Introducing PostGIS WKT Raster

Seamless Raster/Vector Operations in a Spatial Database

Pierre Racine
Research Professional
Centre d'étude de la forêt
Département des sciences du bois et de la forêt
Université Laval
Quebec, Canada

Boreal Avian Modelling Project

The Canadian BEACONs Project
Boreal Ecosystems Analysis of Conservation Networks

western boreal conservation initiative
Environment Canada
l'initiative de conservation boréale de l'ouest
Environnement Canada

Fondation canadienne pour l'innovation
Canada Foundation for Innovation

PostgreSQL

PostGIS
Geospatial Objects for PostgreSQL

Cadcorp
WORLD LEADING GIS SOFTWARE

deimos

PARAGON CORPORATION

azavea

MichiganTech
Create the Future

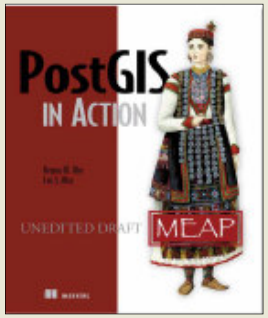
cef
Centre d'étude de la forêt

UNIVERSITÉ LAVAL

España Virtual

Introducing PostGIS WKT Raster

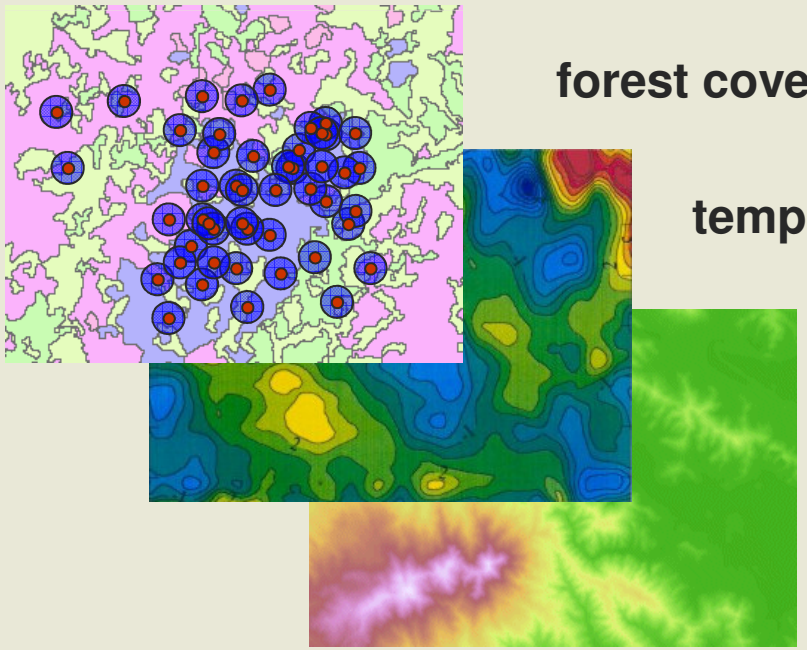
- Support for rasters in the PostGIS spatial database
  - RASTER is a new native base type like the PostGIS GEOMETRY type
  - Implemented very much like and as easy to use as the GEOMETRY type
    - One row = one raster
    - One table = one coverage
  - Integrated as much as possible with the GEOMETRY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



Chapter 13 on WKT Raster

The Context
The Canadian Spatial Data Foundry

- A web site for researchers in **forestry**, **ecology** and **environment**
- Doing **buffer analysis** over HUGE **raster and vector** datasets (covering the extent of Canada)



forest cover

temperature

elevation, etc…

geom	obsID	cutProp	meanTemp	elevation	etc…
polygon	1	75.2	20.3	450.2	…
polygon	2	26.3	15.5	467.3	…
polygon	3	56.8	17.5	564.8	…
polygon	4	69.2	10.4	390.2	…
…		…	…	…	…

Strategies for Implementing the Base Buffering Process

We need code for…	Strategy A	Strategy B	Strategy C
• vector storage & manipulation	database	database	database
• raster storage & manipulation	outside database	database (non-native support)	database (native support)
• analysis processes	specific homemade application	specific homemade application	database

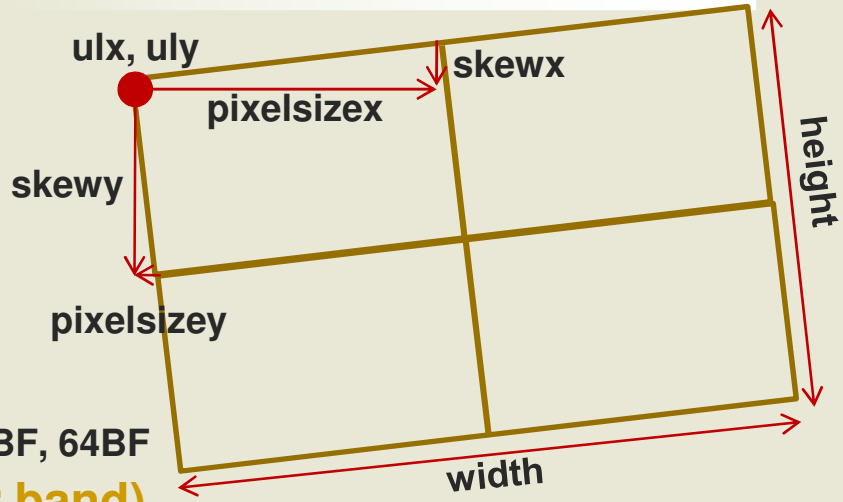
Strategy C (implementing raster as a native type into PostGIS) is a more **elegant and generic solution** answering many more GIS problems

Raster in the Database Requirements
(actually WKT Raster features…)

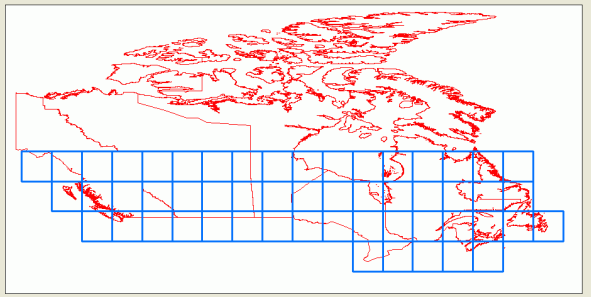
1. Support for georeferenced, multi-band, multi-resolution and tiled raster coverages
   - Efficient storage of non-rectangular coverages
   - Support for nodata value and numerous pixeltypes
2. SQL operators and functions for raster manipulation and analysis
3. SQL operators and functions working seamlessly on raster and vector data
   - Lossless conversion between raster and vector
4. Easy import/export of rasters from/to the filesystem
5. Registration (in the database) of metadata for rasters staying outside the database

1) 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



2) SQL Operators and Functions for Raster Manipulation and Analysis

implemented, being implemented, planned

- All indexing operators: <<, &<, <<|, &<|, &&, &>, >>, |&>, |>>, ~=, @, ~
- Get and set raster properties: width(), height(), upperleft(), setupperleft(), pixelsize(), setpixelsize(), skew(), setskew(), numbands(), hasband()
- Get and set raster band properties: bandpixeltype(), bandnodatavalue(), setbandnodatavalue(), bandhasnodatavalue(), setbandhasnodatavalue(), bandpath(), bandisnodata(), setbandpath()
- Get and set pixel values: value(), setvalue(), values(), setvalues(), reclass(), getstats(), etc…
- Creation: makeemptyraster(), addband(), addrastercolumn(), etc…
- Transformation: resample(), etc…
- Conversion: toimage(), tojpeg(), totiff(), tokml(), etc…

Simple Examples

- SQL

```
SELECT rid, rast, ST_UpperLeftX(rast), ST_UpperLeftY(rast)
FROM mytable
```

- PL/pgSQL

```
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';
```

3) SQL Operators and Functions Working Seamlessly on Raster and Vector

The time is past when we wanted to work on raster data differently than on vector data!

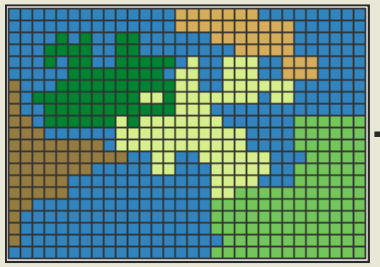
We just want to work on COVERAGES!

(in whatever format they are: vector, raster, TIN, point cloud, etc…)

- Seamless raster versions of existing geometry functions: srid(), setsrid(), convexhull(), envelope(), isempty(), union(), area(), is valid(), centroid(), transform(), rotate(), scale(), translate(), etc…
- Easy raster to vector conversion functions: dumpaspolygons(), polygon(), pixelaspolygon(), pixelaspolygons(), etc…
- Easy vector to raster conversion functions: asraster(), toraster(), interpolate(), etc…
- Major vector-like analysis functions working with rasters: intersection(), intersects(), within(), contains(), overlaps(), etc…
- Major raster-like analysis functions working with vectors: mapalgebra(), clip(), etc…

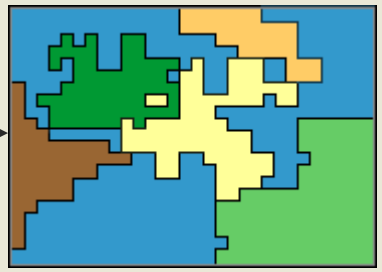
3 b) Lossless Conversion Between Vector and Raster Coverages

- Categorical rasters layers convert **well** to vector layers
  - one variable converts to one column
  - groups together pixels of same value
  - contiguous or not
  - continuous raster layers do not convert as well



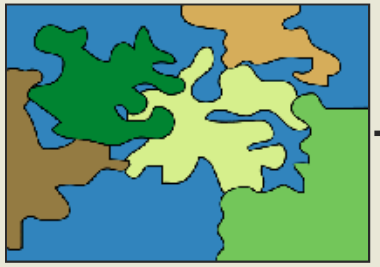
landcover

vectorisation

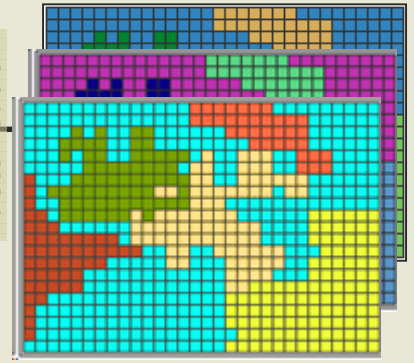


landcover	
geometry	type
polygon	4
polygon	3
polygon	7
...	...

- Vector layers **do not** convert **well** to raster layers
  - each attribute (e.g. type) must be converted to one raster
  - no support for nominal values (e.g. "M34")
  - global values (area) lose their meaning
  - overlaps are lost
  - resolution must be high to match vector precision
  - features lose their unique identities
  - reconversion to the original vector is very difficult or impossible



rasterisation



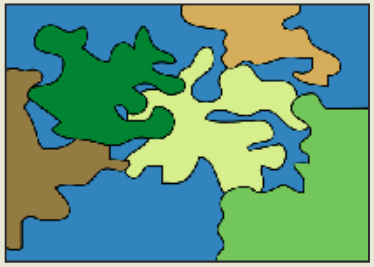
area

landcover			
geometry	type	mapsheet	area
polygon	4	M34	13.34
polygon	3	M33	15.43
polygon	7	M33	10.56
...	...	...	...

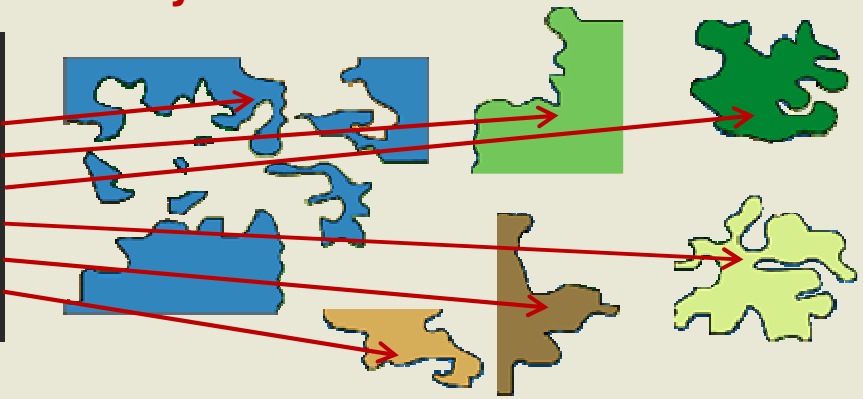
We need a better way to convert vector layers to rasters without destroying objects' identities

3 b) Lossless Conversion Between Vector and Raster Layers

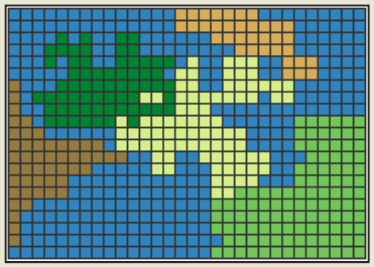
- In a **vector layer**, each object has **its own identity**



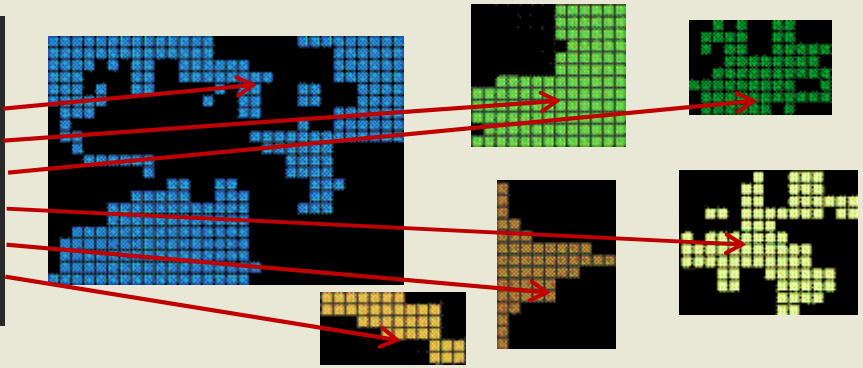
landcover			
geometry	type	mapsheet	area
polygon	4	M34	13.34
polygon	3	M33	15.43
polygon	7	M33	10.56
polygon	9	M34	24.54
polygon	5	M33	23.43
polygon	2	M32	12.34
...	...	...	...



- In a **raster layer converted** from a vector layer, each object should also **conserve its own identity**



landcover			
raster	type	mapsheet	area
raster	4	M34	13.34
raster	3	M33	15.43
raster	7	M33	10.56
raster	9	M34	24.54
raster	5	M33	23.43
raster	2	M32	12.34
...	...	...	...

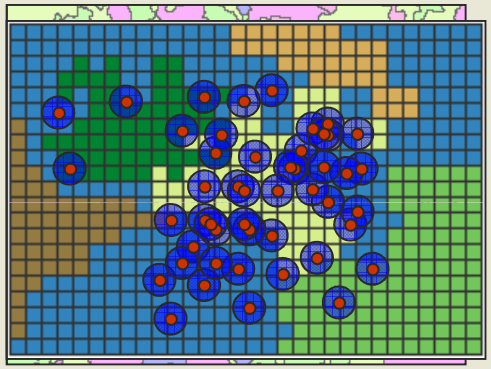


- Each "raster object" has **its own georeference**
- Black pixels are **"nodata values"**
- Like vectors, raster objects may or may not **overlap**
- **Raster algorithms** can be used on the whole layer after a **"blend"** of the objects into a single raster

Rasters become just another way to store geographic features in a more expressive vector object-oriented-like style

ST_Intersection (implemented)

- The goal is to be able to do **overlay operation on coverages** the **same way** we are used to do them on vector coverage but **without worrying if data are stored in vector format or raster format.**



observ	
geom	obsid
polygon	24
polygon	31
polygon	45
...	...

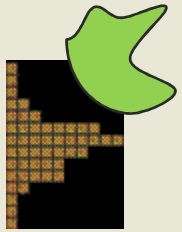
∩

cover	
raster	ctype
raster	4
raster	3
raster	5
raster	2
...	...

=

result			
geom	obsid	ctype	area
polygon	24	4	10.34
polygon	53	3	11.23
polygon	24	5	14.23
polygon	23	2	9.45
...	...	...	...

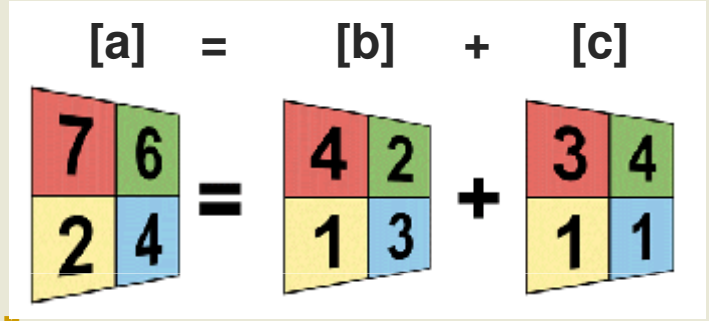
```
SELECT obsid,(gv).geom, (gv).val,  ST_Area((gv).geom ) as area FROM (
    SELECT ST_Intersection(ST_Buffer(observ.geom, 1000), cover.rast ) as gv,
    obsid, ctype
    FROM observation, cover
    WHERE ST_Intersects(ST_Buffer(observation.geom, 1000), cover.rast )
) foo
```



- ST_Intersects **takes nodata value into account.**
- Great **simplification** of applications concepts and graphical user interfaces
- See the **tutorial** on the WKT Raster wiki…

ST_MapAlgebra (being implemented)

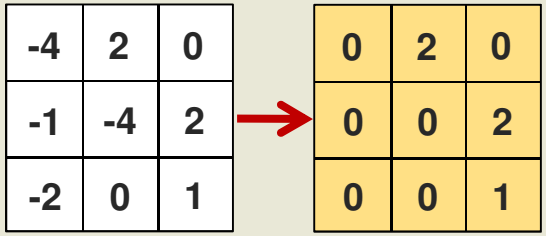
- Generate a **new raster**, pixel by pixel, as a the **result of an expression** involving one, two or more rasters
  - **One input** and **two input** rasters versions
  - Resulting **extent** can the same as be the **first** raster, the **second** raster, the **intersection** or the **union** of both
  - **Misaligned** and **different resolution** rasters are automatically **resampled** to first or second raster
  - **Absent values** (or nodata values) are **replaced with NULL** or a **provided value** (so we can refer to them in expressions)
  - Resulting **pixeltype** can be specified
  - Will allow referring to **surrounding** or **neighbor tile pixels** values for focal & zonal functions. i.e. 'rast2[-1, -1]'
  - Expressions are **evaluated by the PostgreSQL SQL engine** so that users can use their own PI/pgSQL functions
  - Will also allow **passing geometries and values** in place of raster for a seamless integration with vector data



[-1,1]	[0,1]	[1,1]
[-1,0]	[0,0]	[1,0]
[-1,-1]	[0,-1]	[1,-1]

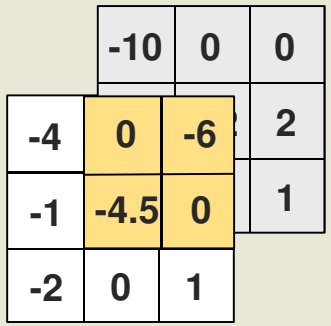
ST_MapAlgebra (being implemented)

- **Example 1:** Reclassifying pixel values (one raster version)
  - SELECT ST_MapAlgebra(rast, 'CASE WHEN rast < 0 THEN 0 ELSE rast END')
    FROM elevation

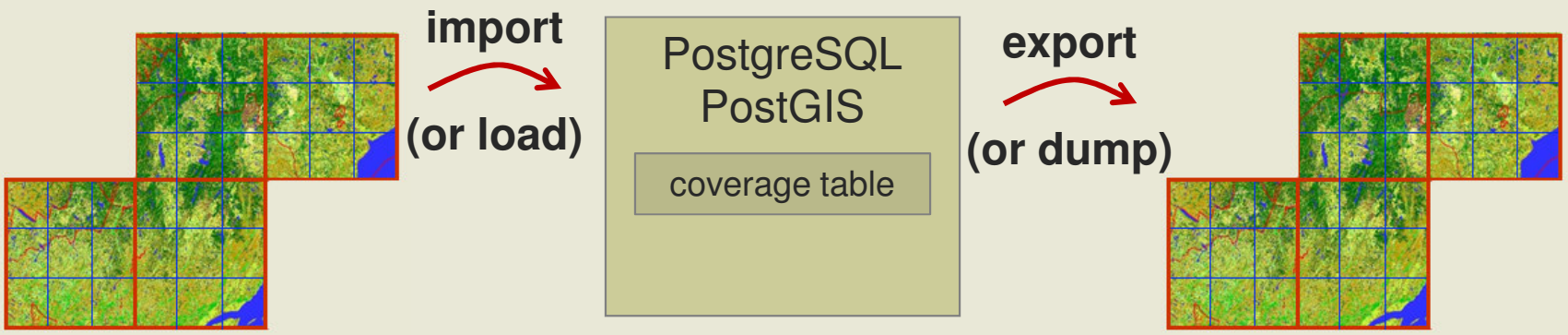


- **Example 2:** Computing the mean + some personnal adjustment (two rasters version)
  - SELECT ST_MapAlgebra(elev1.rast, elev2.rast, 'rast1 + rast2) / 2 + MyAdjustment(rast1, rast2)', '32BF', 'INTERSECTION')
    FROM elev1, elev2 WHERE ST_Intersects(elev1.rast, elev2.rast)

- You can also **intersect** or **merge** rasters, create raster **aggregates**, and **many funny things**!



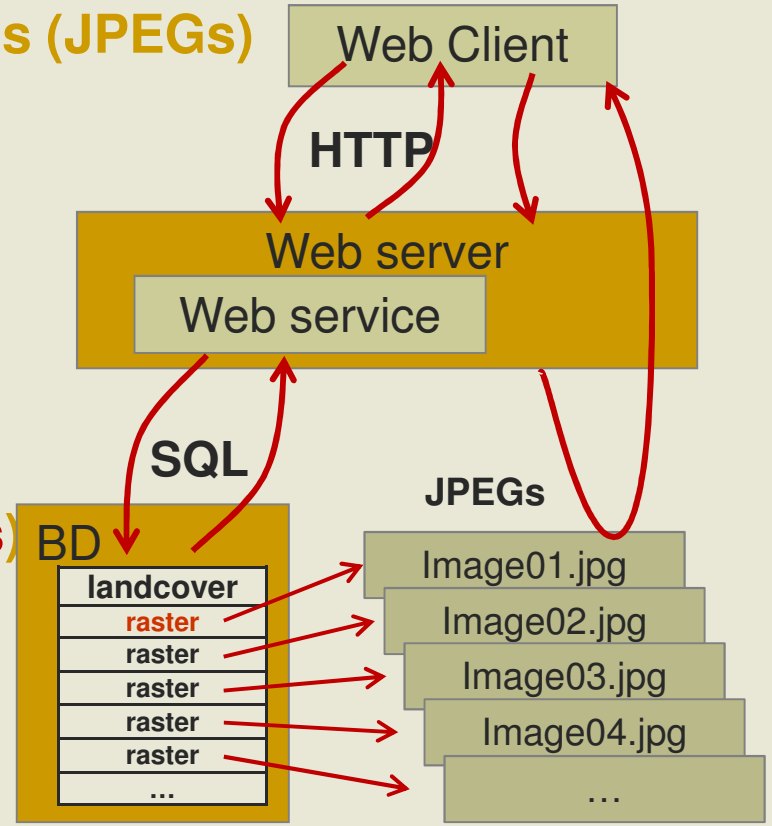
4) Easy Import/Export of Raster From/To the Filesystem



- Import is done with **gdal2wktraster.py**
  - Very similar to PostGIS **shp2pgsql**
  - **Batch import**, production of **overviews** and creation of **tiling** and **index**
  - Can import **many file formats** (thanks to GDAL)
  - Example:
    - **gdal2wktraster.py –r "c:/temp/mytiffolder/*.tif" -t mytable -s 4326 -k 50x50 -l > c:\temp\mytif.sql**
    - **psql -f c:\temp\mytif.sql testdb**
- Export is done using the **GDAL WKT Raster driver**

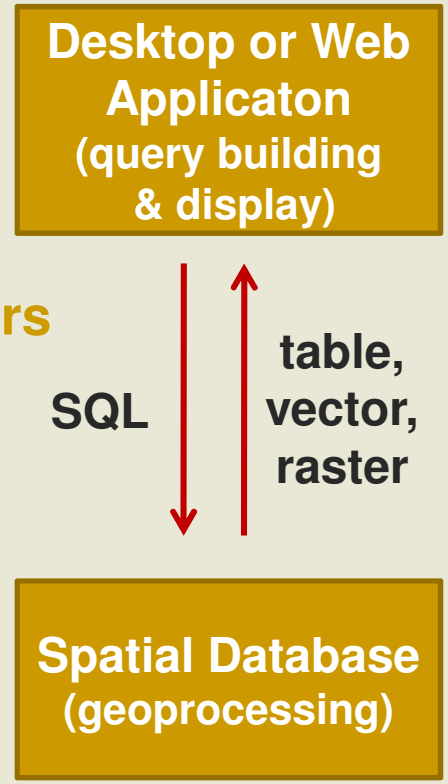
5) Registration of Metadata for Rasters Staying Outside the Database

- Provide **faster loading and export of files** for desktop application
- Provide **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
- All functions should eventually works **seamlessly** with out-db raster
- Data read/write with GDAL (**many formats**)
- Eventual possibility to **convert out-db raster** to **in-db raster** and hence, to load rasters in the DB using SQL
  - CREATE TABLE outraster AS
    SELECT ST_MakeRegisteredRaster('c:/temp/mytiff/*.tif')
  - CREATE TABLE inraster AS
    SELECT ST_MakeBandInDB(rast, band) FROM outraster



A Complete Framework for Light GIS Application Development

- **GIS in the Database:** A complete SQL geospatial API working as seamlessly as possible on any type of coverage
  - Vector, raster, TIN, point cloud, etc…
  - Keep the **processes close to the data** where the data should be: in a database
  - **DBMS client-server architecture** good for desktop and web applications, single and multi-users
- Why SQL?
  - Most used, most easy and **most minimalist though complete** language to work with data in general
  - Easily **extensible** (PL/pgSQL)
- More lightweight applications
  - All the (geo)processing can be done **in the database**
  - Desktop and web applications become **simple SQL query builders and data displayer**

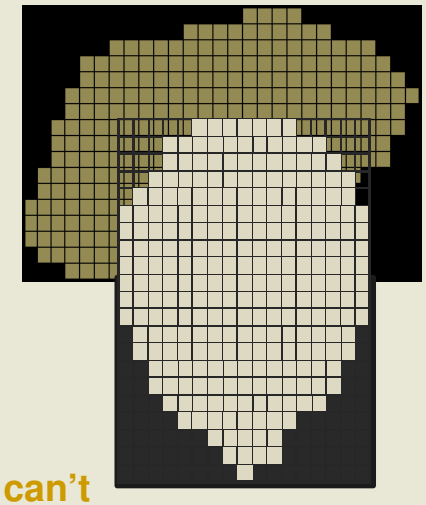


Introducing
WKT Raster "Raster Objects"

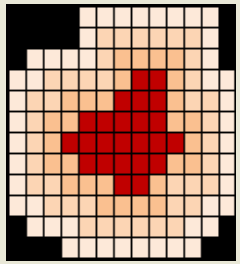
- Rasters created by converting geometries coverage become raster becomes vector like "raster objects".
- Like vector geometries, raster objects:
  - are independent from each others
  - have their own localisation (or georeference)
  - can overlap
  - can change location independently
  - can represent individual objects with their own identity
- Moreover, raster objects can be used to model real life objects better represented as small fields (like fires or fuzzy objects).
- Very new type of GIS object

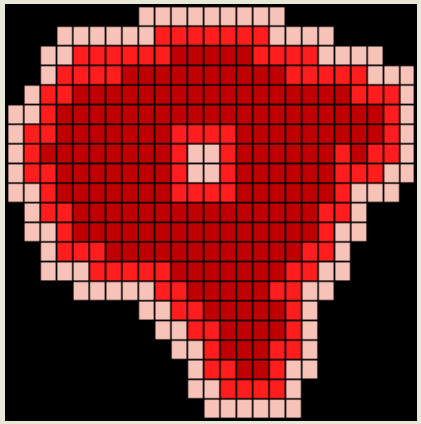
Raster Objects VS Other GIS Objects

- Point and Line Coverages
- Polygon Coverages
  - Objects represent a **constant surface** with an **identity and properties** (like an object in a OO context)



- Raster Object Coverages
  - Constant Raster Objects (categorical)
    - Objects represent a **constant surface** with an **identity and properties** (like a feature or an object)
    - Better modelled as **polygon**, but modelled as raster because they are **better processed using existing raster algorithms** (eg. landcover, basin)
    - E.g.: **land use; land cover**; traditional raster objects that should overlap but can't because they are in raster format (ex. **buffers**, **animal territories**)
  - Variable Raster Objects (field)
    - Objects represent a **variable field** that have an **identity and properties**
    - Generally modelised as a unique raster and **difficult to model as polygons**
    - E.g.: **fire**, **fuzzy objects** (**lakes, land cover, forest stands, soil**), **area of influence, animal territories**





- Traditional Raster Coverages
  - Represent a **variable field** with different values (no unique identity or other properties)
  - E.g.: **elevation**, **climate**, etc…

Comparison with Oracle GeoRaster

See **Jorge Arevalo's presentation**, just following…

Summary

- Lightweight applications (web or desktop) like the Canadian Spatial Data Foundry needs server API to manipulate and analyse vector and raster data. When possible, seamlessly. Ideally in SQL.
- PostGIS WKT Raster aims to provide such an integration
  - Support for multiband, multiresolution, tiled and non-rectangular raster coverages
  - Seamless operators & functions on raster & vector types
    - Lossless conversion between raster & vector layers
    - ST_Intersection and ST_MapAlgebra and many others working seamlessly on raster and vector
  - Storage of metadata for raster stored outside the DB
  - Easy import/export similar to PostGIS shp2pgsql
- A new approach to geospatial application development
  - All GIS processes on raster and vector can now be done in the database
- Introduction of a new kind of GIS raster objects useful for:
  - modelling categorical features needing raster algorithms
  - or fuzzy objects requiring their own identities

Thanks!

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