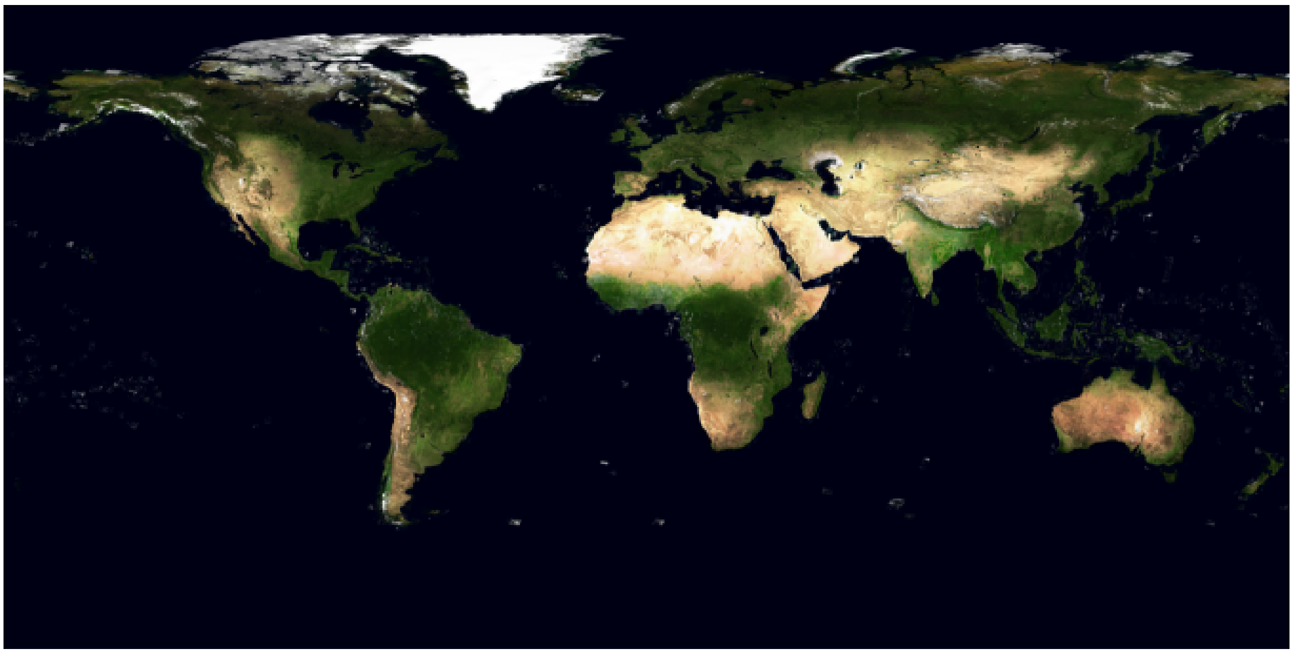
RasterLite

a short **How To** guide based on practical examples

First of all, we need some GeoTIFF to process. Happily we can just download them from the **True Marble** download site. **True Marble** represents a complete world coverage derived from high-quality NASA satellite imagery: this material is released under the ***Creative Commons license***, so we can use in an absolute free way, without any legal issue and at cost zero. True Marble imagery is available at different resolutions:

- 32 km per pixel [*very low resolution, small sized*]
- 16 km per pixel
- 8 km per pixel
- 4 km per pixel
- 2 km per pixel
- 1 km per pixel
- 500 m per pixel
- 250 m per pixel [*best available resolution, huge sized*]

For this tutorial we'll use the ***1 km per pixel*** resolution. Lesser resolutions are two small to be really interesting: higher resolutions are too heavy to be used in a tutorial.
Anyway, if you wish, you can try using the *500m* or the *250m* per pixel resolution: in this case you simply need more time (and more disk space) to complete the tutorial.



Linux / Mac OsX users:

If you are using some smart O.S. (*I intend to mean Linux, or any other Unix-like O.S.*) you can simply launch [*as a shell script*] one the followings:

- to get the *1 Km per pixel* resolution:

```
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.1km.21600x21600.A1.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.1km.21600x21600.B1.tif.gz
```

- to get the *500 m per pixel* resolution:

```
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.500m.21600x21600.A1.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.500m.21600x21600.A2.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.500m.21600x21600.B1.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.500m.21600x21600.B2.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.500m.21600x21600.C1.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.500m.21600x21600.C2.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.500m.21600x21600.D1.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.500m.21600x21600.D2.tif.gz
```

- to get the *250 m per pixel* resolution:

```
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.250m.21600x21600.A1.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.250m.21600x21600.A2.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.250m.21600x21600.A3.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.250m.21600x21600.A4.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.250m.21600x21600.B1.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.250m.21600x21600.B2.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.250m.21600x21600.B3.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.250m.21600x21600.B4.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.250m.21600x21600.C1.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.250m.21600x21600.C2.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.250m.21600x21600.C3.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.250m.21600x21600.C4.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.250m.21600x21600.D1.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.250m.21600x21600.D2.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.250m.21600x21600.D3.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.250m.21600x21600.D4.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.250m.21600x21600.E1.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.250m.21600x21600.E2.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.250m.21600x21600.E3.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.250m.21600x21600.E4.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.250m.21600x21600.F1.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.250m.21600x21600.F2.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.250m.21600x21600.F3.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.250m.21600x21600.F4.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.250m.21600x21600.G1.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.250m.21600x21600.G2.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.250m.21600x21600.G3.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.250m.21600x21600.G4.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.250m.21600x21600.H1.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.250m.21600x21600.H2.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.250m.21600x21600.H3.tif.gz
wget http://ueod-globe.net/globe/TrueMarble_GeoTIFF/TrueMarble.250m.21600x21600.H4.tif.gz
```

Windows users:

If you are using any other O.S. (i.e. one not supporting `wget`), you can directly open your preferred WEB browser and then go to the following URL (*and then manually download any required file, one at each time*): http://www.unearthedoutdoors.net/global_data/true_marble/download

Caution:these files are really huge: some of them has an individual size exceeding 500 MB.
Even if you are using a fast Internet connection, be prepared to wait for several tenths of minutes, or several hours, in order to complete the download.

Once you've finished downloading the complete set of your choice, you have now to `gunzip` any file you've just downloaded (*Windows users can decompress any file using* **7-zip**).
As you can easily notice, each one of these uncompressed TIFF images [21600 by 21600 pixels] has an astonishing size of 1.45 GB.

Warning: never try to get a preview of these images, because such an operation requires a lot of available memory. You can very easily experience some pain and big troubles doing such a thing, and in the worst case your system may crash and/or require a reboot to be reset in a sound state.

All right, now we have any base material needed to get started.

Step 1: creating an empty DB

You simply have to start `spatialite` (*or* `spatialite-gui`*, if you are most familiar using a GUI tool*), and then create an empty DB named **`truemarble.sqlite`** in the same folder where you have placed the TIFF files [*rationale:* `rasterlite_load` *hasn't the capability to fully initialize an empty DB, so you have to prepare the DB using a different* `spatialite` *tool*].

Step 2: checking the GeoTIFFs

Launch the following command from the shell (*set your current directory as the one where you've placed all the GeoTIFFs and the DB you've just created*):

```
rasterlite_load -d truemarble.sqlite -T TrueMarble -D . -t
```

- `-d truebmarble.sqlite` selects the DB you've just crated as the current one to be connected
- `-T TrueMarble` is the name assigned to the raster data source you are going to create
- `-D .` specifies that any valid GeoTIFF contained into the current directory has to be processed
- `-t` means *test-mode-only*: i.e. for now we intend verify the GeoTiffs, without actually inserting them into the DB

Executing this command will produce an output like this one:

```
Processing GeoTIFF: './TrueMarble.1km.21600x21600.A1.tif'
=====================================================
Pixels:     21600h x 21600v
Pixel size: 0.0083333333h 0.0083333333v
EPSG code:  4326
        Upper Left  corner: -180.0000000000 90.0000000000
        Upper Right corner: -0.0000000000 90.0000000000
        Lower Left  corner: -180.0000000000 -90.0000000000
        Lower Right corner: -0.0000000000 -90.0000000000
----------------
Compression:     Uncompressed
BitsPerSample:   8
SamplesPerPixel: 3
TileWidth:       512
TileLength:      512
----------------
Colorspace:      RGB - TrueColor
FormatHint:      JPEG / WAVELET / TIFF
ActualFormat:    JPEG [RGB] quality=75
----------------
RequiredTiles:   1849 tiles [503h x 503v]
----------------


Processing GeoTIFF: './TrueMarble.1km.21600x21600.B1.tif'
=====================================================
Pixels:     21600h x 21600v
Pixel size: 0.0083333333h 0.0083333333v
EPSG code:  4326
        Upper Left  corner: 0.0000000000 90.0000000000
        Upper Right corner: 180.0000000000 90.0000000000
        Lower Left  corner: 0.0000000000 -90.0000000000
        Lower Right corner: 180.0000000000 -90.0000000000
----------------
Compression:     Uncompressed
BitsPerSample:   8
SamplesPerPixel: 3
TileWidth:       512
TileLength:      512
----------------
Colorspace:      RGB - TrueColor
FormatHint:      JPEG / WAVELET / TIFF
ActualFormat:    JPEG [RGB] quality=75
----------------
RequiredTiles:   1849 tiles [503h x 503v]
----------------
```

This represents and exact preview of what `rasterlite_load` will actually do when launched for effective processing.
There is no anomaly detection, so you can now pass to the following ...

Important notice: you'll easily found that `rasterlite_load` will emit some warning message like:

```
TIFFOpen: ./.: Cannot open.
discarding './.': not a GeoTIFF (or read error)
TIFFOpen: ./..: Cannot open.
discarding './..': not a GeoTIFF (or read error)
```

Don't be afraid: this is absolutely harmless. What really happens is simply that `rasterlite_load` doesn't relies on file suffixes (*like* **.tif** *or* **.tiff**) in order to identify the GeoTIFF files contained into the selected directory, but relies instead upon *internal signature* [*magic numbers*] checking for each file.
Consequently, the above warnings simply means: "*Oh yea, this one isn't a GeoTIFF, I'll ignore it*"

Step 3: feeding rasters [*GeoTIFFs*] into the DB

Launch again the same command, this time omitting the -t option, and actual raster loading into the DB will start:

```
rasterlite_load -d truemarble.sqlite -T TrueMarble -D . -v
```

After a few minutes the process terminates regularly, and now your DB containing the Raster Data Source is completely fed.
In the preceding step we used some default setting: and this actually means that now any raster tile is stored into the DB as JPEG (*that implies a lossy compression*), Q=75 (*medium quality, medium compression*). But nothing prevents us using some different image format, or a different compression factor.

We can store our tiles as uncompressed TIFFs:

```
rasterlite_load -d truemarble.sqlite -T TrueMarble -D . -i tiff
```

… as compressed PNG (*loseless compression*):

```
rasterlite_load -d truemarble.sqlite -T TrueMarble -D . -i png
```

… as JPEG Q=90 (*high quality, mild compression*):

```
rasterlite_load -d truemarble.sqlite -T TrueMarble -D . -i jpeg -q 90
```

… as JPEG Q=50 (*modest quality, strong compression*):

```
rasterlite_load -d truemarble.sqlite -T TrueMarble -D . -i jpeg -q 50
```

… as JPEG Q=25 (*infamous quality, very strong compression*):

```
rasterlite_load -d truemarble.sqlite -T TrueMarble -D . -i jpeg -q 25
```

… as WAVELET (*lossy compression*) Q=25 (*high quality, mild compression*):

```
rasterlite_load -d truemarble.sqlite -T TrueMarble -D . -i wavelet -q 25
```

… as WAVELETS Q=50 (*medium quality, medium compression*):

```
rasterlite_load -d truemarble.sqlite -T TrueMarble -D . -i wavelet -q 50
```

… as WAVELETS Q=100 (*modest quality, strong compression*):

```
rasterlite_load -d truemarble.sqlite -T TrueMarble -D . -i wavelet -q 100
```

… as WAVELETS Q=150 (*infamous quality, very strong compression*):

```
rasterlite_load -d truemarble.sqlite -T TrueMarble -D . -i wavelet -q 150
```

Image Type	Visual Example
TIFF (RGB) uncompressed DB size = **2.63GB** PNG (RGB) loseless compression DB size = **397 MB** *both produce an exactly identical result, because a loseless compression algorithm always return a decompressed image exactly corresponding to the original one*	
JPEQ **Q=90** lossy compression, high quality DB size = **82.4 MB**	

Image Type	Visual Example
JPEQ **Q=75** medium quality DB size = **52.6 MB**	
JPEQ **Q=50** modest quality DB size = **39.1 MB**	

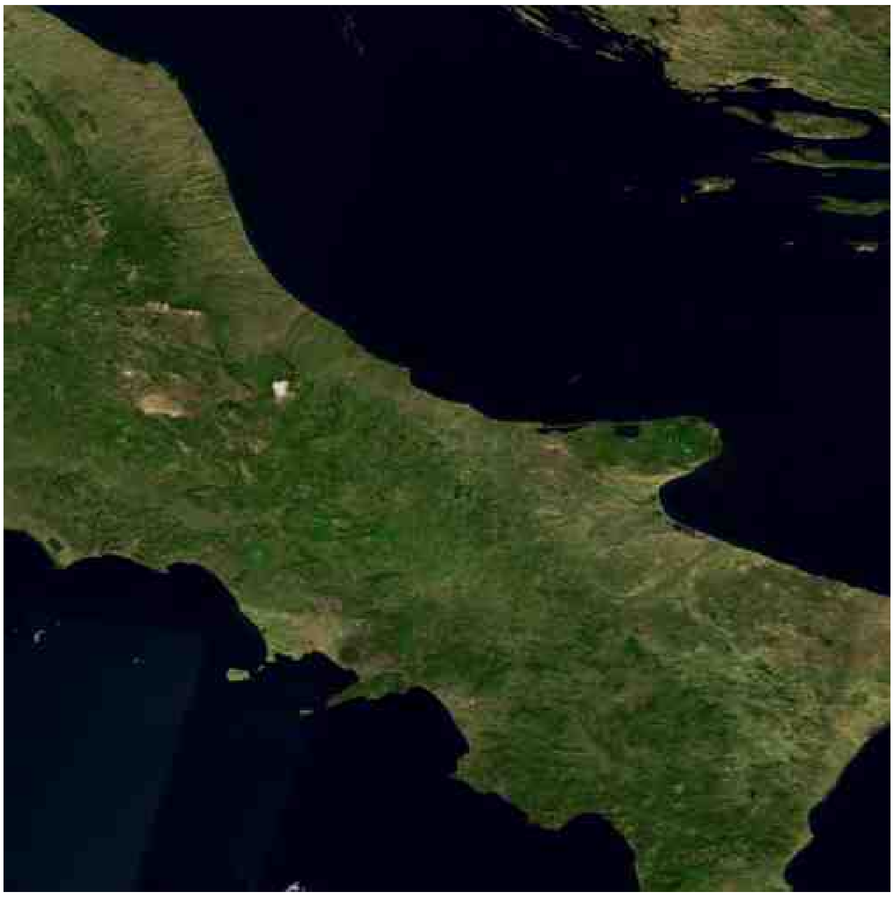
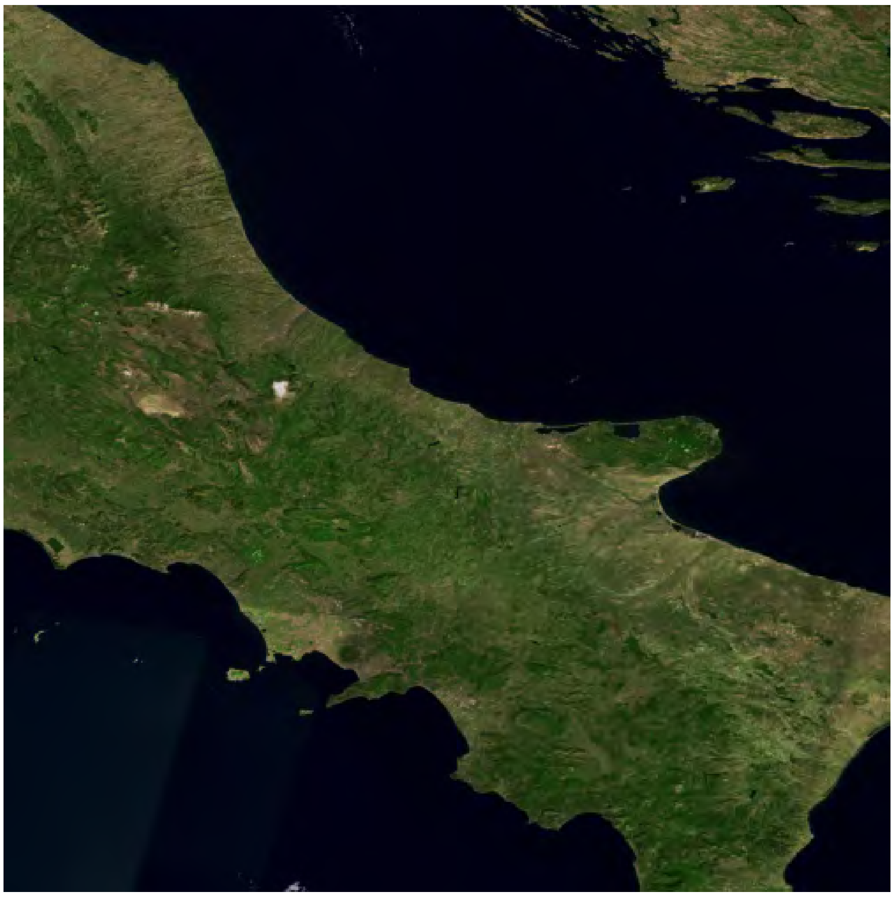
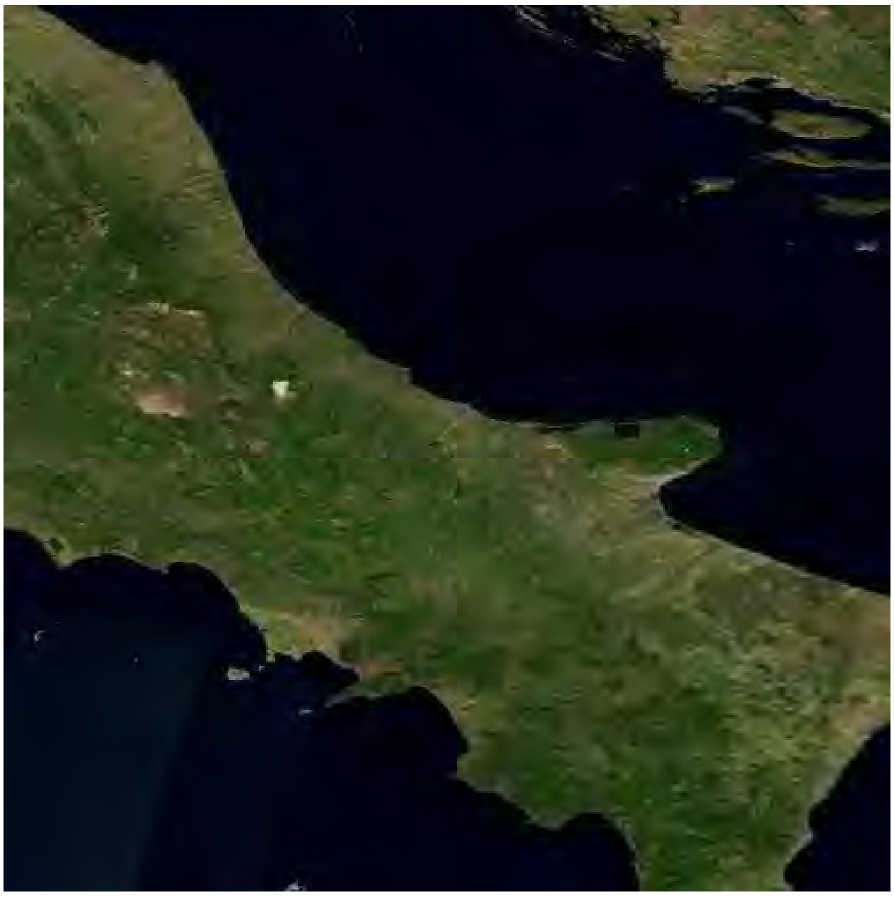
Image Type	Visual Example
JPEQ **Q=25** infamous quality DB size = **29.9 MB**	
WAVELET **Q=25** lossy compression, high quality DB size = **41.3 MB**	

Image Type	Visual Example
WAVELET **Q=50** medium quality DB size = **23 MB**	
WAVELET **Q=100** modest quality DB size = **13.2 MB**	

Image Type	Visual Example
WAVELET **Q=150** infamous quality DB size = **10 MB**	

As you can easily notice, quality offered by lossy algorithms when using very aggressive compression factors, is not at all good: but is still quite useful to be used, because the required data size is really very small: when disk space is absolutely critical, a somewhat compromised quality surely is better than nothing at all.

On the opposite side, using loseless compression algorithms (PNG) leads to an uncompromised quality, but at the expenses of a noticeably bigger data size; using uncompressed TIFFs makes no sense at all, because this will waste an huge amount of disk space, and quality is exactly the same you can achieve using PNG.

A very reasonable half-way compromise may be the one to use lossy compression algorithms (JPEG, WAVELET) applying the lowest compression factors: this will save a lot of disk space anyway, still preserving a really good quality.

Quick and useful format hints:

Tile Image Format	Compression	Notes
TIFF [RGB, Palette, Grayscale]	None	Ill advised. Require too much disk space. Huge-sized tiles make SQL queries to run quite slowly.
TIFF [Monochrome]	CCITT FAX-4 loseless	Best choice for Monochrome images.
PNG [RGB, Palette, Grayscale]	loseless	Best choice, when a loseless compression is required
GIF [Palette]	loseless	Good: applicable only to palette-based images.

JPEG
[RGB, Grayscale]
	lossy, adjustable
	Q=90 best quality
	Q=25 worst
		Very good, when a lossy compression is admissible.
		For optimal results, always use the range:
		Q = 90 / Q = 75.

WAVELET
[RGB, Grayscale]
	lossy, adjustable
	Q=10 best quality
	Q=150 worst
		More or less, the same as JPEG: better than JPEG when applying very extreme compression ratios.
		For optimal results, always use the range:
		Q = 10 / Q = 50.

Step 4: building the Pyramid's Levels

Now you have to perform the another operation to make the Raster Data Source completely ready to be used: building the Pyramid's Levels, i.e. producing the high-quality reduced-scale images representing the Data Source at lowest resolutions.

To perform this last operation, you simply have to launch the following command:

```
rasterlite_pyramid -d truemarble.sqlite -T TrueMarble -v
```

After a few minutes the process terminates regularly, and this time your DB containing the Raster Data Source is completely ready to be immediately used.

Please note: raster images used into Pyramid's Levels are generated by default using the loseless compression PNG format: this is because compressing an already compressed image (*as would be the case, when using some lossy compression algorithm as JPEG*) will introduce lots of artifacts, thus bringing the final result to be far from fully satisfactory.

Anyway, if you are trying to produce a really small-sized DB, using PNG for Pyramid's Levels will waste some extra space.

So in such an evenience you can use e.g. JPEG or WAVELET for Pyramid's Levels images as well:

```
rasterlite_pyramid -d truemarble.sqlite -T TrueMarble -i jpeg
```

```
rasterlite_pyramid -d truemarble.sqlite -T TrueMarble -i wavelet
```

Step 5: building the TopMost Pyramid's Levels

Now you have to perform the very last operation, building the TopMost Pyramid's Levels, i.e. generating the high-quality reduced-scale images joining the various individual GeoTIFFs inserted into the complex Raster Data Source (*a complex Raster Data Source is one containing more than a single GeoTIFF*).

To perform this last operation, you simply have to launch the following command:

```
rasterlite_topmost -d truemarble.sqlite -T TrueMarble -v
```

This one is usually a quite fast step to be performed.

Step 6: testing the Raster Data Source:

All right, the Raster Data Source is finally ready to be used: so we can now try to generate some image using the Data Source in order to check if anything is properly working.
We'll use the rasterlite_tool utility in order to extract such images:

```
rasterlite_tool -o image.jpg -d truemarble.sqlite -T TrueMarble \
  -x 12.5 -y 41.9 -r 0.008333 -w 1024 -h 1024
```

- the **TrueMarble** Data Source uses the **WGS 84** latitude/longitude reference system: so, setting the center-point of the image at **x=12.5, y=41.9** will set such center-point on the city of Rome.
- we'll use a pixel size corresponding to **0.008333** [i.e., *we'll use the best available resolution*]. Please note, this value is expressed into map units as defined by the corresponding SRID; so this value has to be intended as 0.008333 *degrees per pixel*, corresponding to exactly 30 *degree seconds per pixel*.
- and we'll generate a square image, 1024 x 1024.
- we have not set any other arg, so this image will be exported as a JPEG with a 75 default quality.

```
rasterlite_tool -o image.jpg -d truemarble.sqlite -T TrueMarble \
  -x 12.5 -y 41.9 -r 0.008333 -w 1024 -h 1024 -i jpeg -q 20
```

We can generate a second JPEG using a very strong compression …

```
rasterlite_tool -o image.jpg -d truemarble.sqlite -T TrueMarble \
  -x 12.5 -y 41.9 -r 0.008333 -w 2048 -h 2048 -i jpeg -q 90
```

We can generate a third JPEG using a very high quality with a bigger extension …

```
rasterlite_tool -o image.png -d truemarble.sqlite -T TrueMarble \
  -x 12.5 -y 41.9 -r 0.008333 -w 1024 -h 1024 -i png
```

We can generate a PNG image …

```
rasterlite_tool -o image.tif -d truemarble.sqlite -T TrueMarble \
  -x 12.5 -y 41.9 -r 0.008333 -w 1024 -h 1024 -i tiff
```

And finally we can generate a TIFF image. Please note: this actually is a **GeoTIFF:** you can check this loading this image into **QGis**, or using the `listgeo` tool.

Post-Mortem: some final analysis

Base-level tiles format	Pyramid's Levels tiles format	DB size	Notes	Ratio %
TIFF uncompressed	PNG loseless compression	**2.77 GB**	full size, no compression at all	**100%**
PNG	PNG	**543 MB**	loseless compression, fully reversible	**19.1%**
JPEG Q=90	PNG	**229 MB**	lossy compression, moderate high quality	**8.08%**
JPEG Q=75	PNG	**194.MB**	medium quality, medium compression	**6.85%**
JPEG Q=50	JPEG Q=75 poor quality, useful to minimize size	**58 MB**	moderate quality, strong compression	**2.05%**
JPEG Q=25	JPEG Q=75	**48.9 MB**	infamous quality, very strong compression	**1.72%**
WAVELET Q=25	PNG	**189.MB**	lossy compression, moderate high quality	**6.66%**
WAVELET Q=50	PNG	**165 MB**	medium quality, medium compression	**5.83%**
WAVELET Q=100	WAVELET Q=25 poor quality, useful to minimize size	**29.5 MB**	moderate quality, strong compression	**1.04%**
WAVELET Q=150	WAVELET Q=25	**26.2 MB**	infamous quality, very strong compression	**0.92%**

- As you can easily notice, simply using the **PNG** [RGB] format to store both elementary tiles and the Pyramid's Levels ones, will grant a really satisfying 1:5 compress ratio. And all this without sacrificing at all quality, because this one is a loseless (*fully reversible*) compression algorithm.
- Using the **JPEG** [RGB] to store elementary tiles, and **PNG** [RGB] to store the Pyramid's Levels tiles, will grant an excellent **1:12** / **1:15** compress ratio [*Q=90* / *Q=75*]. Quality is really excellent, but remember, this is a lossy compression algorithm, so there is anyway a (*really slight*) quality sacrifice, although this may well be completely unnoticeable to the human eye.
- Using the **WAVELET** [RGB] + **PNG** [RGB] you can reach an even better **1:15** / **1:16** compress ratio [*Q=25* / *Q=50*]. Consider anyway that WAVELET is a little bit slower than JPEG during compression / uncompression.
- Just in an absolute emergency case, using a very compressed **JPEG** [*Q=50* / *Q=25*] can grant an astonishing **1:50** / **1:60** compress ratio, but at the expenses of a very low quality.
- And using a very compressed **WAVELET** [*Q=100* / *Q=150*] can grant a really amazing **1:100** / **1:120** compress ratio: obviously, with a really scarce quality.

Processing the different TrueMarble resolutions:

If you wish, you can obviously process any one of the different TrueMarble resolutions freely available for download.

The following table may help you in evaluating the corresponding DB required sizes:

TrueMarble resolution	DB size
2 km	56 MB
1 km	200 MB
500 m	750 MB
250 m	2.6 GB

Every DB was loaded applying default settings, i.e. JPEG quality = 75 for first-level tiles, and PNG for high-order pyramid's tiles.

Other useful resources available for public download

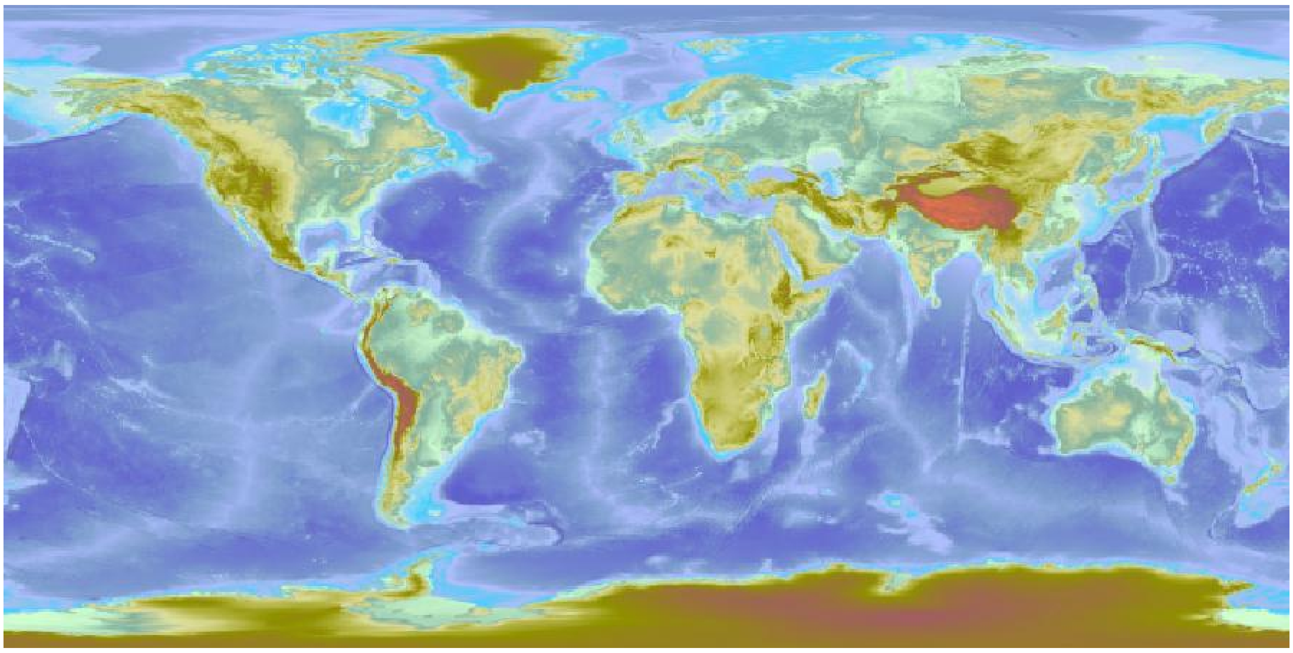
We have explained in fine detail how to build a RasterLite's Data Source using the **TrueMarble** data. But you can try by yourself using some other popular data publicly available for download.

- you can get download lots of high-quality rasters [*US only*] from **National Atlas**: http://www.nationalatlas.gov/atlasftp.html?openChapters=chpgeol#chpgeol

- you can download the **ETOPO-5** DEM grid data from **NOAA / NGDC**: http://www.ngdc.noaa.gov/mgg/global/global.html

- you can download the **SRTM** DEM grid data from **CGIAR – CSI**: http://srtm.csi.cgiar.org/SELECTION/inputCoord.asp

Using ETOPO-5:



Go to the above download site and get one of the **binary float** grid, then uncompress the file you've just downloaded.

```
# gunzip etopo1_bed_g.gz
# ls etopo*
etopo1_bed_g.flt etopo1_bed_g.hdr
#
```

As you can notice, this one actually is a **float** [*binary*] **grid**, consisting of an *header* file [**.hdr**] and a *grid data* file [**.flt**]; accordingly to this, we have first to transform this grid into a corresponding GeoTIFF image:

```
rasterlite_grid -g etopo1_bed_g -c etopo_colors -t etopo.tif \
  -p "+proj=longlat +ellps=WGS84 +datum=WGS84 +no_defs" \
  -f FLOAT -n 0x000000 -v
```

- we are parsing the `etopo1_bed_g` grid [**-g**], which has a FLOAT format [**-f**]
- pixel colors will be set applying the range value definitions contained into `etopo_colors` [**-c**] (*please note: you'll find a copy of this file within the RasterLite downloadable resources*).
- we'll use the WGS 84 [EPSG SRID = 4326] geodetic parameters in order to define a Reference System.
- The output GeoTIFF will be named `etopo.tif`

Please note: this process will be a slowly running one, because this grid is really huge [*about 1GB*], and will produce an huge 21601 x 10801GeoTIFF [*about 667MB*].

Once you've got the GeoTIFF, loading a RasterLite's Data Source will be an absolutely plain task: first of all you have to create a new, empty DB named `etopo.sqlite` [you can easily perform this task using `spatialite-gui`]. Then duly run the following commands:

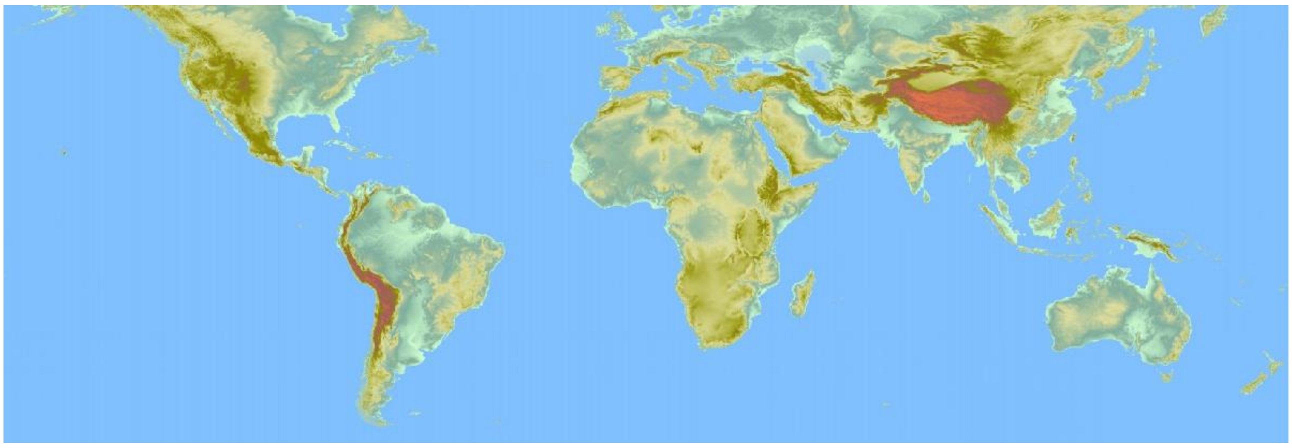
```
# rasterlite_load -d etopo.sqlite -T Etopo -f etopo.tif -v
# rasterlite_pyramids -d etopo.sqlite -T Etopo -v
```

Please note: in this case there is no reason at all to run the `rasterlite_topmost` tool, because there is only one GeoTIFF in the whole Data Source.

When the load process will terminate, you'll get a moderately size DB [*about 93MB*]. You can now visually check some image extracted from the Data Source:

```
rasterlite_tool -o image.jpg -d etopo.sqlite -T Etopo \
  -x 12.5 -y 41.9 -r 0.01666 -w 1024 -h 1024
```

Using SRTM:

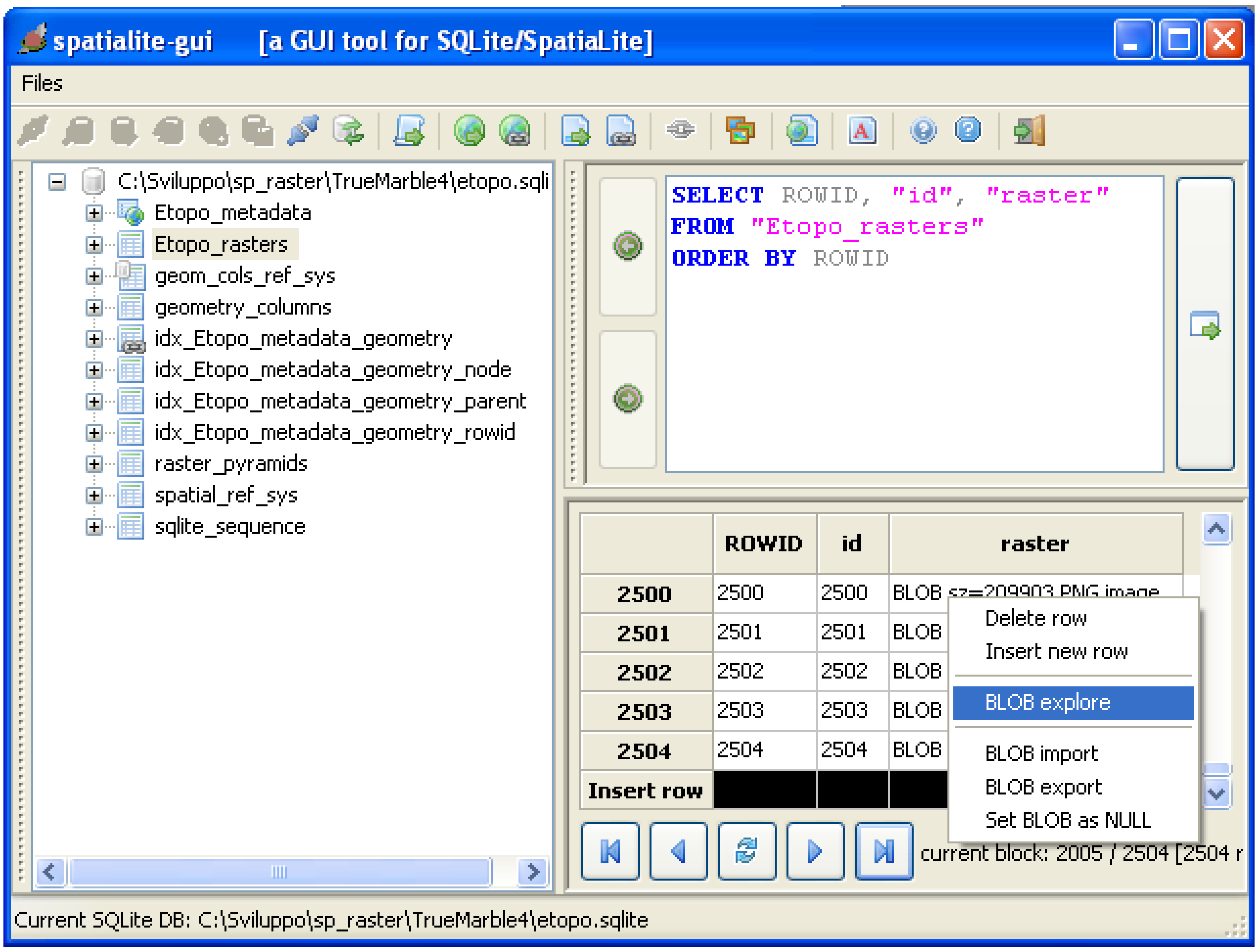


You can process the **SRTM** imagery in way very similar to the one applied for ETOPO. We simply will put into evidence the main differences between these two DEMs:

- ETOPO is a *small resolution* DEM, so it's quite light-weighted.
- SRTM is an *high resolution* DEM, so it's really huge [*more than 70GB of GeoTIFFs*]
- SRTM is distributed as ASCII grids: you can apply the `srtm_colors` file in order to get a predefined Color Table. (*please note: you'll find a copy of this file within the RasterLite downloadable resources*).
- Loading first-level and pyramid's tiles as JPEG quality=75 I was able to load a full-size SRTM world coverage of reasonably good quality: the complete DB requires a disk storage of about 2.5GB.

Using the SpatiaLite GUI tools to check your RasterLite Data Sources

You can use the `spatialite-gui` tool in order to debug / explore your RasterLite Data Sources.



spatialite-gui [a GUI tool for SQLite/SpatiaLite]

Files

C:\Sviluppo\sp_raster\TrueMarble4\etopo.sqli
- Etopo_metadata
- Etopo_rasters
- geom_cols_ref_sys
- geometry_columns
- idx_Etopo_metadata_geometry
- idx_Etopo_metadata_geometry_node
- idx_Etopo_metadata_geometry_parent
- idx_Etopo_metadata_geometry_rowid
- raster_pyramids
- spatial_ref_sys
- sqlite_sequence

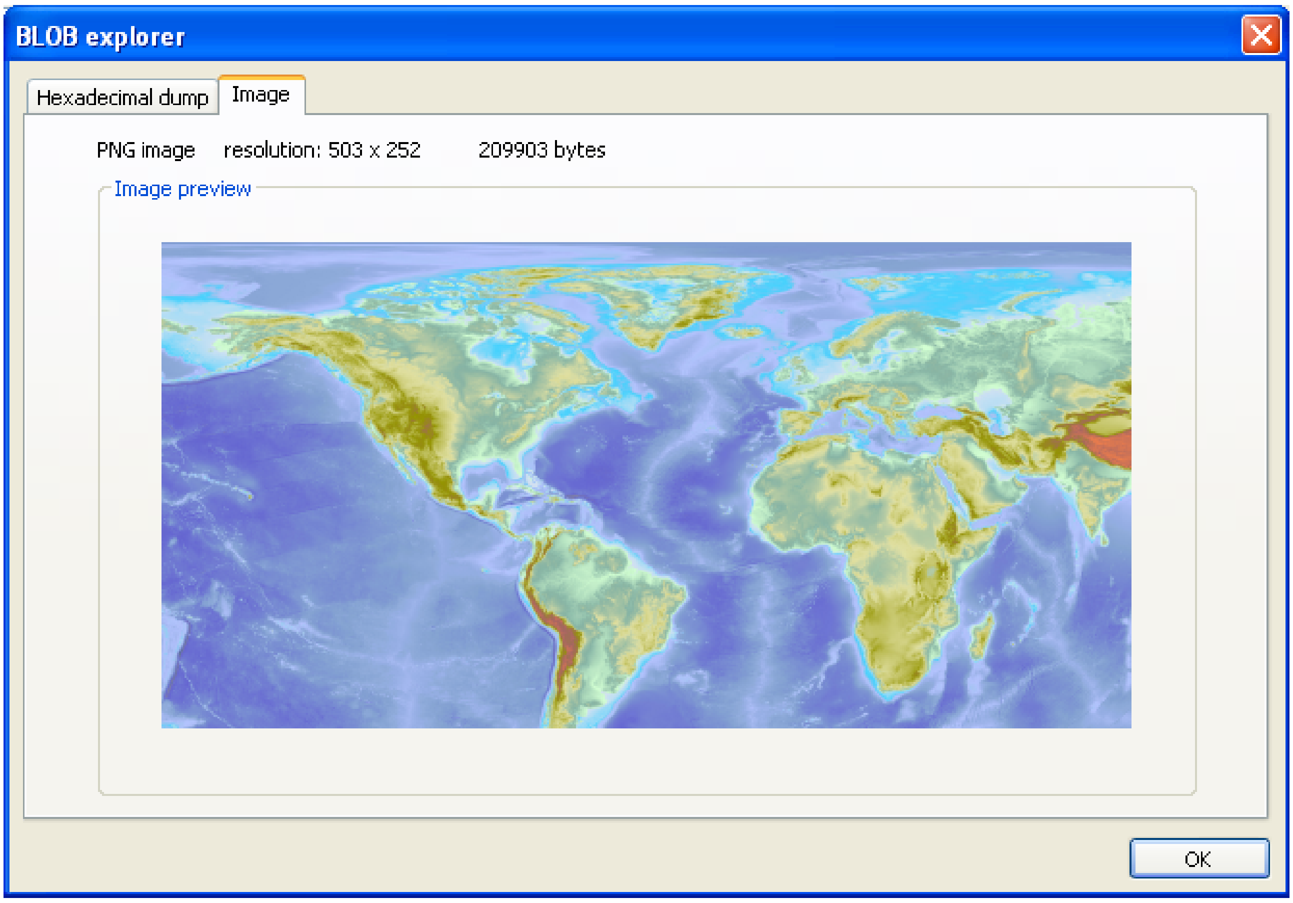
```
SELECT ROWID, "id", "raster"
FROM "Etopo_rasters"
ORDER BY ROWID
```

	ROWID	id	raster
2500	2500	2500	BLOB sz=209903 PNG image
2501	2501	2501	BLOB
2502	2502	2502	BLOB
2503	2503	2503	BLOB
2504	2504	2504	BLOB
Insert row			

current block: 2005 / 2504 [2504 r

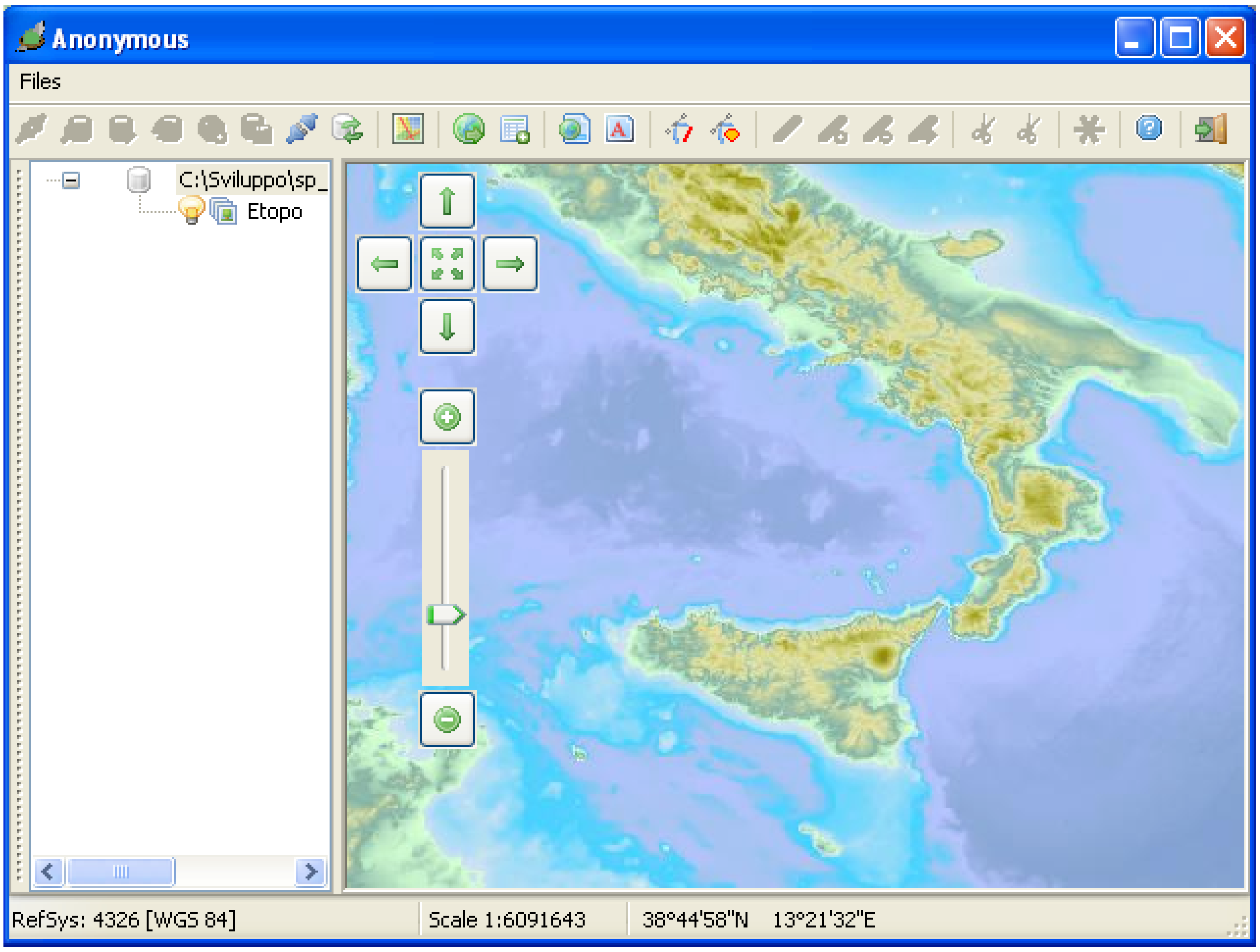
Current SQLite DB: C:\Sviluppo\sp_raster\TrueMarble4\etopo.sqlite

... you can explore the Data Source in a really easy way, simply performing plain SQL queries …



PNG image resolution: 503 x 252 209903 bytes

… and you can as well directly *see* each one individual tile …



But the best way to explore (in a fully interactive way) a Raster Data Source is the one to use the latest `spatialite-gis` tool [*still experimental, ALPHA state, but really useful indeed*]