# 2点間の経路を考慮した
# 簡易地図データの考察

平成 18 年 2 月 3 日

情報電子工学科
山内伸哉

# 目 次

# 概要

WWW 用の地図を作る場合、Unix 上には定番といえる簡単なソフトが見当たらず、Unix の一般的な描画ソフトのデータを再利用することは難しい。2004 年度に齊藤氏がこの研究に取り組み、地図データと WWW 用の地図を作成したが、この研究で使用された SVG 画像形式に対応しているブラウザは少なかった。そこで本研究では、地図データから多くのブラウザに対応している画像形式を直接作成する方法を考察する。
また、この地図データは、道路と建物のそれぞれのデータのみで、建物と道路をリンクさせることや交差点の識別ができなかったので、2 点間の経路が考えられない。そこで 2 点間の経路を考慮するのに必要な地図データについても考察する。

# 1　はじめに

この研究室では大学付近の飲食店マップを WWW 上で公開している (図 1)。この地図は Tgif という描画ソフトで作られているが、Tgif は Unix の一般的な描画ソフトのため、道路や一部の建物を取り出すような再利用することが難しく、地図の一部切り取り等が困難である。また、Tgif では WWW 用のクリッカブルマップが作りにくい。クリッカブルマップとは、WWW ブラウザ上に表示された 1 つの画像の一部をクリックすることで指定された URL にジャンプできるような仕組みである。平成 16 年度に齊藤氏 [1] がこの問題について研究し、SVG 画像形式の特徴を使うことでクリッカブルマップの作成、地図の切り取りを行っている。しかし SVG 画像形式は対応するブラウザが少ないという問題があった。そこで本研究では多くの WWW ブラウザに対応している PNG 画像を作成する方法を考える。

また、研究室で公開されている WWW 用の地図をカーナビゲーションのように 2 点間を結ぶ経路を計算できれば、2 点間を指定して経路を表示させたり、その経路を音声で出力することで障害者でも理解できる地図となる。そこでこの 2 点間の経路を表示できるような地図データについて考察し、作成された地図データを画像、HTML へ変換するツールの作成を行う。

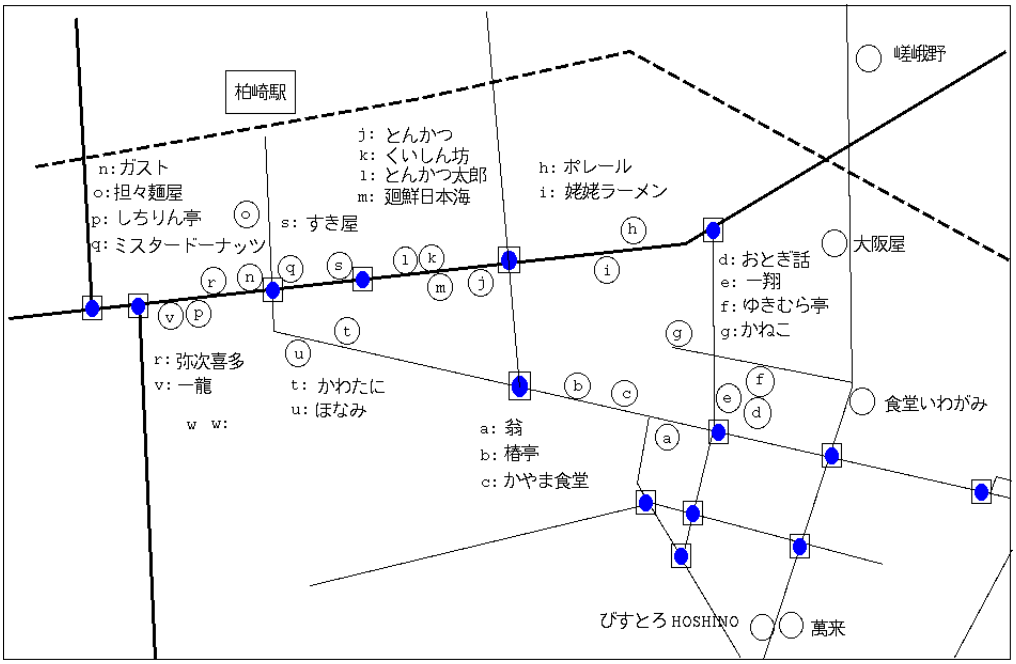


Fig. 1　研究室で公開しているマップ

## 2　過去の研究

過去の研究の概要を紹介し、その問題点をあげる。

平成 16 年度の研究 [1] では、Tcl/Tk の GUI ツールを用いて、地図データを作成し、それを AWK スクリプトを用いて SVG 画像形式に変換し、ビューワソフトである squiggle で WWW で用いることができる PNG 画像に変換したり、その AWK スクリプトによる切り取りを考察している。Tcl/Tk とは、スクリプト言語である Tcl とツールキットである Tk を用いて GUI プログラムを作るものである。SVG 画像形式とは、テキストファイルで線形データを表現する汎用の画像形式で W3C が提唱している WWW 用に開発された画像形式である。過去の研究では、この SVG 画像形式の特徴を使い、以下のようなことを行っている。

- キャンパスの一部分を切り取る長方形の領域を指す SVG 画像のビューポートの指定機能を利用した地図の一部分だけの画像化
- SVG の画像に直接リンクをはれることを利用したクリッカブルマップの実現

しかし、SVG 形式には直接対応するブラウザがまだ少ないのでリンクは貼れても多くのブラウザでは実際には機能はしない。よって、本研究では以下の 2 つのことを研究していく。

1 つ目は、GD モジュールを用いて地図データから直接 PNG 画像への変換を行うことである。GD モジュールは、プログラム上で簡単に PNG 画像を作成できる特徴を持つ。よって、SVG 画像形式で使われたビューワーソフトを使わずに直接 PNG 画像を生成できる。この GD モジュールを使い、地図データから直接 PNG 画像を生成する。

2 つ目は、2 点間の経路を考慮した地図データの考察である。これは、カーナビのように、 ある 2 地点を結ぶ経路を計算できるような地図データを作ることである。 現在ある地図データをもとに 建物と道路のリンク状況や交差点のデータなどもそのデータに入れる必要がある。ここで現在の地図データについて説明する。現在の地図データは表 1 のように 3 つの種類の行より構成される。

| | |
|---|---|
| size 行 | `size` ␣ $x_1$ ␣ $y_1$ ␣ $x_2$ ␣ $y_2$ |
| build 行 | `build` ␣ "名称" ␣ $x_1$ ␣ $y_1$ ␣ 属性 |
| road 行 | `road` ␣ "名称" ␣ $x_1$ ␣ $y_1$ ␣ $x_2$ ␣ $y_2$ ...$x_n$ ␣ $y_n$ ␣ 属性 |

Table 1　現在の地図データ

size 行は、画像のサイズを設定する行である。値は地図の形である長方形の左上の頂点の座標を $(x_1, y_1)$ とし、右下の頂点の座標が $(x_2, y_2)$ となる。build 行は建物のデータを

示し、データには建物の名称、位置を示す座標 $(x_1, y_1)$、建物の種類を表す属性の座標が入っている。road 行は道路のデータを示し、データには道路の名称、道路の始点の座標 $(x_1, y_1)$、中間点の座標、終点の座標 $(x_n, y_m)$、道路の種類を示す属性が入っている。

# 3 GD モジュール

## 3.1 GD モジュールとは

GD モジュールとは、gd lib を perl 上に実装したモジュールである。gd lib とは、グラフィックライブラリである。仮想上のキャンパスに線を引いたり、円を描画したり文字を書いたりできる。フォントの指定によって日本語にも対応している。その仮想上のキャンバスを実際に画像出力するときに PNG、JPEG、GIF 形式などで出力できる。つまり、プログラム上で簡単に PNG 形式の画像を作成できるものが gd lib となっている。本研究では、この gd lib を perl で画像生成する GD モジュールを使用する。GD モジュールの使い方を説明する。

まず、GD モジュールの基本的な構成は以下のようになる。

```
#!/usr/local/GNU/bin/perl

use GD;
  # GD lib を使うことを宣言

$im = new GD::Image(400,400);
  # 新しいイメージを作成

$white = $im->colorAllocate(255,255,255);
$black = $im->colorAllocate(0,0,0);
$red = $im->colorAllocate(255,0,0);
  # いくつかの色を確保

〜 コマンド 〜

binmode STDOUT;
  # バイナリ・ストリームへ書きこむことを確実にする

print $im->png;
  # イメージを PNG に変換し、標準出力に出力
```

このようなプログラムをテキストファイルとして file.pl として保存する。その後、

```
perl file.pl > file.png
xli file.png
```

で PNG 形式に流し、PNG 画像を出力する。

## 3.2 クラス

GD には 3 つのクラスが定義されている。

GD::Image は GD を中心的なイメージクラスで画像データの保持に使われる。GD::Font は、フォントクラスで文字の画像化に使われる。GD にはフォントが 5 種類用意されているがその中に日本語は対応されていない。よって、このフォントクラスで日本語対応のフォントを呼び出す。GD::Polygon は多角形を描画するにあたっての各頂点リストの保持に使われる。

## 3.3 利用する主な関数

**new()**

new() は GD::Image クラスの中心的な生成関数である。デフォルトでは 64 × 64 となる。truecolor イメージは 24 ビットの色データを示す。(R/G/B それぞれ 8 ビット)。デフォルトは 8 ビット。ファイルハンドルやファイル元に画像を生成することができる。

```
$im->new([$width,$heigt],[$truecolor])
$im->new(*FILEHANDLE)
$im->new($filename)
$im->new($data)
```

**newFromPng()**

newFromPng() は、PNG ファイルを指定されたファイルハンドルかファイルのパスから読み込んで画像を生成する。ファイルハンドルは、存在する PNG ファイルかパイプをあらかじめ開いたものである必要がある。成功すれば画像を返し、失敗すれば undef を返す。この呼び出しは、自動的にファイルハンドルを閉じたりしないため、その問題を解消するために ”binmode(FILEHANDLE)” を使う。この関数の引数は STDIN 等の単なるファイルハンドルまたは、ファイルのパス名である。

```
$im->$newFromPng($file, [$truecolor])
$im->newFromPngData($data, [$truecolor])
```

**colorAllocate()**

colorAllocate() を使うことで、GD::Image のカラー表を制御し、操作できる。この関数は、指定した赤、緑、青の成分を持つ色を割り当てる。この方法で割り当てられる最初の色は画像の背景色になる。色の割り当てができなかったときは、−1 を返す。

```
$im->colorAllocate($red,$green,$blue)
```

**line()**

line() は、(x1,y1) から (x2,y2) への指定した色での線分を引く関数である。

```
$im->line($x1,$y1,$x2,$y2,$color)
```

**setThickness()**

setThickness() は、line(),rectangle(),arc() などで描かれる線の太さを変える関数である。デフォルトは 1 ピクセルとなる。例は、$(10, 190)$ から $(140, 10)$ で太さが 3 ピクセルの黒い線を描画する。

```
$im->setThickness($thickness)
 (例) $im->setThickness(3);
      $im->line(10,190,140,10,$black);
```

### ellipse()、filledEllipse()

ellipse() は、円や楕円を描画する関数である。(cx,cy) は円の中心の座標で (width,height) はそれぞれ円の横幅と高さを示す。filledEllipse() は円や楕円の塗りつぶしの関数である。

```
$im->ellipse($cx,$cy,$width,$height,$color)
$im->filledEllipse($cx,$cy,$width,$height,$color)
```

### fill()

fill() は、領域を指定した色で塗りつぶす関数である。その色は点 (x,y) から始まって、それが、開始位置のピクセルと違う色の所に出会うまで画像全体に広がる。

```
$im->fill($x,$y,$color)
```

### unclosedPolygon()

unclosedPolygon() は、指定した色で折れ線を描画する関数である。これは、多角形を描画するにあたっての各頂点リストの保持に使われる new GD::polygon を使い、始点の座標から終点の座標を Polygon に保持することで折れ線を描画することができる。例は、$(250, 10)$、$(200, 80)$、$(280, 100)$、$(250, 180)$ を通る黒い折れ線描画されることになる。

```
$im->unclosedPolygon($poly,$color)
 (例) $poly = new GD::Polygon;
      $poly->addPt(250,10);
      $poly->addPt(200,80);
      $poly->addPt(280,100);
      $poly->addPt(250,180);
      $im->unclosedPolygon($poly,$black);
```

**string()**

GD lib には 5 種類のフォントが用意されているがどのフォントも日本語を使うことができない。

string() は、指定したフォントと色で (x,y) の位置から文字列を描画する関数である。フォントは gdSmallFont、gdMediumBoldFont、gdTinyFont、gdLargeFont、gdGiantFont の 5 種類から選べる。ただし、このフォントはいずれも日本語を使うことができない。

```
$im->string($font,$x,$y,$string,$color)
```

**stringFT()**

stringFT() は、TrueType フォント等を使うことのできる関数である。この関数を使うことで日本語を表示できる。例は、1 行目が指定したいフォントの絶対パスである。2 行目からはその指定されたフォントを使い、(120,150) を始点に 12 ポントの 80 ラジアン半時計回りに回転させた「日本語が使える」という文字が描画される。

```
@bounds = $im->stringFT($fgcolor,$fontname,$ptsize,$angle,$x,$y,
                                                        $string)
```

```
fgcolor  :文字列を描画する色番号
fontname :フォントファイルへのパス、フォントパターン
ptsize   :ポイントサイズ (小数をも可)
angle    :回転角度 (反時計回り、正の値、ラジアン)
x,y      :文字列の描画を始める座標
string   :文字列自身
(例)$jpmfont='/usr/local/x11/share/fonts/ttf/ipam.ttf';
    @bounds = $im->stringFT($black,$jpmfont,12,120,150,80
                            ,"日本語が使える");
```

## 4 描画例

描画コマンドと文字の描画の例を以下のコードで PNG 画像として出力する。

コード

```
#!/usr/local/GNU/bin/perl
use GD;
use encoding "euc-jp";
$jpmfont='/usr/local/x11/share/fonts/ttf/ipam.ttf';

$im = new GD::Image(300,200);

$white = $im->colorAllocate(255,255,255);
$black = $im->colorAllocate(0,0,0);
$red = $im->colorAllocate(255,0,0);

$im->line(0,0,300,200,$red);

$im->setThickness(3);
$im->line(10,190,140,10,$black);

$im->setThickness(1);
$im->ellipse(120,130,150,100,$black);

$im->fill(120,130,$red);

$poly = new GD::Polygon;
$poly->addPt(250,10);
$poly->addPt(200,80);
$poly->addPt(280,100);
$poly->addPt(250,180);
$im->unclosedPolygon($poly,$black);

@bounds = $im->stringFT($black,$jpmfont,12,120,150,80,"日本語が使える");

binmode STDOUT;
print $im->png;
```

以上のコードにより、図 2 の PNG 画像が出力される。

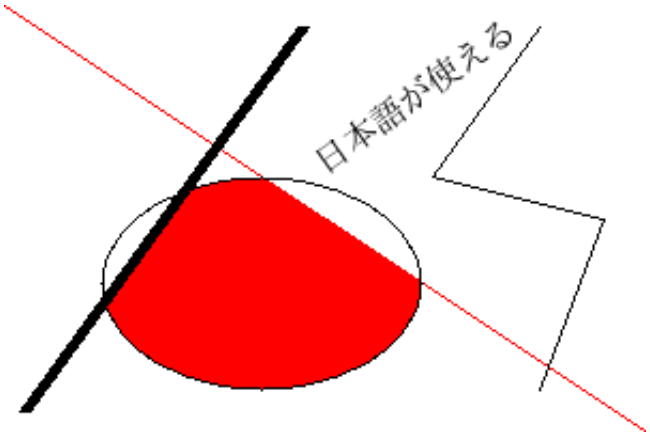


Fig. 2 コードの出力例

## 5 2 点間の経路を考慮した地図データの考察

現在の地図データを元に経路問題を考慮した地図データを作成する。現在の地図データはクリッカブルマップを中心的に考察しているため、建物と道路の地図の必要最低限のデータで構成されている。経路問題を考えるにあたってこの地図データをどのように拡張していくか考察していく。

### 5.1 道路と建物のリンク

2 点間の経路を考える上で建物と道路をリンクさせることが必要であり、そのためにはそれぞれの建物、道路の識別が必要となる。現在の地図データでは、道路、建物に道路名、建物名があるわけだが、この道路名、建物名を識別のために使うのは不都合な点がある。例えば、固有名がない道路もあるし、コンビニやガソリンスタンドの名前のように同じ名前の建物が存在する場合もある。よって、それぞれの道路、建物に識別のために ID をつけることにする。具体的には、それぞれの道路には、$r_1$,$r_2$,$r_3$,$\cdots$、建物には、$b_1$,$b_2$,$b_3$,$\cdots$、のようにつけることにする。ID を入れたそれぞれの構造の例を以下に示す。

```
road r1 "国道 8 号線" w 0 400 700 450
build b1 "くいしん坊" r 195 480
```

例にあるように、ID の追加に加え、属性の位置が変更した。これは地図を再利用するにあたって道路の点数のようにデータ数の変わりやすいものを最後に置くほうが便利だからである。

次に、実際に建物と道路のリンクについて考察していく。図 3 のように (1)、(2) の道路と建物のリンクの方法を考えた。(1) は、道路と建物の間に出入口として新しく道路を作り、道路と建物を繋ぐ方法である。GD モジュールで描画するには建物の座標を始点とし、道路上の点を終点とする道路を実際に描画することになる。今後この道路と建物を繋ぐ道路ををを一般道路と区別するために出入り口という名前で扱う。(2) はリンクする道路上にリンク点 (座標) を作る方法である。この方法は、GD モジュールで描画は行わず、2 点間の経路問題に使用するためにデータとしていれる。

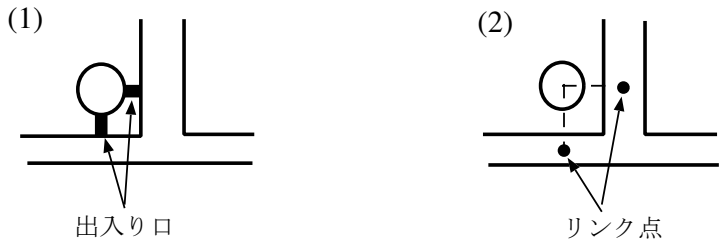


Fig. 3 リンク方法

(1) は実際に出入り口がはっきりしているため、経路を考えるときにわかりやすい。しかし、 (2) のリンク点についた時点で目標の建物に着いたと考えれば (2) の方が有効である。また、道路に接する建物がある場合は、出入口を描画することはできない。よって (2) の方法を採用する。

次に (2) について 2 つのデータ構造を考察する。1 つ目のデータ構造は、基本的な建物の地図データである build 行に、リンク点 ID、リンクする道路の ID、道路の終点の座標を加えることになる。出入り口が複数あるときは、今述べたリンクの地図データの部分を追加していけばよい。例は以下のようになる。

```
build b1 "ガスト" r 20 50 e1 r1 30 40 e2 r2 30 60
```

build から始まり、ID、”建物名”、属性、建物の座標、出入口の ID、リンクする道路の ID となっている。$e_2$ からは、出入口が二つ以上あるときのデータとなる。

二つ目のデータ構造は、build 行に追加する形ではなく、新たに link 行というものを作り、これにリンクのデータをおくという形式である。データ構造は link から始まり、リンク点 ID、リンク点の座標、リンクする建物の ID、リンクする道路の ID を link 行におくことになる。例は以下のようになる。

```
link e1 20 50 b1 r1
link e2 30 60 b1 r2
```

2 つの方法を比較してみると build 行にリンクデータを追加する方法の場合、

- build 行に最小限のデータを追加するため、地図データ全体の量が少なくてすむ
- build 行のデータ量が多くなってしまうため、理解しずらい
- 出入口の数によって build 行のデータが伸び縮みしてしまう

これに対して新たにリンク行を作る方法の場合、

- build 行、link 行ともにデータが少ないため理解しやすい
- データが常に決まった形であるため、処理プログラムを作ることが容易
- 新しく link 行を作るため、全体のデータ量が多くなってしまう

本研究では、データ量が多くなるものの、地図データの処理の行いやすい新たにリンク行を作る方法を採用することにする。

## 5.2 交差点の考察

経路問題を考える上で交差点のデータも必要となってくる。交差点は道路と道路の交差する点であり、経路を考える上で、重要な点である。この交差点の座標をどのよう求め、データ構造はどうするのかについて考察する。まず、データ構造から考えていくが、交差点のデータ構造もリンク点のデータ構造と同様に road 行に追加させる方法と road 行とは別に交差点用のデータ行を作る方法が考えられる。ここで、改めて考えると、リンク点も交差点も道路上のある点 (座標) のデータであるため、同じデータ構造とし、属性等で区別する方法を考えた。この交差点、リンク点のデータを point 行としてまとめることにした。例としては、以下のようになる。

```
point p1 c 20 50 r1 r2
point p2 c 90 30 r4 r6
point p3 l 60 60 b1 r1
```

2 番目の $p_1$、$p_2$、$\cdots$、$p_n$ は、ID を表す。3 番目の c や l は属性を表し、c は交差点を、l はリンク点と意味する。4、5 番目はそれぞれの座標を表し、6、7 番目は、$r_1$ $r_2$ ならば交差する 2 つの道路の ID を表している。$b_1$ $r_1$ ならば、リンクする建物と道路の ID を意味する。

次に、交差点の座標の求め方は以下の 2 つの方法が考えられる。

- 地図データの入力の際に交差点を手動で指定し、座標を求める方法
- road 行の座標より交差点すべての座標を自動計算させる方法

手動で判別する方法は、交差点の場所だけを判別して行うため、交差点であるところのみを計算できるので計算量が少なくなる。しかし、1 つ 1 つを手動で判別して計算するため、手間がかかってしまう。これに対して自動計算で行う方法は、自動計算を行うため、手間はかからない。しかし、 road 行の座標から交差点の座標を計算するため、線分 11 つ 1 つを取り出し、全ての組合せを計算することになり、膨大な計算量となる。また、交差点には地図上で交わっていても実際には交わっていない立体交差点があり、自動計算ではこの立体交差点の識別をすることが難しい。本研究では考察した 2 つの方法それぞれの長所をとり、road 行より交差点の座標を求める支援ツールを作ることにする。この支援ツールで計算された交差点の座標を必要なデータのみ地図データに取り入れることでそれぞれの短所を補った。具体的な支援ツールは考察はで 7 章に示す。

## 5.3 新たな地図データの構成

本研究では、地図データに size 行、build 行、road 行、point 行を使用することにする。それぞれの構成は表 2 のようになる。

| | |
|---|---|
| size 行 | `size` ␣ $x_1$ ␣ $y_1$ ␣ $x_2$ ␣ $y_2$ |
| build 行 | `build` ␣ `ID` ␣ "名称" ␣ 属性 ␣ $x_1$ ␣ $y_1$ |
| road 行 | `road` ␣ `ID` ␣ "名称" ␣ 属性 ␣ $x_1$ ␣ $y_1$ ␣ $x_2$ ␣ $y_2$ $\ldots x_n$ ␣ $y_n$ |
| point 行 (交差点) | `point` ␣ `ID` ␣ "名称" ␣ 属性 ␣ $x_1$ ␣ $y_1$ ␣ $r_a$ ␣ $r_b$ |
| point 行 (リンク点) | `point` ␣ `ID` ␣ "名称" ␣ 属性 ␣ $x_1$ ␣ $y_1$ ␣ $r_a$ ␣$b_a$ |

Table 2 新たな地図データ

size 行は、画像のサイズを設定する行であり、過去の地図データとほぼ同じとなる。ただし座標の位置に違いがあり、左下の頂点の座標を $(x_1, y_1)$ とし、右上の頂点の座標を $(x_2, y_2)$ とした。過去の地図データでは左上の頂点の座標を $(x_1, y_1)$、右下の頂点の座標を $(x_2, y_2)$ となっていた。

build 行は建物のデータを示す。データには建物の ID、名称、属性、位置を示す座標 $(x_1, y_1)$ が入っている。属性はの表 3 ようになる。

| | |
|---|---|
| 飲食店 | r |
| コンビニエンスストアー | c |
| ガソリンスタンド | s |

Table 3 build 行の属性

road 行は道路のデータを示す。データには、道路の ID、名称、属性、始点の座標 $(x_1, y_1)$、中間点の座標、終点の座標 $(x_n, y_m)$ が入っている。属性は表 4n のようになる。

| | |
|---|---|
| 国道 | n |
| その他の道路 | w |
| 路線 | r |

Table 4 road 行の属性

point 行は交差点、建物と道路のリンク点を示す。交差点のデータには、交差点の ID、名称、属性 $c$、交差点の座標 $(x_1, y_1)$、交差点となる道路の ID$r_a$、$r_b$ が入っている。リンク点のデータには、リンク点の ID、名称、属性 $l$、リンク点の座標 $(x_1, y_1)$、リンクする道路と建物の ID$r_a$、$b_a$ が入っている。

現在の地図データと本研究の地図データの違いは ID の追加と point 行による交差点、リンク点の追加である。

これで 2 点間の経路を扱える地図データになった。

## 6 PNG 画像、 HTML への変換

今まで考察してきた地図データを加工するツールとして、自動的に PNG 画像、HTML 形式に変換するツールを作成する。

### 6.1 地図データ

今回使用する地図データを以下の様に設定する。ただし、point 行は示していない。point 行は後の支援ツールで作成する。

```
size 0 0 1000 700
road r1 "国道８号線" n 0 400 700 450 1000 600
```

```
road r2 "" n 100 700 120 405
road r3 "" n 150 415 170 0
road r4 "" w 220 550 228 374 1000 280
road r5 "" w 500 700 530 335
road r6 "" w 900 700 850 350 700 0
road r7 "" w 848 340 700 400
road r8 "" w 740 470 720 315 605 150
road r9 "" w 600 333 580 250 650 0
road r10 "" w 400 150 595 200 800 160
road r11 "" r 0 500 700 650 1000 530
build b1 "担々麺" r 195 480
build b2 "くいしん坊" r 460 450
build b3 "無尽蔵" r 550 420
build b4 "姥姥ラーメン" r 600 420
build b5 "ゆきむら亭" r 770 350
build b6 "一龍" r 180 400
build b7 "ENEOS" s 215 400
build b8 "ENEOS" s 715 375
build b9 "JOMO" s 505 420
build b10 "セブンイレブン" c 400 410
build b11 "柏崎駅" t 220 560
```

## 6.2 プログラムの説明

地図データを読み込み、PNG 画像、HTML 形式に変換する perl プログラムについて説明する。このプログラムは、地図データのテキストを読み込み、直接 PNG 画像を表示し、HTML 形式の文章を指定したテキストファイルに保存するプログラムとなっている。GD lib では、地図の左上の座標が $(0, 0)$ となっているが、一般に座標は左下が $(0, 0)$ であるため、地図データの座標は左下が $(0, 0)$ になるようにしてある。このプログラムは始めに地図データを一行ずつ取り出し、どの行であるか判別する。その後属性によって表 5 のように描画される。

| 種類 | 属性 | 描画されるもの |
|---|---|---|
| 国道 | n | 道幅が 7 ピクセルの灰色の線 |
| その他の道路 | w | 道幅が 4 ピクセルの黒色の線 |
| 路線 | r | 道幅が 4 ピクセルの青色の線 |
| 飲食店 | r | 半径が 8 ピクセルの紫色の円 |
| コンビニエンスストアー | c | 半径 6 ピクセルの緑色の円 |
| ガソリンスタンド | s | 半径 6 ピクセルの青色の円 |
| 駅 | t | 半径 10 ピクセルの黒色の円 |

Table 5 描画例

また、HTML 形式のクリッカブルマップを利用し、build 行を飲食店の詳細情報にジャンプさせるため、以下のような変換が行われる。

地図データの

```
build b1 "担々麺" re 195 480
```

を読み込むと

```
<area shape="circle" coords="195,220,8" href="#b1 " alt="担々麺">
```

と変換される。

この加工ツールで変換した PNG 画像を図 4 に示す。

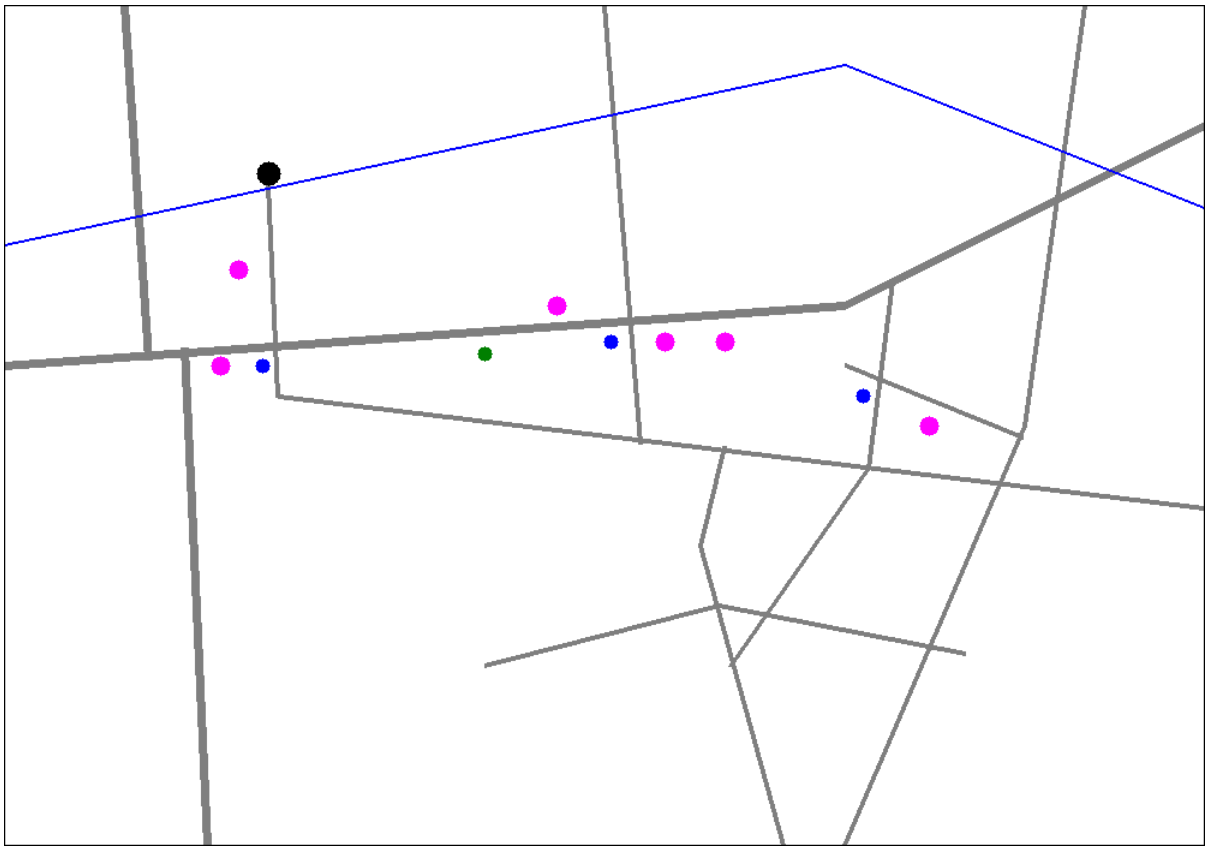

Fig. 4　出力結果

## 7　point 行の計算

road 行から point 行を計算する支援ツールを作成する。まず、交差点を求める方法から考える。

### 7.1　2 つの線分からの交点の計算方法

ベクトルと行列式を使い、交差点を求める。実際の方法は、次のようになる。

図 5 のように線分 A、B とすると、線分 A は次のような式で示すことができる。

$$\overrightarrow{\mathrm{OP}} = \overrightarrow{\mathrm{OP_1}} + \overrightarrow{\mathrm{P_1P}} = \overrightarrow{\mathrm{OP_1}} + t\overrightarrow{\mathrm{P_1P_2}} = \overrightarrow{\mathrm{OP}} + t(\overrightarrow{\mathrm{OP_2}} - \overrightarrow{\mathrm{OP_1}})$$

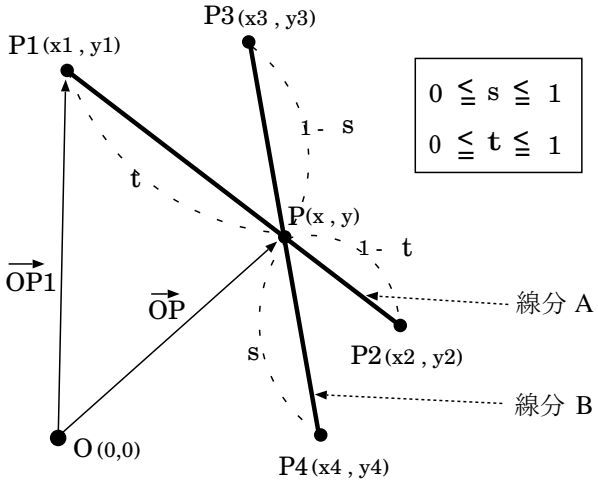


Fig. 5 線分 A、線分 B

$$= \begin{bmatrix} x_1 \\ y_1 \end{bmatrix} + t \begin{bmatrix} x_2 - x_1 \\ y_2 - y_1 \end{bmatrix} = \begin{bmatrix} x_1 + t(x_2 - x_1) \\ y_1 + t(y_2 - y_1) \end{bmatrix}$$

よって

$$\begin{cases} x = x_1 + t(x_2 - x_1) \\ y = y_1 + t(y_2 - y_1) \end{cases} \quad (0 \leq t \leq 1) \tag{1}$$

また、同様にして線分 B は

$$\begin{cases} x = x_3 + s(x_4 - x_3) \\ y = y_3 + s(y_4 - y_3) \end{cases} \quad (0 \leq s \leq 1) \tag{2}$$

となる。よって、この 2 つの線分の交点は (1) 式との (2) 式連立方程式で求めることができる。

$$\begin{cases} x_1 + t(x_2 - x_1) = x_3 + s(x_4 - x_3) \\ y_1 + t(y_2 - y_1) = y_3 + s(y_4 - y_3) \end{cases} \tag{3}$$

この (3) 式の連立方程式を $s$ と $t$ について解くとその交点は求めることができる。

$$\begin{cases} (x_2 - x_1)t - (x_4 - x_3)s = x_3 - x_1 \\ (y_2 - y_1)t - (y_4 - y_3)s = y_3 - y_1 \end{cases} \tag{4}$$

式 (4) を行列に変形すると、

$$\underbrace{\begin{bmatrix} x_2 - x_1 & x_3 - x_4 \\ y_2 - y_1 & y_3 - y_4 \end{bmatrix}}_{A \text{ と置く}} \begin{bmatrix} t \\ s \end{bmatrix} = \begin{bmatrix} x_3 - x_1 \\ y_3 - y_1 \end{bmatrix} \tag{5}$$

となる。この連立方程式が唯一の解を持つ条件は、

$$|A| = (x_2 - x_1)(y_3 - y_4) - (x_3 - x_4)(y_2 - y_1) \neq 0 \tag{6}$$

(6) 式は、$\vec{0}$ でなく、平行でない条件も意味する。(6) 式が満たされた場合、(5) 式を $t$、$s$ について解くと、

$$\begin{bmatrix} t \\ s \end{bmatrix} = A^{-1} \begin{bmatrix} x_3 - x_1 \\ y_3 - y_1 \end{bmatrix}$$

$$= \frac{1}{|A|} \begin{bmatrix} y_3 - y_4 & x_4 - x_3 \\ y_1 - y_2 & x_2 - x_1 \end{bmatrix} \begin{bmatrix} x_3 - x_1 \\ y_3 - y_1 \end{bmatrix} \tag{7}$$

(7) 式のようにすれば、(5) 式を解くことができる。

しかし、ここまでの交点を求める式は 直線 $P_1P_2$ と直線 $P_3P_4$ としての交点を求めるもので、線分としての交点を求めたものとはならない。よって、これを線分の交点かどうか判別するには、

$$0 \leq t \leq 1 \text{ かつ } 0 \leq s \leq 1 \tag{8}$$

という条件を満たせば判別できることになる。(8) 式を満たさないと、2 つの線分は交わらない。よってこの条件を満たすことで 2 つの線分の交点を求めることができる。

### 7.2 point 行の計算ツールのプログラムと実行結果

point 行の計算ツールのプログラムについて簡単に説明する。前の節で述べた方法を折れ線で扱えるようにし、プログラムは perl を使い作成した。あらかじめ入力テキストとして size 行、build 行、road 行を書き込んでおき、このデータを読み込んで交差点の座標を計算し、ID、交差点名、属性以外の point 行を出力する。座標は正数値として表示する。例は以下の通りである。

入力

```
road r1 "国道 8 号線" n 0 400 700 450 1000 600
road r2 "" n 100 700 120 405
road r100 "" w 900 700 0 500
```

出力

```
point p1 "" c 120 409 r1 r2
point p2 "" c 607 443 r1 r100
```

地図データの座標の参考にするために、図 6 を示す。画像の左上の座標を $(0, 0)$ とし、右上の座標が (1000,700) となる。

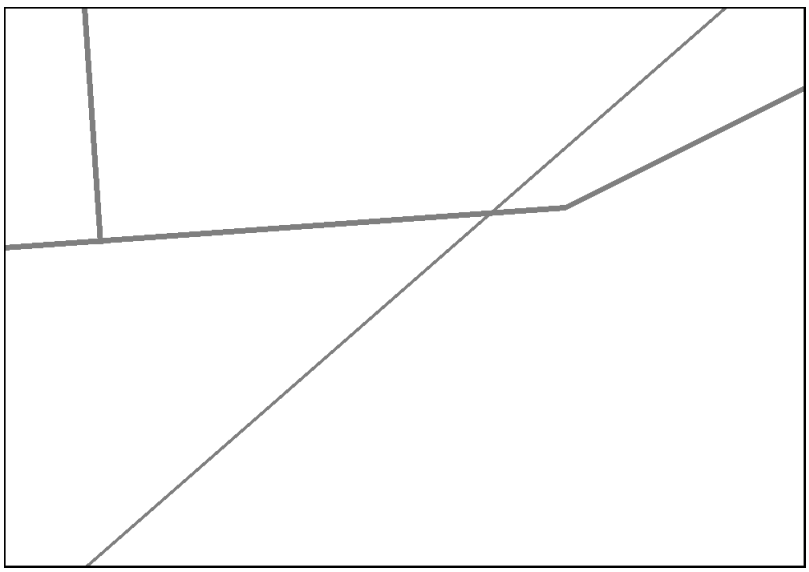

Fig. 6 座標参考の図

この生成された point 行に ID、交差点名、属性の必要な情報を入れ、既にある地図データに追加すればよい。この必要な情報を入れた本研究で使用する point 行は以下の通りである。

```
point p1 "" c 120 409 r1 r2
point p2 "" c 150 411 r1 r3
point p3 "" c 226 416 r1 r4
point p4 "" c 522 437 r1 r5
point p5 "" c 877 538 r1 r6
```

```
point p6 "" c 740 470 r1 r8
point p7 "" c 922 561 r1 r11
point p8 "" c 112 524 r2 r11
point p9 "" c 530 337 r4 r5
point p10 "" c 829 301 r4 r6
point p11 "" c 719 314 r4 r8
point p12 "" c 599 329 r4 r9
point p13 "" c 220 547 r4 r11
point p14 "" c 508 609 r5 r11
point p15 "" c 846 341 r6 r7
point p16 "" c 771 166 r6 r10
point p17 "" c 882 577 r6 r11
point p18 "" c 729 388 r7 r8
point p19 "" c 607 153 r8 r9
point p20 "" c 634 192 r8 r10
point p21 "" c 594 200 r9 r10
```

### 7.3 point 行の拡張

point 行は今のところデータとしてしか扱われていない。そこで、point 行についてさらに考察していく。地図を作成するにあたって必要、またはあると便利なものを考えてみた。その結果、以下の 2 つのことを考えた。

- 信号機の描画
- バス停の描画

信号機は、地図を見る上で 1 つの目印となり、重要な役割をしている。また、既に交差点の地図データがあるため、この交差点データを画像化すればよいことになる。しかし、全ての交差点には信号機があるわけではない。よって信号機のある交差点には属性で ”s” とつけ、区別することにする。本研究の地図データでは、$p_1$、$p_2$、$p_3$、$p_4$、$p_6$、$p_7$、$p_8$、$p_9$、$p_{12}$、$p_{14}$、$p_{15}$、$p_{16}$ が信号機になる。

バス停は、point 行が一つの地点 (座標) を示すことを利用した。バス停にも”b” と属性を与え、バス停のマークを表示する。ただバス停は地図上に表示させるだけでは、ほとんど意味はない。よって、クリッカブルマップを使い、バス停をクリックすることでバスの時刻表にジャンプするしくみにすれば便利だと考えた。例えば以下の通りである。

```
point p22 "穂波町中央停留所" b 300 370 r4
point p23 "田中停留所" b 480 350 r4
point p24 "" b 800 310 r4
point p25 "" b 730 100 r8
point p26 "" b 515 650 r7
point p27 "" b 490 180 r5
```

この信号機とバス停の地図データを既にあった地図データに追加する。

## 7.4 画像、HTML 形式変換の加工ツールの拡張

作成した PNG 画像、 HTML 形式変換の加工ツールを信号機、バス停が表示できるように拡張する。プログラムについて簡単に説明するが、まず地図データを 1 行ずつ取り出し、行の始めが「point」ならば交差点、point 行と判別される。その後、属性ごとに区別され、属性が”s”の信号機ありの交差点、または”b”のバス停ならば、図 7 のような記号が描画される。

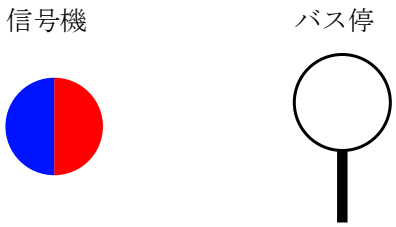


Fig. 7 信号機とバス停の描画例

また、バス停はクリッカブルマップを利用し、バスの時刻情報にジャンプさせるため、以下のような HTML 形式に変換するプログラムに拡張した。

地図データの

```
point p17 "穂波町中央停留所" b 300 370 r4
```

を読み込むと

```
<area shape="circle" coords="300,314,8" href="#p17 "
                                        alt="穂波町中央停留所">
```

と変換される。

拡張された加工ツールを使い、実際に PNG 画像を生成した。この画像には建物の名前が入っておらず、建物の識別がつかない。しかし、地図データとしては、建物名等のデータがある。そのデータを使えば、建物名を表示することはできる。PNG 画像の出力結果は図 8 となる。

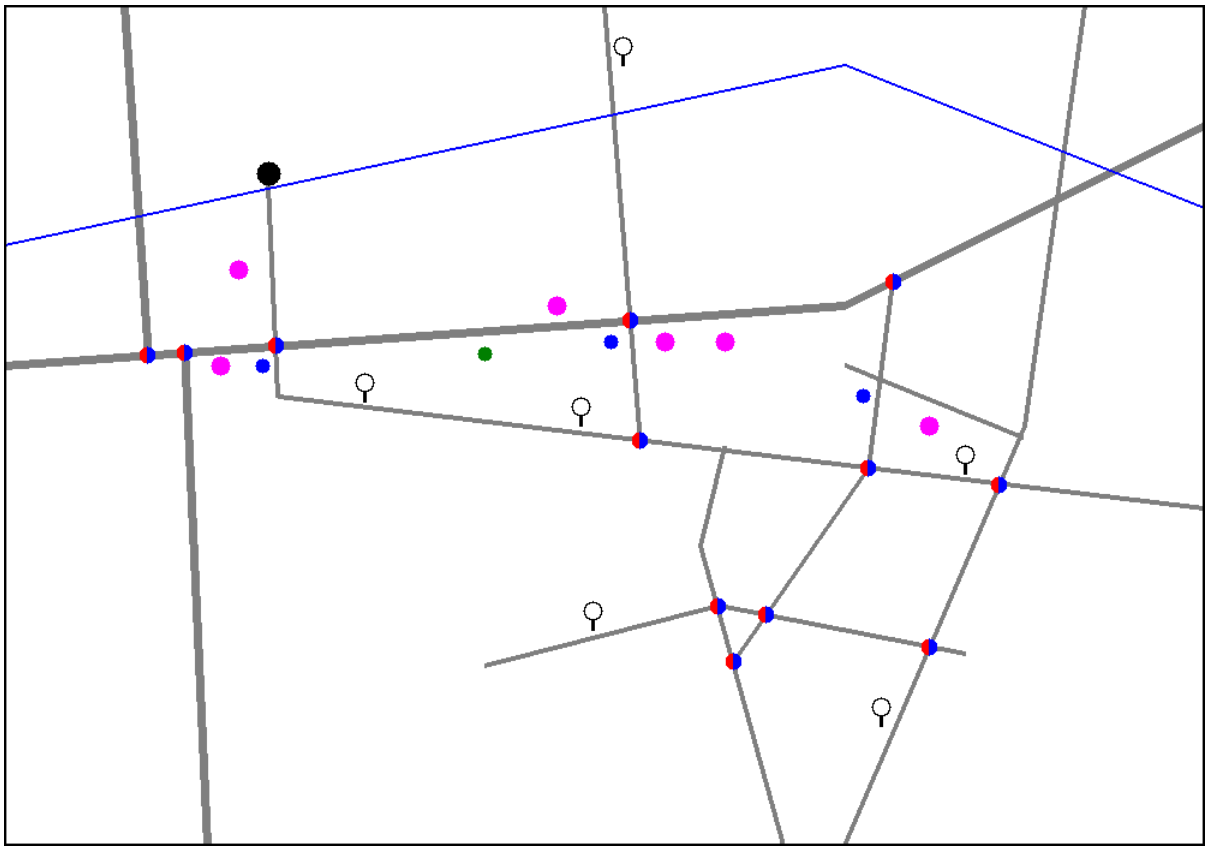

Fig. 8 出力結果

## 8 まとめ

本研究では、perl の GD モジュールを使い、地図データから直接 PNG 画像形式への変換、2 点間の経路を考慮した地図データの作成について研究した。

地図データからの PNG 画像形式への変換は、2004 年度にこの研究を行った齊藤氏 [1)] が使用した SVG 画像形式の短所を改善することに取り組んだ。SVG 画像形式は対応するブラウザが少ないため、ビューワーソフトを使う必要があり、この改善として多くの WWW

ブラウザが対応している PNG 画像形式を直接出力できる GD モジュールを用いた。

また、2 点間の経路を考慮した地図データの考察では、まず、現在の地図データに ID を与え、建物、道路 1 つ 1 つを判別し、取り出せるようにした。また、リンク点、交差点を point 行とし地図データに取り入れ、交差点の座標は perl プログラムで支援ツールを作ることで自動計算させた。これで 2 点間の経路を扱える地図データができたわけだが実際に経路の自動計算は行えず、この後の課題となった。また、point 行はバス停のデータとしても拡張し、HTML 形式でバスの時刻表の情報とリンクさせるような仕組みを作成した。

地図データを PNG 画像、HTML 形式に変換する加工ツールも perl プログラムで作成し、このツールで自動的に PNG 画像の出力、HTML 形式のテキストファイルを作成できるようになった。

今後の課題としては、地図データから 2 点間の経路を計算することである。本研究で作成した地図データを実際に何らかの形で 2 点間の経路を計算するところまでは至ってないため、アルゴリズムなどを使い実際に経路を計算することが今後の課題となる。

## 参考文献

[1] 齊藤 真理子、”再利用可能な地図データの考察とWWW用変換ツールの作成について”、新潟工科大学情報電子工学科卒業論文 (2005)

[2] Lincoln D.Stein、GD モジュール (ver 2.28) 付属マニュアル

[3] 石原繋、浅野重初、”線形代数”(裳華房)