PostGIS WKT Raster Specifications Version 1.0

Seamless operations between vector and raster layers

Pierre Racine (Pierre.Racine@sbf.ulaval.ca)
University Laval, December 2008

These slides:

- **present an argument for the integration of raster data or of references to raster data into PostGIS**
- **suggest specifications of overlay operation between a vector layer and a raster layer**
- **further discuss the specifications of raster integration**
  - **RASTER as a new type of WKT/WKB geometry**
  - **stored inside or outside of the database**

Why integrate raster in PostGIS?
&
Why are seamless analysis operators important?

The Case for Raster Integration in PostGIS

Why:

- For better or worse, there is great demand for it. Ask yourself:
  - How many people do not use (even try) PostGIS because it does not handle raster?
  - How many people reinvented their own “raster in the database” wheel?
- This is an opportunity to redefine raster (beyond a mere collection of tiles in a filesystem) as a:
  - coherent continuous coverage of measures, stored and indexed into mutually exclusive tiles (for storage efficiency) or objects (for expressiveness) comparable to features in a vector layer
  - layer in which both tile extents and pixels have significance
  - dataset fully integrated with other layers in a GIS context
- This is also an opportunity to implement the foundation of a seamless vector-raster analysis toolkit (overlay operations, map algebra, interpolation, summaries, etc…), given that spatial analysis is one of the next big trend in the geospatial industry.
- PostGIS SHOULD provide a standard solution for every kind of geospatial data if we want it to be the BEST foundation for GIS applications, both desktop and web-based.

The Case for Seamless Operation Between Vector and Raster

Why:

- Most GIS packages offer two different sets of analytical tools: one for raster, one for vector data. This makes GIS methods harder to learn for novices and time consuming for experts.

- It is time to integrate, at the lower level, these tools, allowing us to do analysis independently of the data representation.

- This would ease the development of applications (desktop or web), simplify their GUIs and enhance the user experience.

What should be the result of a typical operation (e.g. intersection) between a vector and a raster layer?

3 examples…

The following slides try to design a solution whereby results are stored as rasters or vector geometries.

Three cases will be examined in each example:

-a geometry/geometry operation with results as a vector layer

-a geometry/raster operation with results as a raster layer

-a geometry/raster operation with results as a vector layer

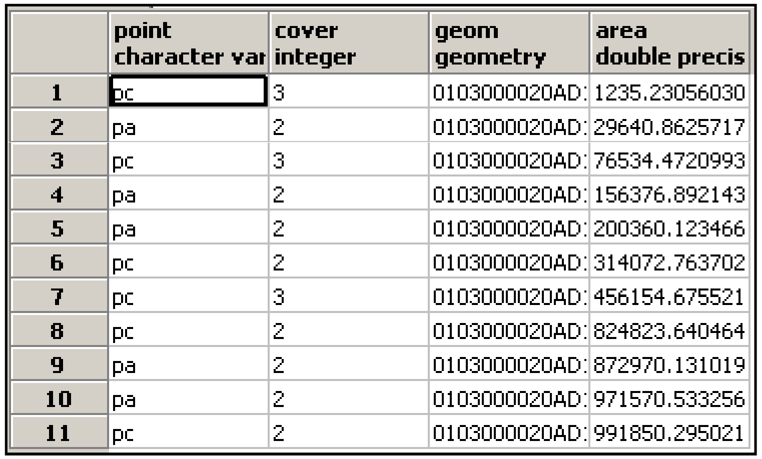
-a raster/raster operation with results as a raster layer

But first a typical SQL postgis geometry/geometry request…

A simplified but typical SQL vector-only overlay operation in PostGIS…

```
SELECT point, cover, geom, ST_Area(geom) as area
FROM (SELECT ST_Intersection(ST_Buffer(point.geom, 1000),cover.geom) as
geom, point, cover
FROM point, cover
WHERE ST_Intersects(ST_Buffer(point.geom, 1000), cover.geom)) cover
ORDER BY area
```

Result:



	point character var	cover integer	geom geometry	area double precis
1	pc	3	0103000020AD:	1235.23056030
2	pa	2	0103000020AD:	29640.8625717
3	pc	3	0103000020AD:	76534.4720993
4	pa	2	0103000020AD:	156376.892143
5	pa	2	0103000020AD:	200360.123466
6	pc	2	0103000020AD:	314072.763702
7	pc	3	0103000020AD:	456154.675521
8	pc	2	0103000020AD:	824823.640464
9	pa	2	0103000020AD:	872970.131019
10	pa	2	0103000020AD:	971570.533256
11	pc	2	0103000020AD:	991850.295021

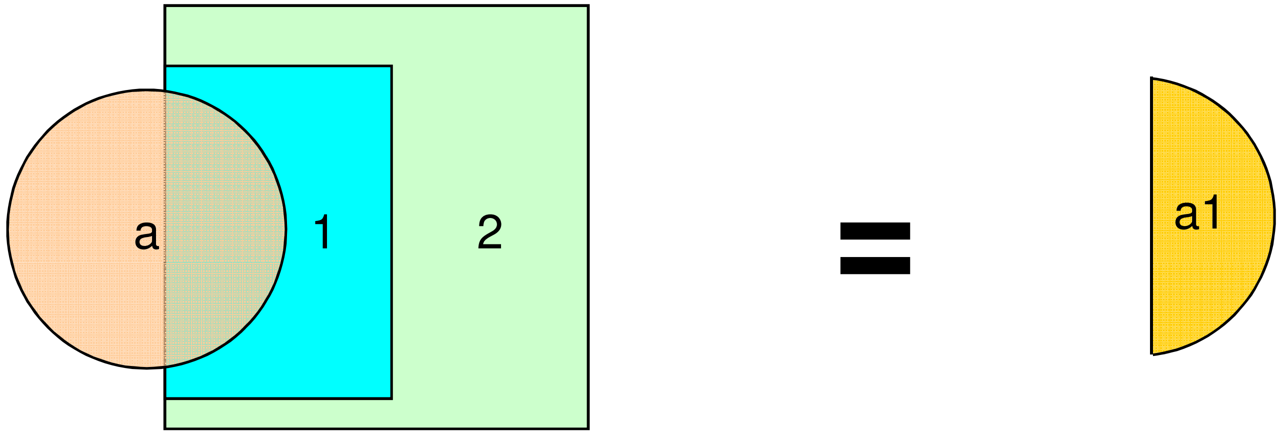
In brief:
- ST_Buffer on a geometry
- ST_Intersection on a geometry
- ST_Area on the result of the previous operation
- ST_Intersects in the 'where' clause (we ignore the &&)

What if the cover layer was a raster coverage instead?

Example 1

Example 1 – Simplest Case
Intersection(geometry, geometry) → geometry

A vector buffer (circle a) is intersected with a vegetation cover - type 1 (blue) and 2 (green)



Tabular form

buffer

geometry	name
polygon(...)	a

∩

cover

geometry	type
polygon(...)	1
polygon(...)	2

=

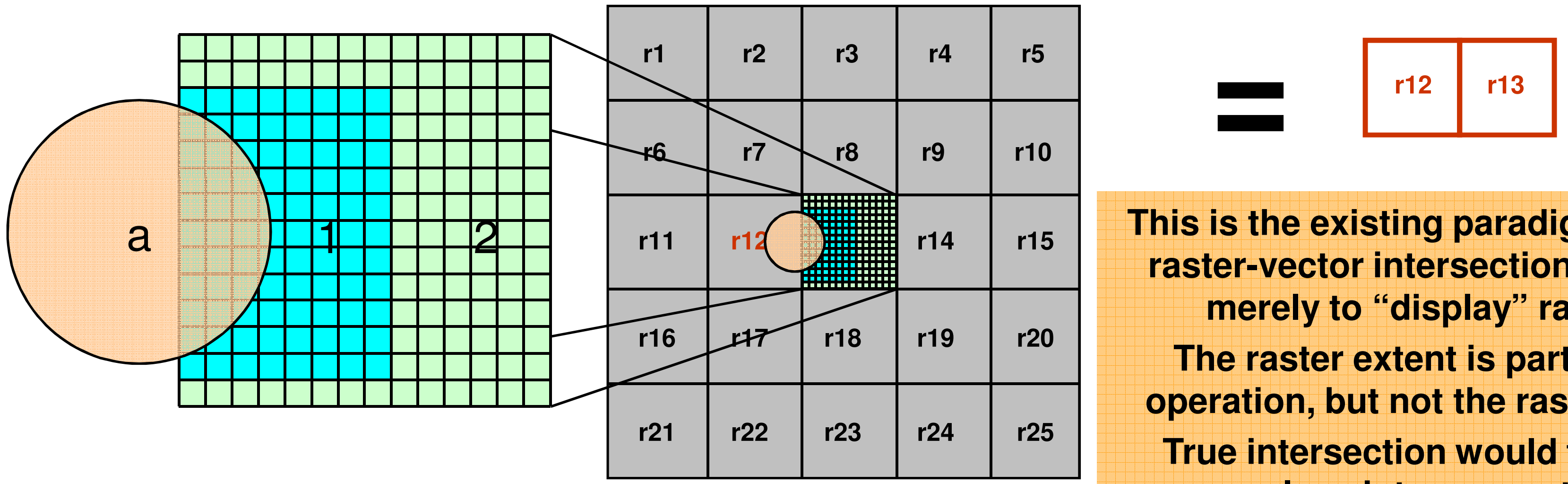
intersection

geometry	bufferName	coverType
polygon(...)	a	1

Here, PostGIS implementation is trivial.

Example 1 – Simplest Case
What do we usually do now?

- Intersection is generally used to select which raster files (tiles) have to be loaded in order to construct a display raster (ex. in ArcGIS or MapServer).
- A rectangle (here a circle), representing viewport extent, is intersected with polygons representing raster (tiles) extents. Every intersecting polygon is part of the result.



This is the existing paradigm where raster-vector intersection is used merely to "display" raster.

The raster extent is part of the operation, but not the raster data.

True intersection would take pixel values into account…

Tabular form

buffer

geometry	name
polygon(…)	a

$\cap$

cover

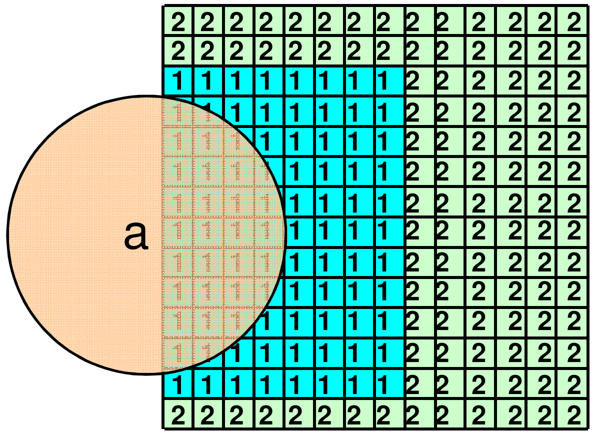
geometry	tile
polygon(…)	…
polygon(…)	r8
polygon(…)	r9
polygon(…)	r10
polygon(…)	…

=

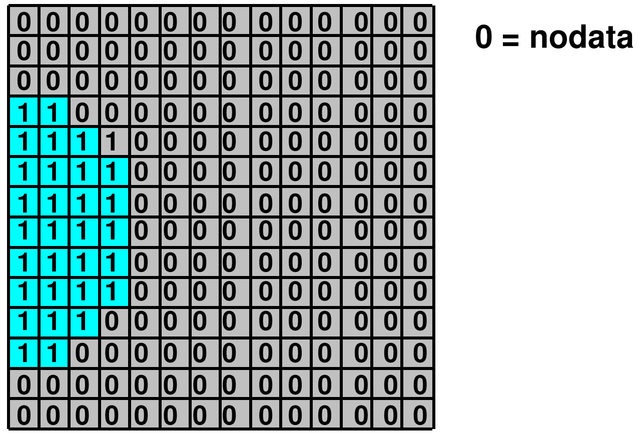
intersection

geometry	bufferName	coverTile
polygon(…)	a	r12
polygon(…)	a	r13

Example 1 – Simplest Case
Intersection(geometry,raster) → raster

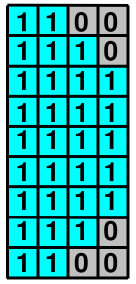


=



Here the result is ALWAYS in raster form

or only



Q1 – What should be the extent of the result? Identical to the source raster, or to the minimal significant area?

Tabular form

buffer	
geometry	name
polygon(…)	a

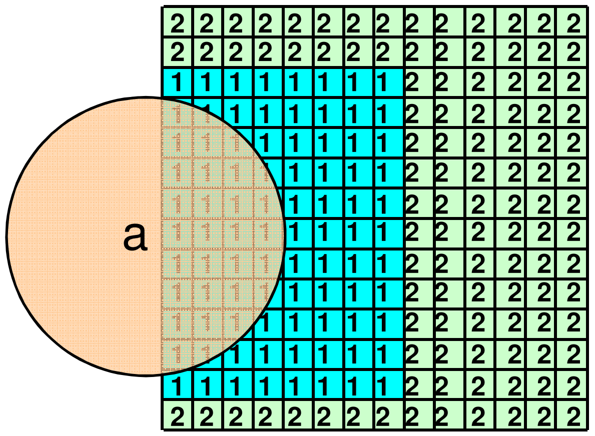
∩

cover
raster
raster(2,2,2…1,1…2,2,2)

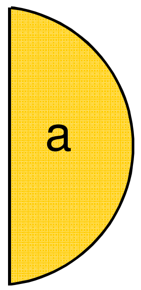
=

intersection	
raster	bufferName
raster(0,0,0,1,1…0)	a

Example 1 – Simplest Case
Intersection(geometry,raster) → geometry



=



Here the result is ALWAYS in vector form

Tabular form

buffer	
geometry	name
polygon(…)	a

∩

cover
raster
raster(2,2,2…1,1…2,2,2)

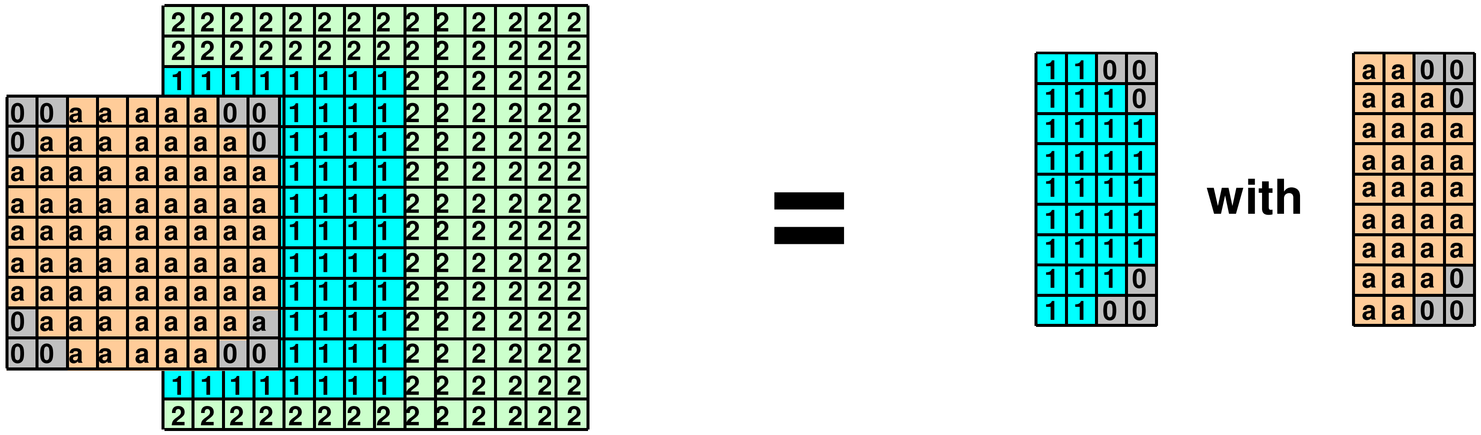
=

intersection	
geometry	bufferName
polygon(…)	a

Here, it is not possible to know the value of intersecting raster pixels (the cover type) since there could be many different values. If we want expressive results in vector form, we must convert rasters to geometries BEFORE intersecting.

Q2-Should the result of overlay operations be vectorial or matricial? Or should we allow both kind of result?

Example 1 – Simplest Case
Intersection(raster,raster) → raster



Tabular form

buffer
raster
raster(2,2,2…1,1…2,2,2)

∩

cover
raster
raster(2,2,2…1,1…2,2,2)

=

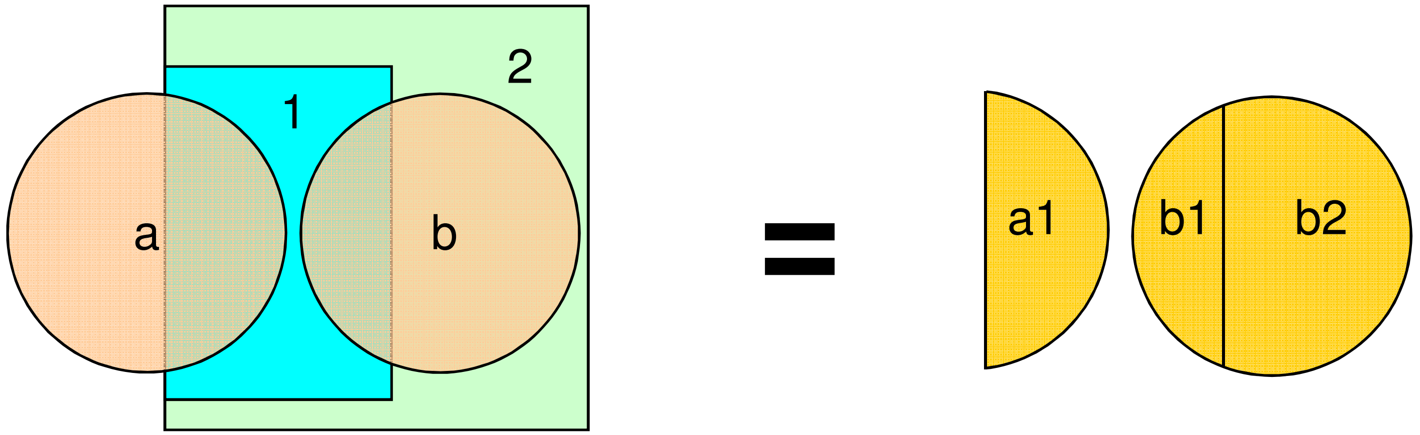
intersection	
name	raster
a	raster(band(1,1,0…0),band(a,a,0…0))

Here, the result must be stored in a multi-band raster.

To obtain a result similar to the geometry/geometry → geometry operation we must vectorize the resulting rasters AFTER the intersection and, morevover, this vectorization must take into account both band.

Example 2

Example 2 – Mutually Exclusive Polygons Intersection(geometry,geometry) → geometry



Tabular form

buffer

geometry	name
polygon(…)	a
polygon(…)	b

$\cap$

cover

geometry	type
polygon(…)	1
polygon(…)	2

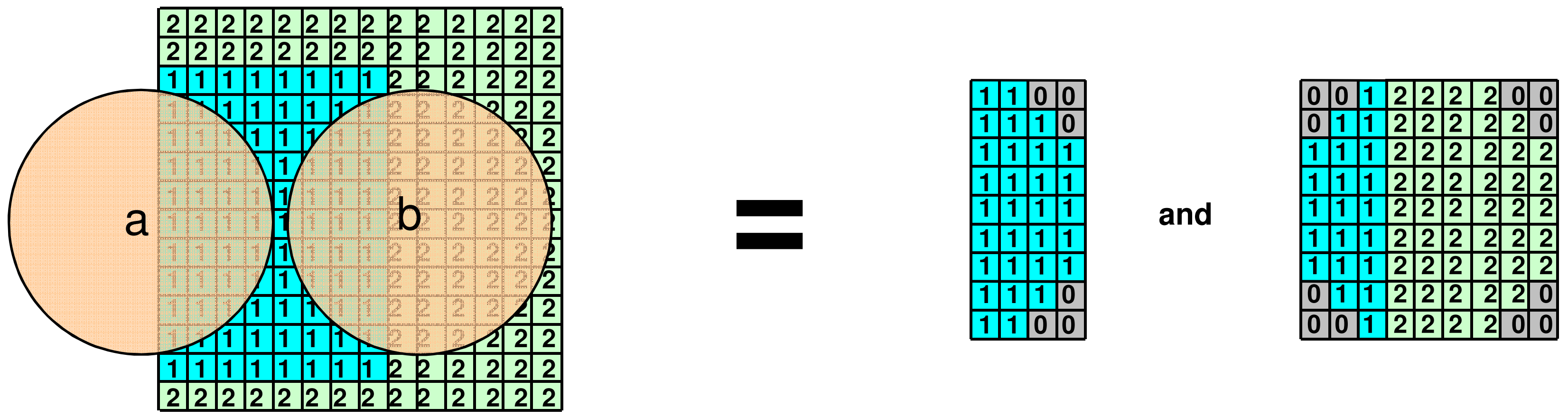
=

intersection

geometry	bufferName	coverType
polygon(…)	a	1
polygon(…)	b	1
polygon(…)	b	2

Here also, PostGIS implementation is trivial.

Example 2 – Mutually Exclusive Polygons
Intersection(geometry,raster) → raster



Tabular form

buffer

geometry	name
polygon(…)	a
polygon(…)	b

∩

cover

raster
raster(2,2…1,1…2,2,2)

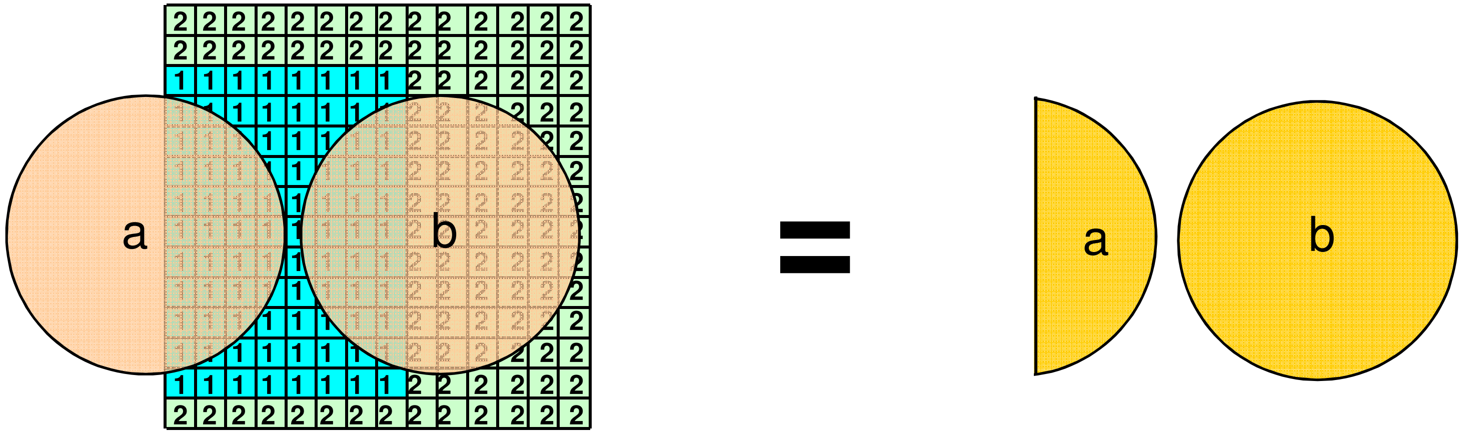
=

intersection

raster	bufferName
raster(1,1,0…0)	a
raster(0,0,1…2,2,2…0)	b

Here, to obtain a result similar to the geometry/geometry → geometry operation we must vectorize the resulting rasters AFTER the intersection.

Example 2 – Mutually Exclusive Polygons
Intersection(geometry,raster) → geometry



Tabular form

buffer	
geometry	name
polygon(…)	a
polygon(…)	b

$\cap$

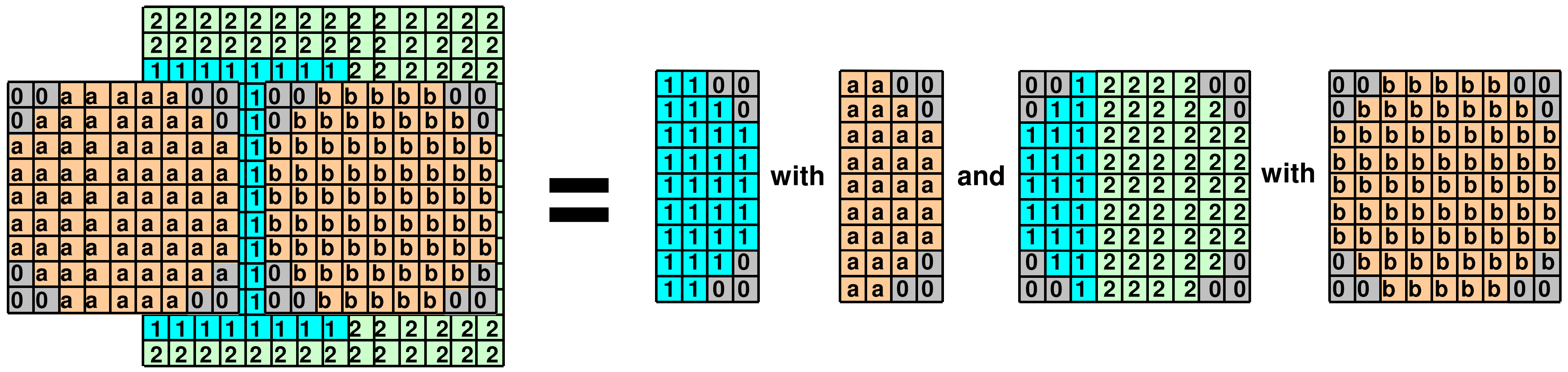
cover
raster
raster(2,2,2…1,1…2,2,2)

=

intersection	
geometry	bufferName
polygon(…)	a
polygon(…)	b

Here also, it is not possible to know the value of intersecting raster pixels (the cover type) without polygonizing the raster according to pixels values. If we want expressive results in vector form, we must then convert rasters to geometries BEFORE intersecting.

Example 2 – Mutually Exclusive Polygons
Intersection(raster,raster) → raster



Tabular form

buffer
raster
raster(0,0,a…0,0…b,0,0)

∩

cover
raster
raster(2,2…1,1…2,2,2)

=

intersection
raster
raster(band(1,1,0…1,0,0), band(a,a,0…a,0,0))
raster(band(0,0,1,2…2,0,0), band(0,0,b…b,0,0))

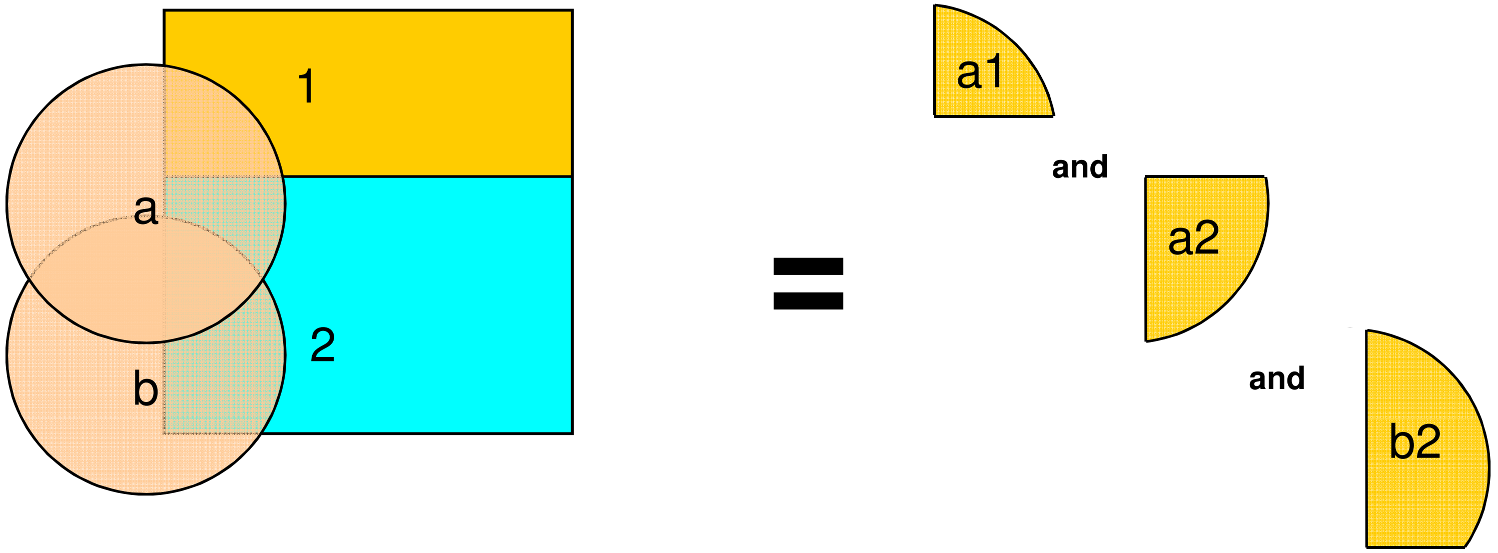
Here also, the result must be stored in a multi-band raster.

To obtain a result similar to the geometry/geometry → geometry operation we must vectorize the resulting rasters AFTER the intersection and, moreover, this vectorization must take into account both band.

Example 3

Example 3 – Non-Mutually Exclusive Polygons

Intersection(geometry,geometry) → geometry



Tabular form

buffer

geometry	name
polygon(...)	a
polygon(...)	b

$\cap$

cover

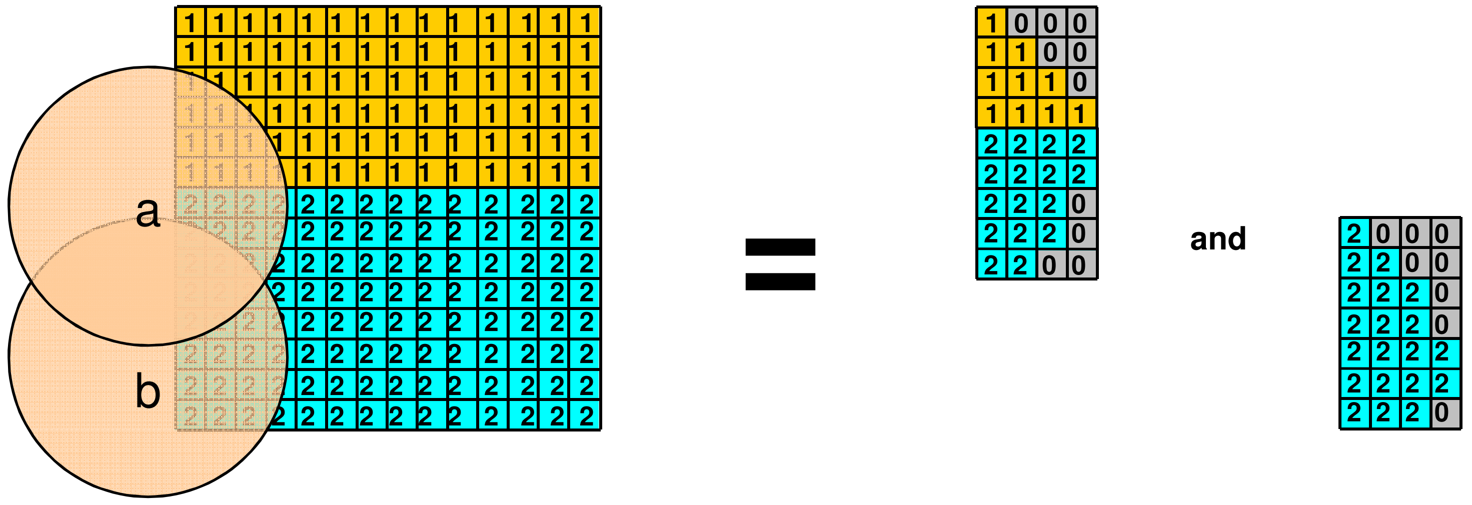
geometry	type
polygon(...)	1
polygon(...)	2

=

intersection

geometry	bufferName	coverType
polygon(...)	a	1
polygon(...)	a	2
polygon(...)	b	2

Example 3 – Non-Mutually Exclusive Polygons
Intersection(geometry,raster) → raster



Tabular form

buffer	
geometry	name
polygon(…)	a
polygon(…)	b

$\cap$

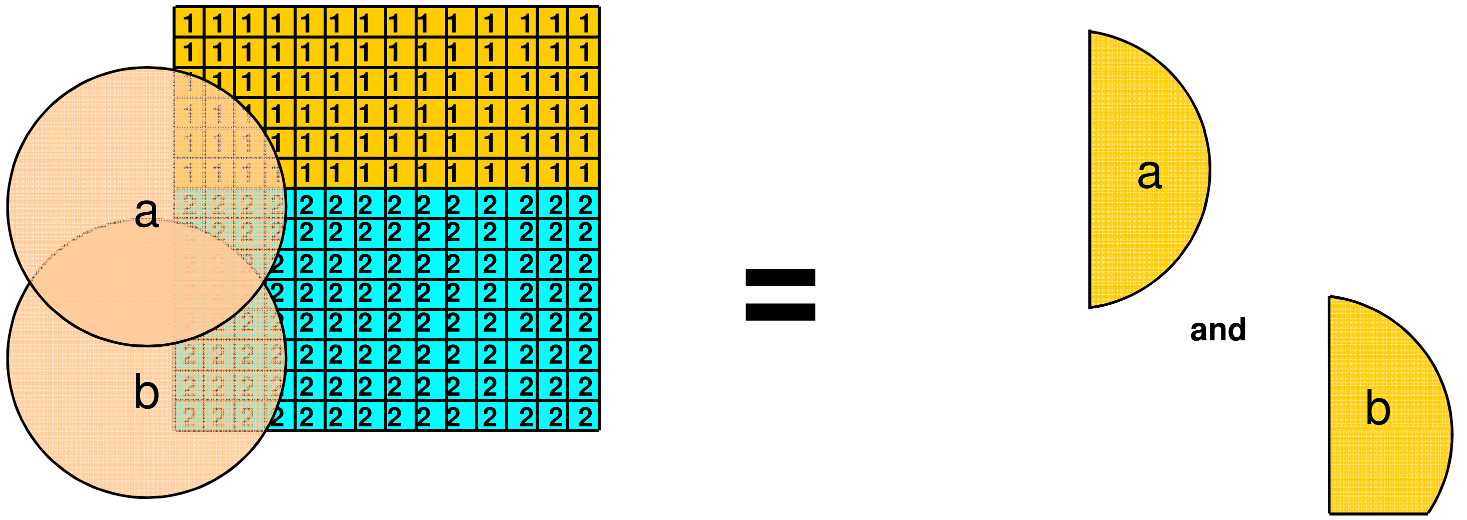
cover
raster
raster(1,1…2,2,2)

=

intersection	
raster	bufferName
raster(1,1,0…2,2…0)	a
raster(2,0…2,0)	b

Here also, to obtain a result similar to the geometry/geometry → geometry operation we must vectorize the resulting rasters AFTER the intersection.

Example 3 – Non-Mutually Exclusive Polygons Intersection(geometry,raster) → geometry



Tabular form

buffer

geometry	name
polygon(...)	a
polygon(...)	b

$\cap$

cover

raster
raster(1,1...2,2,2)

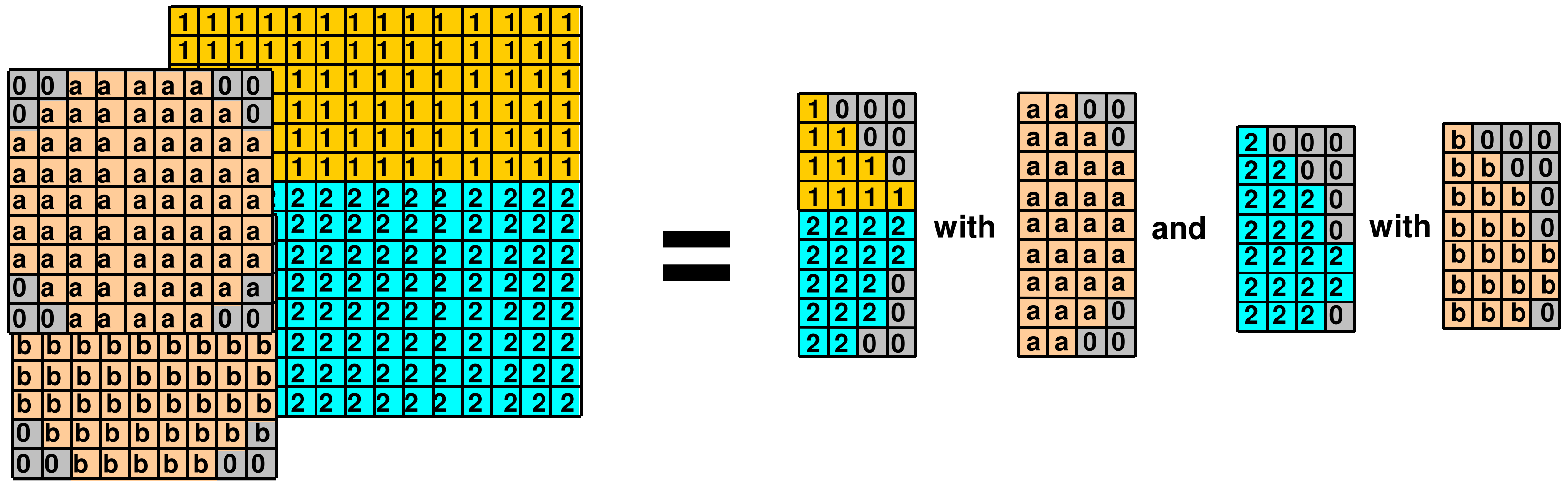
=

intersection

geometry	bufferName
polygon(...)	a
polygon(...)	b

Here also, it is not possible to know the value of intersecting raster pixels (the cover type) without polygonizing the raster according to pixels values. If we want expressive results in vector form, we must convert rasters to geometries BEFORE intersecting.

Example 3 – Non-Mutually Exclusive Polygons
Intersection(raster,raster) → raster



Tabular form

buffer
geometry
raster(0,0,a…a,0,0)
raster(0,0,b…b,0,0)

$\cap$

cover
geometry
raster(1,1…2,2,2)

=

intersection
geometry
raster(band(1,0…2,0,0), band(a,a,0…a,0,0))
raster(band(2,0…2,0), band(b,0…b,0))

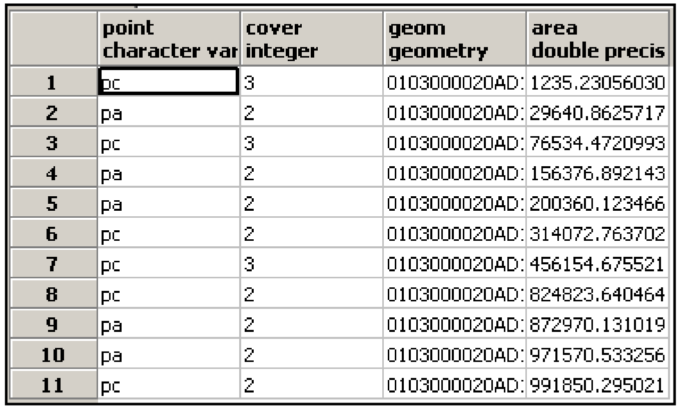
Here also, to obtain a result similar to the geometry/geometry → geometry operation we must vectorize the resulting rasters AFTER the intersection and the vectorization must take into account both band.

Back to our original SQL query…

Our SQL query is very similar to example 3:

- we intersect buffers with a raster forest cover;
- buffers are in vector form and might overlap;

We want a result equivalent to this: →
no matter in which form is
the cover (raster or vector)…



	point character var	cover integer	geom geometry	area double precis
1	pc	3	0103000020AD:	1235.23056030
2	pa	2	0103000020AD:	29640.8625717
3	pc	3	0103000020AD:	76534.4720993
4	pa	2	0103000020AD:	156376.892143
5	pa	2	0103000020AD:	200360.123466
6	pc	2	0103000020AD:	314072.763702
7	pc	3	0103000020AD:	456154.675521
8	pc	2	0103000020AD:	824823.640464
9	pa	2	0103000020AD:	872970.131019
10	pa	2	0103000020AD:	971570.533256
11	pc	2	0103000020AD:	991850.295021

We must be able to compute all the cover areas with the result. We choose to return the result of the intersection in raster form. The resulting rasters are smaller and more simple to vectorize (RT_AsPolygon) AFTER intersecting than if we would have chosen to return the result as geometry. In this latter case, we would have had to vectorize whole and complex rasters BEFORE intersecting. The seamless query looks like:

```
SELECT point, cover, geom, ST_Area(geom) as area
FROM (SELECT RT_AsPolygon(RT_Intersection(ST_Buffer(point.geom, 1000),cover.rast), 'RASTER')
as geom, point, cover
FROM point, cover
WHERE RT_Intersects(ST_Buffer(point.geom, 1000), cover.rast)) cover
```

Only two things are different from the original query:

- the result of RT_Intersection() is explicitely returned as a 'RASTER' when the two inputs are in different forms. (Not when they are in the same form…)
- the resulting raster layer is vectorized with RT_AsPolygon() to isolate each cover raster feature.

Specifications, Open Questions, and Some Query Examples

Specifications for raster in PostGIS

- **We want multi-band support…**
- **We want pixel sizes, variable nodata values and variable pixel types…**
- **Each raster within a coverage is stored as a row in a table.**
- **We don't want to store a specific format (like tiff or jpeg) since we will generally store tiles, not images… Images should be constructed as aggregates of tiles (rows) using GROUP BY.**

We see RASTER as a new type
(like "geometry").

We must store:

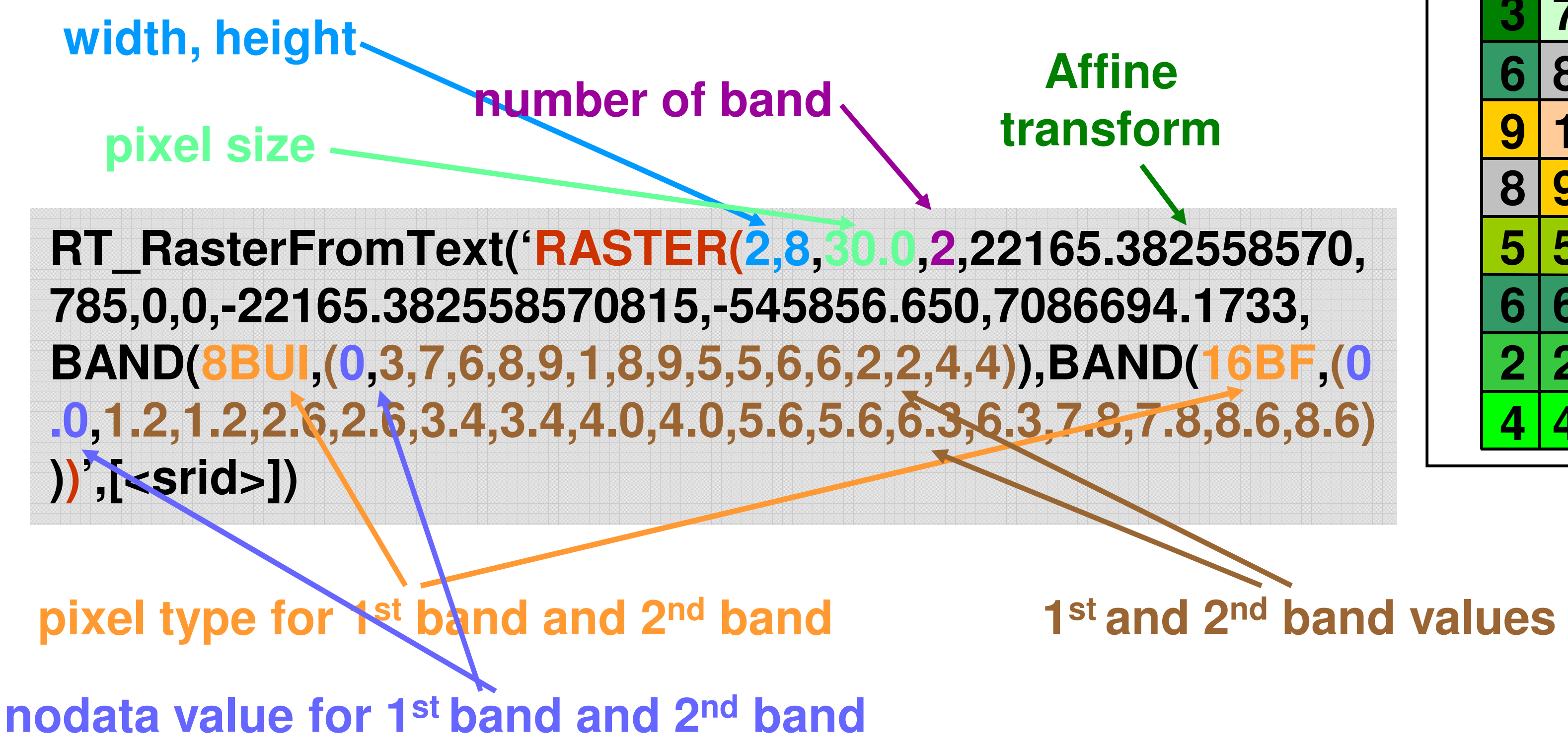
- **For each raster (tile or row)**
  - the width and the height of the raster
  - the pixel size (in the same units as the coordinate system)
  - the number of bands for each raster
  - a polygon representing the bounding box of the raster
  - the georeference (6 floats) (We can probably deduce this from the bbox polygon, the width and the height.)
- **For each band**
  - the pixel type
  - the nodata value
  - the data for each band

Possible pixel types
- 1-bit boolean (1BB)
- 2-bit unsigned integer (2BUI)
- 4-bit unsigned integer (4BUI)
- 8-bit signed integer (8BSI)
- 8-bit unsigned integer (8BUI)
- 16-bit signed integer (16BSI)
- 16-bit unsigned integer (16BUI)
- 32-bit signed integer (32BSI)
- 32-bit unsigned integer (32BUI)
- 16-bit float (16BF)
- 32-bit float (32BF)
- 64-bit float (64BF)

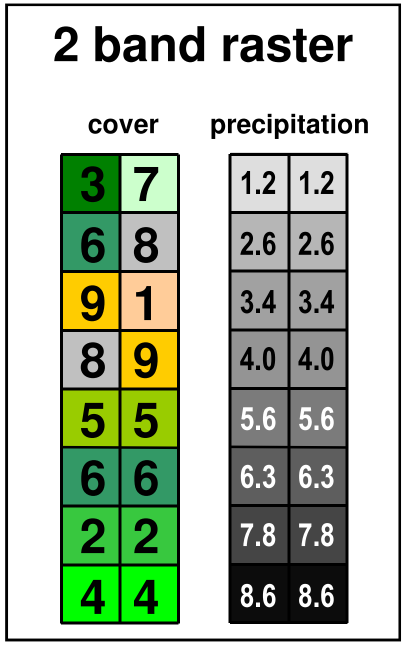
Example…

Example of WKT raster

Creation of a 2x8 raster with 2 bands (8-bit signed integer and 16-bit float) **similar to ST_GeomFromText(text,[<srid>])**



```
RT_RasterFromText('RASTER(2,8,30.0,2,22165.382558570,
785,0,0,-22165.382558570815,-545856.650,7086694.1733,
BAND(8BUI,(0,3,7,6,8,9,1,8,9,5,5,6,6,2,2,4,4)),BAND(16BF,(0
.0,1.2,1.2,2.6,2.6,3.4,3.4,4.0,4.0,5.6,5.6,6.3,6.3,7.8,7.8,8.6,8.6)
))',[<srid>])
```



2 band raster

cover		precipitation	
3	7	1.2	1.2
6	8	2.6	2.6
9	1	3.4	3.4
8	9	4.0	4.0
5	5	5.6	5.6
6	6	6.3	6.3
2	2	7.8	7.8
4	4	8.6	8.6

- **WKB form stores data compressed as deflate**

Raster data inside or outside the database?

- There has been a lot of discussion on this subject. We think it is better to let application developers decide what is best for them given a pro & cons list.
  - Pro inside
    - A single data storage solution (raster are never lost; for small volume, backup is more simple).
    - Faster for analysis (tiled and indexed, no need to extract data from JPEG file).
    - Edition locks provided by DB.
  - Pro outside
    - Reusable files with faster access (TIFF or JPEG) for thin client (WWW) display. No need to convert to JPEG.
    - One time backup (if raster is never edited).
    - No importation (involving copy of huge dataset) needed, just registration.
- We can solve this by allowing raster data (only the band arrays in the previous WKT form) to be stored on disk (in TIFF or JPEG) and only reference them with a path in the WKT/WKB.

```
RT_RasterFromText(‘RASTER(2,8,30.0,2,2,22165.382558570,785,0,0,-
22165.382558570815,-545856.650,7086694.1733,BAND(PT_EXT,0,c:/datastore/
landsat/01b1.tif),BAND(PT_EXT,0.0,c:/datastore/landsat/01b2.tif))’,[<srid>])
```

- Every function listed below work seamlessly wherever the raster is stored.
- Add *RT_GetPath(raster, band)* to know the name of the raster file.
- Add –R option to the importer so no data are copied to the DB, only reference to the files.

Some Questions

- **Georeference: Is it better to…**
  - Store only the bbox and derive the 6-floats-georeference from it?
  - Store only the georeference and derive the bbox from it?
- **Indexing**
  - Is it possible to build a GiST index from bboxes embedded in the raster geometry? If not, how else? Is it a good idea to store it in a different column?
- **New WKT/WKB geometry type or set of new composite types?**
  - Is it better to embed all the raster information in a new WKT/WKB geometry type (like the one described earlier) or to create a set of new composite type like:
    - raster('width', 'height', 'pixelSize', 'nbBand', 'bbox', 'SRID', 'band[]')
    - band('pixelType', 'noDataValue')
- **Pyramids**
  - Should pyramids be stored with each raster tile? Doesn't this lead to an edge effect at lower resolutions? Should them not be stored as a separate raster layer instead, as vector applications do? It would be up to the application to update pyramids when rasters are edited. Maybe both options are useful…
- **Lossless data exchange**
  - It is important that a physical data format supports export and re-import of raster rows without loss of information. Is TIFF a suitable/preferred format for all our needs?

Existing Geometry Constructors to Adapt

Existing for geometries, adapted for raster. (With implementation priority in parenthesis - 1,2 or 3)

- **RT_Centroid**(raster|geometry) → *point geometry (3)*
- **RT_PointOnSurface**(raster|geometry) → *point geometry (3)*
- **RT_Buffer**(raster|geometry, double) → *same type as first arg. (3)*
- **RT_ConvexHull**(raster|geometry) → *same type as input (3)*
- **RT_Intersection**(raster|geometry, raster|geometry, ‘raster’|’geometry’)→*raster/geometry (1)*
- **RT_Difference**(raster|geometry A, raster|geometry B) → *same type as first argument (3)*
- **RT_SymDifference**(raster|geometry,raster|geometry,‘raster’|’geometry’)→*raster/geometry (3*
- **RT_Union**(raster|geometry, raster|geometry, ‘raster’|’geometry’) → *raster/geometry (2)*
- **RT_Accum**(raster set|geometry set, ‘raster’|’geometry’) → *raster/geometry (2)*
- **RT_Envelope**(raster|geometry) → *polygon geometry (1)*
- **RT_Transform**(raster|geometry, SRID) → *same type as input (1)*
- **RT_Affine**(raster|geometry,…) → *same type as input (3)*
- **RT_Translate**(raster|geometry,…) → *same type as input (3)*
- **RT_Scale**(raster|geometry,…) → *same type as input (3)*
- **RT_TransScale**(raster|geometry,…) → *same type as input (3)*
- **RT_RotateZ,Y,Z**(raster|geometry, float8) → *same type as input (3)*
- **RT_Area**(raster|geometry) → *double (2)*

Functions with the ‘raster’|’geometry’ string option return:

- geometries when both input are geometries
- rasters when both input are rasters
- the specified type otherwise

Default is to return a geometry

New Geometry Constructors

New for raster

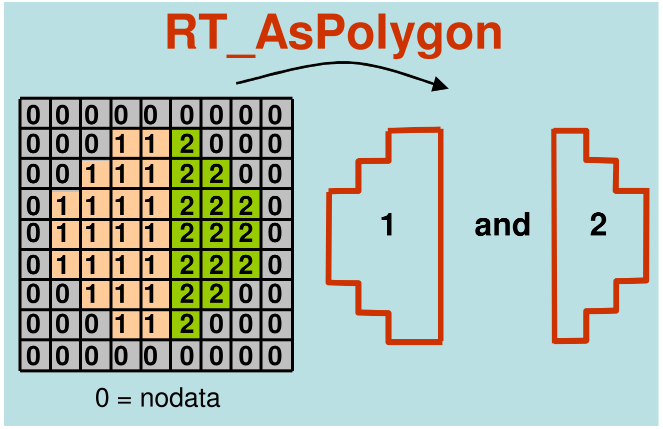
- **RT_RasterFromText**(string, compression, [<srid>]) *(1)*
- **RT_RasterFromWKB**(raster, [<srid>]) *(3)*
- **RT_AsPolygon**(raster) → *polygon geometry set (1)*
- **RT_Shape**(raster) → *polygon geometry (1)*
- **RT_Band**(raster, band) → *raster (1)*
- **RT_Resample**(raster, pixelsize, method) → *raster (2)*

New for raster and geometry

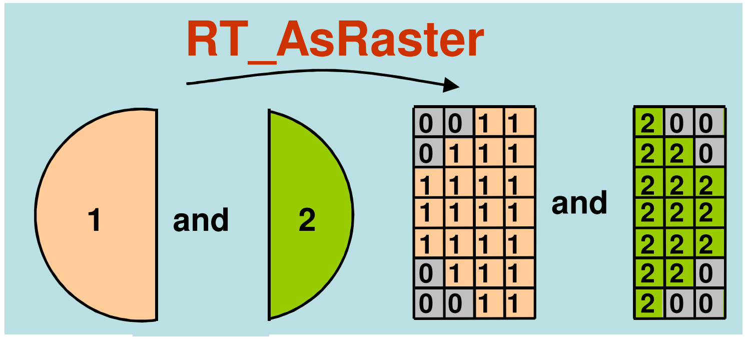
- **RT_Clip**(raster|geometry,geometry) → *same type as first argument (3)*
- **RT_SelectByValue**(raster|geometry, 'expression') → *same type as first argument (2)*
- **RT_Flip**(raster|geometry, 'vertical'|'horizontal') → *same type as first argument (3)*
- **RT_Reclass**(raster|geometry,string) → *same type as first argument (2)*
- **RT_MapAlgebra**(raster|geometry, [raster|geometry,…], 'mathematical expression', 'raster'|'geometry') → *raster/geometry (3)*

New for geometry only

- **RT_AsRaster**(geometry, pixelsize) → *raster (2)*
- **RT_Interpolate**(points, pixelsize, method) → *raster (3)*



0	0	0	0	0	0	0	0	0
0	0	0	1	1	2	0	0	0
0	0	1	1	1	2	2	0	0
0	1	1	1	1	2	2	2	0
0	1	1	1	1	2	2	2	0
0	1	1	1	1	2	2	2	0
0	0	1	1	1	2	2	0	0
0	0	0	1	1	2	0	0	0
0	0	0	0	0	0	0	0	0



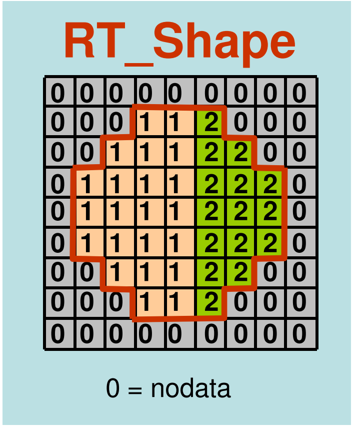
0	0	1	1
0	1	1	1
1	1	1	1
1	1	1	1
1	1	1	1
0	1	1	1
0	0	1	1

2	0	0
2	2	0
2	2	2
2	2	2
2	2	2
2	2	0
2	0	0

Logical Operators to Adapt

Existing for geometries, adapted for raster, return a boolean.

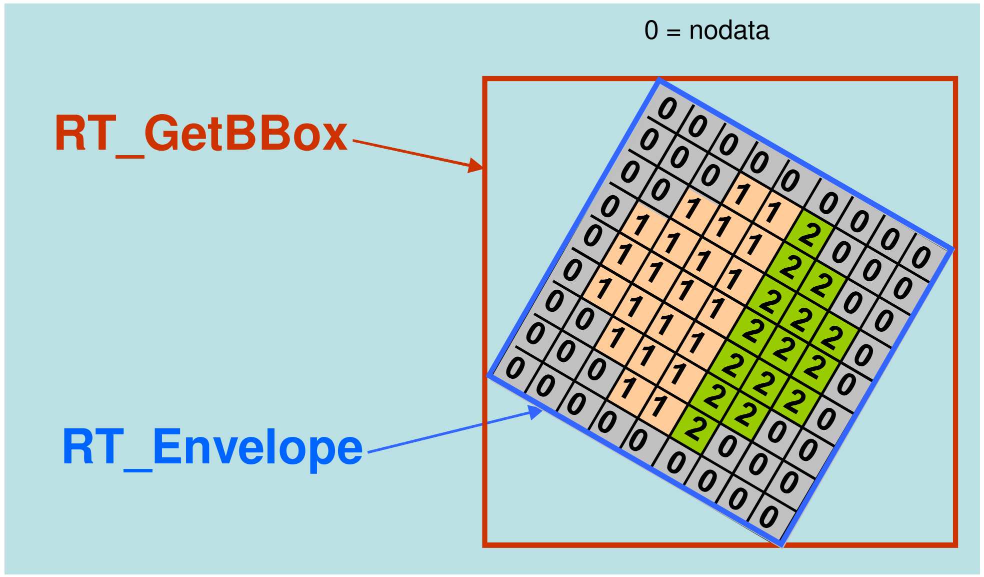
- Operate on two geometries, a geometry and a raster or two rasters.
- On rasters, only pixels with values are taken into account (not the «nodata» values).
- Implies vectorization of the shape of the raster (RT_Shape) before processing in order to isolate pixels with a value from nodata pixels. Should be faster than a true vectorization (RT_AsPolygon) since it does not imply creating different polygons for different values.
- BBox operators (&<, &>, <<, >>, &<|, |>&, <<|, |>>, ~=, @, ~, &&) work with RT_GetBBox(raster|raster) *(1)*
- **RT_Equals**(raster|geometry, raster|geometry) *(3)*
- **RT_Disjoint**(raster|geometry, raster|geometry) *(3)*
- **RT_Intersects**(raster|geometry, raster|geometry) *(1)*
- **RT_Touches**(raster|geometry, raster|geometry) *(3)*
- **RT_Crosses**(raster|geometry, raster|geometry) *(3)*
- **RT_Within**(raster|geometry A, raster|geometry B) *(2)*
- **RT_Overlaps**(raster|geometry, raster|geometry) *(2)*
- **RT_Contains**(raster|geometry A, raster|geometry B) *(2)*
- **RT_Covers**(raster|geometry A, raster|geometry B) *(3)*
- **RT_IsCoveredBy**(raster|geometry A, raster|geometry B) *(3)*
- **RT_Relate**(raster|geometry, raster|geometry, intersectionPatternMatrix) *(3)*



Existing and New Accessors

Existing for geometries, adapted for raster

- **RT_AsText**(raster|geometry) *(1)*
- **RT_AsBinary**(raster, compression) *(2)*
- **RT_AsKML**(raster|geometry) → *KML (3)*
- **RT_AsSVG**(raster|geometry) → *SVG (3)*
- **RT_SRID**(raster|geometry) → *integer (1)*
- **RT_SetSRID**(raster|geometry, integer) *(1)*
- **RT_IsEmpty(**raster|geometry) → *boolean (2)*
- **RT_mem_size**(raster|geometry) → *integer (2)*
- **RT_isvalid**(raster|geometry) → *boolean (2)*



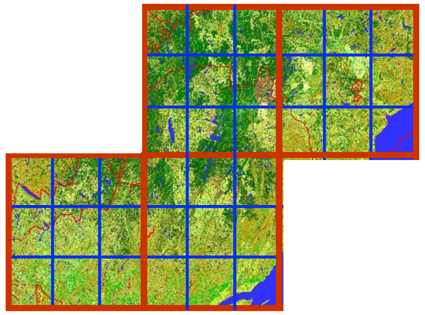
New for raster

- **RT_AsJPEG**(raster, quality) → *jpeg (2)*
- **RT_AsTIFF**(raster, compression) → *TIFF (2)*
- **RT_GetWidth**(raster) → *integer (1)*
- **RT_GetHeight**(raster) → *integer (1)*
- **RT_GetPixelType**(raster, band) → *string (1)*
- **RT_SetPixelType**(raster, band, string) → *string (1?)*
- **RT_GetPixelSize**(raster) → *integer (1)*
- **RT_SetPixelSize**(raster) → *integer (1?)*
- **RT_GetBBox**(raster) → *polygon geometry (1)*
- **RT_GetNumBands**(raster) → *integer (1)*
- **RT_GetNoDataValue**(raster, band) → *string (1)*
- **RT_SetNoDataValue**(raster, band, value) *(1)*
- **RT_GetMaximumValue**(raster, band) → *pixeltype (1)*
- **RT_GetMinimumValue**(raster, band) → *pixeltype (1)*
- **RT_Count**(raster, value) → *integer (2)*
- **RT_GetGeoReference**(raster) → *string (1)*
- **RT_SetGeoReference**(raster, string) *(1)*
- **RT_SetValue**(raster, band, x, y, value) *(3)*

Five ways to use a WKT raster table…

1- As an incomplete non-overlapping tiled coverage

- Stored globally as one table
- many images = 1 table = many tile rows
- Not necessarily rectangular; Many edge tiles may be missing
- Sources is generally a series of non-overlapping images that are converted to small tiles
- This is the most frequent and traditional way of seeing a coverage

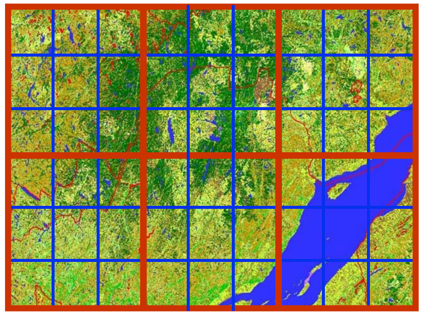


landcover	
tileId	raster
3	raster(…)
4	raster(…)
…	…

Five ways to use a WKT raster table…

2 - As a complete tiled non-overlapping tiled coverage

- Stored globally as one table
- Necessarily rectangular; No missing tiles
- Sources is generally a series of non-overlapping images that are converted to small tiles
- many images = 1 table = many tile rows
- A specific case not as frequent as 1)

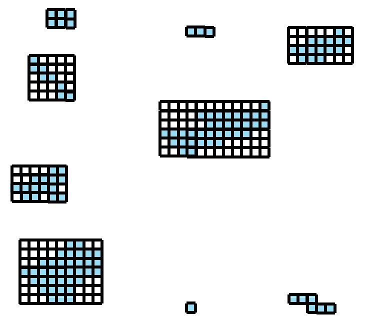


landcover	
tileId	raster
3	raster(…)
4	raster(…)
…	…

Five ways to use a WKT raster table…

3 - As a layer of vector like discrete raster objects

- Stored globally as one table
- Very similar to a vector layer; one raster column with other columns of attributes per row
- 1 layer = 1 table = many raster object rows
- Source is generally the result of an analysis operation implying rasterization of many geometries
  - RT_AsRaster(geom),
  - RT_Intersection(geom, raster,'RASTER')
- Introduced by WKT Raster to support meaningful raster/vector operations

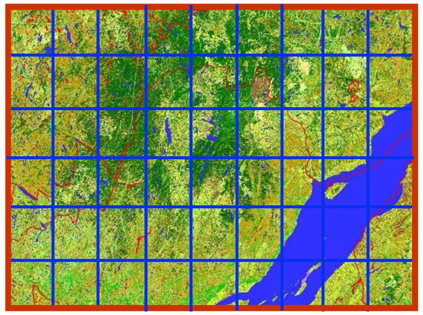


Lakes			
lakeId	code	area	raster
464	03	32.63	raster(…)
375	02	12.53	raster(…)
…	…	6.25	…

Five ways to use a WKT raster table…

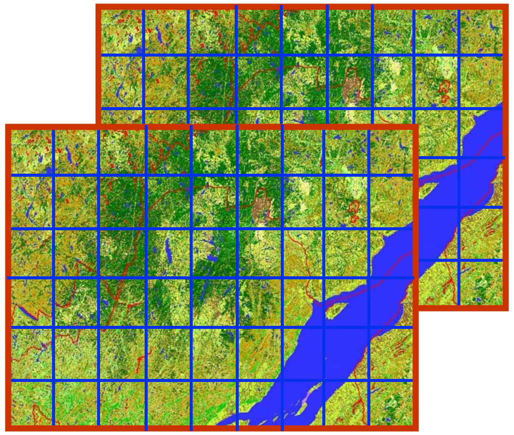
4 - As a "big raster"

- Stored as one table
- Necessarily rectangular; No missing tiles
- Sources is generally an images that is stored as series of small tiles
- 1 image = 1 table = many tile rows



landcoverImg	
tileId	raster
3	raster(…)
4	raster(…)
…	…

- However a coverage is generally composed of many images.
- In this case:
  - many images = many tables of many tile rows
  - images may overlap
- The classical way of using raster in a GIS. Not really useful from an analytical point of view since the same operations must be repeated on each tables



landcoverImg1	
tileId	raster
3	raster(…)
4	raster(…)
…	…

landcoverImg2	
tileId	raster
3	raster(…)
4	raster(…)
…	…

Five ways to use a WKT raster table…

5 - As an image warehouse

- Stored globally as one table
- many images = 1 table = many rows
- Images are not tiled and are not necessarily georeferenced
- Not necessarily intended for the geospatial industry
- Source is generally a folder of images

carPictures

Id	category	geometry
15436	Sport	raster(...)
35665	SUV	raster(...)
...	...	...

raster Importer

USAGE:

raster2pgsql [<options>] rasterfile [rasterfile…] [<schema>.]<table>

- **Create an SQL commands file to create a table of raster. If rasterfile is multiband and –b is not specified, every band are inserted. Multiple band can also be specified using multiple filenames (rasterfile1 is the first band, rasterfile2 the second, etc…). Can process multiple file from a folder.**
- **georeference (and pixel size) must exist directly in the files or in a companion World File.**

OPTIONS:

- **-s <srid> Set the SRID field. Default is -1.**
- **-b <nbband> Specify the number of band. The number of rasterfile must correspond to this number.**
- **-P <pixeltypes> Specify the pixels types in which to store each band. Ex. '8-bit unsigned integer,16-bit float'. conversion may happens.**
- **-n <nodata values> Specify the nodata value for each bands. Ex. '0,0.0'. Default to 'none' for each band.**
- **-t <pixels> Divide rasters into <pixels>x<pixels> tiles, one tile per row. Default is to store whole rasters as one row.**
- **(-d|a|b|c|p) Mutually exclusive options:**
  - d Drops the table, then recreates it and populates it with current raster file data.
  - a Appends raster file into current table, must be exactly the same pixel size, number of band, nodata value and pixel type.
  - B Appends raster files as a new bands. When tiled with the –t option, the new band is inserted tiled in the same way as the original band.
  - c Creates a new table and populates it, this is the default if you do not specify any options.
  - p Prepare mode, only creates the table.

- **-r <raster_column> Specify the name of the raster column (mostly useful in append mode).**
- **-D Use postgresql dump format (defaults to sql insert statements).**
- **-I Create a GiST index on the bbox of the raster column.**
- **-? Display this help screen**

Should rast2pgsql produce a SQL file like shp2pgsql or insert rasters directly in PostGIS?

Example 1 – Import/Export

Importing existing rasters as raster into PostGIS

```
>raster2pgsql -s 32198 -t 128 -i forestcover.tif temperature.tif
public.coverandtemp > c:/temp/coverandtemp.sql
```

File by file version where each file is spitted into tiles

or

```
>raster2pgsql -s 32198 -t 128,tid -i c:/forestcoverfolder/ c:/temperaturefolder/
public.coverandtemp > c:/temp/coverandtemp.sql
```

Folder version where each file in each folder is imported and tiled. tid is a target column storing a unique identifier for every source file (1,2,3,4,5,6,…) Could also come from part of the filename.

Exporting existing rasters as raster files

```
>pgsql2raster -f c:/temp/image#.tif -h localhost -p pwd -u user -r raster
public.coverandtemp
```

Produce many small files or tiles named image1.tif, image2.tif,…

or

```
>pgsql2raster -f c:/temp/image.tif -h localhost -p pwd -u user public ‘SELECT
RT_Accum(RT_Band(raster,1)) FROM coverandtemp WHERE prov=‘BC’ GROUP
BY prov’
```

Produce one big multiresolution raster by aggregation of many tiles.

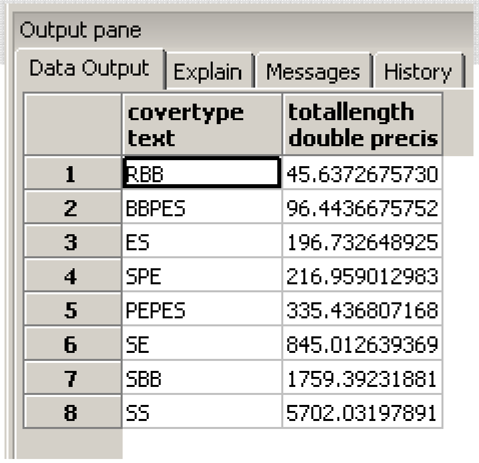
Example 2
Retrieving tiles intersecting an extent

```
SELECT RT_AsJPEG(RT_Band(raster,2),60)
FROM coverandtemp
WHERE RT_BBox(coverandtemp.raster) &&
ST_GeomFromText(‘POLYGON(-350926 351220,-350926
199833,-196958 199833,-196958 351220,-350926
351220)',32198) and
RT_Intersects(coverandtemp.raster,ST_GeomFromText('POLYG
ON(-350926 351220,-350926 199833,-196958 199833,-196958
351220,-350926 351220',32198))
```

Returns a table of jpeg tiles, from the temperature band, intersecting with the specified extent. The intersection takes into account the nodata values (they are not part of the geometry).

Example 3
What is the total length of roads (polylines) crossing different types of forest cover (raster) ?

```
SELECT max(covertype) as covertype,
sum(ST_Length(RT_Intersection(cover.raster,roads.geometry)))
as totallength
FROM cover, roads
WHERE cover.raster && roads.geometry and
RT_Intersects(cover.raster,roads.geometry)
GROUP BY covertype
ORDER BY totallength
```



	covertype text	totallength double precis
1	RBB	45.6372675730
2	BBPES	96.4436675752
3	ES	196.732648925
4	SPE	216.959012983
5	PEPES	335.436807168
6	SE	845.012639369
7	SBB	1759.39231881
8	SS	5702.03197891

Example of a totally seamless operation involving a raster layer and a polyline layer.

Example 4
Raster-Only MapAlgebra Operation
(possible also between a raster & a vector layer)

```
SELECT
RT_SelectByValue(
    RT_MapAlgebra(
        RT_Reclass(
            RT_Resample(
                RT_Transform(rast1,32198),
                30,'CUBIC'),
            '0-99=0,100-199=1,200-255=2'),
        rast2, 'int(0.434*A+0.743*B)'),
    2)
FROM cover1, cover2
WHERE RT_Transform(rast1,32198) ~= rast2
```

One of the coverage has to be reprojected, resampled and reclassed before doing a map algebra operation with the other coverage. There is as many rows in the result as there is tiles having equivalent extent in the two coverages. Only pixels with value '2' are retained in the final result. Coverages are assumed to have only one band.

Only raster having equivalent extent are part of the calculus

Example 5
Rebuilding a regional raster from a global coverage

```
SELECT
RT_AsJPEG(RT_Accum(A.raster), 60)
FROM
(SELECT RT_Band(raster, 2)) as raster
FROM USACoverage WHERE state=‘NY’) A
```

Use the same RT_Accum aggregate function as the one used with geometry.

PostGIS WKT raster VS Oracle GeoRaster*

Oracle GeoRaster*…

- is stored as a relation between two types in different tables:
  - images (SDO_GEORASTER) and
  - tiles (SDO_RASTER)
- is very complicated. Supports:
  - bitmap mask
  - two compression schemes
  - three interleaving types
  - multiple dimensions
  - embedded metadata (color table, statistics, etc…)
  - lots of unimplemented features
- do not allow seamless analysis operations with vector geometries

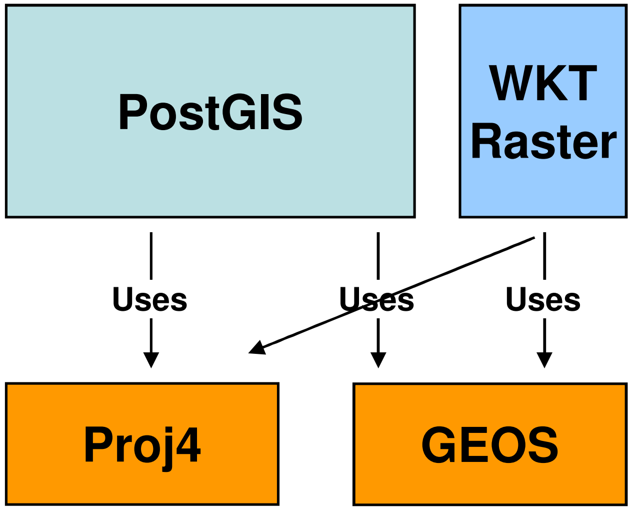
PostGIS WKT Raster…

- is stored as a single type in a table, much like the geometry type.
  - It does not distinguish the tile concept from the image concept. Both concepts are interchangeable.
- is more simple. Supports:
  - masks through band
  - only the deflate compression
  - only one interleaving type
  - only two dimensions
  - leave metadata, color table and statistics to the application level
- allows seamless analysis operations with vector geometries

*Xing Lin's PGRaster is almost identical to Oracle GeoRaster…

Implementation

WKT Raster is installed as a side application to PostGIS…



WKT Raster VS ISO 19123

- ISO 19123 is the "Abstract Specification Schema for Coverage Geometry and Functions"
- No "implementation" standard have been produced yet
- Even though the "raster" type is more easily associated with the notion of "coverage", a raster layer is NOT MORE a coverage than a vector layer. In the standard:
  - some types of coverage can be vectorial. e.g.
    - CV_DiscreteSurfaceCoverage (a vector layer of surfaces)
    - CV_DiscretePointCoverage (a vector layer of points)
  - some types of coverage can be matricial. e.g.
    - CV_DiscreteGridPointCoverage (a raster layer representing a grid of discrete points)
    - CV_ContinousQuadrilateralGridCoverage (a raster layer representing a continuous field)
- We think ISO 19123 should be implemented as a logical layer OVER a vectorial or a raster storage layer.
  - every ISO 19123 function should have the name of a vector or a raster table as argument. e.g. *evaluate(temp, point)* where temp is the name of a table containing a geometry or a raster column

Summary

- rasters are multiband and multiresolution, georeferenced, and support variable extents (per row), nodata values and multiple pixel types.
- raster is implemented as a new WKT/WKB form
  - WKT as RT_RasterFromText('RASTER(…)')
  - WKB as raw raster data, compressed with deflate
- Functions involving only rasters generally return rasters.
- Functions involving only geometries generally return geometries.
- Functions involving rasters and geometries have an option to specify the type of the output in case of ambiguity.
- Some raster-specific functions must be added but most functions become seamlessly usable with geometries or rasters.
- WKT Raster is much more simple to use than Oracle GeoRaster
- WKT raster is not an attempt to implement ISO 19123

Priorities and Planning

See the WKT Raster wiki page for planning and funding:

http://postgis.refractions.net/support/wiki/index.php?WKTRasterHomePage

Acknowledgements

- **Steve Cumming** (Steve.Cumming@sbf.ulaval.ca), Canada Research Chair in Boreal Ecosystems Modelling, for having initiated this project and financing it through a Canada Foundation for Innovation grant.

- **Thierry Badard** (http://geosoa.scg.ulaval.ca), Professor/full time researcher at Centre for Research in Geomatics, Université Laval, Quebec, Canada for his valuable comments, revisions, expertise and discussions.

Funding and Future Opportunities

- **Actual Funding -** The Boreal Avian Modeling (BAM) project and the Canadian Foundation for Innovation (CFI) are financing development of a web-based GIS tool to automate buffer operations on large spatial datasets. The objective is to support ecological analysis by reducing the overhead of GIS expertise and data assembly. A half-time position (Pierre Racine) is supported to develop a system prototype including raster integration in PostGIS.
- **Extended Funding -** Steve Cumming and Thierry Badard aim at initiating a new project to complement the funding of the project (and hence enable the financial support of another developer) and explore new avenues for geospatial data analysis provided by such a raster support (e.g. raster based Spatial OLAP applications).
- **Interested? -** If you are interested in such an implementation of the raster support in/with PostGIS and/or in participating to the new project, do not hesitate to contact Pierre Racine (Pierre.Racine@sbf.ulaval.ca), Steve Cumming (Steve.Cumming@sbf.ulaval.ca) and Thierry Badard (Thierry.Badard@scg.ulaval.ca).