標高タイルを利用した等高線作成 Web サイト「Web 等高線メーカー」の開発とそのアルゴリズム

谷 謙二（埼玉大学）

Ⅰ　はじめに

地形の起伏を地図に表現するには，段彩図や陰影，ケバなどさまざまな方法がある。そうした中で等高線による表現は，複雑な地形を一色で表現でき，標高値を正確に読み取ることができることから，地形図などで広く用いられている。従来，等高線のみを描くには，紙の地形図の等高線をトレースすることが多かったと思われるが，現在では GIS と DEM(Digital Elevation Model)データの普及により，PC 上で処理できるようになった。一般に利用できる DEM として，日本国内に関しては国土地理院の基盤地図情報 5m/10m メッシュ，全球レベルでは ASTER GDEM，SRTM，ETOPO などがある。

これらのデータを処理する GIS ソフトで簡便に使えるものとしては，杉本智彦氏による「カシミール 3D」や，筆者作成の「MANDARA」などがある。しかし，等高線の必要な範囲と使用する DEM の解像度がうまく一致しないと，きれいな等高線を取得することができない。たとえば関東地方全域の等高線を見るために，10m メッシュ標高データから等高線データを作るとすると，解像度が高すぎ，大量のデータのダウンロードが必要になってしまう。

そこで，任意の範囲を指定し，適切な解像度の DEM データを取得して等高線を取得することのできるシステムが求められる。このうち任意の範囲を指定するという点については，Google Maps API 等の Web 地図 API サービスの普及により，容易に自システムに組み込むことができるようになった。その一方で，任意の地域で適切な解像度での DEM データを簡便に入手することは難しかった。しかし，2013 年 10 月から，国土地理院の地理院地図サイトにおいて，「標高タイル」が公開された。これは，Web 地図サービスで用いられる球面メルカトル図法に基づくタイル画像（256×256 ピクセル）を，256×256 個の標高値に置き換えた，カンマ区切りテキストファイルからなるデータである。

この標高タイルは 10m メッシュ標高データを線形的に平滑化して得られた数値とされており，ズームレベル 0～14 までが提供されている（北村ほか 2014）。10m メッシュ標高データは 1/2.5 万地形図の等高線から生成されており，10m より細かい標高差の抽出は適当でない。しかしこのデータの提供によって，日本国内であれば前述の任意の範囲で適切な解像度の DEM データを取得することが可能になった。すなわち，Web 地図において関東地方全域を表示した状態と，埼玉県だけを表示した状態では，地域的な広がりでは差があるものの，画面上の画素数は同じであり，標高タイルの情報量もそれに対応して同じになる。

そこで筆者は，Web ブラウザを用いて簡便に日本の任意の地域の等値線を任意の標高間隔で取得し，Google マップ上に表示したり，KML に出力したりすることのできる Web サイト「Web 等高線メーカー」を開発し，2014 年 4 月に公開した（http://ktgis.net/lab/etc/webcontour/）。本稿では，等高線データを作成するアルゴリズム，Web サイトの機能および作図例について報告する。

開発にあたっては，簡便に等高線を取得するため，ブラウザ上で動作する Web 地図サービスとして Google Maps API を用いた。ただし，OpenLayers や Leaflet といったライブラリでも同様の機能は実装できると考えられる。等高線への変換は，ローカル PC のブラウザで動作する JavaScript によるプログラムを作成して行った。

Ⅱ　処理過程

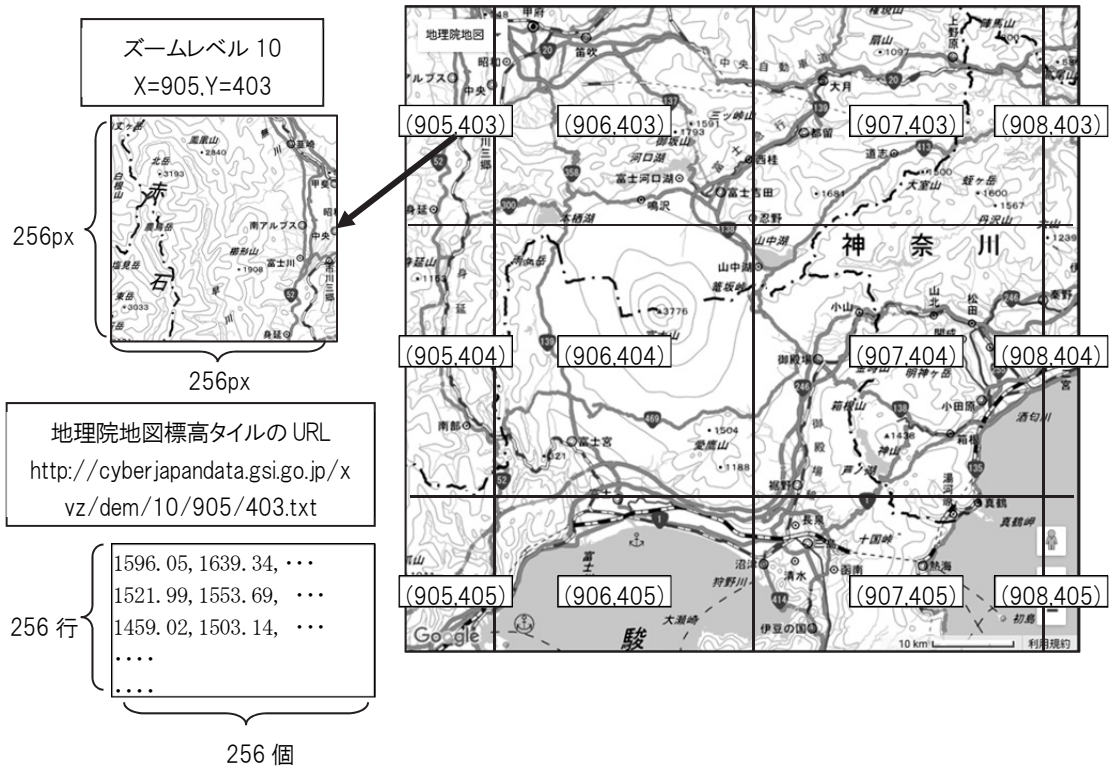


図 1 ズームレベル 10 の座標と画像，標高タイル

図 1 は，Google Maps JavaScript API で表示したズームレベル 10 の富士山付近の地図（地理院地図の標準地図）である。こうした Web 地図サービスは，タイルマップ形式で地図タイルをズームレベル応じて 256×256 ピクセルのサイズに分割して保持しており [1]，図では 12 枚のタイルが含まれている。ここでの座標の原点は球面メルカトルの北西端となる。通常の地図であれば，画像ファイルをサーバから取得して画面上に表示するが，地理院地図の標高タイルは拡張子 txt のカンマ区切りのテキストファイルとなっており，小数点以下 2 桁の数値で，縦横 256 個ずつ並んだファイルとなっている。

表示されている範囲の等高線を取得するには，大まかに以下の 3 つの段階からなる。第 1 段階：表示範囲の標高タイルを取得し，2 次元配列に標高値を入れる。第 2 段階：標高値から等高線ベクトルデータを作成する。第 3 段階：作成したデータを地図上に表示する。

この流れを順番に解説していく。まず第 1 段階の標高タイルの取得である。Google Maps API での通常の地図画像の表示であれば，表示されている範囲の地図タイルが非同期にダウンロードされ，表示される。一方，等高線を作成するに当たっては，非同期にダウンロードされるよりも，必要なファイルを同期モードでダウンロードし，メッシュ標高データとした方が確実に接続した等高線を取得できる。そこで，ここでは JavaScript の XMLHttpRequest オブジェクトを使って同期的に地理院地図のサーバからファイルをダウンロードすることにした。ダウンロードするファイルを特定するには，表示されている地図領域の北西端と南東端の地図タイル座標を求めておく必要がある。表示範囲の緯度経度(lat, lon)とズームレベル(zoom)は，Google Maps API から取得でき，その値を元にタイル座標(X, Y)は次式で求められる。

$$X = 2^{zoom-1}\left(\frac{lon}{180} + 1\right)$$

1) 地図タイルの形式などについては小清水・高桑（2013），北村ほか(2014)に詳しい。

$$Y = \frac{2^{zoom-1}}{\pi}\left(-\log\left(\tan\left(\frac{\pi\left(45+\frac{lat}{2}\right)}{180}\right)\right)+\pi\right)$$

この場合は北西端が(0, 0)である。タイルをファイル単位に特定する場合は，X, Y の小数点を切り捨てる。タイルマップ画像内での座標を求める場合は，小数点以下を 256 倍して求めることができる。

このようにして標高タイルのファイルを取得した後，カンマ区切りの標高値文字列を分解し，二次元配列に標高値を設定した。二次元配列の縦横のサイズは，ブラウザで表示されている Google マップの地図領域縦横のピクセル数に等しい。

第 2 段階の標高値から等高線ベクトルデータを作成する手順については，次章で詳しく解説するので，ここでは第 3 段階の，作成した等高線データの処理を述べる。

作成した等高線データは，二次元配列の範囲の XY 座標列となっている。このデータから等高線だけを表示する場合は，そのままの座標を使って示すことができるが，Google マップ上に表示するには緯度経度に変換する必要がある。作成された XY 座標のデータは，メルカトル図法で投影された座標系の中の局地的な座標である。当該座標を(X, Y)，取得領域北西端のタイル位置を(Xt,Yt)，ズームレベルを zoom とすると，緯度経度(lat, lon) は次式で求められる。これはメルカトル投影の逆変換である。

$$\mathrm{lon} = 180\left(\frac{\left(Xt+\frac{X}{256}\right)}{2^{zoom-1}}-1\right)$$

$$\mathrm{lat} = \frac{360}{\pi}\mathrm{atan}(e^{-My}) - 90$$

$$\text{ただし}\quad My = \pi\left(\frac{\left(Yt+\frac{Y}{256}\right)}{2^{zoom-1}}-1\right)$$

この式を用いて XY 座標を緯度経度に変換後，Google Maps API の google.maps.Polyline オブジェクトに設定することで，Google マップ上に表示される。

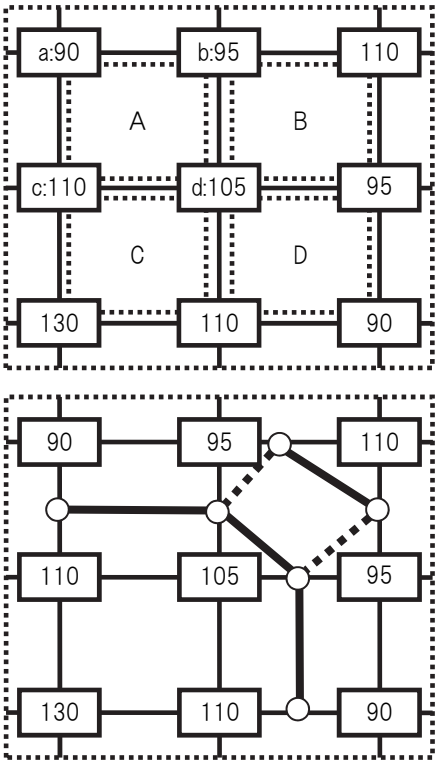


図 2 メッシュでの等高線の引き方
（100m 等高線の場合）

Ⅲ　等高線取得アルゴリズム

1．等高線の取得

前章で述べた処理段階のうち，第 1 段階はサーバとのアクセス，第 3 段階は Google Maps API 側の処理が中心となるため，開発側の工夫による処理速度の向上の余地は小さい。一方，第 2 段階の二次元配列の標高値から等高線ベクトルデータを作成する段階では，アルゴリズムの改良による処理速度の向上が期待できる。本章では，第 2 段階で用いたアルゴリズムについて述べる。なおここで解説する方法は，筆者開発の GIS ソフト「MANDARA」の等値線モードや DEM データからの等高線取得などで用いられているアルゴリズム（谷 2011,p.74-75）を JavaScript に移植したものである。

配列の標高値から等高線ベクトルデータを作成する処理は，次のような工程からなる。①配列から取得したい等高線のかかるメッシュを抽出，②抽出さ

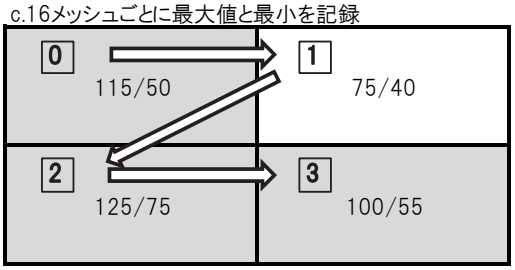
a.元のメッシュ数値

60	60	55	60	70	65	50	40
75	70	65	80	75	70	60	50
90	80	75	65	60	65	60	45
95	100	80	60	50	55	50	45
110	115	90	80	75	70	60	55
110	120	100	90	85	75	75	65
115	125	110	100	95	80	70	60
120	125	125	105	100	90	75	70

b.4メッシュごとに最大値と最小値を記録

0 90/55	1 80/55	4 75/50	5 60/40
2 115/75	3 90/50	6 75/50	7 60/45
8 125/90	9 110/75	12 95/60	13 75/55
10 125/110	11 125/95	14 100/70	15 75/60

c.16メッシュごとに最大値と最小を記録



d.全体の最大値と最小値を記録

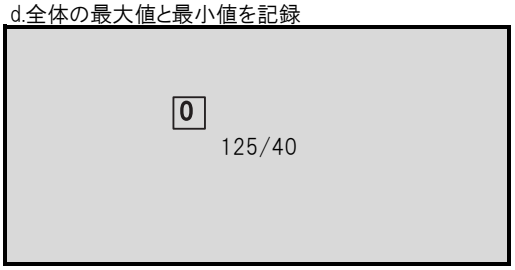


図 3 四分木を使った効率的な標高値のチェック

左側数値が最大値，右側数値は最小値，網掛け部分は 100m にかかるメッシュ，
四角内の数字はモートン番号

れたメッシュでの等高線線分の算出，③等高線線分の結合。この一連の処理を取得する標高の数だけ繰り返すことになる。

①に先だって，②抽出されたメッシュでの等高線線分の算出について説明する。図 2 は模式的に示した 3×3 の格子点からなる標高メッシュデータであり，上側の図に 100m の等高線を引くものとする。100m 等高線を引くためには，隣接する格子点の標高値との間に 100m が含まれるかどうかをチェックする。A のエリア場合，ac，bd 間に 100m が含まれる。従って，ac，bd 間に等高線の線分の端を設定できる。線分の位置は，100m と各格子点の標高値との差をとって按分して決めるが，この場合は ac，bd のちょうど中間となる。同様に，B，C，D エリアとチェックし，線分を追加していく。ここで B のエリアの場合，線分は 4 本引くことが可能だが，4 本すべて引くと等高線に枝分かれが生じて不都合である。そこで本システムでは，図 2 下側のように，4 点が可能な場合には左上から右下に向けての線分を選択するよう設定している。

このように走査していき，100m 等高線の線分をすべて取得する。ここで取得した線分を，つながるように結合して連続した線にまとめるのが③の工程であり，さらにその座標列を地図上に表示するのが，前章の第 3 段階となる。

2．線形四分木を用いた高速化

ここで図 2 の C エリアを見ると，100m の等高線にはかかっていない。また図 2 の範囲で 200m の等高線を引いた場合は，どの格子点もかからないことになる。このように，等高線を引く際，大部分のメッシュは取得したい等高線の標高値から外れている。領域全体にわたって，くまなく同一の等高線が引かれるというケースは少ないと考えられる。したがって，すべてのメッシュの標高値をチェックして線分を作成するのではなく，必要な範囲のみをチェックすることで処理の高速化をはかることができる。これが前節①の処理となる。

効率的な処理のため，本システムでは四分木を用いる。図 3a は 8×8 のメッシュの仮想的な標高値を示している。ここに 100m の等高線を引く場合，すべてのメッシュをチェックすると，(8-1)×(8-1)で 49 箇所のチェックが必要となるが，実際 100m 等高線にかかる箇所は少ない。

階層	d(0)	c(1)				b(2)									
モートン順序	0	0	1	2	3	0	1	2	3	4	5	6	7	・・・	16
配列の位置	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14

図 4 線形四分木の配置

()内は階層番号，網掛けは階層ごとの先頭位置

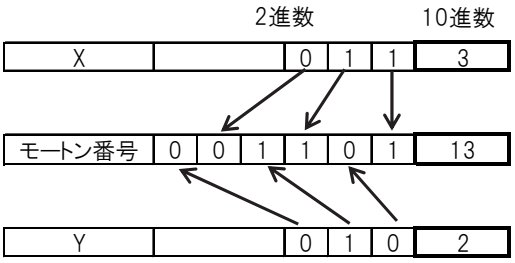


図 5 モートン番号と座標との関係

そこでまず a のメッシュを 4 メッシュごとにまとめたメッシュ b を作成し，それぞれのメッシュに内部の標高の最大値／最小値を持たせる。ただし，等値線を引く際に隣接するメッシュの情報も必要なことから，重複を持たせて a の(2+1)×(2+1)メッシュ分の範囲で a の標高から最大値／最小値を求める。さらに b を 4 つずつにまとめたメッシュ c を作成し，最大値／最小値を求める。この場合はメッシュ b の値を参照すればよく，重複を持たせる必要はない。同様に c から d を作成して全体の最大値と最小値を持たせる。

このような階層的なデータから，目的とするメッシュを探索する手順は次のようになる。ここでは 100m 等高線を引くとすると，まず d のメッシュの最大／最小値を比較し，100 が間に入ることを確認する。次に c のメッシュに移り，0 番のメッシュで間に 100 が入ることを確認する。さらに b のメッシュに移り 2 番で 100 が入ることを確認する。そこで a のメッシュの当該領域を前節で述べた方法で調べ，100m 等高線の線分を記録する。この方法で階層的に遡及して調べていけば，たとえば c のメッシュで 1 番のエリアに 100m 等高線が存在しないことが判別できるので，それ以下の階層のチェックが不要となって処理を高速化できる。a の階層では 20 箇所について図 2 のチェックを行うだけで済み，b,c,d では合わせて 9 箇所であるが，これは最大値と最小値をチェックするだけなので手数はかからない。

このアルゴリズムを実際のプログラムに組み込むには工夫が必要である。それは，図 3 のような複数の 2 次元配列を対象として処理を行うことが難しいためである。そこで各メッシュに対しモートン順序（Morton 1966）と呼ばれる順序で番号をふる（図 3 □内の数字)。この番号は Z を描くように再帰的に配列しており Z-order とも呼ばれる。この番号は興味深い性質があり，c 階層の番号を 4 倍にすると，b 階層の番号の位置を参照することができる。逆に b 階層の番号を 1/4 にすると, c の階層の番号を参照することができる。

さらにモートン番号の順序でデータを並べることで，もともと二次元配列だったデータを一次元配列に収め，線形四分木とすることができる（図 4)。n 階層（d 階層の場合 n=0）のセル数は 4^n なので，各階層の配列中の先頭位置 p は等比数列の和となり，次式で求められる。

$$p = \frac{4^n - 1}{3}$$

こうして一次元配列に四分木データをもたせることができたが, モートン番号と各階層の座標 (X, Y) との関係を示したものが図 5 である。ここでは図 3b において (3 ,2) のセルのモートン順序を求めてみる。まず X，Y それぞれの値を 2 進数で表記すると，X は 011，Y は 010 となる。次に，それぞれの 2 進数をビットごとに Y, X の順に交互に並べると, 001101 となり，これを 10 進数で示すと 13 のモートン番号になる。これを逆変換すれば, モートン番号から XY 座標を求めることができる。

このモートン順序を利用した線形四分木を用いることで，階層間での標高値のチェックを効率的に行うことができる。

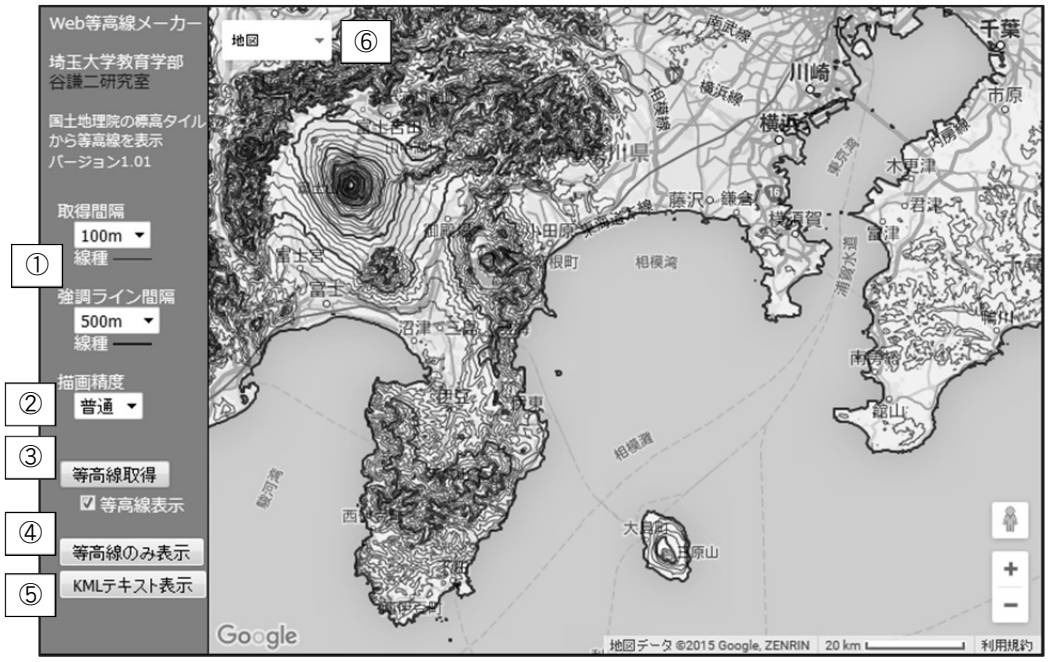


図 6「Web 等高線メーカー」サイトの画面

３．実際の組み込み

標高メッシュデータを取得してから，線分を求めるまでの工程を実際の JavaScript コードを示しながらみていく。

まず階層の数を決定する。図 3 では，元の 8×8 メッシュから 4 メッシュにつき新しく 1 メッシュ作成しているが，実際の処理では Google マップで表示されている範囲を対象とする。そのためデスクトップ画面で最大表示にすると，横 2000 ピクセル，縦 1000 ピクセルほどの広さとなり，4 メッシュごとに 1 メッシュを作成すると階層数が多くなりすぎる。そこでここでは，元のメッシュを縦または横方向で 20 メッシュ分を最低限含むサイズを四分木の最下層に設定した。

そこから線形四分木配列を作成する JavaScript のコードは付録 1 に示した。まず最下層について，領域内の最高地点と最低地点を求め，さらに領域の四隅の位置情報を記憶しておく。この最大値・最小値データをもとに，上位階層のセルも順番に求めていくことができる。付録 2 のプログラムでは，ここで作成された線形四分木配列を探索し，必要な等高線にかかる最下層の線形四分木配列の位置を記録していく。関数 Get_Quad_MeshCell を呼び出すと，再帰処理により当該位置が取得される。

取得された四分位最下層の関連メッシュから，さらに元のメッシュ配列データを参照し，実際にメッシュ間に等高線を引ける場合に呼び出されるのが付録 3 の Get_Mesh_Contour 関数である。これによって，等高線の線分データが記録され，Ⅱ章の第 3 段階の処理に引き継がれる。

Ⅳ　機能と表示例

次に「Web 等高線メーカー」サイト全体の機能と表示例を紹介する。図 6 は画面構成を示したものであり，左側に操作パネル，右側に Google マップと取得した等高線が表示される。図では既に等高線を取得して表示した状態である。

取得するためには，まず右側画面で取得したい範囲の地図を表示させる。その際ズームレベルが 14 を超えると等高線は取得できない。範囲を設定したら，①の欄で等高線取得間隔を設定する。さらにその中で強調する間隔も設定できるようになっており，図では 100m 間隔で取得，500m 間隔で強調表示とな

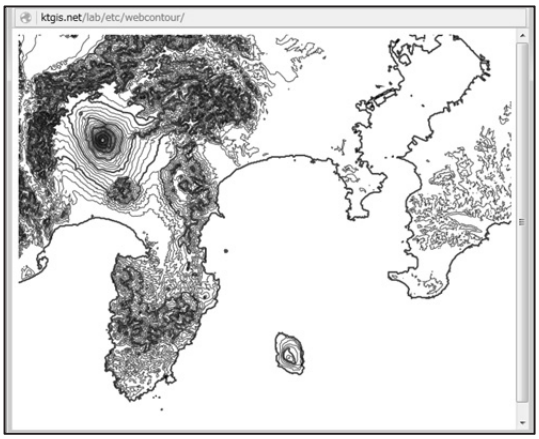


図 7 等高線のみ表示

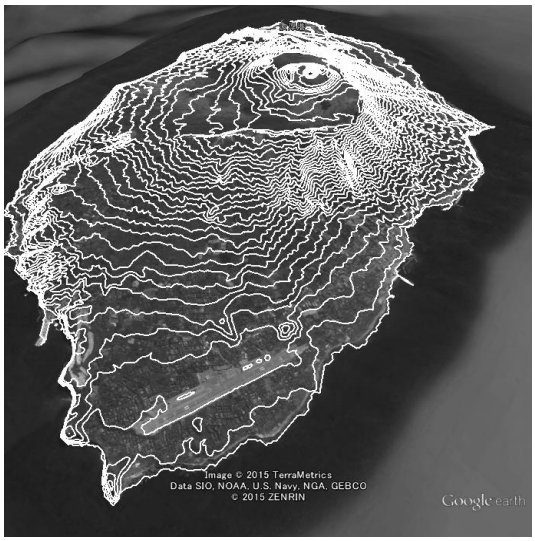


図 8 kml 形式での出力（伊豆大島）

っている。また，それぞれの線種を指定できる。②では描画の精度を指定する。これらの設定を行った後に③の「等高線取得」をクリックすると，指定した等高線がⅡ，Ⅲ章の方法で取得され，右側の地図上にオーバーレイされる。⑥では背景地図を変更することができ，通常の Google マップの他，航空写真，地理院地図，地理院地図色別標高図などを背景に設定できる。

取得した等高線のみを表示したい場合は⑦の「等高線のみ表示」を行う。HTML5 のグラフィックを描画する canvas 要素を用いて別画面に等高線のみ描画され（図 7），画像として保存することができる。さらに外部の出力形式として，Google Earth で表示できる kml 形式をサポートしている。⑧をクリックすると kml 形式のテキストデータが表示されるので，コピーしてテキストエディタなどに貼り付け，保存する。図 8 は伊豆大島の等高線を Google Earth で表示したものである。

図 9 は伊豆諸島の島々を 10m 間隔の等高線で描いたものである。カルデラなど火山地形の特徴を等高線の様子から読み取ることができる。

Ⅴ　おわりに

本稿では，指定した範囲の等高線を表示する Web サイト「Web 等高線メーカー」について，その等高線取得アルゴリズムと機能を解説した。このシステムは国土地理院の地理院地図での標高タイルの公開により可能となったが，そのために日本国内しか表示できない。SRTM や ASTER GDEM など全球をカバーする DEM データもあるので，それらが標高タイルと同様の形式で整備されれば，世界中で表示することができる。そうしたデータの整備も今後検討されるべきだろう。

作成した等高線の活用方法については，地理教育での等高線学習や地形模型作成などでの活用が考えられる。堀・早川(2005)は，弁当パックを利用した立体地形模型を作成している。そうした模型作成に使われる等高線図のベースマップとして，本サイトで作成された等高線画像が有効に活用できるだろう。

本研究の内容は 2015 年日本地理学会春季学術大会にて発表した。研究に際しては科研費 21700854 を使用した。

文　献

Morton,G.M. 1966. A Computer Oriented Geodetic Data Base; and a New Technique in File Sequencing. *Technical Report*, Ottawa, Canada: IBM Ltd.

北村京子・小島脩平・打上真一・神田洋史・藤村英範 2014. 地理院地図の公開. 国土地理院時報 125 : 53-57.

小清水寛・高桑紀之 2013. WEB ブラウザ上での表示に適した配信地図データの作成技法とその応用.

国土地理院時報 123：105-115.
谷 謙二 2011.『フリーGIS ソフト MANDARA パーフェクトマスター』古今書院.
堀 真希子・早川由紀夫 2005. 弁当パック立体模型を使った授業実践. 群馬大学教育実践研究 22：57-66.

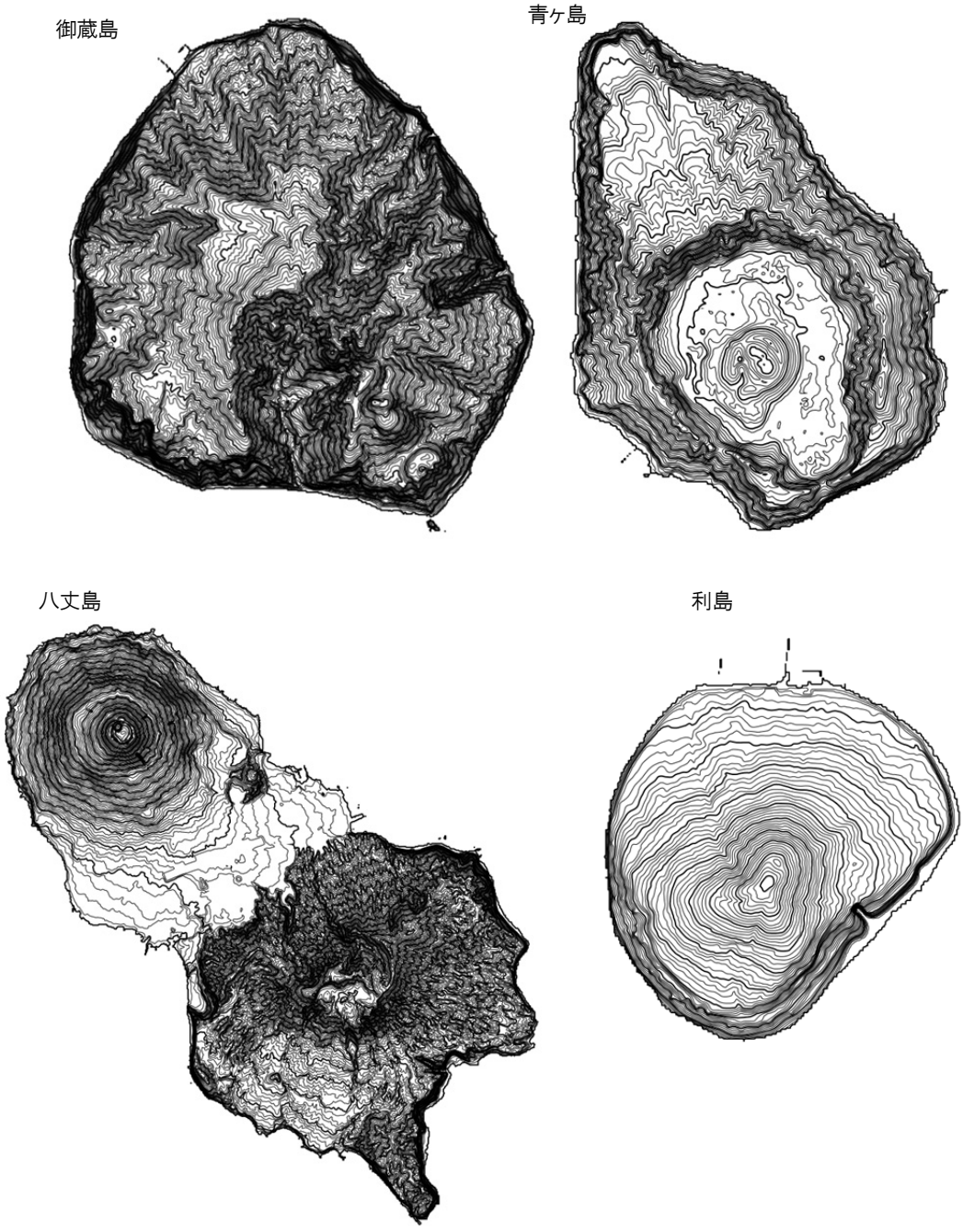


図9 Web 等高線メーカーで作成した伊豆諸島の島々の 10m 間隔等高線

Development and Algorithm of a Web Site to Draw Contour Lines of Japan Using Altitude Tile Data

Kenji TANI

Dept. Geography, Saitama Univ.

付録1　標高メッシュから線形四分木配列を作成

```
var Quad_Mesh_Info = function (Positon, Max, Min, LackF) {
   /// <signature>
   /// <summary>四分木データの配列に入れる情報</summary>
   /// <param name="Positon" >座標（rectinfo）</param>
   /// <param name="Max" >最高地点標高</param>
   /// <param name="Min" >最低地点標高</param>
   /// <param name="LackF" >欠損値の場合true</param>
   /// </signature>
   this.Position = Positon;//rectinfo
   this.Max = Max;
   this.Min = Min;
   this.LackF = LackF;
}

var Quad_MeshData = new Array;

function Set_MeshQuadTree(Mesh, xw, yw, max_PartitiopnLevel) {
   /// <signature>
   /// <summary>メッシュの四分木データをQuad_MeshDataに作成</summary>
   /// <param name="Mesh">元々のメッシュ二次元配列</param>
   /// <param name="xw">メッシュの横サイズ</param>
   /// <param name="yw">メッシュの縦サイズ</param>
   /// <param name="max_PartitiopnLevel">最大分割レベル</param>
   /// </signature>

   var QD = new Array();
   var stp =Math.pow( 2 , max_PartitiopnLevel - 1);
   var w = Math.floor( xw / stp);//最大分割レベルの横セル数
   var H = Math.floor(yw / stp);//最大分割レベルの縦セル数
   for(var i=0;i<stp;i++){
      var h2;
      if(i == stp-1){
         h2= H + (yw % stp); //ループ最後では余り部分を追加する
      }else{
         h2 = H + 1; //1メッシュ分重複を持たせる
      }
      var py2 = i * H;
      for (var j = 0; j < stp; j++) {
         var w2;
         if(j == stp-1){
            w2 = w + (xw % stp);//ループ最後では余り部分を追加する
         }else{
            w2 = w + 1; //1メッシュ分重複を持たせる
         }
         var f = true;//すべて欠損の場合true
         var px2 = j * w;
         var mxv;// = Mesh[px2][py2];;
         var mnv;// = mxv;
         for(var ky=0;ky<h2;ky++){
            var py =py2 + ky;
            for (var kx = 0; kx < w2; kx++) {
               var px = px2 + kx;
               if (typeof Mesh[px][py] != 'undefined') {
                  if (f == true) {
                     mxv = Mesh[px][py];
                     mnv = mxv;
                     f = false;
                  } else {
                     if (mxv < Mesh[px][py]) {
                        mxv = Mesh[px][py];
                     } else if (Mesh[px][py] < mnv) {
```

```
                    mnv = Mesh[px][py];
                  }
                }
              }
            }
          }
          var mon = Get_MortonNumberXY(j, i, max_PartitiopnLevel - 1, max_PartitiopnLevel);
          //線形四分木配列の最下層レベルに，当該メッシュの範囲と，最大標高，最低標高を保存
          Quad_MeshData[mon] = new Quad_Mesh_Info(new Rectinfo(j * w, j * w + w2 - 2, i * H, i * H + h2 - 2), mxv, mnv,
                    f);
        }
      }

      //親レベルの最大標高，最低標高を計算
      for(var k=max_PartitiopnLevel - 1;k>=1;k--){
        var SP = Get_MortonArrayPosition(k); //自レベルの四分木線形配列の開始位置
        var SP2 = Get_MortonArrayPosition(k - 1);//親レベルの四分木線形配列の開始位置
        for (var i = 0; i < Math.pow(4, k) ; i += 4) {  //子4メッシュで親1メッシュ
          var f = true;
          for (var j = 0; j <= 3; j++) {
            var Quad_MeshDataSub = Quad_MeshData[SP + i + j];
            if (Quad_MeshDataSub.LackF == false) {
              if (f == true) {
                mxv = Quad_MeshDataSub.Max;
                mnv = Quad_MeshDataSub.Min;
                f = false;
              }else{
                if(mxv < Quad_MeshDataSub.Max){
                  mxv = Quad_MeshDataSub.Max;
                }
                if(Quad_MeshDataSub.Min < mnv){
                  mnv = Quad_MeshDataSub.Min;
                }
              }
            }
          }
          //親空間レベルの最大／最小値を設定
          var n = SP2 + Math.floor(i / 4);
          Quad_MeshData[n] = new Quad_Mesh_Info();
          Quad_MeshData[n].LackF = f;
          Quad_MeshData[n].Max = mxv;
          Quad_MeshData[n].Min = mnv;
        }
      }
    }


    function Get_MortonArrayPosition(n) {
      /// <signature>
      /// <summary>四分木線形配列の開始位置。</summary>
      /// <param name="n">n</param>
      /// <returns type="Number" />
      /// </signature>
      return -(1 - Math.pow(4, n)) / 3;
    }

    function Get_MortonNumberXY(X, Y, SpaceLevel, max_PartitiopnLevel) {
      /// <signature>
      /// <summary>点の座標値と所属する空間レベルから，四分木線形配列の位置を返す</summary>
      /// <param name="x">X座標</param>
      /// <param name="y">Y座標</param>
      /// <param name="SpaceLevel">空間レベル</param>
      /// <param name="max_PartitiopnLevel">最大分割レベル</param>
      /// <returns type="Number"/>
      /// </signature>

      var zero ="0".repeat(max_PartitiopnLevel);
      var x2 = (zero + X.toString(2)).right(max_PartitiopnLevel); //X座標を2進数に
      var y2 = (zero + Y.toString(2)).right(max_PartitiopnLevel); //Y座標を2進数に
      var Num = "";
      for(var i = 0;i< max_PartitiopnLevel;i++){
        Num += y2.substr(i, 1) + x2.substr( i, 1);
      }
      return parseInt(Num, 2) + Get_MortonArrayPosition(SpaceLevel);
    }
```

付録 2　線形四分木配列から等値線にかかるメッシュ候補を抽出

```
function Get_Quad_MeshCell(value, Qcell, SpaceLevel, Scell, n, max_PartitiopnLevel) {
   /// <signature>
   /// <summary>四分木から等値線にかかるメッシュを抜き出す再帰処理</summary>
   /// <param name="value">取得する等高線の数値</param>
   /// <param name="Qcell">取得する等高線の値がかかる四分木配列の位置の配列（戻り値）</param>
   /// <param name="SpaceLevel">調べる空間レベル（初回呼び出しは0）</param>
   /// <param name="Scell">調べるモートン番号開始位置（戻り値）</param>
   /// <param name="n">Qcell のメッシュ数，参照渡しにするため配列にする</param>
   /// <param name="max_PartitiopnLevel">最大分割レベル</param>
   /// </signature>
   if (SpaceLevel == 0) {
      //初回の呼び出し。最上位階層の最大値・最小値をチェックし，間に入る場合は再帰呼び出し
      Qcell.length = Get_MortonArrayPosition( max_PartitiopnLevel);
      n[0] = 0;
      var Quad_MeshDataSub = Quad_MeshData[0];
      if ((Check_Two_Value_In(value, Quad_MeshDataSub.Min, Quad_MeshDataSub.Max) != chvOuter) &&
          (Quad_MeshDataSub.LackF == false)) {
         Get_Quad_MeshCell(value, Qcell, SpaceLevel + 1, 0, n, max_PartitiopnLevel)
      }
   } else {
      var SP = Get_MortonArrayPosition( SpaceLevel);
      for (var i = 0; i <= 3 ; i++) {
         var Quad_MeshDataSub = Quad_MeshData[SP + Scell + i];
         if ((Check_Two_Value_In(value, Quad_MeshDataSub.Min, Quad_MeshDataSub.Max) != chvOuter) &&
             (Quad_MeshDataSub.LackF == false)) {
            //該当した場合
            if (SpaceLevel == max_PartitiopnLevel - 1) {
               //最下層の場合，線形四分木配列の位置をQcell に記憶
               Qcell[n[0]] = SP + Scell + i;
               n[0] ++;
            } else {
               //最下層でない場合は，さらに下階層を調べる
               Get_Quad_MeshCell(value, Qcell, SpaceLevel + 1, (Scell + i) * 4, n, max_PartitiopnLevel);
            }
         }
      }
   }
}
```

付録 3　等値線にかかるメッシュから等高線線分を取得

```
function Get_Mesh_Contour(con, Mesh, mi, mj, contour_pos, High, High_CN) {
   /// <signature>
   /// <summary>メッシュ内で横切る等値線を取得</summary>
   /// <param name="con">等高線線分の起終点を入れる配列</param>
   /// <param name="Mesh">元の標高メッシュ配列</param>
   /// <param name="mi">調べる標高メッシュの横位置</param>
   /// <param name="mj">調べる標高メッシュの縦位置</param>
   /// <param name="contour_pos">等高線の位置</param>
   /// <param name="High)">取得する標高値</param>
   /// <param name="High_CN)">等高線ごとのカウンタの配列</param>
   /// </signature>

   var V1 = Mesh[mi][mj];
   var V2 = Mesh[mi + 1][mj];
   var V3 = Mesh[mi][mj + 1];
   var V4 = Mesh[mi + 1][mj + 1];
   //取得する等高線の値とメッシュの4すみを比較
   var VH1 = V1 - High;
   var VH2 = V2 - High;
   var VH3 = V3 - High;
   var VH4 = V4 - High;
   var Q = 0; C12 = 0; C24 = 0; C34 = 0; C13 = 0;
   if( (((VH1 <= 0)&&(VH2 >= 0)) || ((VH1 >= 0 )&&( VH2 <= 0)) )&&( V1 != V2 )){ C12 = 1; Q ++;}
   if( (((VH2 <= 0)&&(VH4 >= 0)) || ((VH2 >= 0 )&&( VH4 <= 0)) )&&( V2 != V4 )){ C24 = 1; Q ++;}
```

```
   if( (((VH4 <= 0)&&(VH3 >= 0)) || ((VH4 >= 0 )&&( VH3 <= 0)) )&&( V3 != V4 )){ C34 = 1; Q ++;}
   if( (((VH1 <= 0)&&(VH3 >= 0)) || ((VH1 >= 0 )&&( VH3 <= 0)) )&&( V1 != V3 )){ C13 = 1; Q ++;}

   switch (Q) {
      case 2:
         ContourLineSegment(mi, mj, C12, C34, C24, C13, VH1, VH2, VH3, VH4, V1, V2, V3, V4, CONTOUR_POS, High_CN, con);
         break;
      case 4:
         if(( V2 == High)&&(V3 == High)){
            C12 = 1; C13 = 1;
            ContourLineSegment (mi, mj, C12, C34, C24, C13, VH1, VH2, VH3, VH4, V1, V2, V3, V4, CONTOUR_POS, High_CN,
                  con);
         }else{
            C34 = 0; C13 = 0; C12 = 1; C24 = 1;
            ContourLineSegment(mi, mj, C12, C34, C24, C13, VH1, VH2, VH3, VH4, V1, V2, V3, V4, CONTOUR_POS, High_CN,
                  con);
            C12 = 0; C24 = 0; C34 = 1; C13 = 1;
            ContourLineSegment(mi, mj, C12, C34, C24, C13, VH1, VH2, VH3, VH4, V1, V2, V3, V4, CONTOUR_POS, High_CN,
                  con);
         }
         break;
      case 3:
         if(( C12 == 1 )&&( C24 == 1 )&&( V2 == High )){ C12 = 0;}
         if( (C12 == 1 )&&( C13 == 1 )&&( V1 == High )){ C12 = 0;}
         if(( C24 == 1 )&&( C34 == 1 )&&( V4 == High )){ C24 = 0;}
         if(( C34 == 1 )&&( C13 == 1 )&&( V3 == High )){ C13 = 0;}
         Get_ContourLineSegment(mi, mj, C12, C34, C24, C13, VH1, VH2, VH3, VH4, V1, V2, V3, V4, CONTOUR_POS, High_CN,
               con);
         break;
   }
}

function Get_ContourLineSegment(mi, mj, C12, C34, C24, C13, VH1, VH2, VH3, VH4, V1, V2, V3, V4, CONTOUR_POS, High_CN,
                           con) {
   var T = 0;
   var po = new Array(4);
   if(C12 == 1 ){
      po[T]=new xyinfo(mi + Math.abs(VH1 / (V1 - V2)),mj);
      T ++;
   }
   if(C34 == 1 ){
      po[T]=new xyinfo(mi + Math.abs(VH3 / (V3 - V4)),mj+1 );
      T ++;
   }
   if(C24 == 1 ){
      po[T]=new xyinfo(mi+1 ,mj + Math.abs(VH2 / (V2 - V4)));
      T ++;
   }
   if(C13 == 1 ){
      po[T]=new xyinfo(mi ,mj + Math.abs(VH1 / (V1 - V3)));
      T ++;
   }
   if(T < 2 ){
      Return; //Tはほぼ常に2で2未満のことはない
   }
   var Pon = High_CN[CONTOUR_POS];
   con[0][CONTOUR_POS][Pon] = po[0];
   con[1][CONTOUR_POS][Pon] = po[1];
   if((con[0][CONTOUR_POS][Pon].x == con[1][CONTOUR_POS][Pon].x )&&
                              (con[0][CONTOUR_POS][Pon].y == con[1][CONTOUR_POS][Pon].y )){
      //二つの座標が同じになってしまう場合は含めない
   } else {
       High_CN[CONTOUR_POS] = Pon + 1;
   }
}
```