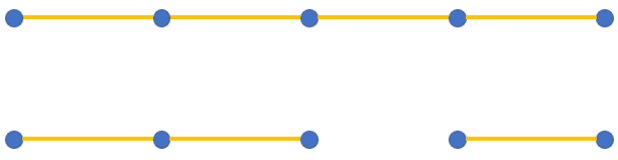

図 5　経緯度点座標を基づいてレンダリングする例

図 4 で示すように，青いポイントは，経緯度の点座標に基づいて道路ネットワークを生成した例である．レンダリングする方法は，次の通りである．索引データを利用する場合，一つのポイントは経度と緯度二つのデータで構成されており，図 5 で示すように 1 から 10 までの 10 個のポイ

ントがある場合，索引データと共に組み合わせ，レンダリングする．ここで WebGL の API，gl.drawArrays 最初のパラメーターで gl.LINES を使う場合で，索引データのコード例を以下のように示す．

```
var indices = [
                    1, 2, 2, 3, 3, 4, 4, 5, // 線 1
                    6, 7, 7, 8,             // 線 2
                    9, 10                   // 線 3
                ]
```

索引データを利用しない場合，上述した indices のように，ポイントデータを二回入力して，レンダリングする．索引データを利用するかどうかは，状況によって判断する．今回の道路ネットワークの計算エンジンは，GraphHopper[7]という技術を利用するため，計算結果の中で索引データが含まれていないため，索引データを利用せずにレンダリングする．上述した GraphHopper は Java 言語を基づいて開発されたオープンソースのルーティングライブラリである．計算方法は，最短距離の Dijkstra 法を基づいて完成した．ルートは，マイナス距離が出てきた場合，計算できなくなるということが，Dijkstra 法の欠点の一つである．しかし今回は，道路ネットワークの計算であり，道路の距離はマイナスがないため，計算スピードが早い．

3. 結果と考察

実験段階で，開発したエンジンを使って，神奈川県内すべての道路データを取得した．既に開発したエンジンを利用し，図 6 で示すように，実験を通して 300 万のデータをレンダリングできるということを実証した．実験指数を表 1 のように示す．

表 1　実験指数

名称	指数
頂点数	311835 個
経度データの長さ	6077832
索引データの長さ	3428218
フロントエンド側処理時間	約 42ms

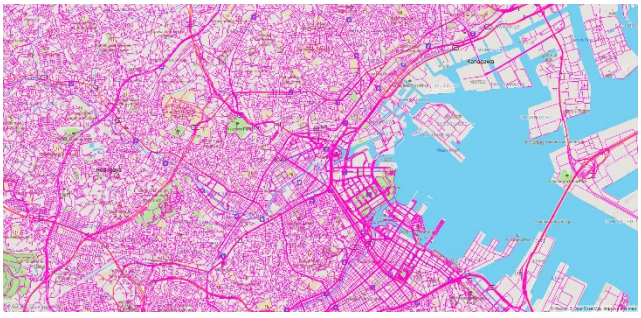

図 6　実験による道路ネットワークレンダリングした効果

また，横浜駅を中心として，駅からの距離 2km と距離 5km の二つの状況について計算を行った．実装したシステムの可視化効果を図 7, 8，表 2 に示す．

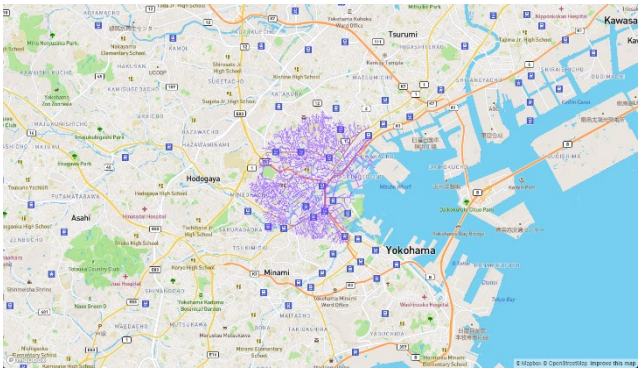

図 7　横浜駅 2km 以内のカバー範囲

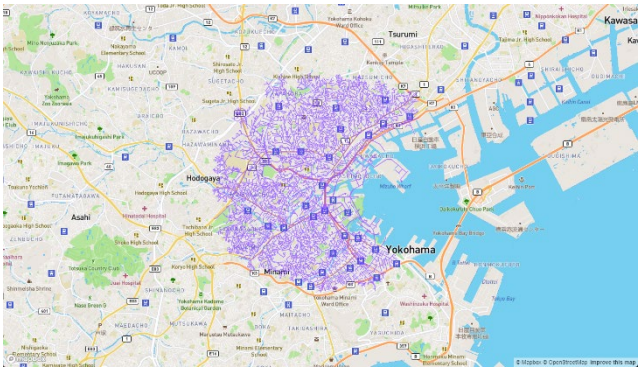

図 8　横浜駅 5km 以内のカバー範囲

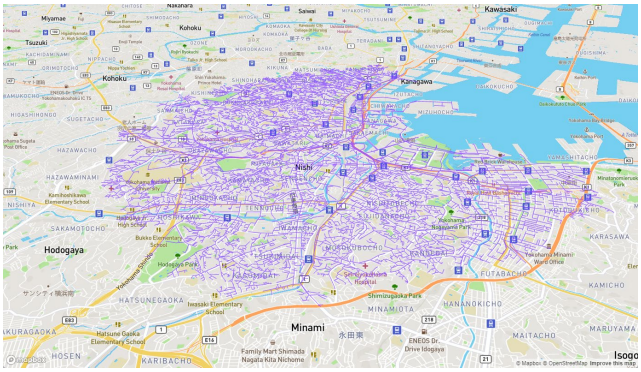

図 9　視角を自由に変更した効果

表 2　横浜駅 5km 以内のカバー範囲の計算した結果指数

名称	指数
頂点数	44430 個
経度データの長さ	88860
フロントエンド側処理時間	約 25ms
CSV データ転送による時間	約 395ms

実験結果から，これまでの円形のカバー効果の可視化よりも，精度が高い結果が得られた．さらに，図 9 に示すように，WebGL に基づいて開発したシステムであるため，自由に視角を変換する機能も搭載している．こうした機能によって，分析を行う際により利便性を向上できると考えられる．

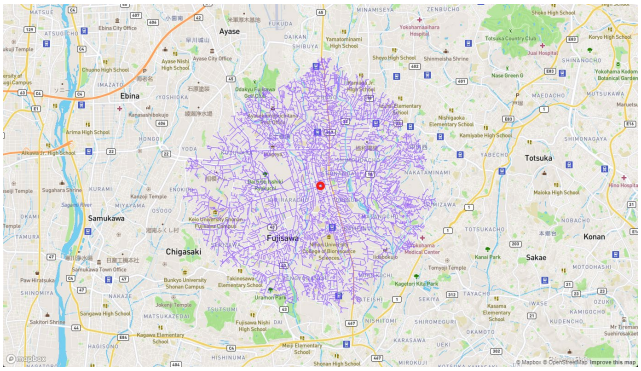

図 10　湘南台駅 5km 以内のカバー範囲

表 3　湘南台駅 5km 以内のカバー範囲の計算した結果指数

名称	指数
頂点数	34575 個
経度データの長さ	69150
フロントエンド側処理時間	約 19 ms
CSV データ転送による時間	約 118ms

また，他の駅を対象とした例として，湘南台駅（神奈川県藤沢市）のカバー範囲を図 10 に示す．赤い点が湘南台駅であるが，西側のカバー範囲は東側のカバー範囲より広いものとなった．これは，駅西側では曲がっている道路が少ないことが考えられる．特に，西側は工業地帯でもあるため，真っ直ぐな幹線道路や計画的な道路計画が行われている．一方，東側には幹線道路がほぼなく，大半の道路は曲がっている．その結果，駅東側のカバー効果は西側と比べるとやや狭いものとなっている．

さらに，駅に近い箇所に国道 467 号線が南北に走っている．この幹線道路により，駅の南側と北側のカバー範囲が広いということも，可視化によって読み取ることが可能である．湘南台駅のカバー範囲の例から見ると，駅を中心として，真っ直ぐ幹線道路があることは，駅のカバー範囲について極めて重要なことである．

さらに，横浜駅と湘南台駅の二つの計算指数から見ると，横浜駅では道路ネットワークの点が多い．これは，横浜駅は湘南台駅より道路密度が高いことを示している．

4. まとめ

本研究は，緯度の点座標データで道路を連結し，駅周辺の道路ネットワークを線状に表すことで，駅周辺のカバー範囲を表すものである．今回道路ネットワークの計算は，OpenStreetMap のデータを利用し，GraphHopper に基づいてサーバーを設置し，カバー範囲を計算した．Dijkstra 法を利用したことによって，リアルタイム計算でも，処理速度は確保できるものとなった．さらに，新しい駅を設置して，その駅のカバー範囲を計算するような模擬実験も，より簡易に実施できると考えられる．また，ブラウザによる技術で開発されたシステムによって，研究者以外の人でもより簡単に利用できるものとなった．今後，ユーザビリティを考えより実用的な機能を追加することとする．

本研究は，WebGL によるデータ可視化であり，情報可視化という手法によって，都市における人々のためのデータを利用し，分析した結果をより分かりやすく伝えることを目指したものである．技術面では，WebGL に基づいてデータ可視化専用のエンジンを開発し，より多くの技術者に提供したいと考えている．

参考文献

[1] Marciuska, Sarunas, and Johann Gamper. Determining objects within isochrones in spatial network databases. East European Conference on Advances in Databases and Information Systems. Springer, Berlin, Heidelberg. 2010, p.1-2.
[2] 安井謙介，貞広幸雄．道路網の変化における時空間分析手法．日本建築学会計画系論文集, 2003, vol.68, no.569, p.147-153.
[3] Markus Innerebner. Isochrones in Multimodal Spatial Networks. Ph.D. Thesis in Computer Science. 2013.
[4] Berg, Joris den, et al. Towards a dynamic isochrone map: Adding spatiotemporal traffic and population data. LBS 2018: 14th International Conference on Location Based Services. Springer, Cham, 2018, 95p
[5] “Webgl API docs”. https://developer.mozilla.org/zh-CN/docs/Web/API/WebGL_API, (参照 2022-12-19).
[6] “OpenStreetMap”. https://www.openstreetmap.org, (参照 2022-12-19).
[7] “Graphhopper API docs”. https://docs.graphhopper.com/, (参照 2022-12-19).