Stockage, manipulation et analyse de données matricielles avec PostGIS Raster

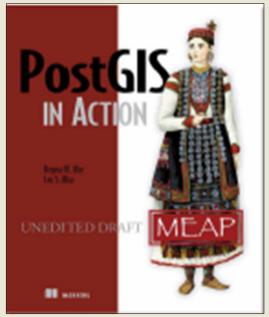
Pierre Racine
Professionel de recherche
Centre d'étude de la forêt

Département des sciences du bois et de la forêt,
Université Laval, Québec

Session PostgreSQL
Paris, juin 2011

Introducing PostGIS Raster

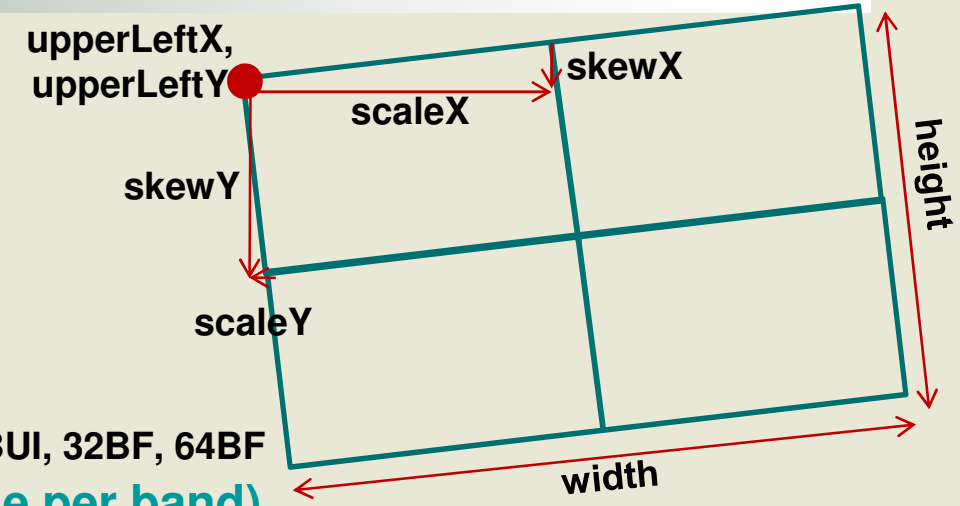
- **Support for rasters in the PostGIS spatial database**
  - RASTER is a new native base type like the PostGIS GEOMETRY type
  - Implemented very much like and as easy to use as the GEOMETRY type
    - One row = one raster
    - One table = one coverage
  - Integrated as much as possible with the GEOMETRY type
    - SQL API easy to learn for usual PostGIS users
    - Full raster/vector analysis capacity taking nodata value into account.
    - Seamless when possible.
  - First release with future PostGIS 2.0
- **Development Team**
  - Current: Bborie Park, Jorge Arevalo, Pierre Racine, Regina & Leo Obe
  - Past: Sandro Santilli, Mateusz Loskot, David Zwarg
- **Founding**
  - Steve Cumming through a Canada Foundation for Innovation grant
  - Deimos Space, Davis University, Cadcorp, Azavea, OSGeo



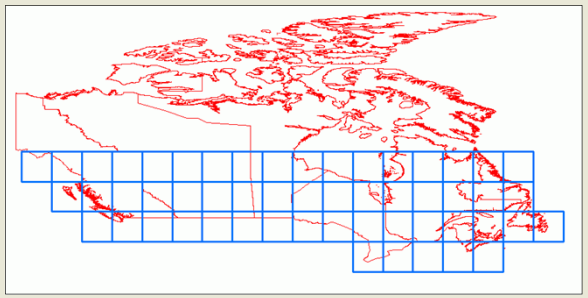
Chapter 13 on PostGIS Raster

Georeferenced, Multiband, Multiresolution and Tiled Coverages

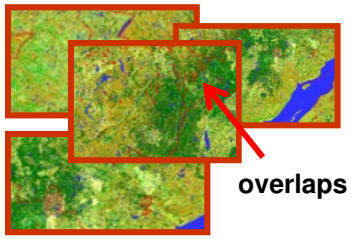
- Georeferenced
  - Each tile/raster is georeferenced
  - Support for rotation (or skew)
- Multiband
  - Support for band with different pixeltypes in the same raster
    - 1BB, 8BSI, 8BUI, 16BSI, 16BUI, 32BSI, 32BUI, 32BF, 64BF
  - Full supports for nodata values (one per band)
  - No real limit on number of band
- Tiled
  - No real distinction between a tile and a raster
  - No real limit on size
    - 1 GB per tile, 32 TB per coverage (table)
    - Rasters are compressed (by PostgreSQL)
  - Support for non-rectangular tiled coverage
- Multiresolution (or overviews) are stored in different tables
- List of raster columns available in a raster_columns table similar to the geometry_columns table



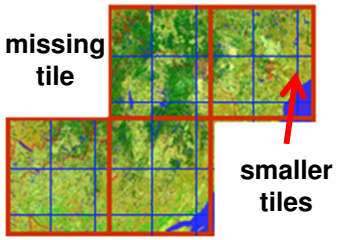
e.g. SRTM Coverage for Canada



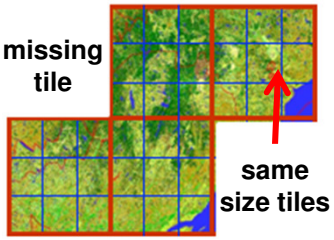
Supports Many Raster Arrangements



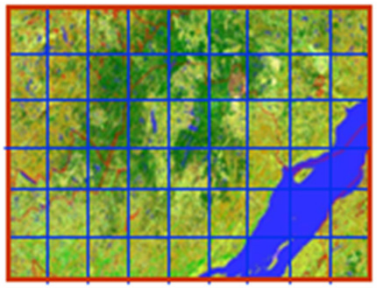
a) warehouse of untiled and unrelated images (4 images)



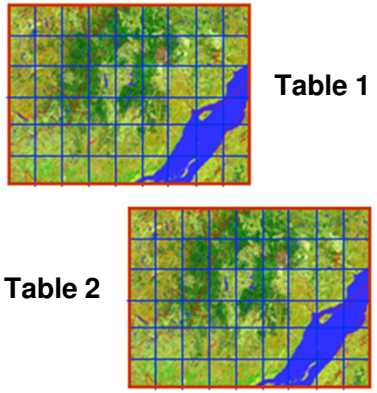
b) irregularly tiled raster coverage (36 tiles)



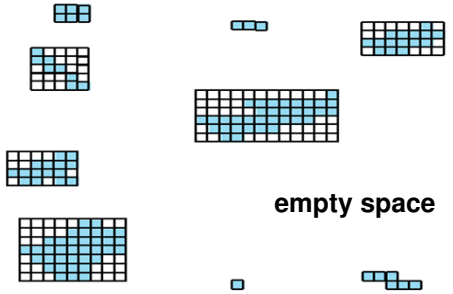
c) regularly tiled raster coverage (36 tiles)



d) rectangular regularly tiled raster coverage (54 tiles)



e) tiled images (2 tables of 54 tiles)



f) rasterized geometries coverage (9 lines in the table)

What You Can Do Now?

Store and manage rasters in the database…

- **Import a series of raster**
  - **raster2pgsql.py -r "c:/temp/mytiffolder/*.tif" -t mytable -s 4326 -k 50x50 -I | psql -d testdb**
  - **Very similar to shp2pgsql**
  - **Any raster format supported by GDAL**
- **Get details about the raster georeference**
  - **ST_UpperLeftX(), ST_UpperLeftY(), ST_Height(), ST_Width(), ST_ScaleX(), ST_ScaleY(), ST_SkewX(), ST_SkewY(), ST_Georeference()**
  - **ST_SRID(), ST_NumBands()**
  - **ST_Metadata()**
- **Get details about bands**
  - **ST_BandPixelType(), ST_BandNodataValue(), ST_BandPath()**
  - **ST_BandMetaData()**

What You Can Do Now?

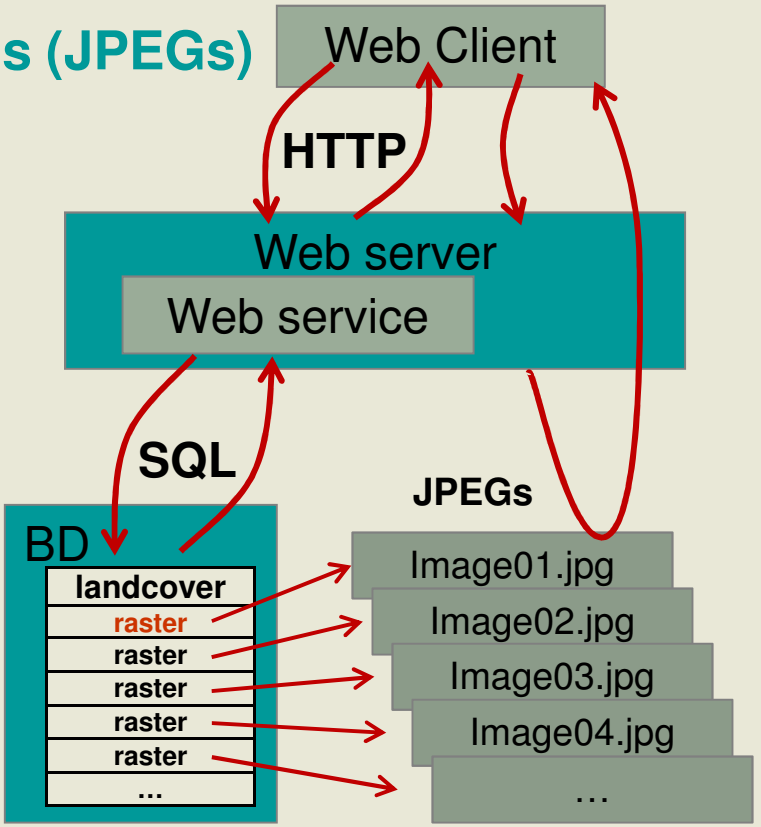
Store and manage rasters in the database…

- Change the georeference and the spatial reference
  - ST_SetScale (), ST_SetSkew(), ST_SetUpperLeft(), ST_SetGeoReference
  - ST_SetSRID()
- Change a band nodata value
  - ST_SetBandNodataValue()
  - ST_SetBandNodataValue(rast, NULL) –to unset nodata value
- Reproject rasters
  - ST_Transform(rast, srid, algorithm, maxerr)
  - NearestNeighbour, bilinear, cubic, cubic spline, lanczos
  - Done with GDAL

What You Can Do Now?

Store and manage rasters stored outside the database…

- Provides faster loading and export of files for desktop application
- Provides faster access for web applications (JPEGs)
- Avoid useless database backup of large datasets not requiring edition
- Avoid importation (copy) of large datasets into the database
- Provides an efficient SQL API to manipulate/analyse raster files
- All functions should eventually works seamlessly with out-db raster
- Data read/write with GDAL (many formats)



What You Can Do Now?
Dump rasters from the database…

- With the GDAL driver 'PostGISRaster'
  - Developed and maintained by Jorge Arévalo
- Read only and still needs optimization
- Two modes
  1. ONE_RASTER_PER_ROW
  2. ONE_RASTER_PER_TABLE (limited)
- gdal_translate "PG:host='localhost' dbname= 'myDB' user= 'me' password= 'toto' table= 'myTable' mode='2' " outputFile.tif

What You Can Do Now?
Get raster statistics…

- **ST_SummaryStats(raster)**
  - Return a set of (min, max, sum, mean, stddev, count (of withdata pixels)) records
  - 10 seconds for one SRTM tile of 3600 x 3600 pixels, 70MB
- **ST_Histogram(raster, bin, width[ ])**
  - Return a set of (min, max, count, percent) records for an array of bins
- **ST_Quantile(raster, quantiles[ ])**
  - Return a set of values for an array of quantile
- **ST_ValueCount(raster, values[ ])**
  - Return the frequency for an array of value

All stats function have:

- A exclude_nodata_value parameter
- A version working on a coverage of many tiles
- A sample_percent parameter (except ST_ValueCount())

UCDAVIS
UNIVERSITY OF CALIFORNIA

What You Can Do Now? Display rasters…

- **Display the true raster**
  - **QGIS plugin by Maurício de Paulo *(mauricio.dev@gmail.com)***
  - **gvSIG plugin by Nacho Brodin *(ibrodin@prodevelop.es)***
  - **MapServer**
  - **Normally any software using GDAL to read raster and allowing passing database connection parameters to GDAL**

- **Display a vectorization of the raster**
  - **OpenJump**
    - **SELECT ST_AsBinary((ST_DumpAsPolygons(rast)).geom),**
      (ST_DumpAsPolygons(rast)).val
      FROM srtm_tiled
      WHERE rid=1869;
  - **ArcGIS 10**
    - **Add Query Layer (same as OpenJump but without ST_AsBinary())**
  - **Any software displaying vector PostGIS queries**

What You Can Do Now? Edit rasters…

- **ST_SetValue(raster, x, y, newval)**
  - ST_SetValue(raster, x, y, pt geometry)
  - More ways to set raster values are planned

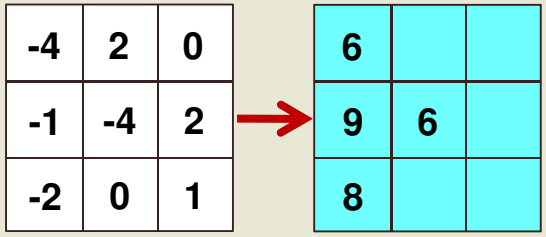
- **ST_Reclass(raster, reclassexpr, pixeltype, nodataval)**
  - reclassexpr is a text string like '0-87:1-10, 88-254:11-15' meaning map 0 to 87 to 1 to 10 and 88 to 254 to 11 to 15
  - You can reset the nodata value
  - You can pass an array of reclassexpr to reclass a multi-band raster
  - Reclass a SRTM tile to a grayscale three band '8BUI' raster (JPEG)

```
SELECT ST_Addband(ST_Addband(ST_AddBand(ST_MakeEmptyRaster(rast),
            ST_Reclass(rast, '-100-2000:0-255', '8BUI')),
            ST_Reclass(rast, '-100-2000:0-255', '8BUI')),
            ST_Reclass(rast, '-100-2000:0-255', '8BUI'))
  FROM srtm_22_03
```

UC DAVIS
UNIVERSITY OF CALIFORNIA

What You Can Do Now?
Edit rasters…

- **ST_MapAlgebra(raster, band, expression, nodatavalueexpr, pixeltype)**



- **Expressions are evaluated by the PostgreSQL parser**
  - **Any, really any, complex SQL expression**
  - **e.g. 'SQRT(rast)/POWER(rast, 3) + ACOS(rast/(rast+1))'**
  - **e.g. 'CASE WHEN rast < 0 THEN rast+10 ELSE NULL END'**
- **A nodatavalueexpr allow specifying an alternative expression when the pixel is nodata**
- **SELECT ST_MapAlgebra(rast, 'rast/2', '32BF', '0') FROM srtm_22_03**

What You Can Do Now?

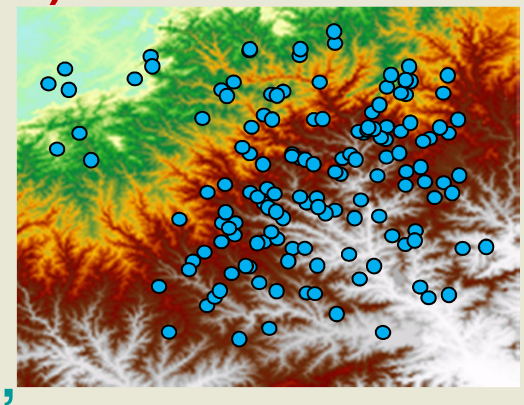
Convert rasters to any GDAL format with SQL…

- **ST_GDALDrivers()**
  - Display the list of GDAL driver available with your version of GDAL
  - SELECT (ST_GDALDrivers()).*
- **ST_AsGDALRaster(rast, format, options[ ])**
  - SELECT ST_AsGDALRaster(rast, 'JPEG')
    FROM srtm_22_03
- **ST_AsTIFF(raster, nbands[], compression)**
  - Compression % can be specified after the compression 'JPEG80'
- **ST_AsJPEG(raster, nbands[], quality)**
- **ST_AsPNG(raster, nbands[], compression)**

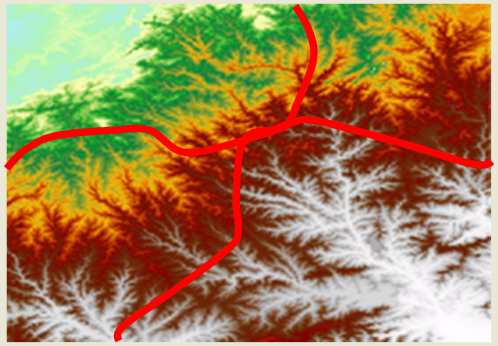
UCDAVIS
UNIVERSITY OF CALIFORNIA

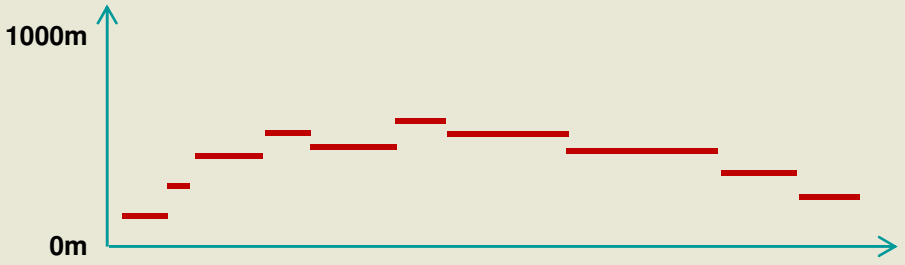
What You Can Do Now?
Do raster/vector analysis…

- **Extract ground elevation values for lidar points…**
  - SELECT lidarPtID, ST_Value(rast, geom) elevation
    FROM lidar, srtm WHERE ST_Intersects(geom, rast)



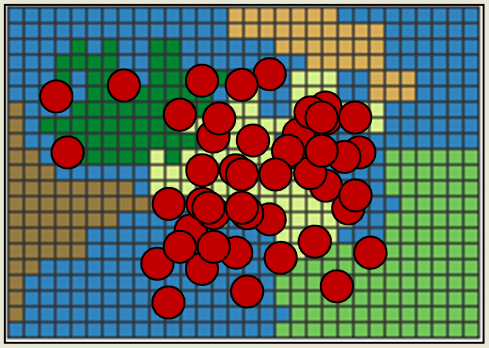
- **Intersect a road network and extract elevation values for each road segment**
  - SELECT roadID,
    (ST_Intersection(geom, rast)).geom road,
    (ST_Intersection(geom, rast)).val elevation
    FROM roadNetwork, srtm WHERE ST_Intersects(geom, rast)





What You Can Do Now?
Do raster/vector analysis…

- **Compute the mean temperature around a series of point**
  1. `CREATE TABLE pointBuffers AS`
     `SELECT pointID, ST_Buffer(geom, 200) FROM pointTable`
  2. `SELECT pointID, (gv).geom pointBuffer, (gv).val temp`
     `FROM (SELECT pointID, ST_Intersection(geom, rast) gv`
     `FROM pointBuffers, temperature`
     `WHERE ST_Intersects(geom, rast)`



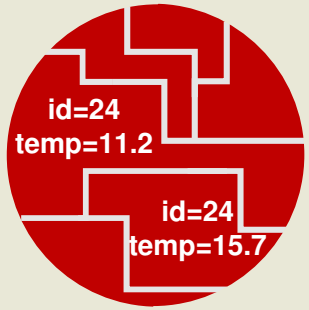
pointBuffers	
geom	pointid
polygon	24
polygon	46
polygon	31
polygon	45
…	…

∩

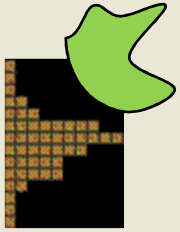
temperature
raster
raster
raster
raster
…

=

result		
geom	pointID	temp
polygon	24	11.2
polygon	53	13.4
polygon	24	15.7
polygon	23	14.2
…	…	…

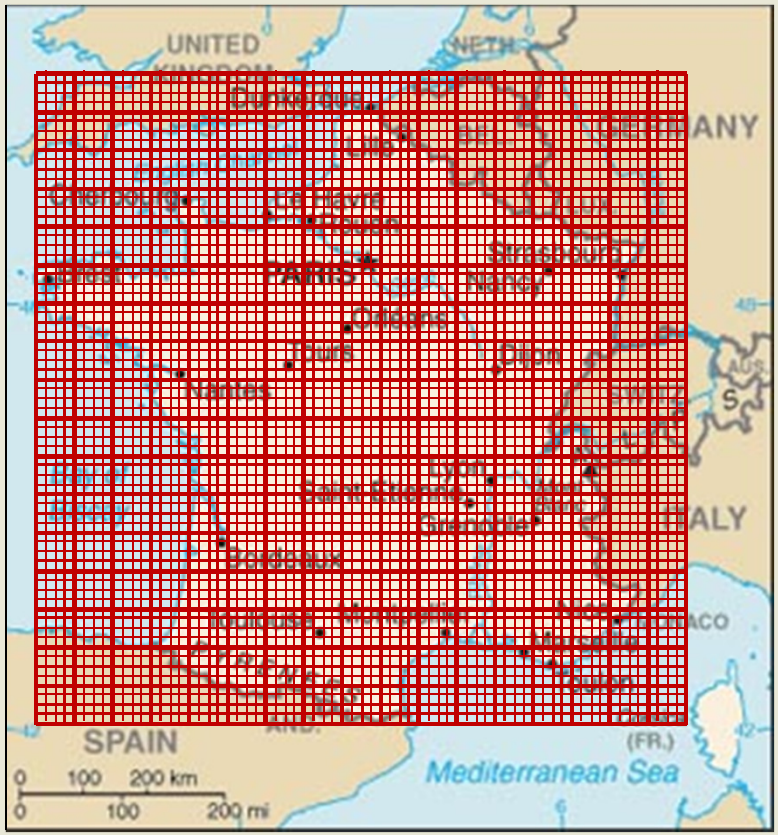


- **Results must be summarized per buffer afterward**
- **All analysis functions take nodata values into account**
- **See the tutorial in the wiki**



What You Can Do Now?

Create a high resolution analysis grid for a large area…



Compute the quantities of many variables for each raster cell

- **Road length, mean temperature, population, water surface, river length, Etc…**
- **Easy in vector mode (1 cell = 1 polygon) but what about all of France at 10m?**

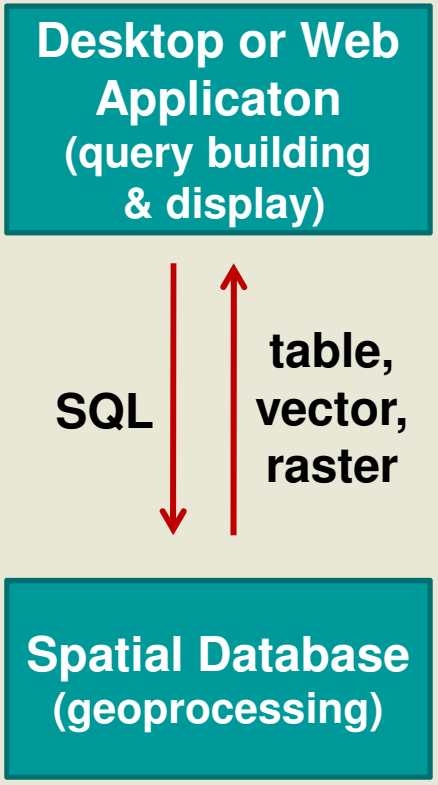
100 000 x 100 000
=
way too many polygons!

- **Manageable in raster format!**
  1. **Intersect your layers with an index raster**
  2. **Summarize per pixel**
  3. **Assign results to new bands**

What You Can Do Now?

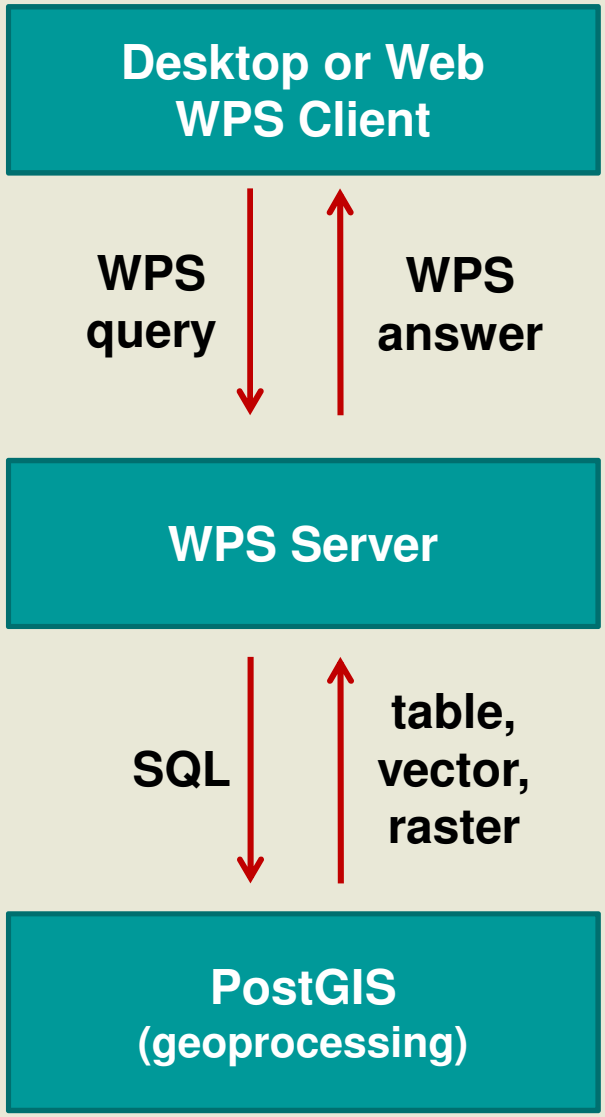
Create a specialised web or desktop GIS application…

- **With the raster API, PostGIS is now a very complete SQL GIS**
  - All data are implicitly **tiled** and **spatially indexed**
  - No need to write **complex** C,C++, Python or JAVA code to manipulate complex geographical datasets.
  - **Use SQL**: The most used, most easy and **most minimalist though complete** language to work with data in general. Easily **extensible** (PL/pgSQL)
  - Keep the **processes close to the data** where the data should be: in a database!
- **Lightweight multi-users specialized desktop and web GIS applications**
  - All the (geo)processing is done **in the database**
  - Applications become **simple SQL query builders** and **data** (results) **viewers**



What You Can Do Now?

Implement a WPS server raster/vector geoprocessor…



What You Can Do Now?

Develop new raster processing functions…

- ST_MakeEmptyRaster()
- ST_AddBand()
  - Empty band or copy a band from another raster
- All georeference setters
  - ST_SetScale (), ST_SetSkew(), ST_SetUpperLeft(), ST_SetGeoReference()
- ST_SetBandNodataValue
- ST_SetValue()
- Coordinates transformation helpers
  - ST_World2RasterCoordX(), ST_World2RasterCoordY(), ST_Raster2WorldCoordX(), ST_Raster2WorldCoordY()
- ST_Intersection() & ST_intersects()
  - To interact with vector data
- Many more…

What You Can Do Now?

Develop new raster processing functions…

- **PL/pgSQL example for ST_DeleteBand**

```
CREATE OR REPLACE FUNCTION ST_DeleteBand(rast raster, band int)
  RETURNS raster AS $$
  DECLARE
      numband int := ST_NumBands(rast);
      newrast raster := ST_MakeEmptyRaster(rast);
  BEGIN
      FOR b IN 1..numband LOOP
                IF b != band THEN
                          newrast := ST_AddBand(newrast, rast, b, NULL);
                END IF;
      END LOOP;
      RETURN newrast;
  END;
  $$ LANGUAGE 'plpgsql';
```

Performance?

- Import of 900MB of uncompressed 16BSI GeoTIFF SRTM
  - 13 SRTM files
  - tiled to 48373 100x100 pixels tiles: 3 minutes
  - tiled to 525213 30x30 pixels tiles: 6 minutes
- ST_Intersection() of 814 buffers with the 30x30 900 MB SRTM coverage
  - 4 minutes
- ST_Intersection() of 100 000 lines with a 300 MB landsat image
  - 8 minutes

Comparison with Oracle GeoRaster

Oracle GeoRaster

- Stored as a one to many relation between two types, in two different tables
  - SDO_GEORASTER (raster)
  - SDO_RASTER (tile)
  - Only SDO_RASTER is georeferenced
- Supports (too) many raster features for any kind of raster application
  - bitmap mask, two compression schemes, three interleaving types, multiple dimensions, embedded metadata (colour table, statistics, etc…), lots of unimplemented features
- Hard to load data
- Designed for raster storage

PostGIS Raster

- Stored as a unique type, in one table
  - RASTER (or tile)
  - Each raster is geoferenced
- Supports the minimal set of characteristics for the geospatial industry
  - georeference, multiband, tiling, pyramids, nodata values
- Easy to load data
- Designed for raster/vector analysis

What You Can Do Soon?
Write to PostGIS raster with GDAL…

- **A write GDAL driver do not exist yet.**
- **It should allows**
  - loading raster in the database using **gdal_translate**
  - loading many raster at the same time
  - any application writing to GDAL to write to PostGIS raster
  - tiling a raster to any tile size
  - to create overviews

What You Can Do Soon?

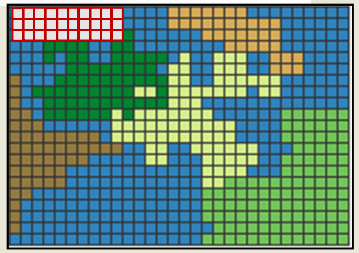
Convert geometries to raster…
Resample/retile a raster coverage…

- **ST_AsRaster(geometry)**
  - Alignment and pixelsize can be determined from:
    1. Parameters
    2. The extent of the geometry
    3. The first encountered segment length
       (to quickly rasterize previously vectorized rasters)
    4. A provided existing raster
- **ST_Resample(raster)**
  - Only realign
  - Resample and realign
  - From parameters or an existing raster
- **ST_Intersection(raster, raster) -> raster**
  - Equivalent to ST_Clip(raster, ST_AsRaster(geometry))
  - Useful for retiling an existing coverage to a new one

What You Can Do Soon? Complex MapAlgebra analyses…

- Already available: One raster version of ST_MapAlgebra()
- Soon: Faster user-defined function version
  - Function taking a pixel value and some parameters and returning a computed value
    - CREATE FUNCTION polynomial(x float,
      VARIADIC args TEXT[])
      RETURNS FLOAT AS $$
      DECLARE
      m FLOAT;
      b FLOAT;
      BEGIN
      m := args[1]::FLOAT;
      b := args[2]::FLOAT;
      return m * x + b;
      END; $$ LANGUAGE 'plpgsql';
    - SELECT ST_MapAlgebra(raster, 'polynomial', ARRAY['1.34', '5.2'])

What You Can Do Soon? Complex MapAlgebra analyses…



- One raster neighbor version
  - User function taking a 3x3, 5x5, 7x7, or more raster and optional parameters and returning a value
  - Useful to implement any focal function ("moving window")
  - Possibility to pass the name of a coverage where to get out-of-bound pixel values
- Two rasters version
  - SELECT ST_MapAlgebra(elev1.rast, elev2.rast, 'rast1 + rast2) / 2', '32BF', 'INTERSECTION')
    FROM elev1, elev2 WHERE ST_Intersects(elev1.rast, elev2.rast)
  - Useful to implement most overlay functions and more
    - ST_Union(raster, raster) -> raster
    - ST_Intersection(raster, raster) - > raster
    - ST_BurnToRaster(raster, geometry, value)…
  - Resample/realign on the fly. Takes care of nodata values.
  - Resulting extent can be FIRST, SECOND, UNION or INTERSECTION.

	-10	0	0
-4	0	-6	2
-1	-4.5	0	1
-2	0	1	

What You Can Do Soon?

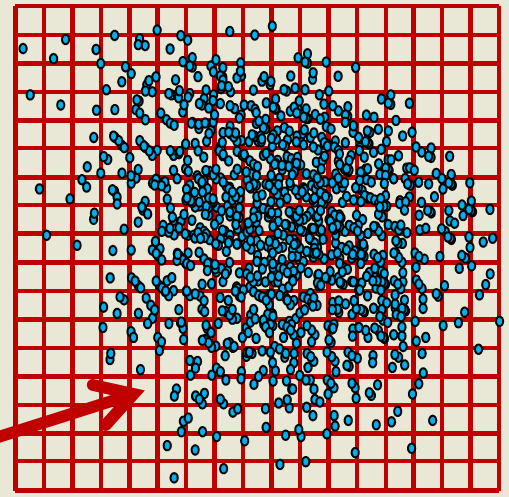
Aggregate many tiles into one raster… (or merge)

- **Use ST_Union as an aggregate function**
  - Taking a **state**, a **temporary** and a **final** function specifying how to aggregate pixel values in a state, a temporary and a final raster
  - User can defines their own expressions or use predefined functions like **FIRST, LAST, MIN, MAX, SUM, MEAN, COUNT**

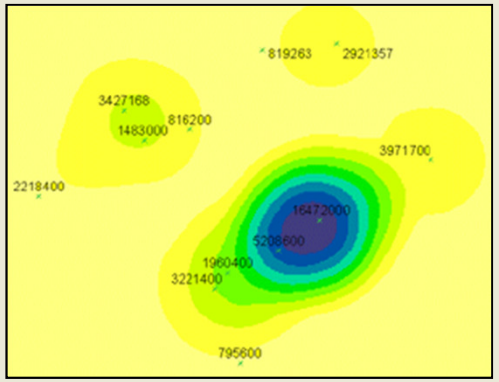
- **Ex. SELECT ST_Union(raster, 'MEAN')**
  - Compute the mean pixel value of many overlapping pixels
  - The state function 'SUM' accumulate pixel values
  - The temporary function 'COUNT' count the number of pixels
  - The final function 'state raster/temporary raster' divide the sum by the count
  - See pl/pgsql code in raster/script/plpgsql/st_union.sql

What You Can Do (maybe not too) Soon?
Interpolate a raster coverage from a point coverage…

- **ST_Interpolate(pts geometry)**
  - Should be an **aggregate** returning one raster (or a set of tiles)
  - Implementing many different interpolation algorythms
    - Nearest neighbor, linear, polynomial
  - Very useful to convert **lidar** data to raster



- **ST_AsDensity(geometry)**
  - **Count** the number of features touching each pixel and then **smooth** the surface using a moving window (neighbor map algebra)



What You Can Do (maybe not too) Soon?

Create a clean raster coverage… from a messy one…

1. Load a bunch of unaligned overlapping rasters (e.g. landsat)

2. ST_SetBrightness() & ST_SetContrast()
   - or ST_NormalizeColor('table', 'rasterColumn')

3. ST_MakeEmptyRasteerCoverage()
   - Create a vector grid or an empty raster coverage based on a set of parameters

4. ST_MapAlgebra(emptyRaster, messyRaster, 'MEAN', 'FIRST') -> raster

What You Can Do (maybe not too) Soon?
Recognize forms from images stored in the DB…

- And automatically convert them to geometries
- Need more research…

Summary

- **PostGIS Raster is multiband, tiled, multiresolution**
  - Each band supports one nodata value, one pixel type.
  - One row = one raster, one table = one coverage.
  - Supports many tile arrangement.
  - Very much like a vector coverage.
  - Import is done the same way as usual with PostGIS: raster2pgsql
- **There are plenty of functions to…**
  - manipulate,
  - edit,
  - do raster and raster/vector analysis,
  - get raster statistics,
  - create new rasters
  - Write web and desktop applications in a client-server context

Thanks!

http://trac.osgeo.org/postgis/wiki/WKTRaster