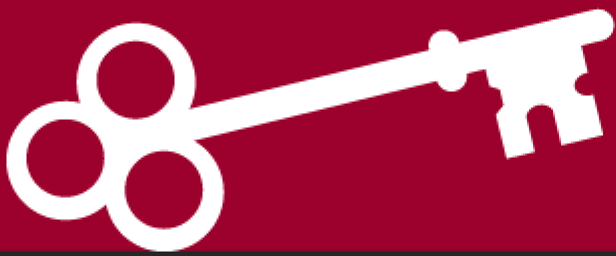

GIS Succinctly

by Peter Shaw

Syncfusion | Technology Resource Portal

Reviewed Document

Chapter 1 So, what exactly is a GIS?

"produced" is misspelled

To most people, what they see as a GIS is in fact just the front-end output layer, such as the maps procuded in Google Maps, or the screen on a TomTom navigation device. The reality of it ~~all~~ extends far beyond that; the output layer is very often the end result of many interconnecting programs along with massive amounts of data.

A typical GIS will include desktop applications used to visualize edit and manage the data several different types of backend databases to store the data and in many cases a huge amount of custom written software tools. In fact, GIS is one of the top industries where a programmer can expect to write a very large amount of custom tooling not available from other companies.

Use proper punctuations

We'll explore some of the applications in detail soon, but for now we'll continue with the 100-foot view. A typical GIS processing setup will look something like as in the above image:

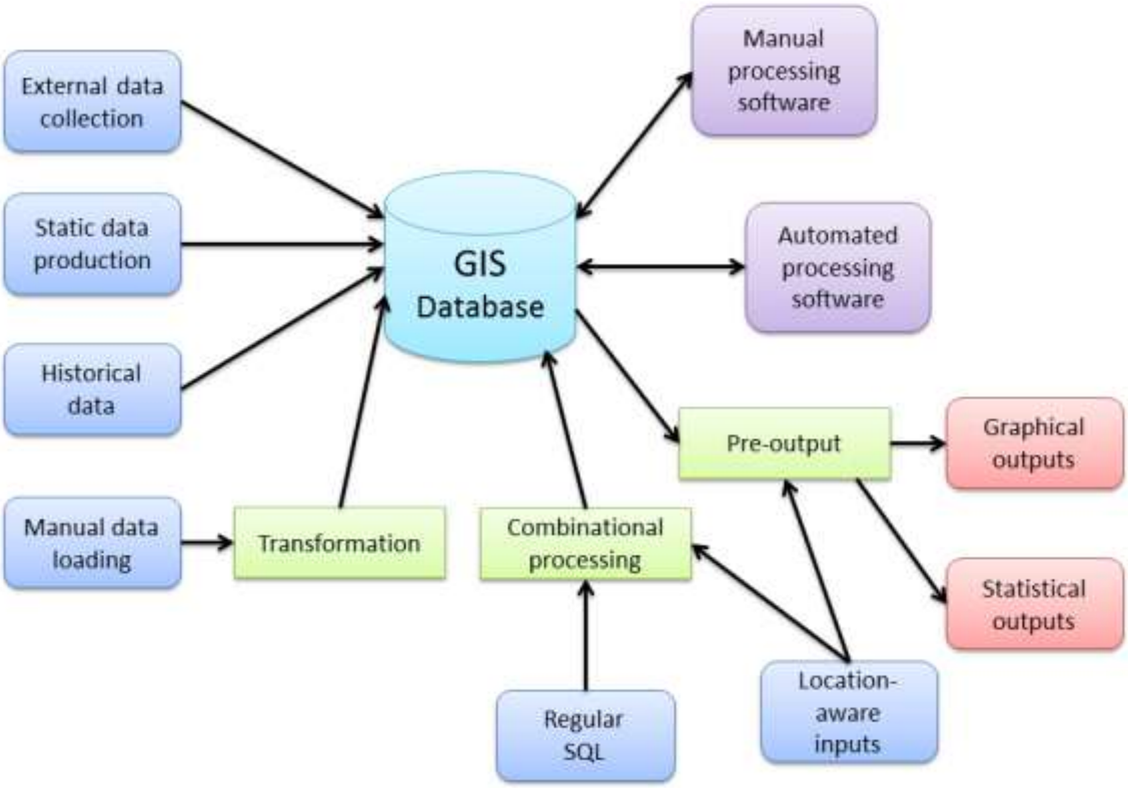

Figure 1: Typical GIS processing setup

As you can see in the diagram, the central part is very often the database itself with a huge number of inputs and processing steps. Finally, the output layers (shown in red) are what people usually associate with being a GIS.

Based on this, we can see that the database is the center of the universe when it comes to GIS.

A Breakdown of the Components

Looking at the diagram in Figure 1, we can see that there are a number of parts that have specific meanings. We have our inputs (blue), outputs (red), in-place processing (green), and end processing (purple). At this point you might be asking yourself,

"How is this different from any other data-centric system I deal with?"

and you'd be right to do so. The main difference here is that in a typical GIS, you have to design everything in each component from the very beginning. With a regular data-centric system, many of the components are often optional or are combined into multifunctional components.

For a typical GIS, none of what you see in Figure 1 is optional, except for possibly your inputs. Even then, the components you'll most likely see omitted are manual and historical data.

So what do these separate entities entail, and why are they often not optional?

External Data Collection

As the name suggests, this is the process of gathering external data specific to the system being designed. Typically this will come from custom devices running custom software (often embedded or small scale) designed to create input data in a very specific form for the system it is being used in. The lack of any in-place processing generally means the data produced is in a format that is already acceptable in the setup.

This component is typically satisfied by many diverse pieces of technology, and in most cases requires some training to use correctly. You'll often see things like digital surveying equipment or specialized GPS devices fitted to vehicles, which in many cases will often feed data back in real time using some kind of radio connection.

Static Data Production

Like external data, this process normally gathers data in a specific format for the system it is being used in. Unlike external data however, you will generally find that static data is produced in-house by scanning existing paper maps or digitizing features from existing building plans, for instance.

Like external inputs, static data is often produced using custom software and processes specific to the business.

Historical Data

Because of the size and amount of data produced in a typical GIS setup, there is often a need to back up data into a separate archival system while still maintaining the ability to work with it if needed. Often, data of this nature is created by planning authorities showing things like land use over time or recording where specific points of interest are. This is treated as a separate input because the data is usually read only, and similarly to external and static data, was at one time produced specifically for the system.

Manual Data Loading

While the name of this type of input may suggest the same as external data, the actual data obtained in this step is usually very different. Data coming into the system via this input will often be in the form of pre-provided data from a GIS data provider. In the United Kingdom, this will often mean data provided by companies like Ordnance Survey. In the United States, this might mean data provided by institutions such as the U.S. Geological Survey or TIGER data from the U.S. Census Bureau.

At this step, wherever data is obtained from, it's almost guaranteed that it will need to be transformed into a format that is useable in the GIS it's destined for. More often than not, it will need to go through some kind of in-place process before it's useable in any way.

Regular SQL Queries

Since most GIS have a large database at the center of them, SQL still plays an important role and probably always will. However, in GIS terms, these queries not only involve the normal SQL that you're used to seeing in a database management system, but also geospatial SQL. We'll cover GIS-specific SQL a little later on; for now, inputs here are usually generated from things like search queries.

As an example, when you type the name of a place or a ZIP code into Google, Bing, or Yahoo Maps, the web application you're looking at will most likely turn your search into a query that uses geospatial SQL to examine data in the core database. This, in turn, will be combined with other processes to produce an output, which in this case will usually be a map displaying the location you searched for. Another example might be an operator in an emergency services control room entering the location of an incident, and combining that with the known locations of nearby emergency vehicles to aid in making a decision as to which vehicle to send to the incident.

Location-Aware Inputs

The last input type is probably the one that is familiar to most people. Location-aware data most often comes from the GPS input on a mobile phone or other GPS-enabled device. It is generally common latitude and longitude information. We'll cover this more when discussing NMEA data.

Graphical Outputs

Now we move to the output layers, the first of which is the graphical one and what most people are familiar with. Output data here is very often in the form of a raster-based map with all operations performed to produce a single output tile in the form of a standard bitmap (such as a .jpeg). However, far more is involved than simple map tiles. Graphical outputs can, and very often are, produced in various vector formats, or as things like AutoCAD drawings for loading into a CAD or modeling package. In fact, even in web environments where people are used to seeing bitmap tiles, it's common for graphical output to take the form of SVG or KML data combined with a custom Google Maps object. Raster tiles are just the tip of the iceberg.

Statitsical Outputs

Outputs in this group are the complete opposite of graphical outputs. Data is often the by-product of several GIS–SQL operations based on the input data and processes going on within the system. Just like general database data, from this output you'll get facts and figures that can be used to report statistics to management or marketing teams. The reason we treat this separately, however, is because of the nature of the information.

While you might be tempted to just say "It's only numbers" in some cases it's numbers that have no meaning unless there is some GIS input involved. As an example, let's say we have a number of geographic areas representing plots of land and with each of those areas we have a monetary value for that plot.

We can easily say, "Give me the values of each plot in descending order," enabling you to see which is the most expensive piece of land overall. This is where the difference stops, however. Let's say we now know that all land in a district has a 1% tax for every square meter a plot consumes. We know by looking at a graphical output of the map that the visually bigger areas are going to be more expensive, but you can't convey that to a computer.

You can, however, ask using GIS–SQL for a statistical analysis based on a percentage of the land's plot value multiplied by however many square meters are in the defined area boundary.

Manual Processing Software

Anything in the system that requires an operator and some software to make changes falls under the category of manual processing software. Typically, this is both an input and an output because in most cases this involves changes being made to the underlying data manually.

This is usually the area where you'll see large GIS packages such as ESRI, DigitalGlobe, and MapInfo used. We'll cover some of these later. An example of what might be performed at this stage is boundary editing. Let's say that you added some town boundaries as area definitions several years ago, and since they were first added the towns have increased in size. You would then find a GIS expert who, with his or her chosen software and some satellite imagery, would edit your boundary data so that its definition better fits the newly expanded imagery.

Automatic Processing Software

Operations running at this stage are generally not much different than those being run manually. The reason we see a clear separation is because some processes simply cannot be automated and need a human eye to pick out details. Going back to our previous example of the town boundaries, it's not beyond imagination that a process can be defined to analyze an aerial image and determine if boundaries need to be removed.

Most often, however, automatic editing is used to perform tasks such as drift correction or height and contour changes due to earth movement.

Transformation Tasks

As mentioned in the discussion of manual data input, when obtaining data for incorporation into a GIS, the data will rarely be in a format suitable for inclusion in the system.

Making the data usable may involve something as simple as a coordinate transform, or something as complex as combining multiple datasets based on common attributes and more. Transformation processes can and often do seriously affect the overall data quality, and many systems can end up with a lot of deeply rooted problems caused by mistakes when transforming data.

In the U.K., these processes are almost always seen when working with latitude and longitude coordinates, as nearly all the data supplied by U.K. authorities will be in meters from the origin, rather than degrees around the center.

Combinational Processing

Combinational processing is generally in-place processing that is the result of various input operations. It's not too different from using a join in a regular database operation. The result is a combination of processes and input data steps that ultimately work in real time to produce a defined input data set.

Pre-Output

Last but not least is the pre-output step. As the name suggests, this is the final processing required before the output is useable. A pre-output process may include transforming an internal coordinate system to a more global one; for example, U.K. meters back to a global scale, or converting a batch of statistics to a different range of values. Location-aware inputs are often included in this step, typically in a navigation system. For example, a location's graphical representation could be combined with current mapping to produce a visual output for a tracking map.

The Database

So just what makes a GIS database so different from a normal database? Honestly, not much. A GIS database is simply specialized for a particular task.

A better way to illustrate what makes a GIS database unique is to look at the growing world of big data. These days, it's hard not to notice how much noise is being made by NoSQL and document-centric database providers. These new-breed databases fundamentally do the same things as a normal database, but use specialized processes that perform particular operations in better, more efficient ways.

Looking at a GIS database through the lens of a non-GIS connection, the geometric data is nothing more than a custom binary field, or blob, that the software and processes working with the system know how to interpret. In fact, it's possible to take a normal database engine and write your own routines, either in the database or in external code, to perform all of the usual operations you would expect but with GIS data.

In general, when a database is spatially enabled, it will have much more than just the ability to understand the binary data added to it. There will be extensions to the SQL language for performing specialized GIS data operations, new types of indexes to help accelerate lookups, and various new tables used to manage metadata pertaining to the various types of GIS data you may need to store.

I'm not going to list every available operation in this book, only the most important things you need to know to get started. At last count, however, there are more than 300 different functions in the last published OGC standards.

OGC What?

The OGC standards are the recommendations set by the Open Geospatial Consortium. They define a common API, a minimum set of GIS–SQL extensions, and other related objects that any GIS-enabled database must implement to be classified as OGC compliant. Because of the diversity of GIS and their data, these standards are rigorously enforced. This enables nearly every bit of GIS-enabled software on the planet to talk to any GIS-enabled database and vice versa using a common language.

Note that when selecting a database to use, there are many that claim to be spatially aware but are not OGC compliant. Prime examples are MS SQL and MySQL.

In general, MS SQL features the OGC-ratified minimum GIS–SQL and functional implementation, but its calling pattern varies significantly from most GIS software. MS SQL also features changes to column names in some of the metadata tables, which means most standard GIS software cannot talk to a MS SQL server. Note also that MS SQL didn't add any kind of GIS extensibility until 2008, and even in the newer 2008 R2 and 2012 versions, the GIS side of things is still not completely OGC compliant.

MySQL has similar restrictions, but also treats a number of core data types very differently, often leading to rounding errors and other anomalies when performing coordinate conversions. You can find the full list of OGC standards documents on the OCG website at http://www.opengeospatial.org/standards/is.

A good place to look for information comparing various databases is on the BostonGIS website at http://www.bostongis.com/?content_name=sqlserver2008r2_oracle11gr2_postgis15_compare#221.

There are also a number of other good starter articles on the site. The downside is that the site is cluttered and sometimes very hard to read.

The Metadata Tables

All OGC-compliant GIS databases must support two core metadata tables called **geometry_columns** and **spatial_ref_sys**. Most GIS-enabled software will use the existence of these tables to determine if it is talking to a genuine GIS database system. If these tables don't exist, the software will often shows an error as shown in the below image.

A good example of this was with early versions of MySQL where the table names were reserved by the database engine, but did not physically exist as tables. This would cause the MapInfo application to attempt to create the missing tables, but it would receive an error on trying doing so, thus preventing the database from being used correctly by the software.

The **geometry_columns** table is used to record which table columns in your database contain geospatial data along with their data type, coordinate system, dimensions, and a few other items of related information.

The **spatial_ref_sys** table holds a list of known spatial reference systems, or coordinate systems as they may be better known.

The entries in the **spatial_ref_sys** table are indexed by a number known as the EPSG ID. The EPSG, or European Petroleum Survey Group, is a working group of energy suppliers from the oil and gas industry who confronted a common problem that arose when surveying the world's oceans for oil reserves: positioning on a global scale. Some companies used one scale, others used a different scale; some used a global coordinate system, while others used a local one.

The group's solution was to record the differences between each scale and the information required to convert from one scale to another reliably without any loss of precision.

Today, every GIS database that claims to be OGC compliant includes a copy of this table to ensure that data conversions from one system to another are performed with as much accuracy as possible.

We'll cover the actual coordinate systems a little later in the book. For now, all you really need to be aware of is that if the **spatial_ref_sys** table does not exist or has no data in it, you will be unable to accurately map or make real-world translations of any data you possess.

Also note that it is possible to save space by removing unnecessary entries from this table. If your data only ever uses two or three different coordinate systems, it's perfectly acceptable to remove the rest of the entries to reduce the size of the table. This can be especially useful when working with mobile devices.

If you only work with data in your own range of values, arguably there can be no data in the **spatial_ref_sys** table at all. I would, however, caution you against removing the table entirely. As previously mentioned, most GIS software will look for the presence of this and the **geometry_columns** table to signify the existence of a GIS-enabled database.

What's Actually in the Metadata Tables?

The **geometry_columns** table holds data pertaining to your data and has the following fields:

f_table_catalog	The database name the table is defined in.
f_table_schema	The schema space the table is defined in.
f_table_name	The name of the table holding the data.
f_geometry_column	The name of the column holding the actual data.
coord_dimension	The coordinate dimension.
srid	The spatial reference ID of the coordinate system in use.
type	The type of geometry data stored in this table.

The **catalog**, **schema**, and **name** fields are used in different ways by different databases. Oracle Spatial, for example, has a single **geometry_columns** table used for the entire server, so the **catalog** field is used to name the actual database. Postgres, however, stores one **geometry_columns** table per database, so the **catalog** field will usually be empty. On the other hand, the **schema** field is used in both Postgres and MS SQL. In Postgres, the field is usually set to **public**, whereas in MS SQL it's normally set to **dbo** for the publicly accessible table set.

The table name and column name are pretty self-explanatory. The coordinate dimension in most cases will be **2**, meaning that the coordinate system has only x-coordinates and y-coordinates. Postgres and Oracle Spatial do have 3-D capabilities, but I've yet to see them used very much outside of very specific circumstances, and I've never seen a **coord_dimension** field set to anything other than **2**.

We'll cover the **srid** field in just a moment. The **type**, however, needs further explanation.

Database Geometry Types

Any OGC-compliant database has to be able to store three different types of primitives. They are:

- point
- line
- polygon

The names themselves are fairly explanatory. A point is a single *x*, *y* location. A line is a single segment connected by two *x*, *y* end points. A polygon is an enclosed area where a number of *x*, *y* points form a closed perimeter.

However, the three base types are not the only geometry types you'll work with. There are variations such as:

- linestring
- multilinestring
- multipolygon

Plus a few others that are rarely used.

A linestring can be thought of as a collection of line objects where each point, except for the start and end points, is the same as the start or end point of the adjacent line. For example:

    1,2 2,3 3,4

would be a linestring that starts at 1,2, goes through two segments, and ends at 3,4.

A multilinestring can be thought of as a collection of linestrings. For example:

    (1,2 2,3 3,4) (6,7 7,8 8,9)

would be two linestrings running from 1,2 to 3,4, and from 6,7 to 8,9, each consisting of two segments. The two linestrings would have a gap between them.

A multipolygon, as the name suggests, is a collection of polygons, but with a twist. Polygon definitions cannot overlap if they are in the same graphical object. This is illustrated in Figures 3 and 4.

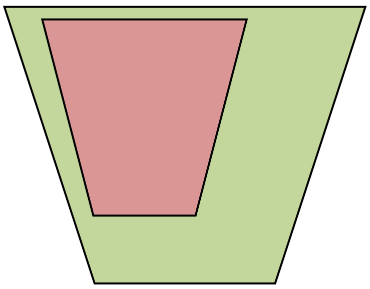

Figure 2: Valid Multipolygon

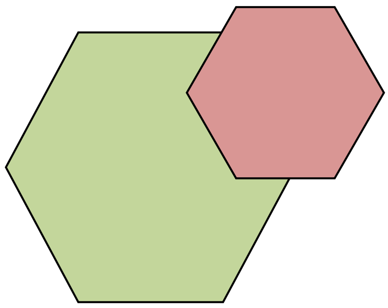

Figure 3: Invalid Multipolygon

A multipolygon must contain at least one polygon that encloses all other polygons in the set. This is known as the **outer ring**. Within this boundary, the other polygons often form holes in the outer ring. This is used for building plans with courtyards, road layouts with roundabouts, anything where an enclosed section needs to be removed from the internal area of the defined shape.

Many spatial databases, however, will define even single polygons as multipolygons. This is done so that it's easy to insert cutouts if needed at a later time.

What Types Should I Use for My Data?

The data types you use depend what your data is representing. If you have a series of locations representing shops, you'll most likely just want to define those as points. If, on the other hand, your data represents roads between those points, a multilinestring is probably a better choice. If you want to mark the building outlines of each shop, you'll want to use a polygon or multipolygon depending on the complexity of the structure.

There are no hard and fast rules for data types. You only have to keep in mind that if you don't use a data type appropriate for the operations you expect to perform, you're almost certain to end up with errors in any calculations you do.

Think back to our shops. If you're searching for the largest one, you need to test for area, and you can't test for area using a single point. On the other hand, if all you want to do is provide a searchable map for a customer to find his or her closest shop, you don't need to store more data than you need, so a simple point will do.

Enough of data layout for now. We'll come back to it in a while. Let's continue with the metadata tables.

Metadata Tables, Part 2

As mentioned previously, the **spatial_ref_sys** metadata table holds conversion data to allow conversions from one coordinate system to another.

Each entry in this table contains specific information such as units of measurement, where the origin is located, and even the starting offset of a measurement.

Most of us are familiar with seeing a coordinate pair such as this:

54.852726, -1.832299

If you have a GPS built into your mobile phone, fire it up and watch the display. You'll see something similar to this coordinate pair. Note that on some devices and apps, the coordinates may be swapped.

This coordinate pair is known as latitude and longitude. The first number, latitude, is the degrees north or south from the equator with north being positive and south being negative. The second number, longitude, is the degrees east or west of the Prime Meridian with west being negative and east being positive. The correct geospatial name for this coordinate system is **WGS84**. Its SRID number is **4326** in the **spatial_ref_sys** table.

We'll come back to the different coordinate systems and why they exist in just a moment. For now, let's continue with the description of the spatial reference table. The **spatial_ref_sys** table has the following fields:

srid	The spatial reference number as defined by the OGC standards.
auth_name	The authenticating body for this SRID, usually the EPSG.
auth_srid	The SRID as defined by the authenticating body, which is normally the same as the SRID defined by OGC standards.
srtext	The definition text used to map the spatial difference in projcs format.
proj4text	The definition text used to map the spatial difference in proj4 format.

Everything in the spatial reference table is straightforward types for integers and strings. The **srtext** and **proj4text** have different meanings depending on what software is reading them.

The **srtext** field holds information for the projection, ellipsoid, spheroid, and other essential information that allows any software to be able to translate from one coordinate set to another. We'll cover this a little more later, but a complete description of everything you will find in this field is well beyond the scope of this small book. In fact, the smallest book I've seen describing the basics was over 500 pages!

The **proj4text** field serves a similar purpose but is used by applications using the open source Proj.4 library.

Proj.4 and Geos were two of the first open source libraries to be used by many different spatial databases and GIS applications. These two libraries are now used in close to 100% of all commercial and open source software used for any kind of spatial or GIS work. Both libraries are still actively maintained and are available for every platform you would expect to work with. We'll meet them again later when we take a brief look at some of the GIS software available for the .NET developer.

For now, all you need to be aware of is that in order to support different spatial coordinate systems, you must have entries in the **spatial_ref_sys** table.

As previously mentioned, you don't need every entry in the table; you can get by using only the SRIDs that your geometry, database, and software use. Since I live in the U.K., I typically use:

> OSGB36, SRID: 27700—Ordnance Survey, meters with false offset at origin.

and

> WGS84, SRID: 4326—Worldwide latitude/longitude, degrees with minute/hour/seconds offset, origin at 0 degrees latitude (the equator) and 0 degrees longitude (the Prime Meridian).

For other territories, you can import the entire table and see which works best, or you can look up your territory on the EPSG site at http://www.epsg-registry.org/ and grab only the definitions you need. If you are using Postgres or PostGIS as your spatial database, the **spatial_ref_sys** table is populated in a database template with all the known SRIDs available when you install the database. Creating your own databases is simply a matter of using this template to have a fully populated table from the start.

One note of caution before we move on: some databases, while they do support the **geometry_columns** and **spatial_sys_ref** metadata tables, don't create them by default. MS SQL 2008 is noted for this; it uses its own methods for storing spatial metadata. You may find that in some cases you will be required to create some of these tables manually before you can use your database. Additionally, you may also find that some databases create the tables but use a slightly different naming convention, especially for the **geometry_columns** table. For this reason, it's always better to use the official OGC-compliant spatial SQL command set (which can be downloaded from http://www.opengeospatial.org/standards/sfs) to manipulate the data in these tables, rather than trying to manipulate the entries directly.

Coordinate and Spatial Location Systems

Before we can get onto the technical fun stuff and start to play, we have to cover a little more theory. You must understand why all these different SRIDs and coordinate systems exist.

I'd like to send you merrily on your way into your first GIS adventure right now and say this stuff really doesn't matter; however, the truth is I can't and it does matter. In fact, it matters a great deal.

If you don't comprehend this coordinate stuff correctly, it's possible to map an automobile's track as being in the middle of the Atlantic Ocean. While this may not matter for the application you're working on—you may be looking at a general overview of customer dispersal, for example—you should still try to make sure your application is as accurate as it can possibly be.

So the answer to the million-dollar question, "Why do we have to deal with all this coordinate stuff?" boils down to one thing, and one thing only:

> *The Earth is not flat.*

There, I said it. And all naysayers out there who still believe it is need to build themselves a top-notch GIS and check it out.

Jokes aside though, it's the fact that our planet is a sphere that causes all these coordinate system headaches. To make matters even worse, our humble home is not even a perfectly round sphere. It's slightly elongated around its axis, a little like a rugby ball, but not quite as pronounced. This causes further complications because the math we need to use as we look at positions closer to the poles must compensate for the differences in the Earth's curvature.

Degrees, Minutes, and GPS

Okay, so how exactly do we deal with this curvature? There MUST be one measurement that makes sense throughout the whole globe, right? If not, then how on Earth do airplanes and ships navigate from country to country without getting lost or having to keep track all of these different SRIDs?

You'll be pleased to know there is, but it's not as straightforward as just mapping an *x* position and a *y* position at a certain place on the globe.

If you look at any geography textbook or world map, you'll see the Earth is divided into rectangles. These rectangles are formed from the lines of latitude and longitude that make up our planet's wireframe model. It looks something like the following:

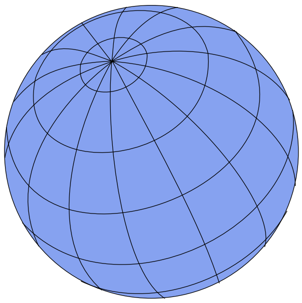

Figure 4: Earth's wireframe model

Each horizontal and vertical line represents one or more whole degrees depending on the scale factor being used. Minutes are then used to offset the position within that grid square.

When we express a latitude of 50° 25' 32" N, what we are actually saying is 50 degrees latitude, plus 25 minutes and 32 seconds north into that square, in simple terms. There's a little more complexity to it if truth be told, but unless you're navigating the high seas or piloting a commercial airliner, you're probably not going to need to go into that much detail.

The same works for longitude. Everything is expressed as a positive number, so west of the Prime Meridian is suffixed with a W, and everything to the east is suffixed with an E. Combining these with the north and south longitude designations divide the planet into four quadrants of 180 degrees each.

How is this of any relevance to the GIS developer?
If you're looking to retrieve the data from any commercial-grade GPS, particularly those built into mobile phones, you'll almost always come face to face with the National Marine Electronics Association and its standards for electronic navigation devices to communicate, known as the **NMEA 0183** standard. Opening the GPS port on just about any device will produce a constant stream of data that looks very similar to the following:

```
$GPGGA,092750.000,5321.5802,N,00630.3372,W,1,8,1.03,61.7,M,55.2,M,,*76

$GPGSA,A,3,10,07,05,02,29,04,08,13,,,,,1.72,1.03,1.38*0A

$GPGSV,3,1,11,10,63,137,17,07,61,098,15,05,59,290,20,08,54,157,30*70

$GPGSV,3,2,11,02,39,223,19,13,28,070,17,26,23,252,,04,14,186,14*79
```

This data stream is the navigation data emitted by the GPS circuitry in the device in response to what it's able to receive from the GPS network orbiting the Earth. We'll come back to this in more detail in a later chapter. For now, I'd like to draw your attention to the first line of this data, specifically the following entries:

5321.5802,N and **00630.3372,W**

These are the GPS' current location expressed as degrees and minutes. Deciphering them is not hard once you get used to it, but it can be a little strange at first.

The format of the string is **DDMM.mmmm** for the latitude (vertical) direction and **DDDMM.mmmm** for the longitude (horizontal) direction.

Starting with the north (latitude) measurement in the string, the first two digits are the number of degrees, and the remaining numbers are the minutes. The numbers after the decimal point are fractions of a minute. This gives us:

53 degrees, 21.5802 minutes north

For the longitude measurement, the first *three* digits are the number of degrees, and the remaining digits are the minutes. All the numbers after the decimal are fractions of a minute. This gives us:

6 Degrees, 30.3372 minutes west

Because this data is string data, it's essentially an exercise in cutting the string at specific points to derive the values you want. Once you have them, the math to convert them to the more familiar latitude and longitude (if you remember that was WGS84) format is very simple.

First, you need to separate the first two digits from the latitude string and the first three from the longitude. This gives us the following:

53 and **21.5802** for the north direction

006 and **30.3372** for west

Because there are 60 minutes in a degree, we must divide the minutes digits by sixty to find what fraction of a degree they are, and then combine them with our whole degrees. So, for our latitude:

53 + (21.5812/60) will give you **53.359686** degrees.

And for our longitude:

6 + (30.3372/60) will give you **6.505620** degrees.

You get simple positions from the numbers. To finish the conversion, you need to apply the north and west directions as positive or negative numbers. The easiest way to manage which directions are positive or negative is to change any west or south measurements to negative. So with our numbers, the final coordinates in WGS84 latitude and longitude are:

53.359686, -6.505620

WGS84 is a global coordinate system standard, and while it is widely used, using it for everything can cause some problems. Because WGS84 is designed to cover the globe, it's designed also to be very lenient with the curvature of the planet. Think back to the wireframe globe in Figure 4. Notice the shape of the rectangles as they near the top of the globe.

You can see in the diagram that the rectangles become longer and narrower. This stretching also has to be accounted for in the coordinate system. Over long distances, it can cause rounding and deviations to occur in your data.

If you're dealing with a territory where you only have a defined area of operation, using a coordinate system more suited to that area is the preferred way of working. As I mentioned previously, for me here in the U.K. it's often better for me to convert these WGS84 coordinates to OSGB36 before storing them in my database. As we'll see later when we start looking at spatial SQL, your GIS database can do this on the fly when set up correctly.

That's pretty much all you need to know as a developer. There's much deeper stuff you can dig into such as spheroid and airy calculations, geodetic measurements, and a lot of that trigonometry stuff from school. The fact is that your GIS database and many of the tools you'll use will actually do the vast majority of the heavy lifting for you. So while having a good knowledge of the actual formulas used by the systems and the Proj.4 strings may be interesting, I assure you of one thing: it will end up giving you a brain ache.

In the next chapter, we start to move onto more interesting things, starting with the software we'll be using.