

PostGIS 2.0 the new stuff



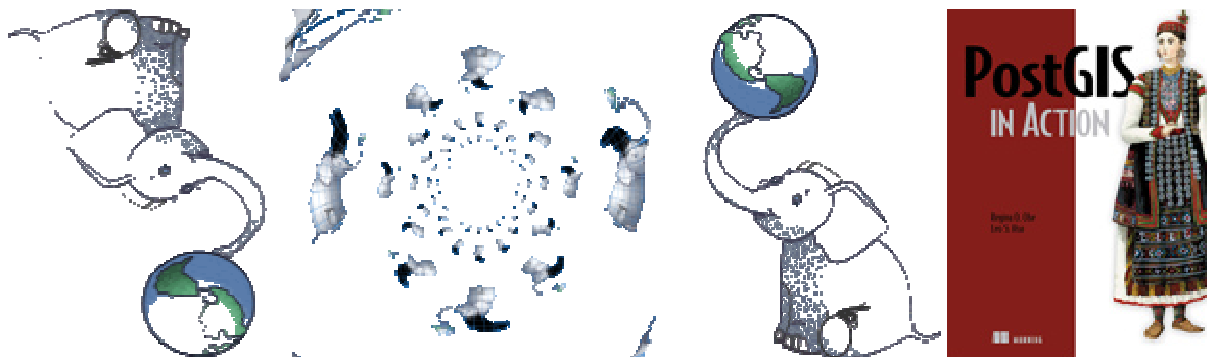
Regina Obe and Leo Hsu

<http://www.postgis.us>

<http://www.bostongis.com>

<http://www.postgresonline.com>

<http://www.paragoncorporation.com>



PostGIS Development Team

What have they been doing for 2.0? (Red are new to PSC)

Project Steering Committee:

Mark Cave-Ayland – bug fixing, spatial index, loader, cleanup

Chris Hodgson – bug fixing, build bot, site maintenance, incubation due diligence

Paul Ramsey (Chair) – bug fixing, nd-Index support, typmod, loader

Regina Obe – tiger geocoder, ST_AsX3D, ST_ConcaveHull, management functions, QA

Sandro Santilli - topology, all new geos functions, raster initial framework and low level api, overall PostGIS cleanup

Core Team working on PostGIS 2.0 – red are new to team in 2.0 Dev Cycle

Jorge Arévalo - Raster intersection, intersects, other stuff, GDAL PostGIS raster driver

Nicklas Avén - 3D spatial relationship and measurement functions

Olivier Courtin – 3D output, parsing, new 3D type serialization, lots of 3D related functions

Mark Leslie – shape file gui loader cleanup, ability to add multiple files

Mateusz Loskot – raster low level api functions, the raster loader

Bborie Park – Raster, all image output functions, many stat functions, many processing, too many to count

Pierre Racine – Raster architect, the man with the plan, the vision, and will to make it happen. He's all over the raster space.

Other contributors:

Jeff Adams – numerous fixes, enhancements to loader/dumper

Leo Hsu – helping with geocoder, testing, PostGIS windows packaging.

Bryce Nordgren – numerous patches to raster and API strategizing

Andrea Peri – QA topology and other parts of PostGIS, contributed several topology functions

Kashif Rasul – many documentation corrections, ST_GeomFromJSON (to be integrated)

David Zwarg - Raster miscellaneous functions

PostGIS 2.0 *ROCKS* your world!

http://www.postgis.org/documentation/manual-svn/PostGIS_Special_Functions_Index.html#NewFunctions_2_0
<http://www.postgis.org/download/postgis-2.0.0SVN.pdf>

- Easier management
- Better SQL/MM compliancy
- New geometry analysis functions
- Topology – SQL/MM
- Real 3D with surfaces and spatial relationships
- Seamless raster / geometry
- TIGER – Loader, Geocoder , Reverse Geocoder , Tiger 2 PostGIS Topology Loader

PostGIS 2.0 *ROCKS*

9.1 Specific features

CREATE EXTENSION

CREATE EXTENSION – the new way to install

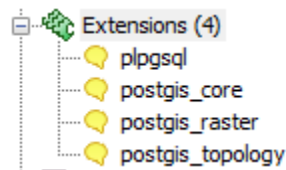
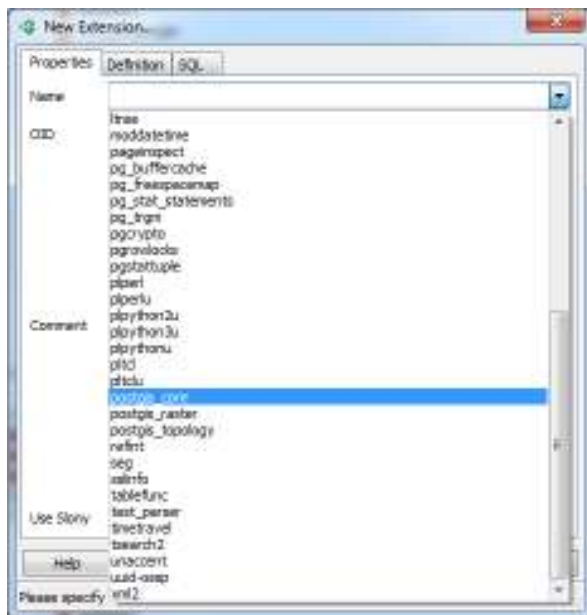
(still in experimental mode /extensions folder of trunk)

Also available in PostgreSQL - 9.1 PostGIS 2.0.0 Windows experimental builds

(The packaging of the components will probably change in final release time)

- http://www.postgis.org/download/windows/experimental.php#PostGIS_2_0_0

- CREATE EXTENSION **postgis_core;**
- CREATE EXTENSION **postgis_raster;**
- CREATE EXTENSION **postgis_topology;**



Extension	Owner	Comment
plpgsql	postgres	PL/pgSQL procedural language
postgis_core	postgres	postgis geometry and geography sql-mm spatial types and functions
postgis_raster	postgres	postgis raster spatial types and functions
postgis_topology	postgres	postgis topology spatial types and functions

PostGIS 2.0 *ROCKS*
9.1 Specific features
KNN GIST

Planned before release not yet committed.

PostGIS 2.0 *ROCKS* Easier Management

In the beginning (pre 2.0)

```
CREATE TABLE .. Without the  
geometry column
```

```
AddGeometryColumn(..what were  
those args?)
```

```
DropGeometryTable(..)
```

and if you screwed up:

```
SELECT populate_geometry_columns(...);
```

PostGIS 2.0 *ROCKS* Easier Management

- In PostGIS 2.0 geometry_columns is no longer a table, but a view that reads from system catalogs.
- geometry columns can now be created using type modifiers (typmod) similar to what we have with geography columns
- The big deal is that geometry_columns can no longer be updated directly and is always in synch with the table definitions.

PostGIS 2.0 *ROCKS* Easier Management

We have this for geography introduced in 1.5, now geometry gets a face lift too.

```
CREATE TABLE buildings(gid SERIAL PRIMARY KEY, geom  
  geometry(MultiPolygon, 26986) );
```

```
CREATE TABLE testpolyhed(gid SERIAL PRIMARY KEY, geom  
  geometry(PolyhedralSurfaceZ, 26986) );
```

```
CREATE TABLE mixed3d(gid SERIAL PRIMARY KEY, geom  
  geometry(GeometryZ, 26986) );
```

```
CREATE TABLE mixed2d(gid SERIAL PRIMARY KEY, geom  
  geometry(Geometry, 26986) );
```


PostGIS 2.0 *ROCKS* Easier Management

In PostGIS 2.0 you can use the old way or the one step / ANSI SQL way:

```
CREATE TABLE myg(gid serial  
primary key, geom geometry(Point,  
26986);
```

```
ALTER TABLE myg ADD COLUMN  
geom_4326 geometry(Point, 4326);
```

```
DROP TABLE myg;
```

PostGIS 2.0 *ROCKS* Easier Management

In PostGIS 2.0 old way has new tricks. It can be constraint based or typmod based. Defaults to typmod if not specified

To get old constraint way: use new `use_typmod` and set it to false.

```
AddGeometryColumn(schema_name,  
table_name, column_name, integer  
srid, geomtype, dimension,  
use_typmod);
```

PostGIS 2.0 *ROCKS* Easier Management

To get old constraint way: use new
use_typmod and set it to false.

```
SELECT AddGeometryColumn('public',  
'myg', 'mygeom', 2249, 'POINT', 2,  
false);
```

PostGIS geometry_columns is a view means easier maintenance

Reads constraints from system catalogs if constraint based and type from system catalogs if typmod based.

```
SELECT f_table_schema, f_table_name, f_geometry_column,  
       coord_dimension, srid, type  
FROM geometry_columns  
WHERE f_table_name IN('buildings', 'testpolyhed');
```

f_table_schema	f_table_name	f_geometry_column	coord_dimension	srid	type
public	buildings	geom	2	26986	MULTIPOLYGON
public	testpolyhed	geom	3	26986	POLYHEDRALSURFACE

PostGIS 2.0 *ROCKS*

What about views?

If built with typmod and no qualifier registers correctly:

```
CREATE VIEW v_myg AS  
SELECT * FROM myg;
```

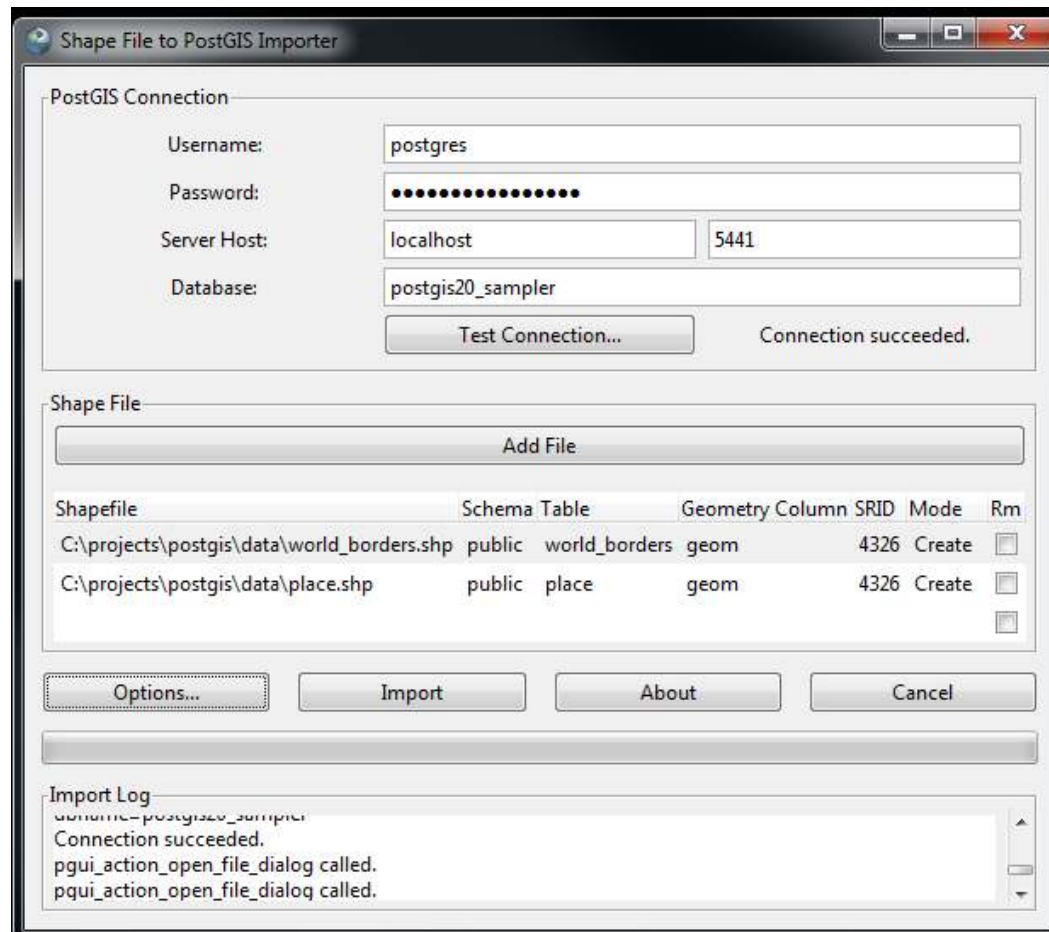
constraint-based or geometry function outputs you need to cast for it to register correctly in **geometry_columns**:

```
CREATE VIEW v_myg AS  
SELECT *,  
ST_Transform(geom, 2249) :: geometry(Point,  
2249) As geom_2249;
```

PostGIS 2.0 *ROCKS*

Easier Management

Loader can load multiple files



PostGIS 2.0 *ROCKS*

FGDB2PostGIS

PostGIS2FGDB

What is this?

You didn't see this slide

Paul is hiding in the garage working on command line loader / dumper for ESRI's new fangled FGDB database format

Requires newest GDAL lib and ESRI FGDB API

<http://trac.osgeo.org/postgis/browser/spike/pramsey/postgis2fgdb>

PostGIS 2.0 *ROCKS* SQL/MM compliance

ZM geometries now output more compliant but ST_GeomFromText will accept old and new formats. We have new SQL/MM types: PolyHedralSurface and TIN

```
POINTZM(1 2 3 4)
```

```
POINTZ (1 2 3)
```

```
TRIANGLE((0 2, 10 4, 12 0, 0 2))
```

```
TRIANGLEZ((0 2 1, 10 4 1, 12 0 1, 0 2 1))
```

```
TIN(((0 2, 10 4, 12 0, 0 2)),((0 2, -2 -6, 12 0, 0 2)),  
((0 2, 10 4, 5 8, 0 2)))
```

```
POLYHEDRALSURFACEZ( ((0 0 0, 0 0 1, 0 1 1, 0 1 0, 0 0 0)),  
((0 0 0, 0 1 0, 1 1 0, 1 0 0, 0 0 0)),  
((0 0 0, 1 0 0, 1 0 1, 0 0 1, 0 0 0)),  
((1 1 0, 1 1 1, 1 0 1, 1 0 0, 1 1 0)),  
((0 1 0, 0 1 1, 1 1 1, 1 1 0, 0 1 0)),  
((0 0 1, 1 0 1, 1 1 1, 0 1 1, 0 0 1)))
```


PostGIS 2.0 *ROCKS*

More Functions for Geometry

We'll demonstrate:

`ST_FlipCoordinates`

`ST_ConcaveHull`

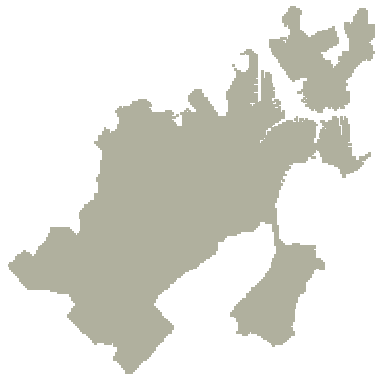
`ST_Snap`

`ST_Split`

PostGIS 2.0 *ROCKS*

ST_FlipCoordinates

Your world is on its side, no problem, just flip it.



```
SELECT ST_ASText(  
    ST_FlipCoordinates(geom)  
) as geom_flipped  
FROM ST_GeomFromText('LINESTRING(762091  
2920414,762588 2920692,762676 2920779) ',26986)  
As geom;  
  
--- geom_flipped ---  
LINESTRING(2920414 762091,2920692 762588,2920779  
762676)
```

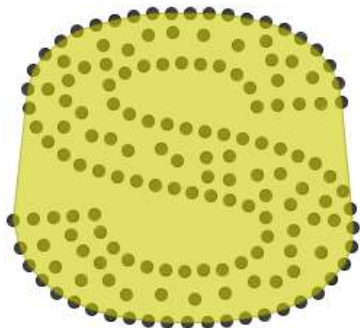
PostGIS 2.0 *ROCKS*

Vacuum Seal: ST_ConcaveHull

Approximation of geometry encasing a set of geometries. Area is always smaller or equal to area of ConvexHull and larger than area of an areal geometry

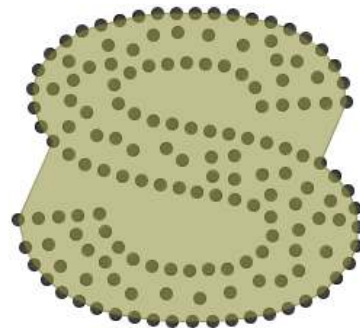
`ST_ConvexHull(geom)`

Same as 100%
Concave Hull



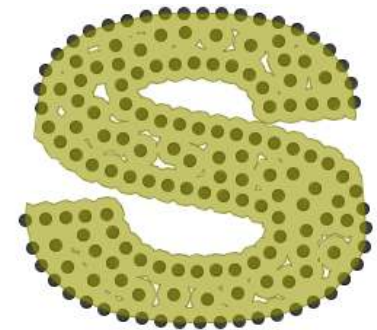
99% area target convex

`ST_ConcaveHull(geom, 0.99)`



90% target allow holes

`ST_ConcaveHull(geom, 0.90, true)`



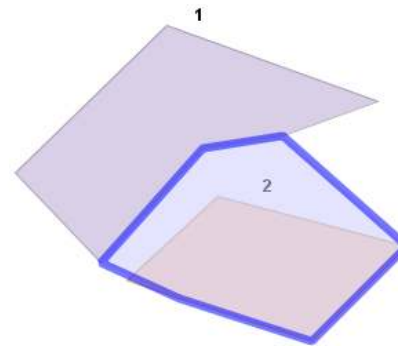
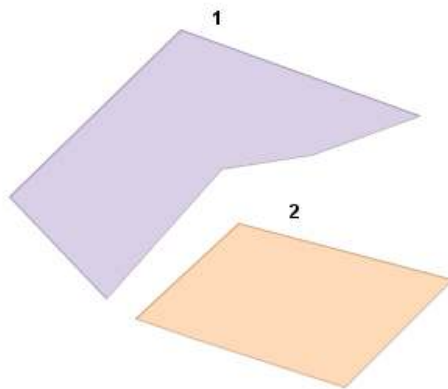
PostGIS 2.0 *ROCKS*

ST_Snap

Geometries don't quite line up, no problem, just snap them. Below returns geometry 2 snapped to tolerance of distance + 2 units.

Dark border represents new location of 2 after snapping.

```
SELECT ST_Snap(p2.geom,p1.geom, ST_Distance(p1.geom, p2.geom) + 2) FROM polys
AS p1 CROSS JOIN polys As p2
WHERE p1.id =1 and p2.id =2;
```



PostGIS 2.0 *ROCKS*

ST_Split

Split a geometry with a line string. This one splits polygon into 3 pieces

```
SELECT (result).path[1], (result).geom
FROM (
  SELECT ST_Dump(ST_Split(poly,line)) As result
  FROM
    ST_Buffer(
      ST_GeomFromText('LINESTRING(10 20, 30 40)'),
      2, 'endcap=flat') As poly
  CROSS JOIN
    ST_GeomFromText('LINESTRING(10 20, 30 40, 20 10)') As line
) As foo;
```



PostGIS 2.0 *ROCKS* and more ...

ST_MakeValid

ST_AsLatLonText

ST_IsValidDetail

ST_OffsetCurve

ST_RemoveRepeatedPoints

ST_SharedPaths

ST_AsRaster – converts a postgis geometry to a postgis raster. Part of postgis raster module – more on that later.

ST_GeomFromJSON – create geometry from JSON input. Work done by Kashif Rasul (not yet committed), but should make it into PostGIS 2.0 if committed soon. Refer to ticket: <https://trac.osgeo.org/postgis/ticket/376> for status details

PostGIS 2.0 *ROCKS*

Better 3D Support

New Types: TRIANGLE, TIN,
POLYHEDRALSURFACE

New Functions: Many existing extended to support new types. New functions specific for 3D.

Spatial Index: nd-Index

PostGIS 2.0 *ROCKS*

New 3D Relationship Functions

Only &&& has support for TINs

- `ST_3DDistance / ST_3DMaxDistance`
- `ST_3DIntersects`
- `ST_3DClosestPoint`
- `ST_3DDWithin`
- `ST_3DShortestLine/LongestLine`
- `&&&` (3D overlaps bounding box operator)

PostGIS 2.0 *ROCKS*

New 3D Input/Output Functions

** extended to support 3D*

- ST_AsGML*
- ST_AsX3D
- ST_GeomFromGML*
- ST_GeomFromEWKT*
- ST_AsText*
- ST_AsEWKT*

PostGIS 2.0 *ROCKS*

Create Table for 3D geometries

```
CREATE TABLE test3d(gid SERIAL  
PRIMARY KEY,  
geom geometry(GeometryZ, 0));
```

```
CREATE INDEX idx_test3d_geom_gist3d  
ON test3d USING gist(geom  
gist_geometry_ops_nd);
```

PostGIS 2.0 *ROCKS*

Triangular Irregular Network (TIN)

2D, 3D, 3DM

```
INSERT INTO test3d(geom)
VALUES ('TINZ(((1 2 3,4 5 6,7 8 9,1 2 3)),
          ((10 11 12,13 14 15,16 17 18,10 11 12)),
          ((19 20 21,22 23 24,25 26 27,19 20 21)))'::geometry);
```

```
INSERT INTO test3d(geom)
VALUES ('TRIANGLEZ((0 0 -1,
                  -5 -5 -5,-4 -4 -4,0 0 -1))'::geometry);
```

PostGIS 2.0 *ROCKS*

Polyhedral Surface

```
INSERT INTO test3d(geom)
VALUES ('POLYHEDRALSURFACEZ (
((0 0 0,0 0 5,0 15 5,0 15 0, 0 0 0)),
((0 0 0,0 15 0,10 15 0,10 0 0, 0 0 0)),
((0 0 0,10 0 0,10 0 5,0 0 5, 0 0 0)),
((10 0 0,10 15 0,10 15 5,10 0 5, 10 0 0)),
((0 15 0,0 15 5,10 15 5,10 15 0,
0 15 0)))'::geometry
);
```

PostGIS 2.0 *ROCKS*

&& vs. &&&

```
SELECT ST_ASEWKT(a.geom) As awkt,  
       ST_ASEWKT(b.geom) As bwkt,  
       a.geom && b.geom As int2d,  
       a.geom &&& b.geom As int3d  
FROM test3d As a CROSS JOIN test3d b  
WHERE a.geom && b.geom AND NOT(a.geom &&& b.geom);
```

awkt	bwkt	int2d	int3d
TRIANGLE((0 0 ...	POLYHEDRALSURFACE(((0 0...	t	f
POLYHEDRALSURFACE(((0 0 0...	TRIANGLE((0 0 -1...	t	f

PostGIS 2.0 *ROCKS*

ST_AsGML (version 3 only)

```
SELECT gid, ST_AsGML(3, geom) As ogml FROM test3d;
```

```
-- result -
```

```
1 |<gml:PolyhedralSurface><gml:PolygonPatches>  
  <gml:PolygonPatch>...</gml:PolyhedralSurface>  
2 |<gml:Tin><gml:trianglePatches>  
  <gml:Triangle>...</gml:Triangle>...  
  </gml:trianglePatches></gml:Tin>
```

PostGIS 2.0 *ROCKS*

ST_AsX3D

```
SELECT gid, ST_AsX3D(geom) As ox3d FROM test3d;
```

```
-- result -
```

```
1 | <IndexedFaceSet  coordIndex='0 1 2 3 -1 4 5 6 7 -1 8 9  
10 11 -1 12 13 14 15 -1 16 17 18 19'><Coordinate point='0  
0 0 0 0 5 0 15 5 0 15 0 0 0 0 0 15 0 10 15 0 10 0 0 0 0 0  
10 0 0 10 0 5 0 0 5 10 0 0 10 15 0 10 15 5 10 0 5 0 15 0  
0 15 5 10 15 5 10 15 0' /></IndexedFaceSet>  
  
2 | <IndexedTriangleSet  index='0 1 2 3 4 5 6 7  
8'><Coordinate point='1 2 3 4 5 6 7 8 9 10 11 12 13 14 15  
16 17 18 19 20 21 22 23 24 25 26  
27' /></IndexedTriangleSet>
```

PostGIS 2.0 *ROCKS*

3D Viewers

Still growing space – No direct support for PostGIS new types yet

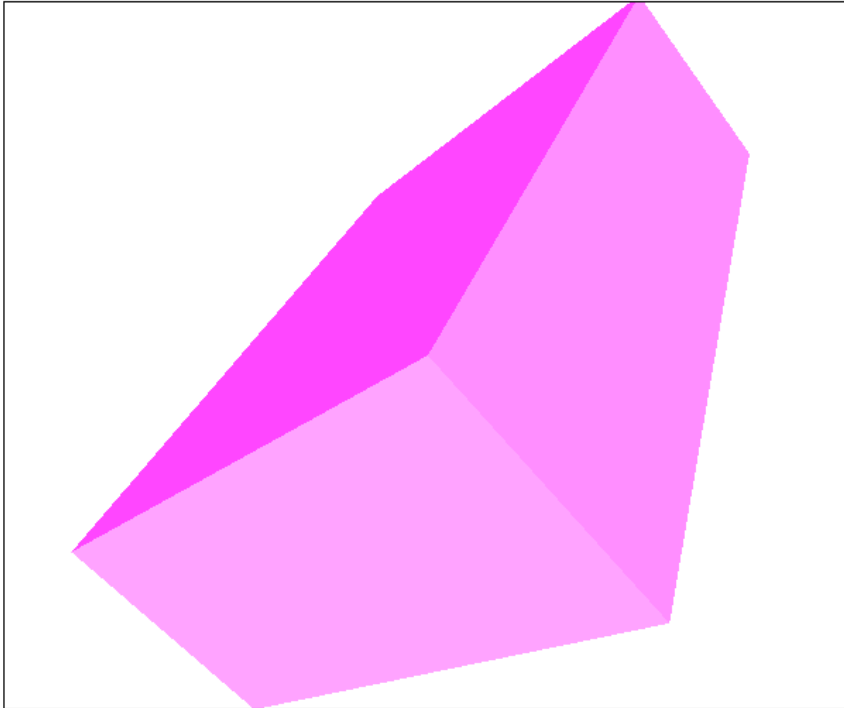
GvSig – preliminary 3D support in gvSig 1.11, but not for new types

Using ST_AsX3D can utilize X3D viewers for new types like TIN and Polyhedral Surfaces – Viewers supporting 3D

- FreeWrl -- <http://freewrl.sourceforge.net/> (supports Linux, Mac, Windows) and Iphone in development
- InstantReality -- <http://www.instantreality.org/examples/> (Windows, Linux, Mac)
- X3Dom – <http://www.x3dom.org/> X3D embedded on web pages piggy backs on WebGL and HTML5 support built-into FireFox 4+ and Google Chrome so no plugins needed for these. For IE uses InstantReality or Flash.
<http://www.instantreality.org/>
- Vivaty Studio (originally FluxPlayer) – (now owned by Microsoft) – questionable what Microsoft plans for it.
- Many others: http://www.web3d.org/x3d/vrml/tools/viewers_and_browsers/

PostGIS 2.0 *ROCKS*

Polyhedral in x3dom html



```
POLYHEDRALSURFACEZ (  
  ((0 0 0,0 0 5,0 15 5,0 15 0, 0 0  
  0)),  
  ((0 0 0,0 15 0,10 15 0,10 0 0, 0 0  
  0)),  
  ((0 0 0,10 0 0,10 0 5,0 0 5, 0 0  
  0)),  
  ((10 0 0,10 15 0,10 15 5,10 0 5, 10  
  0 0)),  
  ((0 15 0,0 15 5,10 15 5,10 15 0,  
  0 15 0)))
```

```
<IndexedFaceSet  
  coordIndex='0 1 2 3 -1 4  
  5 6 7 -1 8 9  
  10 11 -1 12 13 14 15  
  -1 16 17 18 19'>  
<Coordinate point='0 0 0 0 0 5 0 15 5 0  
  15 0 0 0  
  0 0 15 0 10 15 0 10 0 0 0 0 0 10 0  
  0 10 0 5 0 0 5 10 0 0 10 15 0 10 15 5  
  10 0 5 0 15 0  
  0 15 5 10 15 5 10 15 0'  
></IndexedFaceSet>
```

PostGIS 2.0 *ROCKS*

Tiger Geocoder

- Integrated in Documentation
- Upgraded to work with Census Tiger 2010
- Cross Platform Loader script generator to load state by state data
- Reverse Geocoder
- Geocoder revised to allow designating max matches to return
- Geocoder revised to offset to correct side of street
- Index helper functions to generate missing indexes for tables
- Many improvements in speed and accuracy
- TIGER to PostGIS topology loader

PostGIS 2.0 *ROCKS*

Tiger Geocoder

Reverse Geocode

```
SELECT pprint_addy(r.addy[1]) As prim_addr,  
array_to_string(street, ',') As cross_streets  
FROM reverse_geocode(  
    ST_SetSRID(  
        ST_Point(-71.357682,42.455041)  
        ,4326) ) As r;
```

prim_addr

cross_streets

15 Grant St, Concord, MA 01742

Sudbury Rd

PostGIS 2.0 *ROCKS*

Tiger Geocoder

Geocode

```
SELECT pprint_addy((g).addy) As addr
      ,ST_X(ST_SnapToGrid((g).geomout,0.00001)) As lon
      ,ST_Y(ST_SnapToGrid((g).geomout,0.00001)) As lat
      ,(g).rating
FROM geocode('15 Grant Street, Concord, MA 01742',1)
As g;
```

addr

lon

lat

rating

15 Grant St, West Concord, MA 01742

-71.35769

42.45506

0

PostGIS 2.0 *ROCKS*

TIGER meets

PostGIS Topology

```
-- Create Boston Topology in MA State Plane Feet with 0.25 ft tolerance
SELECT topology.CreateTopology('topo_boston',
2249,0.25);
```

```
-- Load in Tiger edges/faces/nodes bounded by Place: Boston
SELECT tiger.topology_load_tiger(
'topo_boston', 'place', '2507000');
```

```
-- Let's see what we have --
```

```
SELECT topology.TopologySummary('topo_boston');
```

```
topologysummary
```

```
-----
```

```
Topology topo_boston (15), SRID 2249, precision 0.25
20576 nodes, 31597 edges, 11109 faces, 0 topogeoms in 0
layers
```

PostGIS 2.0 *ROCKS* Topology / SQL/MM

What is the difference between the topological model and geometry model?

Geometries are denormalized representations of space where objects that share edges, nodes, or faces each have a redundant copy of shared element.

Topology is a normalized representation of space – shared edges, nodes, faces are not redundant .

<http://www.postgis.org/documentation/manual-svn/Topology.html> for details

PostGIS 2.0 *ROCKS* Topology / SQL/MM

Benefits of Topological Based systems

Consistent editing / updating – update one edge and all geometries sharing that edge change also. Simplification of a topology would not result in edges that used to be shared no longer being shared.

Reduced Storage

Explicit Spatial Relationships – gets around some tolerance issues.

PostGIS 2.0 *ROCKS* Topology / SQL/MM

Example Popular Topological based systems:

US Census Topologically Integrated Geographic Encoding and Referencing (TIGER):

consists of faces, edges, nodes, (something else – lets call them features)

OpenStreetMap: consists of nodes, ways, relations, and tags

PostGIS 2.0 *ROCKS*

Topology / SQL/MM

Topology schema: contains all topology functions. Many functions are SQL/MM and prefixed with ST. Non-SQL/MM are not prefixed.

For each topology created, a new schema with same name is created that contains:

edge – linestrings connect at nodes

face – just holds mbr, but defines polygons. Edges separate faces.

node – The joints of topology – edges begin / end at nodes

relation – defines relationship between topogeometry and other elements in topology.

New data type – **TopoGeometry** a normalized geometry aware of its shared neighbors within a topology.

PostGIS 2.0 *ROCKS* Topology / SQL/MM

Topology Metadata tables

topology.topology – registry of all topologies in database

topology.layer - listing of feature tables and which topology defines the elements of the topogeoms in the table.

PostGIS 2.0 *ROCKS*

Topology

Over 50 functions in PostGIS topology

Management:

CreateTopology

DropTopology

ValidateTopology

TopologySummary

AddTopoGeometryColumn etc.

Processing:

Many SQL/MM compliant, several non-SQL/MM but deemed needed and several more planned

Output:

AsGML

geometry

Planned: ToTopoGeom – will convert a PostGIS geometry to a topogeometry

PostGIS 2.0 *ROCKS*

ValidateTopology

Highlights problems

```
SELECT * FROM topology.ValidateTopology('topo_boston');
```

```
-- got no errors -
```

For suffolk:

```
SELECT * FROM topology.ValidateTopology('topo_suffolk');
```

```
error          |   id1   |   id2
-----+-----+-----
coincident nodes | 81045651 | 81064553
edge crosses node | 81045651 | 85737793
edge crosses node | 81045651 | 85742215
edge crosses node | 81045651 | 620628939
edge crosses node | 81064553 | 85697815
edge crosses node | 81064553 | 85728168
edge crosses node | 81064553 | 85733413
```

PostGIS 2.0 *ROCKS*

AddTopoGeometryColumn

LINEAL Layer

```
CREATE SCHEMA boston;
```

```
CREATE TABLE boston.roads (  
  tlid bigint PRIMARY KEY,  
  fullname varchar(100),  
  mtfcc varchar(5));
```

```
SELECT topology.AddTopoGeometryColumn (  
  'topo_boston',  
  'boston', 'roads', 'topo', 'LINE');
```

PostGIS 2.0 *ROCKS*

AddTopoGeometryColumn

Resulting table structure looks like this:

```
CREATE TABLE boston.roads (  
  tlid bigint NOT NULL  
  , fullname character varying(100)  
  , mtfcc character varying(5)  
  , topo topology.topogeometry  
  , CONSTRAINT roads_pkey PRIMARY KEY (tlid )  
  , CONSTRAINT check_topogeom CHECK  
  ((topo).topology_id = 15  
  AND (topo).layer_id = 5 AND (topo).type = 2));
```

PostGIS 2.0 *ROCKS*

CreateTopoGeom LINEAL Features

```
CreateTopoGeom(topology, topotype, layer_id, array{id, elementtype})
```

```
CreateTopoGeom('topo_boston', 2, 5, array{edge_id, 2} )
```

```
topotype: (multi)point=1, (multi)linestring=2, (multi)polygon=3, collection=4
```

```
elementtype: node=1, edge=2, face=3
```

```
INSERT INTO boston.roads (tlid
, fullname, mtfcc, topo)
SELECT edge_id, fullname, mtfcc,
topology.CreateTopoGeom('topo_boston'
, 2, 5
, ('{' || edge_id::text ||
', 2}'))::topology.topoelementarray)

FROM
(SELECT DISTINCT e.edge_id
, fullname
, mtfcc
FROM tiger_data.ma_edges AS m
INNER JOIN topo_boston.edge As e
ON m.tlid = e.edge_id
WHERE m.mtfcc LIKE 'S%') As te;
```

PostGIS 2.0 *ROCKS*

Output: As geometry

- The topogeom can be cast to geometry type
- so can use most geometry functions

```
SELECT fullname, topo,  
       ST_AsText(topo::geometry) As atext  
FROM boston.roads  
WHERE fullname > ' ' LIMIT 1;
```

fullname	topo	atext
Rice St	(15,5,1,2)	LINESTRING(780534.5 2929864.25, 780560 2929839.5, 780581 2929798.75)

PostGIS 2.0 *ROCKS*

AddTopoGeometryColumn

AREAL Layer

I loaded a table called boston.nei with a geom column that holds one organization's vision of where the boundaries of Boston neighborhoods are which in theory were drawn along street segments

```
SELECT topology.AddTopoGeometryColumn (  
    'topo_boston',  
    'boston', 'nei', 'topo', 'POLYGON');
```

Wouldn't it be fun to take this and create topo geoms of it using Census idea of where the street centerlines are?

PostGIS 2.0 *ROCKS*

TopoElementArray_Agg

AREAL Features

Can be formed by aggregating faces

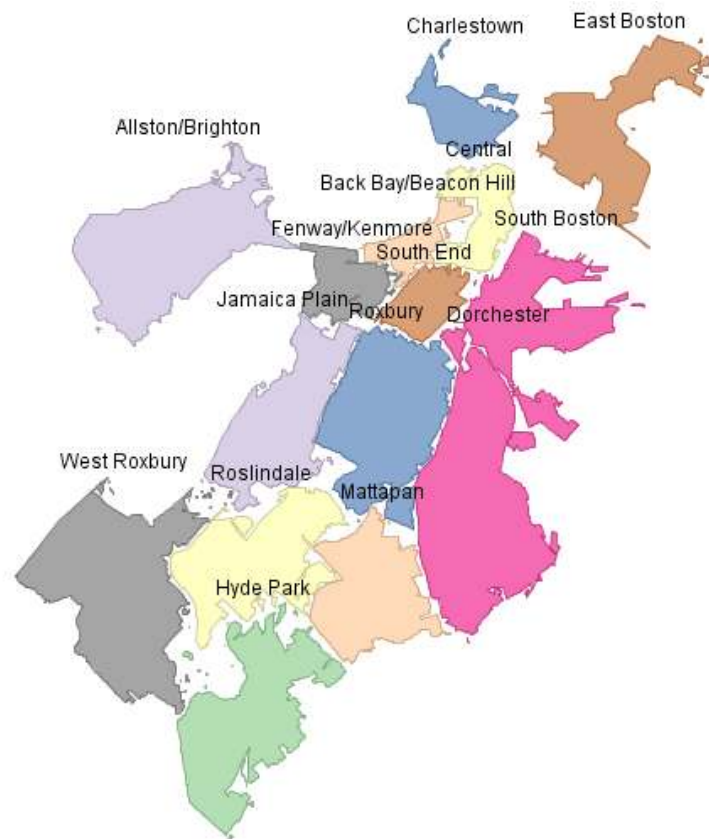
```
UPDATE boston.nei
  SET topo =
    topology.CreateTopoGeom('topo_boston'
      , 3, 6
      , foo.bedges)
FROM (SELECT n.gid,
  topology.TopoElementArray_Agg(ARRAY[f.face_id, 3])
  As bedges
FROM boston.nei As n
      INNER JOIN topo_boston.face As f ON
n.geom && f.mbr
      WHERE ST_Covers(n.geom,
topology.ST_GetFaceGeometry('topo_boston',
f.face_id))
      GROUP BY n.gid) As foo
WHERE foo.gid = boston.nei.gid;
```

PostGIS 2.0 *ROCKS*

But data sets are from different organizations
Boundaries approximate simplifications



geom: Boundaries drawn and lines simplified to be lighter and more appealing they really don't follow street centerlines



topo: Formed from faces of Tiger data which are formed from edges like street center lines. We lost all faces not completely covered by a neighborhood

PostGIS 2.0 *ROCKS*

CreateTopoGeom AREAL Features

Compensate for imperfect data

```
UPDATE boston.nei
  SET topo =
    topology.CreateTopoGeom('topo_boston'
      , 3, 6
      , foo.bedges)

FROM (SELECT n.gid,
  topology.TopoElementArray_Agg(ARRAY[f.face_id, 3]) As bedges
  FROM boston.nei As n
    INNER JOIN topo_boston.face As f ON n.geom && f.mbr
  WHERE
    ST_Covers(n.geom, topology.ST_GetFaceGeometry('topo_boston',
f.face_id))
    OR
    ( ST_Intersects(n.geom,
topology.ST_GetFaceGeometry('topo_boston', f.face_id))
    AND ST_Area(ST_Intersection(n.geom,
topology.ST_GetFaceGeometry('topo_boston', f.face_id) ) ) >
    ST_Area(topology.ST_GetFaceGeometry('topo_boston',
f.face_id))*0.5)
    GROUP BY n.gid) As foo
WHERE foo.gid = boston.nei.gid;
```

PostGIS 2.0 *ROCKS*

If more than 50% of a face falls in a neighborhood boundary it is part of that neighborhood



geom: Boundaries drawn



topo: Formed from faces of Tiger data
which are formed from edges
like street center lines

PostGIS 2.0 *ROCKS*

Topology summary after layers

```
SELECT topology.TopologySummary('topo_boston');

topologysummary
-----
Topology topo_boston (15), SRID 2249, precision 0.25
20576 nodes, 31597 edges, 11109 faces, 25090 topogeoms in 2
  layers
Layer 5, type Lineal (2), 25075 topogeoms
  Deploy: boston.roads.topo
Layer 6, type Polygonal (3), 15 topogeoms
  Deploy: boston.nei.topo
```

PostGIS 2.0 *ROCKS*

Raster

Over 70 functions and growing

For more information:

http://www.postgis.org/documentation/manual-svn/RT_reference.html

http://www.postgis.org/documentation/manual-svn/PostGIS_Special_Functions_Index.html#PostGIS_RasterFunctions

Key Features

- **Loader** – (*python with numpy and gdal required*)
can load individual and chunk them or load folders of raster files
- **Intersections / Intersect with geometry** – returns geometry
(raster equivalents not yet available but ST_Intersection(raster/raster)
planned for before PostGIS 2.0 release)
- **Extract individual pixel values**
- **Output functions** - Output as any GDAL supported raster
or a postgis geometry
- **Constructor Functions** – make rasters from scratch, existing, or geometry
- **Stats** – Statistics about a raster coverage or tile
- **Processing** – morphs to another raster or postgis geometry

PostGIS 2.0 *ROCKS*

Load Raster

This generates an sql file that will load all the jpegs in current folder into a new table called aerials.boston (Massachusetts State Plane Meters (26986)), with each raster record 100x100 pixels width / height. The `-F` will create a column called filename in the table which will list

The jpeg file each raster record tile came from.

The `-I` will create a gist index on convex hull of the raster.

```
python raster2pgsql.py -r bos_tiles\*.jpg \  
    -t aerials.boston -s 26986 -k 100x100 \  
    -F -I -o aerials.sql
```

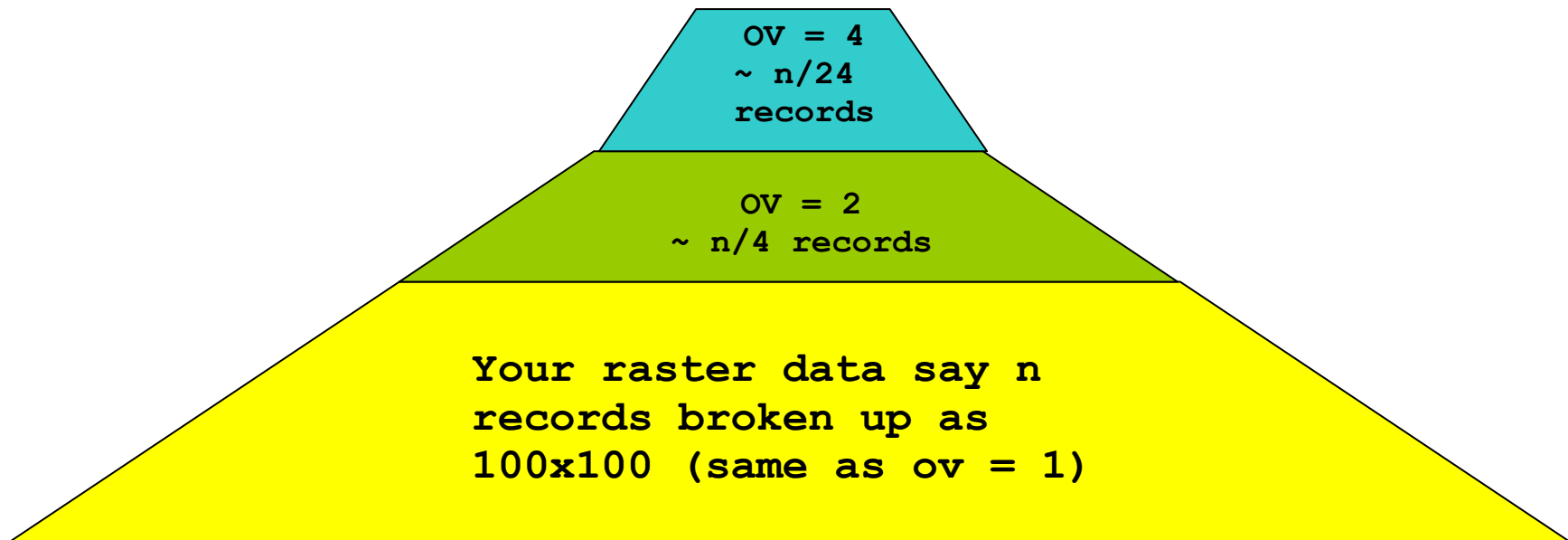
This runs the script loading the data into mygisdb

```
psql -d mygisdb -f aerials.sql
```


PostGIS 2.0 *ROCKS*

Raster Overviews (aka Pyramid)

These are lower resolution raster tables of your primary tables. These are registered in a table called: `raster_overviews` and created using the loader with `-l` level switch
It works kind of like this: (assuming all you set your overviews as same block size as your regular)



PostGIS 2.0 *ROCKS*

Regular compare Overviews

Overviews are good for zoom out and also doing faster but less high res calculations:

For our small sample:

```
--result: 845 records  
SELECT COUNT(*) FROM aerials.o_4_boston;  
--result: 3,125 records  
SELECT COUNT(*) FROM aerials.o_2_boston;  
--result: 20,000 records  
SELECT COUNT(*) FROM aerials.boston;
```

PostGIS 2.0 *ROCKS*

Load Data Overview (Pyramid)

This generates an sql file that will load all the jpegs in current folder into a new table called aerials.o2_boston (Massachusetts State Plane Meters (26986)) for our table aerials.boston, with each raster record 100x100 pixels width / height but lower res.

The -F will create a column called filename in the table which will list

The jpeg file each raster record tile came from.

The -l will create an overview table for aerials.boston with ov level (in this case 4)

Note: The table will be called aerials.o_4_boston (not aerials.boston), but will be Registered in raster_overviews table and associated with aerials.boston

```
python raster2pgsql.py -r *.jpg \  
  -t aerials.boston -s 26986 -l 4 -k 100x100 \  
  -F -I -o aerials_overview4.sql
```

This runs the script loading the data into mygisdb

```
psql -d mygisdb -f aerials_overview4.sql
```

PostGIS 2.0 *ROCKS*

Raster

Seamless Geometry / Raster

`ST_Intersects(geometry, raster)`

`ST_Intersection(geometry, raster)`

PostGIS 2.0 *ROCKS*

Raster ST_Intersects

How many parcels intersect our loaded raster tiles

```
SELECT COUNT(DISTINCT p.map_id)
FROM massgis.parcels_boston As p
     INNER JOIN aeriāls.boston As r
     ON ST_Intersects(p.geom, r.rast);
```

PostGIS 2.0 *ROCKS*

Stats: ST_ValueCount

**Give pixel value distribution of band 1 of
all tiles in samples.downtown_chunked**

```
SELECT (pvc).value As val
        , SUM((pvc).count) As pixcount
FROM
( SELECT ST_ValueCount(rast,1) AS pvc
  FROM samples.downtown_chunked As p
) As rpvc
WHERE (pvc).value BETWEEN 10 AND 254
GROUP BY (pvc).value
HAVING SUM((pvc).count) > 8880
ORDER BY (pvc).value;
```

```
val pixcount
```

```
-----
```

```
32      9652
33     12791
34     15216
35     18146
36     21466
37     25033
:
```

PostGIS 2.0 *ROCKS*

Raster

Output Functions

```
ST_AsPNG (raster ....)  
ST_AsTIFF (raster....)  
ST_AsJPEG (raster....)  
ST_AsGDALRaster (raster...)  
ST_Polygon (raster, band_num)
```

Constructor Functions

```
ST_MakeEmptyRaster  
ST_AsRaster (geometry)  
ST_Band (raster ..)
```

PostGIS 2.0 *ROCKS*

Output: ST_AsPNG, ST_AsJPEG

With these you can make a quickie query viewer.

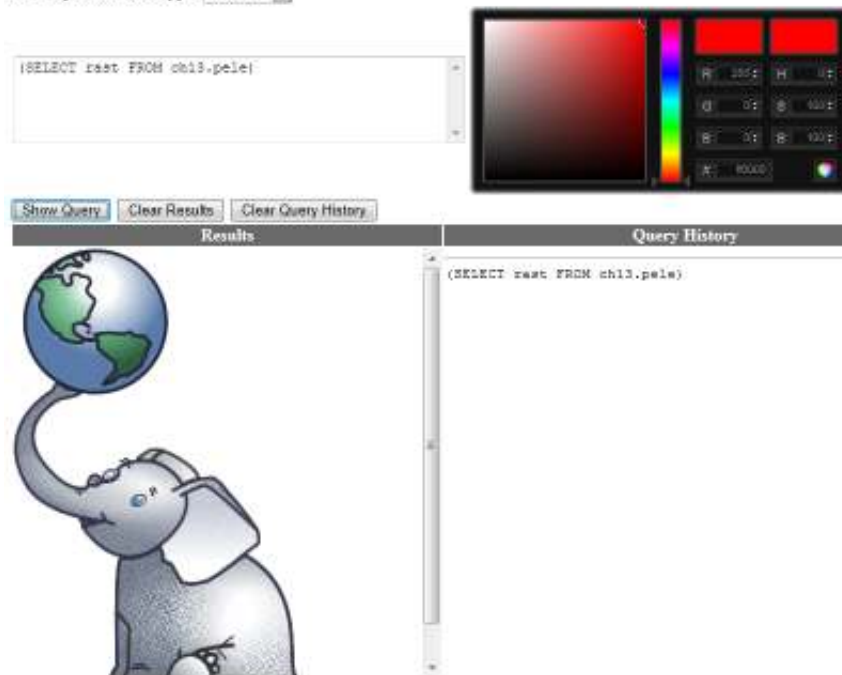
http://www.postgis.us/downloads/postgis_webviewer_php.zip

http://www.postgis.us/downloads/postgis_webviewer_aspnet.zip

Functions currently output only 1 raster tile, not set of

Minimalist PostGIS 2.0 Spatial Query Web Viewer

The Expression is of type: Raster



Behind the scenes:

```
SELECT
    ST_AsPNG (
        (SELECT rast
         FROM ch13.pele LIMIT 1),
        ARRAY[1,2,3]) );
```

Outputs the first 3 bands;

PostGIS 2.0 *ROCKS*

Constructor: ST_AsRaster

The Expression is of type:

```
(SELECT ST_Union(geom) FROM neighborhoods)
```

Show Query

Clear Results

Clear Query History

Results



Used in quickie viewer to render geometries in browser:

Behind the scenes:

```
SELECT ST_AsPNG(  
    ST_AsRaster(  
        (SELECT ST_Union(geom)  
        FROM neighborhoods),  
        200,200,  
        ARRAY['8BUI','8BUI','8BUI'],  
        ARRAY[255,0,0],  
        ARRAY[0,0,0])  
    ) ) ;
```

Creates a raster of 200x200 pixels in same projection as input geometry with 3 8BUI bands where data pixels are initialized to RGB (255,0,0) (a red color) and inactive – nodatavalue are set to black (0)

PostGIS 2.0 *ROCKS*

Constructor: ST_Polygon



```
SELECT ST_Polygon(  
        ST_SetBandNoDataValue(  
        rast,1,255)  
        ,1)
```

```
FROM ch13.pele;
```

Converts band 1 of raster to a polygon/multipolygon geometry. We are combining with ST_SetBandNoDataValue so pixel values = 255 are ignored.

```
MULTIPOLYGON(((244 -381,245 -381,245 -382,244 -382,244 -381)),  
              ((245 -382,246 -382,246 -383,245 -383,245 -382)),  
              ((216 -322,217 -322,217 -323,216 -323,216 -322)),...
```

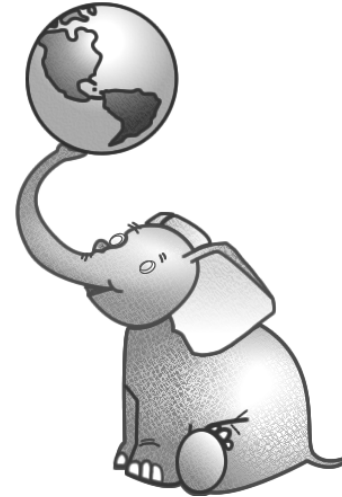
PostGIS 2.0 *ROCKS*

Constructor: ST_Band

Creates a new raster from existing just containing specified bands in specified order:

-- return new raster containing just band 3

```
SELECT ST_Band(rast,3) FROM  
ch13.pele
```



-- return new raster containing first 3 bands
reshuffled in different order

```
SELECT ST_Band(rast,ARRAY[3,2,1])  
FROM ch13.pele
```



PostGIS 2.0 *ROCKS*

Raster

Processing Functions

– lots too many to mention

```
ST_Reclass(raster ....)
```

```
ST_Resample(raster....)
```

```
ST_Transform(raster....)
```

```
ST_MapAlgebra(raster...)
```

some others and more coming

probably before 2.0 release ...

PostGIS 2.0 *ROCKS*

Processing: ST_Reclass

Creates a new raster from existing just containing specified bands in specified order:

-- Before

```
SELECT rast FROM
samples.downtown_chunked WHERE
rid=20
```

-- Reclassify pixels

```
SELECT ST_Reclass(rast,
ROW(1,'0-87]:100, (87-100]:150, (101-
254]:0-0', '8BUI',NULL)::reclassarg,
ROW(2,'0-253]:50, 254:0', '8BUI',
NULL)::reclassarg,
ROW(3,'0-70]:70, (70-100:100, [86-
150):250, [150-255:255', '8BUI',
NULL)::reclassarg)
FROM samples.downtown_chunked
WHERE rid=20
```



PostGIS 2.0 *ROCKS*

Processing: ST_Resample

Creates a new raster from original resampling by specified method

-- Before

```
SELECT rast
FROM aerials.boston WHERE rid=11;
```

-- Reduce size of aerial by 25% using CubicSpline

```
SELECT ST_Resample(rast, NULL,
1.25*ST_ScaleX(rast),
1.25*ST_ScaleY(rast), NULL, NULL, 0, 0,
'CubicSpline')
FROM aerials.boston WHERE
rid=11;
```

-- Reduce size of aerial by 25% using NearestNeighbor

```
SELECT ST_Resample(rast, NULL,
1.25*ST_ScaleX(rast),
1.25*ST_ScaleY(rast), NULL, NULL, 0, 0,
'NearestNeighbor')
FROM aerials.boston WHERE
rid=11;
```



PostGIS 2.0 Raster Intersection with geometry

**Pick a parcel / show average pixel value –
faster to work with lower res but less accurate**

-- band 3 average for overview - (avg pixval: 89.12 - 991 ms)

```
SELECT SUM(ST_Area((gv).geom)*(gv).val)/SUM(ST_Area((gv).geom))
FROM (
    SELECT ST_Intersection(r.rast,3, p.geom) As gv
    FROM massgis.parcels_boston As p INNER JOIN aerials.o_4_boston As r
    ON ST_Intersects(p.geom, r.rast)
WHERE p.map_id = '2010306000') As foo;
```

-- band 3 average for overview - (avg pixval: 136.7 - 3 secs)

```
SELECT SUM(ST_Area((gv).geom)*(gv).val)/SUM(ST_Area((gv).geom))
FROM (
    SELECT ST_Intersection(r.rast,3, p.geom) As gv
    FROM massgis.parcels_boston As p INNER JOIN aerials.o_2_boston As r
    ON ST_Intersects(p.geom, r.rast)
WHERE p.map_id = '2010306000') As foo;
```

-- band 3 average for full - (avg pixval: 137.8 -- 12 secs)

```
SELECT SUM(ST_Area((gv).geom)*(gv).val)/SUM(ST_Area((gv).geom))
FROM (
    SELECT ST_Intersection(r.rast,3, p.geom) As gv
    FROM massgis.parcels_boston As p INNER JOIN aerials.boston As r
    ON ST_Intersects(p.geom, r.rast)
WHERE p.map_id = '2010306000') As foo;
```


Mapserver Layer

```
LAYER
  NAME boston_aerials
  TYPE raster
  STATUS ON
  DATA "PG:host='localhost' port='5432'
        dbname='ma' user='ma' password='test'
        schema='aerials' table='o_2_boston' mode='2'"
  PROJECTION
    "init=epsg:26986"
  END
END
  Using aerials.o_2_boston
```

Using aerials.boston



Open Source Tools that work with PostGIS raster

GDAL – 1.8+ has PostGIS raster driver (looking for funding to improve performance)

<http://trac.osgeo.org/postgis/wiki/WKTRaster/GDALDriverSpecificationWorking>

QGIS beta support now via plug-in

GvSig beta support will be integrated in next release available as a plug-in now for current (but only works with older WKT Raster (0.1.6))

MapServer – the first to work – via GDAL driver 1.7+ (better to use 1.8+ GDAL driver)

Make your own with favorite tools using output functions:

http://www.postgis.us/downloads/postgis_webviewer_php.zip

http://www.postgis.us/downloads/postgis_webviewer_aspnet.zip

Questions