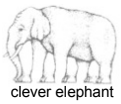
Zen and the Art of Web Mapping

Getting our hands dirty with open source geospatial software

Paul Ramsey
pramsey@cleverelephant.ca


clever elephant

While you sip your coffee,

Find someone with the workshop DVD,

Copy DVD contents to C:\

When finished, you should have directories

c:\ms4w

Mapserver for Windows
TileCache
OpenLayers

c:\ms4data

Medford Oregon Data

c:\ms4fun

Other Useful Software
This Presentation

Do not try and copy into sub-directory,
things will not work right.

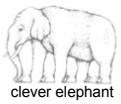
Get viral!

Copy your copy onto a thumb drive and give it to your neighbor!

The Art

Not everyone understands what a completely rational process this is, this maintenance of a motorcycle. They think it's some kind of a "knack" or some kind of "affinity for machines" in operation. They are right, but the knack is almost purely a process of reason, and most of the troubles are caused by what old time radio men called a "short between the earphones," failures to use the head properly.

- Robert Pirsig, Zen and the Art of Motorcycle Maintenance


clever elephant

We suffer in information technology, as much as any technology (motorcycles, CD players) from a black box mentality.

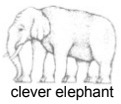
There is the box, and there is ourselves. The box does as we say, but we do not know how. What is inside the box, it's better not to know.

Open source takes the walls off the box, exposing the mechanisms, and we have to find it within ourselves to trust that the mechanisms are as rational and understandable as the other mechanisms we have learned, and mastered, in our lives.

The Zen

The truth came knocking at my door, and I said,
"Go away! I'm looking for the truth!"
And so it did.

- Robert Pirsig, Zen and the Art of Motorcycle Maintenance

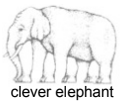

clever elephant

However, in our pursuit of understanding, let's not loose sight of the larger goals.

We want to make things that work.

Open source?

- Open source is software you can legally modify and redistribute
  - Linux
  - Apache
  - Firefox
- So what? I'm not a programmer.


clever elephant

There are a lot of misunderstandings about what open source software is, and is not.

Open source is not "freeware" or "shareware". That is, the primary feature of interest about open source is not the price. In fact, there is no requirement that open source software be distributed free of charge.

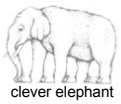
At the core, open source is an agreement between the person distributing software, and the people receiving it. The distributor gives the recipient the right to make changes to the software, and to redistribute the software with those changes. That's it.

I like to compare open source software to recipes. You can buy a cook book, alter one of the recipes, and give the new recipe to a friend. You won't be contravening an "End Cooker Licensing Agreement" by doing so.

All this sharing of source code seems rather beside the point to an end-user or implementer, who is not going to be looking at the source code.

Open source?

- You're not an auto mechanic either.
- "Buying closed-source software is like buying a car with the hood *welded shut*." – Bob Young, Red Hat founder
- Would you buy a car with the hood welded shut?
  - If not, why not?


clever elephant

A car with a welded hood eliminates customer choice. There is only one place to get that car serviced, at the original dealer.

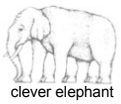
A car with a standard hood, can be serviced at any corner repair shop, or even in your own driveway, if that's the kind of person you are.

We recognize that the welded car is a bad deal, to us as customers, but we are not quite at the point yet where we recognize the bad deal that closed-source software represents.

Some of that is just because open source is still relatively obscure, the concept is a bit foreign to us, we are used to thinking that using black-box software is just the Way Things Are Naturally.

Customer Rights

- "Microsoft reserves all rights not expressly granted to you in this EULA … The Software is licensed, not sold."
- "The license rights granted under this EULA are limited to the first thirty (30) days after you first install the Software unless you supply information required to activate your licensed copy in the manner described during the setup sequence of the Software."


clever elephant

The reason we don't buy cars with hoods welded shut is because it allows the vendors to do things like this to us.

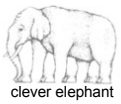
First, in order to get around the fact that we have very wide rights to do what ever we want with our own property, they ensure that the product is **not our property**. That is not your copy of Windows XP, Bill Gates has just rented you the (limited) right to use it.

Second, Bill will revoke your rights, if you do not send him your registration information within a month of clicking through the EULA.

Source: Windows XP End User Licensing Agreement, http://www.microsoft.com/windowsxp/home/eula.mspx

Customer Rights

- "If an implied warranty or condition is created by your state /jurisdiction … you also have an implied warranty or condition, but only as to defects discovered during the period of this limited warranty (90 days). As to any defects discovered after the 90 day period, there is no warranty or condition of any kind."



clever elephant

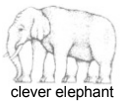
Well, that's a bit draconian. But at least if something goes wrong, you have a $250B company standing behind you, right?

No, not really.

If your jurisdiction mandates a warranty period by law, they will give you one, for 90 days. After that, you're on your own. If your jurisdiction doesn't mandate a warranty period, you're on your own from the get-go.

Customer Rights

- "Your exclusive remedy for any breach of this Limited Warranty is as set forth below. Except for any refund elected by Microsoft, YOU ARE NOT ENTITLED TO ANY DAMAGES, INCLUDING BUT NOT LIMITED TO CONSEQUENTIAL DAMAGES, if the Software does not meet Microsoft's Limited Warranty, and, to the maximum extent allowed by applicable law, even if any remedy fails of its essential purpose."


clever elephant

Oh. Well, if something goes really FUBAR, I can always lawyer-up and get a pound of flesh out of Bill that way, right?

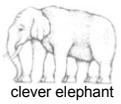
No, if you want to use the software, you have to forswear any right to damages if the software causes you grievous harm.

And once you have accepted the EULA, it applies to all updates too, so if Microsoft sends you an update that fries your computer (by accident, of course, they aren't malicious, just avaricious) and loses your Nobel-prize-winning manuscript for the Great American Novel… you are SOL.

clever elephant

Imagine if we bought cars this way. We'd still be driving Pintos that did this.

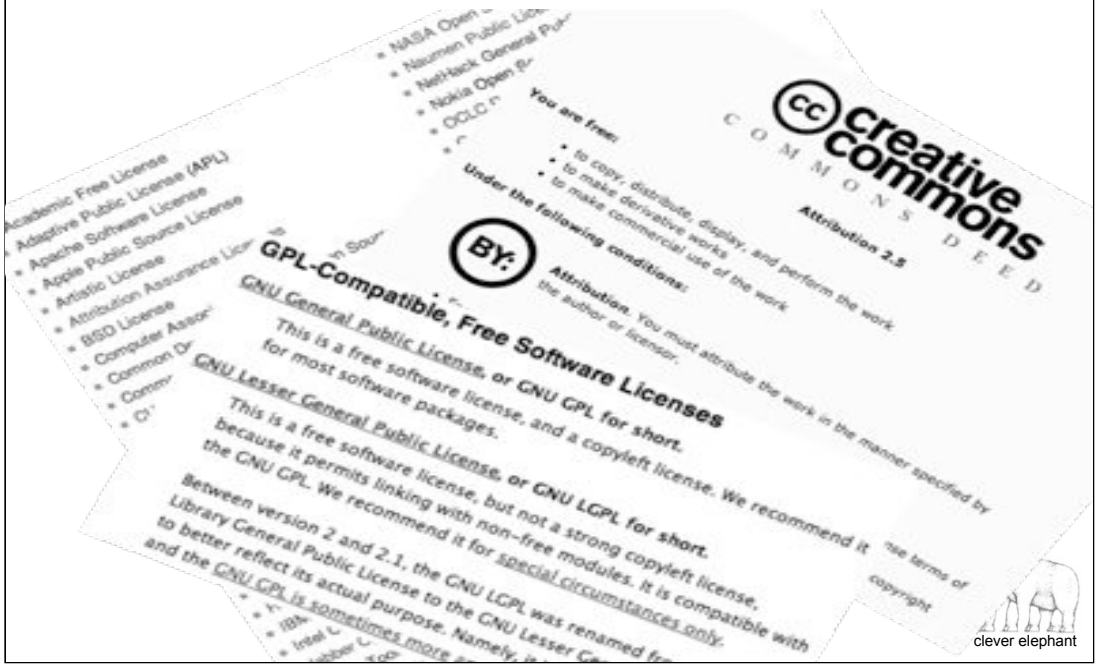
Open source myths



clever elephant

In the geospatial world and the general IT world, open source labors against a number of myths or misunderstandings, and I'm going to discuss a few of them.

“Open source is about licenses”



Discussions about open source bog down too frequently in license arcana:

•Why does the GPL really require?

•Does BSD allow companies to walk away with the software?

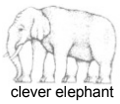
•Should I be afraid of GNU licenses?

•What is a “real” open source license?

Licenses are a means not an end. The “end” of open source software is an active, global, collaborating group of users and developers. The secret sauce is ensuring that everyone has an equal stake in the outcome, and open source licenses allow that. But freely licensed software without a community is just so much bit-waste, it is not the license that is the important ingredient.

“Open source has no support”

- Community support
  - Mailing lists
  - IRC
- Consulting support
  - Hourly help
- Commercial support
  - Fixed contracts



clever elephant

Vendors make us buy support, it’s part of the cost of doing business. For vendors, it can be extremely lucrative too! Oracle’s 2007 annual report indicates that they brought in $8.3B in support revenue, but that support cost them just $1.3B to provide -- that’s an 84% profit margin. A large part of that is customers religiously paying up fixed annual contracts for support services they never actually use. Why don’t they use them? Because they aren’t that helpful.

The open source model includes some companies providing fixed support, but they too follow the Oracle model: sell contracts to N customers and hope only N/5 of them actually call for help. If they all call, you won’t make any money.

A better bet is to make use of the support channels that work for you. Judge not on the price, or the “traditional” nature of the support, but the quality of the help.

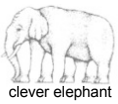
The free online mechanisms are many times far better than any paid support -- it is a comment that recurs on the mailing lists over and over.

But for everyone who receives excellent help on the mailing lists, there will be someone who receives no help at all, and when you aren’t paying for help there is no recourse beyond stamping your feet.

That leaves the consulting support option: identify an expert and put them on contract to help you when you need help. The reality of these things is, the amount of help you need will rarely exceed the amount you would pay in fixed maintenance/support contracts anyways.

“Open source is for tinkering hippies”

The new paradigm always looks a little radical compared to the traditional ways…



clever elephant

First of all: What do you have against hippies? Or tinkering?

Second of all: This is technology, it is not about how you look, it is about what you *do*. The open source idea is changing the way customers relate to the software they use, in radical ways, and radical changes usually come with a certain amount of associated “scruffiness” in the messengers and the message.

However, over time, and not very much time, if it is a winning idea, the suits will jump on board.

On the left is Richard Stallman, the founder of the Free Software Foundation and father of the ideological free software movement.

On the right is Linus Torvalds, author and still the maintainer of the Linux operating system kernel.

These aren't hippies

So, for example, these little organizations all have major investments in open source.

•Google runs on Linux and open source.

•HP sells Linux-enabled servers and invests in kernel developers on staff to make sure they work right.

•IBM was the first major established corporation into the open source pool, initially with the Apache Foundation, and then with their big push into supporting Linux on every platform, and finally with the inception of the Eclipse project.

•Oracle has been buying open source companies (InnoDB, Berkley DB) left and right (make a play for Jboss) and have their own Linux distribution now (Unbreakable Linux)

•Sun Microsystems has open sourced Java, open sourced Solaris, and open sourced OpenOffice. All their major software platforms are open source now.

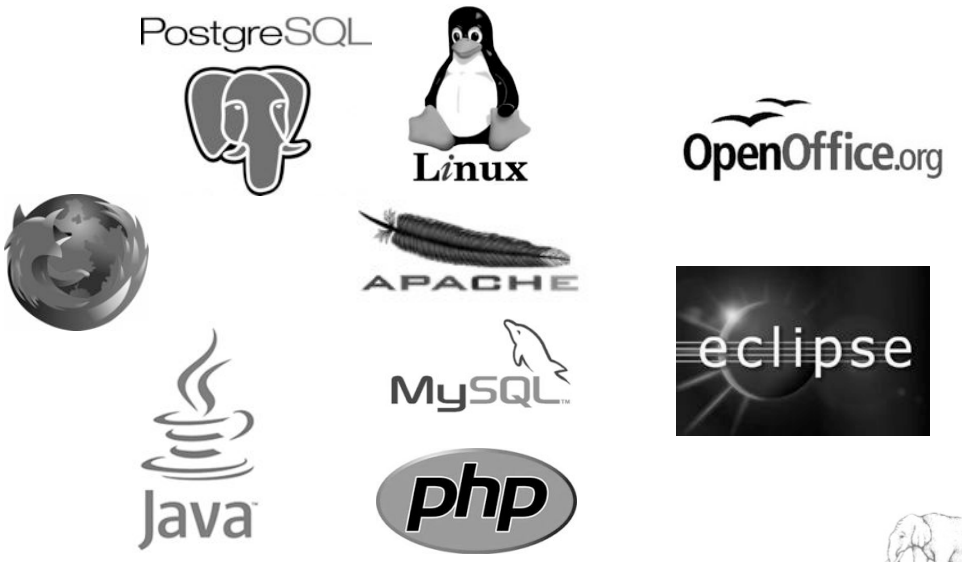
•Red Hat was an open source startup, now they are worth $3.8B on the markets.

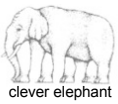
•Novell owns the SUSE Linux distribution and purchased Xandros, a maker of desktop tools for Linux. They have migrated their NetWare suite onto Linux.

•In the geospatial realm, NASA built and open sourced Worldwind, as a data viewer initially for NASA public data.

•The NSA was one of the first big government agencies to do an open source project, with their SE Linux (Security Enhanced). SE Linux is now part of the mainline distributions.

These aren't tinkerings





clever elephant

•OpenOffice is starting to take real market share away from MS Office. This going to be tough on Microsoft, since Office is a top earner, and one of the best reasons for running Microsoft on the desktop is so you can run Office.

•Apache runs a huge percentage of the internet.

•PHP is the scripting language of the internet, because of its availability on hosted web servers. It's also used on some very big sites, like FaceBook, for example, and Yahoo!.

•MySQL is the default database of the internet, also used by Yahoo!.

•The combination of Linux, Apache, MySQL and PHP is called "LAMP", and it's a ubiquitous combination in web hosting sites.

•Firefox has done what no one thought was possible a few years ago, and actually reduced the market share held by IE, despite having to be manually downloaded and installed, unlike IE or even Apple's Safari browser.

•Java is the new language of server enterprise applications, thanks to Oracle and others promoting it with Sun.

•Eclipse has basically taken over the Java IDE category and is now impinging into other application categories.

PCs were for Hippies Too

clever elephant

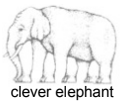
Here's the last thing to remember when thinking about the "Open source is for hippies" myth:

PCs were for hippies and small-time tinkerers too. The IT managers of 1978 looked down their noses at this crew, and the products they were putting out, but in the end the world changed, and the IT managers had to change along with it.

Microsoft circa 1978, taken in Albuquerque, New Mexico shortly before the move to Redmond, Washington.

The wheel turns…

- Open source adoption is part of a generational technological shift



clever elephant

Technology changes, sometimes in incremental ways relative to the dominant paradigm, sometimes in ways that grind the dominant paradigm into dust. We've been through a couple of those changes in my professional memory, though I only observed the echos of the first one and heard second-hand tales.

Technological change is continuous, but some changes alter the marketplace and others are simply integrated into the existing system.

The paradigm shifters so far have been

•the standardized commodity PC, and

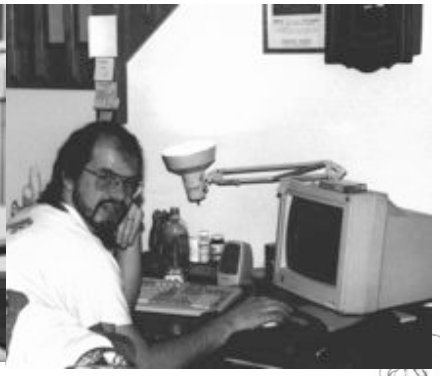
•ubiquitous network connectivity, and its ultimate expression, the internet.

Because these shifts have taken decades to work through, they have been accompanied by physical generational shifts, as decision makers comfortable with one technical and economic paradigm are replaced over time by younger decision makers with new perceptions.

Each stage is marked by more openness, more componentization of information processing, and more market competition in the various components of information processing.

1975 ⇒ 1995

- Mainframe Dominant ⇒ PC Dominant
- IBM down, Microsoft up



clever elephant

In 1975 you got your hardware from IBM, and your software from IBM, and your support from IBM. One vendor, little competition.

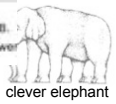
In 1995 you got your software from Microsoft, and your hardware from anywhere. The hardware/software link had been broken and the market opened up. We had a very free market in hardware, thanks to platform standardization, and a semi-free market in software, with Microsoft dictating how much competition would be allowed at each level (none at operating system, little for core applications, an every growing category and lots for “third party” or “too small for Microsoft to care about”).

Note that the while the technology changed radically **so did the economics** of the technology market. There was a lot more competition, and margins for everything except Microsoft products fell dramatically. And the systems were more open: the hardware was now standardized and swappable, even the software side was open, at least at the application level, on the standard Windows API.

Also note that by 1995 IBM wasn’t out of business. It was even still selling mainframes. It hadn’t been “crushed” by Microsoft. But it no longer dictated the course of the computer industry: Microsoft did.

1995 ⇒ 2015

- PC Dominant ⇒ Internet Dominant
- Microsoft Down, Google Up



clever elephant

Ubiquitous networking means information processing no longer depends on having physical access to a device powerful enough to store all the information and handle all the processing… it just needs to be powerful enough to view the results. That means things can be centralized, driving down the cost of service provision so far that for some services it's possible to provide them for "free", or rather, for the relatively small amount of money available from advertising.

The internet has changed the way we work with information, but more importantly it has changed the economics of how we work with information, and that is driving the current cycle of generational change. Why is no one starting shrink-wrap software companies these days? Same reason no one was starting mainframe companies in 1985.

Note that the economics of providing "nearly free" services to people are hostile to proprietary software. Internet scale services require massive horizontal scaling, and proprietary software just costs too much to be economical in an ad-supported world, where the answer to "how do they do it?" is always "volume!"

Note I am not predicting that Microsoft is doomed. I am predicting, and in many ways this predication has already come to pass, that Microsoft will not be the central fact of the computer industry. In many ways, the rise of the Google, the pure internet company, is already pushing Microsoft aside.

Each time these technical transitions occur, there is all kinds of room for new ways of doing business, new companies, and new economies. The economy of the swappable PC component, and shrink-wrapped software was pretty alien to the mainframe culture, but it was the backbone of PC culture.

The economy of free services, and the need to scale systems economically to millions of users is a feature of the internet economy, and it pretty much requires open source as a building block. Note the pervasiveness of the LAMP stack… you can copy that stack onto as many servers as you want, and serve as many users with it as you want.

Facebook, for example, was written in LAMP.

2015 ⇒ 2035

- Internet Dominant ⇒ ???
- ??? Down, ??? Up

clever elephant

Pointing out this pattern leads inevitably to the question of what the “next” generation will be. Someone knows, probably.

I don’t think “mobile” will be it, since “mobile” is just an extension of “ubiquitous networking” and “device independence”. Maybe something more surprising, like AI and semantic technology.

Sun was founded in 1982, and in 1984 when PC dominance was obvious, one of its founders, John Gage, coined the phrase “the network is the computer”, which became Sun’s company slogan. It didn’t stop them from suffering horribly for being in the wrong niche for the Internet age, internet servers instead of internet services.

We're only partway through…

clever elephant

It took 20 years for Microsoft to go from this…. To this.

We're only partway through…

It's taken 10 years for the open source corporate community to go from

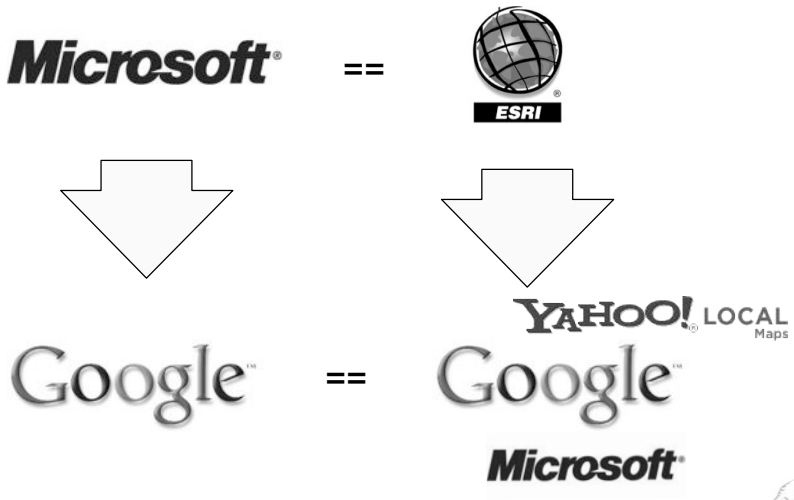
This (Red Hat had 2 employees in 1995) ….

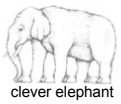
To this just over 10 years later.

Note who is missing from this collection of open source supporting companies… Microsoft

And then note that the core of Microsoft's revenue comes from two products, Windows and Office, that are firmly rooted in the old PC-centric paradigm. They can see the train coming down the tracks, they desperately want to get into the new paradigm, but their legacy products and the need to maintain their revenue base are holding them back, and the train keeps getting closer and closer.

We're only partway through…





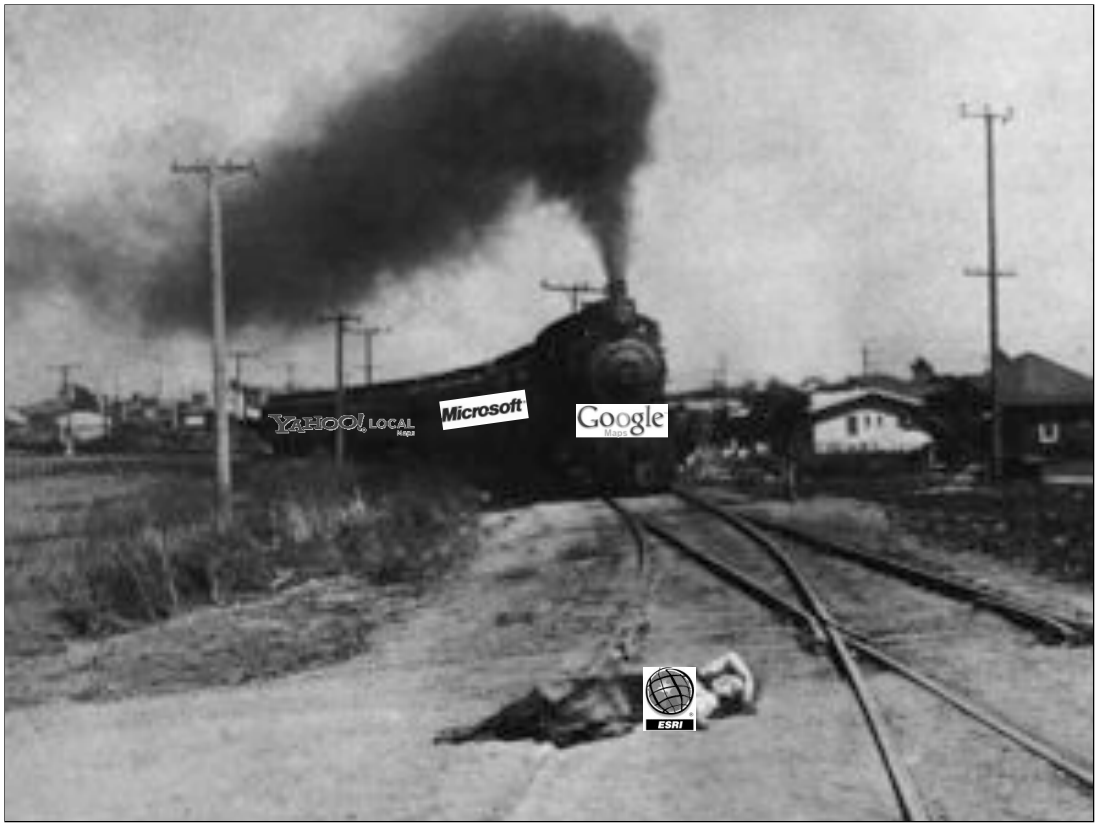
clever elephant

In the geospatial world, ESRI plays the role of Microsoft. Their revenue is rooted in desktop PC software, shrink-wrapped product.

And now the internet is barging into their space.

If Google is the internet-era analogue to Microsoft, who is the internet-era analogue to ESRI?

Oh, it's also Google. And Microsoft. And Yahoo.



So the internet is barreling down the train tracks, and like Microsoft, ESRI is desperately trying to invent some version of itself that can survive the collision.

Meanwhile, the services offered by the GYM providers are perfectly complemented by open source software.

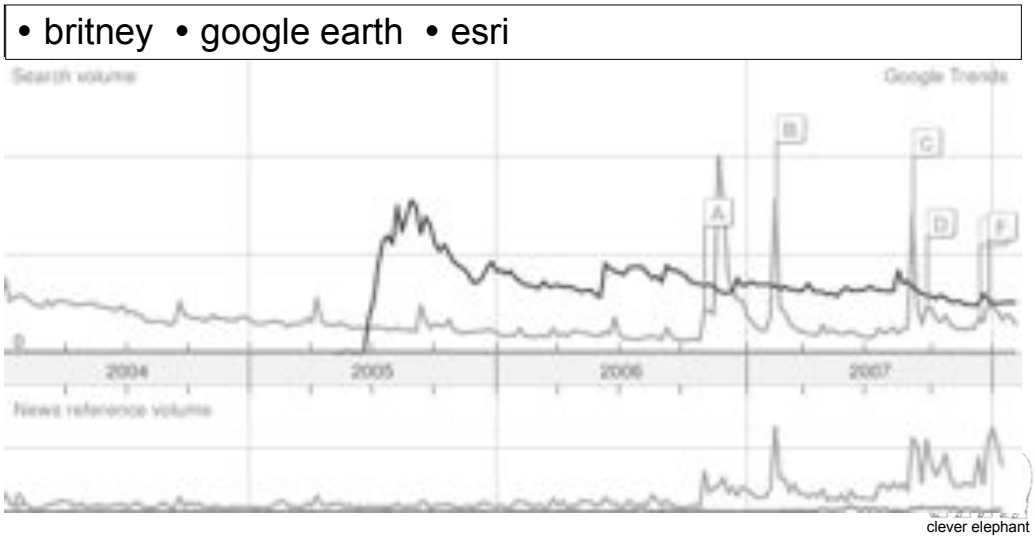
They provide a base map and a user interface, but leave open the questions:

•where do I store my data?

•how do I do a complex spatial query?

"Web mapping isn't 'real GIS'"

- PCs weren't "real computers" either



But remember, we are in the middle of a transition. The future is pretty clear, both as an echo of the past and as a momentum trend.

To give you an idea of the strength of the trend…

•Here's the rate of searching for the term "britney" from 2004 to the present.

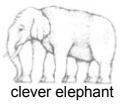
•And here's the rate of searching for "google earth" superimposed on that.

•And here's the rate of searching for "esri" superimposed on that.

In case you're interested, "B" is filing for divorce, "C" is entering rehab, "D" is leaving rehab, "E" is opening the MTV awards (and completely screwing up) and "F" is a rumor about a sex tape.

Open source GIS is Young

- Mapserver was started in 1999
- PostGIS was started in 2001
- Few projects are more than 5 years old
- Already competitive in a wide range of use cases


clever elephant

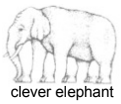
Now, open source GIS feeds into the transition to internet services, and the most successful projects have been those that fit the internet mold: server-side, reliable, simple to configure and work with, modular, scriptable.

The success of open source in that niche has to be evaluated against the relative ages of the projects.

They are all very young.

Web Mapping and Open source

Go together like chocolate and peanut butter!



Web mapping breaks apart applications into components. Rendering, data, querying, user interface. Having all the functionality on one process space (ala, ArcGIS) is not required for web mapping. In fact, it's contra-indicated -- bundled functionality increases complexity and lowers speed.

Web architectures:

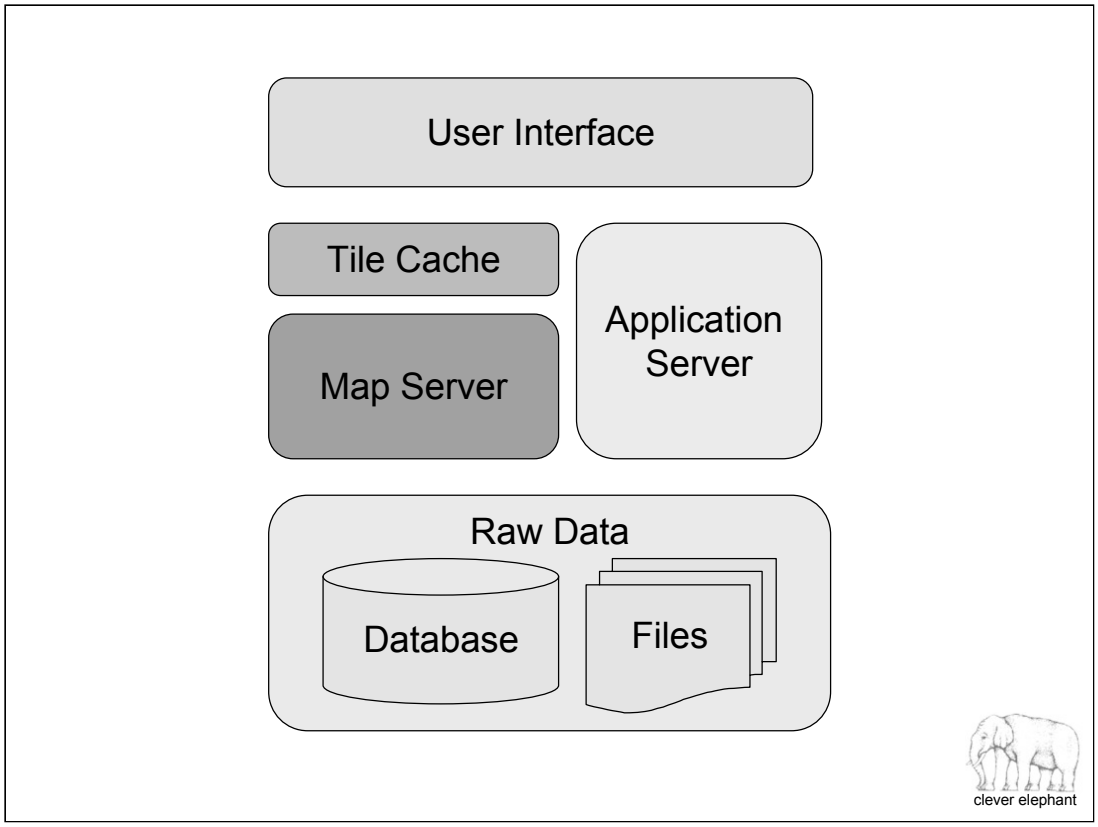
•Break up application function into discrete components

•Optimize each component

•Deploy components separately

•Use different vendors for each component

Open source provides a ready supply of components for use in applications. There is no vendor trying to tie sales together by lowering interoperability with other products. There is no software license cost, so scaling is not an economical problem.

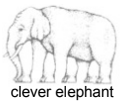


These are the major functional components of a web mapping architecture.

Note that the functional components do **not** map 1:1 to products. In particular, the fuzzy “application server” category can be fulfilled by a wide range of products in different ways.

Raw Data

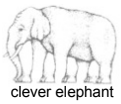
- **Vector Files:** Shape, MIF/MID, DXF, TAB, GeoDatabase
- **Raster Files:** TIFF, SID, ECW, IMG, JPG, PNG, GIF
- **Databases:** PostGIS, Oracle Spatial, SQL Server, ArcSDE, MySQL
- **Web Services:** WMS, WFS


clever elephant

Open source can read natively from a very wide range of raw data sources. The collection of sources supported will vary from product to product, but with the tools we are looking at today, all the sources above can be supported.

Map Server

- Take raw data and output cartographic map. ***Quickly.***
- Inputs:
  - Raw data, styling rules
- Outputs:
  - PNG, GIF, JPEG, KML, SVG, etc

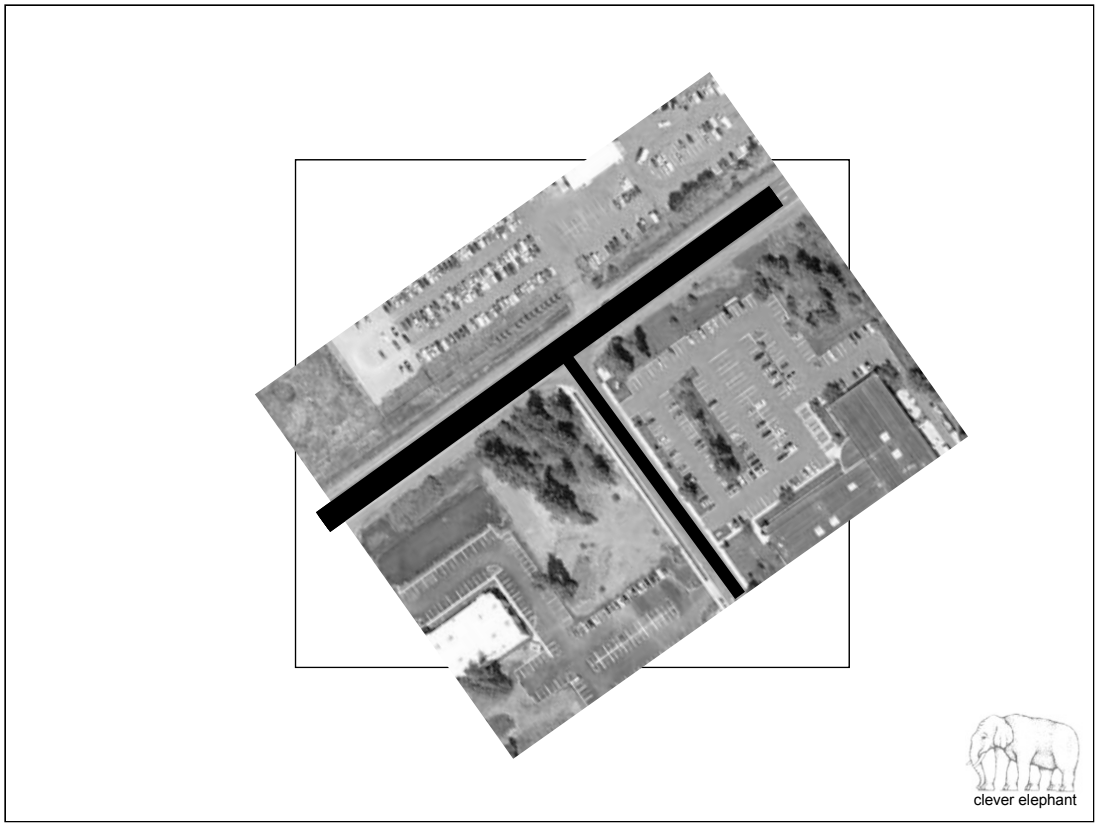

clever elephant

The Map Server is the core of the architecture, taking raw data and making it visually understandable for humans.

We are going to be exploring an open source Map Server, conveniently named “Mapserver”. Because it was originally developed at the University of Minnesota, it is sometimes called the “UMN Mapserver” to distinguish it from the generic term.

ArcIMS is another Map Server you may have heard of. Functionally, it is very similar to UMN Mapserver, though it speaks a different dialect (ArcXML) for configuration and querying than Mapserver.

Both servers speak the standard WMS dialect for map requests, and can be used interchangeably in architectures that hue to the standard.



This is raw data, with a very simple style.

Black, one pixel width.

Now with a color, and some width.

Now a compound style, draw in wide black, with slightly narrower yellow on top, to provide a visual border effect.

Now add labels, using the underlying raw data as a source of information. Each segment has attributes we can leverage for labeling.

Now use the attributes to provide different styles for different kinds of segments.

Now composite multiple layers together into a final output.

These are the kinds of functions

-Colors

-Compound styles

-Labels

-Thematic style rules based on attributes

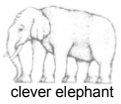
-Layer composition

That are standard in a desktop GIS. They are also the core functionality of any Map Server in a web architecture.

You have to be able to turn raw data into a pretty map on-the-fly.

Map Server

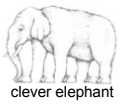
- Arbitrary scale
- Arbitrary extent
- Arbitrary number of layers
- Arbitrary number of output formats
- Arbitrary inputs (format, projection)
- Lots of flexibility
- Requires lots of processing power


clever elephant

Map Servers are expected to take in a lot of complex information about desired scalea spatial extent, process complex information from different formats and projections, and output a single uniform product: an image. Doing this requires a lot of processing power. Your desktop GIS does the same thing, and how long does it take to do a screen refresh? Could it handle two concurrent users? Twenty? Two hundred?

Tile Cache

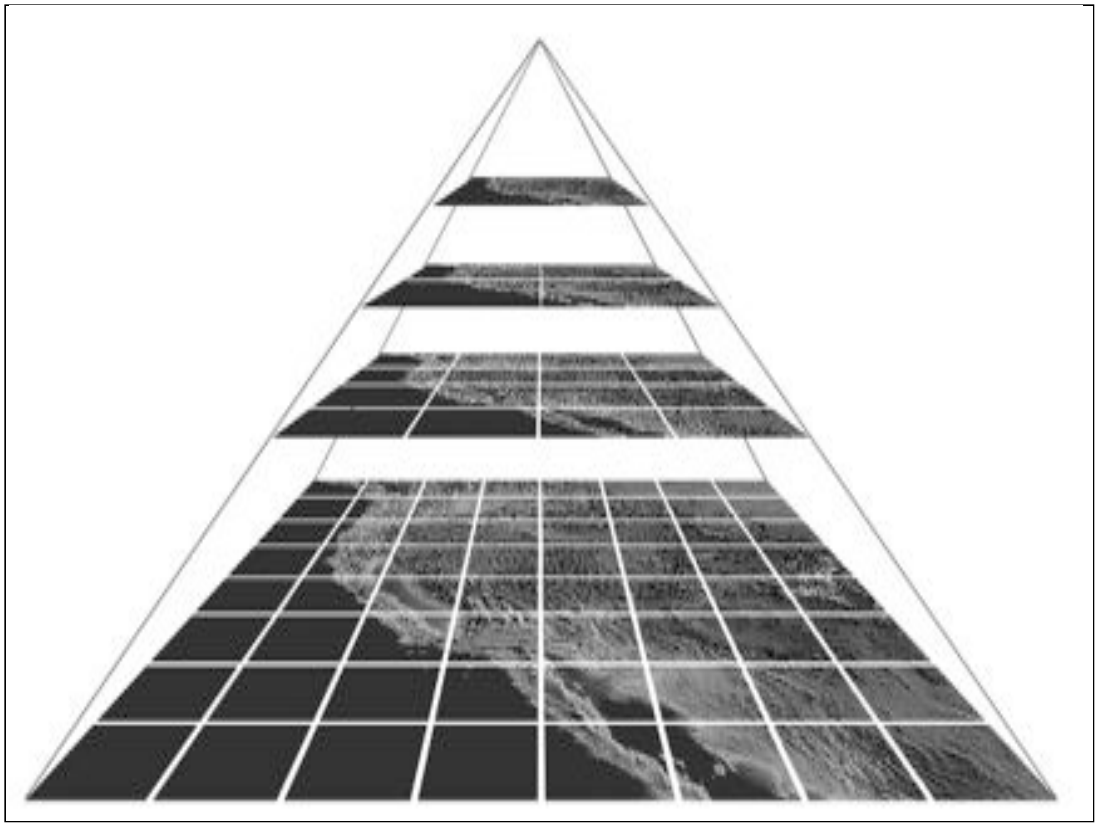
- Tiled maps use fixed scales, *therefore*
- Tiled maps do not have to be rendered on-the-fly, *therefore*
- Tiled maps are faster that on-the-fly maps.
- http://tilecache.org



clever elephant

Tile Caches are added to web mapping architectures to allow large numbers of users to concurrently access the maps without sacrificing performance.

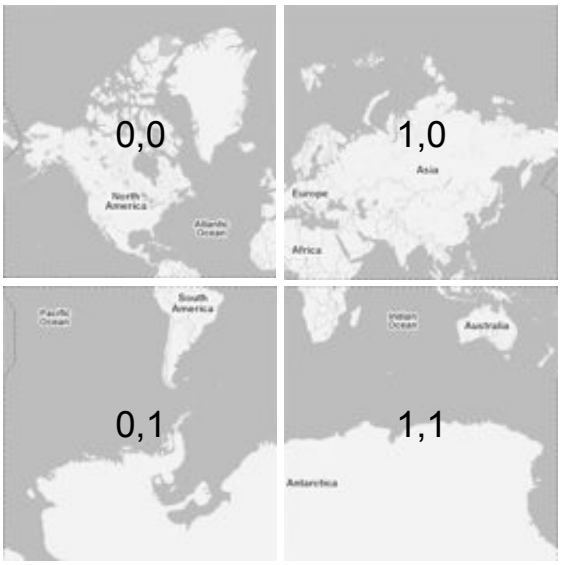
They achieve this magic by throwing away some degrees of freedom: tile caches only allow the application to expose a fixed number of scales; tile caches only allow the application to expose a small finite number of map layers.

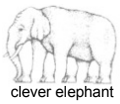


Tile caches are usually (but not necessarily) organized as a nested set of partitions of the plane.

A root area is subdivided into children, which are in turn subdivided, and so on, down to a level which exposes the maximum amount of information available in the source data.

Tile Cache





clever elephant

Let’s explore Google’s tile pyramid:

The Google pyramid consists of 16 zoom levels, and the tiles are numbered with X and Y coordinates starting from an origin at the top left.

This is level 16, which consists of four tiles.

Each tile is 256 by 256 pixels in size.

Tile Cache

- http://mt.google.com/mt?x=0&y=0&zoom=16

If you type in the the magic URL, you can get the image back. The URL encodes the zoom level and the tile coordinates.

Simple, simple, simple.

What if we want a closer view of Greenland?

At zoom level 15, there are four tiles inscribed within the level 16 tile.

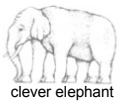
Greenland is at X=1 and Y=0.

What about Africa?

That's at X=2 and Y=1.

Tile Cache

- http://mt.google.com/mt?x=0&y=0&zoom=16
- http://mt.google.com/mt?x=0&y=1&zoom=15
- http://mt.google.com/mt?x=1&y=2&zoom=14
- http://mt.google.com/mt?x=2&y=5&zoom=13
- http://mt.google.com/mt?x=5&y=11&zoom=12
- http://mt.google.com/mt?x=10&y=23&zoom=11
- http://mt.google.com/mt?x=20&y=46&zoom=10
- http://mt.google.com/mt?x=40&y=93&zoom=9



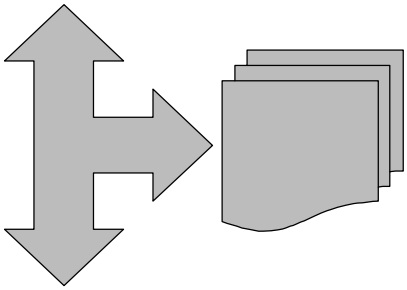
clever elephant

Try out these URLs.

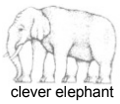
At each zoom level, the number of tiles is larger, as we get closer in to our target (Eugene) the values of X and Y go up, because the tile is farther from the origin in tile coordinates.

Tile Cache

- http://host.com/mt?x=20&y=46&zoom=10



- http://wms.host.com/?request=getmap&bbox=10101,120202,10300,120302&layers=roads,rivers&width=256&height=256



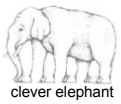
clever elephant

The Tile Cache takes in requests that are asking for tiles, and turns them into requests that a general purposes Map Server can service.

It then stores the result locally for re-use by later requests. The next time that tile request comes in, the Tile Cache responds with its local copy and does not use the Map Server at all.

Application Server

- Answers questions,
- Allows customized processing,
- Runs models.
- Can be as simple as a script, and
- As complex as a whole platform.
- We'll be using scripts.


clever elephant

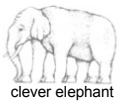
Where does the logic live to do things like "find all the houses in 100m of this address". Or "find the coordinates of this adress". Or "give me a summary of land-use classes in this area".

These are the kinds of questions that web applications provide interfaces to answer, and they have to run in a process somewhere. That process needs to provide access to the data needed to answer the question. It needs to provide a language within which to express the question. And it needs to be able to format the output for the user interface to display.

Those are pretty loose requirements, and they are met by everything from scripting languages to SQL databases to full products like ArcServer and MapGuide.

User Interface

- Allows user to interact with map
- Allows user to query application server
- Map component
  - http://openlayers.org
- Mapping framework
  - http://cartoweb.org
  - http://mapbender.org



clever elephant

Maps require a lot of user interaction to manipulate, and they are visually very rich. Maps are not always the core of the application, often they are ancillary information around a business area.

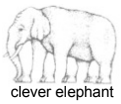
A sales management system might allow you to map where various products where shipped to. But mapping is not the core information component, it's the units of sales that are of interest.

•For organizations adding maps to business applications, a component often works better than a framework.

•For organizations where the map is itself the organizing principle, a mapping framework is sometimes a better approach.

Be cautious about adopting frameworks though -- often GIS people thing the maps are more important than other people do.

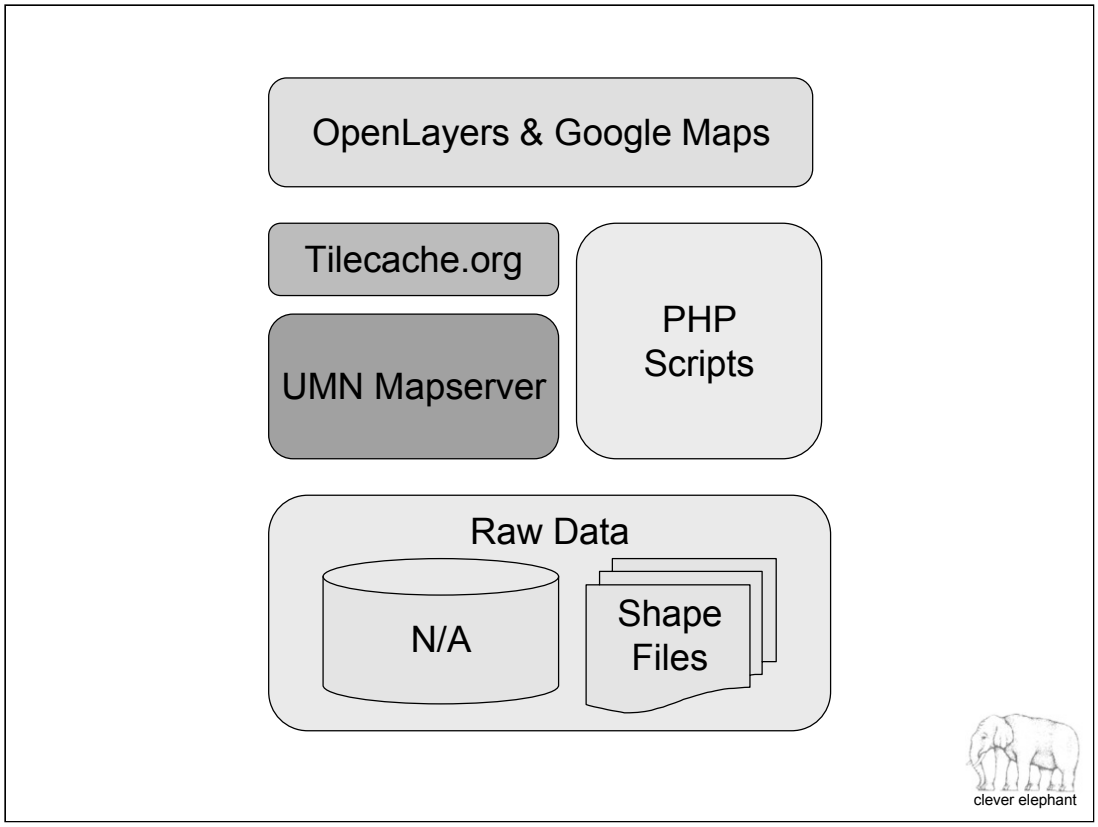
	PostGIS	Mapserver	MapGuide	GeoServer	OpenLayers	ArcIMS	ArcServer	ADF
Raw Data	☑							
Map Server		☑	☑	☑		☑	☑	
Tile Cache			☑	☑			☑	
App. Server	☑	☑	☑	☑		☑	☑	
User Interface			☑		☑			☑



clever elephant

Here is a selection of geospatial products, some open source, and some not, and how they overlay against our functional architecture.

Note that some fulfill multiple functional roles. This is one of the things that makes it hard to compare products -- if they do different things, the comparisons are unfair, unless you qualify what function you are doing the comparison on.

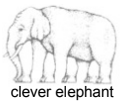


Here is our functional architecture again.

We’re going to be building some web mapping applications, using only open source components (initially) to fill in the boxes in our architecture.

Install DVD

- Copy DVD contents to C:\
- When finished, should have directories
- **c:\ms4w**
  - Mapserver for Windows
- **c:\ms4data**
  - Medford Oregon Data
- **c:\ms4fun**
  - Other Useful Software

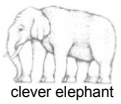

clever elephant

Have you copied your data to C:\ yet?

Now would be a good time… put your hand up if you need a DVD…

Useful Paths

- **c:\ms4w\apps\medford**
  - Where all the example code lives
  - Go here to edit map files
- **http://localhost/medford**
  - The URL that exposes the example code

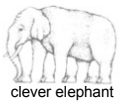

clever elephant

We will be looking at a lot of .map files and .html files. If you want to open them up in an editor and **change** them (please do!) you can find them at the above path.

Once we install ms4w (coming soon) point your web browser at the above address to get the easy-to-browse web version.

Raw Data

- City of Medford Shape Files
- C:\ms4data
- schools, streets, taxlots, buildings, city limits, hydrology, planning, parks, storm drains, wards, wetlands, zoning, imagery (SID)
- Oregon StatePlane South (NAD 83) International Feet



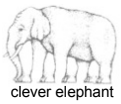
clever elephant

The City of Medford has donated data for this training course. It includes all the usual ingredients of a municipal base map, and we'll be concentrating on publishing this data using open source tools.

Note that the projection is **EPSG:2270**, Oregon State Plane, South Zone, NAD 83, International Feet. We'll be using the EPSG number frequently in configuring other tools, like WMS services.

Map Server

- UMN Mapserver
- http://mapserver.gis.umn.edu
- Multi-format, Multi-protocol
- "Mapserver is **not** a full featured GIS system, nor does it aspire to be. Instead, MapServer excels at rendering spatial data (maps, images, and vector data) for the web."



clever elephant

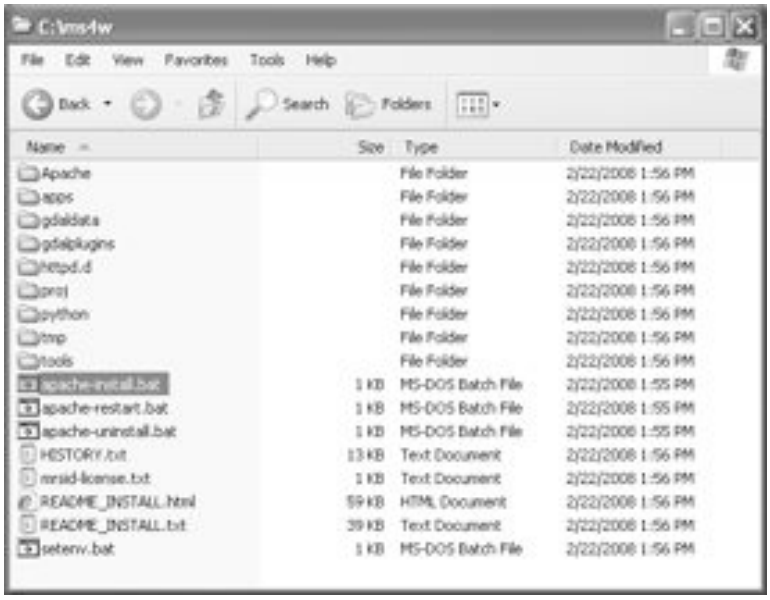
While we are going to be concentrating on rendering shape files in this class, Mapserver supports a wide range of input formats, including: MIF/MID, TAB, DXF, DGN, ArcCoverage, GeoDatabase, PostGIS/PostgreSQL, OracleSpatial, SQLServer Spatial, MySQL, TIFF, SID, ECW, IMG, JPG, PNG, GIF, and many more.

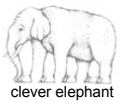
Much of Mapserver's format support is provided by the GDAL/OGR libraries, which Mapserver uses transparently. GDAL provides multi-format raster support, and OGR provides multi-format vector support. GDAL is widely used in proprietary software, including from Google and ESRI, for multi-format raster support. More information about GDAL and OGR is available from http://www.gdal.org/

Mapserver has implemented may output protocols over the years. Originally built with its own CGI interface, which is still around and highly useful, it now also supports Open Geospatial Consortium standards, such as Web Map Server (WMS), Web Feature Server (WFS) and Web Coverage Server (WCS). Output formats include GIF, PNG, JPEG, all GDAL formats, PDF, and Flash.

The front page of the web site includes the technical mission statement, which is often brought forward when considering new feature requests on the developer's mailing list. Unlike some packages, Mapserver does not intend to take over the world, or fulfill every need. In order to remain the best Map rendering engine, the developers have explicitly limited their scope, and will reject feature requests that go beyond that scope. Mapserver provides just enough functionality to fulfill the role of quality fast map rendering. That includes some things like buffers that are occasionally considered "analysis", but avoids other things, like read/write support.

Map Server




clever elephant

MS4W Installation

Unzip into c:\ms4w (done!)

1. Run apache-install.bat
2. Allow Apache network access

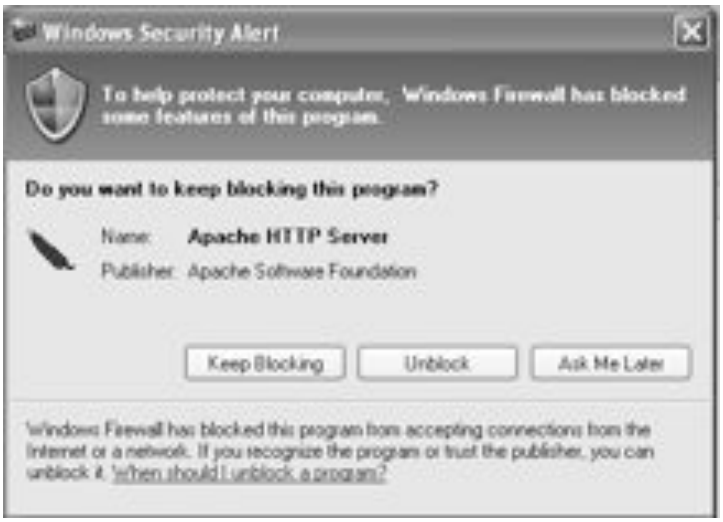
- Gotchas
  - Already running an HTTP server?
  - Super restricted computer?
- Other notes
  - After changing apache configurations (.conf files) run apache-restart.bat
  - The apache-install.bat adds Apache to the Windows registry as an auto-started service.
  - Use apache-uninstall.bat to remove the service entry for Apache.

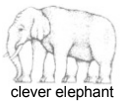
Examples

The examples are in

http://localhost/medford

Map Server




clever elephant

Do not block Apache, or nothing else will work in this lesson!
Select “Unblock”, so that Apache can listen on your web server port.
(Same thing goes for TileCache later on.)

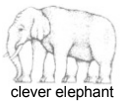
Examples

The examples are in

http://localhost/medford

Map Server (01)

```
MAP
  SHAPEPATH "/ms4data/"
  EXTENT 4180000 130000 4550000 500000
  SIZE 500 500
  LAYER
    DATA cadstreets
    STATUS DEFAULT
    TYPE LINE
    CLASS
      COLOR 0 0 0
    END
  END
END
```



clever elephant

The simplest possible map file, one layer, one color.

The map file structure is a nested tree. It was invented just prior to the introduction of XML, or it might have ended up as an XML format.

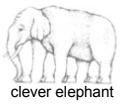
Each element starts with a named token (eg “MAP”, “LAYER”) and ends with “END”.

The most common elements are:

- MAP - the root element
  - LAYER - define map layers within a MAP
    - PROJECTION - defines the projection of the LAYER
    - WEB - defines extra information for web services for a LAYER
    - CLASS - define different kinds of features within a LAYER
      - STYLE - defines a rule for coloring a feature within a CLASS
      - LABEL - defines a rule for labeling a feature within a CLASS or LAYER
  - REFERENCE - defines a reference map for a MAP
  - SCALEBAR - defines a scale bar for a MAP
  - LEGEND - defines a legend for a MAP
  - WEB - defines extra information for web services for a MAP

Mapserver map file reference manual: http://mapserver.gis.umn.edu/docs/reference/mapfile

http://localhost/cgi-bin/mapserv.exe
?map=c:/ms4w/apps/medford/01/simple.map
&mode=map



Web mapping is “cartographic production as remote procedure call”.

The end point here (the procedure name, as it were) is
`http://localhost/cgi-bin/mapserv.exe`

The parameters are “map”, with an argument that tells the procedure what configuration file to process, and “mode”, which tells the procedure to output an image (as opposed to a “legend” or “scalebar”) at the end of processing.

No special software is required to view this map, it is just an image file.

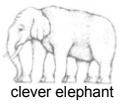
Note how quickly the rendering occurs, this is a 20Mb shape file, with 17K features.

Change the layer you are using from “cadstreets” to “taxlots”, to use a different shape file as an input.

1. Navigate to `c:\ms4w\apps\medford\01`
2. Open `simple.map`
3. Change “`DATA cadstreets`” to “`DATA taxlots`”
4. Save the file, and reload the URL

Map Server

- Spatial indexes make rendering faster
- C:\ms4w\tools\mapserv\shptree.exe
- Builds quadtree spatial index file (.QIX)
- Important for large files


clever elephant

Spatial indexes are required to get fast mapping performance when working with large shape files.

The ESRI SBN/SBX index format is undocumented, so the open source community uses an open quadtree index file, QIX. Mapserver, QGIS and uDig all use the same quadtree index format.

Here are the example steps to create a new index on the taxlots file.

1. Open a command terminal: Run ⇒ `cmd.exe`
2. Type `C:\ms4w\sentenv.bat`
   This puts all the msw4 utility programs on your path.
3. Type `cd \ms4data`
4. Type `shptree taxlots`

Note that there are already indexes on all the large files, but re-running the command will simply over-write them harmlessly.

Map Server (02)

clever elephant

Open up the map file in section 02, `thematic.map`

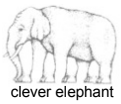
This is the same `cadstreets` shape file, but rendered with a much more intricate map file.

Note, we have multiple colors, multiple widths, and if we zoom in and out, we get features turning off and on.

This map file demonstrates thematic mapping, labeling, and scale dependency.

Map Server (02)

- One **LAYER**, many **CLASS**es
- Map files are processed top to bottom
- Features are checked against **CLASS**es from top to bottom.
- The first **CLASS** a feature matches is the one that it is rendered as.


clever elephant

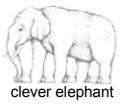
Maps are made from many strokes of the digital brush, and by being aware of the order of operations you can manipulate the look of your map.

Because map files are processed from top to bottom, it is important to put your LAYERs in a good order. If your streams come after your roads, it's possible you will get areas where it looks like the road has been washed out by the stream.

Because classes are processed from top to bottom, if you put a less restrictive class before a more restrictive one, it's possible that you'll never see the more restrictive class on your map. For example, A10 is a highway, and A6-A9 are arterials. If you put an "A*" CLASS before an "A10" CLASS, *all* the features will end up in the "A*" CLASS.

Map Server (02)

```
• CLASS
    EXPRESSION ([TYPE] = 3)
    NAME "Arterial”
    MAXSCALE 60000000
  …
  END
```


clever elephant

The mechanism behind the CLASS is very simple.

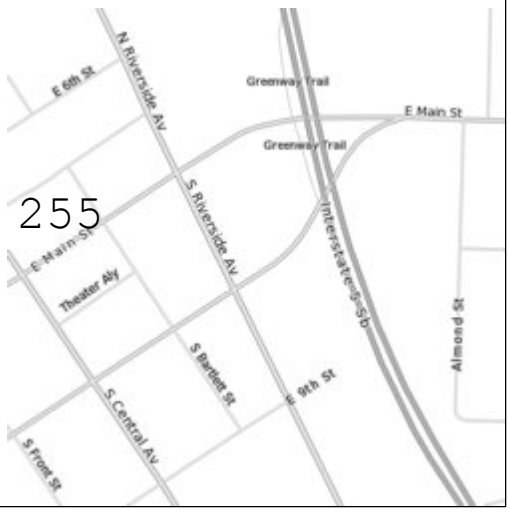
The EXPRESSION is evaluated and if it is true, the feature is passed into that class for rendering. If not, it is passed down to the next class. Note that the final CLASS in the thematic.map file has **no EXPRESSION** so that it catches all the features that failed the previous four CLASSes.

The NAME is the human readable name, and can be seen in the LEGEND to the side of the browsable map template.

The MAXSCALE specifies the largest scale at which the CLASS is active. The scale calculation assumes a 72dpi output device. In general, scales are not accurate in digital media, because the output device resolution is never accurately known.

Map Server (02)

```
• STYLE
    COLOR 180 180 180
    WIDTH 3
  END
  STYLE
    COLOR 255 255 255
    WIDTH 1
  END
```



This pattern shows up repeatedly in our map file. It is the trick to getting “pipe shaped” roads out of map server. Rather than specifying a line with width and edges, “pipes” are created by first laying down the edges color in a wide swath, then drawing the interior color in a slightly narrow swath right on top. The final effect is the “Google-style” wide pipe roads.

Map Server (02)

```
• LABEL
    COLOR 0 0 0
    OUTLINECOLOR 235 235 235
    MINFEATURESIZE 10
    TYPE truetype
    FONT "vera"
    POSITION AUTO
    ANGLE FOLLOW
    PARTIALS FALSE
    MINDISTANCE 250
    BUFFER 15
    SIZE 10
  END
```

There are a lot of dials and knobs on the Mapserver labeling code, because it can be used so many ways.

COLOR and OUTLINECOLOR are fairly obvious.

MINFEATURESIZE gives the smallest feature (in pixels) that will be labeled.

FONT and TYPE specify that we are using a TrueType font and the font name we want. Note the FONTSET directive higher up in the MAP object, that specifies where the TrueType font collection is stored.

POSITION AUTO places the label so as to minimize collisions.

ANGLE FOLLOW strings the label along the geometry of the feature. Only good for linear features.

PARTIALS FALSE avoids any labels that are cut off by the edge of the image.

MINDISTANCE specifies the minimum distance (in pixels) that must be between any two labels that have the same string.

BUFFER specifies the minimum distance (in pixels) that must between labels.

SIZE specifies the font size in points.

Map Server (02)

Random | Ordered by TYPE

What is the difference between the picture on the left and the picture on the right?

They are both using identical styling rules in the map file.

They are both showing identical spatial extents.

They are both at the same scale.

But the labels on the left are awful and the labels on the right are acceptable.

What gives?!?

The data on the right as been pre-ordered, using the TYPE field, the same field used to color the street according to their "importance".

By getting the "important" labels drawn first, we allow the label collision algorithm to do the "right thing", winnowing out "less important" labels.

When the data is in random order, it is just as likely that an unimportant label will get into the label cache first, thereby filtering out late-arriving important labels.

1. Run ⇒ cmd.exe
2. Type `C:\ms4v\setenv.bat`
3. Type `cd \ms4data`
4. Sort the data, type `sortshp cadstreets cadstreets_sorted type descending`
5. Index the new file, type `shptree cadstreets_sorted`
6. Navigate to `\ms4w\apps\medford\02`
7. Open the map file `thematic.map`
8. Change "`DATA cadstreets`" to "`DATA cadstreets_sorted`"

Note that we sort cadstreets in **descending** order, the exact opposite of what we should want. This is because the index will cause the features to be drawn in reverse file order… if we didn't have an index, we would sort in the "proper" order. Yes, this is an absurd situation, but it works.

Map Server (02)

Whole Pixel | Sub-Pixel

What is the difference between the picture on the left and the picture on the right?

They are both using identical styling rules in the map file.

They are both showing identical spatial extents.

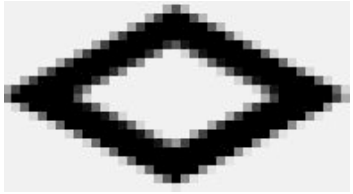
They are both at the same scale

The one on the right was rendered with an old-style whole pixel renderer and the one on the left with a sub-pixel “anti-aliased” renderer.

The old renderer is called “GD” and the new renderer is called “AGG”. The names are meaningless.

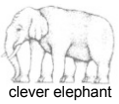
The important thing is that one looks better, but it is about 50% slower to draw the same pictures.

Map Server (02)



- OUTPUTFORMAT
  NAME "PNG"
  DRIVER **"AGG/PNG"**
  IMAGEMODE RGB
  MIMETYPE "image/png"
  EXTENSION "png"
  TRANSPARENT OFF
  FORMATOPTION "INTERLACE=OFF"
  END

- OUTPUTFORMAT
  NAME "GIF"
  DRIVER **"GD/GIF"**
  IMAGEMODE PC256
  MIMETYPE "image/gif"
  EXTENSION "gif"
  END



The OUTPUTFORMAT directives at the top of the thematic.map file are what control the two drawing modes demonstrated on the last slide. Which mode is actually used is controlled by the IMAGETYPE line at the top of the file.

The older GD graphics library originally used by Mapserver does not support anti-aliased line drawing.

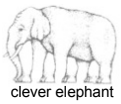
The Mapserver graphics output subsystem was upgraded recently to allow different graphics output drivers. Any format GDAL can write, can now be produced by Mapserver.

In addition, the drawing routines were modularized, so that alternate graphics libraries might be used, opening the door for the new AGG library to be plugged in along side GD. AGG is slower than GD, in general, but produces much higher quality graphical output.

When we are caching our maps, using a Tile Cache, the speed of rendering will become less important, and an acceptable trade-off for higher quality output.

Map Server (02)

- The Rules
  - Only draw what you need to
    - Not too many features
    - Not too many layers
  - A label is worth 1000 lines
    - Place names trump road names
  - Test every scale
  - Stand on the shoulders of giants
    - Copy existing styles
      - National Geographic
      - Google



clever elephant

GIS experts and other cartography whizzes will look at web map styling and think they can whip off a map before lunch.

But they forget: it is not **one** map, it is **sixteen**, one for every effective scale.

And at the lowest scales the amount of raw data is very large. Visually reviewing the **whole map** is well-nigh impossible. The result is that even highly professional multi-scale mapping exercises like Google's will have occasional blunders where the algorithm and the data come together to create… confusion.

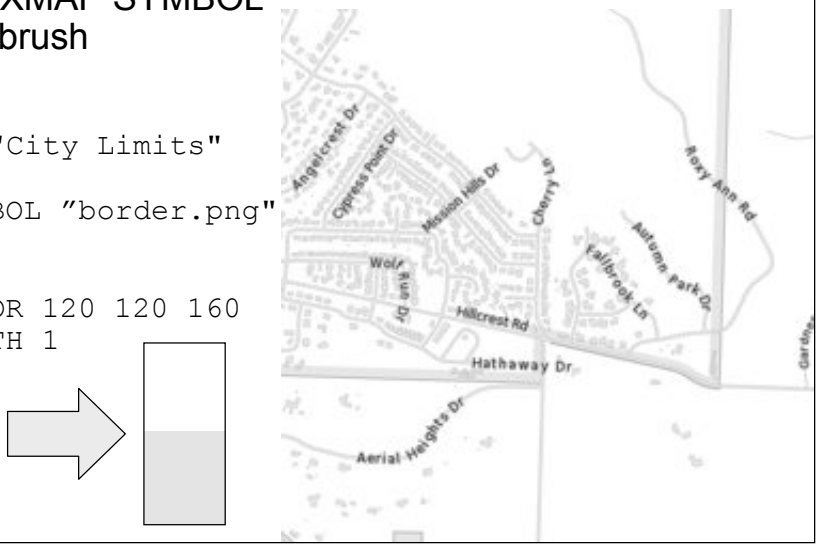
Remember that you are probably not a cartography expert, but that it is cheap and easy to copy people who are. Use familiar styles, like the ones already in wide use on the GeoWeb, or ones that people are familiar with from atlases. (Notice how the new Google "Terrain" style is very similar to an atlas look and feel.)

Context is more important than polygons, and labeling, in particular place names, provide the user with context. A GNIS file, sorted by population order, can provide an easy way of ensuring that the larger population centers are always labeled on your map.

Map Server (03)

- City limits layer: “National Geographic border”
- Use a PIXMAP SYMBOL as paint brush

```
CLASS
  NAME "City Limits"
  STYLE
    SYMBOL ”border.png"
  END
  STYLE
    COLOR 120 120 160
    WIDTH 1
  END
END
```



The wms.map file is much much much much bigger than our previous files. Our previous examples only had one layer, wms.map has multiple layers. It also has an imagery layer, and some new symbolization examples. Before discussion the WMS standard, here is a quick overview of some of the symbolization tricks in the wms.map file.

The municipal boundary of Medford has what I call the “National Geographic border” look. A wide line denoting the interior of the boundary, with a sharp dividing line on the actual border. I have been wanting to reproduce this effect for a long time, and only figured it out finally while preparing for this workshop.

By using a rectangular image brush, with one half of the brush transparent, and the other half the desired color, you can get the “broad line to one side” effect. The second style just adds the thin delineating line on top in a darker color.

Note that this will only produce the right effect if the line work is consistent in its direction.

In the case of the Medford boundary, the input is actually polygonal, but by choosing to render it with the LINE type, we get a linear rendering. We are also guaranteed that the direction will be correct, because the polygon must have consistent edge direction.

Map Server (03)

- Schools LAYER: “colorful symbols”
- PIXMAP SYMBOL for points

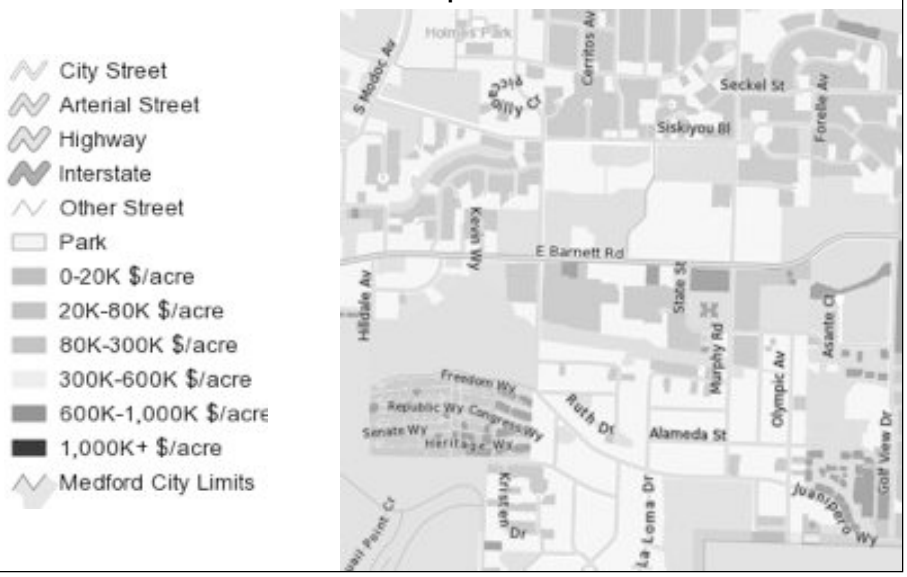
```
CLASS
  EXPRESSION "High School"
  NAME "High School"
  STYLE
    SYMBOL "button_0.png"
  END
END
```

Symbology for points can be difficult, because so often one colored ellipse just starts to blend into another. Using cartographic symbols or images can make points really pop off a map. For things like adding city markers and other forms of contextual labeling, different symbols can make a map much more readable.

Any GIF or PNG image can be used as a PIXMAP symbol.

Map Server (03)

- Tax Lots LAYER: “thematic map on land value”



The tax lots valuation layer is thematic, based on a **computed value**. The raw tax lots shape file has a LANDVALUE field, and a couple area fields (ACREAGE and SHAPE_AREA) but now value/area summary field. So we use the Mapserver EXPRESSION syntax to calculate the value on the fly. Because the ACREAGE field had limited precision, I calculated the dollars value per square foot, then multiplied that up (43560 feet to the acre!) to get the dollar value per acre.

```
CLASS
  NAME "20K-80K $/acre"
  EXPRESSION ( 20000 < 43560*[LANDVALUE]/[SHAPE_AREA] AND
               80000 >= 43560*[LANDVALUE]/[SHAPE_AREA] )
  STYLE
    COLOR 190 175 212
  END
END
```

This is the class definition I would have used for one of the categories, except…

I FOUND A BUG!

DIGRESSION

- When you find a bug
  - Prove to yourself that it is a bug
  - Create the simplest situation that displays the bug
  - Show your case to someone who might know better
  - File a bug report including your simple case
  - Even if you find a work around

My bug was that the correct expression syntax was not returning the right answer. This always returned false:

```
EXPRESSION (20000 < 43560*[LANDVALUE]/[SHAPE_AREA] and
43560*[LANDVALUE]/[SHAPE_AREA] <= 80000)
```

Stranger still, this expression worked:

```
EXPRESSION (1000000 < 43560*[LANDVALUE]/[SHAPE_AREA])
```

Practically the same thing. However, this expression did not work:

```
EXPRESSION (3>2/1 and 2/2<6)
```

Even though it was demonstrably true. It wasn't failing because it had more than one logical term. This worked:

```
EXPRESSION (3>2*1 and 2*2<6)
```

I tried a number of other combinations, and shared them on the #mapserver IRC channel. One of the developers noted that the "two terms with division" case looked a lot like a regular expression (which also has two / symbols) and that might be confusing the parser. So I hit upon an ugly idea.

The Mapserver EXPRESSION syntax includes support for exponents. And for negative numbers. Would it support negative exponents? It did. So I rewrote all my EXPRESSIONS to use negative exponentiation instead of division, and the rest is on the map.

```
EXPRESSION (20000 < 43560*[LANDVALUE]*[SHAPE_AREA]^-1 and
43560*[LANDVALUE]*[SHAPE_AREA]^-1 <= 80000)
```

The bug is filed at http://trac.osgeo.org/mapserver/ticket/2527

This is what my bug looks like.

http://trac.osgeo.org/mapserver/ticket/2527

Note that the description includes the very simple directions for exercising the bug.

It has been assigned to Steve Lime, the father of Mapserver (and the resident expert on the map file parsing code, which is where this bug lives).

If you go to the URL for the bug, you'll see the developers discussing various means of circumventing it. As usual in engineering, every approach has trade-offs. The backwards compatible invisible fix will create complexity in the code base, the clean easy-to-code fix will break old expressions. Mapserver has a better history than most projects of preserving backwards compatibility between releases. This means that old .map files work on new Mapserver releases.

The only downside to this philosophy is that when improvements in the map file syntax come along, you might not learn them and go on using your old ugly methods. I learned of a number of useful improvements to map file syntax while preparing this workshop.

02/28/08 09:47:06 changed by dmorissette — Reply

I'd have a preference for keeping the // syntax if we can, not only for backwards compatibility, but also for expression readability.

i.e. this

```
EXPRESSION ([name] =~ /regex_here/)
```

is much more intuitive than this

```
EXPRESSION ([name] =~ 'regex_here')
```

Is the parser sequential or does it have the concept of priority of tokens? Could we make it so that if the division are enclosed inside brackets like this

```
EXPRESSION (3>(2/1) and (2/2)<6)
```

then the divisions are parsed as single entities before the // are processed as a regex?

03/01/08 13:11:13 changed by sdlime — Reply

Actually I think I can set a lexer state so that // means regex only after first seeing =~. Otherwise /'s would be seeing as division operators. Trying that now...

Steve

03/02/08 09:50:44 changed by sdlime — Reply

- **status** changed from *assigned* to *closed*.
- **resolution** set to *fixed*.

Fixed in both the 5.0 branch (r7421) and the dev branch (r7420). Closing...

Steve

And here is how the story ends.

Note the date. I reported the issue on 02/26/08.

And this is just background bug fixing. I did not have to pay for this support or the speed. Had I been a paying customer of one of the developers I could probably have gotten a fix in even **less** time.

Daniel Morissette runs a company in Chicoutimi, Quebec called MapGears that exclusively does Mapserver work.

Steve Lime works for the Minnesota DNR, but does do some moon lighting, from time to time.

Map Server (03)

- Imagery LAYER: “Huge MrSID file”

Mapserver must be compiled with support for MrSID in order to read that format, but fortunately MS4W includes that support out-of-the-box, so using the big SID images provided by Medford was trivial. The below is the entirety of the LAYER definition needed to bring in the 2 gigabyte MrSID file.

```
LAYER
  NAME imagery
  METADATA
    "wms_title" "Imagery"
  END
  PROJECTION
    "init=epsg:2270"
  END
  TYPE RASTER
  DATA Medford_2007_mosaic_10to1.sid
  STATUS ON
  #PROCESSING "RESAMPLE=AVERAGE"
END # End of imagery!
```

Can you spot the difference between the image on the left and the one on the right? The one on the left has been resampled using “nearest neighbor”, a quick’n’dirty technique that just finds the closest target pixel from the source raster. The one on the right has been resampled using the “average” filter, which averages the neighboring pixels to find a value for a target pixel. Average is more expensive that nearest neighbor, by a factor of around 2:1, but results in smoother looking outputs. When using a caching system, it might be best to opt for the expensive technique.

Just uncomment the PROCESSING line to enable average filtering. Nearest neighbor is the default method.

OGC
Open GIS Consortium, Inc.

- Web Map Server (WMS)
  - Open Geospatial Consortium
  - 2000, Version 1.0
  - 2003, Version 1.3
  - 2007, Tiled WMS Discussion Paper
- Standard URL pattern for requesting maps

Like Rodney Dangerfield, the WMS specification can't get no respect, particularly since tile based maps have become ascendant. However, WMS remains a core building block for geospatial architecture, because it allows components to communicate using a standard dialect for map requests.

Among the products that support consuming WMS services are:

•ArcGIS
•MapInfo
•Geomedia
•Google Earth
•ArcExplorer
•Worldwind
•uDig
•QGIS
•Mapserver

Hey, isn't Mapserver a **server**?? Yes, but a WMS service can be thought of as a raw data source, where the data type is imagery. So a map server can use a WMS service as an input, and combine other data with it, then publish the result as… another WMS service. This is called "service chaining".

Among the (many, many) products that support serving WMS are:

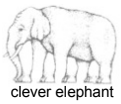
•ArcIMS
•ArcServer
•Mapserver
•MapGuide
•GeoServer

The WMS specification is available online at http://www.opengeospatial.org/standards/wms

Mapserver WMS configuration guide is at http://mapserver.gis.umn.edu/docs/howto/wms_server/

Map Server (03)

- http://localhost/cgi-bin/mapserv.exe?map=/ms4w/apps/medford/03/wms.map&
  SERVICE=WMS&
  VERSION=1.1.1&
  REQUEST=GetCapabilities
- http://localhost/cgi-bin/mapserv.exe?map=/ms4w/apps/medford/03/wms.map&
  SERVICE=WMS&
  VERSION=1.1.1&
  REQUEST=GetMap&
  SRS, BBOX, LAYERS, FORMAT, HEIGHT, WIDTH, TRANSPARENT, STYLES


clever elephant

There are two frequently used WMS modes: GetCapabilities, and GetMap.

GetCapabilities returns an XML document describing the layers available from the service.

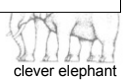
Here is a full WMS GetMap request:

```
http://localhost/cgi-bin/mapserv.exe?
map=/ms4w/apps/medford/03/wms.map&
SERVICE=WMS&
REQUEST=GetMap&
SRS=EPSG:2270&
LAYERS=buildings&
BBOX=4279981,255620,4283080,257106&
EXCEPTIONS=application/vnd.ogc.se_xml&
FORMAT=image/jpeg&
HEIGHT=358&
WIDTH=747&
TRANSPARENT=FALSE&
STYLES=default&
VERSION=1.1.1
```

It seems byzantine. Why go to the trouble? Because configuring your map service to speak WMS is not that hard, and opens up your service to 100s of pieces of client software that **you do not have to write or provide**. Your WMS service can be read directly by ArcGIS, by Google Earth, by many many web frameworks.

Try out the sample GetCapabilities and GetMap URLs in section 03 of the web site and see what they return.

```
<GetMap>
    <Format>image/png; mode=24bit</Format>
    <Layer queryable="0" opaque="0" cascaded="0">
        <Name>citylimits</Name>
        <Title>Medford City Limits</Title>
        <SRS>EPSG:2270</SRS>
        <LatLonBoundingBox
                minx="-122.914" miny="42.287"
                maxx="-122.775" maxy="42.4006" />
        <BoundingBox SRS="EPSG:2270"
                minx="4.26928e+006" miny="235477"
                maxx="4.30572e+006" maxy="275849" />
    </Layer>
  </GetMap>
```



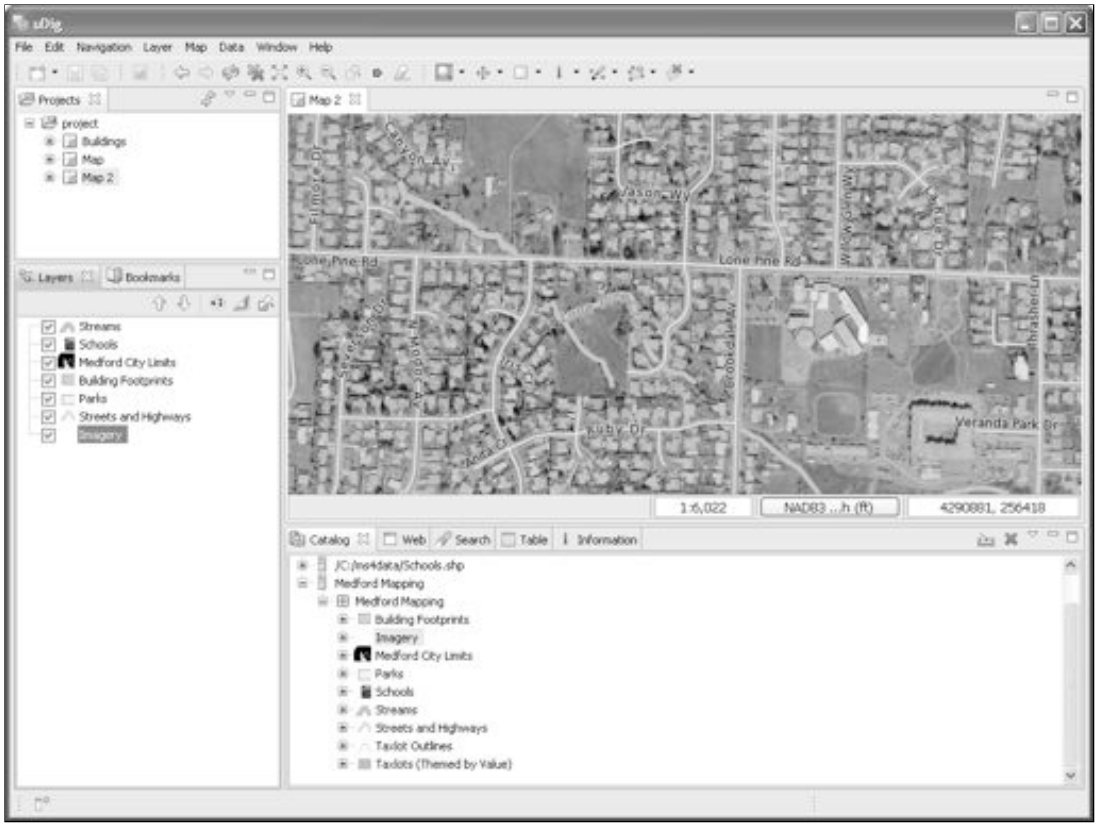
clever elephant

The capabilities document is very rich in metadata. Open the capabilities link to see the whole document. The best thing is to save it as capabilities.xml then re-open it in IE to see it formatted nicely as XML.

At the <Service> level are descriptions of the server itself, optionally contact information for the server operator, the organization, the data provider and others, as well as room for full descriptions.

The next major section is <Request> and it tells all the different requests our server honors. I mentioned GetMap and GetCapabilties. Read through the full file, and you'll see Mapserver also supports GetFeatureInfo (for query tools), GetLegendGraphic (for accurate legends, see the uDig legend view of a WMS for example), GetStyles (for clients that support on-the-fly restyling of the service) .

Finally, the server lists all the <Layer>s available, in a nested tree of layers. We have organized our server with a flat layout, but we could have used the Mapserver GROUP keyword to organize our layers into folders. Each <Layer> has a <Title>, extent information, and information about available spatial reference systems (SRS).



Here are a couple WMS client examples.

I found this Flash WMS viewer on the web while preparing this workshop, and got it up and running in about 20 minutes. Because my server is a WMS standard server, there was nothing to configure on the server side, just had to point the viewer to the right URL. Try it out, it's available in section 03 of the web-based content. Or hit the URL directly:

http://localhost/medford/03/flamingo/minimal.html

Heavy-weight applications can consume WMS services too. Anything that speaks HTTP (and that is **everything**) can consume WMS services. The uDig desktop client can consume WMS. There is a copy of uDig in the C:\ms4fun directory that came with the workshop materials. Install it, and then use Layer ⇒ Add…, select "Web Map Server", and then type in the capabilities URL of our demonstration server:

http://localhost/cgi-bin/mapserv.exe?map=/ms4w/apps/medford/03/wms.map

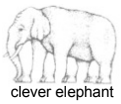
Choose the layers you want to see and press finish.

You can also add shape files to the view, with Layer ⇒ Add or just by dragging them in from the explorer, to see how the data lines up.

And of course both OpenLayers and TileCache can be used as WMS clients, as we will be seeing later. That means that you can build an OpenLayer/TileCache user interface on top of any server than speaks WMS, including options like ArcIMS, ArcServer and MapGuide.

Map Server (03)

- WMS-enabling Mapserver
  - Explicit PROJECTION on MAP and LAYER
  - METADATA on MAP
    - wms_title
    - wms_abstract
  - METADATA on LAYER
    - wms_title
    - wms_abstract
- mapserver.gis.umn.edu/docs/howto/wms_server
  - 100s of special metadata keywords


clever elephant

Most of the extra configuration needed to make a standard Mapserver .map file into a WMS-ready map file is about adding metadata to the service.

The .map file includes layer names (in the NAME field) but the WMS specification also requires human-readable **titles**, so the "wms_title" metadata must be added to every LAYER.

```
LAYER
  NAME imagery
  METADATA
    "wms_title" "Imagery"
  END
  PROJECTION
    "init=epsg:2270"
  END
  …
END
```

In addition to the metadata, every layer must state the input projection of the data. This allows Mapserver to re-project the data on the fly to a common output projection. WMS services are all **explicit** about what projection their maps are coming out in, so that client software can make intelligent choices about how to overlay it with other data.

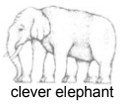
There needs to be a little extra metadata at the MAP level as well. As with the layer, a "wms_title" and "wms_description" are needed, and an explicit default output projection needs to be set.

```
PROJECTION
  "init=epsg:2270"
END
WEB
  METADATA
    "wms_title" "Medford Mapping"
    "wms_abstract" "Demonstration data assembled for a URISA event. Extracted
from the Medford SDE into shape format."
    "wms_srs" "EPSG:4326 EPSG:900913 EPSG:26910 EPSG:2270 EPSG:2839 EPSG:2914
EPSG:2991 EPSG:2992 EPSG:3643 EPSG:3644 EPSG:3647 EPSG:3648 EPSG:32037
EPSG:32127"
  END
END
```

Note that the server advertises all the SRS outputs it supports.

User Interface

- Any WMS user interface will do
  - www.mapbender.org
  - www.cartoweb.org
- We use the JavaScript tiled interface
  - www.openlayers.org


clever elephant

Because the map service we have configured uses WMS, any user interface framework that supports WMS can be used to build the user-facing side of the application.

We already saw one example, in the Flamingo Flash client.

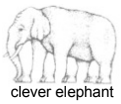
The Mapbender and Cartoweb frameworks are from Germany and France respectively, and each backed by a strong regional open source company. The Wheregroup (www.wheregroup.com) backs Mapbender and Camptocamp (www.camptocamp.com) backs Cartoweb. Both frameworks require PHP on the server, and some extra configuration.

We are looking for a Really Simple client, and Openlayers fits the bill. It requires no server components, it can be used just by invoking Javascript on a web page. This makes it very light to deploy, but building complex applications may require a lot of code. The larger frameworks, like Cartoweb (version 4), are moving to embed Openlayers as their mapping component and providing extra application building tools to make it easier to build apps.

If we were building using MapGuide Open source as our Map Server, we might use the Fusion framework that is being bundled with it in upcoming releases. Like CartoWeb4, Fusion uses OpenLayers as an embedded component and adds extra tooling for building complex sites. Fusion is tightly tied to MapGuide, unlike the more generic Mapbender and Cartoweb.

OpenLayers

- No server side components required
- Pure JavaScript, object oriented
- Tile-based maps
- Multiple tile sources
- Component architecture
- Do one thing, and do it well


clever elephant

OpenLayers was built by Metacarta, a company whose core business is geoparsing -- reading through piles of textual documents and intelligently extracting the location references from the text. Once you have extracted location information from a text the obvious next step is to show those references on a map. Metacarta wanted a modern looking user interface they could include in their search appliance, but without the restrictions imposed by the Google Maps interface in terms of cost and deployment locations (Metacarta sells to a lot of intelligence agencies, who run in disconnected internal networks).

So they built their own. The result has been so good that it is taking over from every other client-side web map component.

In addition to simple browsing, capabilities for drawing markers, drawing polygons and lines, querying feature sources like simple WFS servers, doing animations, digitizing tools, have been added to the framework.

When you start with OpenLayers you will find that it's strength is its weakness: it is built to be very small and simple for a default case, global browsing of tile-based maps. When you deviate from the simple case, you have to do more research to figure out the implications of your design. So it is only a few lines of code to publish a world map in lat/lon. It takes several more to get a map in planar coordinates.

The component architecture of OpenLayers is visible in the tile source design: OpenLayers can consume from multiple tile sources (WMS, Google, Microsoft, Yahoo) and all the tile sources are extensions of the OpenLayers.Layer class.

The documentation for OpenLayers, in terms of tutorials and guides is very thin. It is balanced by an extremely rich set of examples, in the /examples directory of the source distribution, and also online, here: http://www.openlayers.org/dev/doc/examples.html

OpenLayers (04)

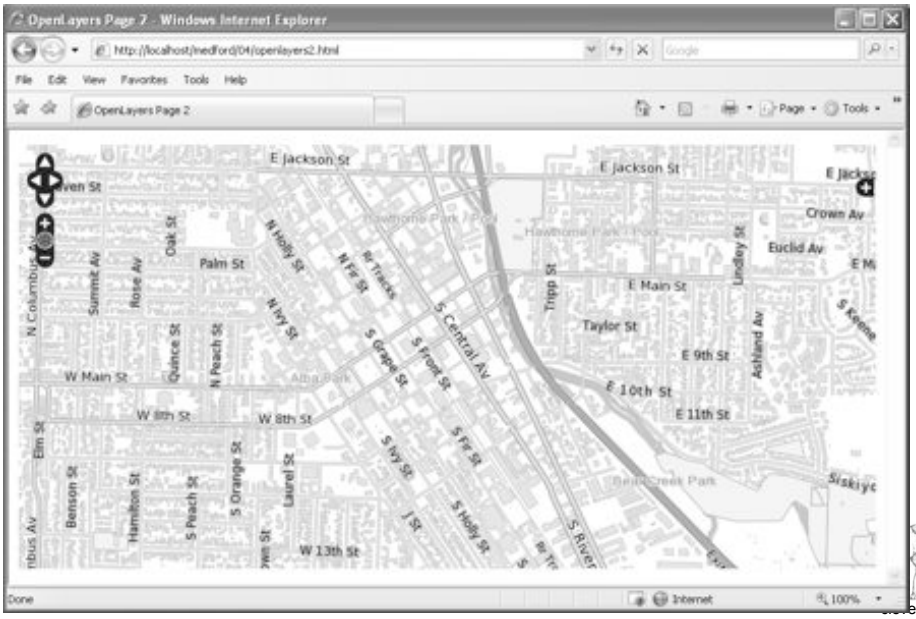
clever elephant

Here is OpenLayers in action on the simplest possible file, reproduced below in its entirety (see section 04 example 1):

```
<html>
<head><title>OpenLayers Page 1</title></head>
<body>

<div id="map"></div>

<script type="text/javascript"
src="http://www.openlayers.org/api/2.5/OpenLayers.js"></script>

<script type="text/javascript">
  var map = new OpenLayers.Map('map', {'maxResolution': 1.40625/2});
  var ol_wms = new OpenLayers.Layer.WMS( "World Map",
               "http://labs.metacarta.com/wms-c/Basic.py?",
               {layers: 'basic', format: 'image/png' } );
  map.addLayers([ol_wms]);
  map.zoomToMaxExtent();
</script>

</body>
</html>
```

Note that OpenLayers is so devoted to the “no install” principle, that it is actually possible to build an open layers map (as we have just done) without putting the OpenLayers source code on your computer! This code references a copy of OpenLayers residing on the openlayers.org site. By using remote WMS services and remote code, you’ve built a web application that requires nothing more than an HTML page.

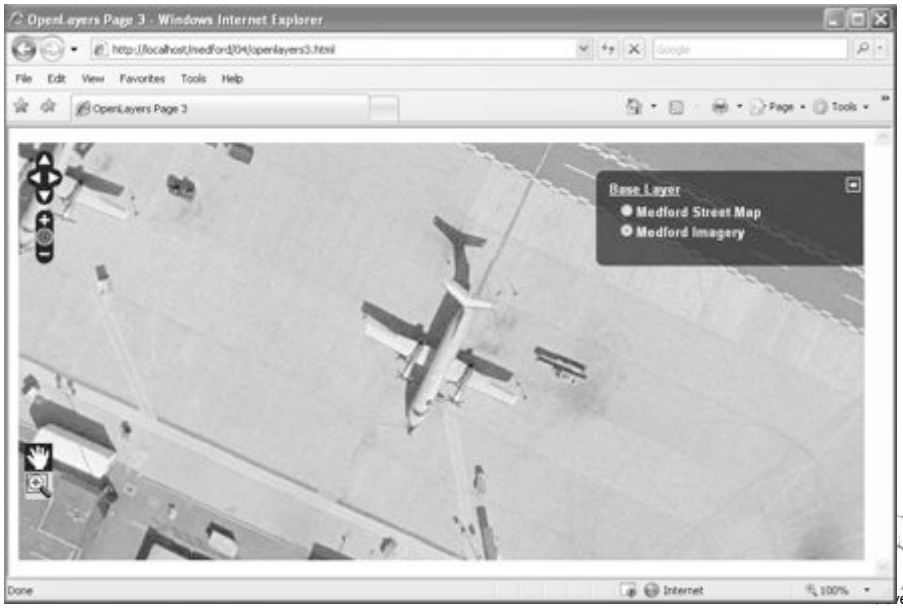
OpenLayers (04)



Now let's get our Medford data into OpenLayers (see section 04 example 2). Because OpenLayers supports WMS as an input, this is relatively simple. We are working against OpenLayers "lat/lon is the default" assumption here, so it will require a few more configuration parameters. The complete examples that come with OpenLayers come in handy: this is just a modified version of examples/projected-map.html. Note that this time we are using a local copy of OpenLayers, so we are not dependent on an internet connection.

```
<div id="map"></div>
<script type="text/javascript" src="../OpenLayers-2.5/OpenLayers.js"></script>
<script type="text/javascript">
  var center=new OpenLayers.LonLat(4280000, 249000)
  var zoom = 1;
  var map = new OpenLayers.Map( 'map' );
  var basemap = new OpenLayers.Layer.WMS( "Medford",
   "http://localhost/cgi-bin/mapserv.exe?",
    { map: '/ms4w/apps/medford/03/wms.map',
      layers: 'hydrology,streets,parks,buildings,schools',
      format: 'png',
      transparent: 'off' },
      // These are the important parts for creating a non-epsg:4326
      // map: Maxextent is the boundary of the map/tile loading area,
      // maxResolution is the units/pixel at the highest zoom.
      {
       maxExtent: new OpenLayers.Bounds(4171956,142812,4389715,355721),
       maxResolution: 100,
       projection:"EPSG:2270",  // Used in WMS/WFS requests.
       units: "ft"              // Only necessary for working with scales.
      } );
  map.addLayer(basemap);
  map.setCenter(center, zoom);
  map.addControl(new OpenLayers.Control.LayerSwitcher());
</script>
```

OpenLayers (04)



Finally, let's get our imagery into OpenLayers. Rather than just replace the vector base map, we'll add a second base map. We'll also add a rubber band zoom tool, so us GIS folks can feel empowered.

To see the whole file, check out section 04 example 3 in the web materials.

The items of interest are:

An extra OpenLayers.Layer.WMS is created, using the same WMS service, but asking just for the "imagery" layer.

```
var imagery = new OpenLayers.Layer.WMS( "Medford Imagery",
   "http://localhost/cgi-bin/mapserv.exe?",
    {
     map: '/ms4w/apps/medford/03/wms.map',
     layers: 'imagery',
     format: 'image/jpeg',
     …
```

The addLayers() method is used, to add multiple layers at once.

```
map.addLayers([basemap, imagery]);
```

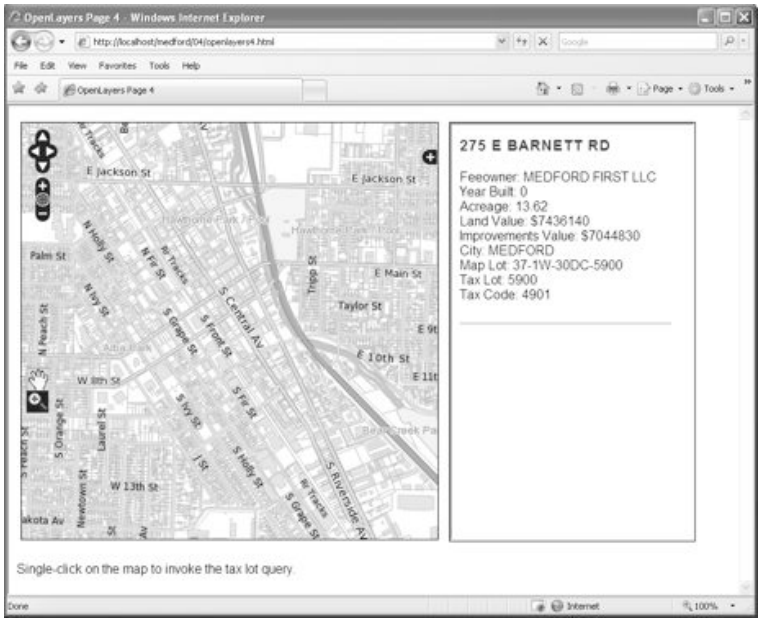
The layer switcher is added.

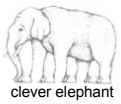
```
map.addControl(new OpenLayers.Control.LayerSwitcher());
```

The rubber band zoom tool is added as part of the NavToolbar.

```
map.addControl(new OpenLayers.Control.NavToolbar());
```

OpenLayers (04)




clever elephant

Finally, let's get an action tied to the mouse click. We only have to add one line to our application:

```
map.events.register("click", map, queryTaxlots );
```

This registers the click event and binds it to a JavaScript function "queryTaxlots" which looks like this:

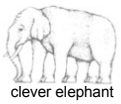
```
function queryTaxlots(e) {
  var lonlat = map.getLonLatFromViewPortPx(e.xy);
  var url = "/cgi-
bin/mapserv.exe?map=/ms4w/apps/medford/04/query.map&";
  url += "mode=nquery&qlayer=taxlots&";
  url += "buffer=10&";
  url += "mapxy=" + lonlat.lon + "+" + lonlat.lat;
  document.getElementById("query").src = url;
}
```

All the query function does is build a URL and pass it to Mapserver. The JavaScript supplies the X/Y coordinate (again, OpenLayers mis-identifies them as "lon" and "lat" because of the world-centric nature of the design) of the mouse click and Mapserver does the rest.

We use an <iframe> HTML element here to hold the results, but there are all sorts of options for viewing the query results.

Mapserver (04)

- http://localhost/cgi-bin/mapserv.exe?
  map=/ms4w/apps/medford/04/query.map&
  mode=nquery&
  qlayer=taxlots&
  buffer=10&
  mapxy=4300000+270000
- LAYER
  NAME taxlots
  DATA taxlots
  TEMPLATE taxlot-template.html
  …
  END


clever elephant

We are using Mapserver as a simple application server here, running a spatial query with a point input, a tolerance of 10 feet, and a target layer.

The result is returned to us, using a template file. Our taxlot-template.html file looks like this:

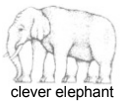
```
<h3>[SITEADD]</h3>
<p>Feeowner: [FEEOWNER]
<br/>Year Built: [YEARBLT]
<br/>Acreage: [ACREAGE]
<br/>Land Value: $[LANDVALUE]
<br/>Improvements Value: $[IMPVALUE]
<br/>City: [CITY]
<br/>Map Lot: [TM_MAPLOT]
<br/>Tax Lot: [TAXLOT]
<br/>Tax Code: [TAXCODE]
</p>
<hr/>
```

All it is is pure HTML, with substitution items set off with [] square brackets. For each record that is returned by the query, Mapserver runs through the template and does substitutions for the attributes in the record.

Note that, while our example uses HTML, there is no requirement for Mapserver to return HTML as a query result. XML or KML could just as easily be generated with this technique, opening lots of possibilities in terms of integrating Mapserver into an AJAX application, or providing KML query results directly to Google Earth.

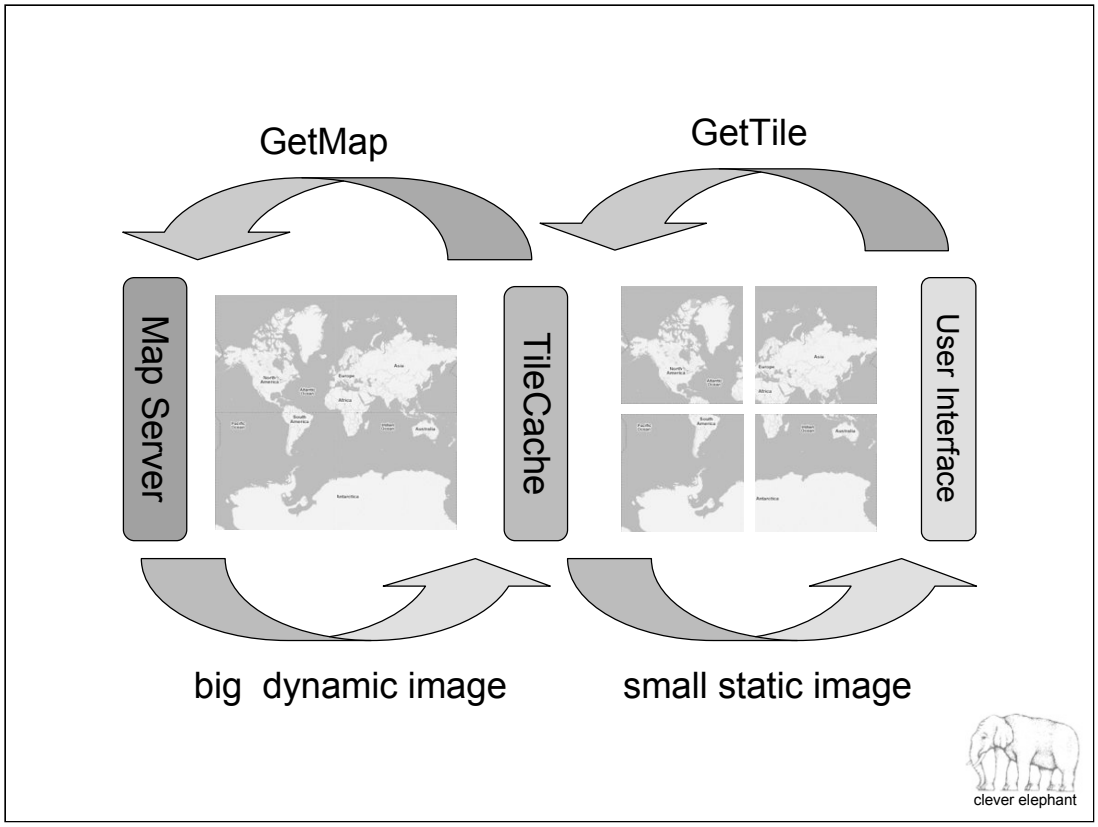
TileCache (05)

- www.tilecache.org
- Client to:
  - WMS, ArcIMS, Mapserver
- Server of:
  - WMS-C, TMS, KML, Worldwind


clever elephant

Tilecache was a response to what happens when you take a tiled map client like OpenLayers and point it directly at a WMS server. OpenLayers merrily generates several concurrent map requests, one for each rendering tile, and the WMS gets bogged down under load trying to service all the requests. And that is silly, since the whole tile/fixed-resolution scheme is designed to make tiles re-usable.

So Tilecache works as a buffer between tile-hungry clients like OpenLayers, Google Earth and others, and the rendering engines like Mapserver, ArcIMS and others.



When a client asks TileCache for a tile:

•It first checks its internal cache.

•If it already has that tile, it returns it directly.

•If it does not have the tile, it makes a request to the map server to render the tile.

When the rendered tile comes back, it returns it to the client, and keeps a copy in the cache.

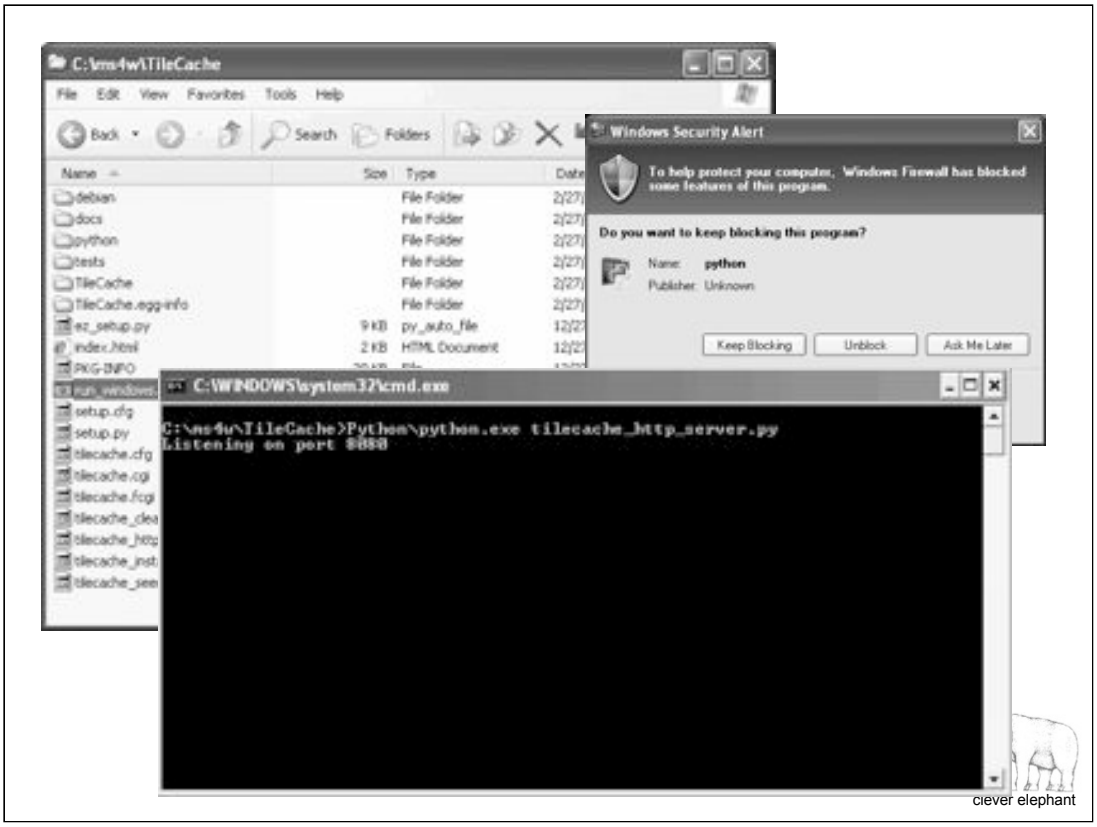
TileCache comes with a utility script, tilecache_clean.py, for cleaning tiles out of the cache. It removes the least recently accessed files from the cache, with a size target for the cache directory supplied with the --size option.

TileCache comes with a script for populating (or "seeding") the cache. This fills up the disk cache ahead of time, so that clients get cache hits more frequently.

```
tilecache_seed.py <url> <layer> [<zoom start> <zoom stop> [<bbox>]]
```

Arguments:

```
url
   http://example.com/yourdir/tilecache.cgi? Or
   http://example.com/yourdir/tilecache.py
layer
   same layer name that is in the tilecache.cfg
zoom start
   Zoom level to start the process
zoom end
   Zoom level to end the process
bbox
   The bounding box to seed
```



To start up TileCache you will need a copy of Python. I have included the Windows version in our software package and made a BAT file to start-up TileCache.

1. Navigate to c:\ms4w\TileCache
2. Double-click run_windows.bat
3. Unblock the program, so it can access the network port.
4. See the application window up and running.

TileCache is controlled by a simple configuration file, **tilecache.cfg**, located in the main program directory.

We will create a couple layers:

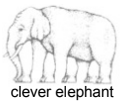
- One for the Medford street map, in Oregon StatePlane South
- One for the Medford imagery, in WGS84, lat/lon

Because our architecture is web based, any of our components can be remote:

- Our map server
- Our tile server (as we have already demonstrated, with the OpenLayers default example)
- Our user interface (as we demonstrated with the OpenLayers default example)

TileCache (05)

- var basemap = new OpenLayers.Layer.WMS("Medford",
  "http://localhost:8080/tilecache.py?",
  {
  layers: 'medford_streetmap',
  format: 'image/png',
  transparent: 'off'
  },
  {
  maxExtent: new
  OpenLayers.Bounds(171956,142812,4389715,355721),
  maxResolution: 100,
  projection:"EPSG:2270",
  units: "ft"
  });


clever elephant

We'll convert the old OpenLayers / WMS viewer from http://localhost/medford/04/openlayers2.html into a tilecache-enabled viewer in http://localhost/medford/05/tilecache1.html.

This requires us to

•Create a TileCache layer that pulls from the WMS layers that are currently in the OpenLayers configuration.

•Change the OpenLayers configuration to pull from TileCache instead of directly from the WMS.

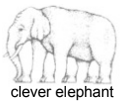
Note that only two things change in the OpenLayers configuration.

•The WMS URL now points to the TileCache end point.

•And the 'layers' is no longer the long list of WMS layers from the source map server, but the single virtual layer being cached by TileCache.

TileCache (05)

- [medford_streetmap]
  type=WMS
  extension=png
  url=http://localhost/cgi-bin/mapserv.exe…/wms.map
  bbox=4171956,142812,4389715,355721
  srs=EPSG:2270
  layers=hydrology,streets,parks,buildings,schools
  maxResolution=100
  metaTile=yes
  metaSize=2,2


clever elephant

It is important that everything in the OpenLayers configuration and the TileCache configuration (tilecache.cfg) match up:

•The OpenLayers ‘layers’ parameter **must** match the TileCache [layer_name]

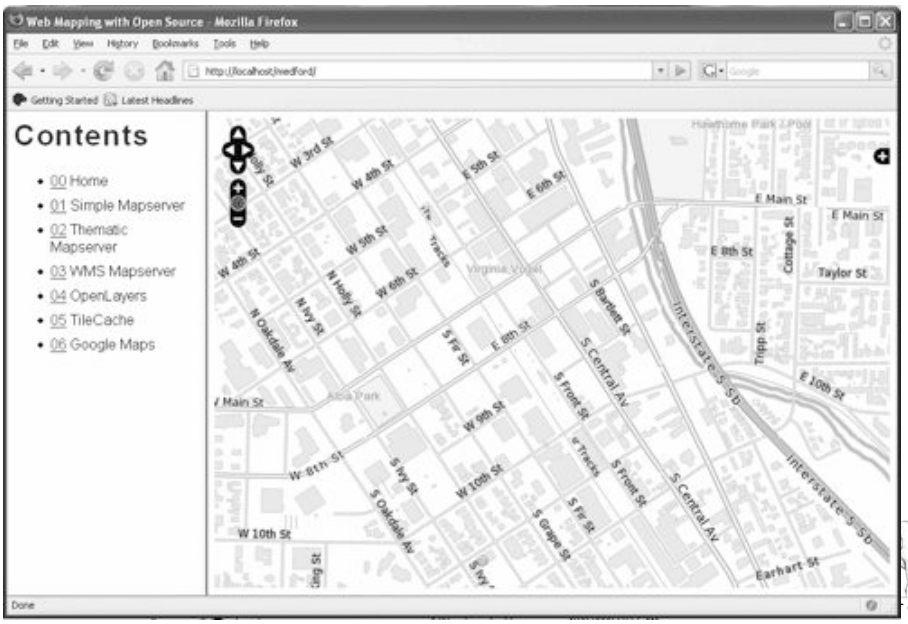
•The OpenLayers ‘format’ parameter **must** match the TileCache ‘extension’

•The OpenLayers ‘maxResolution’ **must** match the TileCache ‘maxResolution’

•The OpenLayers ‘maxExtent’ bounds **must** match the TileCache ‘bbox’

All these items must match because of the delicate game of mathematics being played between OpenLayers and TileCache. OpenLayers figures out what WMS extents to request based on the initial maxExtent and maxResolution. TileCache figures out what tiles it will cache based on the ‘bbox’ and ‘maxResolution’, as well as what format from the extension. If OpenLayers asks for a tile that TileCache doesn’t have, by requesting a tile that has an extent that doesn’t match the pyramid calculated by TileCache, TileCache will respond with an HTTP error: 404, Not Found

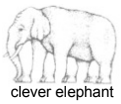
TileCache (05)



Having done all this work, we now have a viewer that works exactly like the one we had before, just with one extra layer of indirection. If there were more users than just us, or we were coming back to the site a second time, we would see more of a speed benefit.

TileCache (05)

- [medford_imagery]
  type=WMS
  extension=jpg
  url=http://localhost/…/wms.map
  srs=EPSG:4326
  layers=imagery
  bbox=-122.918,42.2854,-122.776,42.4056


clever elephant

Our second TileCache example is a classic piece of “service chaining”.

We have a service that speaks WMS (Mapserver).

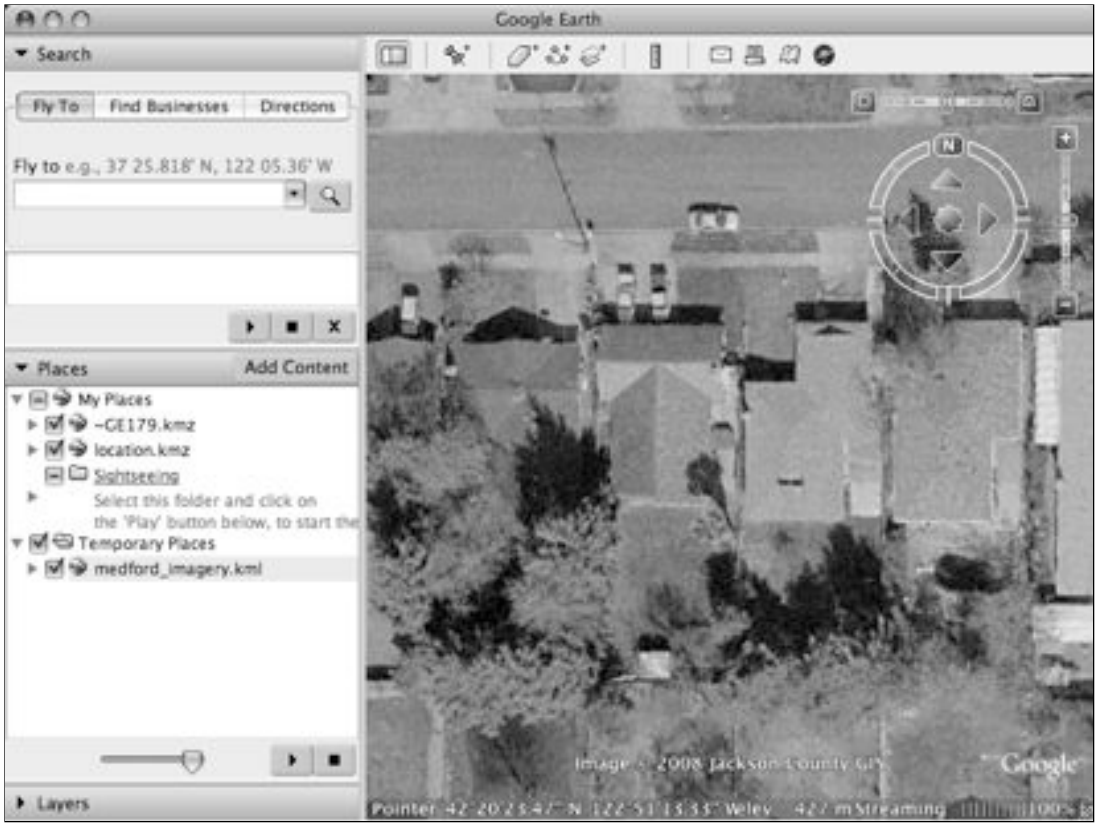
We chain it into a service that speaks KML (TileCache).

The result is a dynamic service that can service arbitrarily large image pyramids, and we show it doing its thing using the Medford image data published in our wms.map example.

Note that we are specifying an SRS of 4326 (Long/Lat WGS 84) in our TileCache configuration. This means that when TileCache makes an image request to Mapserver, it will ask for the results in Long/Lat, instead of Oregon Stateplane South. So Mapserver will have to do an image reprojection on the fly to service the request.

Note also the bbox line. This helps TileCache constrain the area it will service tiles for. Without a bbox, TileCache assumes it has to service tiles for the whole world (-180,-90,180,90).

If you have Google Earth on your laptop, go to the section 05 index page and click on the KML file in example 2. Once you load the file, you should see things start to happen. Remember, you must have TileCache running for this to work.



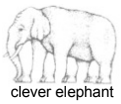
Here is the default Google Earth imagery, and the Medford imagery on top.

You can fade back and forth between the two by using the slider bar at the bottom of the “Places” panel on the left.

Google Earth also supports transparent images, to it is possible to publish an overlay layer, a “mylar”, that just has a few features (building outlines, for example) and nothing else in between.

The main thing to remember as you build up different tile caches is that it uses a fair amount of space. The source image for this super-overlay example is 2GB, and that is at 10:1 compression via MrSID. Even assuming we use just a low quality JPEG compression in our tile cache, we will spend at least 3GB because we have to store all the sampled overview tiles, as well as the full resolution ones.

Google™



clever elephant

Do I have your attention now?

The 800lb gorilla in the mapping space, because they have captured the imagination of both the general population, and the developer community.

Google Maps was the first “slippy” AJAX-style map that caught on. At the time it came out, all the other consumer maps were static click-n-reload systems. Within a year, everyone had re-tooled to catch up to Google.

Google Maps was also the first “open” mapping API. Originally not by design, but the Javascript had to be downloaded to the client to be run. So it was reverse engineered very quickly. The original mash-ups, chicagocrime.org and housemaps.com were based on extending the API using reverse engineering. The official API and API keys came over half a year after the beta release of maps, and the speed of that release may have been driven by the huge enthusiasm of the developer community.

As well as Microsoft, Google recognizes that the developer community must be courted -- they are the ones who make technology recommendations to the bosses, so keeping them happy and being the coolest thing on the block is important.

Google Maps (06)

...er elephant

If brevity is the soul of wit, then the Google Maps API is witty indeed (as witty as OpenLayers!). Here is the full page of code needed to put a map into a page (example 1 in section 06).

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Google Maps JavaScript API Example</title>
  <!-- Load Google Maps API from Google, using API key -->
  <script src="http://maps.google.com/maps?file=api&v=2&key=A…Q"
type="text/javascript"></script>
  <!-- Create a map on the page, using the API -->
  <script type="text/javascript">
    function load() {
      if (GBrowserIsCompatible()) {
        var map = new GMap2(document.getElementById("map"));
        map.addControl(new GSmallMapControl());
        map.setCenter(new GLatLng(42.32652, -122.87559), 13);
      }
    }
  </script>
</head>
<body onload="load()" onunload="GUnload()">
  <div id="map" style="width: 600px; height: 400px"></div>
</body>
</html>
```

Google Maps (06)

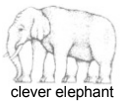
We want to add data on **top** of the default Google Maps. That way we can let Google handle the annoying job of serving up the base mapping data, and we can concentrate on serving up value-added information that Google doesn't care about.

The trick to making this happen is to use a "GTileOverlay". This is an object in the API that runs a few key methods necessary to get appropriate overlay images for the map.

We'll start with an example that provides useless overlay images, to show the mechanism. Example 2 section 06.

```
// Create Google-required copyright structures
var myCopyright = new GCopyrightCollection("© Medford");
var myBounds = new GLatLngBounds(new GLatLng(-122.918,42.2854), new
GLatLng(-122.776,42.4056));
myCopyright.addCopyright(new GCopyright('Demo', myBounds, 0,'©2007
Medford'));

// Create the tile layer and
// implement the three abstract methods for tile requests
var tilelayer = new GTileLayer(myCopyright);
tilelayer.getTileUrl = function() { return "256x256.jpg"; };
tilelayer.isPng = function() { return false;};
tilelayer.getOpacity = function() { return 0.2; }

// Create the overlay, and add to the map
var myTileLayer = new GTileLayerOverlay(tilelayer);
var map = new GMap2(document.getElementById("map"));
map.setCenter(new GLatLng(37.4419, -122.1419), 13);
map.addOverlay(myTileLayer);
```

Google Maps (06)


clever elephant

The final step is doing a tile overlay, by generating WMS URLs in the getTileUrl method. We make the following changes to our static example to make our final demo, example 3 section 06.

We load a helper script that contains a function that can calculate a WMS URL, given a Google tile coordinate and zoom level, which are the inputs to the getTileUrl method.

```
<script src="wms236.js" type="text/javascript"></script>
```

Then we build the GTileLayer as usual:

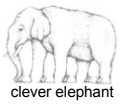
```
var myLayer = new GTileLayer(myCopyright);
myLayer.myBaseURL = "http://localhost/…/wms.map";
myLayer.myLayers = "buildings";
myLayer.getTileUrl = CustomGetTileUrl;
myLayer.isPng = function() { return true;};
myLayer.getOpacity = function() { return 1.0; };
```

We set two global variables, for use by the custom method, myBaseUrl and myLayers, which tell the code what information to get from the WMS server.

We overload the getTileUrl method, as before, and have it use the CustomGetTileUrl method instead. This method is defined in the wms236.js file.

Google Maps (06)

- GTileOverlay gotchas
  - It is new functionality, you must use “v=2.x” in your API key call to get the latest API
  - You must call map.setCenter()
  - The EPSG code being used in wms236.js for Mercator must be supported by your WMS (I use 54004)


clever elephant

The Google Maps API is magic when it works, but not a friendly beast, if you run afoul of it. The errors tend to be obscure, because the code is obfuscated and compressed using single and two-letter variable names.

The API is in continuous development, and new versions are rolled out all the time. You might not even know you’re getting a new one. They try to be backwards compatible, of course.

The “2.x” API key call says “give me the latest version 2 API”. Using just “2” will get you the stable version, but that doesn’t include the GTileOverlay.

In order to discover all the above gotchas, I had to work back from my complex application to the simplest one I could get to work, then slowly add in my functionality until I found what part of the process I was doing wrong.

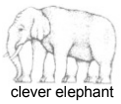
Google Maps (06)

Tiled Map Service (TMS)
http://host.com/1.0.0/layer/**Z**/**X**/**Y**.png

Google Maps
http://mt.google.com/mt?x=**X**&y=**Y**&zoom=**Z**

GTileLayer has tileUrlTemplate option
http://host.com/1.0.0/layer/{Z}/{X}/{Y}.png

Need to ensure that everyone agrees on Z, X, and Y.


clever elephant

Our example of overlaying the buildings onto Google Maps is nice, but notice that it is accessing the WMS directly. Given enough clients the Google Maps overlay could overwhelm the WMS. What to do? TileCache to the rescue!

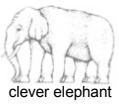
A TMS tile request looks something like this: http://localhost:8080/1.0.0/medford_taxlots/5/12/23.png Three numbers, a zoom Z, a tile X and a tile Y: Z/X/Y.png. This is the same scheme that Google uses.

Because the tile schemes are practically the same, we can access the TileCache TMS from Google Maps without any special extensions, just using an option when instantiating our GTileLayer.

```
var myLayer = new GTileLayer(null,null,null,{
  tileUrlTemplate:
'http://localhost:8080/1.0.0/medford_taxlots/{Z}/{X}/{Y}.png',
  isPng:true,
  opacity:0.5 });
var map = new GMap2(document.getElementById("map"));
map.addOverlay(new GTileLayerOverlay(myLayer));
```

The code is actually shorter! Of course, TileCache must also be configured, to accept the Google requests and turn them into appropriate WMS requests to the map service.

Google Maps (06)



clever elephant

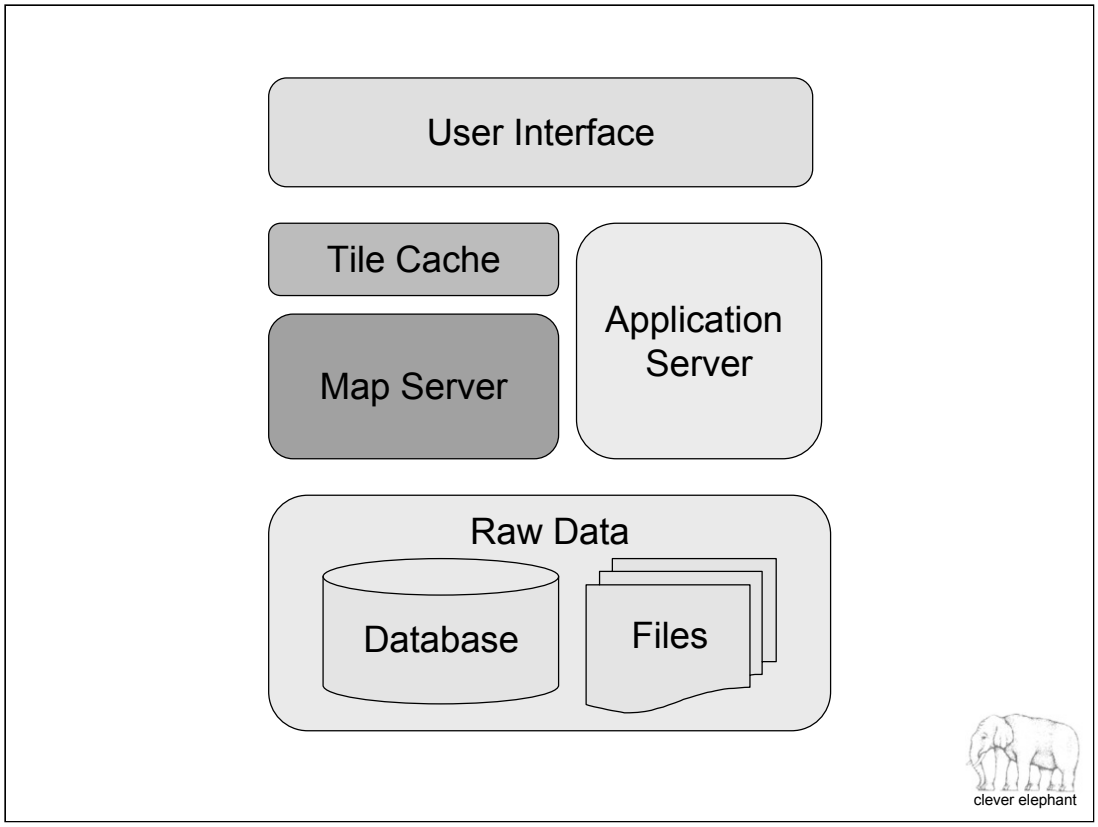
To make sure our TMS has the same understanding of zoom and tile coordinates, we have to make sure it translates them into spatial coordinates using the right bounds and referencing system.

```
[medford_taxlots]
 type=WMS
 url=http://localhost/cgi-bin/mapserv.exe?map=/ms4w/apps/medford/03/wms.map
 extension=png
 layers=taxlots_value
 bbox=-20037508.34,-20037508.34,20037508.34,20037508.34
 maxResolution=156543.031
 srs=EPSG:900913
 tms_type=google
```

The numbers in the bbox and maxResolution are very important, as is the use of the EPSG code 900913. They ensure that the map server will render the right data into the right tiles, that the map server is working in the same reference system as Google Maps.
EPSG 900913 is this projection (in proj4 syntax)

```
# Google Maps mercator-on-a-sphere
<900913> +proj=merc +a=6378137 +b=6378137 +lat_ts=0.0 +lon_0=0.0
+x_0=0.0 +y_0=0 +k=1.0 +units=m +nadgrids=@null
```

All this talk, and no ready-to-run application to deploy back at the home office?



How are you going to apply this information when you return to the office?

Remember, we are interested in finding the truth, not looking for it: open source is a means, not an end.

One way is to look at the software you have, and how it fits into the web architecture. Are you missing any components? This is an opportunity to broaden your infrastructure by adding open source.

Maybe you already have ArcSDE and ArcIMS, but you are looking for other technologies to push information out to the web? You could add TileCache on top of your ArcIMS, OpenLayers above that, and use some ASP or PHP scripting to add application services on your existing raw data and map server.

What, That's It?

- Full featured frameworks are available
  - MapGuide / Fusion
  - Mapserver / Cartoweb
  - Mapserver / Mapbender
- A component architecture
  - Gives you flexibility
  - Helps you think small

The key thing to take home is that you build your web application from parts, and the parts do not have to be tightly coupled.

They don't have to be from the same vendor.

They don't have to run on the same machine.

They don't even all have to be owned by you.

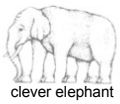
Full-features open source frameworks are available, but they hide some of the "magic" and so it is harder to use them as a teaching tool about web architecture concepts.

Starting from minimal parts, hopefully you understand more about how web services mapping work. The Zen.

Homework

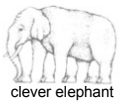
• **postgis**.refractions.net	• database
• www.**gdal**.org	• image translator
• www.**qgis**.org	• desktop viewer
• **udig**.refractions.net	• desktop viewer
• **grass**.itc.it	• desktop analysis
• www.**geoserver**.org	• j2ee server
• www.**ossim**.org	• image processor

OSGeo
Your Open Source Compass



clever elephant

If this workshop has you jazzed about open source geospatial tools, here is a list of top open source projects that are worth your time to investigate and try out.



clever elephant