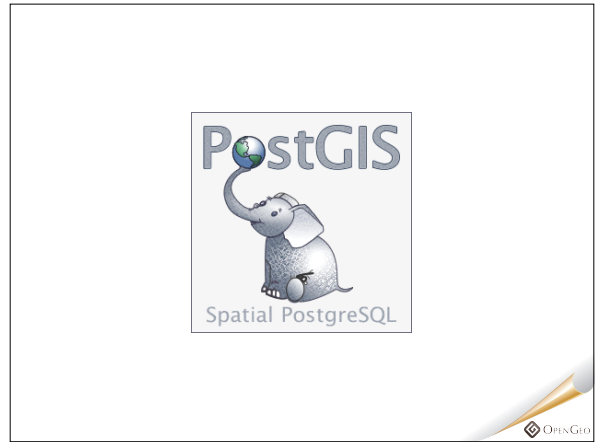


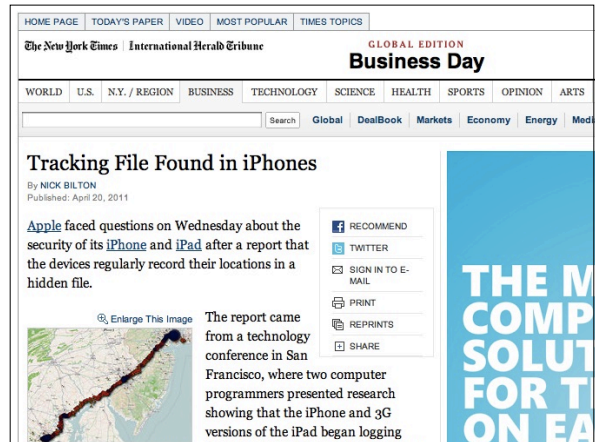
so, PostGIS knows where you are  
should you be worried?



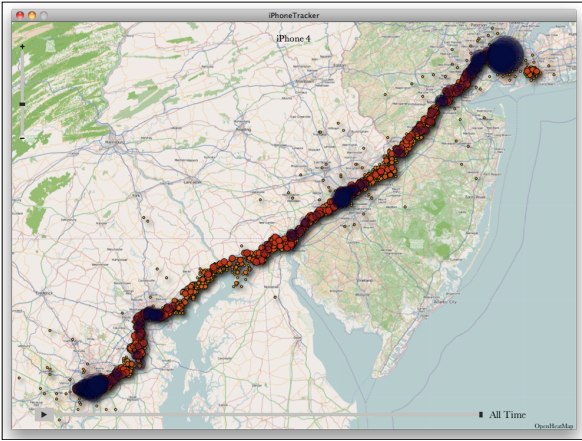
Of course not, PostGIS is a friendly elephant,  
with a ball  
what should be worrying you is



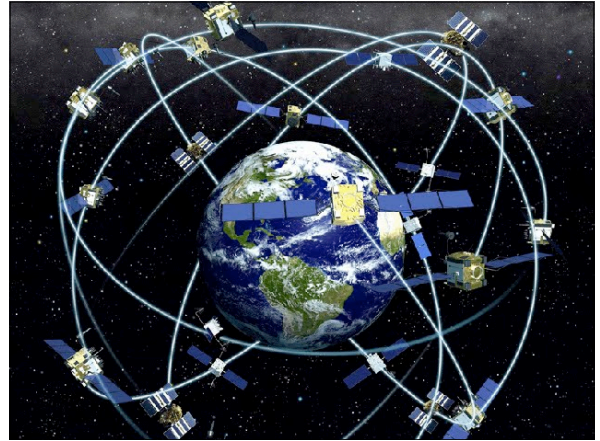
Apple knows where you are!  
at least, that is what the mass media have been  
telling us



some security researchers found a file of locations  
in their iPhone backups



and put them on a map  
and lo, it was a track of all the places they had  
been  
but Apple doesn't really care where you are,  
**you** care where you are  
so, Apples stores a record of all the WiFi points  
and Cell towers you've encountered, to more  
**quickly get a fix**



Because **GPS alone** is fairly slow, you have to find  
satellites and  
get signals and people are impatient  
and triangulating wifi and cell is faster, if you  
know where the signals originate to start with



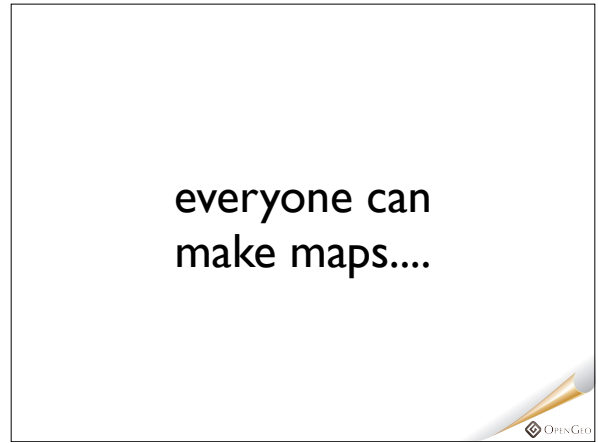
so Apple stores the locations in a little file  
a SQLite database  
but what was **interesting** was the mass outcry  
location! location! location!  
it's a relatively **new obsession** in the general  
world of IT



the whole story came out at Where 2.0,  
a conference about spatial data that **didn't exist**  
**until 2005**



what's special about 2005?  
 It's when Google Maps was released  
 where 2.0 was a response to the huge interest  
 generated by Google Maps and the Maps API  
 location was suddenly a mass IT interest



everybody now had the ability to put data on a map  
 and when everybody **can** do something, it follows that



everybody **must** do that thing  
 and now the use of location is now seen as a  
 critical plane of competition for technology giants

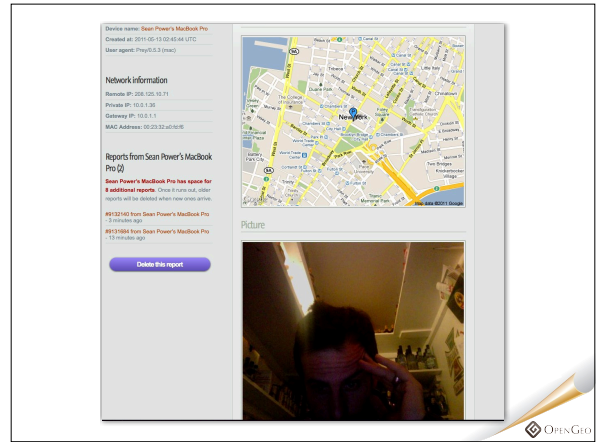


so all the major players have rolled out maps and/  
 or location APIS  
 <explain each>  
 and some of the hotter start-ups of the last few  
 years have been pure location plays  
 any foursquare users in the audience to day? get a  
 life.

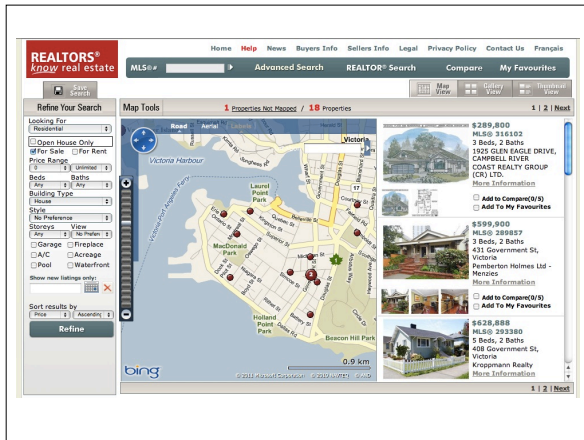
everyone is being followed...



but before you can put things on a map you need to know where they are, and nowadays we know where a lot of things are instantly and all the time



whether it's your stolen laptop in this case tracked and mapped using the Prey Project



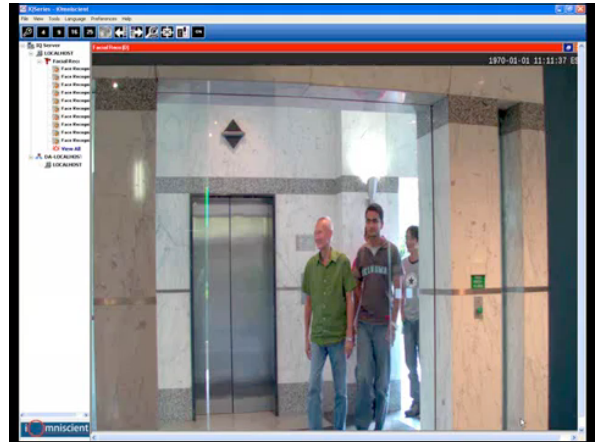
or your future dream house, here just a simple mash-up of house addresses on Bing Maps



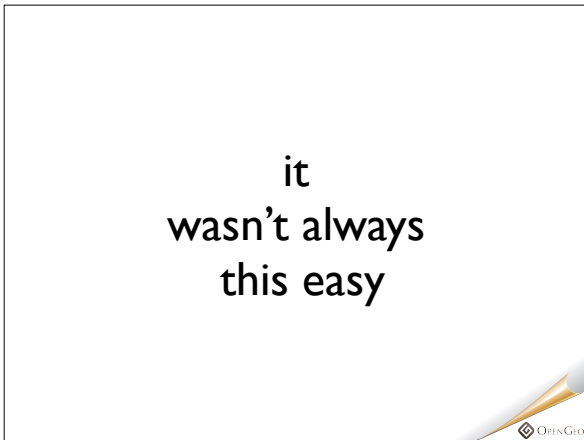
or your terrorist mastermind, surveilled in real time with the latest in aerial and satellite technology



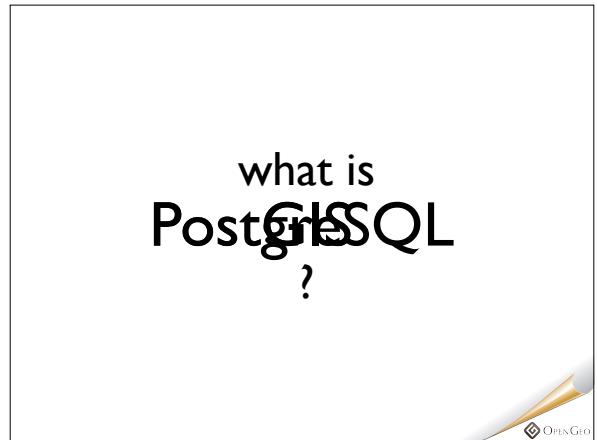
or you. And your phone.  
We're now willfully carrying around location sensors  
**all the time.**



And in our day-to-day lives, we're walking in front of sensors all the time.  
We're swimming in an ocean of location information

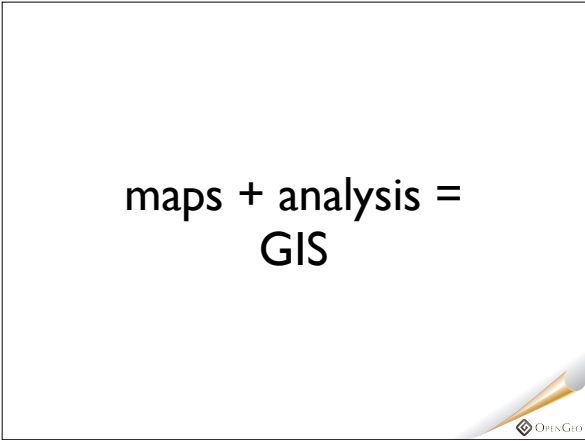


but it was not very long ago that location information was expensive to get, expensive to use, and expensive to display back in the days when we said "G" "I" "S" and not "location"



PostGIS is a fun name, it started out as a simple sound play  
Post-Gres-Q-L transforms to Post-Jis  
And it incorporates GIS, which is what we were aiming for  
But what is GIS?





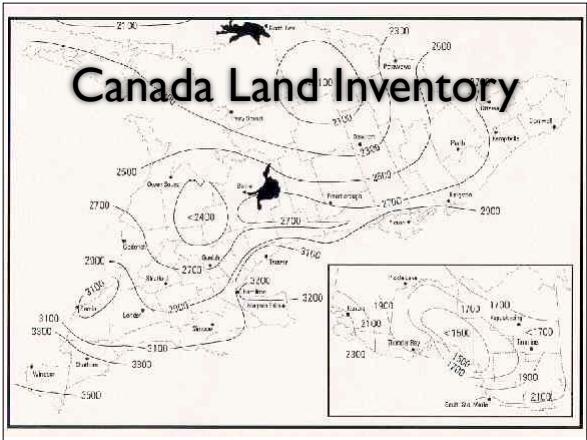
so, in GIS a map provides context,  
but the item of interest is often something else  
laid on top  
and the analysis can be calculation-based or just  
visual, like Snow's



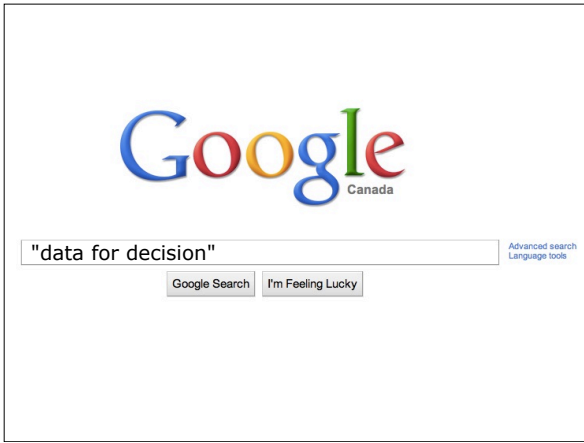
in the late 60s and 70s the item of interest to  
governments was soils.  
aerial photography, and resource inventory had  
generated lots of data  
but analysis was manual and slow



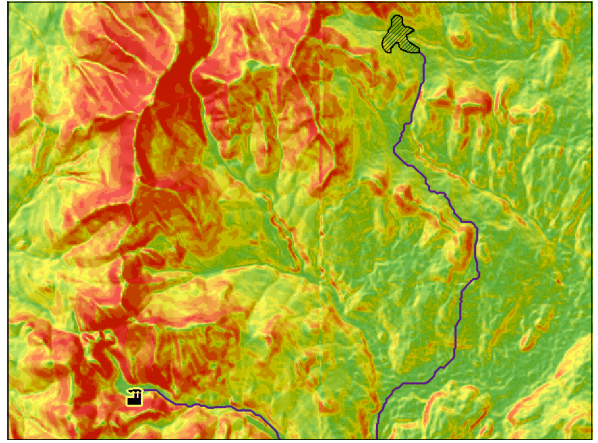
In 1968, Roger Tomlinson, father of GIS  
convinced the government of Canada to try  
computers.  
He digitized maps of much of Canada using a  
drum scanner.  
His system provided first-ever country-scale  
analysis tools.



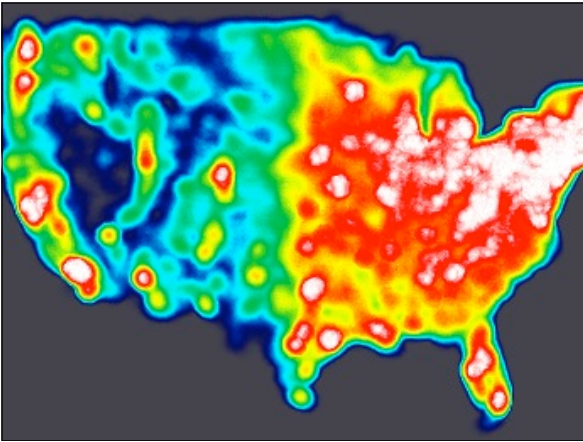
Tomlinson's work became  
the "Canada Land Inventory"  
really the first true "GIS"  
So, there's another thing for Canadians to be  
smug about,  
We invented GIS.



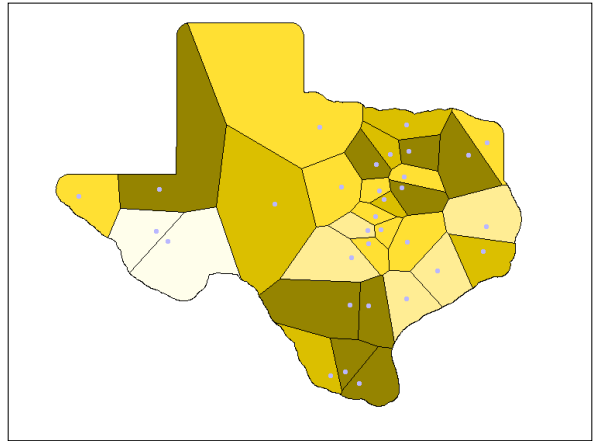
Those video clips can from a National Film Board of Canada documentary about the Tomlinson era GIS, which is very cool for both the history and the early technology it shows off punch cards, drum scanners, old map digitizers and so on just type "data for decision" into Google to find it.



Once you have your map data in a computer you can do all kinds of fun analysis with this is transportation cost surface for moving timber from cutblock to mill. Red areas are high cost.

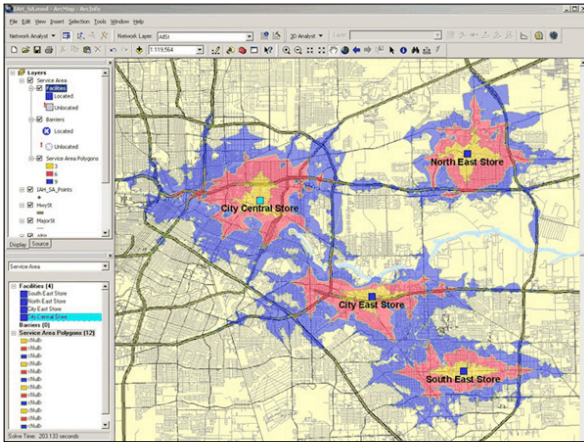


You can built heat maps from aggregations of point data (in this case concentrations of people).



You can segment your spaces with partitioning algorithms. Here creating Thiessen polygons around Texas cities.

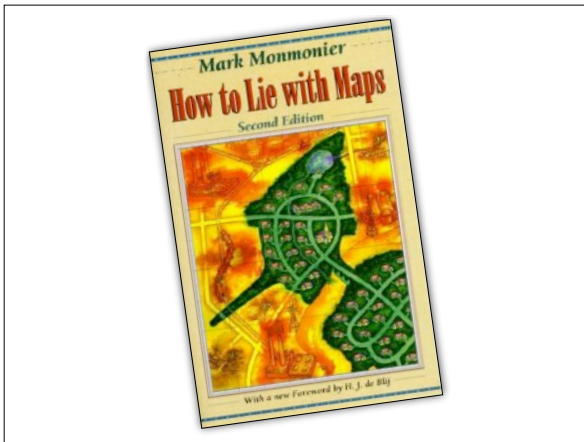




You can calculate the trade areas of your stores, trade areas are usually defined in terms of drive times.



Or most commonly, you can just print out meaningless colourful pictures. Who won in 2008? Choropleth maps of population information can be very misleading. A classic book taught in GIS schools is

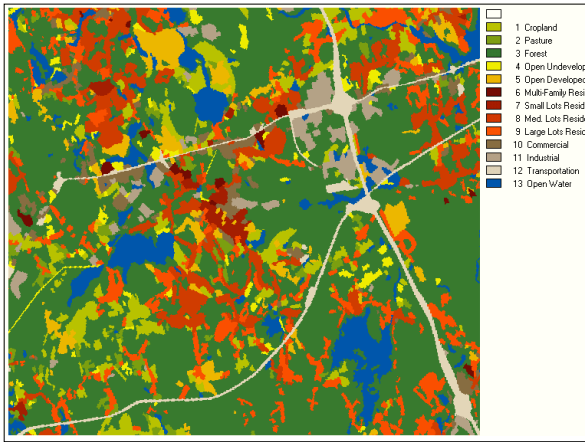


how to lie with maps. Even more than charts, maps are a great visual tool for telling misleading stories.

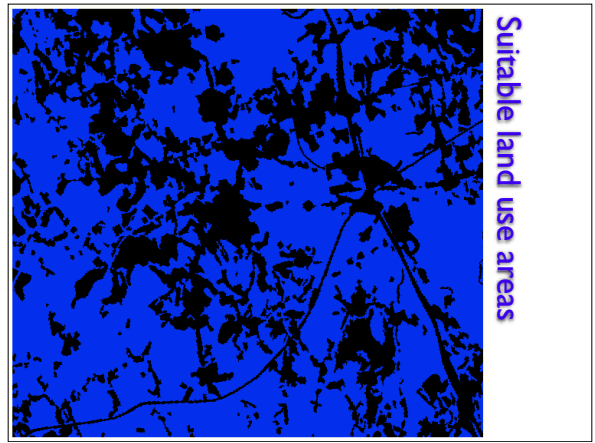
**Residential development**

- area 50 hectares or larger
- 50-meter buffer zone around lakes and streams
- area not already developed
- area currently farm, forest, pasture, or undeveloped
- no slopes > 15%

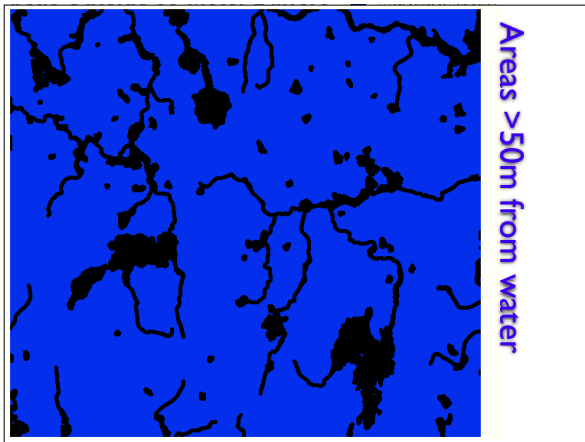
Anyway, the GIS view of the world is that you can decompose reality into a series of logical layers, that can be recombined to answer questions. Let's answer the play question "where is a good place for residential development" using some toy data and rules. The rules are ...



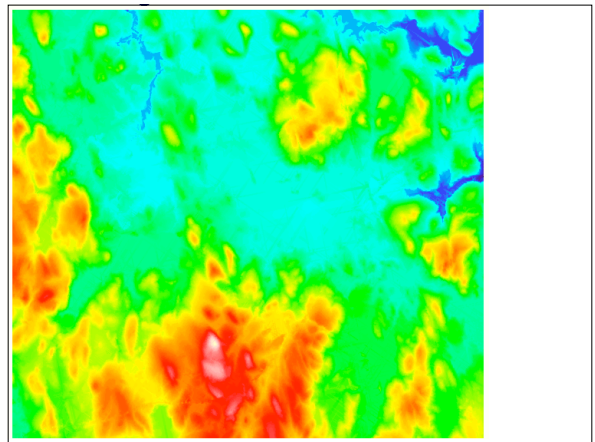
start with land use and strip away all the inappropriate current uses



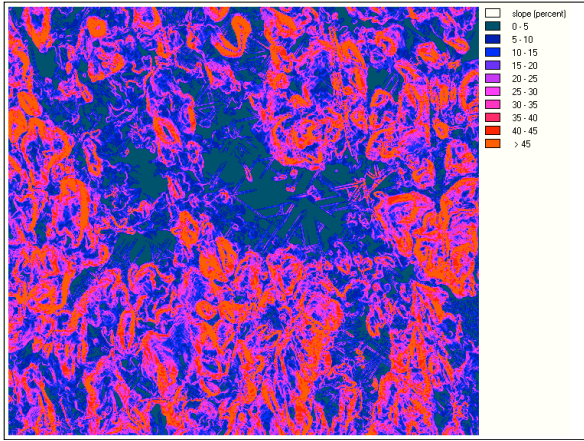
so our suitable land use areas are in blue



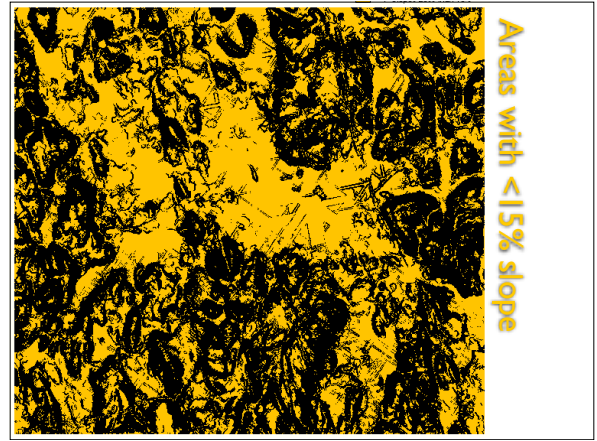
now take all the water features and buffer them to create another layer of areas more than 50 meters from water



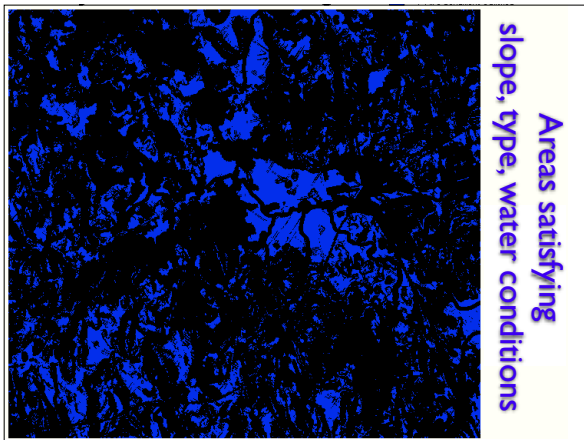
now take a grid of elevation measurements and from that calculate the slope at each grid cell



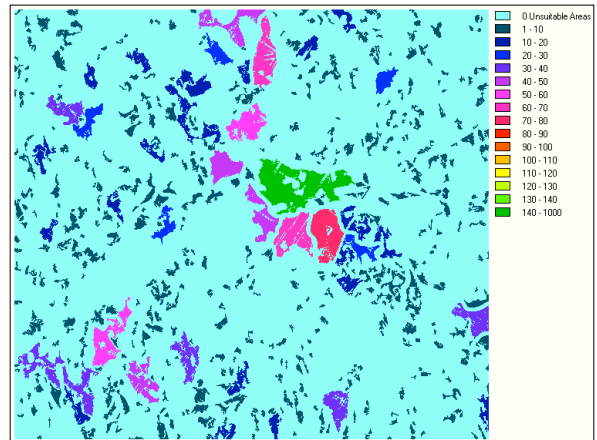
the high slopes are red and low ones are blue  
mask that to get just the low slopes



areas that are flat enough to build on  
now combine our three masks



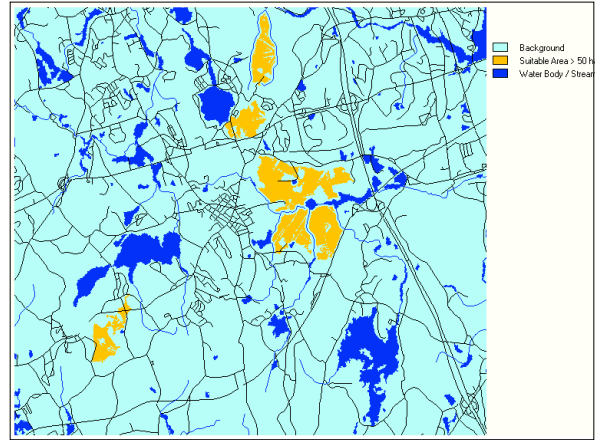
merging the land use, water buffer and slope masks  
gives us many areas big and small  
but we want areas > 50ha



so filter for groups of cells that form areas



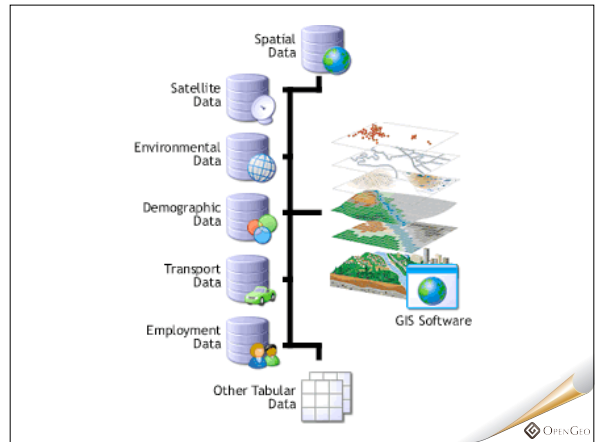
find the areas of > 50ha



and produce an output map with some roads and lakes for context  
that's a basic piece of GIS analysis  
a multi-factor combination of geographic layers

maps + analysis =  
GIS

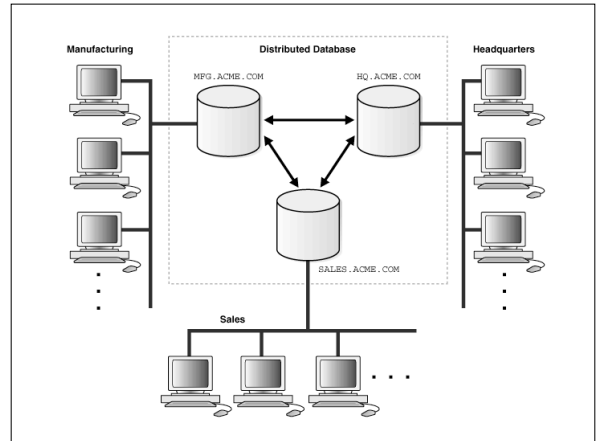
and until recently that kind of analysis has been done exclusively with "GIS software"



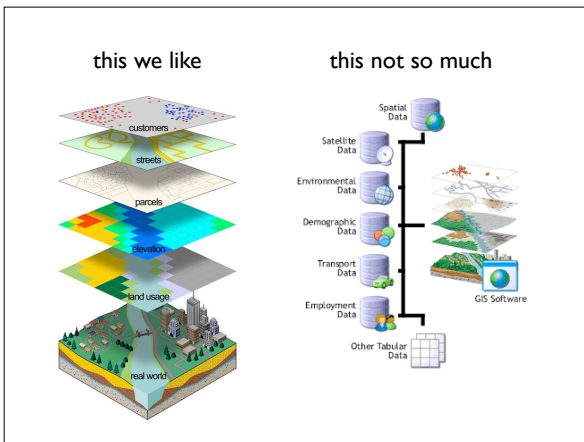
and the GIS community has an interesting view of its place in the world.  
see all the databases and where they end up?  
everything feeds the GIS...



there is an understandable historical bias among GIS practitioners to put GIS at the center of every workflow



But as we database people know, what is actually at the centre of **real** systems infrastructures is the **database**. So as GIS and location have become more important the worlds of IT and GIS are colliding and the IT folks are saying



“we like this model of the world, where objects have spatial representations, that’s a very powerful decomposition **but**, we don’t like your systems architecture, it’s kind of silly and, hey, we can take your layer model and decompose it even more..”

parcel_id	5123141
owner	Paul Ramsey
address	144 Simcoe St
assessment	128000
zoning	R2
tax_code	55
geometry	

Take your parcels layer for example, each parcel has information that fits in a database row, and if we put each parcel shape into the row too, we can

parcel_id	owner	...	geometry
5123141	Paul Ramsey	...	
5123141	John Simmons	...	
6124912	Beth Williams	...	
5123145	Jo Arvay	...	

actually represent the whole layer as a **single table** in the database!

layer as table?

but can database software handle this use case?

- able to support “non-standard” types
- able to handle arbitrarily large objects
- able to efficiently index objects in  $\mathbb{R}_2$  or even  $\mathbb{R}_n$
- able to provide functions for non-standard types

it can, if we have a database that is capable of a few things

...  
can anyone think of a database that can do all that?

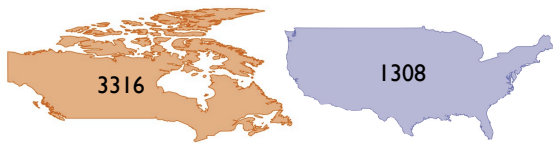
PostgreSQL

Version 7.1

of course, although it was not until version 7.1 and the TOASTable tuple that we could handle large enough geometries

## Large enough?

Page size = 8192 bytes  
Coordinate = 2 \* 8 = 16 bytes  
Max coordinates = 512



surely an 8K page size is large enough?  
not to do GIS practically  
even assuming no headers and no other  
information in a table except geometry  
maxcoords = 512  
but Canada = 3316 and USA = 1308

# PostGIS SQL

OpenGeo

Anyhow, so, there's actually another meaning to PostGIS  
It's not just a sound play on PostgreSQL  
It's also an historical statement  
Post-Gres is what came after In-Gres  
Post-GIS is what comes after traditional GIS  
It's a new way of thinking about GIS

“We need to contact  
everyone within 5000m  
of the reactor!”

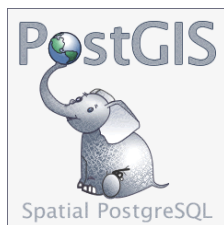
```
SELECT owner_phone  
FROM house_parcel  
WHERE ST_DWithin(  
    geom,  
    'POINT(...)',  
    5000 );
```

so we can answer questions that used to require  
GIS software  
using database queries

“Does bus #12 need maintenance? How far did it travel last week?”

```
SELECT
  Sum(ST_Length(geom))
FROM
  bus_paths
WHERE
  bus_id = 12
AND
  path_date > (Now() - '7d');
```

One SQL statement to answer a location services query.  
SQL in a database is very powerful, **more powerful** than desktop GIS in terms of amount of code required and the size of datasets than can be queried.



These are all questions we can answer with PostGIS

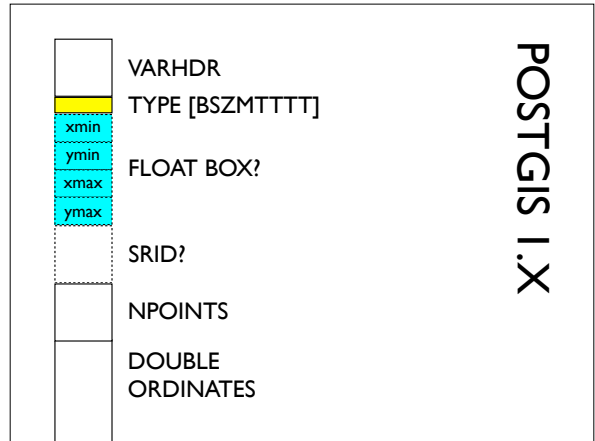


PostGIS came into being in 2001, in a Victoria consulting company named Refrations Research the first revision wasn't by me, but by Dave Blasby (until 2004) and Sandro Santilli (until 2006) and me from 2008.

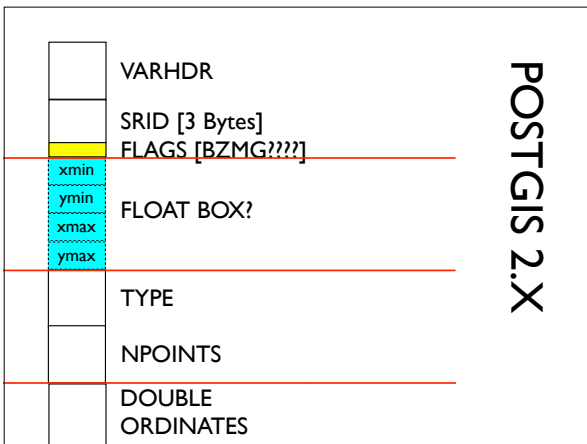


```
CREATE TYPE geometry (
  internallength = variable,
  input = geometry_in,
  output = geometry_out,
  send = geometry_send,
  receive = geometry_recv,
  delimiter = ':',
  analyze = geometry_analyze,
  storage = main
);
```

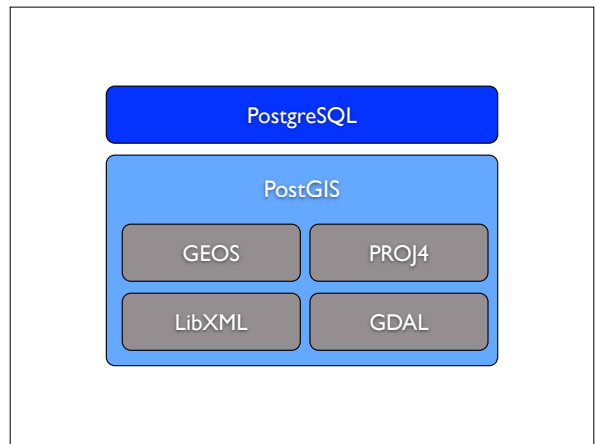
in PostgreSQL, PostGIS **geometry** is just a variable length type



as a PgSQL variable length object, PostGIS structure starts with VARHDR, then... the BOX and SRID are optional, so that small objects (points, short lines) have lower metadata overhead



For 2.0, we have reordered things a bit and added some space for extra flags and type numbers (with only 4 bits for type numbers in 1.0 we ran out). The objects are a bit heavier, but are now double-aligned for the coordinates, so we hope to add more efficient coordinate access.



The overall architecture of PostGIS makes use of three specialized libraries from the GIS realm and one for XML, we have more obscure dependencies than PostgreSQL

## GEOS

- C++, LGPL, computational geometry
- PostGIS ST\_Relate() support
- `bool geometry::touches(geometry)`
- `bool geometry::contains(geometry)`
- `geometry geometry::union(geometry)`
- `geometry geometry::buffer(double)`

## PROJ4

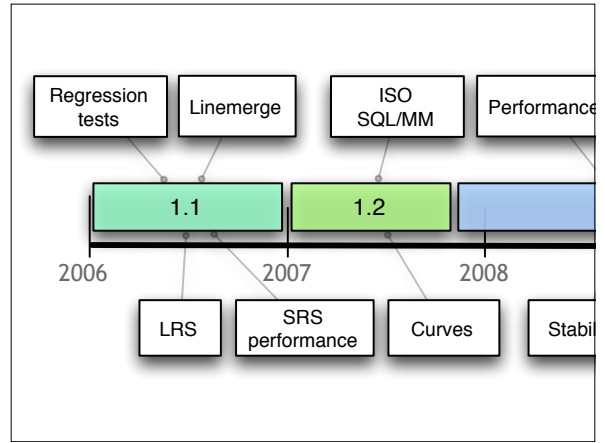
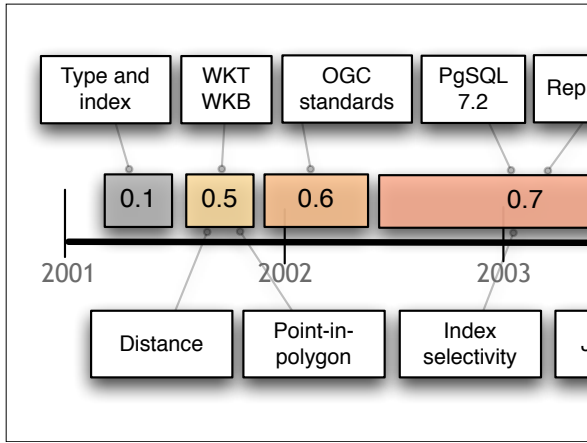
- C, BSD, coordinate transformation
- PostGIS ST\_Transform() support
- `int pj_transform(projPJ src, projPJ dst, long point_count, int point_offset, double *x, double *y, double *z );`
- `+proj=aea +lat_1=55 +lat_2=65 +lat_0=50 +lon_0=-154 +x_0=0 +y_0=0 +ellps=clrk66 +datum=NAD27 +units=us-ft`

## GDAL

- C++, BSD, raster operations & formats
- PostGIS RASTER support
- `GDALDataset::GetRasterXSize();`
- `GDALWarpKernel::PerformWarp();`
- `GDALDataset::BuildOverviews();`

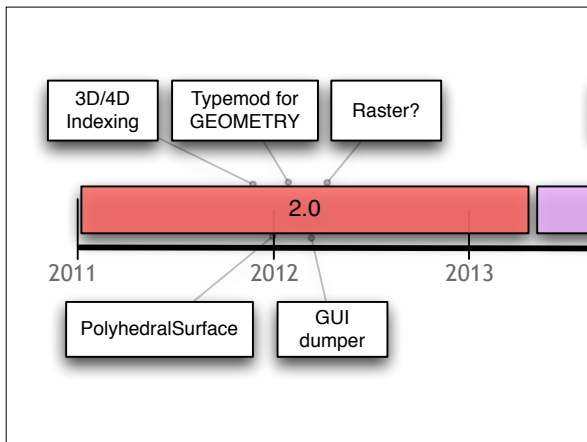
## LibXML2

- C++, XML parsing / generating
- PostGIS ST\_GeomFromGML()
- PostGIS ST\_GeomFromKML()
- Not ST\_AsKML() or ST\_AsGML()!



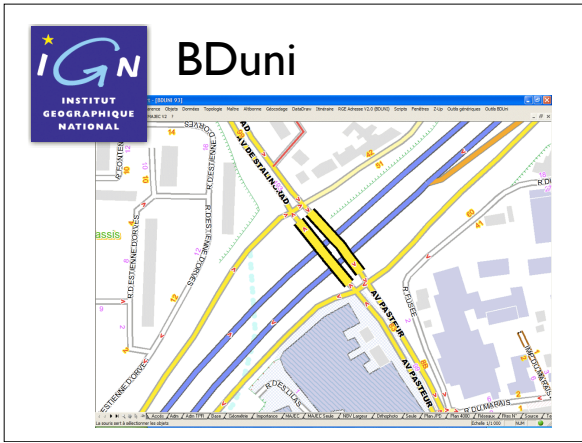
Basic functionality all done in year one (2001).  
 Mapserver! P-i-P!  
 Full SFSQL by year four (2004).  
 Geoprocessing (2007)  
 poly-build, LRS, non-standard utility functions  
 (dump, make)  
 More standards / performance (present)

Basic functionality all done in year one (2001).  
 Mapserver! P-i-P!  
 Full SFSQL by year four (2004).  
 Geoprocessing (2007)  
 poly-build, LRS, non-standard utility functions  
 (dump, make)  
 More standards / performance (present)



who's using PostGIS?

Basic functionality all done in year one (2001).  
 Mapserver! P-i-P!  
 Full SFSQL by year four (2004).  
 Geoprocessing (2007)  
 poly-build, LRS, non-standard utility functions  
 (dump, make)  
 More standards / performance (present)



IGN, managed national base map in GIS software BDuni 120M features, nationwide in 2003, was managed with files and GIS software wanted to start using a database

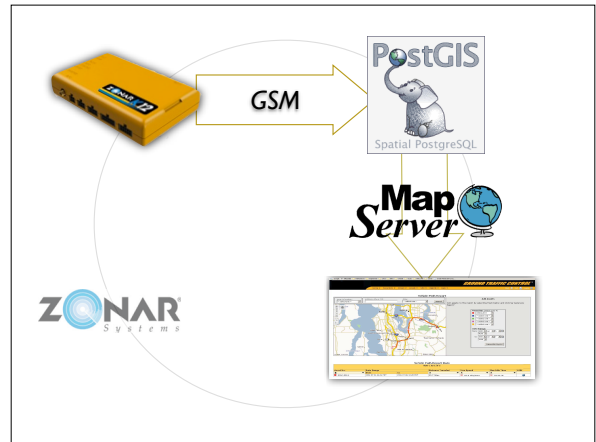


## DB Evaluation

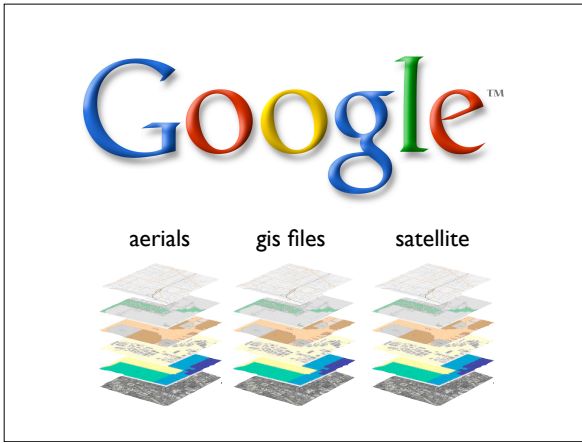
- PostGIS? DB2? Oracle?
- Can DB handle 100M spatial features?
- Can DB do spatial transactions?
- Yes! Yes! Yes!



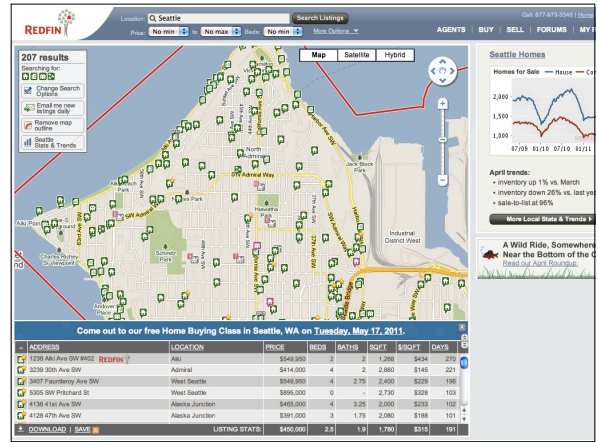
startup companies are using postgis  
zonar makes vehicle tracking hardware  
and runs software service to map fleets



locations come in from hardware  
are stored in PostGIS  
and mapped by MapServer



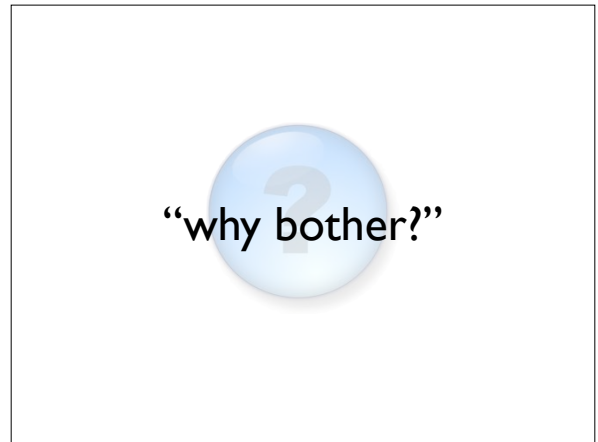
Google uses PostGIS in a similar way to manage metadata about all the GIS data they are storing and how they have processed it



Real-estate listings company, Redfin started with MySQL, found spatial queries too slow (bad planner), moved to PostgreSQL and PostGIS.



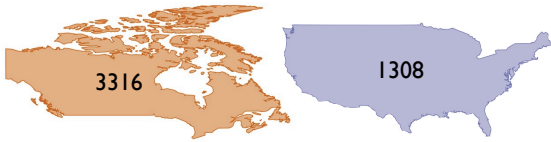
So talking to PostgreSQL people about PostGIS, there's some questions that come up naturally that I wanted to address.



Why even write PostGIS? PgSQL already has native PATH and POLYGON and BOX and CIRCLE, isn't PostGIS redundant?

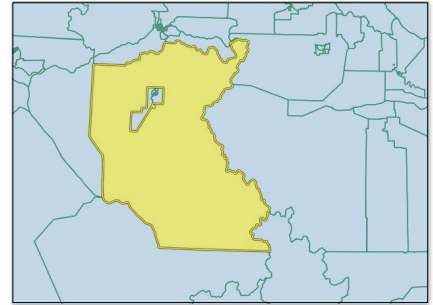
1. Native types were computer graphics primitives not GIS

- GIS data objects are large, type must be TOASTable



size  
...

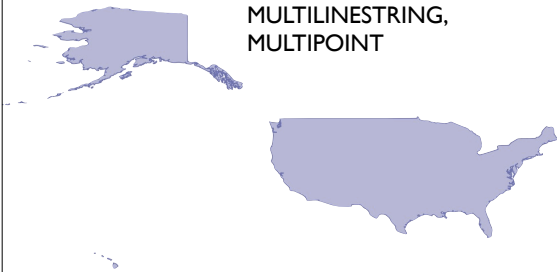
- GIS polygons have holes



holes  
...

- GIS objects require “aggregation”

- MULTIPOLYGON,  
MULTILINESTRING,  
MULTIPOINT



finally aggregation

...  
ok, so if the native types weren't good enough,  
why not just improve them? why isn't PostGIS part  
of PostgreSQL

“why isn't PostGIS just  
part of PostgreSQL?”

and that answer is fairly complex, but it comes  
down to a number of **lucky** historical accidents,  
because it's a **VERY GOOD THING** that PostGIS is  
not inside PostgreSQL

“why isn’t PostGIS just  
part of PgSQL?”

Necessity:  
Don’t really **need** to

In 2001, at the start, because PgSQL had this nifty type extension mechanism.

Why patch existing code if the patch might not be accepted when we could try out our ideas in an extension?

So that’s what we did, and it worked great.

“why isn’t PostGIS just  
part of PgSQL?”

Licensing:  
GPL vs BSD



I submitted PostGIS to patches in 2001, and the license was raised as an issue. At the time, all the PostGIS contributors were in the company, so we could have re-licensed (that’s no longer true). But there were other objections also.

“why isn’t PostGIS just  
part of PgSQL?”

Size:  
That’s a lotta code.

The size objection was only raised by a couple folks, but it was raised.

Between java hooks and regression tests, PostGIS 0.5 was about 400k compressed.

“why isn’t PostGIS just  
part of PgSQL?”

Quality:  
That’s some ugly code...

Fortunately, we never actually got to the point in 2001 where anyone looked closely at the code, but honestly, it was **fugly** then. It’s **fugly** now. You probably would have hated it.

“why isn't PostGIS just part of PostgreSQL?”

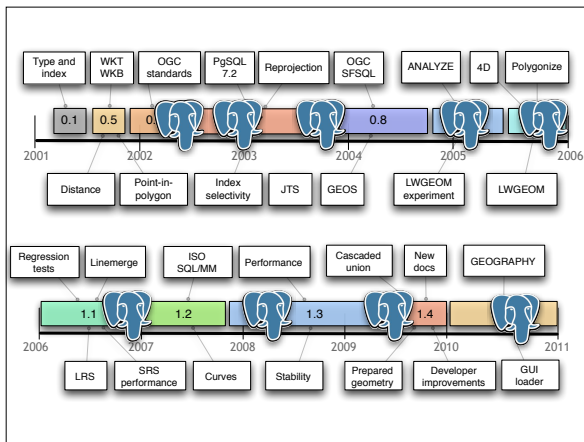
Necessity:  
Don't really **need** to

And in the end, given the lack of strong agreement in the core team and the fact that we didn't **need** to be integrated to work, it just didn't happen.

“why isn't PostGIS just part of PostgreSQL?”

It's **better** this way.

And I think that's a good thing  
Being an add-on has had very positive effects

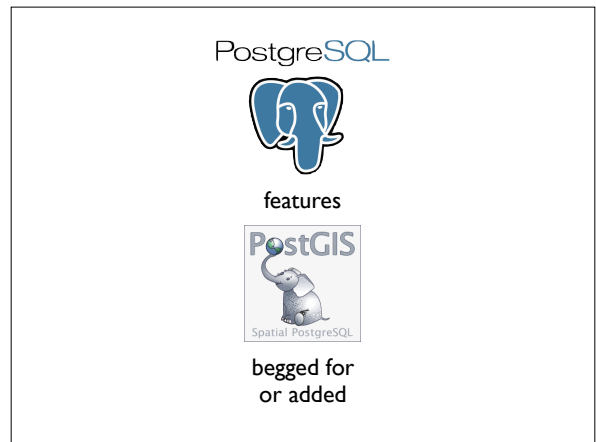


First, we have a very small team, 5-6, of which maybe 1-2 people are actively working at any given time.

We find it hard to synchronize and test on our **own schedule**.


Our schedule doesn't necessarily match the PostgreSQL schedule.

Having to put features out on the PostgreSQL schedule would mean many features would be delayed and delivered much later than otherwise.




The existence of a complex and active add-on like PostGIS has helped improve the support for add-on's in PostgreSQL in general.




PostgreSQL features  begged for or added


- space for more dimensions in table stats
- adding an ANALYZE hook for user defined types
- building add-ons without requiring a full source tree
- support for more complex projects in PGXS
- typmod support for user defined types

PostgreSQL features  begged for or added



**Tom Lane:**  
 PostGIS is our biggest, highest profile example of a third-party add-on, it pushes the limits, which is good.

So PostGIS has helped push the envelope, Tom, I hope you'll forgive the paraphrase, but I couldn't find the original email

PostgreSQL features  users funded



and the presence of a skilled PostgreSQL development community has allowed us access to new advanced features, with the simple application of money

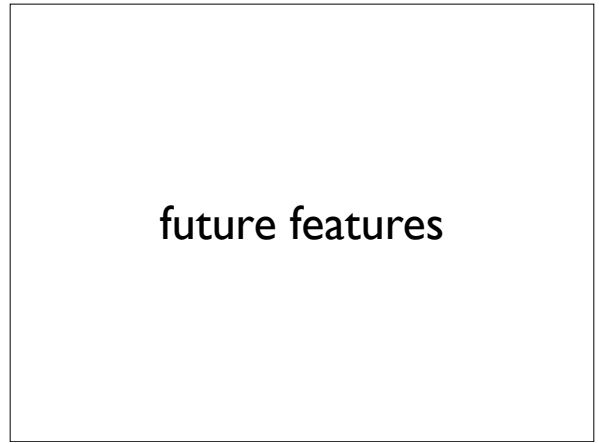
PostgreSQL features  users funded 

- improved GiST concurrency (8.1)
- GiST KNN searching (9.1)

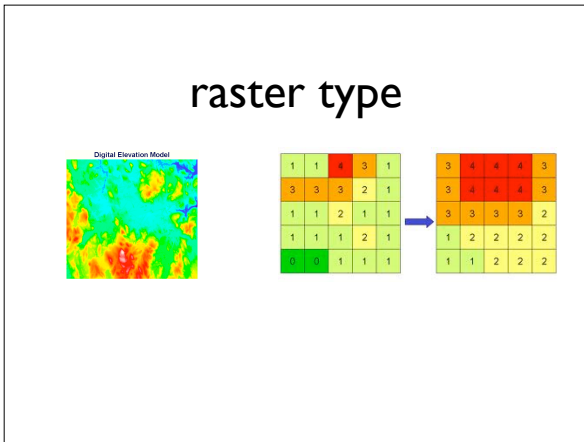
GiST concurrency was funded by a consortium of companies with an interest in high-speed PostGIS. KNNGiST was funded by a client of OpenGeo.



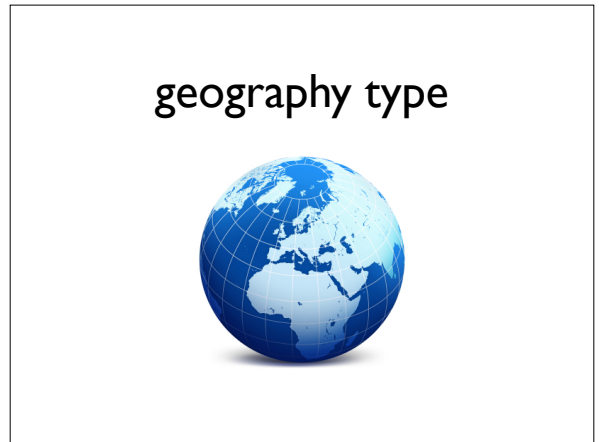
so, PostgreSQL and PostGIS go together like chocolate and peanut butter is there room for improvement?



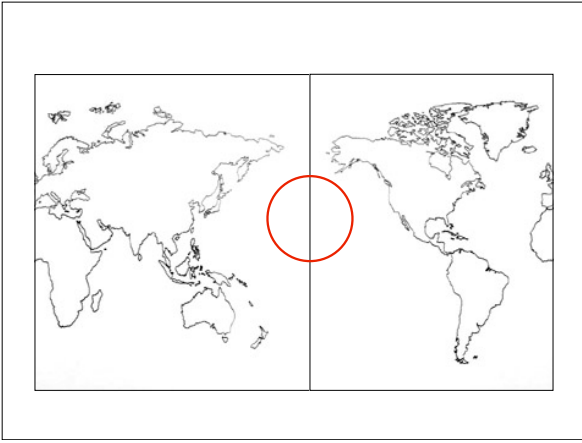
here's some new and upcoming features we know are happening



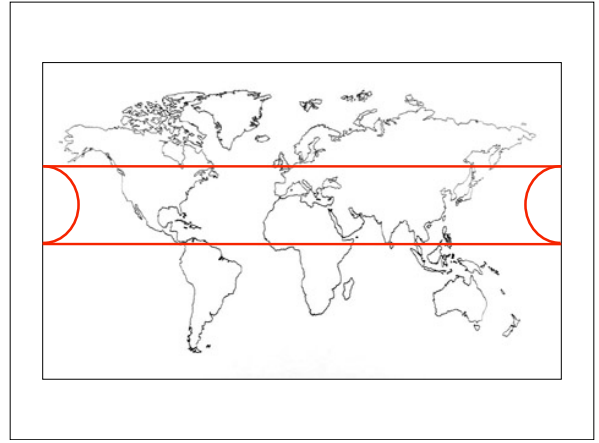
some data, like continuous surfaces are easier to model in raster and raster map algebra, combining grids and running kernels on grids is powerful combined with raster/vector and vector/raster conversions, a good addition for PostGIS 2.0



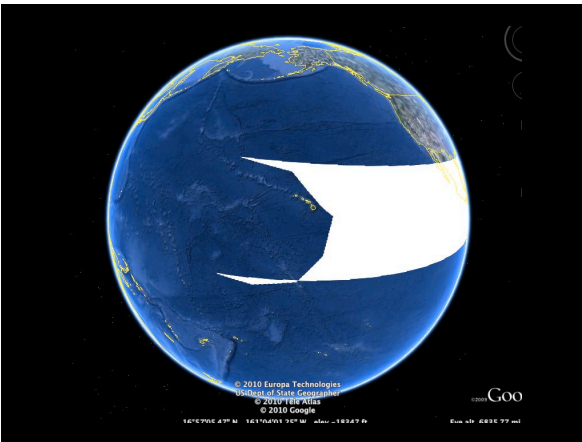
Up until 1.5, PostGIS only had a "geometry" type. The "geography" type added in 1.5 allows direct support for spherical coordinates, that is, latitude/longitude points.



You could store latitude/longitude in “geometry”, but things didn’t always work the way you might expect. For example, a simple thing like a circle around Hawaii, when you interpret it in a plane instead of on a sphere,



You get very bad results. Instead of a circle around Hawaii, you get a box that covers almost everything **except** Hawaii. It’s that an obvious error, why did it take so long to support spherical coordinates?



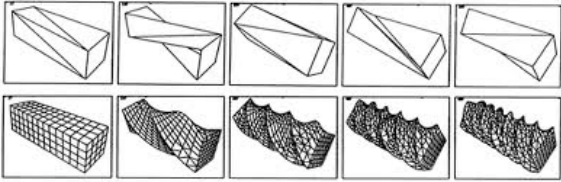
Because it’s mathematically hard, and you can mostly work around it, except when you can’t. Even Google Earth has a hard time with polygons at the dateline and over the poles.

3d/4d indexing

POINT(XYT)

for PostGIS 2.0 handling data streams from devices will increasingly involve good analytics and speed against large temporal data tracks, which means higher dimensional indexing

## 3d objects



building modelling in 3D data collection  
and underground reservoir modelling is driving  
the need  
for real volumetric objects  
PostGIS 2.0

## future workloads

looking at the future of PostGIS workloads,  
the ways people are **really exercising** PostGIS and  
PostgreSQL hard  
this is what I'm seeing

## write-scalability

this is a problem outside the spatial world,  
but I'm seeing it inside as well

foursquare



SimpleGeo

foursquare started on PostGIS and had to  
transition relatively early on to MongoDB to  
handle the write load  
all the check-ins coming in could not be written  
fast enough  
SimpleGeo "AWS of geo" built their system from  
scratch on Cassandra

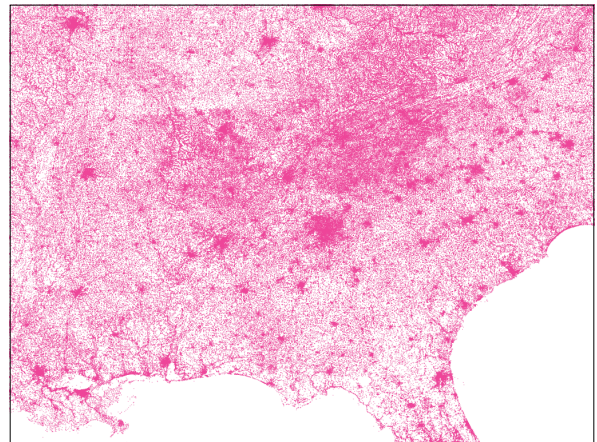


Postgres-XC?

but it's not just web start-ups, anyone monitoring a fleet of GPS devices has a pretty large data stream to deal with  
Zonar deals with it by partitioning by customer, but some customers fleets are so big they also have to partition by region  
it's not ideal

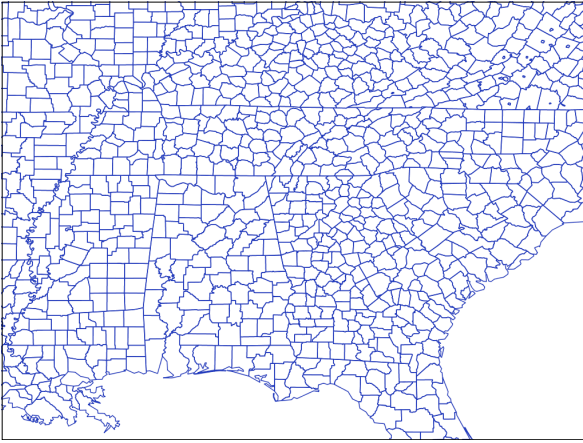
This is a problem outside my abilities, but it's nice to know other folks are already thinking about it and working in detail  
Hopefully the solutions will continue to support user types like PostGIS

parallel processing

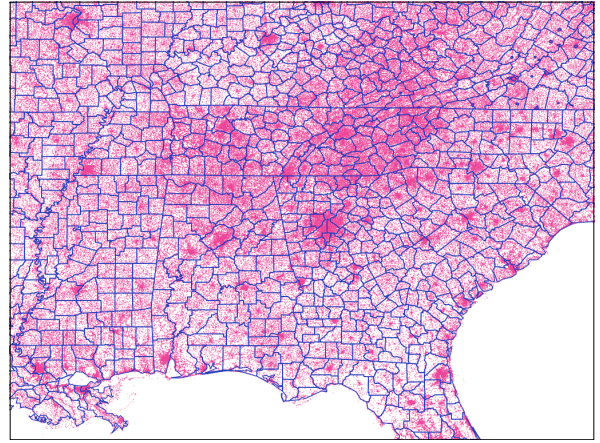


another **big future issue** is that people are using PostGIS for larger and larger analysis runs

Suppose you had a big customer database, millions of customers, with geocoded addresses, suppose you're Walmart.  
And suppose that, for marketing purposes, you want to know race, income, and other things about your customers you **can't get** from the cash register purchase records.



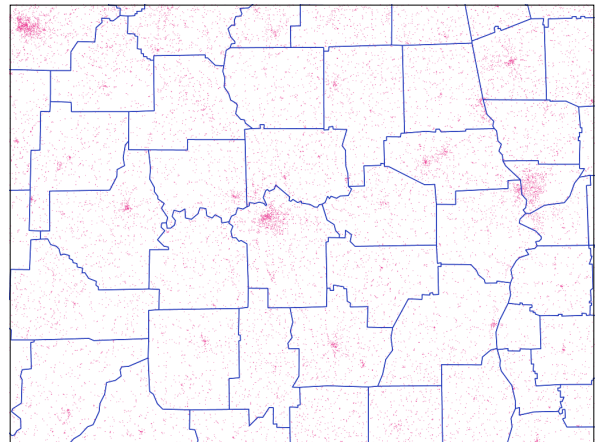
You might take the US census data, which contains the race, income and demographic fields you're interested in,



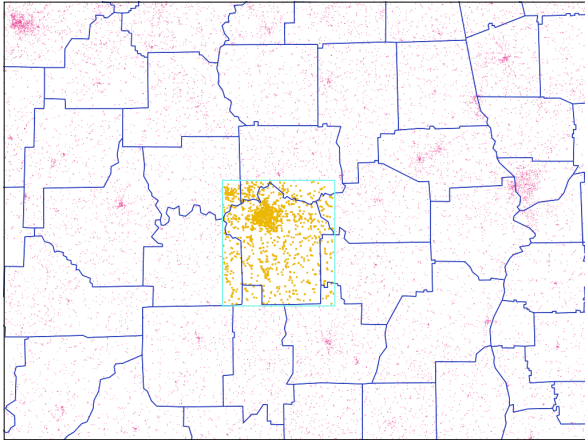
and join the census data to the customer data using a spatial query,

```
SELECT
  census.*, customers.*
FROM
  census, customers
WHERE
  ST_Contains(
    census.geom,
    customers.geom);
```

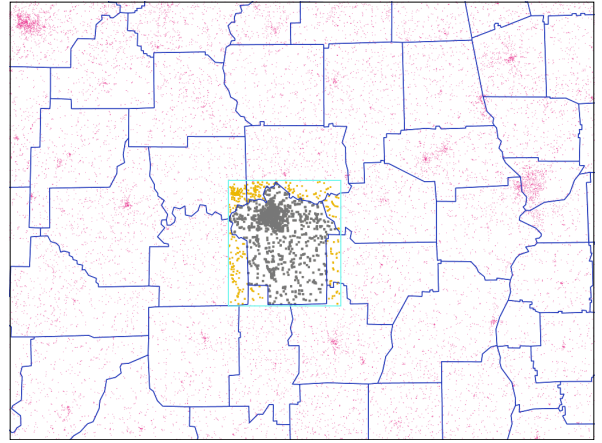
something like this, which would spit out a table that added census attributes to customers, and you could pipe into a statistical analysis to get your answer.



However, in running the query, the index will perform a nested loop, so for each county we'll get the set of points that meet the index condition



and within that set we will go through each point and test to see if they meet the exact condition of containment within the shape



to get the final set, but the painful part is that each of these millions and millions of customer points will be tested sequentially as each county is handled in sequence and each point in each county is handled in sequence

one core per query

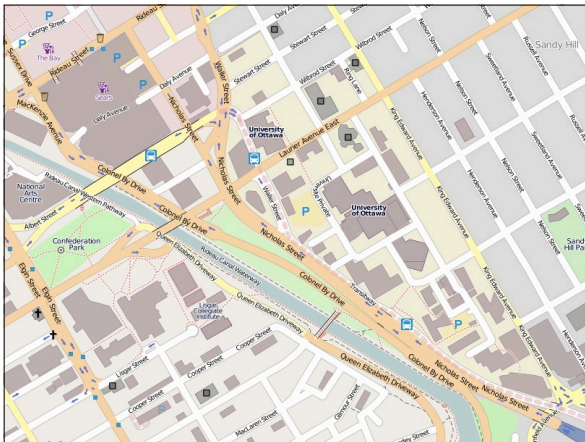
even if you have a 16-core computer, the query will only heat up one core

if your queries are short and cheap sequential is OK

for ordinary workloads parallelism is from multiple simultaneous queries postgres forked back-end works fine

if your queries are long  
and expensive  
sequential is !OK

do these polygons intersect?  
calculate the union of these shapes.  
what is the geodesic distance between these  
shapes  
are all very very very expensive questions,  
compared to the usual simple database  
calculations



Hard problem to solve, but very useful in real  
world data processing  
OpenStreetMap, is a map wiki, and the data they  
gather is road lines and boundary edges.  
To form polygons to fill in pretty maps they run  
their data through PostGIS to condition it, a  
process that takes many hours.  
Parallelization would improve their process  
hugely.

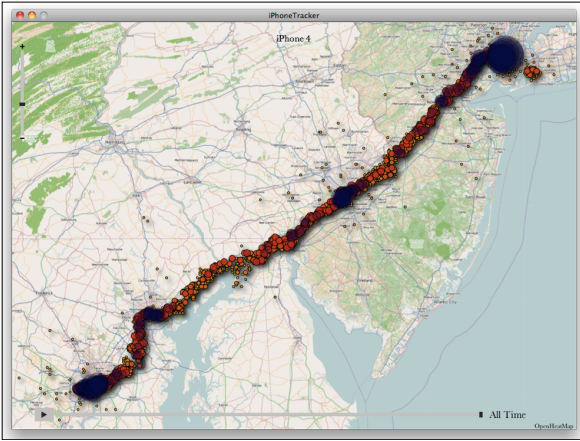
GIS analysis could be  
parallelized to great  
effect

but for GIS analysis workloads, there is just one  
query, with many trivially parallelizable parts  
each county-versus-pointset test is independent  
each point in polygon test inside that is  
independent  
within each point in polygon routine, even point-  
versus-edge tests are independent

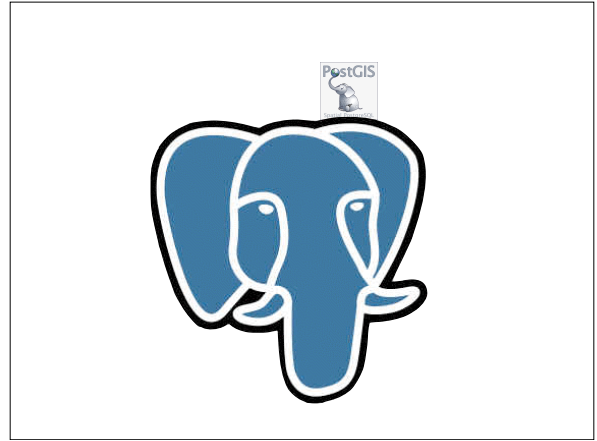


The future of PostGIS and location in general  
revolves around all these sensors we're all  
carrying in our pockets, and all the new satellites  
being blasted into space.  
The volume of data flowing in is only going to get  
higher, the analysis needs only going to get  
larger, the dimensionality is going up, from 2d to  
3d to temporal.





We're all generating immense corpuses of data, and the tools to analyze and map our personal data should be open source. We need to control our data and to control our tools.

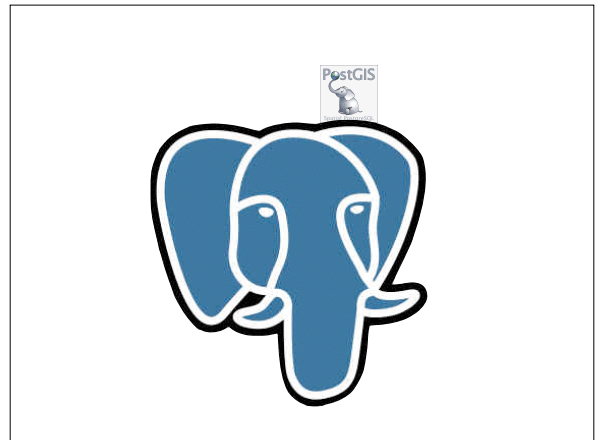


As a community, PostGIS been fortunate to be able to build on your incredible work. We've definitely been standing on the shoulders of giants.

## “Sure...”

- “PostGIS has triggers and foreign keys and procedural languages”
- “PostGIS has full ACID transactions”
- “PostGIS can do hot back-ups”
- “PostGIS can do replication and warm stand-by”

and thanks to you giants, I've been able to have some wonderful conversations about the “features of PostGIS”...



So let me say right now, on behalf of the PostGIS community, “thank you for making us look so damn good”. You're building the worlds most advanced open source database, you're giving people the power to manage their own data, and the tools to build amazing things.



thanks you!

Thanks to your help, we're look forward to many more years of great development. Thanks for having me here today, let's have a great conference.