Store, manipulate and analyze raster data within the PostgreSQL/PostGIS spatial database

Pierre Racine
Research Assistant
Centre for Forest Research

Steve Cumming
Centre for Forest Research
Wood and Forest Science Department
University Laval, Quebec, Canada

FOSS4G Denver
September 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 is as easy to use as the PostGIS GEOMETRY type
    - One table row = one raster (or tile)
    - One table = one coverage
  - Integrated as much as possible with the GEOMETRY type
    - SQL API easy to learn for users already familiar with PostGIS
    - Full raster/vector analysis capacity taking nodata value into account
    - Operators & functions works seamlessly when possible
  - First release with PostGIS 2.0 (soon)

• Development Team
  - Current: Bborie Park, Jorge Arevalo, Pierre Racine, David Zwarg, Regina & Leo Obe
  - Past: Sandro Santilli, Mateusz Loskot

• Founding
  - Steve Cumming through a Canada Foundation for Innovation grant
  - Deimos Space, Davis University, Cadcorp, Azavea, OSGeo
Georeferenced, Multiband, Multiresolution and Tiled Coverages

- Each raster/tile is georeferenced
  - Supports rotation (or skew)

- Supports multiple bands with different pixeltypes in the same raster
  - 1BB, 8BSI, 8BUI, 16BSI, 16BUI, 32BSI, 32BUI, 32BF, 64BF
  - One nodata value per band

- Tiled & indexed
  - No real limit on size
    - 1 GB per tile, 32 TB per coverage (table)
    - Rasters are compressed (by PostgreSQL)
  - Supports irregularly tiled & overlapping coverages

- Other resolutions (or overviews) are stored in sister tables

- List of raster columns available in a raster_columns table similar to the geometry_columns table
Supports Many Raster Arrangements

<table>
<thead>
<tr>
<th>Arrangement</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>a) warehouse of untiled and unrelated images (4 images)</td>
<td>overlaps</td>
</tr>
<tr>
<td>b) irregularly tiled raster coverage (36 tiles)</td>
<td>missing tile</td>
</tr>
<tr>
<td>c) regularly tiled raster coverage (36 tiles)</td>
<td>missing tile</td>
</tr>
<tr>
<td>d) rectangular regularly tiled raster coverage (54 tiles)</td>
<td>smaller tiles</td>
</tr>
<tr>
<td>e) tiled images (2 tables of 54 tiles)</td>
<td>same size tiles</td>
</tr>
<tr>
<td>f) rasterized geometries coverage (9 lines in the table)</td>
<td>empty space</td>
</tr>
</tbody>
</table>
What You Can Do Now?
Store and manage rasters in the database...

- Batch import rasters
  - `raster2pgsql.py -r "c:/temp/mytiffolder/*tif" -t mytable -s 4326 -k 50x50 -l | psql -d testdb`

- Get and set the raster properties
  - Upper left corner coordinates & transformation parameters
  - SRID & number of bands

- Get and set band properties
  - Pixel type & nodata value

- Reproject raster (**ST_Transform**)

- **ST_Resample**(raster), **ST_Rescale**(), **ST_SnapToGrid**()

- Convert a geometry to a raster (**ST_AsRaster**)

- Convert a raster to a set of geometries-values (**ST_DumpAsPolygons**)
What You Can Do Now?
Dump rasters from the database...

- With the 'PostGISRaster' GDAL driver
  - Developed and maintained by Jorge Arévalo

- Read only, optimization in progress

- The write part is still to do (by you?)

- Two modes
  1. ONE_RASTER_PER_ROW
  2. ONE_RASTER_PER_TABLE
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
What You Can Do Now?

Display rasters…

• **QGIS** plugin by Maurício de Paulo *(mauricio.dev@gmail.com)*
• **gvSIG** plugin by Nacho Brodin *(ibrodin@prodevelop.es)*
• **MapServer** through GDAL
  - 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 and compute new rasters...

- **ST_SetValue()** of a pixel
- **ST_Reclass()** a raster
- **ST_MapAlgebra**(raster, band, expression, nodatavalueexpr, pixeltype)

- Expressions are evaluated by the PostgreSQL parser
- You can use any complex SQL expression
- e.g. 'CASE WHEN rast < 0 THEN rast+10 ELSE NULL END'
- You can provide a nodatavalueexpr to handle source nodata values
What You Can Do Now?
Convert rasters to any GDAL format in SQL…

• Get the list of GDAL drivers available (ST_GDALDrivers)

• Convert to any of the available format (ST_AsGDALRaster)
  - SELECT ST_AsGDALRaster(rast, 'USGSDEM')
    FROM srtm_22_03

• ST_AsTIFF(), ST_AsJPEG(), ST_AsPNG()
What You Can Do Now?
Intersects rasters with points and lines...

- Extract ground elevation values for lidar points...
  - SELECT pointID, ST_Value(rast, geom) elevation
  FROM lidar, srtm WHERE ST_Intersects(geom, rast)

- Intersect a road network to 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?
Intersects rasters with polygons...

• Compute the mean temperature for each polygons of a table

```sql
SELECT bufID, (gv).geom buffer, (gv).val temp
FROM (SELECT bufID, ST_Intersection(geom, rast) gv
FROM buffers, temperature
WHERE ST_Intersects(geom, rast))
```

• Results must be summarized per buffer afterward

• All analysis functions take nodata values into account

• Have a look at the tutorial in the PostGIS Raster wiki page!
What You Can Do Now?
Create a high resolution analysis grid for a large area…

Compute values of many variables for each cell of a grid

- Road & river length, mean temperature, population, water surfaces, etc…
- Easy in vector mode (1 cell = 1 polygon) but

- What about all of USA at 10m?

\[ 500 \,000 \times 300 \,000 \]

= Way too many polygons!

- Manageable in raster format!
- 15 000 000 tiles 100x100 pixels
  
  1. Create a raster having a uid per pixel
  2. Intersect your vector layers with your raster grid
  3. Summarize per pixel uid
  4. Create a new band for each variable and assign the values
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...

- Desktop or Web
- WPS Client

WPS query
WPS response

- WPS Server
- attribute, vector or raster table

- SQL

- ZOO
- GeoServer WPS
- PyWPS
- 52 North WPS

- PostGIS (geoprocessing)
Performance?

• Import of 1 GB SRTM DEM files
  - tiled to 48373 100x100 pixels tiles: 3 minutes
  - tiled to 525213 30x30 pixels tiles: 6 minutes
• \texttt{ST\_Intersection()} of 814 buffers with the 30x30 SRTM
  - 4 minutes
• \texttt{ST\_Intersection()} of 100 000 lines with a 300 MB landsat coverage
  - 8 minutes
• Recently selected by the main Canadian governmental provider of geospatial data (\texttt{GeoBase})
  - online on-the-fly and internal elevation product generation
• PostGIS raster is still a baby, many optimizations are still possible
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.
Summary

• Roadmap…
  - Two raster version of ST_Intersection()
  - Neighbor version of ST_MapAlgebra()
  - Two rasters version of ST_MapAlgebra()
  - Aggregate rasters with ST_Union()
  - Statistic functions as aggregates
  - ST_Interpolate() from irregular grid of point (lidar)
  - ST_AsDensity() to produce density maps

• Third party developments…
  - GDAL write driver
  - Support in GeoServer
  - Read/write in FME
What You Can Do Soon?
More complex analyses...

- Two rasters `ST_Intersection()`
  - `ST_Intersection(raster, raster) -> raster`
  - Equivalent to `ST_Clip(raster, ST_AsRaster(geometry))`

- One raster **neighbor** version of `ST_MapAlgebra()`
  - or “focal function” or “moving window” computation
  - User function taking a 3x3, 5x5, 7x7, or more raster and optional parameters and returning a value

- Two rasters version of `ST_MapAlgebra`
  - Useful to implement most overlay functions and more
    - `ST_Union(raster, raster) -> raster`
    - `ST_Intersection(raster, raster) -> raster`
    - `ST_BurnToRaster(raster, geometry, value)`...
  - Resulting extent can be **FIRST, SECOND, UNION or INTERSECTION**.
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/pqsql code in raster/script/plpgsql/st_union.sql
Thanks!

http://trac.osgeo.org/postgis/wiki/WKTRaster
Some extra slides...
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 georeferenced
- 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 Now?
Store and manage rasters stored outside the database...

- By default rasters are stored **INSIDE the database** in the PostGIS raster format
- It is also possible to register in the database rasters stored **OUTSIDE** in the file system
  - Stored in any GDAL format
  - Faster direct access for web apps (JPEGs)
  - Avoid useless database backup of large datasets not requiring edition
  - Avoid importation (copy) of large datasets into the database
  - Provides an easy SQL API to manipulate/analyse raster files
  - Use the –R raster2pgsql.py option
  - All functions should eventually work seamlessly with out-db raster. Now only a few.
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**

```sql
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';
```
What You Can Do Soon?
Write to PostGIS raster with GDAL…

• A write GDAL driver does 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?
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 (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 algorithms
    - 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_MakeEmptyRasterCoverage()**
   - Create a vector grid or an empty raster coverage based on a set of parameters

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