目的別空間データベースサーバを用いた Web システムの設計と実装

島根大学　総合理工学部　数理・情報システム学科

計算機科学講座　田中研究室

S033033　木村 眞吾

平成 20 年 2 月 8 日

目次

第１章　序論

１．１．研究目的

現在，Google Earth[1]や Google Maps[2]のような地図サーバを用いた Web サイトが増えてきている．こういったサイトは地図上にレストランなどの店舗情報を表示し，今までは住所しか分からなかったものを地図上に表示することでより店舗などの位置を探しやすくなったといえる．しかし，現在位置の周辺店舗の検索といった空間検索を行なっているものは少ない．

このことから，人口・道路などの更新の少ない整備されたデータだけでなく，気候・店舗情報(例：広告のチラシやレストランのメニュー)といった更新の多い整備されていないデータ，これらについて必要な部分だけ抽出し，一般に公開された地図上に追加して描画するシステムがあれば有益であろう．

そこで，本研究では，こういったシステムの基本的なものとして国道沿いの人口データを検索し，結果を Google Earth 又は Google Maps で描画するというシステム[3]と PC と携帯端末で使える近隣バス停検索が可能なバス停時刻表検索システムの実装を行った．携帯端末では GPS 又は簡易位置情報を利用し，PC では地図上から位置を指定することで近隣バス停位置検索を行う．

１．２．研究概要

前述したシステムの実装にあたり，座標データの管理や検索には DBMS を用いた．本システムではその中でも座標データといった文字数値以外のデータ容易に扱うことが出来，近傍検索といった空間検索も行うことができることから ORDBMS を使用した．

また，Web サーバには動作が軽く，高いシェアを持ち，拡張性が高いオープンソースである Apache を用いた．クライアントーWeb サーバ間をつなぐ言語も必要となってくるのでその言語には効率性，可搬性などの点から Java サーブレットを用い，サーブレットコンテナには安定性が高く，オープンソースであることから Tomcat を使用した．Web サーバーDB サーバ間の接続にはクライアントーWeb サーバ間に Java を用いたことから JDBC ドライバによる用いることとした．

クライアント上への描画には Google Earth の場合は，Google Earth 上に追加して描画することが出来る KML を用い，Google Maps の場合は，Google Maps 上に追加して描画することが出来る Google Maps API を用いた．ただし，携帯端末の場合は Google Maps API を使うことが出来ないので，Google Maps を jpg 画像として出力機能を用いた．

PC と携帯端末で使える近隣バス停検索が可能なバス停時刻表検索システムの地図には誰でも使用でき，動作性も良く，携帯端末でも使用できることから Google Maps を用いる．

１．３．先行研究

本研究では先行研究調査として，世界の 4,000 以上の出版社から出版される 15,000 以上の科学・技術・医学・社会科学のタイトルを網羅する世界最大級の書誌・引用文献デ

ータベースである Scopus を使用した（表 4.1.1）.

検索キーワードを「Google Earth database」又は「Google Maps database」とした場合 30 件の結果が得られた（表 1.3.1）.

表 1.3.1　先行研究検索結果

キーワード	検索結果（件）
(Google Earth database)or(Google Maps database)	30

30 件の検索結果の中の文献はほとんどが Google Earth に海洋．地質，農業などといったデータを重ねあわせ視覚的に見やすくすることで何か新しい発見するといったもので本研究と関連のあるもとしては表 1.3.2 に記載してあるものだけであるといえる．これらの研究は不動産データを Google Maps に重ね合わせ，ユーザがより不動産データの検索がいやすくなるような Web ページを作成したものや空間データが大きなものであることから生じる応答時間の遅延を最新のネットワーク技術と Web 地図技術により抑えるものや GPS を用いたルート検索の手法の提案といったものであり，本研究とは類似しているが違うものであることが分かる．以上のことから Google 環境を用いた本研究と同様の研究は大変少ないことがわかる．

表 13.2　関連研究

著者名	タイトル	掲載誌名
Hwang, J.	Application based on ArcObject inquiry and Google maps demonstration to real estate database（2007）	Proceedings of SPIE – The International Society for Optical Engineering 6754 (PART 2), art. no. 67542F
Yu, Z., Wang, Y., Luo, B.	Study on architecture and implementation of adaptive spatial information service（2007）	Proceedings of SPIE – The International Society for Optical Engineering 6754 (PART 2), art. no. 675426
Letchner, J., Krumm, J., Horvitz, E.	Trip Router with Individualized Preferences (TRIP): Incorporating personalization into route planning（2006）	Proceedings of the National Conference on Artificial Intelligence 2, pp. 1795-1800

第２章　DBMS

２．１．ORDB

空間検索オブジェクトリレーショナルデータベース（ORDB）とは，従来のリレーショナルデータベース（RDB）の延長線上に位置しており，RDBにオブジェクト指向の環境を提供するものである．例として，オブジェクト指向環境の「クラス」をORDBMSのユーザ定義型と読み換えるのみで，あとはそのまま「メソッド」，アクセス制御の「private or public 」，型継承関係などSQL-99で提供されている．このため，ＲＤＢＭＳにおいて別々のＤＢに格納していた汎用データと空間データをＯＲＤＢＭＳでは同一のＤＢに格納することが可能となる[4]．

２．２．SQL-99 によるユーザ定義型の例

(x,y) 座標から構成される「点データ型」の場合（ユーザ定義型とはいっても，想定しているのはエンドユーザではなく，アプリケーション開発者である）．

```
１） CREATE TYPE ST_Point
２） UNDER ST_Geometry AS (
３）    ST_PrivateX DOUBLE PRECISION DEFAULT NULL,
        ST_PrivateY DOUBLE PRECISION DEFAULT NULL
    )
４） INSTANTIABLE
５） NOT FINAL
６） METHOD ST_X()
    RETURNS DOUBLE PRECISION
    LANGUAGE SQL
    DETERMINISTIC
    CONTAINS SQL
    RETURNS NULL ON NULL INPUT,
    ・・・
```

１）ST_Point というデータ型を定義する．

２）ST_Point は ST_Geometry 型（論理的な整合性を取るための実体の無い型）を継承する．

３）ST_Point を倍精度の x,y 座標値から構成する．

４）インスタンスを持つ．つまり実体がある．（ST_Geometry は NOT INSTANTIABLE）

５）さらに継承可能

６）ST_Point 内のX 座標を返すメソッドを，ユーザ定義型内部で定義する．

このユーザ定義型を使ったイメージとしてバス停名「松江駅」とその位置を格納する例を考えてみると，表2.2.1のようなイメージでST_Geometry型に位置を格納している．

表2.2.1　ユーザ定義型の例

バス停名	ST_Geometry
松江駅	

２．３．ORDB の利点

ＲＤＢＭＳで汎用データと空間データを同一ＤＢに格納した場合について．ある　区画内においての人口総数をＲＤＢＭＳに格納した場合を例として示す．ＤＢに格納する要素は次のものとし，空間情報については座標点Ｘ，Ｙを４点用いて作られた四角形を１区画として扱うものとする．

・識別ＩＤ

・人口総数

・空間情報（ＸＹ座標×　４点）

まず，次のようなものが挙げられる．

表１．２．１　横長にした場合

識別ＩＤ	人口総数	X1	Ｙ１	Ｘ２	Ｙ２	Ｘ３	Ｙ３	Ｘ４	Ｙ４
1	100	10	10	10	20	20	20	20	10
2	200	20	20	20	30	30	30	30	20
3	300	30	30	30	40	40	40	40	30
4	400	40	40	40	50	50	50	50	40
5	500	50	50	50	60	60	60	60	50
・	・	・	・	・	・	・	・	・	・
・	・	・	・	・	・	・	・	・	・

表１．２．１ のように横長にした場合，例のような４点で構成される単純な図形ならまだしも，複雑な図形になってくると，ＸＹ座標の数が多くなるにつれ表のカラム数も多くなるという欠点がある．

表１.２.２　縦長にした場合

識別ＩＤ	人口総数	X	Y
1	100	10	10
1	100	10	20
1	100	20	20
1	100	20	10
2	200	20	20
2	200	20	30
・	・	・	・
・	・	・	・

表１.２.２ のように縦長にした場合も横長のものと同様に，ＸＹ座標の数が多くなるにつれ行数も増えてしまうだけでなく冗長性が生じてしまうという欠点がある．

このようにＲＤＢＭＳは文字，数値以外のデータを扱うには不向きであるのに対し，ＯＲＤＢＭＳは同一に格納することにより汎用データからの検索が行え，さらに空間データを検索対象とした検索も可能であるという利点がある．

２．４．空間データの操作

ORDB では大まかに Point（点）型，LineString（折線）型，Polygon（多角形）型が存在している．これらのデータを Buffer メソッドにより Point 型又は LineString 型を Polygon 型へ変換したり，Intersects メソッドにより検索範囲の図形の交差を判定したり，Within メソッドにより検索範囲の図形が完全に含まれるかを判定する．

２．４．１．空間データのフォーマット

空間データの表現方法として Well-Known Text（WKT）フォーマット形式がある．これは，OGC によって定義されている図形をテキストで表現する方法であり，空間検索システムや SQL 等のこれをサポートしているソフトウェア同士でデータの受け渡しを容易に行うことができる[5]．本研究で用いている HiRDB＋空間検索プラグインでも用いている．以下に，WKT の例を挙げる（図 2.4.1）．

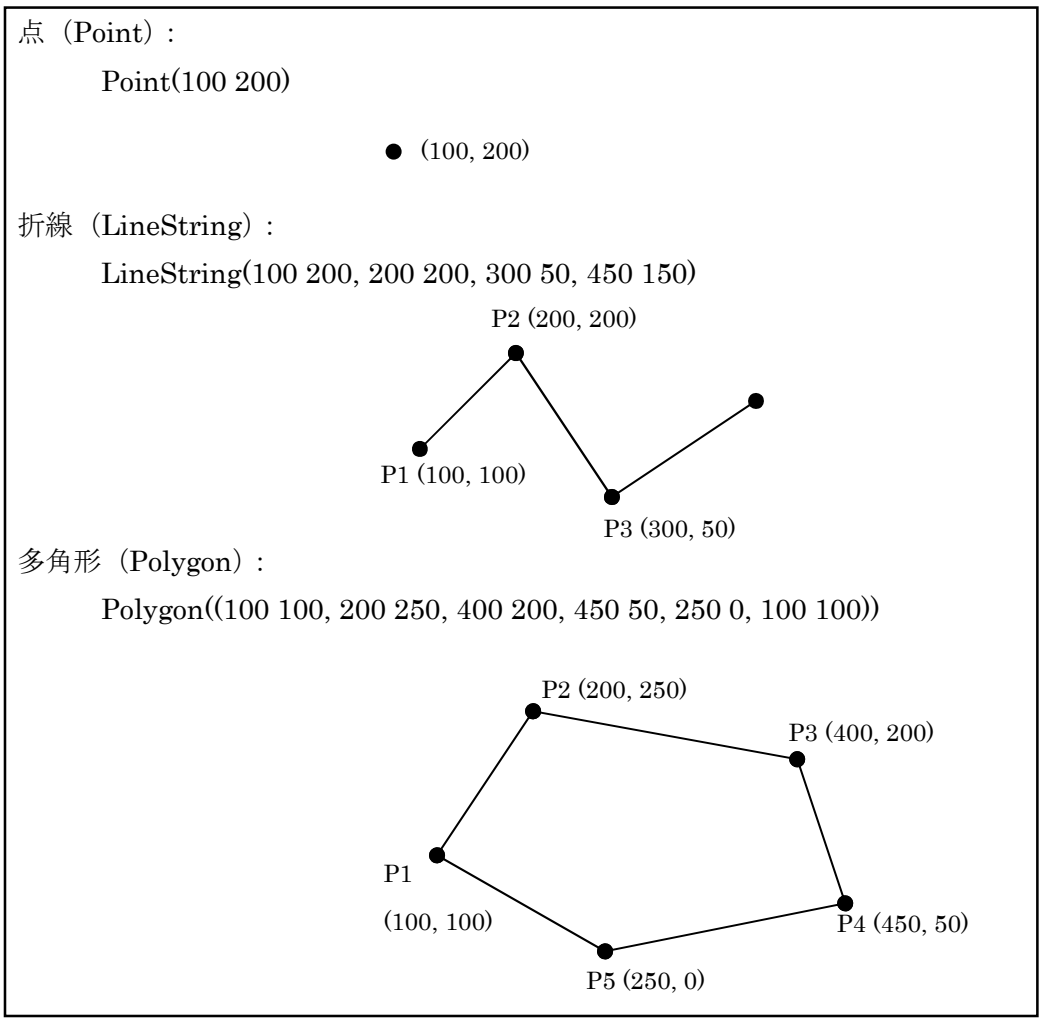


図 2.4.1

２．４．２．Buffer メソッド，Intersects メソッド，Within メソッド

Buffer メソッド[5]というのは Point 型に対しては指定した値を半径とする円を生成し，LineString 型のデータに対しては指定した値の幅を持つ Polygon 型を生成するという機能を持つ.

Intersects メソッド[5]というのは検索する列にある Point，Line String，Polygon 型で格納されているデータと検索範囲の図形との空間関係をチェックして，前者が後者に「交差するかどうか」を判定する機能を持つ.

Within メソッド[5]というのは Intersects メソッドに対して「交差するかどうか」ではなく「完全に含まれるかどうか」は判定する機能を持つ.

これらのメソッドを組み合わせることで Buffer メソッドにより，現在位置から 500m 以内の距離にある円を生成し，Within メソッドにより，その円の中に Point 型で格納されているコンビニの座標が含まれているどうか判定することで現在位置から 500m 以内にあるコンビニといった検索を行うことが可能となる（図 2.4.2）．また，Line String 型のデータに Buffer メソッドを用いれば，国道から 500m以内の距離にあるコンビニといった検索も可能となる（図 2.4.3）.

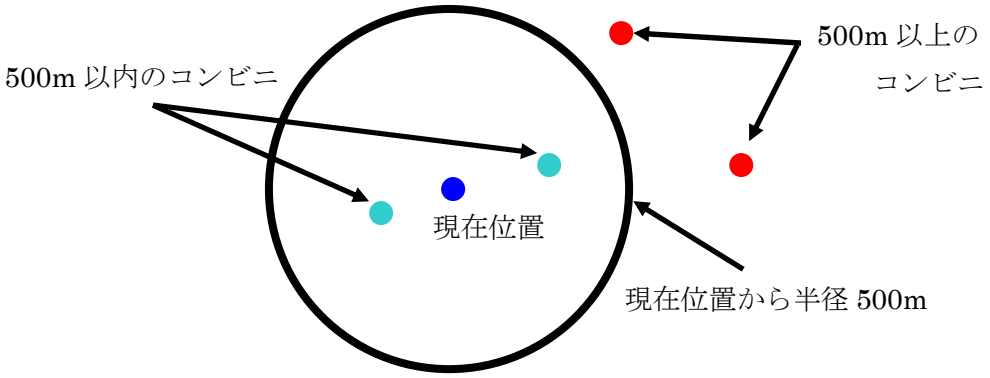


図 2.4.2　半径 500m以内にあるコンビニの検索

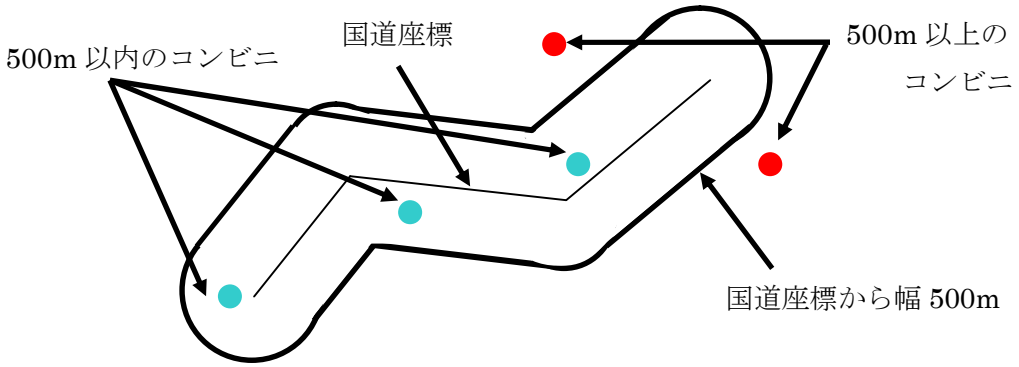


図 2.4.3　国道から 500m以内にあるコンビニの検索

第３章　Web システム

３．１．Java サーブレット

Java サーブレットとは，Web サーバ上で実行されるモジュール(部品)化された Java プログラムのことである．サーブレットを追加することにより，Web サーバの機能を拡張することができる．サーブレットは Java 言語で記述されているため，特定の OS やハードウェアに依存することがなく，サーブレット API を実装したあらゆる Web サーバで稼動させることが出来る．また，サーブレットと似た技術に CGI がある．CGI に対するサーブレットの利点を次に示す[6].

- 効率的

  CGI の場合, HTTP リクエスト毎に新たなプロセスが生成される. それに対し，サーブレットの場合, Java 仮想マシンが唯一のプロセスとして常に動作しており，個々のリクエストはスレッドによって処理される．そのため，プロセス起動のためのオーバーヘッドが生じない．また，処理終了後もメモリ上に存在するため，複数のリクエストから簡単に利用できる.

- 開発しやすさ

  サーブレットは HTML フォームデータの構文解析，HTTP ヘッダの読み取りと設定，クッキーの処理，セッションの管理，といった多くの基礎的機能を豊富に備えている．また，信頼性も高く再利用性にも富む.

- 強力

  複数サーブレットでのデータベース接続の共有など，資源を共有する最適化を容易に実装することができる．また，サーブレットは複数のリクエストにまたがって情報を保持するため，セッション管理や，前の処理結果のキャッシュなどの技術も簡単に実現できる.

- 可搬性

  サーブレットは Java の標準 API を使用するため，一般的にプラットフォーム非依存である.

- 安全

  CGI は通常 OS のシェルから実行するため，特殊文字（¥n など）に対して細心の注意が必要である．また，配列に対するバッファオーバフローの攻撃対象になりうる．これに対し，サーブレットはこれらの問題点がない.

３．２．JDBC

JDBC とは Java プログラムからリレーショナルデータベースにアクセスするための API のことである．SQL 言語による命令を発行してデータベースの操作を行なうことができ，データベースの種類によらない汎用性の高いプログラムを開発することが可能である[7]．また，JDBC は 4 つのタイプに分類される．本システムではタイプ２を用いる．

・タイプ１：JDBC-ODBC Bridge ドライバ
　既存の ODBC ドライバを使ってデータベースへアクセスする方式.

・タイプ２：Native API partly ドライバ
　データベースシステム固有のドライバソフトウェアを使用する．JDBC API をデータベース固有 API へ変換し，このドライバへ渡す.

・タイプ３：Net Protocol All-Java ドライバ
　クライアントとデータベースの間に中継サーバを配置し，このサーバに Java で作成されたドライバを格納する．クライアントは実行時にこのドライバをダウンロードして使用する．クライアント側で実行した JDBC 処理は，サーバ上の中継プログラムがデータベース固有の命令に変換して実行される.

・タイプ４：Native Protocol All-Java ドライバ
　データベース固有のドライバの機能を Java で作成し，JDBC ドライバ内部に取り込んだもの.

３．３．Web システムの概要

本システムでは Apache と Tomcat を連携させることにより，Java サーブレットや JDBC を用いた Web システムの構築をおこなっている．具体的には静的コンテンツ Apache が処理し，動的コンテンツを Tomcat が処理するといった方法をとっている．Tomcat にも Web サーバの機能は存在するが，Apache に比べ処理が遅く，細かな設定が出来ないという理由から本システムでは使用していない.

Apache と Tomcat の連携には Web サーバアダプタを使用している. DBMS と Tomcat の連携には JDBC ドライバを用いた Data Source[8]による接続を行っている.

３．３．１．Data Source

Data Source とは，Tomcat に JDBC ドライバを定義することで，サーブレットが DBMS へ接続から切断までの処理を行う必要がなくなる，また，１度接続したらその接続を保持することで，DBMS への接続時間も短くなり，さらにその接続を他のサーブレットに使わせることも可能であるというメリットがある.

それに対し, Data Source と同様の処理が行える Driver Manager では、個々の Java プログラム内に JDBC ドライバを定義し，接続はそれぞれのサーブレットが行い，サーブレットが生成されるたびに，DBMS への接続から切断までの処理が行われる.

第４章　国道沿いの人口分布を表示するシステム

４．１．システムの概要

本システムは「島根県内の国道沿いの左右帯域幅 x km($0.5 \leq x \leq 5$: 500m 毎) 内に含まれる選択された人口データに対する1km × 1km の基準地域メッシュの表示」という検索をWebサーバやDBサーバを用いて行い，Google Earth又はGoogle Maps上に結果を表示するというシステムの設計と実装を行った.

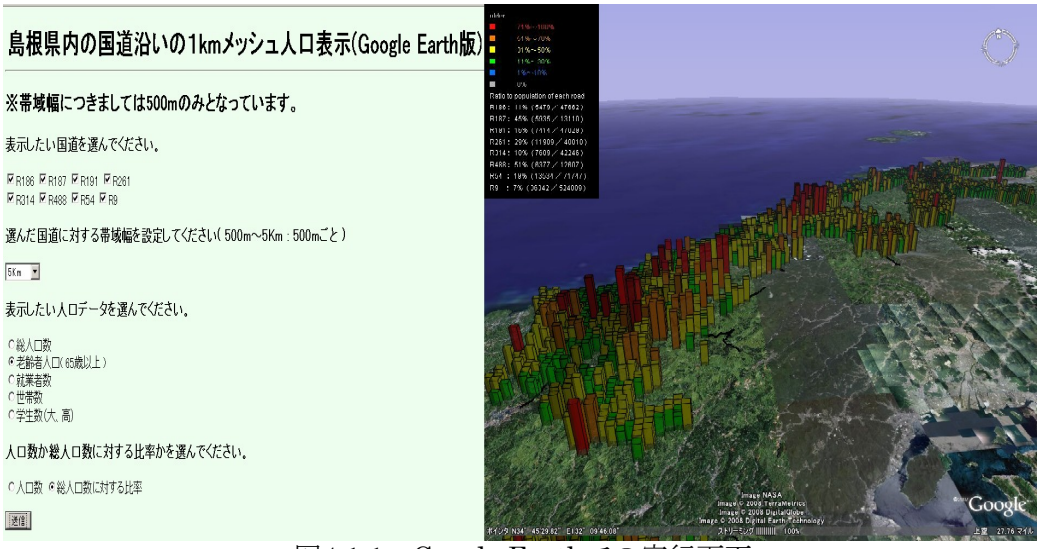


図4.1.1　Google Earthでの実行画面

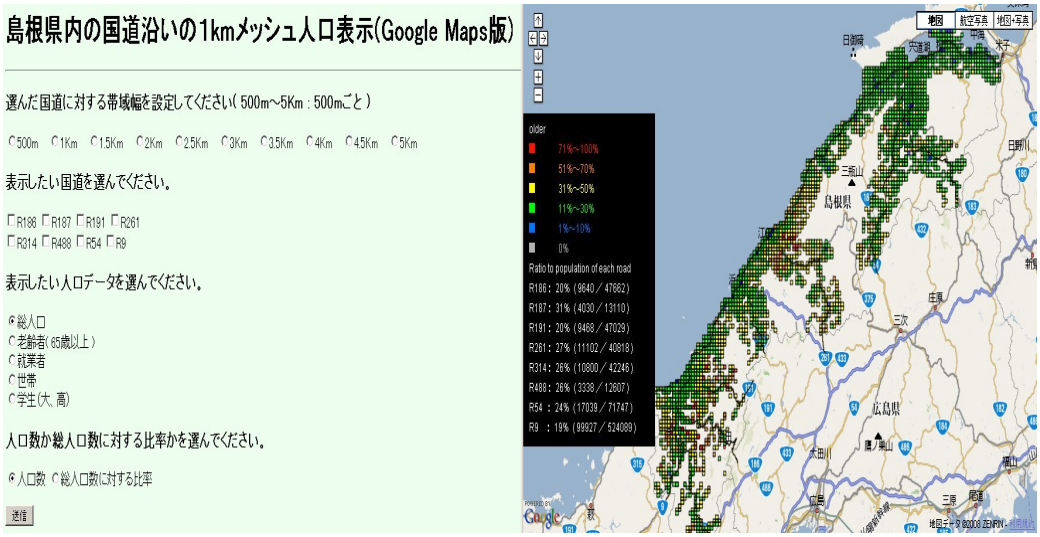


図4.1.2　Google Mapsでの実行画面

４．１．１．システムの流れ（使用方法）

使用方法はまず，ユーザがブラウザ上で表示したい国道をチェックボックスで選択（複数可）し，その国道に対する検索範囲である帯域幅を指定する．帯域幅の範囲は500mから5kmまで500m毎の値を選択ボックス又はラジオボタンから１つ選択する．次に，知りたい人口データの種類と表示方法を選択する．表示方法は人口データの結果を人口数にするか比率にするかをラジオボタンで選択する（①）．最後に，送信ボ

タンをクリックし必要なデータをWebサーバへ送信する（②）．

Webサーバはそれらの引数からJavaサーブレットを用いてDBの操作に必要なSQL言語を生成する（③）．そして，JDBCを用いてＤＢMSに接続し，検索を行なう（④⑤）．最後に，DBMSから受け取った検索結果を用いてJavaサーブレットでKML又はGoogle Maps APIを生成し，Google Earth又はGoogle Mapsに結果を描画する（⑥⑦⑧⑨）．

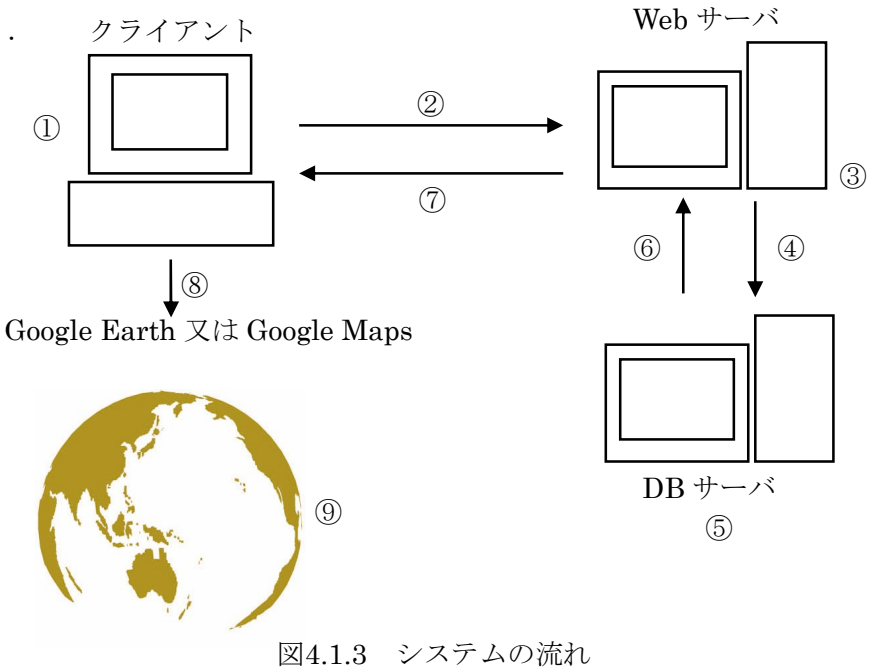


図4.1.3　システムの流れ

４．２．使用データ

本研究で必要な国道座標値データと人口メッシュデータの取得元は（財）日本地図センター刊行のデジタルマップ「平成11年JMC マップ」[9]と（財）統計情報研究開発センター刊行の「平成7年度国勢調査に関する統計」[10]である．これらの取得元データの必要な部分を抽出し，本システムで扱えるように加工したものを用いた．

４．３．地域メッシュコード

「平成11年JMC マップ」と「平成７年度国勢調査に関する地域メッシュ統計」に共通する規定である地域メッシュコードについて説明する．地域メッシュコードは，「標準地域メッシュ・システム」（昭和４８年行政管理庁告示第１４３号「統計に用いる標準地域メッシュ及び標準地域メッシュコード」及び日本工業規格「JIS X 0410 地域メッシュコード」）に基づくもので，一定の経度，緯度によって地域を網の目状に区画したときの位置を示すコードである．地域別に情報を表示する方法（ メッシュ法）は，統計データの表示をはじめとして，地形，自然環境，行政地域，道路・鉄道，公共施設，文化財などの位置・範囲等を数値化して表示するなど，多方面で利用されている．

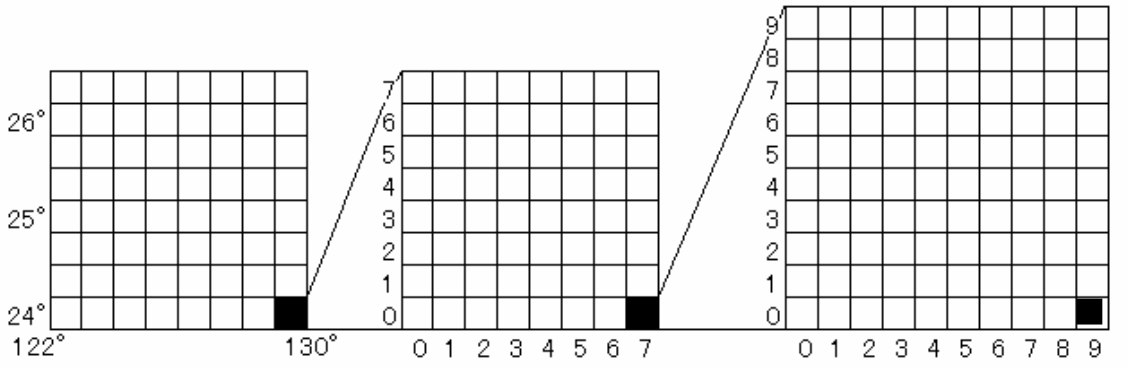
４．３．１．地域区画区分

地域メッシュには，主に第１次地域区画，第２次地域区画，第３次地域区画（基準地域メッシュ）の３つの区画がある（第４次地域区画は用いないため説明を省略する）．第１次地域区画というのは全国の地域を 1 度毎の経線と 3 分の 2 度毎の緯線によって

分割したものである（20 万分の 1 地勢図：ほぼ 80 平方キロメートル）．第 1 次地域区画に 8 等分したものが第 2 次地域区画（2 万 5 千分の 1 地勢図：ほぼ１０平方キロメートル）である．第 2 次地域区画を縦横に 10 等分したものが第 3 次地域区画（ほぼ 1 平方キロメートル）である．

例：メッシュコードが５４３８０７０９のとき

１次メッシュ　　２次メッシュ　　３次メッシュ



５４３８　　５４３８０７　　５４３８０７０９

１次メッシュは４桁のメッシュコードで表す．

上２桁：南端緯度　×　１．５（ただし，分の単位も含む）

下２桁：西端経度の下２桁

【例】　南端緯度　３６度　００分　　西端経度　１３８度　　の場合

上２桁：３６×１．５＝５４　　下２桁：３８　→　５４３８

２次メッシュは６桁のメッシュコードで表す．

上４桁：１次メッシュ地域区画のメッシュコード

５桁目：１次メッシュに８等分し，南から０～７の番号を付け，これをそれぞれの区画を示す数字とする．

６桁目：１次メッシュを横に８等分し，西から０～７の番号を付け，これをそれぞれの区画を示す数字とする．

【例】　５４３８０７

３次メッシュは８桁のメッシュコードで表す．

上６桁：２次メッシュのメッシュコード

７桁目：２次メッシュを縦に１０等分し，南から０～９の番号を付け，これをそれぞれの区画を示す数字とする．

８桁目：２次メッシュを横に１０等分し，西から０～９の番号を付け，これをそれぞれの区画を示す数字とする．

【例】　５４３８０７０９

４．４．国道データ

ＪＭＣマップは日本地図センター刊行のデジタルマップで，日本全国を２０万～５０万分の１程度の縮尺に対応する数値地図情報である．行政界・道路・鉄道・河川・海岸線が数値化された線データと，行政名・自然地名等の点データから構成されており，広い範囲を概観するのに適した地図の骨格をなす情報を収めたデータである．なお，データの座標値は第２次地域メッシュ区画単位で記録されており，各座標値は左下を（0,0），右上を（10000,10000）とする正規化座標となっている．

４．４．１．データの記録方式

１．データの概要

データファイルは１次メッシュ単位になっており，以下の図のような構造になっている．

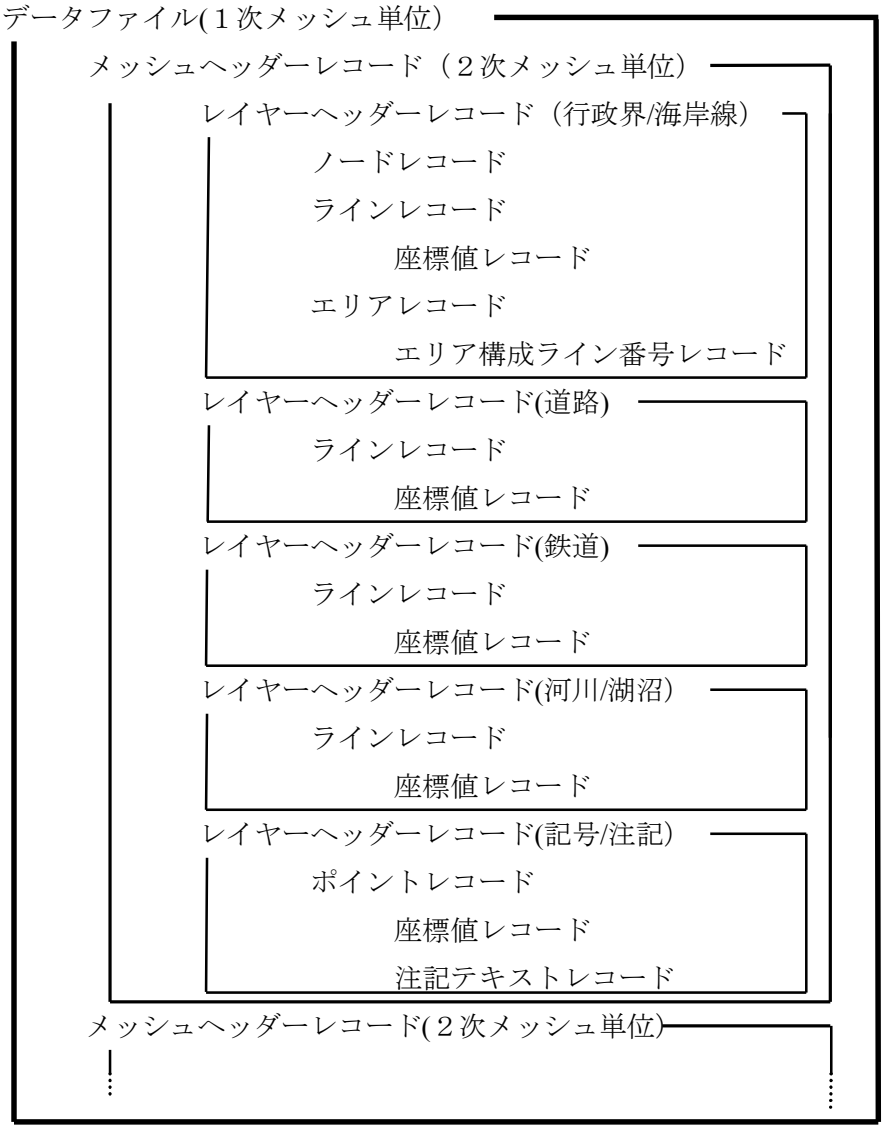


図4.4.1 データファイルの概要

２．メッシュヘッダーレコード

項目	開始	終了	形式	内容
レコードタイプ	1	2	A2	”M”を記録
2次メッシュコード	3	8	I6	2次メッシュコード
図名	9	28	N10	当該2万5千分1地形図名
レイヤー総数	29	31	I3	当2次メッシュに含まれるレイヤー総数
ノード総数	32	36	I5	当2次メッシュに含まれるノード総数
ライン総数	37	41	I5	当2次メッシュに含まれるライン総数
エリア総数	42	46	I5	当2次メッシュに含まれるエリア総数
ポイント総数	47	51	I5	当2次メッシュに含まれるポイント総数
レコード総数	52	56	I5	当ヘッダーレコードを除いた 当2次メッシュに含まれるレコード総数
空白	57	72	16X	

例: M 303650 沖ノ鳥島 2 3 3 1 19

３．レイヤーヘッダーレコード

項目	開始	終了	形式	内容
レコードタイプ	1	2	A2	”H1”:構造化が行われていないレイヤー
レイヤーコード	3	4	I2	”H2”:構造化が行われているレイヤー
ノード総数	5	9	I5	レイヤーの内容を表す 1:行政界海岸線 2:道路 3:鉄道 5:河川湖沼 7:記号注記
ライン総数	10	14	I5	当レイヤーに含まれるノード総数
エリア総数	15	19	I5	当レイヤーに含まれるライン総数
ポイント総数	20	24	I5	当レイヤーに含まれるエリア総数
レコード総数	25	29	I5	当ヘッダーレコードを除いた当レイヤー内のレコード総数
空白	30	30	I5	
レコード総数	31	34	I5	当レイヤーの最初の生成年月西暦年の下2桁と月2桁
空白	35	35	IX	
レコード総数	36	39	I4	当レイヤーのデータ更新年月西暦年の下2桁と月2桁
空白	40	72	33X	

例: H2 1 3 3 3 0 15 9410 9503

４．ライン（項目）・座標値レコード

・ラインレコード

項目	開始	終了	形式	内容
レコードタイプ	1	2	A2	”L”を記録
レイヤーコード	3	4	I2	1:行政界海岸線 2:道路 3:鉄道 5:河川湖沼
データ項目コード	5	6	I2	下のライン項目コード表参照
ライン一連番号7	7	11	I5	レイヤー内で当ラインが何番目に位置するかを示す一連番号
ライン種別コード	12	17	I6	ライン種別コード表参照
始点ノード番号	18	22	I5	
始点接続情報	23	23	I1	0:図葉内ノード 1:隣接図葉に接続する図郭線上のノード
				2:隣接図葉に接続しない図郭線上ノード
終点ノード番号	24	28	I5	
終点接続情報	29	29	I1	始点接続情報と同じコード
左側行政コード	30	34	I5	ラインの向きに対して左側の行政コード
右側行政コード	35	39	I5	ラインの向きに対して右側の行政コード
座標点の数	40	45	I6	当ラインを構成するXY座標点の数(始終点ノードを含む)
空白	46	72	X27	

例:

```
L    1    5    1    0    10    101342199999          5          0
```

・ライン種別コード

レイヤー	コード	データ項目
行政界海岸線	0	確定境界線
	1	仮説境界線(陸部)
1	2	仮説境界線(水部)
	9	図郭線
道路	0	地上
2	1	地下・トンネル
鉄道	0	地上
3	1	地下・トンネル
河川湖沼	0	河川湖沼通常の河川
5	1	湖沼内の河川

・ライン項目レコード

レイヤー	コード	データ項目
行政界海岸線	1	都府県境
	2	北海道の支庁界
1	3	都市特別区界
	4	町村指定都市の区界
	5	海岸線
	9	図郭線
道路	1	高速道路及び自動車専用道
2	2	一般国道
	3	主要地方道
	4	一般都道府県道
鉄道	1	JR
3	2	公営鉄道
	3	民営鉄道
	9	未設定
河川湖沼	1	河川流路
5	2	湖沼の水涯線

・座標値レコード

項目	開始	終了	形式	内容
1点目のX座標	1	6	I5	ラインレコードに続くレコードで，ラインを構成する座標点の数NだけXY座標地のペアが記録される．座標値レコードの数は， (N－1)／7＋1 である(余りは切り捨て)．最後のレコードに空きが生じる場合0が埋められる．
1点目のX座標	6	10	I5	
1点目のX座標	11	15	I5	
1点目のX座標	16	20	I5	
:				
1点目のX座標	61	65	I5	
1点目のX座標	66	70	I5	
空白	71	72	X2	

例: 5776 443 5786 443 5785 456 5775 456 5776 443

・エリアレコード

例: A 113421 1 5780 450 1

・エリア構成ライン番号レコード

例: -1 0 0 0 0 0 0 0 0 0 0 0 0 0

・ポイントレコード

例: P 7 7 1 5780 451 0 1

・注記テキストレコード

例: 1 4 5780 451 0 沖ノ鳥島

４．４．２．データファイルの内容

データファイルの一部を以下に記す.

```
M 513100小月                    5    8   53    5    1  215
H2 1    8   12    5    0  103 9410 9503
N  1 1    1    0   83 1 3    1    6  -12    0    0    0    0    0    0
N  1 1    2   45    0 1 3   -1   -6    7    0    0    0    0    0    0
N  1 1    3  657    0 1 3    2   -7    8    0    0    0    0    0    0
N  1 2    4 5617 4283 0 3   -2    4    5    0    0    0    0    0    0
N  1 1    5 1983    0 1 3    3   -9   10    0    0    0    0    0    0
N  1 1    6 1964    0 1 3   -3   -8    9    0    0    0    0    0    0
N  1 1    7 9913    0 1 3   -4  -10   11    0    0    0    0    0    0
N  1 1    8 876410000 1 3   -5  -11   12    0    0    0    0    0    0
L  1 5    1      0    11    213520199999      5 0  0
0   83    2   78   56   45   43    2   45    0
L  1 5    2      0    31    403520199999    135 0  0
657    0  659    5  306  274  367  398  352  409  376  463  261  553
287  620  218  677  290  826  325  796  451 1066  416 1090  536 1347
497 1374  506 1393  415 1463  463 1577  446 1588  472 1647  422 1704
727 2189  853 2056  600 1630  649 1462 1264 1011 1881 2018 1659 2519
1193 2793 1165 2856 1193 2880 1215 2891 1286 2893 1317 2904 1336 2920
1366 2959 1371 2974 1377 2971 1387 2964 1402 2956 1418 2943 1425 2956
1429 2964 1426 2972 1419 2977 1412 2982 1404 2988 1396 3002 1432 3079
1441 3098 1458 3125 1430 3149 1528 3295 1648 3419 1702 3470 1672 3511
                              ・
                              ・
                              ・
```

これらのデータのうち，必要なものはレコード「M」に記されている島根県を示す第２次地域メッシュコードとレコード「Ｌ」の行の道路を表す「２」，さらに国道を表す「２」，つまり，「L2 2」を示していす個所である．そして次の行から記されている座標値は左から（Ｘ座標１［５バイト], Ｙ座標１［５バイト]），（Ｘ座標２［５バイト], Ｙ座標２［５バイト]），・・・の順に１行に最大７個記録されている．この個所をプログラムにて取得し，島根県内の各国道の座標値データファイルを生成した.

なお，詳細についてはＪＭＣマップCD-ROM 付属の説明書を参照のこと.

４．５．人口データ

４．５．１．データの記録方式

１．仕様

・レコード形式：　　固定長ブロック
・レコード長：　　２５３４バイト
・ブロックサイズ：　　１７７３８バイト
・文字コード：　　ＥＢＣＤＩＣ
・データのソート順：　　地域メッシュコードの昇順
・データの属性：　　キャラクター形式

２．コードの説明

・編成区分：

「３」・・・平成７年国勢調査第１次・２次・３次基本集計及び従業地・通学地集計に係る編成であることを示す.

・地域メッシュ区画区分：

「２」・・・第２次地域メッシュ区画の数値であることを示す．当該第２次地域メッシュ区画内の第３次地域メッシュ区画データの合算値である.

「３」・・・第３次地域メッシュ区画の数値であることを示す．当該第３次地域メッシュ区画が人口集中地区地域である場合は，当該第４次地域メッシュ区画データの合算値である.

「４」・・・人口集中地区地域における第４次地域メッシュ区画の数値であることを示す.

・市区町村数：

「９９」・・・当該地域メッシュ区画に同定された市区町村数が９以上となる場合を示す.

・秘匿・合算識別符号：

「X」・・・秘匿対象地域メッシュであることを示す.

「@」・・・合算地域メッシュであることを示す.

「Z」・・・２次メッシュについて，ひとつの他の２次メッシュに合算される場合の秘匿地域メッシュであることを示す

「W」・・・２次メッシュについて，複数の他の２次メッシュに合算される場合の秘匿地域メッシュであることを示す.

３．秘匿措置について

一つの地域メッシュに表彰される世帯総数又は人口総数が極めて少ない場合，秘匿措置として当該地域メッシュの数値は単独では表彰せず，近接する地域メッシュの数値に合算した上で表彰している．ただし，人口総数（ 総数，男，女），世帯総数（総数，一般世帯，施設等の世帯），年齢別人口（総数，男，女），性比及び年齢別人口の割合（総数，男，女）については，集計したままの原数値と，秘匿措置を講じた数値の両方を，表章している．データは秘匿措置を行なったデータをとり，世帯数が「X」の個所は0として取得した.

４．５．２．データファイルの内容

データは以下のように地域メッシュコードの昇順にならんでいる.

１．データの記録順序

・ ・ 第１次地域メッシュ区画データ（５２３２）
第２次地域メッシュ区画データ（５２３２００）
第３次地域メッシュ区画データ（５２３２００００）
第４次地域メッシュ区画データ（５２３２０００１） ・ ・ 第４次地域メッシュ区画データ（５２３２０００４）
第３次地域メッシュ区画データ（５２３２０００１）
第４次地域メッシュ区画データ（５２３２０００１１） ・ ・ 第４次地域メッシュ区画データ（５２３２０００１４）
第３次地域メッシュ区画データ（５２３２０００２） ・ ・ 第３次地域メッシュ区画データ（５２３２００９９）
第２次地域メッシュ区画データ（５２３２０１）
第３次地域メッシュ区画データ（５２３２０１００） ・ ・ 第３次地域メッシュ区画データ（５２３２９９９９）
第４次地域メッシュ区画データ（５２３２９９９９１） ・ ・ 第４次地域メッシュ区画データ（５２３２９９９９４）
第１次地域メッシュ区画データ（５２３３）
第２次地域メッシュ区画データ（５２３３００）
第３次地域メッシュ区画データ（５２３３０００００） ・ ・

次にデータのレコードを以下に記す.

（113 バイト目以降は「就業者数」，「学生数」等の人口データ）

２．データレコード

項目	開始バイト数	終了バイト数
調査名コード	1	2
調査年	3	6
ブランク	7	7
編成区分	8	8
ブランク	9	9
地域メッシュコード	10	18
ブランク	19	19
地域メッシュ区画区分	20	20
ブランク	21	22
第2次地域メッシュ区画の 1/25,000 地形図名 JIS 漢字12文字	23	46
ブランク	47	48
当該地域メッシュ区画に同定された都道府県・市区町村コード	49	90
ブランク	91	91
秘匿・合算識別符号	92	92
ブランク	93	93
合算先地域メッシュコード	94	102
ブランク	103	105
人口総数	106	112
・	・	・
・	・	・

実際のデータ例：

								（人口総数）	・・・
Z51995	3	53330293	3	母里	132206338	・・・		338	・・・
Z51995	3	533310	2	松江	43230632201	・・・		135374	・・・
Z51995	3	53331000	3	松江	132306	・・・		451	・・・
Z51995	3	53331003	3	松江	132201	・・・		68	・・・
Z51995	3	53331004	3	松江	132201	・・・		214	・・・
Z51995	3	53331007	3	松江	23220132305	・・・		547	・・・
Z51995	3	533310072	4	松江	23220132305	・・・		421	・・・

人口分布を視覚的に表示させるために生成する1km×1kmの四角形ポリゴン内にこれらのデータを格納することで指定した国道沿いの人口データを抽出すると言った検索を行うことが出来る.

４．６．座標変換

国道や人口分布を Google Earth 又は Google Maps 上で表示させるためにこれらの座標を世界測地系の座標系[11]に変換する必要がある．しかし，左右帯域幅 x m の国道沿い人口の検索を行うためには，メートル単位の計測が必要となるが，世界測地系など緯度経度座標で計測するのは困難である．このことから，緯度経度のまま計測するのではなく，実際に JMC マップに格納されている２次メッシュ単位の正規化座標を絶対座標化したもの用い，その座標に対し，Buffer メソッドを利用し，幅を持たせた Polygon 型を生成する．その後，世界測地系に変換するという方法で世界測地系の幅を持たせた Polygon 型の国道座標を生成する．

４．６．１．国道座標の絶対座標化

国道座標値は２次メッシュ単位に左下を（０，０），右上を（１００００，１００００）とする正規化された相対座標で格納されている．これにより，２つ以上の 2 次メッシュにまたがっている国道座標を検索対象とするには国道を絶対座標化する必要がある．

絶対座標化について図 4.6.1 のＡという２次メッシュファイルの代表点を用いて説明を行う．図 4.6.1 の左に示している単数の２次メッシュファイル（A）の代表点の 座標は，（５０００，５０００）であることが分かる．また，このＡが，図 4.6.1 の複数の２次メッシュファイルの中で存在しているときの代表点は，ファイルの書式形式上では，（５０００，５０００）という座標で記載されているが，このままの状態のままでは，複数の２次メッシュファイルから構成されている国道では，正確な位置を表すことができない．したがって，この国道の正確な座標をＢファイルの左下（０，０）を起点として，右上（３００００，２００００）の範囲内で表すと，その代表点は（２５０００，１５０００）という新たな座標に変換することが出来る．このような処理が絶対座標化である．JMC マップ内に記載されているすべての座標値はこの絶対座標化の処理を行っていないデータであるため，そのままの座標データを利用しようとしても，検索対象として扱うことが出来ないことになる．全ての座標値を正確な座標に変換する絶対座標化の処理を行うことで，検索対象として扱うことができるようになる事がわかる．

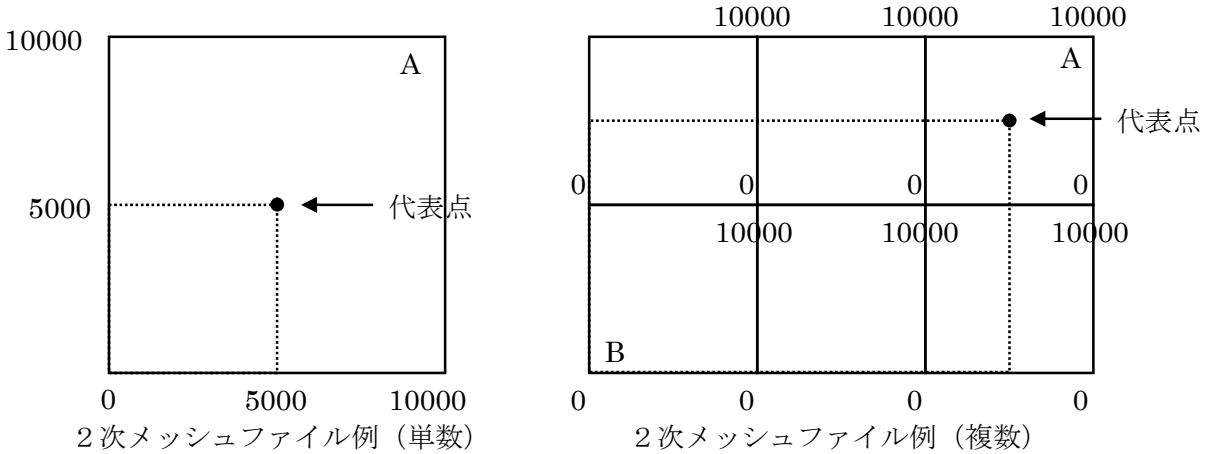


図 4.6.1　絶対座標化

１．２次メッシュ単位の絶対座標化について

図 4.6.2 は図 4.6.3 を「００」の２次メッシュファイルを基準に座標化したものである．全座標を１００００で割ったために，１次メッシュに含まれる座標値の範囲は（０，０）～（８，８）までとなる．ここで，２つの図を比較してみると，図 4.6.2 の各２次メッシュ番号と，図 4.6.3 の XY 座標値に，ある関係がみられる．これは，２次メッシュ番号の２桁を A，B として割り当てると，座標値が X 軸方向に増加するにつれて，B の値も増加し，Y 軸方向に増加するにつれて，A の値も増加し，２次メッシュの大元のデータを絶対座標化すると，

２次メッシュ X 座標　＝（(基本座標）／ １００００）＋ B
２次メッシュ Y 座標　＝（(基本座標）／ １００００）＋ A

となることがわかる．よって，以上のことから，上式を用いることで，２次メッシュの座標値を絶対座標化することが出来ることがわかる．

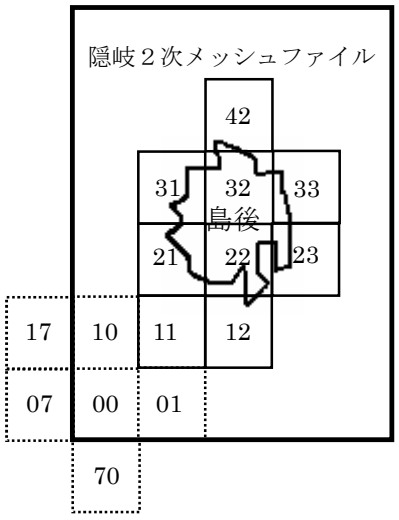


図 4.6.2　Ks5433　隠岐１次メッシュファイル

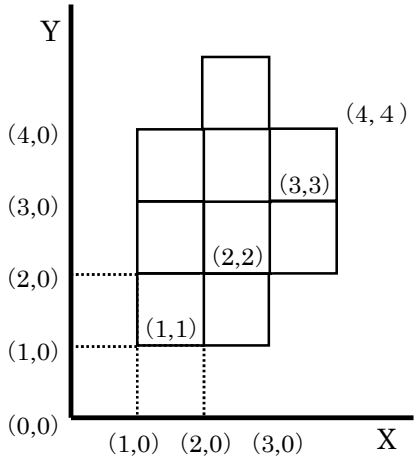


図 4.6.3　絶対座標化図（2 次メッシュ）

２．１次メッシュ単位の絶対座標化について

図 4.6.4 は１次メッシュの構成を示しており，図 4.6.5 はメッシュコードが「5231」の１次メッシュファイルを基準にして，絶対座標化したものである．これらにおいて，図 4.6.4 の１次メッシュ番号と図 4.6.5 の XY 座標との間に２次メッシュの時と同様に，ある関係が見られる．これは，図 4.6.4 の１次ファイル番号に，順にA，B，C，Dと割り当てていくと，X 軸方向にファイル数が増すごとに，C，Dの値も増加し，また Y 軸方向にファイル数が増すごとに，A，Bの値も増加していくことである．

図 4.6.4 では，ファイル１つの最大 XY 座標が８である．先ほどの X 座標ファイル番号C，Dは，D の場合，１毎に値が変わっていくため，8 ずつ座標値が変わっていき，Cの場合は，１０毎に値が変わっていくため，80 ずつ座標値も変わっていく．

これはＹ座標ファイル番号Ａ，Ｂについても同じことが言える．この関係を用い，１次メッシュの大元のデータを絶対座標化すると，

１次メッシュＸ座標　＝（（８×Ｄ）＋（８０×Ｃ））＋ ２次メッシュのＸ座標計算結果
１次メッシュＹ座標　＝（（８×Ｂ）＋（８０×Ａ））＋ ２次メッシュのＹ座標計算結果

となることがわかる．よって，以上のことから，２次メッシュと同様の方法である上式を用いることで，１次メッシュの座標値を絶対座標化することが出来ることがわかる．

5431	5432
5331	5332
5231	5232

図 4.6.4　１次メッシュ（複数）

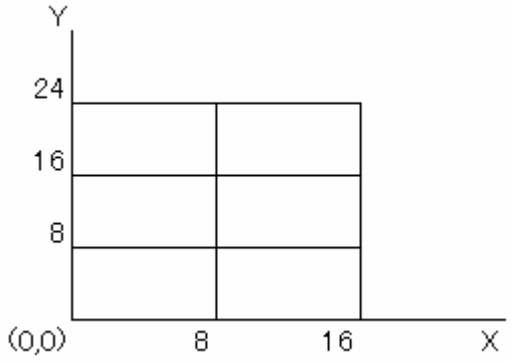


図 4.6.5　１次メッシュ絶対座標化

４．６．２．地域メッシュポリゴン

人口データを Google Earth 又は Google Maps 上で表示させるために 1km×1km 四角形の Polygon 型図形の世界測地系の緯度経度を求めなければならない．この図形は第３次メッシュの４隅の緯度経度を求めることで生成できる．

実際にこの緯度経度を求めるには，メッシュコードが左下の座標を示しているので，必要となる第３次メッシュコードとそれに隣接する３つのメッシュコードから求める必要がある．具体的には必要となるメッシュコードの上のメッシュコード，右上のメッシュコード，右のメッシュコードの３つのメッシュコードである．

３次メッシュメッシュコードから座標を求めるには，メッシュコードの構成を知る必要がある．これについては「４．３．地域メッシュコード」で述べように，３次メッシュコードというのは上４桁が １ 次メッシュコード，次の２桁が２次メッシュコード，残りの２桁が ３ 次メッシュコードである．１次メッシュコードというのは全国の地域を １ 度毎の経線と３分の２度毎の緯線によって分割したものである．２次メッシュコードというのは１次メッシュを８等分し，０～７の番号を付けたものである．つまり，３次メッシュコードというのは２次メッシュを１０等分し，南から０～９の番号を付けたものである．このことから，２次メッシュは緯度１２分の１度毎，経度８分の１度毎に分割したものであり，３次メッシュは緯度１２０分の１度毎，経度８０分の１度毎に分割したものだといえる．これらのことを踏まえて例を用いて説明する．

例：第３次メッシュコード「52404646」の４隅の緯度経度を求める．

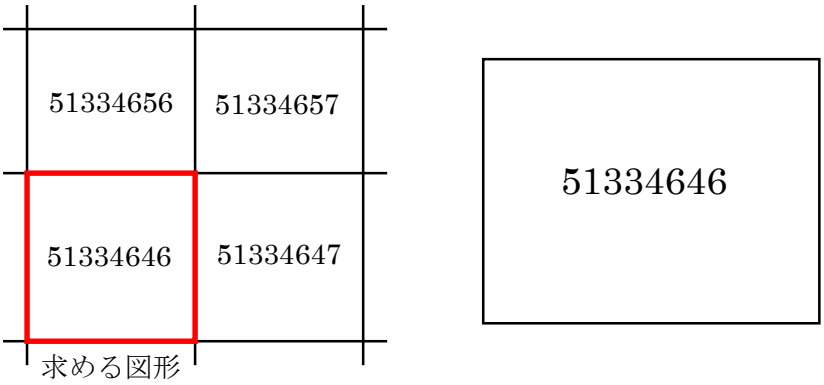


図 4.6.6　第３次メッシュの例

上４桁が１次メッシュコード，次の２桁が２次メッシュコード，残りの２桁が３次メッシュコードなので，それぞれ４隅の緯度経度は，

左下の座標：

緯度＝（５１×２／３）＋（４／１２）＋（４／１２０）＝　３４．３６６…

緯度＝（３３＋１００）＋（６／８）＋（６／８０）　＝１３３．８２５

上の座標　：

緯度＝（５１×２／３）＋（４／１２）＋（５／１２０）＝　３４．４０６…

緯度＝（３３＋１００）＋（６／８）＋（６／８０）　＝１３３．８２５

右上の座標：

緯度＝（５１×２／３）＋（４／１２）＋（５／１２０）＝　３４．４０６…

緯度＝（３３＋１００）＋（６／８）＋（７／８０）　＝１３３．８３７５

右の座標　：

緯度＝（５１×２／３）＋（４／１２）＋（４／１２０）＝　３４．３６６…

緯度＝（３３＋１００）＋（６／８）＋（７／８０）　＝１３３．８３７５

この方法を用い，３次メッシュコードを緯度経度座標に変換する．しかし，この緯度経度は日本測地系の緯度経度なので，世界測地系に変換する必要がある．これについては，後述する．

４．６．３．日本測地系から世界測地系への座標変換

日本測地系を世界測地系に変換するのには，３次メッシュコードと日本測地系の緯度経度が必要となる．国道座標の場合について説明する．国道座標は２次メッシュで格納されているので，これを３次メッシュに変換する必要がある．その変換方法は３次メッシュというのは，２次メッシュを１０等分に分割したものなので，２次メッシュ座標範囲（０，０）～（１００００，１００００）であることから，３次メッシュコードは２次メッシュコードに座標値の上から１桁目が縦の座標ならば７桁目，横の座標ならば８桁目を足したものになる．ただし，座標値が「１００００」の場合には上もしくは右の２次メッシュコードとなる．

この３次メッシュコードから「４．６．２．地域メッシュポリゴン」で述べた方法

により，日本測地系の３次メッシュコードの左下の緯度経度を求める．この求めた日本測地系の緯度経度を国土地理院測地部が配布している変換パラメータ（URL：http://vldb.gsi.go.jp/sokuchi/program.html）[12]を用いて変換する．このパラメータは１行毎に３次メッシュコードとその３次メッシュコードの左下の日本測地系緯度経度を世界測地系に変換する補正量が格納されている．

実際の変換方法は「４．６．２．地域メッシュポリゴン」と同様に，求める座標値を含む３次メッシュを調べ，そのメッシュの上，右上，右に隣接する３次メッシュのコードを含む４つのメッシュコードに対する補正量（注１）を，変換パラメータファイルをサーチして，求める（図参照）．補正量は左下の点の補正量なので，４つの３次メッシュコードが必要である．この値から必要な点を求める方法としてバイリニア補間法（Bilinear interpolation）（注２）を使用する．

人口メッシュ座標の場合は４隅の座標だけがわかればよいので，バイリニア補間法を使う必要はなく，変換パラメータを用い４点の座標を世界測地系に変換するだけでよい．

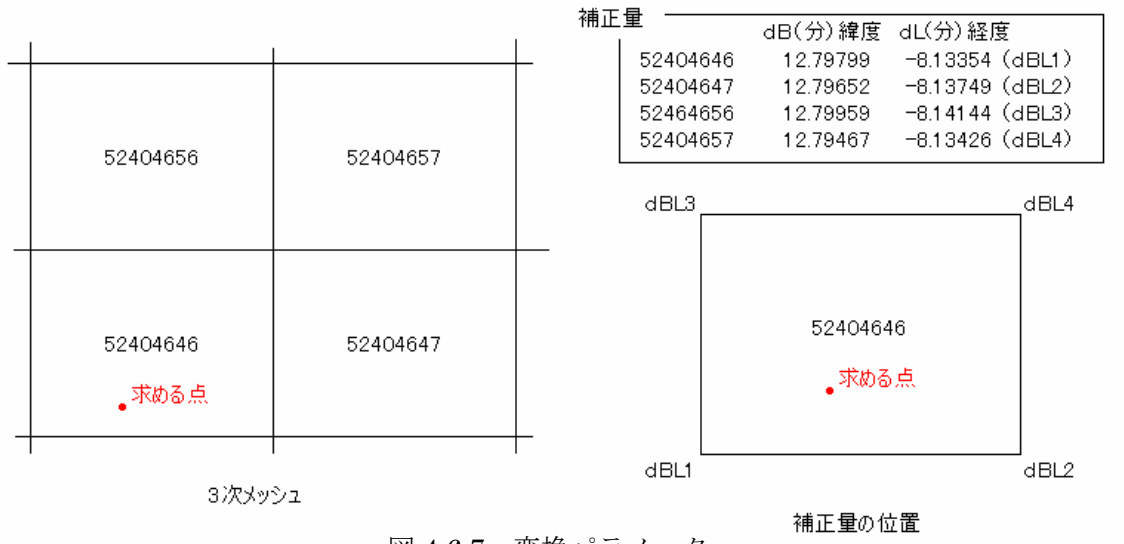


図 4.6.7　変換パラメータ

注１）補正量というのは旧座標値の準拠する楕円体で新座標値を計算し，その緯度差，経度差を求め，これらの格子点の座標差に旧座標値を加え，格子点の新座標値（ベッセル楕円体）を求め，楕円体高と合わせて世界測地系に変換し，この座標から旧座標値を引いた格子点の座標変換パラメータのことである．

注２）バイリニア補間法とは求める点の値を，周りの 4 つの格子点の値を 2 つのペアに分け，それぞれの線形補間を行い，さらに出てきた点で再び線形補間を行うことによって求める方法である．

具体的には，下図を使って説明する．この図において，Z00 は点（０，０），Z01 は点（０，１），Z10 は点（１，０），Z11 は点（１，１）での既知の値とし，点（x, y）での値 Z を求める．

最初に Z00 と Z10 を使って，点（x，0）での値 Z0 を次式で求める．

Z0 ＝ x × Z10 ＋ （1－x）× Z00

次に Z01 と Z11 を使って，点（x，1）での値 Z1 を次式で求めまる．

Z1 ＝ x × Z11 ＋ （1－x）× Z01

最後に Z0 と Z1 を使って，点（x，y）での値 Z を次式で求める．

Z ＝ y × Z1 ＋ （1－y）× Z0

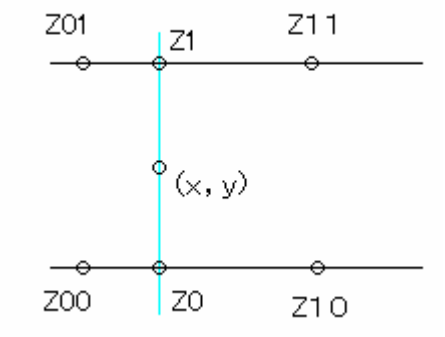


図 4.6.8　バイリニア補間法

４．７．DB の操作

国道沿いの人口を検索するために，それぞれの国道に対して Buffer メソッドを用い，幅を持たせた Polygon 型の国道座標を生成する．この生成した国道座標に対して，Intersects メソッドを用い，人口メッシュとの交差を判定し，「交差している」又は「完全に含まれている」ものという検索を行う．この検索の際にボトルネックとなるのが，Buffer メソッドである．このボトルネックを解決するために，幅を持たせた Polygon 型を検索要求があるたびに生成するのではなく，前もって生成しておくことで大幅な時間短縮を行うことが出来た．このことについては「４．９．計測」で説明する．また，DBMS の最適化についても後述する．

４．７．１．表の定義，データの格納

格納するデータは国道座標値データ，Polygon 型に変換した国道座標値データ，人口データ，基準地域メッシュのポリゴン座標値データである．Polygon 型に変換した国道座標値データの生成方法については後述する．

まず，これらの空間データを格納するには，ユーザ定義型の列を含む表を生成する必要がある．

・国道座標を格納する表「KOKUDOU」の生成

この表には，データ ID，国道名，国道座標（世界測地系）を格納する．

```
CREATE TABLE    KOKUDOU (
    ID              INTEGER,
    NAME            VARCHAR(100),
    GEO_CLMN        GEOMETRY
);
```

・人口データを格納する表「JINKOU」の生成

この表には，３次メッシュコード，総人口数，老齢者数，就業者数，世帯数，学生数，人口メッシュの座標（世界測地系）を格納する．

```
CREATE TABLE    JINKOU (
    MESH_CODE    INTEGER,
    JINKOU       INTEGER,
    OLDER        INTEGER,
    WORKER       INTEGER,
    FAMILY       INTEGER,
    STUDENT      INTEGER,
    GEO_CLMN     GEOMETRY
);
```

座標を格納する列の型名に GEOMETRY 型を指定している．GEOMETRY 型とは HiRDB 空間検索プラグイン(HiRDB Spatial Search Plug-in Version 3)によって定義されているユーザ定義型で，Point 型，Line String 型，Polygon 型といった空間データを格納することができる．国道座標の表については帯域幅毎に別の表を生成する．

次に空間座標データ格納方法について空間座標データは抽象定義型である GEOMETRY 型の列に格納するので，データの格納にはコンストラクタ関数を使用する．

・国道座標データ格納例

```
INSERT INTO KOKUDOU ( ID, NAME, GEO_CLMN)
    VALUES( 8, 'R9', GeomFromText('LINESTRING(60421 -50000,60570
            -50386,61473 -51420,61643 -51706, ･･･ , 186815 -171489,1
            86916 -171608)',9,1));
```

表「KOKUDOU」

ID	NAME	GEO_CLMN
8	R9	LineString(60421 -50000, ･･･　　)

・人口データ格納例

```
INSERT INTO JINKOU(MESH_CODE,JINKOU,OLDER,WORKER,FAMI
                        LY,STUDENT,GEO_CLMN)
    VALUES( 51313688, 8, 5, 0, 0, 0, GeomFromText('POLYGON((6800
            0 -38000,68000 -39000,69000 -39000,69000 -38000))',1,1));
```

表「JINKOU」

MESH_CODE	･･･	GEO_CLMN
51313688	･･･	Polygon((68000 -38000, ･･･))

GeomFromText 関数が GEOMETRY 型のコンストラクタ関数である．この関数を使用すると WKT 形式のあらゆる空間データを GEOMETRY 型データ列に格納することができる．この関数についての詳細は文献[13]を参照.

・空間インデックス生成方法

空間座標データを操作するには索引付けを行う必要がある．空間検索プラグインにおける空間インデックス生成例を以下に示す.

```
国道座標データへの空間インデックス作成
> CREATE INDEX  GEOIDX
    using type spatial on KOKUDOU（ GEO_CLMN ）
    in（ RLOB1 ）
    PLUGIN 'EXTENT=100,-174810,186916,300;
                DEPTH=1;PRECISION=1';

人口座標データへの空間インデックス作成
> CREATE INDEX  GEOIDXJ2
    using type spatial on JINKOU（ GEO_CLMN ）
    in（ RLOB2 ）
    PLUGIN 'EXTENT=54000,-280000,191000,-38000;
                DEPTH=4;PRECISION=1';
```

ここでは国道座標データへの空間インデックス生成について説明する．CREATE INDEX 文がインデックスを生成する SQL 文である. ここでは GEOIDX という名前で, 表「KOKUDOU」の GEO_CLMN 列（Geometry 型）への空間インデックスを生成している. HiRDB では検索に Quad-Tree を用いている. この例では, 深さ 1（DEPTH=1）の整数座標系（PRECISION=1）としている.

これらに関する詳細は[11]を参照．インデクスについては後述する.

・空間座標データ検索方法

空間座標の検索も一般的な SELECT 文を用いて行う. 座標データを抽出したい場合, AsText()関数を使用すると，空間座標データを WKT として返す．また，WHERE 句に条件を指定することも可能である．以下にその例を示す.

・国道座標を WKT 形式で検索する例

国道名が‘R54’である国道の座標を抽出する

```
> SELECT AsText(GEO_CLMN,1) FROM KOKUDOU
      WHERE NAME='R54' WITHOUT LOCK NOWAIT;

出力結果
linestring(140000 -111658,139458 -112439, ･･･,152496 -168087,152419 -168393)
```

この例では，国道座標データは GEO_CLMN 列に格納されている．AsText 関数によって，国道名が'R54'と一致するレコードの国道座標データを WKT 形式で出力している．

・国道沿いの人口を検索する例

国道座標を拡張（Buffer()）し，その範囲内（Intersects()）の人口データを抽出する．

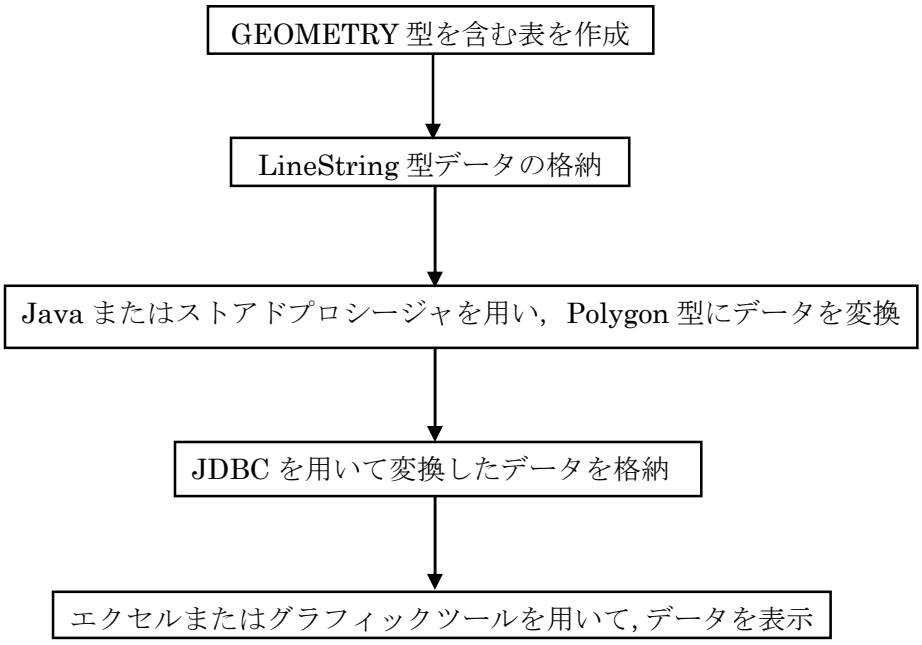
```
> SELECT jinkou FROM JINKOU
      WHERE IntersectsIn(GEO_CLMN, RegionFromText(:R_GEOM, 500)
IS TRUE WITHOUT LOCK NOWAIT;
```

GEO_CLMN 列に人口メッシュ（Polygon），変数:R_GEOM は幅を持たせた国道座標（Polygon）が格納されているものとする．

この拡張した国道座標と人口メッシュ（GEO_CLMN）の共通集合演算をおこない，国道沿いの圏内に含まれていればその人口数を抽出する．

４．７．２．Line String 型から Polygon 型を生成

Line String 型から Polygon 型への変換手順について示す．



１．Polygon 型への変換方法

変換方法にはストアドプロシージャ，Java，埋め込み SQL の 3 種類がある．ここでは生成の比較的容易なストアドプロシージャと Java を用いた方法を説明する．

・ストアドプロシージャ

ストアドプロシージャとはデータベースへの一連の操作をプログラムとして DBMS に定義したものである. ストアドプロシージャのメリットとしては DBMS にプログラムを定義することにより，複数のアプリケーションプログラムから呼び出して使用することが出来ます. デメリットは Java のようにプログラムを実行するだけでなく，登録などをしなくてはならないことである．

・ストアドプロシージャの生成・登録

ストアドプロシージャは「CREATE PROCEDURE 文」を用いて生成し，登録することが出来る．Line String 型から RegionBuffer 関数を用いて Polygon 型を生成するプログラムを以下に示す．

例：LineString 型を左右帯域 500m に拡張した Polygon 型に変換する．

```
CREATE  PROCEDURE  RegionBuffer_proc (OUT  geodata  VARCHAR(32000))
        BEGIN
                DECLARE  str  BLOB(32000);
                DECLARE  SearchRegion  BLOB(32000);

                SELECT  AsBinary(GEO_CLMN,0,2,0)  INTO  str  FROM
                                KOKUDOUBP  WHERE  NAME  =  'R54';

                SET  SearchRegion  =
                                RegionBuffer(LineFromWKB(str,0,1),500);

                INSERT  INTO  KOKUDOUB(ID,  NAME,  GEO_CLMNB)
                        VALUES(500,'R54',GeomFromWKB(SearchRegion,9,1));

               SELECT  AsText(GEO_CLMNB,1)  INTO  geodata  FROM
                                KOKUDOUB  Where  NAME  =  'R54';
        END
```

まず，R54 という名前の LineString 型の座標を Binary 型で抽出し，変数 str に代入する. 次に，RegionBuffer(LineFromWKB(str,0,1),500)により，左右帯域 500m に拡張した Polygon 型のデータを変数 SerchRegion に代入．INSERT INTO KOKUDOUB(ID, NAME, GEO_CLMNB) VALUES(500,'R54',GeomFromWKB(SearchRegion ,9,1))により，先述した「KOKUDOU」という表と同じ定義の表である「KOKUDOUB」という表に一時的に格納し，SELECT AsText(GEO_CLMNB,1) INTO geodata FROM KOKUDOUB Where NAME = 'R54'で Polygon 型のデータを

抽出し，出力変数 geodata に代入，出力するものである．

・ストアドプロシージャの呼び出し

ストアドプロシージャの呼び出しには，埋め込み SQL を使った方法などがあるが，今回は開発の容易さから Java で呼び出すことにする．Java でデータベースにアクセスするには JDBC を用いた．以下に先ほどのプログラムを呼び出すのに必要なコードを示す．

ストアドプロシージャを呼び出すコード

```
CallableStatement cst    =    con.prepareCall("{call RegionBuffer_proc(?)}");
cst.registerOutParameter(1, java.sql.Types.LONGVARBINARY);
cst.execute();


Blob b = cst.getBlob(1);
```

このコードはストアドプロシージャを呼び出し，抽出した Polygon 型のデータを受け取るコードである．これにより，ストアドプロシージャを用いて LineString 型のデータを Polygon 型に変換したデータを取得することが出来る．

２．データの簡易表示方法

データを作成した後にはそのデータが間違っていないか視覚化させ確認する必要がある．視覚化する方法の例を挙げると，取得したデータを Java で JPG に変換し２D で出力する方法，X３D や KML を用い，３D で出力する方法などさまざまある．しかし，今回は確認のためだけなのでもっとも容易に出力できると考えられる Excel のグラフ機能を用いた方法でデータを視覚化させる．

・Excel を用いたデータの視覚化

例：R54 を帯域幅 500，100，50 で Polygon 型に変換したものを表示

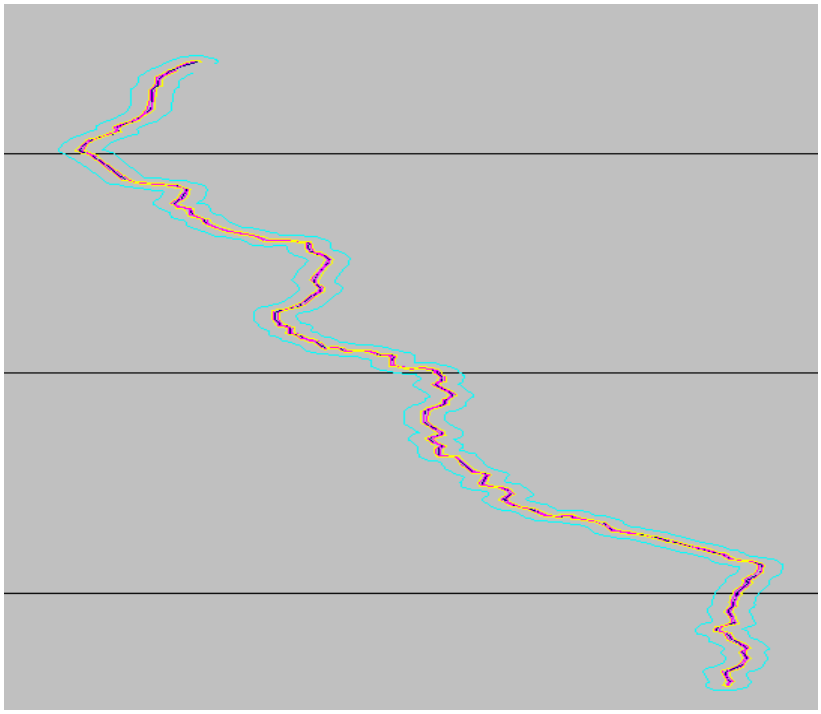


図 4.7.1　Polygon 型の座標を Excel で表示

４．７．３．LineString 型から Polygon 型に変換した際の点数の変化

Line String 型から Polygon 型に変換したの際の点数の変化を考察することにより，Polygon 型作成までにかかる時間の推測を行うことが出来る．ここではその推測を行うためのデータとそれについての考察について述べる．

１．LineString 型から Polygon 型に変換した際の点数についての比較

Polygon 型に変換するデータを島根県の国道（R186，R187，R191，R261，R314，R488，R54，R9）とし，左右帯域 500m とした場合の Polygon 型へ変換した際の座標の点数についての表 4.7.1 と図 4.7.2 を下記に示す．

表 4.7.1　各国道の左右帯域 0 と 500 の点数

国道名	帯域 0	帯域 500
R186	73	272
R187	92	355
R191	158	530
R261	143	523
R314	117	439
R488	169	540
R54	153	547
R9	475	1682

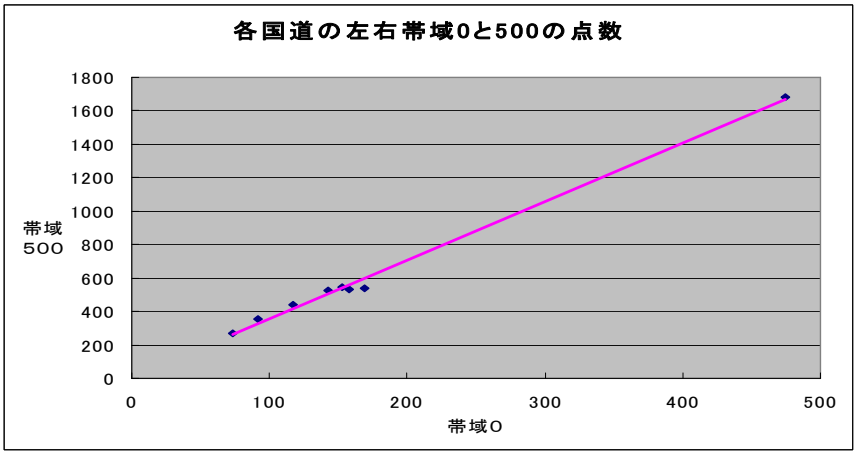


図 4.7.2　各国道の左右帯域 0（Line String 型の場合）と 500 の点数

この結果から分かるようにどのような LineString 型の座標でも帯域 0 と 500 の点数の関係はほぼ比例になっていることが言える．このことから，複雑な形の LineString 型の図形でも単純な形の Line String 型の図形でも座標点数の増加率は変わらないことが言え，座標点数さえわかれば点数の推測を行うことが出来る．

２．Polygon 型の帯域についての比較

図 4.7.3 は Line String 型のデータを R54 とし，左右帯域幅を外から 10000m，500m，100m，50mの場合を表示した図である．

また，その下の図 4.7.4 は図 4.7.3 と同様に Line String 型のデータは R54 とし，左右帯域幅を外から 500m，100m，50mの場合を表示した図である．

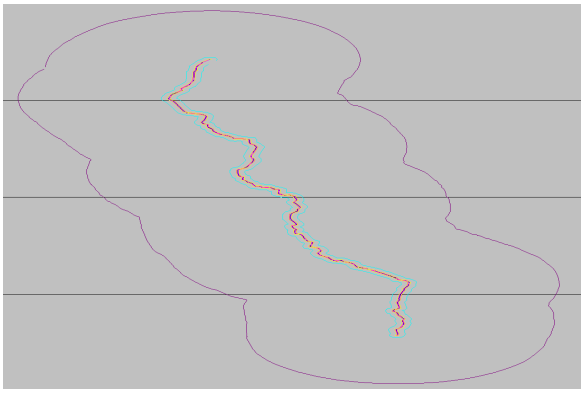


図 4.7.3　帯域 10000，500，100，50 の場合の Polygon 型

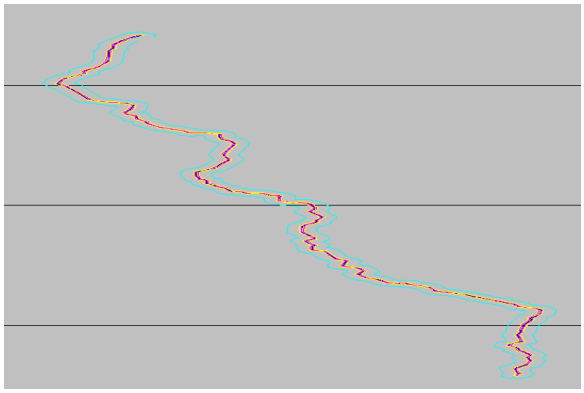


図 4.7.4　帯域 500，100，50 の場合の Polygon 型

この結果から帯域を大きくすると，図が簡略化されていくことがいえる．これについてさらに帯域と点数の関係について R54 の場合で考察する．下記に帯域と点数についての表 4.7.2 とその図 4.7.5 を示す．

表 4.7.2　R54 の点数(153)

帯域	点数
50	592
100	590
500	547
600	524
700	503
800	496
900	478
1000	459
2000	345
3000	277
4000	231
5000	211
6000	197
7000	180
8000	172
9000	161
10000	159

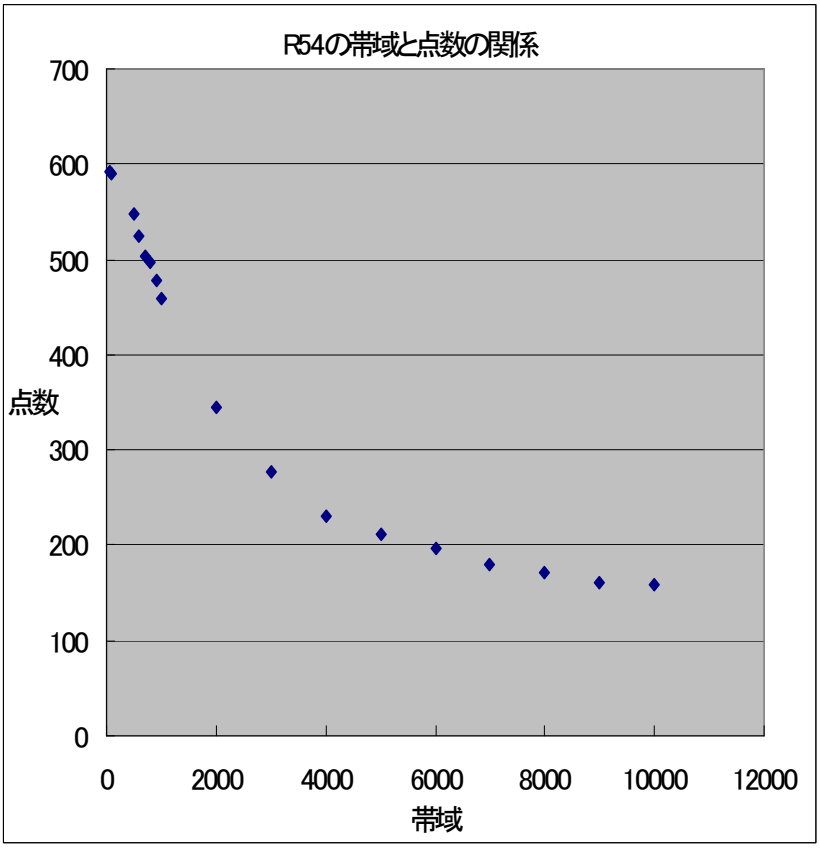


図 4.7.5　R54 の帯域と点数との関係

この結果から帯域が大きくなるにつれて点数が減っていっているのがわかる．これは図が簡略化されていることが要因といえる．

次に点数の最も多いもの（R9 : 475）と最も少ないもの（R186 : 73）を比べてみる．

表 4.7.3　R9 の点数

帯域	点数
50	352
100	343
500	272
1000	210
5000	97
10000	71

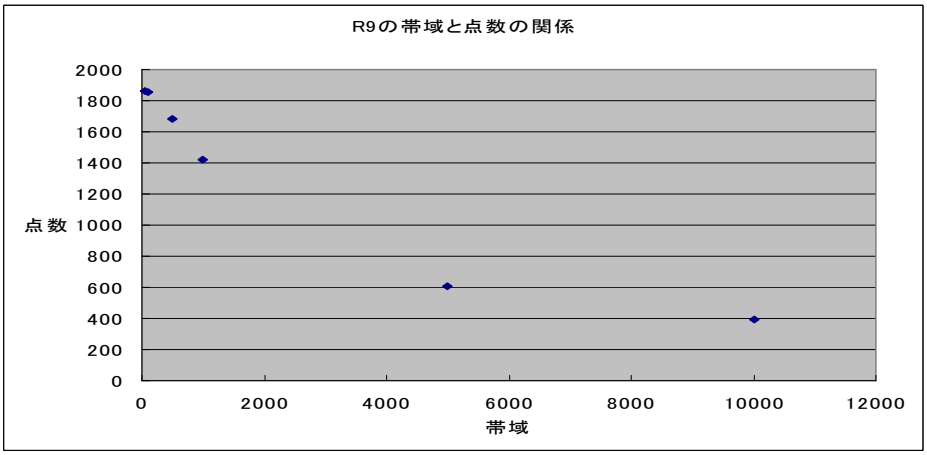


図 4.7.6　R9 の帯域と点数の関係

表 4.7.4　R186 の点数

帯域	点数
50	1859
100	1856
500	1682
1000	1423
5000	608
10000	394

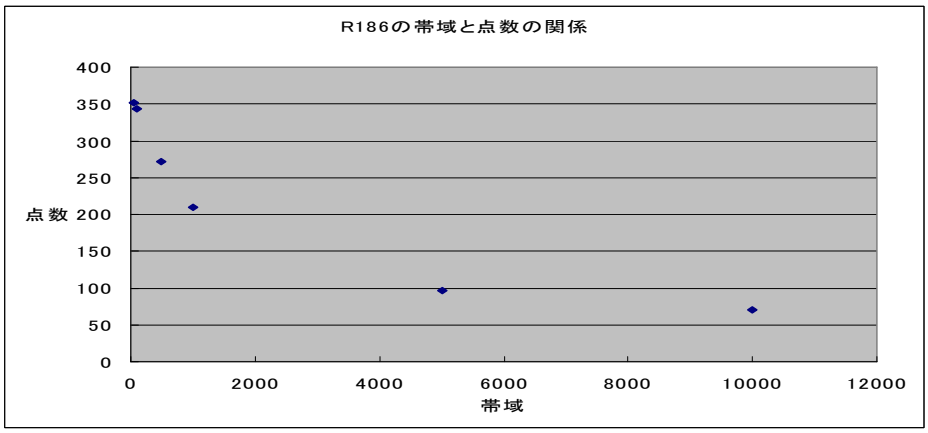


図 4.7.7　186 の帯域と点数の関係

この結果から点数の多い少ないにかかわらず，似たような結果になる．このことから，図の簡略化が点数には影響されないことわかる．

３．考察

これらの結果から LineString 型を Polygon 型に変換したときの点数の増加は弧の部分によって決まるといえる．例えば，帯域を大きくした場合，この部分でとる点数を減らしている．これにより，帯域を大きくした場合は図が簡略化されてしまうといったことが生じる．

よって弧の部分の点の取り方に応じて図の詳細さが決まる．つまり，詳細にしたい図の場合にはこの部分の点を増やし，簡略化して良い場合はこの部分の点数を減らせば良いといえる．

４．７．４．DB の最適化

DB の最適化には下記の項目の設定が必要である．最適化の例は上記で示した KOKUDOU と JINKOU という表を用いる．() の内は計測結果の列名である．

・PDISLLVL（①）

データ保障レベルで０のときは排他待ちをしないで検索できる．２のときはトランザクションが終了するまで検索できない.

・PDIPC（②）

サーバとクライアントが同一ホストにある場合，プロセス間の通信方法を指定する. MEMORY を指定した場合, プロセス間メモリ通信を行う. DEFAULT を指定した場合，プロセス間の通信に，各プラットフォームでのデフォルトの通信方式を用いる.

・PDUAPREPLVL（③）

a のときすべての情報が出力される．省略したとき，SQL トレース情報だけ出力される.

・PDSQLTRACE（④）

SQL トレースファイルのサイズを指定した場合，UAP の SQL トレースを出力する．省略した場合は SQL トレースを出力しない.

・RDDATA（⑤）

JINKOU 表と KOKUDOU 表が格納されている RD エリアのグローバルバッファの面数

・RDINDX（⑥）

JINKOU 表と KOKUDOU 表のインデクスが格納されている RD エリアのグローバルバッファの面数

・INDEX（⑦）

人口メッシュコードと国道 ID の INDEX

・空間 INDEX 国道（⑧）

KOKUDOU 表の空間インデクスの深さ

・空間 INDEX 人口（⑨）

JINKOU 表の空間インデクスの深さ

・RLOBK（⑩）

KOKUDOU 表の空間インデクスが格納されている RD エリアのグローバルバッファの面数

・RLOBJ（⑪）

JINKOU 表の空間インデクスが格納されている RD エリアのグローバルバッファの面数

この項目を最適化した場合の結果を表 4.7.5 に示す. 方法は国道 54 の左右帯域幅 500 mの Polygon 型への変換時間とその図形と人口メッシュとの「交差」又は「完全に含まれているか」の検索時間を足した時間を計測する．人口の種類は総人口とする.

表 4.7.5　R54 帯域幅 500m 人口の種類：総人口とした場合の最適化に関する表

①	②	③	④	⑤	⑥	⑦	⑧	⑨	⑩	⑪	計測時間
2	DEFAULT	a	10000	400	200	なし	なし	なし	なし	なし	15.093
0	DEFAULT	a	10000	400	200	なし	なし	なし	なし	なし	15
2	MEMORY	a	10000	400	200	なし	なし	なし	なし	なし	14.953
2	DEFAULT	省略	10000	400	200	なし	なし	なし	なし	なし	10.921
2	DEFAULT	a	省略	400	200	なし	なし	なし	なし	なし	6.75
2	DEFAULT	a	10000	1000	1000	なし	なし	なし	なし	なし	15.75
2	DEFAULT	a	10000	400	200	あり	なし	なし	なし	なし	15.484
2	DEFAULT	a	10000	400	200	あり	14	なし	100	100	15.041
2	DEFAULT	a	10000	400	200	あり	8	なし	100	100	15.078
2	DEFAULT	a	10000	400	200	あり	1	なし	100	100	15.031
2	DEFAULT	a	10000	400	200	あり	なし	14	100	100	計測不能
2	DEFAULT	a	10000	400	200	あり	なし	10	100	100	53.953
2	DEFAULT	a	10000	400	200	あり	なし	9	100	100	21.718
2	DEFAULT	a	10000	400	200	あり	なし	8	100	100	14.359
2	DEFAULT	a	10000	400	200	あり	なし	7	100	100	12.75
2	DEFAULT	a	10000	400	200	あり	なし	6	100	100	13.031
2	DEFAULT	a	10000	400	200	あり	なし	5	100	100	13.375
0	MEMORY	省略	省略	1000	1000	あり	1	7	1000	1000	4.56

①PDISLLVL について変化がなかった理由は今回の実験では同じ行に同時にアクセスすることがなかったからである．

②PDIPC について変化がなかった理由はクライアントとサーバが同一ホストでなかったからである．

③，④PDUAPREPLVL と PDSQLTRACE について速くなった理由は今まで出力していたデータを出力しなくなるからである．

⑤，⑥RDDATA と RDINDX について変化がなかった理由は４００と２００の時点である程度の性能が出ていたからである．

⑦INDEX について変化がなかった理由は検索条件として INDEX 付けした項目を使っていなかったからである．

⑧空間 INDEX 国道について変化がなかった理由は下図のように深さを深くしてもはみ出た場合はレベルを上げるのでもともと大きい今回の POLYGON では意味がないからである．

⑨空間 INDEX 人口について深さが 7 のときがもっとも速かった．これは図 4.7.8 で示しているように深さを深くしてもはみ出した場合はレベルを上げるので深くしすぎても逆にオーバーヘッドが大きくなり性能が落ちてしまうからである．

⑩，⑪RLOBK と RLOBJ について変化がなかったのは RDDATA と RDINDX と同様に 100 の時点である程度の性能が出ていたからである．

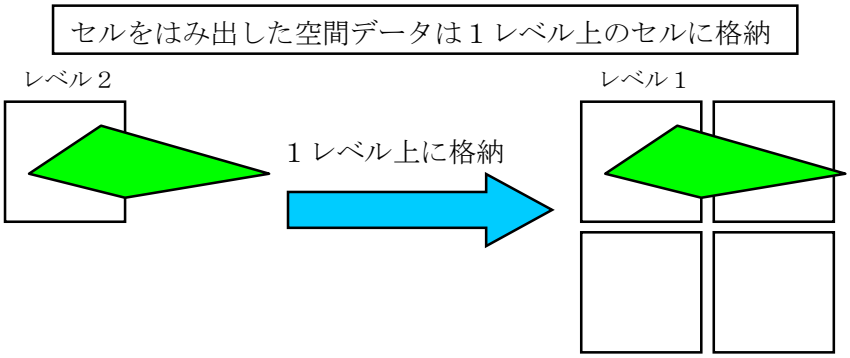


図 4.7.8　空間インデクスのセルがはみ出した場合の動作

４．８．KML

KML（Keyhole Markup Language）とは Google Earth や Google Maps などで地理情報を表示するために使われているファイルフォーマットであり，KML はタグベース構造で XML 基にしている．さらに，KML を ZIP 圧縮することで KMZ となる．

KML は HTML や XML と似たような方法で処理されていので, Google Earth や Maps は KML のブラウザの機能を果たす.

４．８．１．KML の生成

DB から受け取ってきたデータを国道座標は<LineString>タグを用いて表示し, 人口メッシュデータに関しては<Polygon>タグを用いて表示する. 人口メッシュデータに関しては人口数ごとに高さと色を変更している.

実際の KML データについて説明する.

例：R186 の一部

```
<Placemark>
    <name>LineString</name>
    <styleUrl>#black2</styleUrl>
    <LineString>
      <tessellate>1</tessellate>
      <coordinates>
        132.217193,34.753199
        132.213768,34.755365
                  :
        132.078776,34.896174
       132.078601,34.899958
      </coordinates>
    </LineString>
</Placemark>
```

<Placemark>タグを用いることで，位置情報だけではなく，名称や表示スタイルも指定することが出来る．<LineString>タグ内では<tessellate>タグにより，地表にフィットさせることが出来，<coordinates>タグにより，頂点の位置（世界測地系の緯度経度）を決定する．これにより，国道座標を Google Earth に表示させることが出来る．

例：人口メッシュ座標の一部

```
<Placemark>
  <visibility>1</visibility>
  <name>Population96</name>
  <styleUrl>#blue2</styleUrl>
  <Polygon>
    <extrude>1</extrude>
    <altitudeMode>relativeToGround</altitudeMode>
    <outerBoundaryIs>
      <LinearRing>
        <coordinates>
        132.184994,34.778196,96
        132.184994,34.786528,96
        132.197493,34.786529,96
        132.197493,34.778196,96
        132.184994,34.778196,96
        </coordinates>
      </LinearRing>
    </outerBoundaryIs>
  </Polygon>
</Placemark>
```

国道座標と同様に<Placemark>タグを用い，位置情報や名称，表示スタイル，表示のＯＮ／ＯＦＦを指定する．<Polygon>タグ内では，<extrude>により，棒グラフを生成し，<altitudeMode>タグにより，標高を地表面からの高度として扱う．<outerBoundaryIs>タグで外枠であることを指定し，<LinearRing>ににより，閉じた線分であることを指定する．<coordinates>タグでは，国道座標と同様に位置（世界測地系の緯度経度）と標高を指定する．

４．８．２．Google Maps API

Google Maps に多角形や折線を表示する場合，KML を用いた方法も使用できるが，この方法を用いた場合，KML を Google Maps API 変換し，表示させるため，処理に冗長が生じてしまう．これらの理由により，Google Maps では KML ではなく，直接に Google Maps API を用いることとする．

・Google Maps API の生成

例：R186 の一部

```
var pointsR186 = [];
pointsR186[0] = new GLatLng(34.753199,132.217193);
pointsR186[1] = new GLatLng(34.755365,132.213768);
                    :
pointsR186[71] = new GLatLng(34.896174,132.078776);
pointsR186[72] = new GLatLng(34.899958,132.078601);
var polyline = new GPolyline(pointsR186,"#000000", 1, 1);
map.addOverlay(polyline);
```

ここでは，pointsR186 という配列を生成し，この配列に国道の位置（世界測地系の緯度経度）を格納し，GPolyline クラスで線の色，透明度を設定し，addOverlay メソッドで表示する．

例：人口メッシュデータの一部

```
var pointsj = [];
pointsj[0] = new GLatLng(34.778196,132.184994);
pointsj[1] = new GLatLng(34.786528,132.184994);
pointsj[2] = new GLatLng(34.786529,132.197493);
pointsj[3] = new GLatLng(34.778196,132.197493);
pointsj[4] = new GLatLng(34.778196,132.184994);
var polyline = new GPolygon(pointsj,"#000000", 1, 1,"#0000FF", 0.6);
map.addOverlay(polyline);
```

ここでは，国道座標と同様に pointsj という配列を生成し，この配列に人口メッシュデータの置（世界測地系の緯度経度）を格納し，GPolygon クラスで線の色，線の透明度, 塗りつぶしの色, 塗りつぶしの透明度を設定し, addOverlay メソッドで表示する．

４．９．計測と性能評価

システム全体のボトルネックとなっている箇所が Buffer メソッドであるかを特定するためにシステム全体の細かな処理時間を計測する．まず，システム全体のデータ・処理の流れについて説明する．

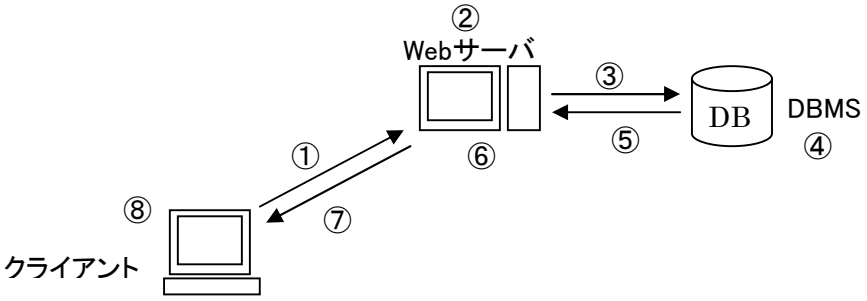


図 4.9.1　システム全体の流れ

① クライアントの Web ブラウザ上から Web サーバにリクエストを送信する．
② Web サーバがリクエストを受信すると，サーブレットに処理が渡される．
③ サーブレットは必要なデータ（座標，人口など）を得るために DBMS へ接続する．
④ DBMS 内部で必要なデータを検索する．
⑤ 検索したデータをサーブレットへ渡す．
⑥ サーブレットは受け取ったデータをもとに，KML 又は Google Maps API を生成する．
⑦ 検索結果である KML 又は Google Maps API を Web ブラウザへ送信する．
⑧ KML の場合は，Web ブラウザは自動実行により，Google Earth を実行し，結果を表示する．Google Maps API の場合は，Web ブラウザは Google Maps にアクセスし，結果 Google Maps 上に表示する．

①と⑦ではクライアントの Web ブラウザーサーバ間の通信である．①では国道，人口の種類，帯域幅などのデータを送信し，⑦では KML や Google Maps API のデータを送信している．ここでは，大きなデータの送信はしていない．

②と⑥では同じサーブレットが処理を行っている．ここで行っていることは，リクエストに含まれるパラメータ（国道，人口の種類，帯域幅など）を JDBC を用いて，国道および人口を検索する SQL 文（SELECT 文）を作成し，DB にアクセスして実行する．ここでの主な処理は KML 又は Google Maps API を生成することである．

③と⑤では Web サーバーDB サーバ間の通信である．③では生成した SQL 文を送信し，⑤では検索結果を送信している．ここでは，大きなデータの送信はしていない．

④の箇所では，主に２つのデータの検索を行っている．１つは，指定した国道の空間データ（LineString）の検索である．この検索は，国道名もとに格納された空間データを AsText 関数で WKT 形式に返す，というものである．２つは，人口の検索である．１つ目で検索した国道座標（LineString）に Buffer メソッドを用い，帯域幅をもたせ Polygon に変換し，その Polygon と全ての人口メッシュ（Polygon）との共通集合演算

（intersection()）を行う．その結果 TRUE となる人口データを返す，というものである．今回の人口データのレコード数は 3894 である．この箇所がボトルネックとなった場合，Intersects メソッドと Buffer メソッドが実行されているので，そのどちらが，ボトルネックとなっているか特定しなければならない．そのため，事前に Buffer メソッドを実行した場合とそうでない場合での処理時間の計測を行う必要がある．

ここでは，Buffer メソッド以外にボトルネックがあることも考えられるので，システム全体の処理時間①～⑥間の各箇所において計測を行うことにする．⑧については Google Earth 又は Google Maps に依存するので計測が困難であることから，本研究では，この計測を省くこととする．

４．９．１．計測方法

①～⑥で実際に測定した箇所について，具体的に説明する．

計測した箇所は

１．Web ブラウザ→Web サーバ間の通信時間（①の部分）
２．国道検索 SQL の発行時間（③～⑤の部分）
３．国道検索 SQL の実行時間（③～⑤の部分）
４．人口検索 SQL の発行時間（③～⑤の部分）
５．人口検索 SQL の実行時間（③～⑤の部分）
６．サーブレット全体の処理時間（①～⑥の部分）

の計６箇所である．

１．Web ブラウザから Web サーバまでの通信時間

まず，①でクライアントからリクエストを送信してから Web サーバで受信するまでの時間を計測する．リクエストの送信は，HTML を用いた送信フォームを用いている．ここでの送信時に，送信ボタンが押されたときの時刻を送信データに含めて送る．<FORM>エレメント（フォーム名：form1）の中に次のコードを記述する．

```
       :
01：<INPUT type="hidden" name="SendTime">
02：<INPUT type="hidden" name="TZOffset">
03：<SCRIPT language="JavaScript">
04：  <!−
05：  function nowTimeSet(){
06：    now = new Date;
07：    form1.SendTime.value=now.getTime();
08：    form1.TZOffset.value=now.getTimezoneOffset();
09：  }
10：  -->
11：</SCRIPT>
      :
12：<INPUT onClick="return nowTimeSet(form1)" type="submit" value="送信">
      :
```

01：と 02：は送信時刻とその時差を格納するフィールドである．これらは type=”hidden”とすることで非表示としている．03：～11：は 01：02：で用意したフィールド SendTime，TZOffset にそれぞれ時刻と時差を入力するための関数 nowTimeSet()（Java Script で記述）である．12:が送信ボタンの INPUT タグであるが，クリック時（onClick）に nowTimeSet()関数を呼び出し，時刻を入力後，リクエストを送信する．

送信されたデータは Web サーバが受信し，サーブレットが呼び出される．このサーブレットでは次のようにして送信データを受信する．

```
        :
13：java.util.Date rsvTime=new Date();//時刻を取得
        :
14：//クライアントの送信時刻(GMT,ミリ秒)
15：Long sendTime=Long.valueOf(req.getParameter("SendTime"));
16：Long ctTZ=Long.decode(req.getParameter("TZOffset"));
17：//eTime1:クライアント→Web サーバの通信時間
18：long eTime1=(rsvTime.getTime())+(rsvTime.getTimezoneOffset()*6000)
                          -(sendTime.longValue()+ctTZ.longValue()*6000);
        :
```

13：はサーブレット実行開始時に時刻を記録しておく．15：16：で HTML フォームからのフィールド名 SendTime および TSOffset のデータを取得する．18：がサーブレット実行開始時刻からフォームデータの送信時刻の差をとることで求める．getParameter()メソッドを用いて HTML フォームからデータを取得しているが，文字列として取得する．差をとるためには，long 型などの数値データに変換する必要がある．ここでは，longValue()メソッドを用いて変換をおこなっている．

２．SQL による検索時間の計測

次に，SQL による検索時間を計測する（③～⑤の区間における計測）．国道検索および人口検索は JDBC による SQL プログラムを用いている．HiRDB 内部による計測は困難であるため，Java プログラムによる測定を行う．

Java プログラム内では，Date クラスの Date()メソッドにより時刻を取得し，上記 C 言語の場合と同様に求める．以下はその記述例である．

```
        :
09：java.util.Date start=new java.util.Date();//開始時刻取得
10：
11：//計測対象のコード
12：
13：java.util.Date finish=new java.util.Date();//終了時刻取得
14：long eTime=finish.getTime()-start.getTime();//実行時間
        :
```

09：で開始時刻を取得する．11：の部分に計測対象のコードを記述する．この場

合計測対象とは，SQL 文を作成する箇所と，DB へ接続し SQL によってデータを取得する箇所である．13：により終了時刻を取得し，14：によって実行時間を計算する．

３．サーブレット全体の実行時間

最後に，②～⑥の区間おけるサーブレット全体の実行時間の計測である．サーブレットは Java 言語を用いているので，「２．SQL による検索時間の計測」と同様に Date クラスの Date()メソッドにより時刻を取得する．

```
        :
09：java.util.Date start=new java.util.Date();//開始時刻取得
10：
11：//計測対象のコード
12：
13：java.util.Date finish=new java.util.Date();//終了時刻取得
14：long eTime=finish.getTime()-start.getTime();//実行時間
        :
```

この場合，計測対象はサーブレットプログラム全体であるので，リクエストを受信しサーブレットが呼び出された時点から，結果をブラウザへ返す直前までである．

４．実行環境

実行環境には Web サーバ，DB サーバ，クライアントの性能と DBMS の設定の最適化の４点が挙げられる．処理時間の差が分かりやすいように最適化は行っていない．性能は下記ようになっている．

Web サーバの性能

CPU	: Celeron　2.4GHz
メモリ	: 512MB
OS	: Windows 2000 Advanced Server (SP4)
Web サーバ	: Apache2.2.4
Web サーバ･サーブレットコンテナ	: Tomcat 5.5.23

DB サーバの性能

CPU	: Celeron　2.8GHz
メモリ	: 512MB
OS	: Windows 2000 Advanced Server (SP4)
DBMS	: HITACHI HiRDB/Single Server Ver.7
空間検索プラグイン	: HITACHI HiRDB Spatial Search Plug-in Ver.3

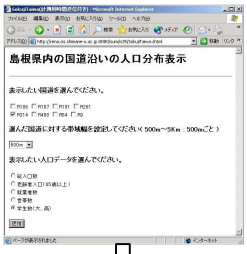
クライアントの性能

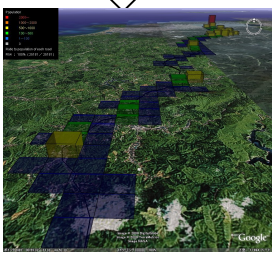
CPU	: Celeron　2.0GHz
メモリ	: 512MB
OS	: Windows 2000 (SP4)
Web ブラウザ	: Internet Explorer6.0

ネットワークの性能

LAN	：100BASE-TX
ハブ	：CentreCOM MR820T

４．９．２．計測結果





検索条件

国道　：R54（153 点）

帯域幅　：500

人口　：総人口

検索結果

クライアント→Web サーバの通信時間	： 0.03 sec
国道検索 SQL 発行時間	： 0.04 sec
国道検索 SQL 実行時間	： 0.04 sec
人口検索 SQL 発行時間	： 0.04 sec
人口検索 SQL 実行時間	：15.62 sec
サーブレット全体処理時間	：15.21 sec

この結果からわかるように，人口検索 SQL の実行がシステム全体の処理時間のほぼ全体を占めており，ボトルネックとなっていることがわかった．

なお，SQL 発行時間は SQL 文を動的に作成し，実行の準備にかかる時間である．SQL 実行時間は SQL 文（SELECT 文）を実行しデータを抽出するのにかかった時間である．

人口検索では Intersects メソッドおよび Buffer メソッドを用いている．さらに，このどちらのメソッドに時間がかかっているかを特定するために，つぎのような計測を行った．

１．Buffer メソッドと Intersects メソッドの処理時間の計測

計測方法は Buffer メソッドを事前に実行し，Intersects メソッドのみを実行した場合の処理時間と Buffer メソッドと Intersects メソッドの両方のメソッドを実行した場合の処理時間の比較を行う．

検索条件

国道	：R54（153 点）
帯域幅	：500～5000(500m 毎)
人口	：総人口

表 4.9.1　Buffer メソッドの処理時間

帯域幅(m)	Intersects のみ(秒)	Buffer＋Intersects(秒)	速度向上率	Buffer 点数
500	2.453	15.063	6.1	547
1000	2.297	13.312	5.8	459
1500	2.266	12.265	5.4	405
2000	2.297	11.812	5.1	345
2500	2.344	11.500	4.9	306
3000	2.391	11.328	4.7	277
3500	2.500	11.281	4.5	256
4000	2.610	11.250	4.3	231
4500	2.813	11.547	4.1	224
5000	2.984	11.594	3.9	211

この結果からわかるように，Buffer メソッドを事前に実行し，Intersects メソッドのみを実行した場合の処理時間と Buffer メソッドと Intersects メソッドの両方のメソッドを実行した場合の処理時間では 4～6 倍の速度向上が見られる．

そのため，ボトルネックとなる Buffer メソッドが作成する Polygon を事前に作成することで，大幅な処理時間の短縮が得られたといえる．

これらのことから，本システムの目的である「国道沿いの人口分布を表示するシステム」の実装を行うことを達成することが出来ただけ出なく，Google Earth 又は Google Maps に表示することにより，視覚的にも見やすいものにすることが出来た．さらに，ボトルネックとなっていた箇所を高速化することにより，システムの性能向上が図れ，より実用的なシステムを構築することが出来たといえる．

第５章　PC と携帯端末を用いた近隣バス時刻表検索システム

５．１．システムの概要

本システムは「PC 又は GPS 携帯端末を用いた現在位置の情報と目的地の情報を元に近隣バス停の中で最適なバス時刻表」という検索をブラウザや Web サーバ，DB サーバを用いて行い，Google Maps 上に結果を表示するというシステムの設計と実装を行う．

図 5.1.1　携帯端末での結果

図 5.1.2　PC での結果

５．１．１．システムの流れ（使用方法）

携帯端末の場合の使用方法は，まず，GPS 又は簡易位置情報機能付きのものである場合は現在位置を取得し，その情報を Web サーバに送信する（①②）．Web サーバはこの情報を元に周辺地図を生成し，クライアントに送信する（③④）．クライアントはその周辺地図から現在位置を選択し，その結果を Web サーバに送信する（⑤⑥）．GPS 又は簡易位置情報ない場合は，松江市全体の地図が表示されここから現在位置を選択する（①②）．Web サーバは目的地の検索のために松江市全体の地図を生成し，クライアントに送信する（⑦⑧）．クライアントはこの地図から目的地を選択する．選択後は日時を選択し，その情報を Web サーバに送信する（⑨⑩）．Web サーバは送信されてきたデータを元に JDBC を用いて SQL を発行し，DBMS に接続し，現在位置周辺のバス停と目的地周辺のバス停を検索し，検索結果のバス停から路線を検索する．この路線を元に時刻を検索し，その中で最適なものと出発のバス停と目的のバス停の位置を表示した地図をクライアントへ送信する（⑪⑫⑬⑭⑮）．クライアントは地図と時刻を取得する（⑯）．次の時刻を検索したい場合は「次の時刻」を選択することで表示される（⑰⑱⑲⑳㉑㉒）．

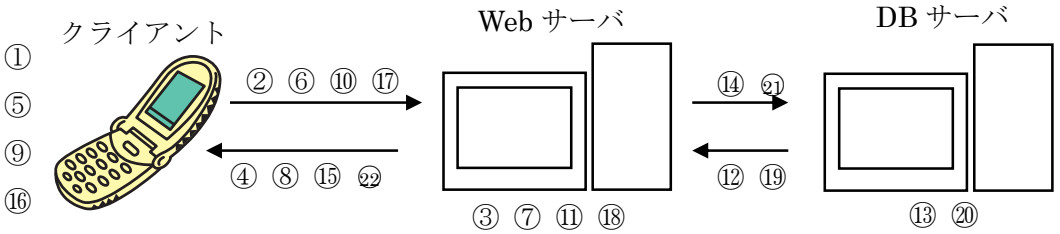


図 5.1.3　携帯端末の場合のシステムの流れ

PC の場合の使用方法は，地図上から出発地と目的地をクリックし，その後に日時を選択する（①）．その情報を Web サーバに送信する（②）．Web サーバは受け取ったデータを元に，携帯端末の場合と同様の処理を行い，出発のバス停と目的のバス停の位置と時刻を取得する（③④⑤⑥）．取得した情報をクライアントへ送信する（⑦）．クライアントはその情報から時刻と地図上に出発のバス停と目的のバス停を表示する（⑧）．Web サーバとの通信には Ajax による非同期通信を用いることで画面遷移のない高速な処理を行っている．携帯端末と同様に「次へ」をクリックすることで，次の時刻を表示する（⑨⑩⑪⑫⑬⑭）．

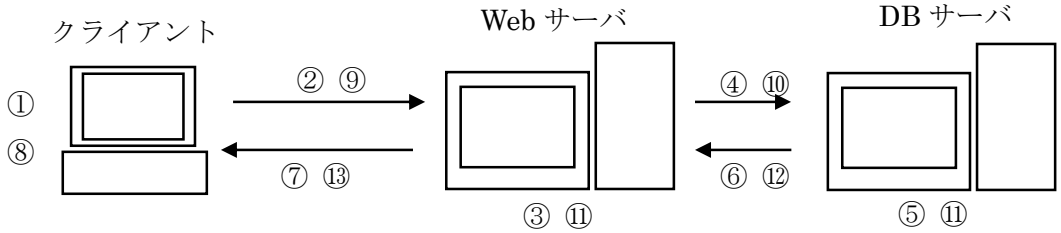


図 5.1.4　PC の場合のシステムの流れ

５．２．携帯端末を用いた場合の位置情報取得方法

まず，携帯端末を用いた位置情報の取得方法は現在位置の取得には GPS や簡易位置情報というものがある．

GPS というのは Global Positioning System（全地球的測位システム）の略で，米国防総省が管理する衛星からの電波を利用し，緯度，経度，高度などを数十メートルの精度で割り出し，現在地を知らせるシステムのことである．

簡易位置情報というのは，コンテンツ側からの要求により，利用者同意の上で，端末の大まかな位置情報を送信するというものである．ただし，送信される位置情報は端末のおおよその位置を示すものであり，正確な位置とは異なる(500m～十数 km の誤差が発生する．)．

GPS 機能は au についてはほとんどの機種に搭載してあるが，docomo，SoftBank については搭載している機種は多くない．しかし，すべての携帯会社がこの機能を提供している．また，簡易位置情報については docomo 以外の携帯会社がこの機能を提供しており，ほとんどの携帯端末にこの機能が搭載している．

そこで，本システムでの現在位置取得方法として GPS と簡易位置情報の両方に対応したものを実装することとした．

５．２．１．現在位置取得

現在位置取得方法は GPS と簡易位置情報であるが，簡易位置情報については大きな誤差生じてしまう．また，GPS でも大きな誤差が生じてしまうかもしれないということから，GPS 又は簡易位置情報を用いて，周辺地図を生成し，周辺地図から選択させる方法をとることとした．また，GPS や簡易位置情報といった機能を搭載していないものについては松江市全体の地図から現在位置を選択させることとする．

５．２．２．目的地取得

目的地の取得には GPS や簡易位置情報といった機能を用いることが出来ないので，松江市全体の地図から目的地を選択する方法をとる．

５．２．３．GPS，簡易位置情報

GPS 又は簡易情報の取得方法は URL にそれぞれ必要なキーワードを付加することで使用することができる．

実際に，GPS 又は簡易位置情報を取得する Java サーブレットである http://rena.cis.shimane-u.ac.jp/s033033/servlet/Position に位置情報を取得させるには以下のように記述する[14]．

・au の GPS の場合

```
<a href="device:gpsone?url=http://rena.cis.shimane-u.ac.jp/s033033/servlet/Position&ver=1&datum=0&unit=0">au（GPS）版</a>
```

※ここで，位置情報取得は""で囲まれた URL へのリンクを表すタグであり，"位置情報取得"が実際の画面に表示される．

・au の簡易位置情報の場合

```
<a href="device:location?url=http://rena.cis.shimane-u.ac.jp/s033033/servlet/Position">au（簡易位置情報）</a>
```

という記述となる. device の後ろが gpsone の場合は GPS による測位となり, location の場合は簡易位置情報による測位となる．また，これらに記述しているパラメータについては,

・ver　　：GPS のバージョン
・datum　：測地系(0：世界測地系 WGS84，1：日本測地系 TOKYO)
・unit　　：緯度経度の表記方法（0 : dd.mm.ss.sss 度分秒表記，1 : dd.ddd 度表記）

の 3 つのパラメータに関しては，GPS 取得の際に必要である．ver については情報が無かったため 1 を，本システムに関して datum について世界測地系を用いているので 1 を，unit についても dms(度分秒)表示の方が扱いやすいため 0 を入力値としてある．また，実際にこの URL を記述したページにアクセスし，位置情報を取得すると，以下のようなパラメータを付加した上で，その url=で指定した URL に移動する．

```
http://rena.cis.shimane-u.ac.jp/s033033/servlet/Position?ver=1&datum=0&unit=0&lat=+35.28.52.82&lon=+133.04.12.85&alt=51&time=20031219154915&smaj=15&smin=10&vert=25&majaa=14&fm=0
```

これらに記述している付加されたパラメータについては,

・lat　　：緯度
・lon　　：経度
・alt　　：高度
・time　：取得日時
・smaj　：長軸半径誤差
・smin　：短軸半径誤差
・vert　：高度誤差
・majaa　：誤差楕円長軸角度
・fm　　：測位方法（何を用いて測位したか数値が小さいほど精度が良い）

であると思われる.

・SoftBank の場合

```
<a href="location:auto?url=http://rena.cis.shimane-u.ac.jp/s033033/servlet/CurrentPosition">SoftBank 版</a>
```

という記述になる．location の後ろが cell の場合は簡易位置情報による測位となり，gps の場合は GPS による測位となり，auto の場合は端末で優先された測位となる．本システムでは汎用性の高さから auto とする．また，実際にこの URL を記述したページにアクセスし，位置情報を取得すると，以下のようなパラメータを付加した上で，その url=で指定した URL に移動する．

```
http://rena.cis.shimane-u.ac.jp/s033033/servlet/Position?pos=N35.28.52.82E133.04.12.85&geo=wgs84&x-acr=3
```

これらに記述しているパラメータについては,

・pos：座標値（1/100 秒単位で度分秒表記）N 北緯，S 南緯，E 東経，W 西経
・geo：測地系（wgs84：世界測地系，tokyo：日本測地系，itrf：ITRF 系）
・x-acr：精度 1：簡易位置情報(300m 以上)2：GPS(50m～300m)3：GPS(50m 以内)

・DoCoMo の場合

```
<a href="http://rena.cis.shimane-u.ac.jp/s033033/servlet/CurrentPosition&com=docomo" lcs>
docomo 版</a>
```

という記述となる．lcs を付加することで GPS を取得することが出来る．また，実際にこの URL を記述したページにアクセスし，位置情報を取得すると，以下のようなパラメータを付加した上で，その指定した URL に移動する．

```
http://rena.cis.shimane-u.ac.jp/s033033/servlet/Position?lat=+35.28.52.82&lon=+133.04.12.8
5&geo=wgs84&x-acc=3
```

これらに記述しているパラメータについては，

・lat：緯度（全て±dd.mm,ss.sss の度分秒表記）

・lon：経度（全て±dd.mm,ss.sss の度分秒表記）

・geo：測地系（wgs84：世界測地系，tokyo：日本測地系）

・x-acc：水平誤差（3：50m 以内，2：50m 以上 300m 未満，3：300m 以上）

このような方法により，現在位置を取得し，得られた緯度と経度のデータを Web サーバへ送信する．

５．３．PC での位置情報取得方法

PC では Google Maps API を用いてクリックした位置の情報を取得することで出発地と目的地の位置情報を取得することが出来る．以下に具体的な方法としてプログラムを記述する．

Google Maps API を用いた位置情報取得方法

```
function onLoad(){
  //発場所の地図作成
  map = new GMap2(document.getElementById('map'));
  map.addControl(new GScaleControl());
  map.setCenter(center, 13);
  //クリックした位置を取得
  GEvent.addListener(map, 'click', onsMapClick);
}
//クリック処理
function onsMapClick(overlay, point){
  y=point.y;
  x=point.x;
}
```

ここでは，GEvent.addListener(map, 'click', onsMapClick);によりクリックした位置の緯度経度をそれぞれ point.y，point.x に格納する．その緯度経度を x と y に格納することに

より，緯度経度を取得することが出来る．PC ではこの方法により，クリックした位置から緯度経度を取得している．

５．４．格納データ

格納データに関して，今回このシステムを作成するにあたり，バス停や路線のデータ，時刻表のデータを松江市の方から提供をして頂くことができたため，松江市の実際のデータを格納している．ただし，時刻表の変更などで現在の時刻表とは合っていない部分も存在するとは思われる．

また，このシステムでは，一畑バスと松江市営バスの 2 つのバスの時刻表に対応している．

５．４．１．バス停位置テーブル

各バス停の情報を格納した表である．バス停を検索する場合，この表を利用することになる．

このバス停位置テーブルには，バス停 ID，バス停名，バス停の経度・緯度のデータを格納している．バス停 ID は各バス停毎にユニークに付けた ID を格納している．このユニークな ID は提供して頂いたデータに付けてあった ID とほぼ同じものを付けている．

次にバス停名は，提供して頂いたデータをそのまま使用させて頂いている．

そして，バス停の経度・緯度に関しては，提供して頂いたデータは，日本測地系で計測したデータを 1/1000 秒単位の 10 進整数で記述してあった（例えば経度が 133 度 12 分 34.567 秒であったとすると度と分を秒に直し，1000 倍したもので，つまり (133×3600＋12×60＋34.567)×1000 を計算した値）．

そのため，Google Maps で扱えるようにするために世界測地系に変換する必要がある．

世界測地系への変換方法は「４．３．１．地域区画区分」で示したように，まず，3600000 で割り，度で表す．次に，その座標の整数部分から１次メッシュコードを導く．さらに，その座標の整数部分をひいたものからそれぞれ緯度を 12 倍，経度を 8 倍し，その整数部分を２次メッシュコードとする．最後に，2 次メッシュコードを導いた座標から整数部分を引いたものからそれぞれ緯度を 120 倍し，経度を 80 倍し，その整数部分を 3 次メッシュコードとする．

導いた 3 次メッシュコードを「４．６．３．日本測地系から世界測地系への座標変換」で行った方法と同じように変換することで世界測地系の座標が得られる．

しかし，このままでは地図からの検索をする上で，経度と緯度の 1 度あたりの長さが異なっているため，円形で検索をすると誤差が出てしまう．そのため，長さを問う位置する必要がある．この問題に対して，３次メッシュがほぼ１ｋｍということから，緯度経度をそれぞれ 1200 倍，800 倍することで，１が 100m となり，この値を格納している．

また，この経度・緯度は HiRDB Spatial Search Plug-in を用いて検索ができるようにジオメトリー型（ユーザ定義型）で定義し，格納している．このプラグインを

使用することで，比較的簡単な SQL 文を発行することで円形などの検索が可能となる．実際に格納した表は以下のようなものである．下表は松江市営バスの場合の表の一部である．一畑バスの表も同じく格納している．

表 5.4.1　バス停位置テーブル

バス停 ID （VARCHAR 型）	バス停名 （VARCHAR 型）	X 座標 （VARCHAR 型）	Y 座標 （VARCHAR 型）	座標 （ユーザ定義(ジオメトリー)型）
10012	上乃木	35.4469	133.0644	point(42536.3443 106451.5507)
10024	相生町	35.4567	133.0605	point(42548.1166 106448.4793)
10033	相生町入口	35.4594	133.0607	point(42551.3966 106448.5927)
:	:	:	:	:
10045	朝日町	35.4644	133.0602	point(42557.3156 106448.2234)
10054	上谷南	35.4489	133.0846	point(42538.7177 106467.6855)
10062	三中前	35.4623	133.0687	point(42554.4490 106454.9923)
:	:	:	:	:
10082	朝酌公民館前	35.4646	133.1017	point(42557.5303 106481.3724)
:	:	:	:	:
10094	石橋町	35.4815	133.0594	point(42577.8928 106447.5393)

表 5.4.1 のようにユニークに付けたバス停 ID，バス停名は VARCHAR 型，表示に使用する座標である X 座標，Y 座標を VARCHAR 型，検索に使用する座標である座標をユーザ定義型で定義し，格納を行った．プライマリーキーはバス停 ID とする．空間 INDEX は秒単位での分割を行っている．実際に格納したバス停数は，松江市営バスの表で 475 カ所，一畑バスの表で 622 カ所であった．

５．４．２．バス路線テーブル

バス路線テーブルは，時刻表を検索する場合に必要となる，バスの路線の情報を格納した表である．出発バス停と目的バス停を使った検索(両方のバス停を通る路線の検索)を行うためのデータを格納してある．

具体的にはユニークに付けた路線 ID，路線名，バス会社，通過するバス停 ID の列で構成されており，通過するバス停 ID の列を検索することで，路線の検索を行っている．路線の検索において，通過するバス停 ID の列はスペースで区切った文字列として，格納している．SQL 文発行の際には，SQL 文の条件文で，”%出発バス停 ID%目的バス停 ID%”とすることで，検索がすることが出来る．ここで，%は 1 文字以上の文字列を表している．このデータに関しては，提供頂いたデータを元に作成を行った．

実際に格納した表は以下のようなものである．松江市営バスの表の一部である．通過するバス停 ID のバス停数は各路線によって，異なっている．また，一畑バスも同じように格納している．

表 5.4.2　バス路線テーブル

路線 ID （VARCHAR 型）	路線名 （VARCHAR 型）	通過するバス停 ID （VARCHAR 型）
10020101	県合同庁舎－川津 竪町～大橋	11612 11463 12552 10944 10952 10414 … 10274
10120101	平成車庫－川津 桧山～駅～くにびき	12321 12331 12341 12351 12454 12464 … 10274
10140101	平成車庫－川津 八重垣･作橋･駅･大	12321 12331 12341 12351 12454 12464 … 10274
10190102	松江駅－平成車庫 桧山･八重垣	11224 10173 10493 11181 13053 10983 … 12322
10400101	馬潟－県庁前松江駅 経由せず	11192 11302 11372 11292 10932 11202 … 10384
10510102	川津－竹矢 大橋～駅～作橋	10273 10732 10262 10093 10112 10103 … 10820
10700101	松江駅－女子高前 くにびき	11228 10374 10684 10691 11241 11251 … 10541
:	:	:
17120101	北循環（内回り）福祉センター経由せず	11221 10374 10684 11514 10674 11164 … 11220

表 5.4.2 のようにユニークに付けた路線 ID，路線名，路線情報の列は全て VARCHAR 型で定義し，格納を行った．プライマリーキーは路線 ID とする．格納したバス路線の数は，松江市営バスが 102 路線，一畑バスが 159 路線であった．格納した行数はバス停位置テーブルより少ないが，こちらの表の方が時刻表の改訂で変更される可能性が高く，大幅な変更があった場合など，DBMS を使用せずに更新することは不可能である．

５．４．３．バス編成テーブル

バス編成テーブルは，各バス停の時刻表を格納している表である．バス停位置テーブルよりバス停 ID を検索し，その ID より，路線テーブルからどの路線の何番目に通るバス停 ID かを検索した後で，時刻表を検索する．

具体的には，路線 ID，時刻，バス ID，備考の列で構成されている．路線 ID はバス路線テーブルの路線 ID と同じもの(その時刻の路線名の ID)が格納されており，バス ID については同じ路線でも何本もバスがあるので何本目のバスなのかを決定するために用いる．時刻については通過するバス停と同形式で時刻が格納されている．そのため，何番目に通過するバス停かがわかれば，時刻を特定することが出来る. このデータに関しては, 提供していただいたデータの状態を元に作成を行った.

実際に格納した表は以下のようなものである．松江市営バスの表の一部である．時刻の数は各路線によって，異なっている．また，一畑バスも同じように格納している．

表 5.4.3　バス編成テーブル

路線 ID （VARCHAR 型）	備考 （VARCHAR 型）	バス ID （INTEGER 型）	時刻 1 （TIME 型）	…	時刻 18 （TIME 型）	…
10020101	平日	1	7:40:00	…	8:03:00	…
10120101	平日	2	6:25:00	…	6:41:00	…
10120101	休日	2	16:55:00	…	17:15:00	…
10510102	休日	3	18:50:00	…	19:14:00	…
:	:	:	:	:	:	:
10610102	平日	2	15:30:00	…	0:00:00	…
11390101	休日	22	15:45:00	…	16:06:00	…
11400102	平日	1	7:25:00	…	7:50:00	…
11850101	平日	6	16:00:00	…	0:00:00	…
17040101	平日	2	17:45:00	…	18:08:00	…
:	:	:	:	:	:	:
17080101	平日	3	19:00:00	…	19:18:00	…
17140101	1, 3, 5 土曜	2	20:00:00	…	20:19:00	…

表 5.4.3 のように，路線 ID，備考は VARCHAR 型で，バス ID は INTEGER 型，時刻は TIME 型で定義し，格納した．格納した時刻表の数は，松江市営バスが 1238 個，一畑バスは 818 個であった．

５．５．検索方法

検索手順は

①出発地と目的地で共に半径 500m 以内にバス停があるか判定する．なければ終了．

②受け取った出発位置からまず，半径 100m 以内の周辺のバス停を検索する．なければ，半径 200m 以内の周辺のバス停を検索する．これを 100m 毎に 500m まで行い，１件も見つからなければ，終了する．

③目的位置からも出発位置と同様にまず，半径 100m 以内の周辺のバス停を検索する．なければ，半径 200m 以内の周辺のバス停を検索する．これを 100m 毎に 500m まで行い，１件も見つからなければ，終了する．

④周辺のバス停が見つかった場合，そのバス停と出発のバス停との路線を検索する．路線があれば，時刻表を検索し，なければ，次の目的位置周辺のバス停を検索する．それでもない場合は②へ戻る

⑤路線があった場合，この路線の何番目のバス停かを取得する．

⑥何番目のバス停かと日時を元に時刻を検索する．

このような手順により，検索を行う．以下にフローチャートを示す．

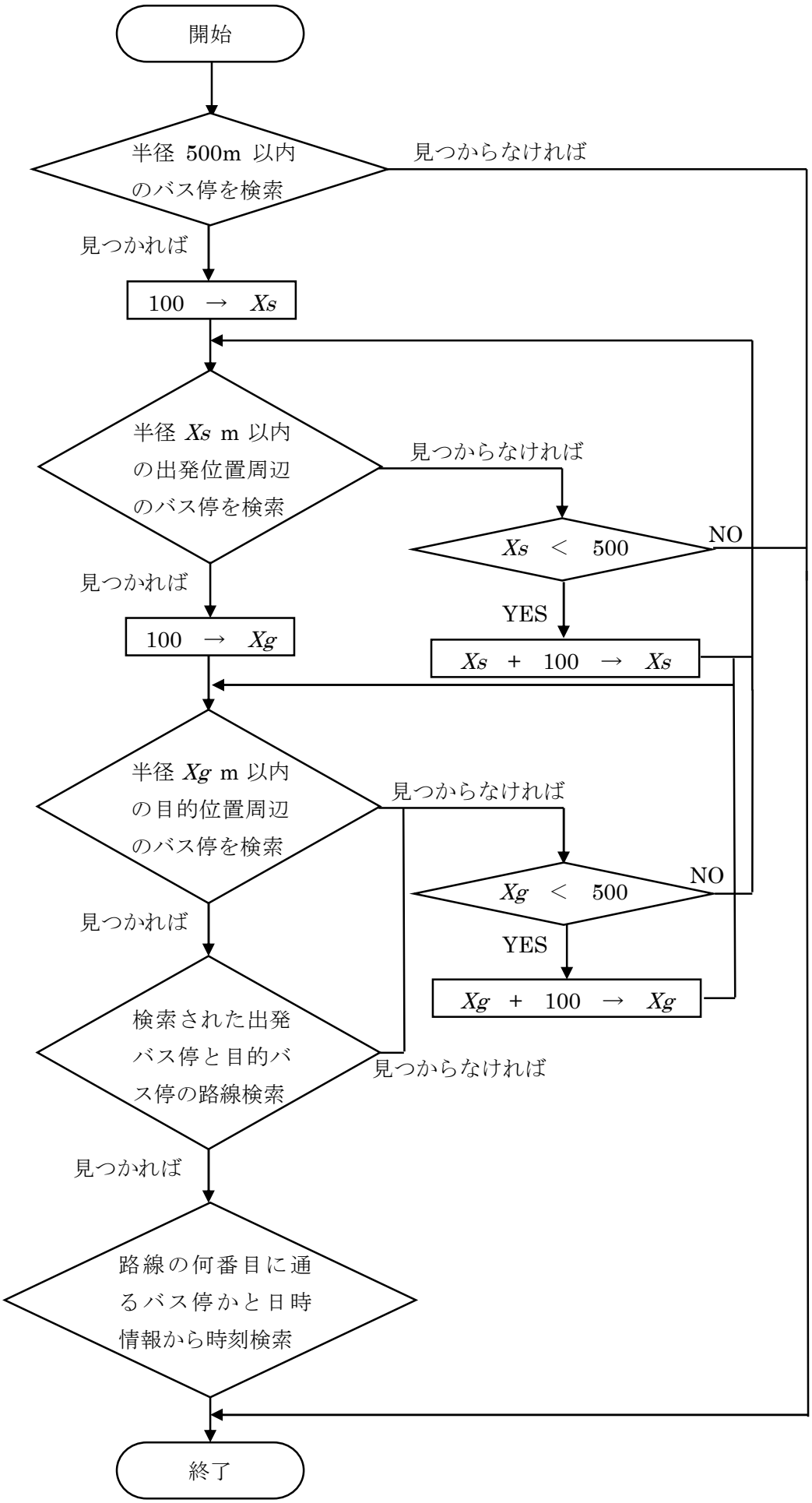


図 5.5.1　検索手順のフローチャート

５．５．１．周辺のバス停位置検索

周辺のバス停位置の検索には，取得した位置情報と各バス停の位置情報を HiRDB の空間検索プラグイン(Spatial Search Plug-in)利用して DBMS で検索を行っている．まず，取得した位置情報を変換して，各バス停の位置情報を用いた検索を出来るようにする．ここでは，例として半径 100m 以内のバス停の検索を行う．

実際の HiRDB へ発行する SQL 文は，以下のようなものである．

```
SELECT バス停名 FROM バス停位置 WHERE WITHIN(座標, RegionFromText('CIRCLE(経度 緯度, 半径[100m] )')) IS TRUE;
```

※”バス停名”は列名．”バス停位置”は表名．”WHERE WITHIN”以下が検索の条件，”RegionFromText( )”は HiRDB Spatial Search Plugin の独自のもので空間検索する場合に( )内に条件を指定する(ジオメトリー型を指定)．”CIRCLE( )”も Spatial Search Plug-in 独自のものであり，円形を表し，( )内は x 座標，スペース y 座標，カンマ，半径の順で入力する．

経度・緯度を変換したものをこの SQL 文内に代入し，SQL 文を発行することで，ユーザの情報を元に検索することができる．

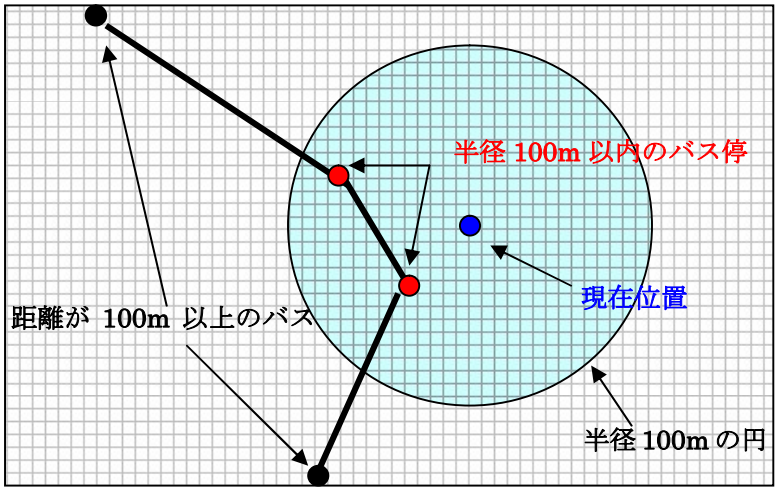


図 5.5.2　空間検索

５．５．２．時刻検索

時刻の検索には乗換えがある場合と乗換えがない場合の検索，この２つが考えられる．

１．乗換えなしの場合

乗換えがない場合の検索方法は，まず，周辺検索により出発地のバス停と目的地のバス停をバス停位置テーブルから検索する．バス停位置テーブルから取得したバス停 ID を元にバス路線テーブルから，路線 ID と何番目に通るバス停かを検索する．この検索結果を元にバス編成表から同じ位置にある時刻を検索する（図 5.5.2）．この方法により，時刻を取得する．実際の HiRDB へ発行する SQL 文は，以下のようなものである．

```
SELECT 路線 ID FROM バス路線 WHERE ルート一覧 LIKE '%10873%00873%';
```

※"路線 ID"は列名．"バス路線"は表名．"WHERE "以下が検索の条件，"LIKE"は部分検索を行うもので"%"には任意の 0 文字以上の文字列が挿入される．

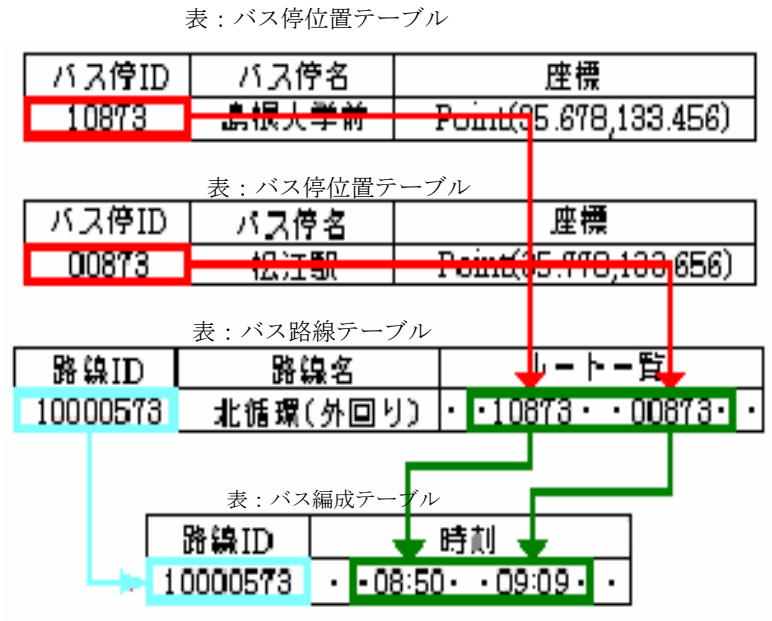


図 5.5.3　検索方法（乗換えなし）

2．乗換えありの場合

乗換えありの場合の検索方法は，まず，乗換えなしの時と同様に出発地のバス停と目的地のバス停をバス停位置テーブルから検索する．ここで，路線が存在しなかった場合，乗換えありの検索を実行する．検索方法は，まず，出発バス停から乗換え駅である松江駅までで乗換えなしと同様の検索し，次に松江駅から目的バス停までででまた乗換えなしと同様の検索を行うというものである（図 5.5.3）．

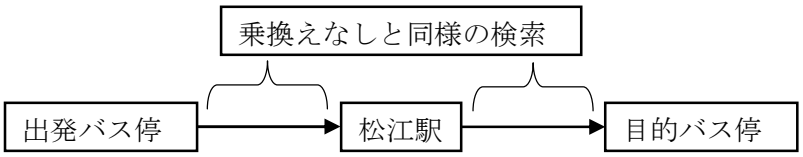


図 5.5.4　検索方法イメージ図(乗換えあり)

3．時刻検索の問題点とその解決法について

次の時刻検索の方法は，まず，問題となってくるのが，出発バス停の時刻と目的バス停の時刻が同じバスの時刻であるかどうかである．例えば，下記のような時刻表（表 5.5.1）があったとする．ここで，出発バス停の路線 ID が 1 で時刻 1，目的バス停の路線 ID が 1 で時刻 2 とした場合，実際には出発時刻 09：00，到着時刻 09：20 とならなければならないものが，時刻が早いものを先に検索してしまうので出発時刻 09：00，到着時刻 09：15 となってしまい，矛盾が生じてしまう．

実際の HiRDB へ発行する SQL 文は，以下のようなものである．

出発時刻

```
SELECT 時刻１ FROM バス編成;
```

※”時刻１”は列名．”バス編成”は表名．

到着時刻

```
SELECT 時刻２ FROM バス編成;
```

※”時刻２”は列名．”バス編成”は表名．

表 5.5.1　バス編成（問題あり）

路線 ID	時刻１	時刻２	時刻３	…
1	09：00	09：20	09：30	…
1	09：10	09：15	09：35	…

ここで矛盾を解決するためにバス自身の ID であるバス ID を設ける．すると，表は以下のようになる（表 5.5.2）．

表 5.5.2　バス編成（問題なし）

路線 ID	バス ID	時刻１	時刻２	時刻３	…
1	1	09：00	09：20	09：30	…
1	2	09：10	09：15	09：35	…

ここで，この表を用いた場合の先ほどと同様の時刻検索の方法は，出発時刻の検索の際にバス ID を取得し，到着時刻の検索の際にそれを条件とし検索を行う．そうすることで，問題を解消することが出来る．先述したものと同様のものを例とすると，出発バス停の路線 ID が 1 で時刻 1，目的バス停の路線 ID が 1 で時刻 2 とした場合，出発時刻 09：00，バス ID は 1 となる．この場合，到着時刻はバス ID が 1 の時刻 2 となるものなので 09：20 が検索され，正しい結果が表示されるようになる．実際の HiRDB へ発行する SQL 文は，以下のようなものである．

出発時刻

```
SELECT 時刻１,バス ID FROM バス編成;
```

※”時刻１”は列名．”バス編成”は表名．

到着時刻

```
SELECT 時刻２ FROM バス編成 WHERE バス ID=1;
```

※”時刻２”は列名．”バス編成”は表名．”WHERE ”以下が検索の条件，”バス ID=1”は出発時刻を検索した際に取得したバス ID である 1 と同じバス ID を検索するものである．

５．６．結果の表示

携帯端末での結果の表示には Google Maps を画像化したものを用い，PC での結果表示には Google Maps API を用いている．以下で方法について説明する．

５．６．１．携帯端末での結果の表示方法

指定した座標の Google Maps を画像化し，送信するという機能を用いて携帯端末に結果の地図画像などを表示している．この機能を用いることで地図上にアイコンを表示することも出来る．

・使用方法

下記のアドレスにアクセスすることで，画像データを取得することが出来る．

```
http://maps.google.com/mapdata?cc=JP&min_priority=1&w=240&h=120&latitude_e6=35658632
&longitude_e6=139745411&zm=1200&Point=b&Point.latitude_e6=35658632
&Point.longitude_e6=139745411&Point.iconid=33&Point=e
```

これらのパラメータについては，

・w：画像の幅

・ｈ：画像の高さ

・cc：カントリーコード 日本では JP

・min_priority：不明

・latitude_e6：緯度

・longitude_e6 経度

・zm：ズーム

・Point：マーカー指定の開始箇所に Point=b，マーカー指定の終了箇所に Point=e として指定する．

・Point.latitude_e6：アイコンの緯度

・Point.longitude_e6：アイコンの経度

・Point.iconid：アイコンの種類を指定

これらのパラメータによって Google Maps を画像化し，アイコンを付加させて表示することが出来る．

５．６．２．PC での表示

PC では主に Google Maps API を用いて，表示を行っているが，より実用的なシステム構築のため，時刻表の表示を非同期通信で行った．非同期通信とは，クライアントとサーバ間で同期を取らずに通信することである．

同期的な通信の場合，現在ブラウザに表示しているページから他のページへ移動する際にはリロードを行う必要があるのに対し，非同期的な通信の場合は，他のページへ移動する際にも絶えずサーバとの通信を行うため，ページ遷移が発生しない．さらに，部分的に更新も行えるので軽い処理となり，高速化にもつながる．このように，非同期通信はシステム利用者にサーバの存在を意識させず，ブラウザの制御を奪うことなしに HTTP 通信を行うことができる．

本システムでは時刻表の表示と次への時刻表の検索の部分で非同期通信を用いた．これにより，高速かつ扱いやすいシステムとなった．

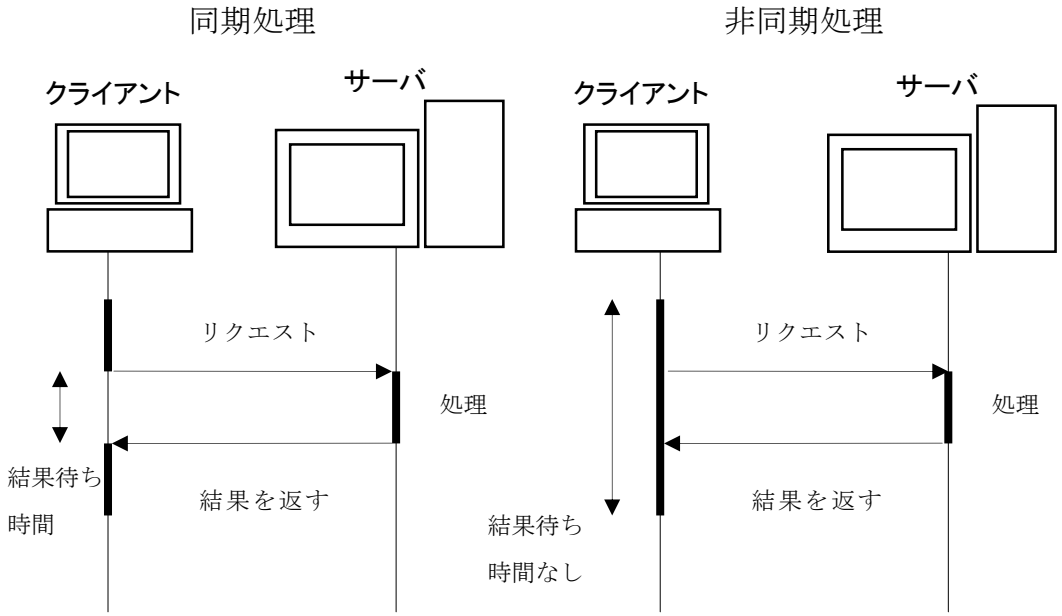


図 5.6.1　同期通信と非同期通信の違い

５．７．類似研究

類似研究として鳥取大学を提供している「バスネット」という Web サイトがある[15]．

この鳥取大学が提供している Web サイトでは，バス路線が２地点間を最短経路で結ぶという発想ではなく，利用者が多い場所を通るように設計されているということから，多数の路線が複雑に入り組み，また，バス停の設置間隔が狭いといえると考え，徒歩移動による乗り換えが可能かつ有効な場合が多く存在するとした．また，出発地や目的地の周辺には複数のバス停が存在している場合が多いことから，行き先によっては距離的に最も近いバス停から乗車することが，出発地から目的地までの所要時間を最短にするとは限らないとした．これらのこと考慮した経路探索を行っている．

５．７．１．経路探索

この鳥取大学が提供している Web サイトではバス経路探索をネットワークの単一始点最短路問題と捉え，バス停を頂点，バスが運行する経路と徒歩移動を辺とし，路線バスネットワークを構築し，経路探索を行っている．辺の重みは所要時間と乗換え回数とする．この所要時間は待ち時間を含めたものを利用するので，静的に定めることは出来ない．例を挙げると，あるバス停に 10：00 に到着し，そのバス停から次のバス停へ行く際の出発時刻は 10：05 であり，次のバス停の到着時刻は 10：15 であった場合，乗車時刻は 10 分であるが，待ち時間も含めると 15 分かかったことになる．このように，到着時間により，辺の重みが変わってしまうので，静的に定めることは出来ない．

この方法を用い，出発地から目的地までの全経路を，出発地を表す節を根とする木と見なして深さ優先探索し，最適経路を見つけるというのがこの Web サイトの経路探索アルゴリズムの基本的な考え方である．ここで，最適経路とは以下のような経路で

ある．

（1）所要時間最短の経路
（2）（1）の経路が複数ある場合は，そのうち乗り換え回数最少の経路
（3）（2）の経路が複数ある場合は，そのうち乗車時間がなるべく短い経路

この生成した経路の探索には計算量が $O(m+nC)$ である 1 レベルバケット法で用いている．

これに対し，本システムの検索方法は前節までで示したような方法をとっており，所要時間最短しか考慮しておらず，検索の点では劣っている．ただし，本システムでは，類似研究とは違い，路線バスネットワークを構築せずに検索を行うので，近隣バス停検索だけでなく，さまざまな検索が行える．例えば，国道沿いにあるコンビニを検索し，さらに何が売っているのかということも検索可能である．類似研究ではこのような検索を行うことは困難である．このように，拡張性においては勝っているということが言える．さらに，運用中の空間データの更新には類似研究では一旦停止して行わなければならないのに対し，本システムでは停止する必要なく更新を行うことが出来るという点でも勝っていると言える．

第６章　終論

本研究では ORDBMS の Buffer メソッドや Intersects メソッドを用いることで，地図上に指定した範囲内の人口データの検索を実装すること出来た．さらに，Buffer メソッドを事前に作成することにより, システムの高速化も図ることが出来た. 表示には Google Earth 又は Google Maps といった技術を用いることで，視覚的にも見やすいものとなった．

もうひとつのシステムでは携帯端末の GPS 機能や簡易位置情報機能を用いることで，現在位置を取得し，さらに，ORDBMS の Buffer メソッドの円検索機能と within メソッドを用いることで近傍バス停検索を実装することが出来た．時刻表の検索では乗換え機能を実装し，表示には Ajax による非同期通信を用いることで，より高速で使いやすい実用的なシステムとなった．

今後は，PC と携帯端末を用いた近隣バス時刻表検索システムでは経路探索の結果をより実用的なものにするために，乗換えバス停を松江駅だけでなく，すべてのバス停に適応させ，これに伴う検索時間の増加を解消するために検索の高速化も図っていきたいと考えている．

さらに，これらの機能を用いたシステムとして人口データや時刻表など更新の少ない整備されたデータだけでなく，気候状況や店舗情報といった更新の多いリアルタイムデータを組み合わせたシステムの実装をしていきたいと考える．

システムの例としてコンビニやスーパーといった店舗の情報を DB に格納し，検索するといった方法とコンビニやスーパーといった店舗は現在ほとんどホームページを持っていることから，そのホームページ上にある住所や商品情報を利用するといった方法がある．これらの方法により，DB や Web からの検索により所得した住所等の情報をもとに店舗の位置を特定し，これにより現在位置又は道路沿いにあるコンビニといった店舗を検索し，そこで売られている商品情報も同時に表示するといった機能の実装も可能であり，こういったシステムの実装も行っていきたいと考えている．

第７章　謝辞

本研究にあたり，最後まで熱心な御指導をいただきました田中章司郎教授には心より御礼申し上げます．また，同じ研究室の今井拓也さん，熊丸恵太さん，的場祥仁さん，皆木健太さん，岩脇正浩さん，加藤広朗さん，清水洋志さんには本研究に関しまして，数々の御協力と御助言を頂きました．厚く御礼申し上げます．

なお，本論文，本研究で作成したプログラム及びデータ，並びに関連する発表資料等の全ての知的財産権を本研究の指導教官である田中章司郎教授に譲渡致します．

文献

[1]「Google Earth」
http://earth.google.co.jp/

[2]「Google Maps」
http://maps.google.co.jp/

[3] 木村眞吾,田中章司郎：「開放型地球儀サーバに空間情報を描画する Web システムの実装」平成 19 年電気・情報関連学会中国支部連合大会論文集

[4] Michael Stonebreaker 著「オブジェクトリレーショナル DBMSs」
太田佳伸 訳
株式会社ビー・エヌ・エヌ
International Thomson Publishing Japan（平成８年)

[5] OpenGIS® Simple Features Implementation Specification for SQL Version1.1 1999
http://www.opengeospatial.org/standards/sfs

[6] Marty Hall 著「コア・サーブレット&JSP Java サーバ技術による Web 開発」
岩谷博 訳
ソフトバンク パブリッシング（2001 年）

[7] 菊田英明 著「実践 JDBC Java データベースプログラミング術」
オーム社開発局（1998 年）

[8] Tomcat5 サーブレット/JSP コンテナ JNDI データソースの手引き
http://www.jajakarta.org/tomcat/tomcat5.0/ja/docs/tomcat-docs/jndi-datasource-examples-howto.html

[9] 総務庁統計局 刊行
平成７年国勢調査地域メッシュコード（平成７年)

[10] （財）日本地図センター 刊行
ＪＭＣマップ CD-ROM 添付ファイル（平成１０年)

[11] 世界測地系移行の概要
http://www.gsi.go.jp/LAW/G2000/g2000.htm

[12] 地理地殻活動研究センター 飛田 幹男 「世界測地系移行のための座標変換ソフトウェア“TKY2JGD”」国土地理院時報（2001 年）
http://cais.gsi.go.jp/Research/space/jiho97tb.pdf

[13] HiRDB マニュアル「HiRDB Spatial Search Plug-in Version 3 解説・手引・文法・操作書」

[14] GPS 携帯 位置情報 基礎知識
http://www.yaskey.cside.tv/mapserver/note/gps.html

[15] 川村尚生,菅原一孔：「バスネットワークのための実用的な経路探索システム」情報処理学会論文誌, Vol.48,No.2,pp.780-790 (2007 年)