2014/10/31　14:00 ～ 17:00
FOSS4G Tokyo ハンズオン「PostGIS ラスタ入門」追加資料

(a) GeoTiff インポート例

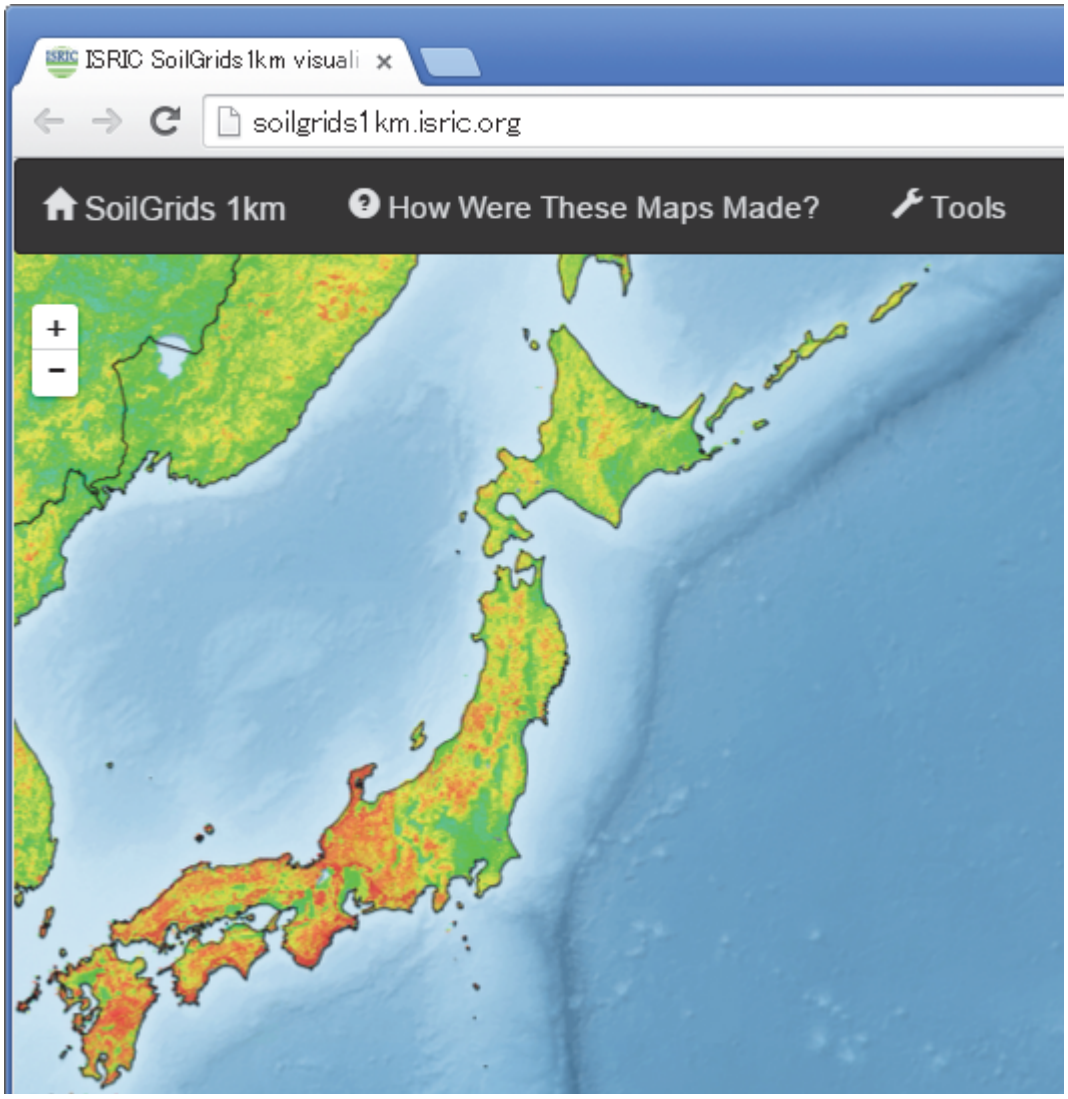
（1）配布データ

PHIHOX_sd1_M_1km_T427.tif など 5 つの TIFF ファイル … インポートするサンプルデータ
raster2pgsql_bat.txt … インポート用バッチファイル例（拡張子を .bat に変えて使用）
add1.sql … 本章にある SQL
add1_output.htm … 出力結果例

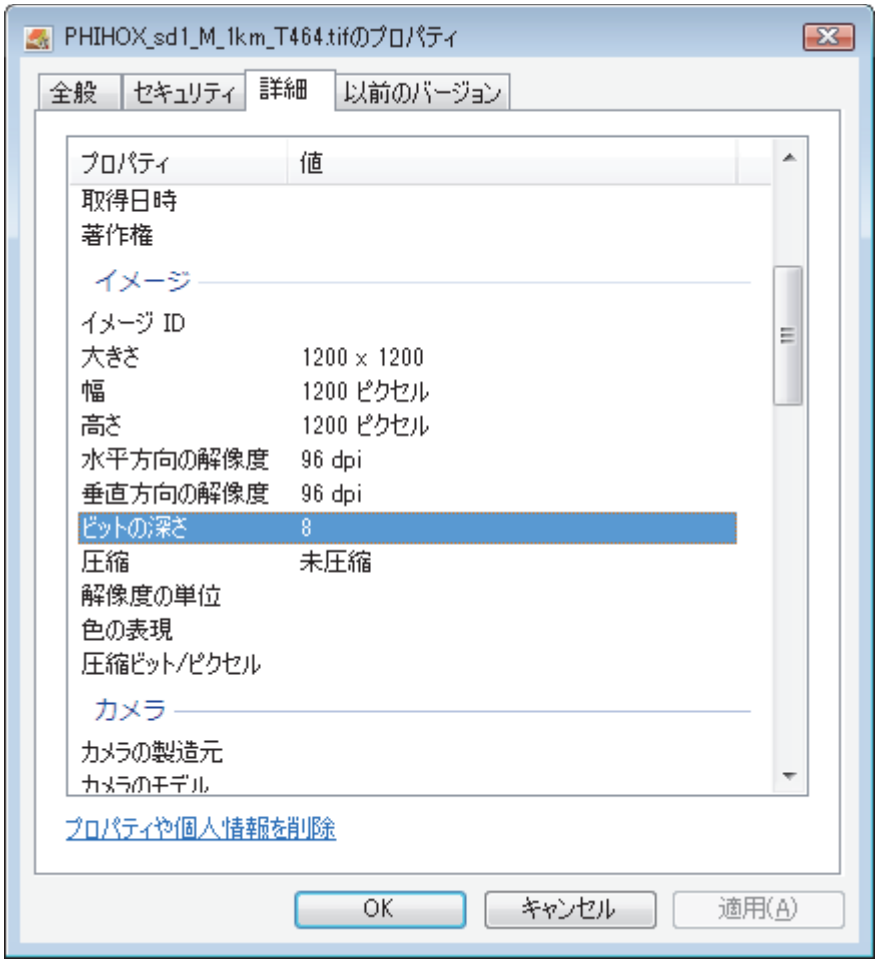
（2）サンプルデータの紹介

・ISRIC（World Soil Information）という研究機関が今年春から公開している、全世界の 1km メッシュ土壌成分です。データは 9 つの大分類に分かれ、経緯度 10°ごとの GeoTiff で簡単にダウンロードできます。今回は地表面の水の pH（酸性・アルカリ性の指標）で、日本列島を含む 5 つのファイルを用意しました。

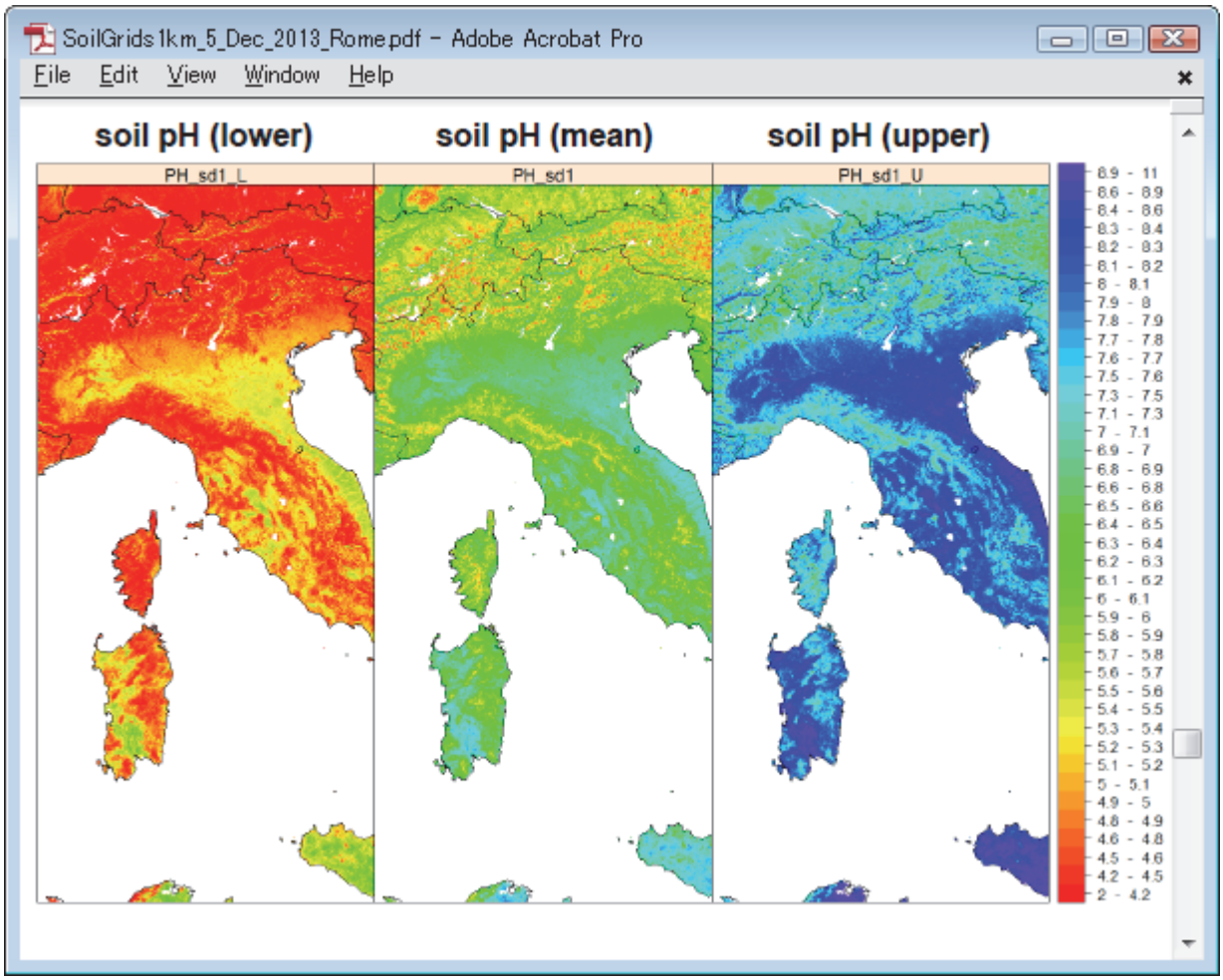
・地図閲覧とダウンロード http://soilgrids1km.isric.org/
・データの詳細（英語）http://www.isric.org/content/soilgrids



・GeoTiff のデータは 1 バンド、8 ビット符号なし整数で（pH を 10 倍）、グレースケール画像と同じなので普通の画像ビュワーで確認できます。



・ISRIC のドキュメントに、同じ pH データを可視化した例があります。
ftp.soilgrids.org/docs/SoilGrids1km_5_Dec_2013_Rome.pdf

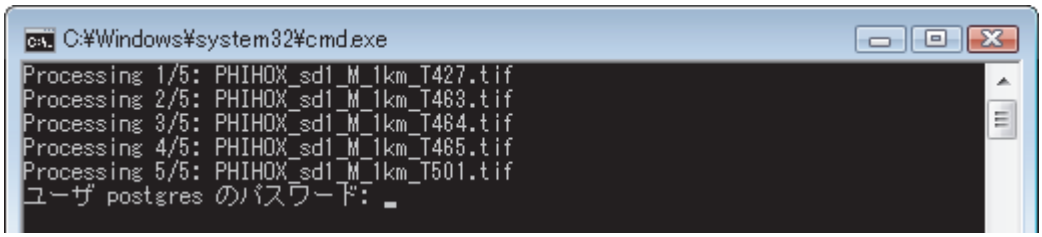


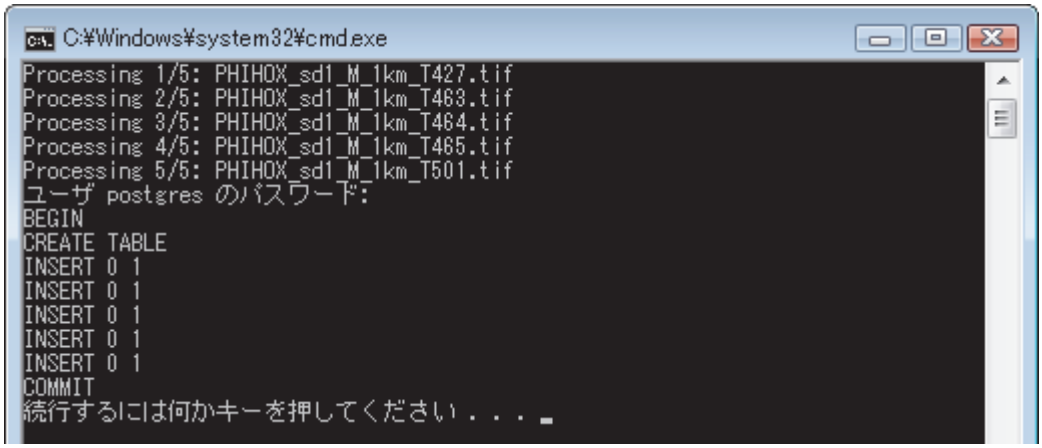
（3）作業例

・前半資料の 13 ページと同様に raster2pgsql.exe でインポートします。下のような中身でバッチファイルを作り、GeoTiff と同じフォルダにおいて実行します。

```
@ECHO OFF
"C:\Program Files\PostgreSQL\9.3\bin\raster2pgsql.exe" *.tif -F raster_add1 > tmp_add1.sql
"C:Program Files\\PostgreSQL\9.3\bin\psql.exe" -d handson2014 -U postgres -W -f tmp_add1.sql
PAUSE
```

・前半で使った PNG と違って、今回は GeoTiff に位置情報が入っているので raster2pgsql.exe は何もメッセージを出しません。psql がパスワードを要求するので入力し、二つ目の画像のように終われば完了です。





↓ ラスタの属性を ST_MetaData と ST_BandMetaData で確認すると、確かに日本列島付近の経緯度になっており、元データと同じ 8BUI のピクセルタイプです。

```
SELECT filename, (ST_MetaData(rast)).* FROM raster_add1 ;
```

	filename text	upperleftx double precision	upperlefty double precision	width integer	height integer
1	PHIHOX_sd1_M_1km_T427.tif	120	30	1200	1200
2	PHIHOX_sd1_M_1km_T463.tif	120	40	1200	1200
3	PHIHOX_sd1_M_1km_T464.tif	130	40	1200	1200
4	PHIHOX_sd1_M_1km_T465.tif	140	40	1200	1200
5	PHIHOX_sd1_M_1km_T501.tif	140	50	1200	1200

scalex double precision	scaley double precision	skewx double precision	skewy double precision	srid integer	numbands integer
0.00833333333333333	-0.00833333333333333	0	0	4326	1
0.00833333333333333	-0.00833333333333333	0	0	4326	1
0.00833333333333333	-0.00833333333333333	0	0	4326	1
0.00833333333333333	-0.00833333333333333	0	0	4326	1
0.00833333333333333	-0.00833333333333333	0	0	4326	1

```
SELECT filename, (ST_BandMetaData(rast, 1)).* FROM raster_add1 ;
```

	filename text	pixeltype text	nodatavalue double precision	isoutdb boolean	path text
1	PHIHOX_sd1_M_1km_T427.tif	8BUI	255	f	
2	PHIHOX_sd1_M_1km_T463.tif	8BUI	255	f	
3	PHIHOX_sd1_M_1km_T464.tif	8BUI	255	f	
4	PHIHOX_sd1_M_1km_T465.tif	8BUI	255	f	
5	PHIHOX_sd1_M_1km_T501.tif	8BUI	255	f	

↓ 適当な一点を含む 1km メッシュでの値（表面の pH を 10 倍したもの）を取得した例。ST_Value 関数の第 3 引数に点ジオメトリを渡します。点から外れるラスタは NULL になります。

```
WITH a AS (
	SELECT ST_SetSrid(ST_Point(139.939, 35.902), 4326) geom
)
SELECT filename, ST_Value(rast, 1, geom) FROM a, raster_add1 ;
```

	filename text	st_value double precision
1	PHIHOX_sd1_M_1km_T427.tif	
2	PHIHOX_sd1_M_1km_T463.tif	
3	PHIHOX_sd1_M_1km_T464.tif	63
4	PHIHOX_sd1_M_1km_T465.tif	
5	PHIHOX_sd1_M_1km_T501.tif	

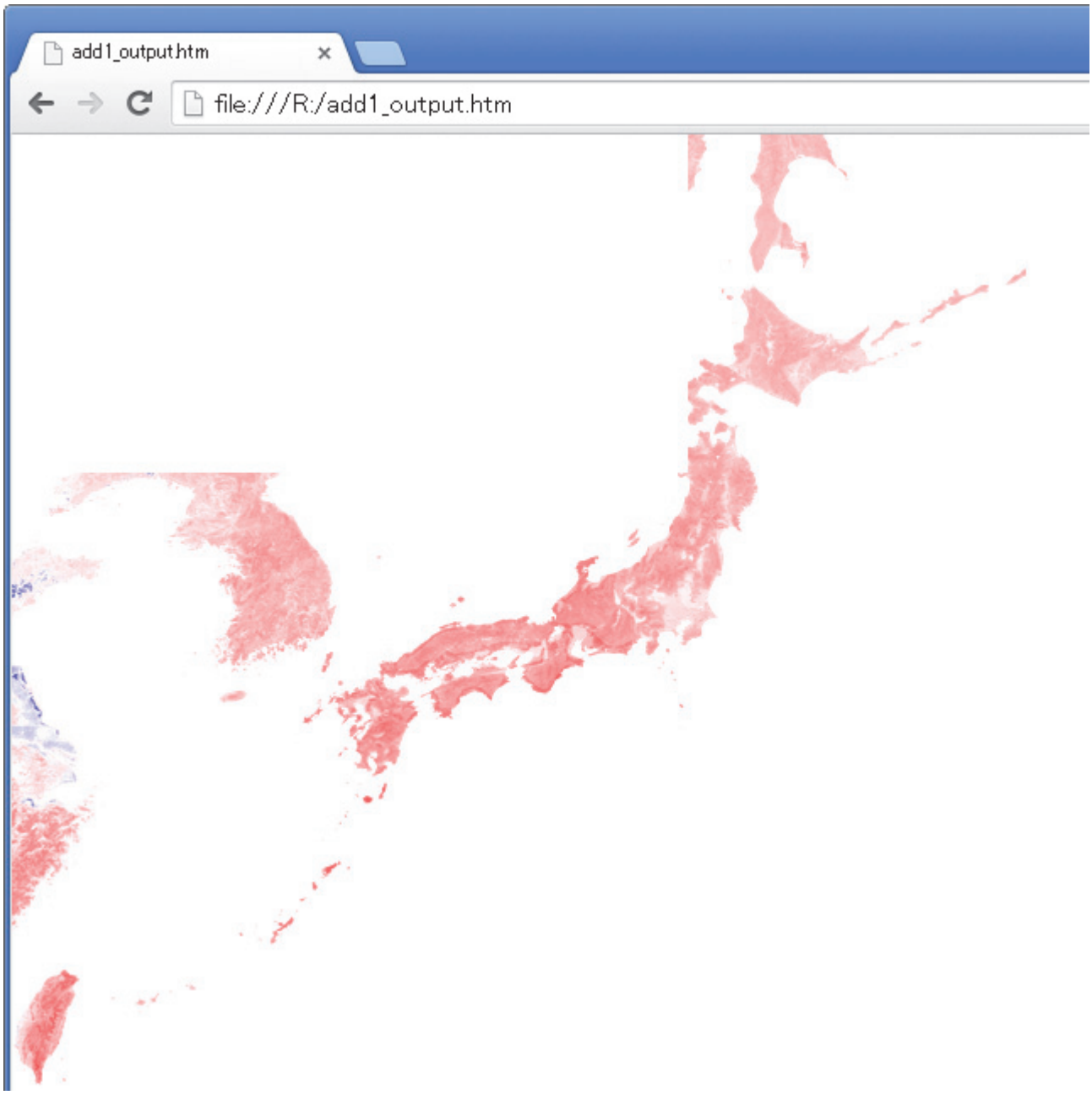
↓ 前半と同様に「画像埋め込み HTML」で可視化する例です。ST_Reclass 関数を使い、酸性（pH7.0 未満）が強いほど濃い赤で、アルカリ性（pH7.0 超）が強いほど濃い青になるよう 3 つのバンドを作り、PNG に変換し HTML に埋め込んでいます。最初の a ブロックが色の設定部分です。

```
COPY (
	WITH a (rce) AS (
		VALUES (ARRAY[
			'[20-70:255,   [70-80:255-0, [80-:0',
			'[20-70:0-255, [70-80:255-0, [80-:0',
			'[20-70:0-255, [70-:255'])
	), b AS (
		SELECT ST_UpperLeftX(rast) :: int x, ST_UpperLeftY(rast) :: int y, rast
		FROM raster_add1
	)
	SELECT concat('<img style="position:absolute; width:200px; height:200px; left:',
		(x - 120) * 20 :: float, 'px; top:', (50 - y) * 20 :: float,
		'px" src="',
		replace(encode(ST_AsPNG(ST_AddBand(NULL, ARRAY[
			ST_Reclass(rast, 1, rce[1], text '8BUI', 255)
			, ST_Reclass(rast, 1, rce[2], text '8BUI', 255)
			, ST_Reclass(rast, 1, rce[3], text '8BUI', 255)
			])), 'base64'), E'\n', ''), '" />')
	FROM a, b
) TO 'C:/Program Files/PostgreSQL/9.3/data/add1_output.htm' ;
```

・出力される HTML は次頁にあります。ST_Reclass 関数の詳細は下記マニュアルを参照して下さい。

■ http://www.finds.jp/docs/pgisman/2.2.0/RT_ST_Reclass.html

↓ 前ページのクエリで作成された HTML をブラウザで開いたところです。



(b) 日本の「標準地域メッシュ」データからのラスタ作成

(1) 配布データ

add2_meshdata.tsv … 国土数値情報「道路密度・道路延長メッシュデータ」から作成したサンプル
N04-10_5339-jgd_GML.zip … 上記サンプルの元データ。作業では使いませんが参考まで。
add2.sql … 本章にある SQL
add2_output.htm … 出力結果例
add2_output.png … 上記 HTML をブラウザで表示し保存した画像

(2) 本章の内容について

・ブログの過去記事 http://kenpg.seesaa.net/article/372615229.html を簡単にしたものです。
・サンプルデータの元は国土数値情報 http://nlftp.mlit.go.jp/ksj/gml/datalist/KsjTmplt-N04.htm です。

(3) 作業例

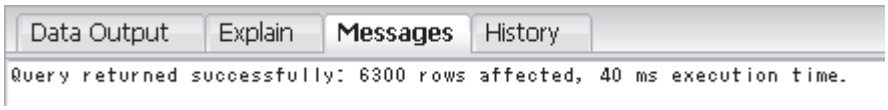
・配布データの add2_meshdata.tsv（タブ区切りテキストファイル）を PostgreSQL にインポートします。データ範囲と解像度は「一つの 1 次メッシュ区画を、3 次メッシュ単位で区分したもの」です。中身は下記のとおりで、国土数値情報「道路密度・道路延長メッシュデータ」2010 年の東京都・埼玉県あたりの 1 次メッシュ区画 5339 の XML ファイルから 3 次メッシュコードと 3 つの数値を抽出しました。数値は左から 1 ㎢当たりの道路延長（全幅員）、同じく幅員 25m 以上、同じく幅員 19.5 ～ 25m です。

・所々ある -1 は、元データでは unknown となっていたメッシュを、独自に変換したものです。
・行数は 6300 あります。本来は 80 × 80 = 6400 あるべきですが、元々のデータから、東京湾にある 2 次メッシュ一つ分の 100 行がまるまる入っていません。PostGIS ラスタにする際、何らかの NODATA 値を設定します。

・以下、インポート先テーブル名を data_add2 とします。
・上のように単純なタブ区切りテキストなので PostgreSQL の COPY コマンドで取り込めます。
・ファイルを PostgreSQL のデータフォルダに置き、次のような SQL でインポートします。

```
CREATE TABLE data_add2 (mesh3 text, val1 int, val2 int, val3 int) ;
COPY data_add2 FROM 'C:/Program Files/PostgreSQL/9.3/data/add2_meshdata.tsv' ;
```

↓ インポートが終わったようす。6300 行が入力されたことが分かります。



↓ テーブルの中身はこんな感じ。以下では列 val1 を PostGIS ラスタへ入力していきます。

```
SELECT * FROM data_add2 ;
```

	mesh3 text	val1 integer	val2 integer	val3 integer
1	53390000	932	0	0
2	53390001	-1	-1	-1
3	53390002	293	0	0
4	53390003	-1	-1	-1
5	53390004	1963	0	0
6	53390005	5883	0	0
7	53390006	1616	0	0
8	53390007	4357	0	0
9	53390008	10475	0	0
10	53390009	10793	0	0
11	53390010	306	306	0
12	53390011	1495	0	0
13	53390012	404	0	0
14	53390013	-1	-1	-1
15	53390014	3601	0	0
16	53390015	3444	0	0

↓ テーブルに入っている値の範囲を確認した様子。これに合わせてラスタのピクセルタイプを決めます。値の範囲とピクセルタイプの関係（前半資料の 7 頁）から、val1 を入れるラスタを 32 ビット整数（32BSI）にします

```
SELECT min(val1), max(val1),
       min(val2), max(val2),
       min(val3), max(val3)
FROM data_add2 ;
```

---

	min integer	max integer	min integer	max integer	min integer	max integer
1	-1	45138	-1	3653	-1	4842

↓ メッシュコードのうち 1 次区画の数値から、ラスタ左上端の経緯度を算出するクエリ。全体で一つの 1 次区画なので、本来は LIMIT 1 で先頭行だけ使えば済みますが、ここでは念のため全行から算出し結果を DISTINCT でまとめています。もし想定外に別の 1 次区画が紛れ込んでいると、この結果が複数行になります。

```
SELECT DISTINCT substr(mesh3, 3, 2) :: float8 + 100 AS upperleftx,
       (substr(mesh3, 1, 2) :: float8 + 1) * 2 / 3 AS upperlefty
FROM data_add2 ;
```

---

	upperleftx double precision	upperlefty double precision
1	139	36

・上の経緯度に加え、標準地域メッシュの定義から決まる width , height , scalex , scaley と、元データの測地投影系から決まる SRID を合わせて ST_MakeEmptyRaster に入れ、一つの空ラスタを作り、続けて ST_AddBand 関数で、先ほど確認したピクセルタイプ、ピクセルの適当な初期値、NODATA 値を設定する SQL は次頁のようになります。ここではピクセルの適当な初期値として -100 を入れています。これが「入れ物」としてのラスタになります。一連の作業で適切にメッシュの値を入力できたら、-100 という値を持つピクセルはなくなります。

```
WITH a AS (
	SELECT DISTINCT substr(mesh3, 3, 2) :: float8 + 100 AS upperleftx,
		(substr(mesh3, 1, 2) :: float8 + 1) * 2 / 3 AS upperlefty
	FROM data_add2
)
SELECT ST_AddBand(
	ST_MakeEmptyRaster(
		80, 80, upperleftx, upperlefty,
		1 :: float8 / 80,
		-2 :: float8 / 3 / 80, 0, 0, 4612),
	text '32BSI', -100, -1) rast
FROM a;
```

	rast raster
1	01000001009A9999999999893F11111111111181BF000000000060614000000000000042400000000000000000000(

↓ 上で「入れ物」ができました。次に、ラスタに投入する「中身」すなわち 2 次元配列を作ります。配列の要素の順番は、ラスタ内のピクセル位置から決まります（左上から、行 → 列の順で右上へ）。ピクセル位置は、次の SQL のように 3 次メッシュコードから算出できます。列 x と y が、左上端を基準とするピクセル位置です。

```
SELECT mesh3,
	substr(mesh3, 6, 1) :: int * 10 + substr(mesh3, 8, 1) :: int + 1 AS x,
	80 - (substr(mesh3, 5, 1) :: int * 10 + substr(mesh3, 7, 1) :: int) AS y,
	val1, val1, val2
FROM data_add2 ;
```

	mesh3 text	x integer	y integer	val1 integer	val1 integer	val2 integer
1	53390000	1	80	932	932	0
2	53390001	2	80	-1	-1	-1
3	53390002	3	80	293	293	0
4	53390003	4	80	-1	-1	-1
5	53390004	5	80	1963	1963	0
6	53390005	6	80	5883	5883	0
7	53390006	7	80	1616	1616	0
8	53390007	8	80	4357	4357	0
9	53390008	9	80	10475	10475	0
10	53390009	10	80	10793	10793	0
11	53390010	1	79	306	306	306
12	53390011	2	79	1495	1495	0
13	53390012	3	79	404	404	0
14	53390013	4	79	-1	-1	-1
15	53390014	5	79	3601	3601	0
16	53390015	6	79	3444	3444	0
17	53390016	7	79	2205	2205	0
18	53390017	8	79	6370	6370	0

・上の列 x と y と val1 から、ラスタに投入する 2 次元配列を作る SQL は次頁のようになります。PL/pgSQL を使い他のプログラミング言語のように書くこともできますが、後で全体を一つの SQL にまとめて実行するため、今回はこうしました。

```
WITH a AS (
	SELECT * FROM generate_series(1, 80) AS foo (x), generate_series(1, 80) AS bar (y)
), b AS (
	SELECT mesh3,
		substr(mesh3, 6, 1) :: int * 10 + substr(mesh3, 8, 1) :: int + 1 AS x,
		80 - (substr(mesh3, 5, 1) :: int * 10 + substr(mesh3, 7, 1) :: int) AS y,
		val1
	FROM data_add2
), c AS (
	SELECT y, array_agg(val) AS ary
	FROM (
		SELECT x, y, coalesce(val1, -1) AS val
		FROM a LEFT JOIN b USING (x, y)
		ORDER BY y, x
	) AS foo
	GROUP BY y
)
SELECT ('{' || string_agg(tmp, ',') || '}') :: int[][]
FROM (
	SELECT '{' || array_to_string(ary, ',') || '}' AS tmp FROM c ORDER BY y
) foo ;
```

	int4 integer[]
1	{{561,1709,4697,813,3979,7647,22548,10966,5630,3958,1511,2999,1334,2717,4051,2380,3950,2036,

・上の結果の先頭がラスタの左上端です。3 次メッシュコードの下四桁でいえば 7090 ～ 7099 に当たります。そこの元データの数値 ↓ と照合すると、確かに合っています。これがずれている場合、2 次元配列作成の過程で何らかのミスをした可能性があります。

・以上で、入れ物（ラスタ）と中身（データ）の両方が揃いました。前者に後者を入力するには PostGIS 2.1 で追加された関数 ST_SetValues を使います。全体の処理を汎用的に使いやすくし、テーブル作成を先頭に付けた SQL が次頁のものです。WITH 句の最初のブロックでテーブルを指定し、投入対象の列 val1 を val という列名に付け替えています。val1 を、サンプルテーブルにある val2 または val3 に変更しても動きます。

```
CREATE TABLE raster_add2 AS
WITH t AS (
	SELECT mesh3, val1 AS val FROM data_add2
), a AS (
	SELECT * FROM generate_series(1, 80) AS foo (x), generate_series(1, 80) AS bar (y)
), b AS (
	SELECT mesh3,
		substr(mesh3, 6, 1) :: int * 10 + substr(mesh3, 8, 1) :: int + 1 AS x,
		80 - (substr(mesh3, 5, 1) :: int * 10 + substr(mesh3, 7, 1) :: int) AS y,
		val
	FROM t
), c AS (
	SELECT y, array_agg(val) AS ary
	FROM (
		SELECT x, y, coalesce(val, -1) AS val
		FROM a LEFT JOIN b USING (x, y)
		ORDER BY y, x
	) foo
	GROUP BY y
), d AS (
	SELECT ('{' || string_agg(tmp, ',') || '}') :: int[][] AS dat_ary
	FROM (
		SELECT '{' || array_to_string(ary, ',') || '}' AS tmp FROM c ORDER BY y
	) AS foo
), r1 AS (
	SELECT DISTINCT substr(mesh3, 3, 2) :: float8 + 100 AS upperleftx,
		(substr(mesh3, 1, 2) :: float8 + 1) * 2 / 3 AS upperlefty
	FROM t
), r2 AS (
	SELECT ST_AddBand(
		ST_MakeEmptyRaster(
			80, 80, upperleftx, upperlefty,
			1 :: float8 / 80,
			-2 :: float8 / 3 / 80, 0, 0, 4612),
		text '32BSI', -100, -1) rast
	FROM r1
)
SELECT ST_SetValues(rast, 1, 1, 1, dat_ary) AS rast
FROM d, r2 ;
```

↓ クエリが成功した様子。

・次頁にテーブルのメタデータを示します。標準地域メッシュ 5339 の定義どおりに位置情報が設定されています。

```
SELECT (ST_MetaData(rast)).* FROM raster_add2 ;
```

	upperleftx double precision	upperlefty double precision	width integer	height integer	scalex double precision	scaley double precision	skewx double precision	skewy double precision	srid integer	numbands integer
1	139	36	80	80	0.0125	-0.00833333333333333	0	0	4612	1

・ページ add 8 で書いたように、ラスタのピクセル初期値として適当に -100 を設定しました。もし入力漏れがあれば -100 のピクセルが残っていますが、それを確認した SQL が下記です。ST_Quantile 関数を使い、NODATA 値も含めて（第 3 引数 = FALSE)、最小値と最大値（第四引数の配列）を得ています。その結果、最小値は -1 で初期値は残っておらず、入力漏れがなかったと分かります。

```
SELECT ST_Quantile(rast, 1, FALSE, ARRAY[0, 1]) FROM raster_add2 ;
```

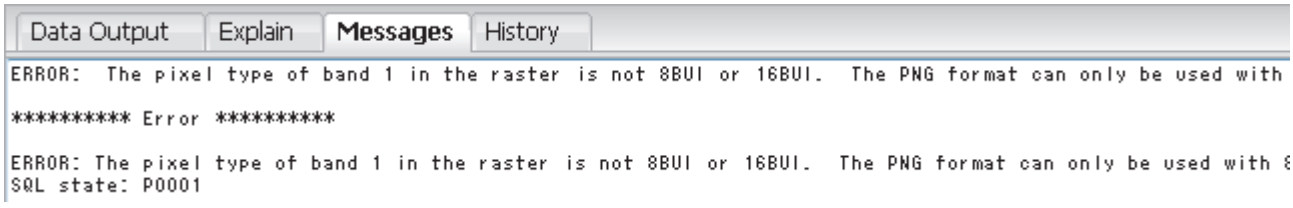
	st_quantile record
1	(0,-1)
2	(1,45138)

・ST_Quantile 関数の詳細は、下記マニュアルを参照して下さい。

■ http://www.finds.jp/docs/pgisman/2.2.0/RT_ST_Quantile.html

・ところで、このラスタはピクセルタイプが 8BUI でないので、PNG など画像ファイルには直接変換できません。例えば ST_AsPNG 関数にラスタを渡すと、下のようにエラーになります。

```
SELECT ST_AsPNG(rast) FROM raster_add2 ;
```



・一方 ST_ColorMap 関数を使うと、全ピクセル値が、画像ファイルに収まる値に自動的に変換され ST_AsPNG 関数に渡せるようになります。それを前半同様に「画像埋め込み HTML」に出力する SQL が下記です。

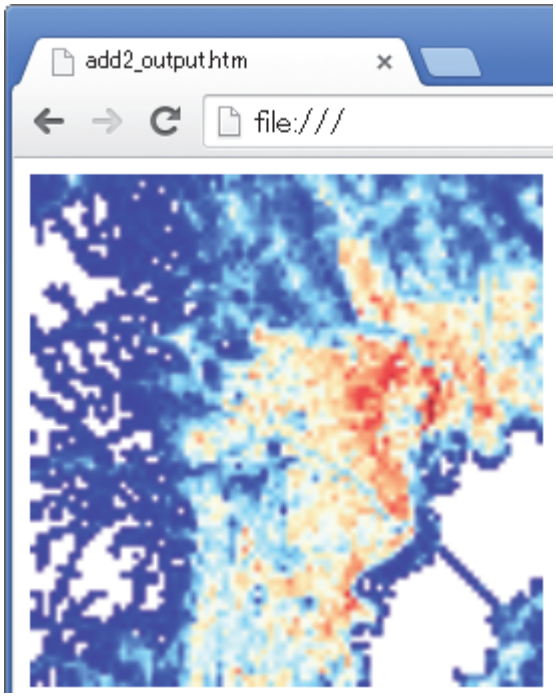
```
COPY (
	SELECT '<img width=256 height=256 src="' ||
		replace(encode(ST_AsPNG(ST_ColorMap(rast, 'bluered')), 'base64'), E'\n', '') ||
		'" />'
	FROM raster_add2
) TO 'C:/Program Files/PostgreSQL/9.3/data/add2_output.htm' ;
```

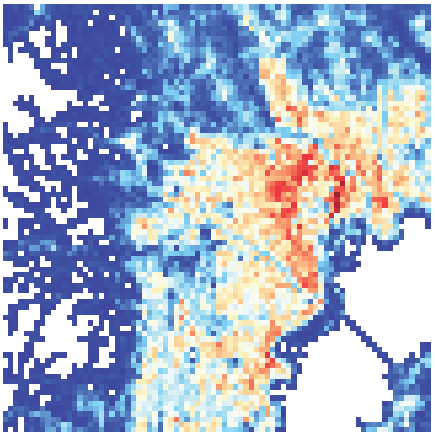
・出力された結果は次頁に示します。

・ST_ColorMap 関数の詳細は、下記マニュアルを参照して下さい。

■ http://www.finds.jp/docs/pgisman/2.2.0/RT_ST_ColorMap.html

↓ HTML をブラウザで表示すると自動的に補間されてぼやけますが（左）、ブラウザの機能で画像だけ保存して確認すると（右）メッシュデータがきちんと再現されています。東京・神奈川を中心とする道路密度が何となく分かると思います。東京湾にかかっている線はアクアラインです。





(c) ジオメトリをラスタ化して使う例

（1）配布データ

add3_geomdata.tsv … 国土数値情報「行政区域データ」から作成した埼玉県 2007 年市区町村ジオメトリ
N03-071001_11_GML.zip … 上記サンプルの元データ。作業では使いませんが参考まで。
add3.sql … 本章にある SQL
add3_output_1.htm … 出力結果例
add3_output_2.htm …　〃
add3_output_border.htm …　〃

（2）本章の内容について

・次のブログ過去記事の基本となる手順に、前半で用いた「画像埋め込み HTML」による出力を加えました。行政区域の面ジオメトリを PostGIS ラスタに変換し「色塗り地図」として出力します。使うのは SQL だけです。

市区町村の塗り分け地図を SQL だけで作成（2013/08/24）
■ http://kenpg.seesaa.net/article/372856792.html

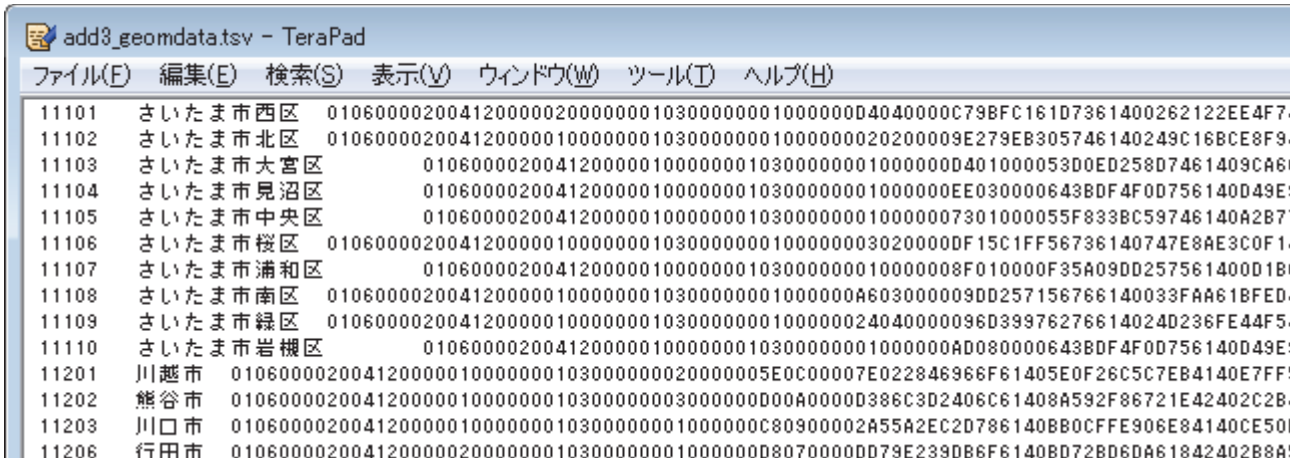
市区町村とメッシュデータを重ねて PNG 出力（2013/08/27）
■ http://kenpg.seesaa.net/article/373157491.html

任意の地物（点・線・面）を重ねて PNG 出力（2013/08/28）
■ http://kenpg.seesaa.net/article/373263308.html

・サンプルデータの元は国土数値情報 http://nlftp.mlit.go.jp/ksj/gml/datalist/KsjTmplt-N03.html です。

（3）作業例

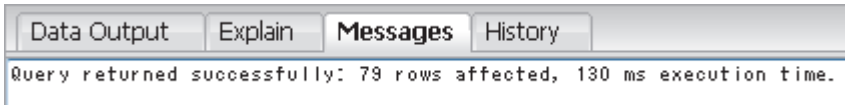
・配布データの add3_geomdata.tsv（タブ区切りテキストファイル）を PostgreSQL にインポートします。埼玉県の 2007 年時点の市区町村が、市区町村ごとのマルチポリゴンになっています。ジオメトリは PostGIS データそのもので、PostgreSQL の COPY コマンドで直接 PostGIS にインポートできます。他の列は市区町村コードと市区町村名だけです。下が TSV の先頭部分。



・以下、インポート先テーブル名を geom_add3 とします。
・ファイルを PostgreSQL のデータフォルダに置き、次のような SQL でインポートします。

```
CREATE TABLE geom_add3 (jcode text, jname text, geom geometry(MULTIPOLYGON, 4612)) ;
COPY geom_add3 FROM 'C:/Program Files/PostgreSQL/9.3/data/add3_geomdata.tsv' ;
```

↓ インポートが成功すると、79 行が書き込まれた旨が表示されます。



↓ インポート先テーブルの先頭。TSV と同じです。

```
SELECT * FROM geom_add3 LIMIT 10 ;
```

	jcode text	jname text	geom geometry(MultiPolygon,4612)
1	11101	さいたま市西区	0106000020041200000200000001030000000100000D4040000C79BFC161D7361400262122
2	11102	さいたま市北区	0106000020041200000100000001030000000100000020200009E279EB30574614024 9C16E
3	11103	さいたま市大宮区	0106000020041200000100000001030000000100000D401000053D0ED258D7461409CA6CF0
4	11104	さいたま市見沼区	0106000020041200000100000001030000000100000EE030000643BDF4F0D756140D49E927
5	11105	さいたま市中央区	0106000020041200000100000001030000000100000073010000055F833BC59746140A2B7787
6	11106	さいたま市桜区	0106000020041200000100000001030000000100000003020000DF15C1FF5673614074 7E8AF

・上のテーブルの各行に、色塗り用の適当な数値を random 関数で与え、同時にマルチポリゴンを ST_AsRaster 関数で PostGIS ラスタに変換して新しいテーブル rast_add3 を作ります。この時、ラスタの解像度やピクセルの値のタイプを入力します。↓ はその一例で、最初の a ブロックで解像度を設定しています。この値が小さいほどラスタが精細になり、処理時間とテーブルサイズが増加します。

```
CREATE TABLE rast_add3 AS
WITH a AS (
       SELECT 2e-003 AS res -- 解像度
), b AS (
       SELECT jcode, jname, floor(random() * 256) AS val, geom
       FROM geom_add3
)
SELECT jcode, jname, val,
       ST_AsRaster(geom,
              res, res * -1,-- scalex, scaley
              res, res,     -- gridx, gridy
              '8BUI', val) AS rast
FROM a, b ;
```

・ST_AsRaster 関数の詳細は、下記マニュアルを参照。

■ http://www.finds.jp/docs/pgisman/2.2.0/RT_ST_AsRaster.html

↓ 作成したラスタテーブルの確認。val に適当な数値が入り、設定した解像度に丸めた位置情報が入っています。

```
SELECT jcode, jname, val, (ST_MetaData(rast)).* FROM rast_add3 ;
```

	jcode text	jname text	val double	upperleftx double precision	upperlefty double precision	width integer	height integer
1	11101	さいたま市西区	237	139.54	35.946	33	38
2	11102	さいたま市北区	225	139.588	35.968	28	27
3	11103	さいたま市大宮区	229	139.594	35.928	28	21
4	11104	さいたま市見沼区	36	139.624	35.966	39	37
5	11105	さいたま市中央区	116	139.604	35.904	20	25
6	11106	さいたま市桜区	243	139.568	35.89	35	29
7	11107	さいたま市浦和区	106	139.632	35.902	19	29
8	11108	さいたま市南区	249	139.616	35.868	42	20
9	11109	さいたま市緑区	11	139.66	35.92	43	32
10	11110	さいたま市岩槻区	34	139.652	36.004	53	62

・前頁で作ったテーブルは、市区町村の 1 行ごとに小さなラスタが作られています。各市区町村のラスタの全ピクセル値は val 列と同じです。これを ST_ValueCount 関数で確認したのが下の SQL 結果です。関数の戻り値は複合型で、1 列目が値、2 列目がその値を持つピクセル数です。

```
SELECT jcode, jname, val, ST_ValueCount(rast) FROM rast_add3 ;
```

	jcode text	jname text	val double	st_valuecount record
1	11101	さいたま市西区	237	(237,732)
2	11102	さいたま市北区	225	(225,418)
3	11103	さいたま市大宮区	229	(229,322)
4	11104	さいたま市見沼区	36	(36,767)
5	11105	さいたま市中央区	116	(116,212)
6	11106	さいたま市桜区	243	(243,468)
7	11107	さいたま市浦和区	106	(106,286)
8	11108	さいたま市南区	249	(249,340)
9	11109	さいたま市緑区	11	(11,662)
10	11110	さいたま市岩槻区	34	(34,1225)
11	11201	川越市	29	(29,2710)
12	11202	熊谷市	248	(248,3994)
13	11203	川口市	199	(199,1396)
14	11206	行田市	53	(53,1701)
15	11207	秩父市	46	(46,14414)
16	11208	所沢市	178	(178,1801)
17	11209	飯能市	227	(227,4823)
18	11210	加須市	32	(32,1484)
19	11211	本庄市	43	(43,2243)
20	11212	東松山市	59	(59,1637)

・ST_ValueCount 関数の詳細は、下記マニュアルを参照。

■ http://www.finds.jp/docs/pgisman/2.2.0/RT_ST_ValueCount.html

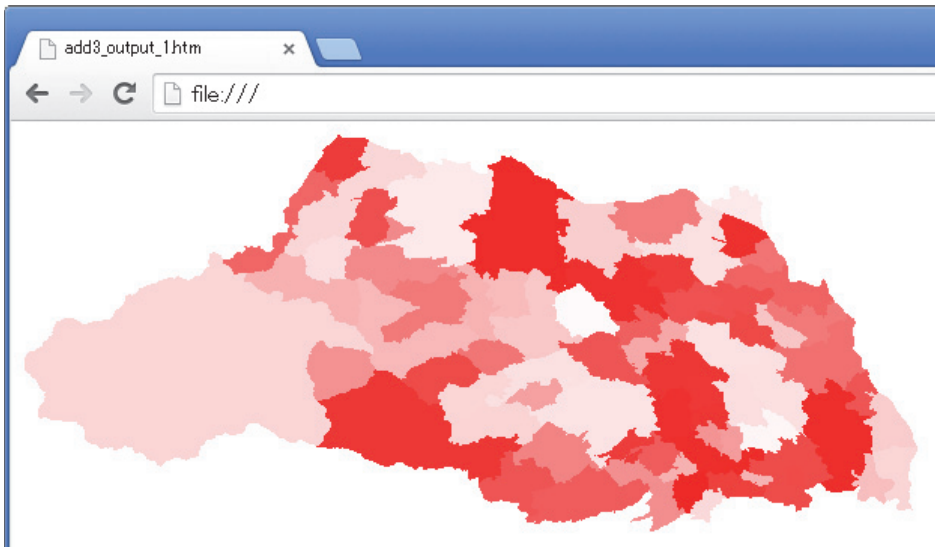
・上のように市区町村ラスタごとに一つの値が付いているので、全市区町村に対し ① 共通の値スケールで何らかの色を付け、② 一つのラスタに統合すれば簡易な「色塗り地図」になります。処理の順序は ② → ① でも同じというか、その方が効率的です。これを行い、続けて画像埋め込み HTML として出力する SQL が下記です。ラスタを一つにまとめるには ST_Union 関数（ラスタ用）を使います。

```
COPY (
	WITH a AS (
		SELECT ST_AddBand(ST_Union(rast),
			ARRAY[
				ROW(1, '8BUI', 255, NULL),   -- R
				ROW(2, '8BUI', 0, NULL),     -- G
				ROW(3, '8BUI', 0, NULL)      -- B
			] :: addbandarg[]) AS rast
		FROM rast_add3
	)
	SELECT '<img src="' ||
		replace(encode(ST_AsPNG(ST_Union(rast)), 'base64'), E'\n', '') ||
		'" />'
	FROM a
) TO 'C:/Program Files/PostgreSQL/9.3/data/add3_output_1.htm' ;
```

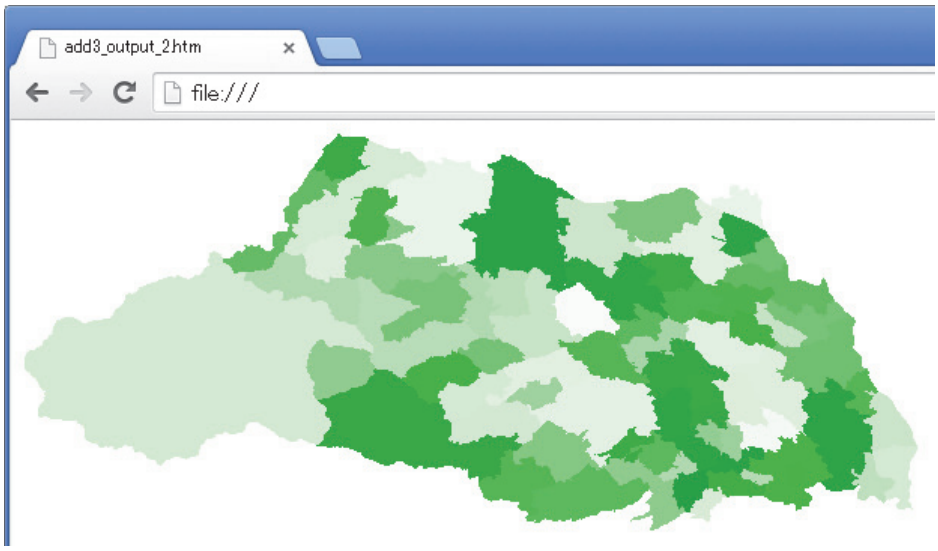
・前頁の SQL で使った ST_Union 関数（ラスタ用）の詳細は、下記マニュアルを参照。

■ http://www.finds.jp/docs/pgisman/2.2.0/RT_ST_Union.html

↓ 前頁の出力結果です。市区町村の値（ラスタのピクセル値）が高いほど色が濃くなっています。濃淡は PNG の透明度として表現されており、RGB 自体はどの市町村も同じ (255, 0, 0) です。このベースカラーを変えるだけで全体の色調を簡単に変えられて便利です。



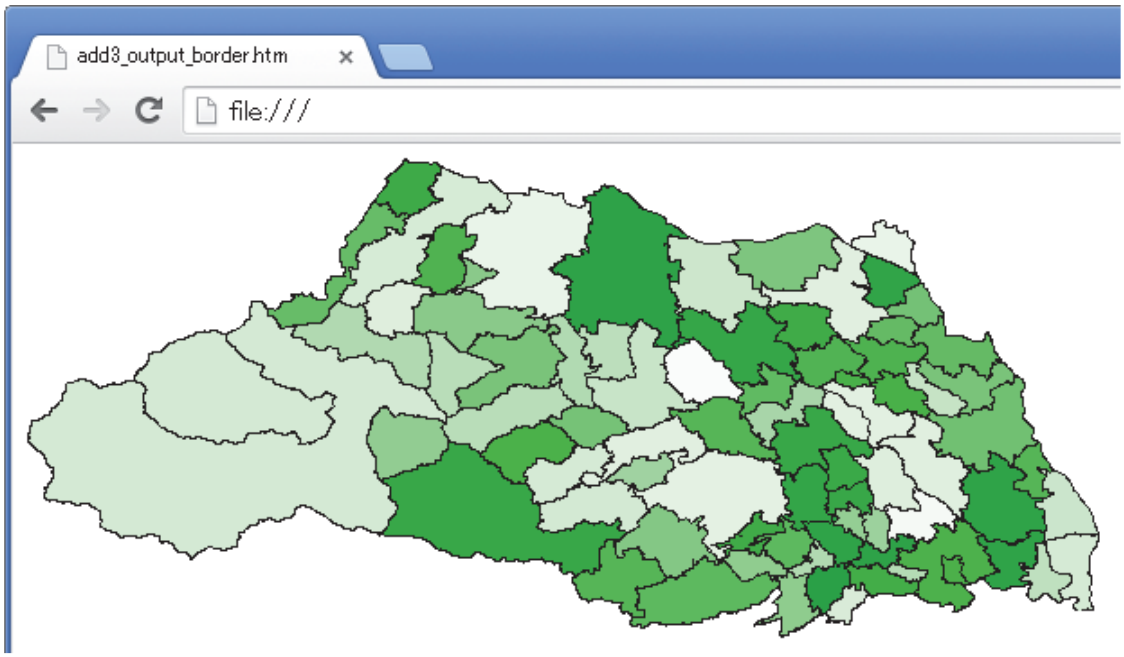
・ベースカラーの変更は、前頁 SQL で -- R -- G -- B とコメントを付けた箇所で行います。255, 0, 0 となっているのを、例えば 0, 155, 0 に変えると ↓ のようになります。また 0, 0, 255 とかにすると青系統になります。



・ブラウザで画像だけを表示 or 保存すれば、画像サイズが分かります。今回は 595 × 266 ピクセルで、市区町村地図としてはかなり粗いですが、市区町村別の値を簡易に可視化する手段としては有効です。また、もう少し解像度を上げれば文章や Web には結構使えます。

・最後に、せっかく市区町村地図なので境界線を加えて出力する SQL と、出力結果の例を次頁に示します。境界線もラスタ化することで SQL だけで完結する手順になります。

```
COPY (
	WITH a (res, ptype, nodata) AS (
		-- 解像度 , ピクセルタイプ , NODATAVALUE 配列
		VALUES (2e-003, '{8BUI, 8BUI, 8BUI, 8BUI}' :: text[], ARRAY[255, 255, 255, 0])
	), b AS (
		SELECT val, geom
		FROM geom_add3 JOIN rast_add3 USING (jcode)
	), c AS (
		SELECT ST_ExteriorRing((ST_Dump(geom)).geom) geom
		FROM geom_add3
	), d AS (
		SELECT ST_AsRaster(geom,
				res, res * -1,-- scalex, scaley
				res, res,     -- gridx, gridy
				ptype, ARRAY[0, 155, 0, val], nodata) AS rast
		FROM a, b
		UNION ALL SELECT ST_AsRaster(geom,
				res, res * -1,-- scalex, scaley
				res, res,     -- gridx, gridy
				ptype, ARRAY[0, 0, 0, 255], nodata)
		FROM a, c
	)
	SELECT '<img src="' ||
		replace(encode(ST_AsPNG(ST_Union(rast)), 'base64'), E'\n', '') ||
		'" />'
	FROM d
) TO 'R:/add3_output_border.htm' ;
```



・前半資料の目次に記した (d) 空間補間した気象データの取り込み・可視化は、まず時間がなさそうなので割愛し、題材のみ紹介します。ちょうど先週行った学会発表に使ったものです。

・埼玉県内の 500m メッシュ気温データを R で作り（gstat パッケージで空間補間）、結果の数値を PostgreSQL のテーブルにインポートし、(b) で説明した標準地域メッシュからラスタを作る手法で PostGIS ラスタ化しました。その結果を様々なカラータイプで可視化する際、本資料で紹介した ST_Reclass、ST_ColorMap など PostGIS ラスタの関数が大変役に立ちました。また、さいたま市だけの値を抽出して最大・最小値やその差を検討する際（下のスライドの折れ線グラフがその部分）、さいたま市のジオメトリと ST_Clip 関数を使って必要な部分だけ抽出し、ST_Quantile 関数などでピクセル値の分布を把握しました。

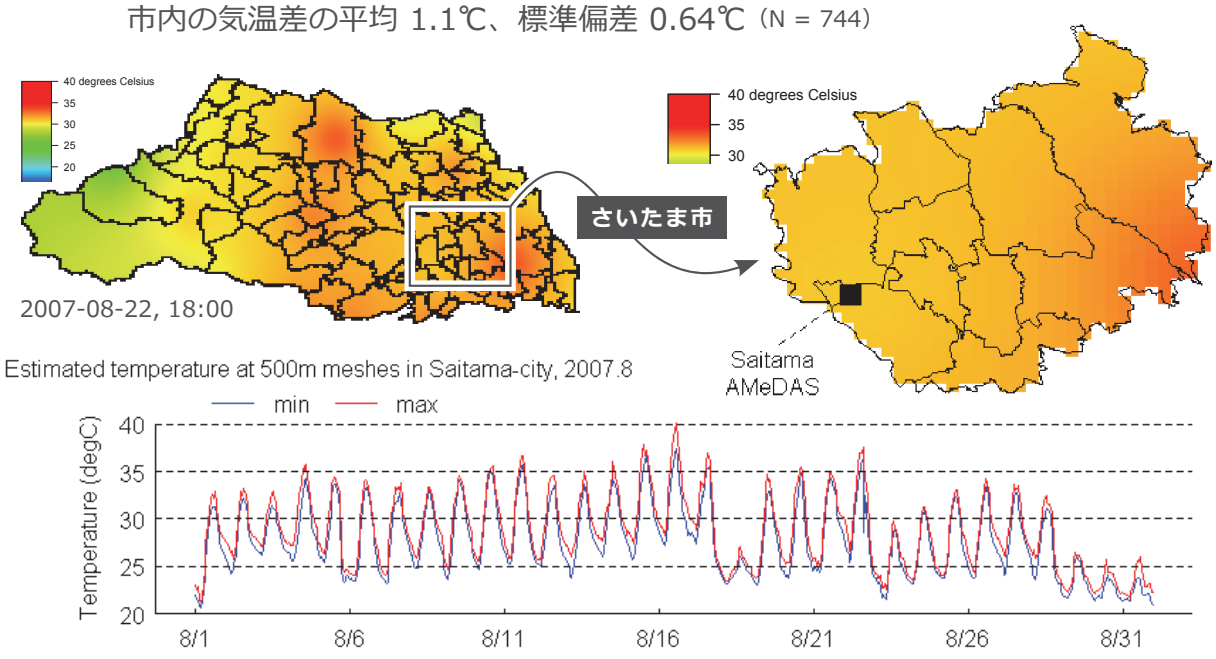
2014/10/24 第 53 回日本生気象学会大会

さいたま市の 2006～2013 年における熱中症発生の特徴

埼玉大学大学院 理工学研究科
国府田 諭 , 藤野 毅 , 高橋 忠司

■ 埼玉県の 500m メッシュ気温推計（2007 年 8 月の全時間値）

さいたま市内アメダスなど気象庁観測データから空間補間した結果、
市内の気温差の平均 1.1℃、標準偏差 0.64℃（N = 744）



（追加資料は以上です。）