Jo Wood

The LandSerf Manual

The LandSerf Manual
Version 1.0, 3rd December 2009
for LandSerf 2.3.1

Contents

1 The LandSerf Tutorial

This simple tutorial describes the process that you might go through when using LandSerf for terrain analysis. You can follow the steps using the sample data provided with LandSerf or you can adapt it to use with your own elevation data.

You can start LandSerf by using one of the following methods:

- Selecting it from the Start menu (*Windows*)
- Click on the LandSerf icon  on the desktop (*Windows* or in the Applications folder (*MacOSX*)
- running `landserf.sh` from the command line (UNIX/Linux/MacOSX)
- run or click on `landserf.bat` (Windows)

1.1 Importing an elevation model.

The first stage of any terrain analysis is to import a digital model of the land surface to analyse. Much of LandSerf's functionality concerns processing *Digital Elevation Models* (DEMs) so we will concentrate on that here. A DEM is simply a matrix of elevation values that describes the height of some surface at regular spatial intervals.

For this tutorial, we shall import a DEM from the United States' *National Elevation Dataset* representing part of the Columbia River area in Washington state. You may wish to substitute an alternative area of interest (in which case you can find further details on importing data into LandSerf). The DEM we wish to import for this exercise is taken from the United States Geological Survey (USGS)Seamless data server that allows data to be selected from user-defined regions of the American continent. The data are provided in ArcGIS*Binary Interleaved Layer (BIL)* format and can be found in the `data` sub-directoryof the LandSerf installation.

To import the elevation file:

- Select either the `File`→`Open...` menu or the  button.
- In the drop-down menu labelled `Files of Type`, select the `ArcGIS Binary Image (.bil)` format.
- Use the dialogue window to navigate to the `data` sub-directory of the LandSerf directory (if you have used this version of LandSerf before, the file dialogue will default to the last directory you have used). Select the file `lincolnNED.bil` to load this file.

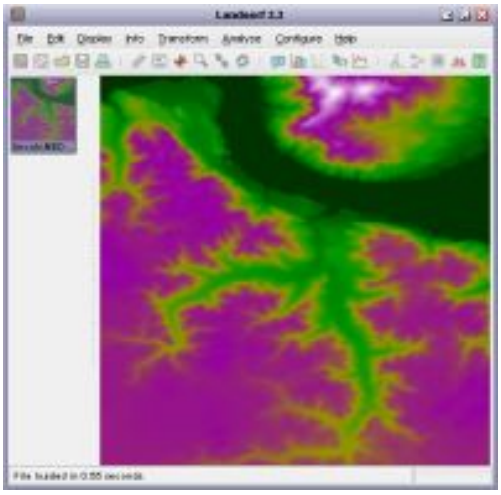


Figure 1.1: Lincoln DEM in LandSerf

You should see a square image of the elevation surface as shown in Figure 1.1. The smaller thumbnail image on the left-hand side of the LandSerf window acts as an 'index' entry. Every time a new object is loaded into LandSerf, a new thumbnail will be added to this index.

1.2 Georeferencing Information.

The elevation model provided by the USGS is currently *georeferenced* using global latitude and longitude values. That is, each of the four corners of the surface are associated with particular locations on the earth's surface. To find out these values we can view the georeferencing information by doing the following:

1.3 Importing an elevation model.

Figure 1.2: Georeferencing information

- Select either the `Info`→`Summary Info` menu item or the button. This should produce a new window, part of which is shown in Figure 1.2.

The problem with using global latitude and longitude values for analysis is that they do not correspond to fixed distances on the ground. One degree of longitude will vary in length depending on the latitude of the measurement. It is therefore useful for us to be able to reproject the surface onto a *planar coordinate system* where there is a fixed scale between the units used and measurements on the ground. For this tutorial, we will firstly store the fact that the imported file uses global latitude/longitude coordinates, then transform the DEM onto a *Universal Transverse Mercator (UTM)* projection:

- Double-click on the thumbnail image of the raster on the left of the main display, or select the `Edit`→`Edit raster` menu item. In the new window that appears, click the `Edit` button in the `Map Projection` section.
- Set the projection information to `Latitude/longitude` from the drop-down menu. Press the `OK` buttons in the projection and edit windows to confirm your selections.
- Select the `Transform`→`Reproject...` menu item and in the window that appears, select `UTM` as the `New Projection`. This will produce a new window allowing you to set the dimensions of the new projected raster (see Figure 1.3)
- In the new window, enter 30 in both the `E-W Res` and `N-S Res` fields. All the remaining fields can be left with their default values.
- Click the `OK` button to create the reprojected raster.

Figure 1.3: Editing georeferencing information

After a few seconds, a new surface should appear as shown in Figure 4. Note that the index area now shows two images - the original as well as the reprojected surfaces. You can select any of the index objects at any time by clicking on the chosen image with the left mouse button. The more rectangular shaped surface now has a fixed resolution where each DEM cell represents a square of 30m x 30m on the ground.

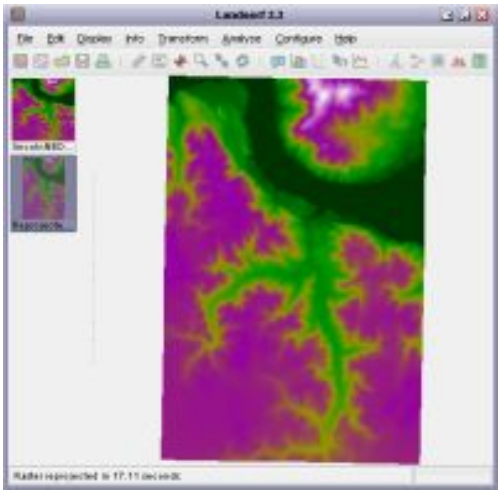


Figure 1.4: The reprojected raster map

- Remove the original unprojected (square) DEM by clicking on its small thumbnail image with the left mouse button and then selecting `File`→`Close raster`. Once closed, click on the reprojected raster in order to select it.
- Check that the resolution of the reprojected raster is indeed 30m in both an east-west and north-south direction by displaying the summary information as you did above. You should also confirm that the raster now has 321 columns and 469 rows.
- We will use this new surface in later stages of this tutorial, so save it by selecting either the `File`→`Save...` menu or the button. Call the file `lincolnNED30m.srf` and save it somewhere where you can later open it.

1.4 Displaying the Surface.

The images we have seen so far that represent the surface are made by allocating a colour to each cell value in the raster model. So for example, all cells that represent an elevation of around 500m above sea level are coloured green and all those that are 800m above sea level are purple. LandSerf can use this same elevation information to model the amount of light or shade on each raster cell by calculating local slope direction and steepness. This can be combined with the colouring scheme you have already seen to produce a *shaded relief* map of the surface:

- To view a shaded relief representation select the `Display`→`Relief` menu item. This should produce an image similar to that shown in Figure 1.5.
- Try altering the shaded relief parameters by selecting either the `Configure`→`Shaded relief` menu item or the button. In the new window that appears, you can move the sliders for each parameter and see the effect each has on the image. When you are happy with the parameter settings you have selected, press the `OK` button. After a second or two, the new shaded relief image should be displayed in the main window.

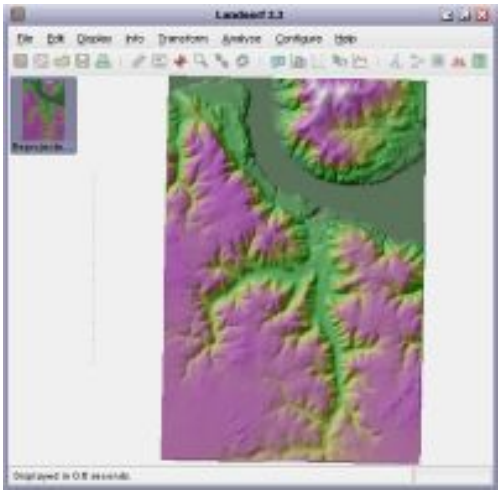


Figure 1.5: The surface with shaded relief

The colour scheme associated with any object can be changed, either by selecting one of LandSerf's pre-defined colour schemes, or by creating a series of colour rules to be associated with the object. At this stage, we will select one of the pre-defined colour schemes:

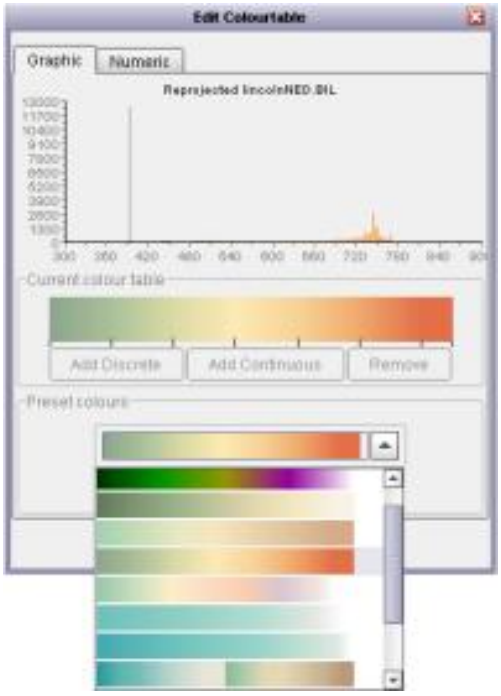


Figure 1.6: Selecting preset colour schemes

- Double-click on the raster thumbnail or select the `Edit`→`Edit raster` menu option. This will display a window allowing you to alter many of the characteristics of the elevation surface.

- Click the `Edit` button next to the colour bar about three quarters of the way down the window. This should display a colour editor window similar to that shown in Figure 1.6.
- From the drop-down menu in the colour editor window, select the fourth colour scheme from the list (green to orange scheme).
- You can refine the selected colour table by dragging any of the black vertical lines below the colour bar in the centre of the window. Each one of these lines represents a single colour and by dragging it to the left or right, you are attaching that colour to a given elevation value. You can also use the frequency histogram at the top of the window to see how many raster values will be associated with each colour.
- When you are happy with the colour scheme you have created, press the OK buttons on the colour editor window and the `Edit Raster` window to confirm your changes to the colour table.

You should now have a new shaded relief representation of the Lincoln elevation model shown in the main display area. LandSerf has used the key colours you have selected and *interpolated* new colour combinations between each of them to produce the continuously varying colours you now see.

You will notice that the northern part of our DEM is dominated by part of a large meandering river (the Columbia River) that is coloured green similar to the banks on either side. In order to distinguish the river from its surroundings, we can associate a unique colour with it by finding its elevation and attaching a *discrete colour rule* to that elevation alone:

Figure 1.7: Interactive query to show river elevation

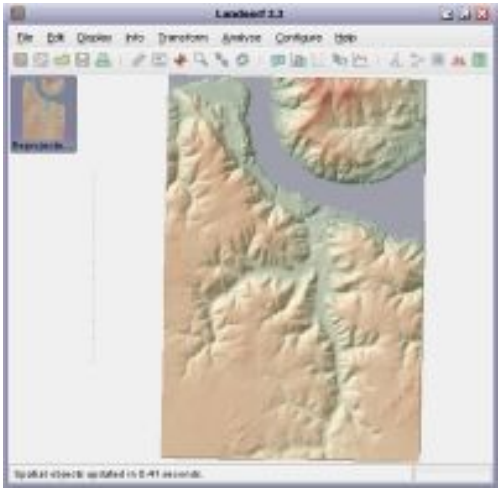


Figure 1.8: Blue colouring for river elevation

- To find out at what elevation the Columbia River lies within our model, we can put LandSerf into *query mode* by selecting either the `Info→query map` menu item or by pressing the button.
- Once in query mode, try moving the mouse over the DEM representation. You should see the geographic position and elevation of the current mouse location being reported at the bottom of the window. Make sure you move the mouse over the river area and note its elevation (see Figure 1.7).
- To move out of query mode, simply select the same option again which will toggle LandSerf back into its normal mode.
- Bring up the colour editor window just as you did previously. This time, select the `Numeric` tab at the top of the window. This will display the colour scheme as a series of numeric rules rather than the graphical display we have previously seen.
- To associate the elevation of the river we have just found out (394m above sea level) with a blue colour, type the following four numbers in the `Edit colour rule` field: `394 141 141 166`. and then press the `Add Discrete` button. This will associate the red-green-blue colour combination represented by the triplet (141,141,166) with any raster cells with the value 394.
- Finally, accept the changes you have made to the colour scheme by pressing the `OK` buttons in both the colour editor window and the raster edit window. You should now be presented with an image of the Lincoln DEM similar to that shown in Figure 1.8.

Sometimes the level of detail represented by an elevation model is greater than that displayed on the screen. It is therefore useful to be able to 'zoom in' to an image in order to examine it in greater detail. This is one way of exploring how the characteristics of a surface vary with scale.

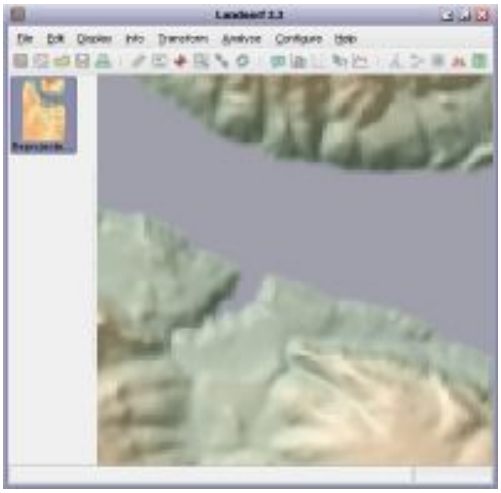


Figure 1.9: Zooming in to the river tributary

- Put LandSerf into *zoom mode* by selecting either the `Display→zoom mode` menu item or by pressing the button.
- Holding the left mouse button down, try moving the mouse up and down over the main image in LandSerf. This should allow you to zoom in and out of the map, centred on

the location at which you first press the mouse down. If you have a scroll wheel on your mouse, this will also allow you to zoom in and out.
- Holding the right mouse button down (or Command and mouse button on MacOSX), try moving the mouse up, down, left and right over the main image. This should allow you to 'pan' across the map. In combination with the zooming function, you should now be able to centre the map on any location of interest at any scale you wish. Note that the currently viewed area of the raster is highlighted on the thumbnail image.
- Use the zooming and panning features to centre the map on the tributary identified in Figure 1.9.
- To come out of zoom mode, simply press the zoom button or `Display→zoom mode` item again. You can restore the image to its original position and scale by selecting either the `Display→Full image` menu item, or the button.
- Finally, save the surface with its new colour scheme just as you did at the end of Step 2.

1.5 Attaching metadata to Rasters.

A spatial object in LandSerf consists of the data (elevation values in the DEM we have used so far) and a collection of *metadata* attached to that object. We have already seen some of those metadata in the form of the georeferencing of the boundaries of the surface, the map projection used, the resolution of each raster cell and the colour rules used to display the raster. We can also create and examine other metadata such as the title, lineage and attribute categories associated with a spatial object.

- If you haven't just moved on from the previous step, load a saved copy of `lincolnNED30m.srf` into LandSerf.
- Bring up the editable metadata by selecting the `Edit→Edit raster` menu option. This should display a window similar to Figure 1.10.
- The title of the raster was created automatically when we previously projected it into UTM coordinates. Change it to something more explanatory by typing `Lincoln 30m DEM` in the `Title` field.
- The 'Notes' field at the bottom of the window contains details of the original file used to create the raster as well as the UTM projection information. This information may be useful if we later wish to trace the *lineage* of the data. We can however add some further useful information about the source of the data. Add the following text to the bottom of the Notes field: `Elevation data from USGS National Elevation Dataset (NED) available from http://seamless.usgs.gov`
- Accept these amendments to the raster by pressing the `OK` button, and save a copy of this object using either the `File→Save...` menu or the button.

LandSerf allows several datasets to be stored and displayed simultaneously. We will add a further raster layer representing *landcover* measured and classified from satellite remote sensing. We will then add some additional metadata to this new layer. Unlike elevation, landcover classes are *categorical* rather than direct measurements. Each cell is allocated a numeric value that represents a class of landcover (water, shrubland, woodland etc.). The value of that number is not directly important other than providing a unique identifier to a given landcover type.

Figure 1.10: Editable raster metadatad'

- Load the file `lincolnLandcover.bil` that is in ArcGIS *Binary Image Layer (BIL)* format and is stored in the `data` sub-directory of LandSerf (see Step 1 if you have forgotten how to do this). If you have just moved on from the previous step, you may wish to set the display from 'relief' back to 'raster' (`Display`→`Raster`).
- These data are again georeferenced using latitude/longitude angles and therefore should be reprojected into UTM coordinates. As you did in Step 2, edit the raster information setting the projection type to `Latitude/longitude` (leaving the ellipsoid set to `WGS84`). Reproject the raster to UTM by selecting `Transform`→`Reproject` and in the window that appears, set the N-S and E-W resolutions both to 30m as you did previously, but this time make sure the `Interpolate to new resolution` box is *not* selected. This is because we wish to create a reprojected raster that only contains the same category values as the original dataset and not create new interpolated categories.
- Give the raster a new title by typing `Lincoln landcover` in the `Title` field (see Figure 1.11). Press the `OK` button to perform the transformation.

We will change 4 elements of the new landcover dataset - the attribute labels; the colour scheme; the type of raster; and the lineage notes of the layer. These changes will make it easier to process

Figure 1.11: Edited landcover metadata

and interpret any analysis we apply to the data at a later stage.

- Double-click on the landcover thumbnail or select `Edit`→`Edit raster` to bring up the raster editor window.
- The first element we will change will be the colours, so press the `Edit` button adjacent to the colour bar. This time, we select the colours from a colour table file, so press the `Load colours` button and select `landcover.ctb` and press the `OK` button to accept these colours (but leave the main `Edit Raster` window open). Note that because the landcover layer consists of categories, all the colour rules are *discrete* rather than the *continuous* rules we used for elevation.
- Next we will associate labels with each of the categories of landcover. To do this press the `Edit` button of the `Attributes` area of the `Edit Raster` window. This should display a new `Attribute Table` window.
- Press the `Load table` button and open the file `landcover.atr`. This file contains the labels associated with each of the numeric attribute values used by the raster. You should now have a table displayed similar to that shown in Figure 1.12. Click on one of the values in the third column to make these attributes 'active'. Finally press `OK` in this attribute window to accept the new table.

Before leaving the raster editor, we will change two more items of metadata.

Figure 1.12: Landcover attribute table

- Notice that the `Raster Type` identified in the `Type` are of the window is set to `Elevation`. Select the drop-down menu in this area and change it to `Other`. This will prevent LandSerf from attempting to perform elevation-specific operations on the raster.
- Finally, we will add some lineage information to the raster. Copy and paste the following into the `Notes` field of the editor window:

```
Originator:  U.S. Geological Survey (USGS)
Publication_Date:  19990631
Title:  Washington Land Cover Data Set
Edition:  1
Geospatial_Data_Presentation_Form:  raster digital data
Publication_Information:
Publication_Place:  Sioux Falls, SD USA
Publisher:  U.S. Geological Survey
Online_Linkage:  http://landcover.usgs.gov.
```

- Save a copy of this newly amended raster as `lincolnLandcover.srf`.
- We can query the new raster by putting LandSerf into *query mode* (see Step 3 above). As you move the mouse over the raster, the numeric and 'active' text labels associated with each cell in the landcover raster should now be displayed (see Figure 1.13).

1.6 Simple morphometric analysis.

We have already gained some impression of the landscape represented by our DEM simply by visualising it. We can assume for example that the landscape contains a large meandering river, some apparently mountainous region to the north, several tributaries flowing from the south of

Figure 1.13: Raster map with landcover colour table

the DEM northwards into the river. We might characterise the region as a whole as 'hilly'. We also know from the landcover information that large parts of the landscape are grassland with some patches of woodland and agriculture.

Such description however has obvious limitations. It is subjective (you may have gained a different impression of the same landscape), it is rather vague (what exactly does 'hilly' mean?), and might simply be incorrect (are we sure that is a mountain to the north of the river or is it just a small undulation?). We can use LandSerf to provide us with a more systematic and objective description of the landscape; one that we can use to compare with other landscapes described by other people.

Firstly it is useful to make sure we have a good idea of the scale and extent of the landscape. We can do this by viewing the metadata associated with the DEM (see Figure 1.14).

- If you haven't just moved on from the previous step, load a saved copy of `lincolnNED30m.srf` and `lincolnLandcover.srf` into LandSerf.
- Make sure the Lincoln DEM is selected as the *primary raster* by clicking on its thumbnail view with the left mouse button. You can confirm that it is selected because the thumbnail should be highlighted with a blue background.
- Additionally we can select or deselect a *secondary raster* by clicking on one of the thumbnails with the right mouse button (or the command key and button press on MacOSX). Doing so will produce turn the thumbnail background pink. Select `Lincoln Landcover` as the secondary raster and then and then `Display`→`Relief`. LandSerf should display a shaded relief map in the main area using elevation from the primary raster to calculate shading, and the thematic colours from the secondary raster to produce the colours of the map.
- Try deselecting the secondary raster by clicking on its thumbnail with the right mouse (cmd-mouse press on MacOSX) button and then redisplaying shaded relief. You should now have a display of the DEM relief without any landcover information.
- View the DEM information by selecting `Info`→`Summary Info` or the button. You should see that the raster consists of 321 columns and 469 rows with a resolution of 30m

Figure 1.14: Linconln DEM metadata

per DEM cell. This means we are viewing a landscape that is approximately 10km by 14km in extent. We also know from the information that the range of heights within this region is approximately 392m to 891m - a vertical range of about 500m.

- Click the `OK` button to close this information window.

This vertical range or *relief* we have identified from the raster information window only gives us a broad summary of how elevation changes over the surface. We can get a more detailed view by examining the frequency distribution of elevation values.

- Select either the `Info`→`Histogram` menu item or the  button. This should display a frequency histogram in a new window.
- You should notice that by far the most frequently occurring elevation value is at 394m, which corresponds to the height of the river Columbia. You can force the histogram to ignore values at this elevation by clicking the `Ignore values at` box and typing 394 in the text area to the right (see Figure 1.15).
- You can change the category widths of the histogram by moving the slider left and right. The most obvious feature of the distribution is the peak in the histogram at about 750m. This corresponds to the plateau area that dominates the southern half of the DEM.
- Close the histogram window by pressing the `OK` button.

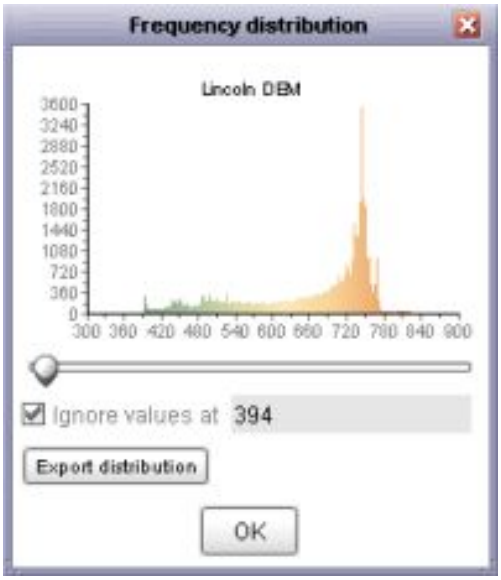


Figure 1.15: Frequency histogram of Lincoln elevations

So can we get a better idea of the 'roughness' of the terrain? One way of doing this is to measure the steepest slope at all cells in the DEM and visualise the results:

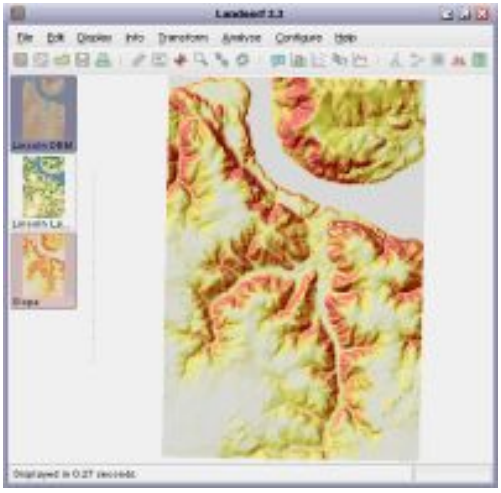


Figure 1.16: Lincoln slope map

- Select either the `Analyse→Surface Parameter` menu item or the button. Select `Slope` from the list of properties and press the `OK` button. You should see a message `Calculating Surface Parameterisation...` appear at the bottom of the LandSerf window and a percentage progress indicated in the bottom-right corner. After a few seconds a new slope image should be displayed (see Figure 1.16). Slope is coloured from white (horizontal) through yellow to red (steepest slope). This new image shows the flat plateau in the south-west and the steeper slopes bordering the Columbia river and its larger tributaries.

- We can examine the slope map in more detail by selecting it as the *primary raster*. Do this by clicking on the slope map thumbnail image with the left mouse button, and then `Display`→`Raster` to redisplay the slope map.
- Try using the *query mode* and *zoom mode* to explore the slope map in more detail (see Step 3 if you have forgotten how to do this).
- Display the frequency histogram of slope map (`Info`→`Histogram` or ). You may wish to widen the graph window so that the histogram bars are more obvious. The histogram is dominated by the large number of horizontal cells (corresponding to the plateau and the river surface) with steeper cells being relatively less common. This distribution shows an approximately *log-normal* distribution typically exhibited by many slope maps.
- Save a copy of this slope map as `lincolnSlope.srf` for later analysis.

Slope represents one of the four *surface parameters* that are often used to characterise surface behaviour. The other three are *aspect* which identifies the azimuthal direction of steepest slope, *profile curvature*, which describes the rate of change of slope in profile, and *plan curvature*, which describes the rate of change of aspect in plan. We shall use LandSerf to view each of these in turn:

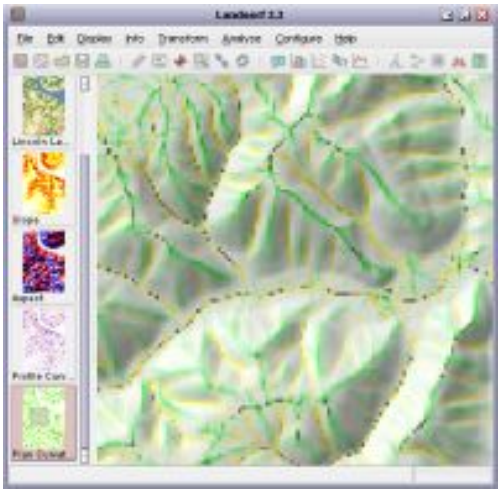


Figure 1.17: Lincoln plan curvature

- Make sure the Lincoln DEM is selected as the primary raster by left-clicking its thumbnail view. Set the display to show shaded relief.
- Select `Analyse`→`Surface Parameter` (or the  button), but this time choose `Aspect` from the list of properties and press the `OK` button.
- Use the techniques you have used so far, such as zooming, querying and histogram calculation to identify any further characteristics of the DEM. Repeat this task for `Profile Curvature` and `Plan Curvature` (see Figure 1.17) measures.

Finally, we shall examine one further characterisation of the surface. Rather than measure *surface parameters* we can classify the terrain into *surface features* using some of the parameters we

have already investigated. One of the commonest (and most useful) classifications is to group all points on a terrain into one of the following:

pits
channels
passes
ridges
peaks
planar regions

This can be done by measuring both the slope and curvature of each cell in the DEMand performing the appropriate classification (for more details on how this may be done see the theory documentation). We will perform such a classification on our DEM:

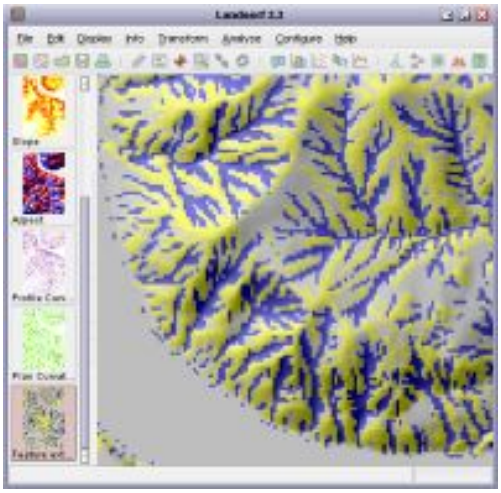


Figure 1.18: Lincoln feature classification

- Make sure the Lincoln DEM is selected as the primary raster by left-clicking its thumbnail view. Set the display to show shaded relief.
- Select `Analyse`→`Surface Parameter` (or the  button), and choose `Feature extraction` from the list of properties and press the `OK` button.
- You should see a predominantly grey, blue and yellow image appear. The grey areas represent planar regions, blue areas represent channels and yellow ridges.If you look carefully (you may have to use the zoom mode to see this), you should also see occasional red cells that represent local peaks and green cells that represent passes in the landscape (see Figure 1.18).
- Save a copy of this feature classification by selecting it as the primary raster (left-click its thumbnail image) and saving it with the name `LincolnFeatures.srf`.

Feature classifications like this are useful for several reasons. The pattern of channels appears somewhat similar to a drainage network, thus giving us an idea of where water would flow over the surface. Perhaps less obviously the pattern of yellow cells gives us an equivalent ridge network, identifying portions of the landscape where water is likely to flow from. Consequently, the ridge network is related to the drainage divides/watersheds that define drainage basins. In

theory at least, the green passes identify points of intersection between the channel and ridge networks. Along with pits and peaks, passes represent what are sometimes called *surface specific points* - points on the landscape that carry special significance in its structure.

1.7 Advanced morphometric analysis.

When we wish to characterise a real terrain using a computer, the closest we can get is to characterise the *elevation model* that represents it. In doing so we hope that the model is sufficiently accurate for the purposes of analysis. As far as possible we should strive to produce characterisations of landscape that are as dependent as possible on the surface we are interested in, but as independent as possible of the particular model we have used.

One of the most significant limitations of the models and processing we have carried out so far has been the scale of analysis implied by the DEM resolution. You will remember that each cell in the Lincoln DEM is 30m x 30m. Since most of the analytical functions we have used so far work by comparing one cell with its 8 adjacent neighbours, the results of that analysis are likely to be at the resolution of approximately 3 times the grid cell size. So for example,the blue valleys we identified in the previous step will be those that have a width around 30-90m. This scale is really rather arbitrary; we might be more interested in characterisation of features have expression over1km or 10cm. Unfortunately, the DEM cannot provide us with direct information at a finer scale than its resolution (30m), but we can use it to perform analysis at a broader scale.

The basis for most of the analytical functions in LandSerf is the process of *quadratic approximation*(see theoretical details on surface characterisation for further information). This involves taking a local window of DEM cells (3 x 3 in the examples we have seen so far), and fitting the most appropriate continuous quadratic function through them. LandSerf then uses this quadratic function to calculate parameters such as slope and curvature. We can set the size of the local window that LandSerf uses before it does any calculation. This allows us to perform analysis at much broader scales than that implied by the DEM resolution.

Firstly, we will view the effect of changing window size by smoothing out features that are smaller than about 330m wide. Since each cell is 30m, we will need to process the surface using window sizes of 11x11 cells rather than 3x3:

- If you have just moved on from the previous step, close all rasters except the Lincoln DEM. To remove an object from LandSerf, select it as the primary raster with the left mouse button, then select the `File`→`Close raster` menu option.
- Make sure the Lincoln DEM is the primary raster and that display is set to `Shaded Relief`.
- Select either the `Configure`→`Window scale...` menu item or the  button, and set the window size to 11 rather than 3 (we shall leave the 'distance decay exponent' set to 0). Press the `OK` button to close the dialogue box.
- By changing window scale, any subsequent analysis will be performed at this new 330m scale. To demonstrate this effect we shall reinterpolate our DEM at the new scale. Select `Analyse`→`Calculate Surface Parameter` (or the  button) and choose the `Elevation` item. You will notice that processing at this new scale takes longer than

previous examples since LandSerf is having to process 13 times as many cells for each calculation.

- To view the effect of this smoothing, select the `Smoothed Elevation` layer as the primary raster and then `Display`→`Refresh` (see Figure 1.19). You can quickly alternate between the original and smoothed DEM by selecting either as the primary raster and then pressing `Ctrl-R` to refresh the display.

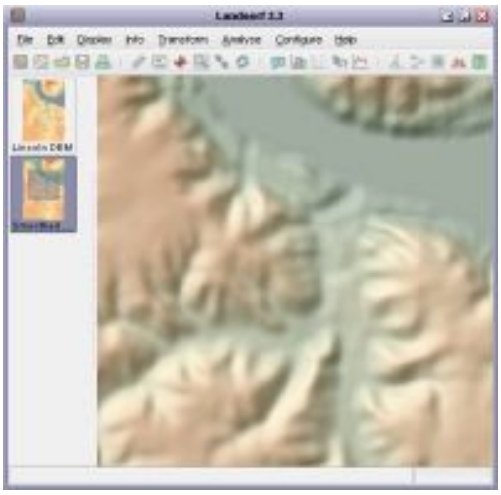


Figure 1.19: Shaded relief at 11x11 cell window size

The result of broadening the scale of analysis is to smooth out many of the finest scale details while leaving the larger features of the landscape unaltered. You may also notice that the entire raster also appears smaller than the original. This is produced because analysis based on square windows of size *n*x *n* cannot process cells within *(n-1)/2* cells of the edge. Asa consequence, LandSerf fills the border cells of the raster with *null values* that are not displayed or processed.

We can see the effect of further broadening the scale of analysis by setting the window size to65 (corresponding to a distance on the ground of 65*30m = 1.95km):

- Make the original DEM the primary raster and then set the window size to 65 (`Configure`→ `Window scale...` or).
- Select the `Analyse->Calculate Surface Parameter` menu item, and choose the `Elevation` option again. Processing will be much slower this time as LandSerf is now processing 470 times as many cells as the original 3x3 window. This might be a good time to make yourself a cup of coffee. If you ever find yourself waiting for a very slow process to complete in LandSerf, you can always stop it by clicking somewhere on the process bar in the bottom-right corner.
- When you have drunk your cup of coffee and LandSerf has finished calculating the surface, view it by making it the primary raster and refreshing the display (see Figure 1.20).
- Save a copy of this smoothed DEM as `lincolnNED65.srf` for later use.

So do the parameters that we can measure from the surface also vary if we calculate them at this new broader scale? We can find out the answer to this question by measuring a surface parameter at many different scales and seeing how it varies:

Figure 1.20: Shaded relief at 65x65 cell window size

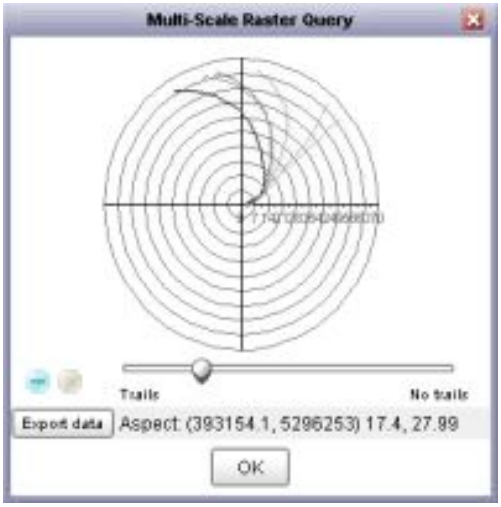


Figure 1.21: Multi-scale aspect query window

- Make the original unsmoothed DEM the primary raster and ensure the display is refreshed.
- Select `Info`→`Multiscale query...` and choose `Aspect` as the surface parameter to examine. Press the `OK` button to display the multi-scale query window (see Figure 1.21).
- Move the new window so it is not overlapped by the main LandSerf window and resize it if it is too small. Try clicking the mouse at various point on the main DEM display. The query window shows how aspect varies with scale. The line drawn on the graph shows the direction of steepest slope measured from 3 x 3 windows at the centre to 65 x 65 cell windows at the outer edge.If the line remains in a relatively constant direction, aspect is said to be scale-invariant at this location. If the line appears to meander as it moves out from the centre, the slope direction must be dependent on the scale at which it is measured. A numerical measure of this dependency is shown in the bottom of the window. The first two numbers give the location of the point being queried, while the second two give the (circular) mean and standard deviation of the aspect measure.

- If you try dragging the mouse over the DEM, the aspect graph is updated continuously. This allows you to investigate the spatial pattern of scale dependency. By moving the 'trails' slider to the left or right, you can control how much of a 'query history' is displayed on the graph. This allows you to explore visually, the spatial variation in scale dependency.
- By dragging the mouse over the main DEM display, identify which parts of the landscape produce scale-invariant measures, and which produce scale-dependent measures. Are there any critical scales at which aspect does not vary over space?

We will use a similar process to explore the scale dependency of other measures.

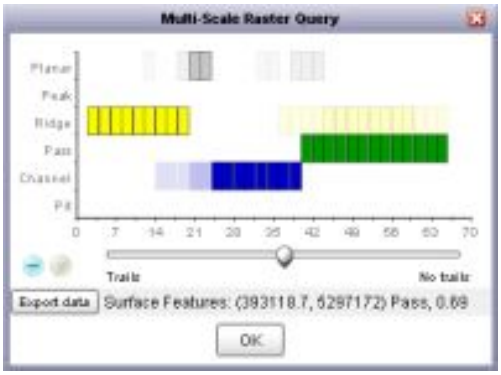


Figure 1.22: Multi-scale feature query window

- Leaving the aspect query window open reselect the `Info->Multi-scale query...` menu item. Choose `Profile curvature` and click or drag various points on the DEM. The maximum size of the query window (representing processing of a 65x65 cell window in this case) is indicated by the black square on the main display. Try to relate the curve plotted and the location and character of the landscape you have queried.
- Repeat this step with some of the other measures, including feature classification (see Figure 1.22) in order to investigate the degree to which their measurement is dependent on the scale of analysis.
- When you have finished, close all the query windows by clicking their `OK` buttons.

One of the advantages of multi-scale processing of terrain is that we can relate our analysis to scales that are of most interest to us. We shall round of this section by calculating the network of surface features at two different scales. First we shall identify the network of ridges and channels at a fine scale:

- Making sure the Lincoln 30m DEM is selected as the primary raster, select `Configure`→ `Window scale...` (or) and set the window size to 5 and the distance decay exponent to `1.0`. This will set the processing back to using a 5 x 5 cell window, but additionally, by specifying a positive distance decay value, we have told LandSerf to give a greater weighting to cells nearer the centre of a window than ones that are further away. The higher the value of this exponent, the greater the importance given to near cells in the calculation of parameters such as slope and aspect.

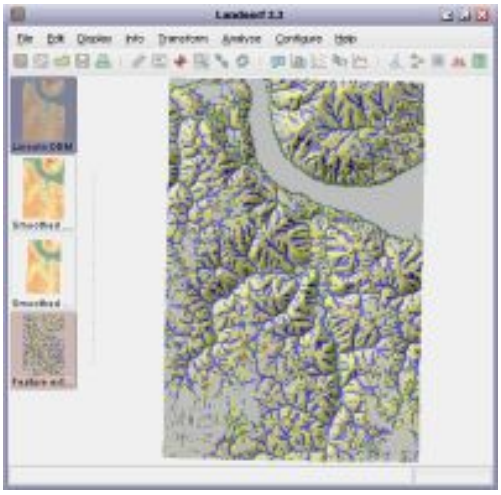


Figure 1.23: Raster feature network at 5x5 window scale

- Select `Analyse->Calculate Surface Parameter` (or ) and this time choose the `Feature network extraction` option. This will will produce a similar classification to the surface features seen previously, but will emphasise the network of channels and ridges (see www.soi.city.ac.uk/~jwo/sdh98 for more details of this process). Even so, you should see that the resultant image is still somewhat discontinuous - both the blue channel network and the yellow ridge network have plenty of gaps.

We can improve the representation of the network by forcing LandSerf to connect some of these ridges and channels together. As this network would describe the *topology* of the surface,LandSerf will represent it using a *vector* rather than raster coverage:

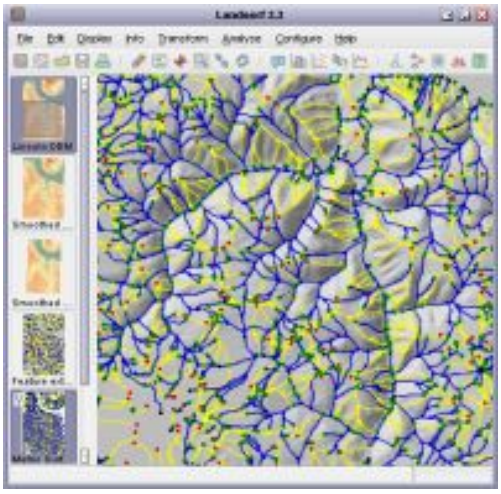


Figure 1.24: Vector feature network at 5x5 window scale

- Select either the `Analyse->Vector Feature Network` menu item or the  button. This

will process the raster network and after a minute or so, will produce a vector network and new thinned raster version of the network.
- To display the vector over the original DEM, unselect the thinned raster network as the secondary raster by right-clicking (ctrl-clicking on MacOSX) on its thumbnail view. Select the `Display->Vector` to see the *metric surface network* (see Figure 1.24). You may wish to use the zoom mode to examine the network in greater detail.
- For more details on Metric Surface Networks, see www.soi.city.ac.uk/~jwo/geoComp2000.

Notice how both the blue channel network and the yellow ridge network appear similarly tree-like.The plateau to the south-west is dominated by local passes, pits and peaks (green, black and red dots).This is because at this scale, errors in the DEM tend to dominate flatter regions. We can broaden our scale of analysis, and therefore reduce the impact of DEM error, by repeating the process using a larger window size:

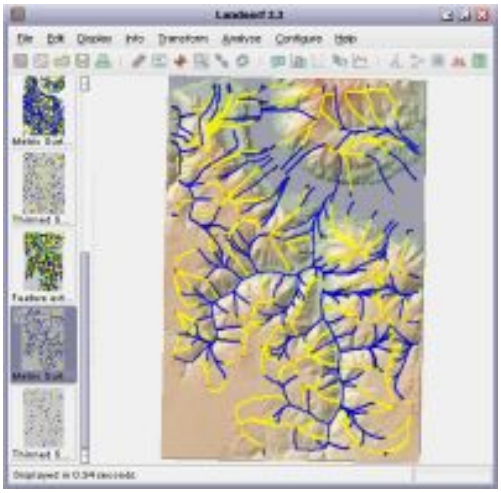
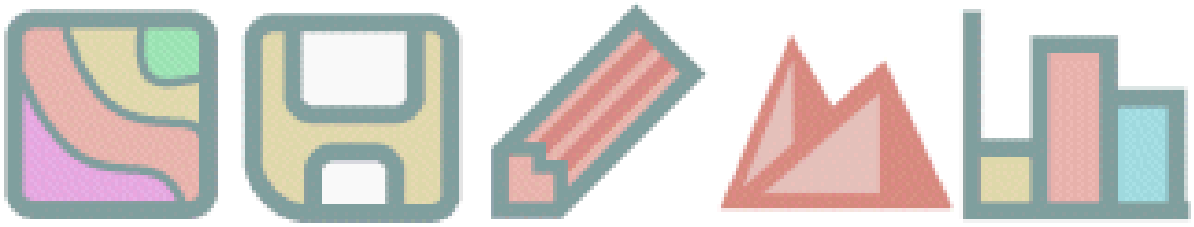


Figure 1.25: Feature network at 25x25 cell scale

- Set the window size to 25 (corresponding to a distance of 25 x 30 = 750m on the ground) leaving the distance decay value at `1.0`.
- Select the `Analyse->Calculate Surface Parameter` menu item and again choose `Feature Network Extraction` and press the `OK` button.
- Select `Analyse->Create Vector Feature Network` ( ) to start the network calculation.
- When complete, you can again display the vector over the original DEM, by unselecting the new raster thinned network and refreshing the display (see Figure 1.25). You can contrast this broader scale network that represents the topology of larger ridges and channels with that of the finer scale network produced previously.
- You may wish to save a copy of the vector network for later analysis. To do this select `File`→`Save...` ( ) and then set the `Files of type` drop-down menu to `LandSerf vector` and save the file with the name `LincolnMSN25.vec`.

2 User's Guide



LandSerf allows you to visualise and perform analysis on spatial data. It is primarily designed to be used with surface and elevation models, but will work with most types of 'GIS' data. It currently supports raster DEM and vector TIN, contour and metric surface network models of elevation. What makes the software unique, is the ability to perform surface analysis over a range of scales, and characterise any scale dependent behaviour.

This chapter takes you though the basics of using LandSerf, from getting data into LandSerf, though editing and analysing data to visualising output.

2.1 Starting LandSerf

Depending on your operating system, you can start LandSerf in a number of ways:

Platform	**Start menu**	**Screen icon**	**Command line**
Windows	Yes	Yes (double-click icon on desktop)	Yes (`LandSerf.bat`)
Linux	No	No	Yes (`landserf.sh`)
MacOSX	No	Yes (double-click the `LandSerf` icon in the Applications folder)	Yes (`landserf.sh`)

By starting from the command line, various command line parameters can also be provided that will start LandSerf with raster and/or vector files preloaded. The options are given below, where names in <*triangular_brackets*> should be substituted with an appropriate file name and the | symbol indicates alternative options.

Command line parameter	**Explanation**
`surface|primary <`*`raster_file`*`>|<`*`vector_file`*`>`	Starts with the given file loaded as a primary object.
`drape|secondary <`*`raster_file`*`>|<`*`vector_file`*`>`	Starts with the given file loaded as a secondary object.
`raster <`*`raster_file`*`>`	Starts with the given file loaded as an unselected raster.
`vector <`*`vector_file`*`>`	Starts with the given file loaded as an unselected vector map.
`display raster|relief|clear|vector`	Starts with the given display type.
`numeric true|false`	Enables or disables numeric display of raster values. By default these are not displayed.
`experimental true|false`	Enables or disables extra experimental functionality (used for testing). By default experimental functionality is turned off.

For example,
`LandSerf primary c:\data\mountains.srf secondary "c:\My Images\photo.jpg" display relief numeric true`
starts LandSerf (in Windows) with the file `mountains.srf` displayed as a shaded relief map with the image `photo.jpg` draped over it and numeric cell values displayed if user zooms in.

`./landserf.sh vector ~/data/coast.vec primary ~/data/landuse.shp display vector`
starts LandSerf (in Unix) with the files `coast.vec` and `landuse.shp` pre-loaded, with the second of these displayed.

You can also start a *LandScript engine* (see LandScript documentation) for editing and processing script files. While this is available from the standard LandSerf application, it can also be started as a separate program from the command line:

Platform	**Command line**
Windows	`landscript.bat [-gui] <`*`landscript_file.lsc`*`>`
Linux	`landscript.sh [-gui] <`*`landscript_file.lsc`*`>`
MacOSX	`landscript.sh [-gui] <`*`landscript_file.lsc`*`>`

For example,

`landscript.bat c:\scripts\combineRasters.lsc`
processes the script `combineRasters.lsc` (on a Windows platform).

`landscript.sh -gui`
starts the graphical LandScript editor on a Linux or MacOSX platform.

If the `-gui` option is used, a window containing the LandScript editor is created. If not, the given file *`landscript_file.lsc`* is processed.

2.1.1 Using LandSerf

The opening screen when starting LandSerf is shown in Figure 2.1. The software is largely controlled via a series of menus or buttons that either perform actions or open further dialogue boxes. Graphics appear in the main area of thewindow, while 'thumbnail' images of all loaded data appear in the left-hand area. The status of the system is reported at the bottom of the window. The entire window can be resized at any stage with all graphics being rescaled accordingly.

For processing that is likely to take some time, a 'percentage complete' bar is displayed in the bottom right-hand corner. Any process can be stopped by clicking on this bar.

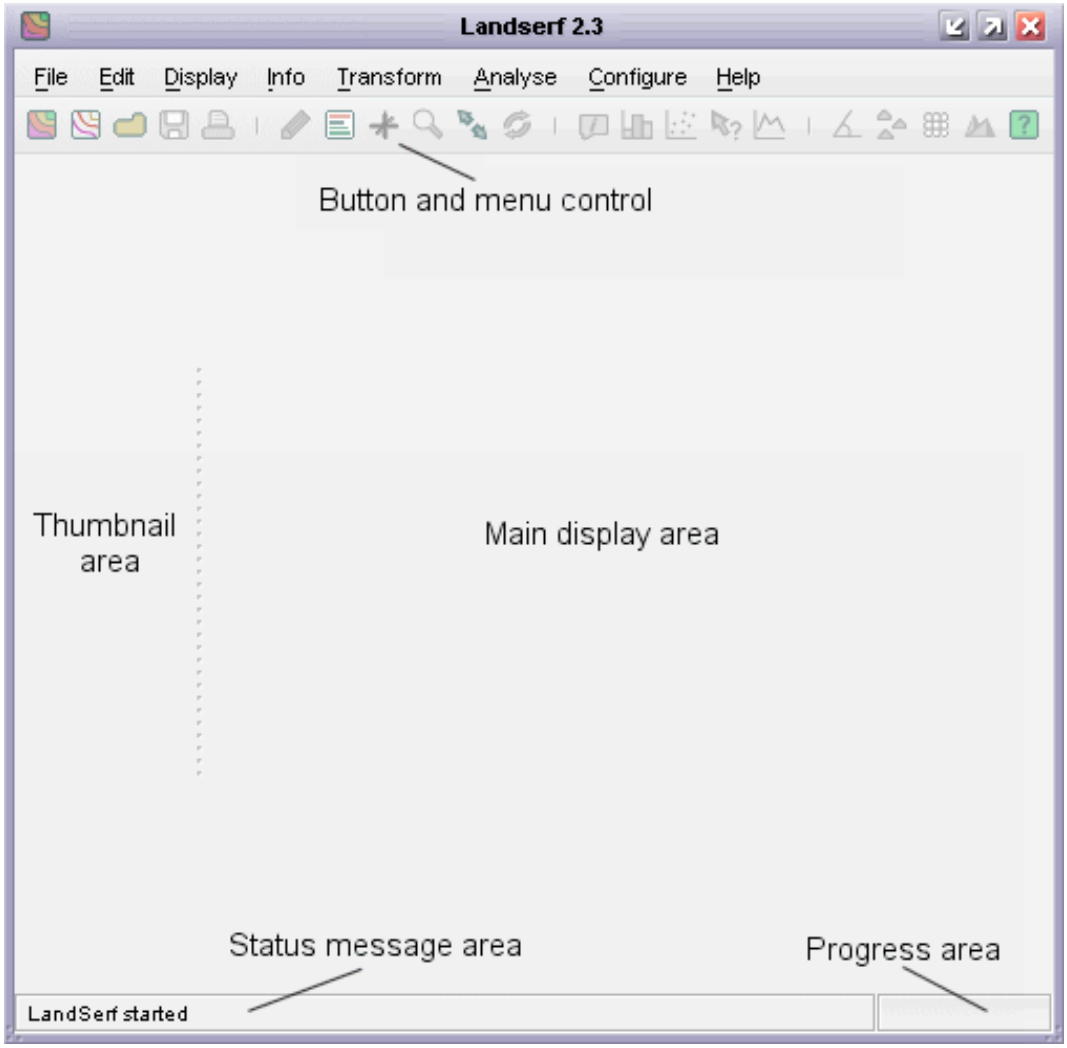


Figure 2.1: LandSerf's graphical user interface.

MacOS X users have the additional option of selecting the native 'Aqua' look and feel. The arrangement of the main windows is identical, but the menu is placed at the top of the screen in common with other MacOS X applications.

LandSerf can store any number of raster or vector maps (known collectively as *spatial objects*), limited only by the memory of the computer. Much of the raster analysis and display is applied to the current *primary raster* and possibly a *secondary raster* (see Figure 2.2. To select a primary raster, click on the relevant thumbnail with the left mouse button. The primary raster is indicated by the highlighted blue thumbnail image. Secondary raster selection can be made by clicking on a thumbnail with the right mouse button or with the left button while holding down the *shift* key. It is indicated by a pink thumbnail background. Note that the vertical order of thumbnails does not determine their primary/secondary status.

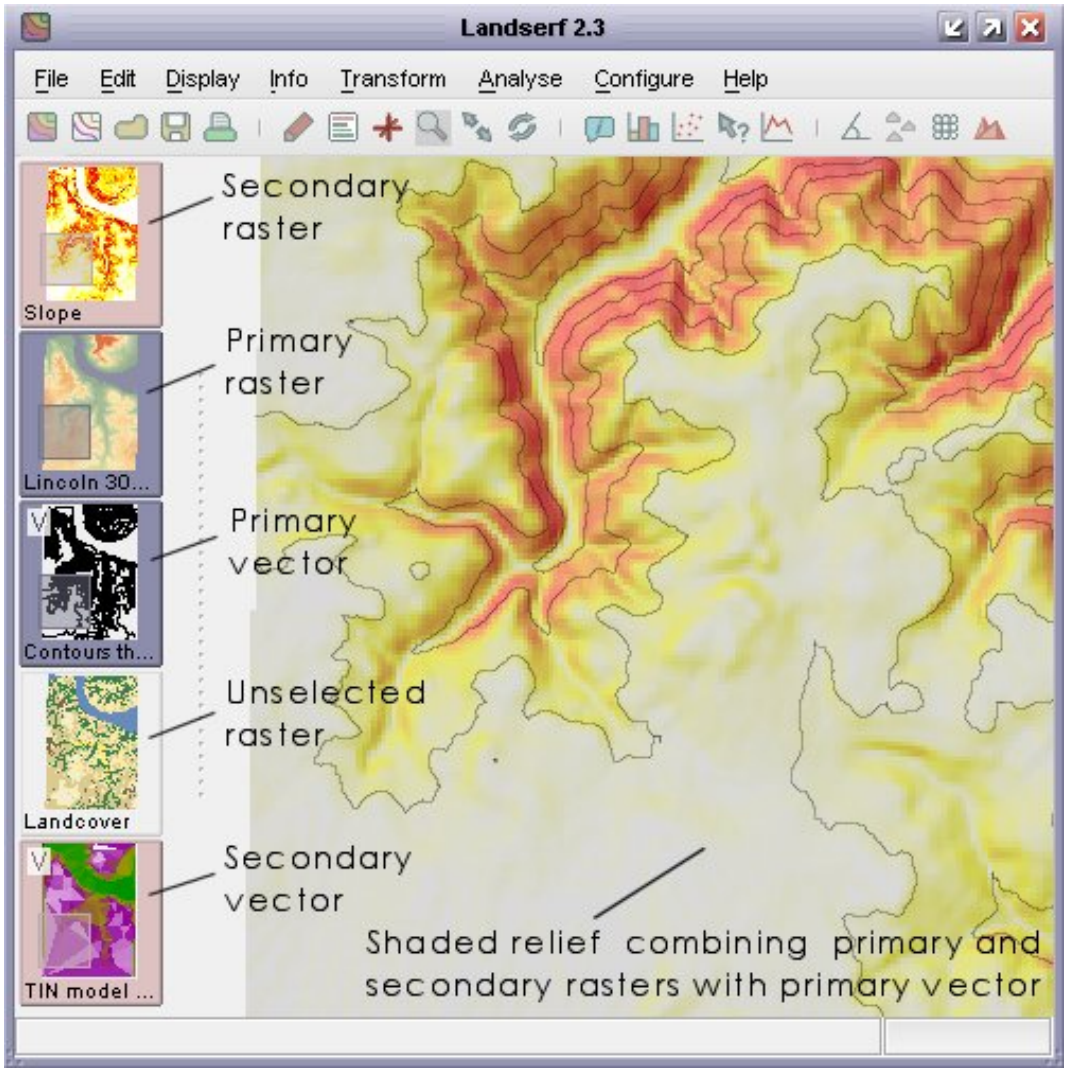


Figure 2.2: Spatial object selection.

The same selection rules apply to vector maps. Primary vectors are selected with the left mouse button, secondary with the right (or mouse button with the shift key held down). Thumbnail images of vector maps can be distinguished by the 'V' symbol in the top-left corner.

When zooming in and panning within the main display area, the current display is highlighted on any primary or secondary spatial objects.

The range of visualisation and analysis options available will depend on the number and type of spatial objects that have been been selected.

2.1.2 Console Output

LandSerf keeps a record of all options chosen by the user as well as any messages displayed in the status bar at the bottom of the screen. This can be useful if you wish to recall a sequence of operations previously undertaken. The record is stored in a file called `landserfLog.txt` in the main LandSerf directory. This file is reset every time a new session is started. Additionally, the logging output is echoed to the *LandSerf console* (see Figure 2.3), which can be viewed at any time by selecting the `Help→Show console` menu option. The console will colour-code output depending on its source. Menu and button selections are shown in grey italics, errors in red, status bar messages in black and script output in blue italics.

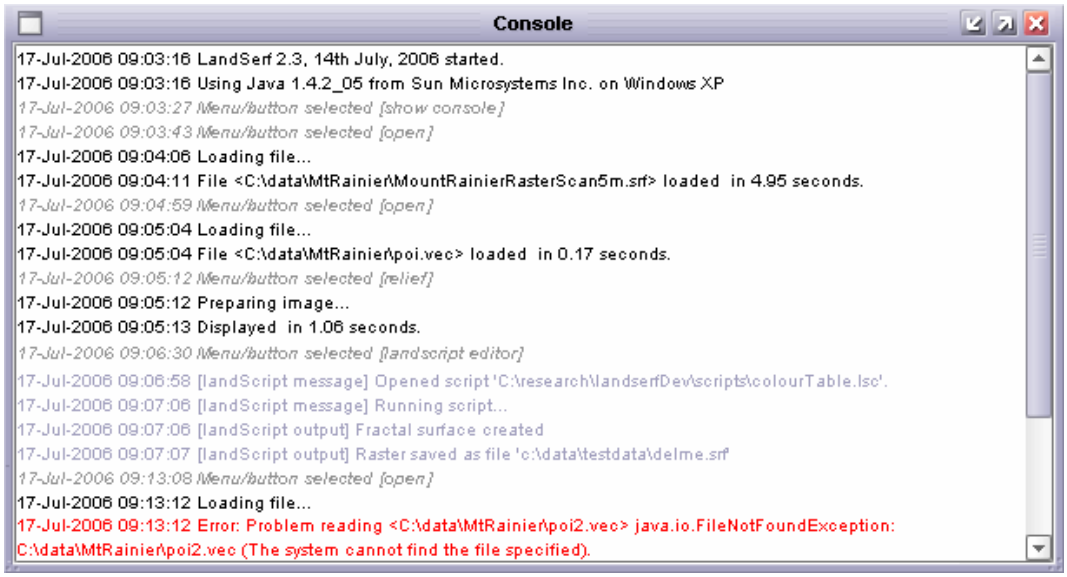


Figure 2.3: LandSerf's console output..

Any errors that stop LandSerf from working correctly are also reported to the console/log file. If these errors appear to be a bug with the program, please report them to the *LandSerfing forum* attaching the `landserfLog.txt` file that reports the error.

2.1.3 Functionality

The following features are currently available in LandSerf. Users of the previous version may also wish to view the changes since the previous release.

Input/Output.

- Raster and vector import in over 30 GIS and other formats - LandSerf's own (platform independent) raster, vector and 3d file formats; ArcGIS rasters and vectors (ASCII grid,

binary images, 'generate' vectors, shapefiles); EDX ASCII DEMs; GRASS vectors and rasters (text only); GPX Global Positioning System waypoints, routes and tracks; Global Hierarchical Shoreline vectors; visualisation software VistaPro (ASCII and binary); Virtual Terrain Project 'BT files'; Terragen (terrain and binary formats); Ordnance Survey NTF DEMs and vectors (NTF 2.0); Ordnance Survey MasterMap vector maps (compressed and uncompressed); generic (ASCII and binary) rasters, vectors and point files; image files (GIF, JPEG, PNG, TIFF and GeoTIFF file formats); USGS native DEM files; GTOPO30 files and SRTM/SRTM30Plus global DEMs.

- Raster and vector export in a range of over 25 GIS and other formats - LandSerf's own (platform independent) raster, vector formats; ArcGIS rasters and vectors (Shapefiles, ASCII grid, binary images, 'gridFloat' and 'generate' files); GoogleEarth KMZ and KML raster and vector output; generic (ASCII and binary) rasters, vectors and point files; GPX Global Positioning System waypoints, routes and tracks; GRASS GIS (ASCII only); image files (GIF, JPEG, PNG, BMP); Scalable vector graphics (SVG); 3d vector flythroughs over elevation models; visualisation software VistaPro (ASCII and binary); Terragen (binary and .ter formats); VRML virtual worlds (compressed and uncompressed); graph output (histograms, scatterplots, profiles etc. can be saved as text files for import by spreadsheets).

- LandSerf files can be printed directly. Output includes the current raster image and vector overlay, titles, bounds and supplementary notes.

- If you have a Garmin GPS receiver and serial cable, you can import Waypoints, Routes and Tracks directly from the unit. Vector objects in LandSerf can be exported to the GPS receiver as waypoints.

Editing Functionality.

- Raster title and supplementary notes.
- Raster boundary positions and resolution including interactive subsetting and reinterpolation.
- Vector map line simplification and line joining
- Vector duplication based on attribute selection
- Raster layer combination.
- Vector layer combination.
- Raster and vector layer combination.
- Vector map title, boundary positions and supplementary notes.
- On-screen digitizing of vector maps.
- Map projection information (projection type and ellipsoid)
- Colour tables including selection from preset palettes.
- Editing of multiple attributes in an attribute table.
- *LandScript* editing.

Display Functionality.

- Interpolated 'k-colour' raster.
- Shaded relief with variable blending with 'k-colour' rasters.
- Interactive zooming and panning
- Coloured vector overlay.
- Vector map labelling
- Blended primary and secondary raster images.
- Hue-Intensity images (primary raster representing hue, secondary representing intensity).
- Hue-Saturation images (primary raster representing hue, secondary representing saturation).
- Numerical overlay of raster cell values
- 3D perspective views with interactive 'flythrough' navigation.

Surface Query.

- Raster map information - title, bounding area, min/max values, colour table, raster type, supplementary notes.
- Vector map information - title, bounding area, min/max values, colour table, supplementary notes.
- Interactive query of raster values, reported as (x,y,z) triplets.
- Interactive query of vector objects including attributes, length (for lines) and area (for polygons).
- Interactive generation of surface profiles
- Interactive query of changing surface parameters/features with scale, reported as a parameter-scale graph.
- Statistical summary of raster data (measures of average, dispersion, spatial autocorrelation and fractal dimension).
- Scatterplot of surface and drape values.
- Log-log variogram of raster values.
- Frequency histogram of surface values.
- 'Hammock Plot' of surface values (detects interpolation artifacts).

Surface Transformation.

- Vector to raster conversion.
- DEM to TIN and TIN to DEM conversion.
- Delaunay triangulation of points.
- Contour threading of DEMs.
- Polygon to centroid transformation.
- Removal of voids from raster maps
- Raster value transformations (scaling, translation etc.) including generation of null values
- Interactive rectification of rasters with RMSE error reports.
- Projection between lat/long, UTM, OSGB, French, Canadian and Swiss National Grid coordinate systems .

Surface Analysis.

- Calculation of elevation, slope, aspect, 7 measures of curvature, and feature type at any scale.
- Average and variation of any surface parameter/feature over a range of window scales.
- Surface feature network derivation (creates linear networks of ridges and channels bounded by pits, passes and peaks).
- Flow accumulation and channel identification.
- Peak detection using relative drop methods.
- Hydrologic flow accumulation and drainage basin calculation/
- Pit removal from surfaces.
- Fuzzy feature classification.

LandSerf Configuration.

- User defined selection of analysis scale and distance decay parameters.
- User defined selection of feature extraction tolerances.
- Shaded relief parameter settings (sun elevation and azimuth, reflectance properties etc.)

2.2 Getting Data In and Out of LandSerf

LandSerf uses its own platform independent compressed file format for storing surface models. Opening and saving of files in this format is achieved via the `File`→`Open` and `File`→`Save` menu items or the and buttons respectively. When saving a file to disk, it is the *primary raster map* or *primary vector map* that is output.

Import and export of files to and from LandSerf can be in a range of formats. See file formats used by LandSerf for more details. You may also wish to view some guidelines on importing elevation models to help with this process.

The file format for both importing and exporting data is selected from the File dialogue in the `Files of type:` drop-down menu (see Figure 2.4). When typing the name of a file to save, if a file extension is not provided (e.g. `.srf`, `.shp` etc.), one will be added automatically. Note however that in some cases, the type of file extension will determine the precise format of the output file (e.g. image files can have the extension `.jpg`, `.gif` or `.png`; VRML files can will be uncompressed if given the extensions `.wrl` or `.vrml`, but compressed if given the extension `.wrz`).

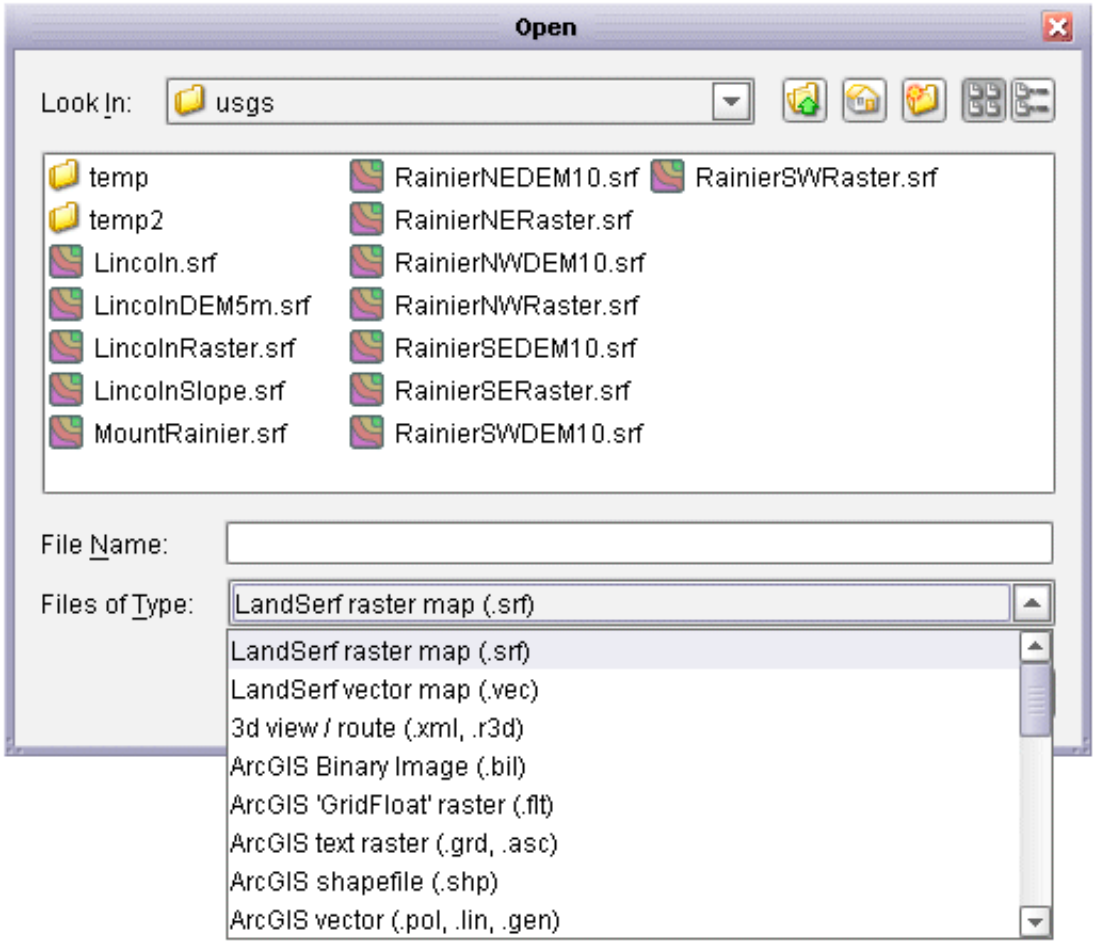


Figure 2.4: File type selection.

Any number of raster or vector maps may be imported into LandSerf, limited only by the memory capacity of the machine running the software. Multiple files of a given type can be imported in a single operation by shift-selecting a range of files in the file chooser. Maps for processing or display can then be selected by clicking on the relevant thumbnail image on the left-hand side of the LandSerf window.

2.2.1 GIS Files

LandSerf can import and export rasters as ArcGIS ASCII or binary (BIL) files. Even if you do not have access to ArcGIS or ArcView, you may find these facilities useful for sharing data. ArcGIS vector files can also be imported and exported as shapefiles and ASCII 'generate' vector coverages and attribute tables.

If you have access to the GIS GRASS, both rasters and vectors may be exported using the GRASS modules `r.out.ascii` and `v.out.ascii`. These files may be read directly into LandSerf if they have been given a `.txt` filename extension. Note that to import vector attributes as well as geometry, you will need the relevant file from GRASS' `dig_att` file in addition to the output produced by `v.out.ascii`.

Ordnance Survey Raster DEMs at both 10m ('Profile data) and 50m ('Panorama' data) resolutions may be read using the `Ordnance Survey NTF raster DEM` format. Ordnance Survey vector data (e.g. LandLine, Strategi, Meridian and MasterMap) can also be imported using either the `Ordnance Survey NTF vector file` or `Ordnance Survey MasterMap` formats. On import of vector data with multiple attributes, you are presented with the option of importing only a subset of these attributes (see Figure 2.5). For convenience, attributes of given types may also by chosen by selecting from the `Points`, `Lines`, `Areas`, `Symbols` and `Text` list at the bottom of the window. If importing MasterMap data, you have the option of assigning an individual ID to each object by selecting `Store object IDs as attributes`. Note that doing so can produce very large attribute tables so should only be used if a relatively restricted number of object types are to be imported. Once imported, data are stored in LandSerf's own attribute table that may be edited later.

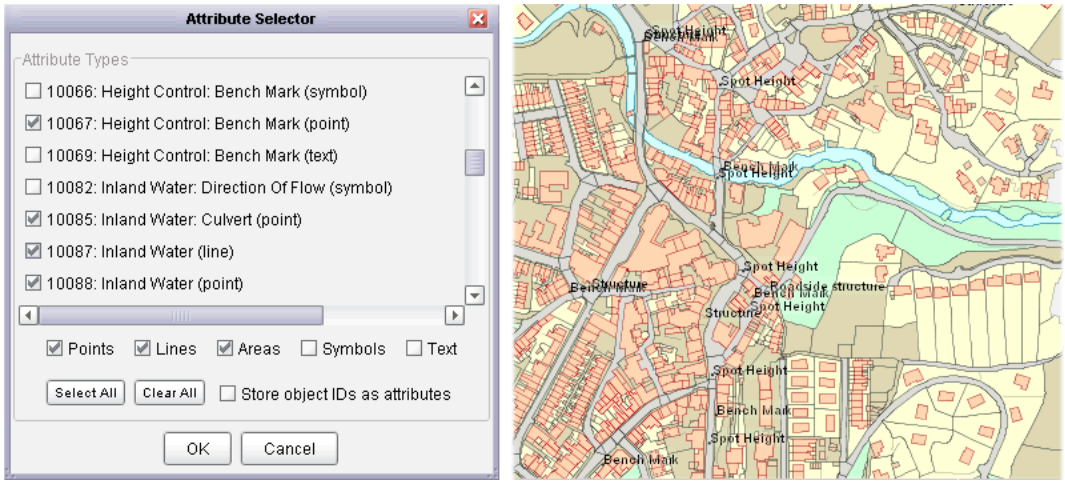


Figure 2.5: Ordnance Survey attribute selection and Mastermap output.

Most elevation data from the US Geological Survey (USGS) are now distributed in one of the ArcGIS raster formats. However, some are still distributed in 'Native DEM' of DTED format. These data tiled to either 1 degree or 7.5 minutes can be imported directly by selecting either the `USGS Native DEM` or `DTED raster` formats.

Image data such as areal photos, rasterized scanned maps and remotely sensed images can be imported directly using the `image` format selector. Formats available for import include GIF, JPEG, PNG, TIFF and GeoTIFF (LZW compression not yet supported). Georeferencing information may be stored in a 'world' file sharing the same name as the image, but with a modified extension name as indicated below:

Image name	**World file name**
photo.tif	photo.tfw
map.gif	map.gfw
landcover.jpg	landcover.jgw
band7.png	band7.pgw

To import other raster text files, use the `Generic text raster` format option. Files in this formatconsist of text of *nrows* rows running North to South, with each row consisting of *ncols* columns of whitespace separated z-values. These numbers may be integer or real. There should be no header information associated with the file, although lines starting with a # symbol may be used as comments and are ignored. Note that to import ASCII data that run row-wise from South to North use the `VistaPro raster` format instead. To import text rasters stored column-wise from South to North, select the `EDX` format.

The Shuttle Radar Topography Mission provides freely available elevation data for most of the globe. These are available at a 3 arc-second resolution, which corresponds to about 90m in an East-West direction at the equator and approaching 50m for northern Europe. An SRTM 'hgt' file can be imported into LandSerf by selecting the SRTM format from the file dialogue window. If importing the older '*Version 1*' (uncorrected) data, you may wish to remove any voids from the surface by selecting the `Transform→Void removal` menu item. After removal, the surface can be reprojected to a more useful coordinate system by selecting the appropriate option from the `Transform→Reproject` menu. For further details, see LandSerf's How to import SRTM data guide.

Other freely available global elevation data including GTOPO30, SRTM30Plus and ETOPO2 bathymetry can also be directly imported into LandSerf by selecting the appropriate file format.

If you wish to share data with software (or data) that use (x,y,z) coordinate triplets to locate elevation values (e.g. a spreadsheet), you can use the import/export `Generic Point File` formats. Export converts the primary raster values into a set of (x,y,z) triplets row-wise from North to South. Import will create a series of dense vector points that can be converted into a raster using `Transform→Vector to raster`.

2.2.2 Virtual Reality and Graphics Software

Both raster and vector maps may be exported as *KML* files for display in GoogleEarth. Data must be using `Latitude/Longitude WGS84` before they can be exported to KML (select `Transform→`

`Reproject...` if they are not in this coordinate system). To save a vector map, select `KML Vector` as the file type in the `Save` menu. To save a raster map, select `KML Archive (KMZ)` which will save the current screen display as a raster overlay displayable in Google Earth (see Figure 2.6).

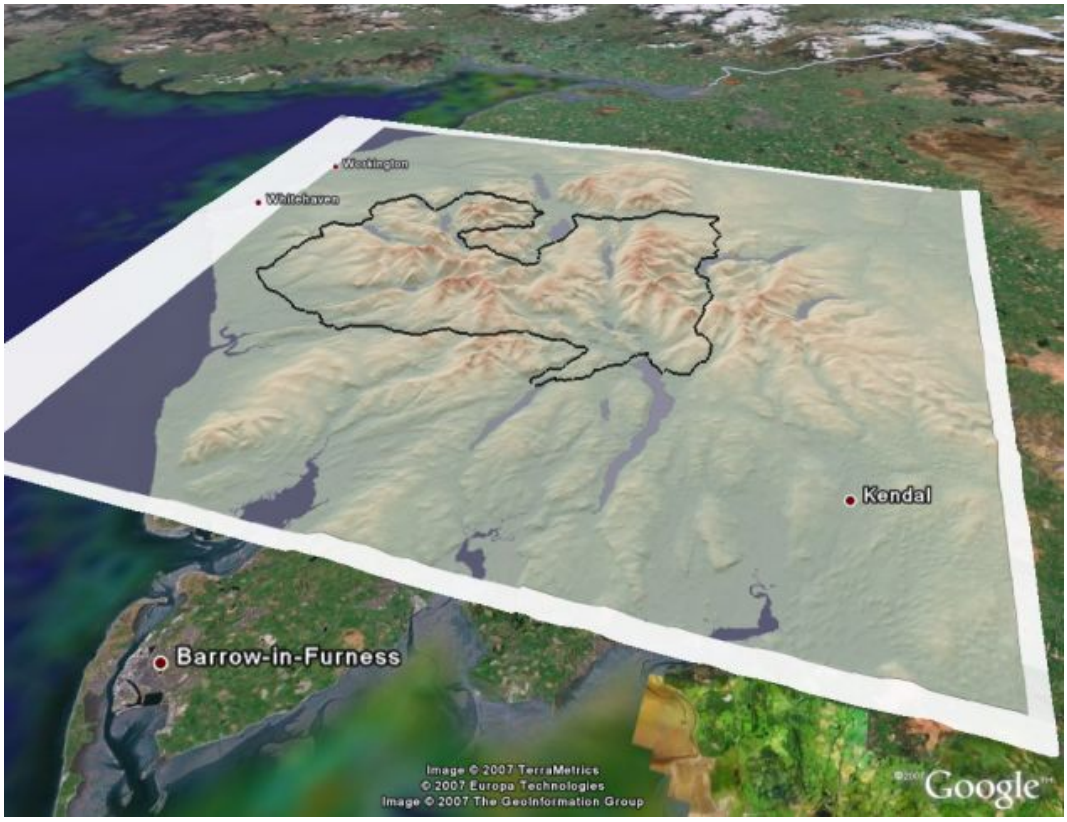


Figure 2.6: KML vector and KMZ raster overlay displayed in GoogleEarth.

While LandSerf contains its own 3d viewer for interactive exploration of surfaces, you may also wish to export a surface as a VRML world so that it may be viewed by common VRML browsers. Selecting the `VRML world` formatwill export the primary raster surface and create a simple VRML world in which to display it. Figure 2.7 shows such a file viewed by the Cortona VRML browser. Note that to store the VRML file in compressed format, the file extension `.wrz` should be used. If you do not wish to compress the file (some older VRML viewers cannot process compressed files), choose either `.wrl` or `.vrml` as a file extension.

If you have access to the landscape visualisation software Terragen or VistaPro or the you can import and export rasters directly to and from LandSerf. To export from VistaPro, choose the finest polygon resolution (1), and select `Save ASCII Z` from the `Save` menu. To export or import files to or from Terragen, rasters are scaled to the nearest 'power of 2 plus 1' (e.g. 257x257, 513x513, 1025x1025) pixels by adding a padding border of zero values where necessary.

LandSerf will also import the binary 'BT' raster format used by the Virtual Terrain Project. Again, even if you do not use any of the VTP software directly, this can be a useful DEM interchange format for exchanging data.

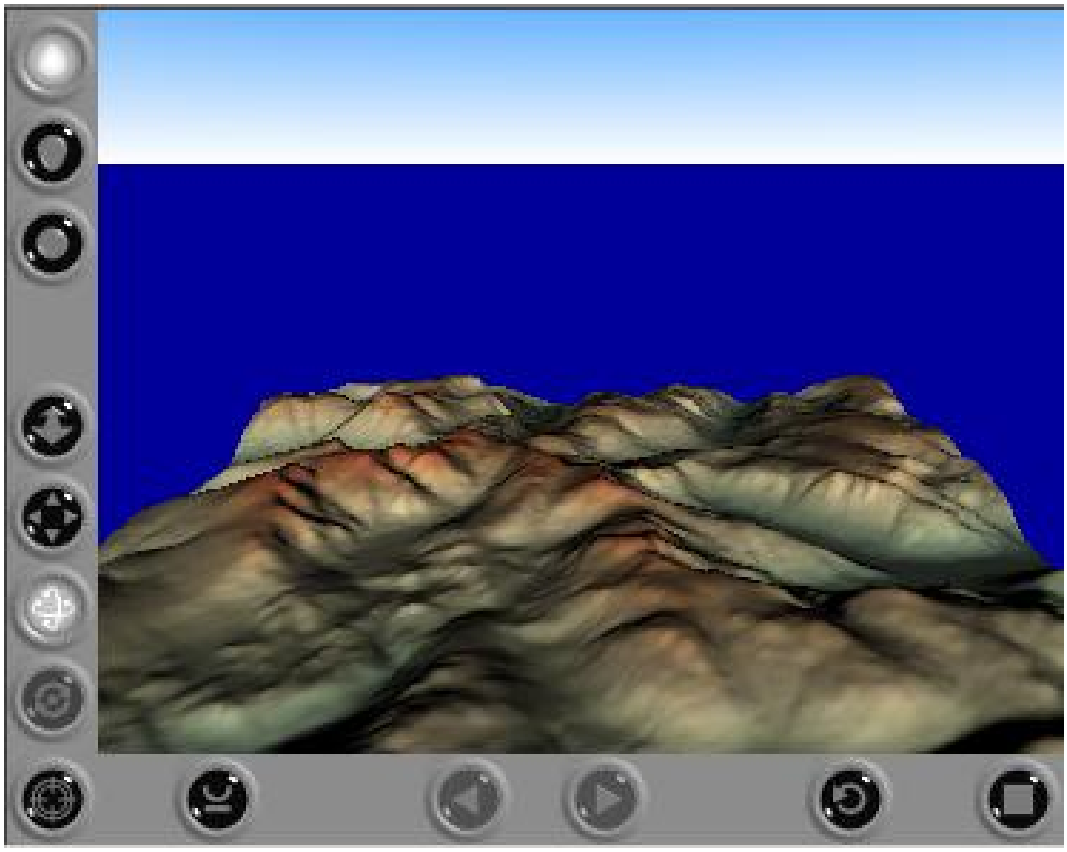


Figure 2.7: VRML output from LandSerf displayed in the Cortona VRML client

2.2.3 Graphical Export

The spatial objects currently displayed in LandSerf's main window can be saved as a GIF, PNG or JPEG graphics file by selecting the `Image` format from the save dialogue. The image dimensions will be determined by the full extent of any raster or vector maps displayed and will be independent of any particular zoomed or panned view. Note that if a vector map is displayed over a raster, and its extent is significantly wider than the raster, the resulting image can be quite large, possibly leading to memory problems. To reduce the risk of such problems, consider subsetting the vector map to be closer in extent to the raster.

Vector data may be exported in SVG format for sharing on the Web. For browsers with a suitable SVG plug-in, web-based maps may be created that allow users to zoom and pan while maintaining high quality output. This is also useful for producing high quality printed output. Data may be compressed depending on whether a `.svgz` (compressed) or `.svg` (uncompressed) extension is supplied.

2.2.4 Printer Output

LandSerf can print summary sheets of the current raster and vector display as well as output from the 3d viewer. 2D print output consists of the display in the main window and the title of

the spatial objects displayed (see Figure 2.8). A preview of the printed page can be viewed using `Print Preview` from the `File` menu. Printer output can be selected from the same menu or the  button.

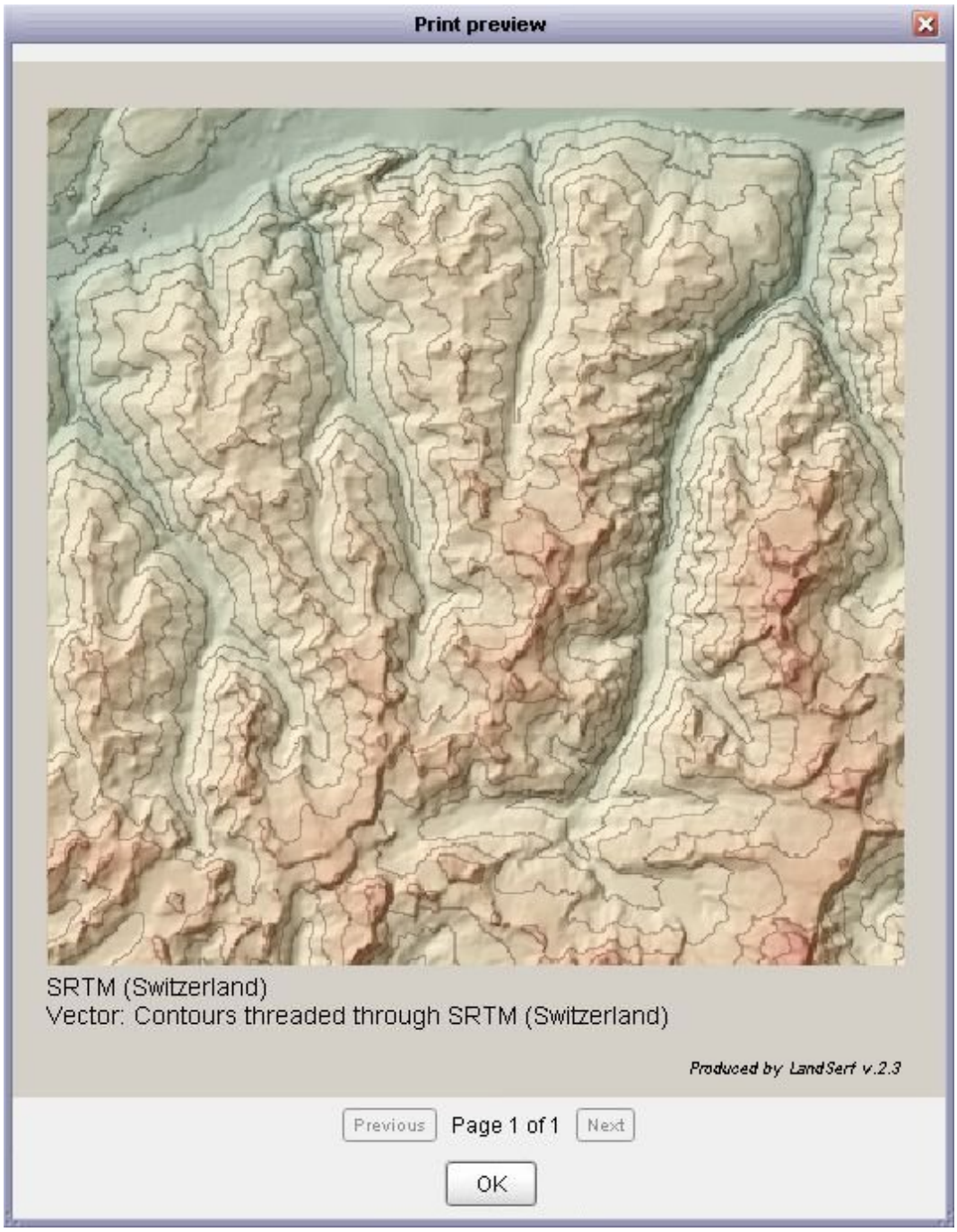


Figure 2.8: Print preview window.

2.2.5 GPS Input and Output

If you have a Garmin GPS (Global Positioning System) receiver and a serial or USB connection with your computer, you can import and export vector data directly from and to the device. First ensure that the receiver is connected to your computer and switched on. Select the `Configure→GPS...` menu item and then press the `Scan for GPS` button (see Figure 2.9). After a pause of a few seconds, LandSerf should detect the GPS device and the port used to connect to it. If you have more than one GPS connected to your computer, select the one you wish to communicate with using the appropriate button.

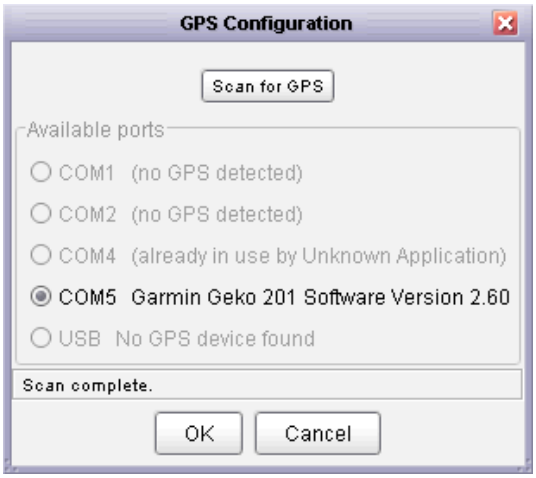


Figure 2.9: GPS detection window.

After configuring the GPS connection, you will be able to import waypoints, routes and tracks from the GPS or export waypoints to it. Selecting the `File→Import from GPS...` menu item will bring up a window similar to the one shown in Figure 2.10. You can choose to download waypoints (sets of named point locations), tracks (lines representing the location of the GPS recorded at regular intervals) or routes (an ordered collection of waypoints defining a path) by clicking the appropriate button in the window. Depending on the volume to be downloaded, importing data can take from a few seconds to several minutes. Progress is reported in the status bar at the bottom of the window.

Once downloaded you can choose which items are stored as a LandSerf vector by selecting the appropriate tick box. By default, position data are downloaded as global latitude/longitude coordinates. You can reproject directly into OSGB or UTM coordinate systems by selecting the relevant tick box. Alternatively, you can reproject later using the `Transform→Reproject` menu. If you have downloaded any tracks from the GPS, you have the additional option of saving a detailed 'position/time' file which records (x,y) position, elevation (if recorded on the GPS receiver) and time. These data can be analysed in a spreadsheet or reimported into LandSerf as a collection of vector points (see Figure 2.10).

If you have a LandSerf vector containing point values, these can be exported to the GPS as waypoints. This can be useful for route planning by, for example, digitizing key points along a route in LandSerf with a raster map as backdrop. To export waypoints, select the `File→Export to GPS...` menu item to bring up window similar to the one shown in Figure 2.11. If the

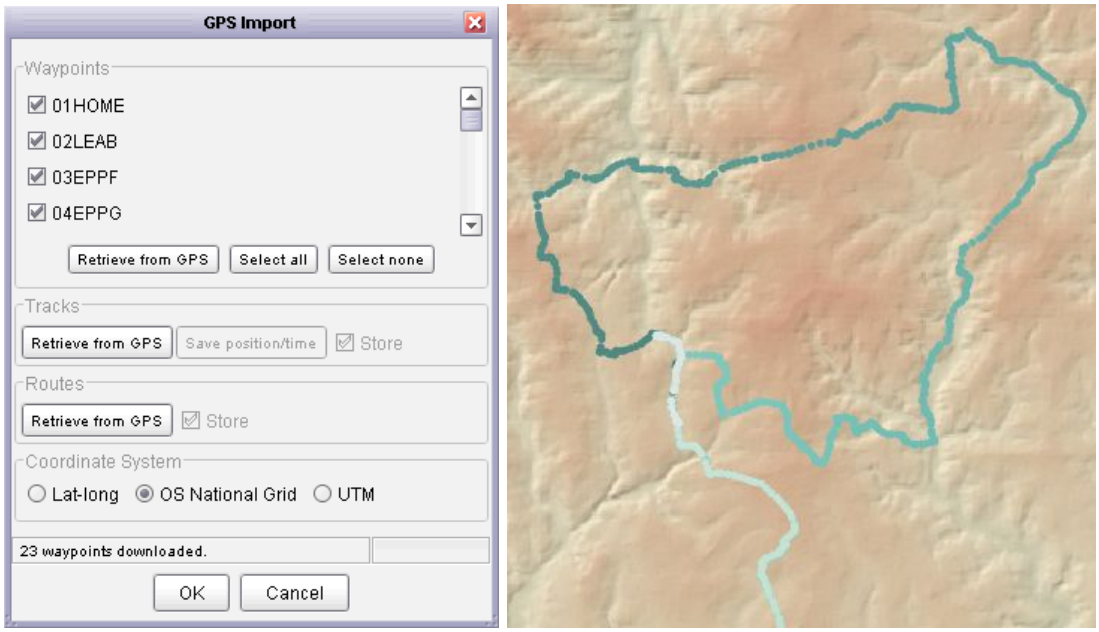


Figure 2.10: GPS import (left) and display of (x,y,time) points along GPS route (right).

currently selected vector contains point values, these will be identified in the window as available for upload. Note that only vector maps with a defined map projection can be exported to the GPS.

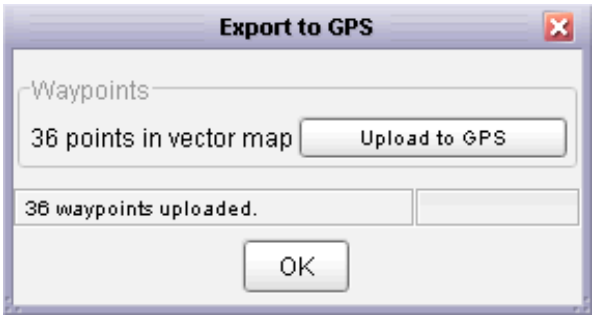


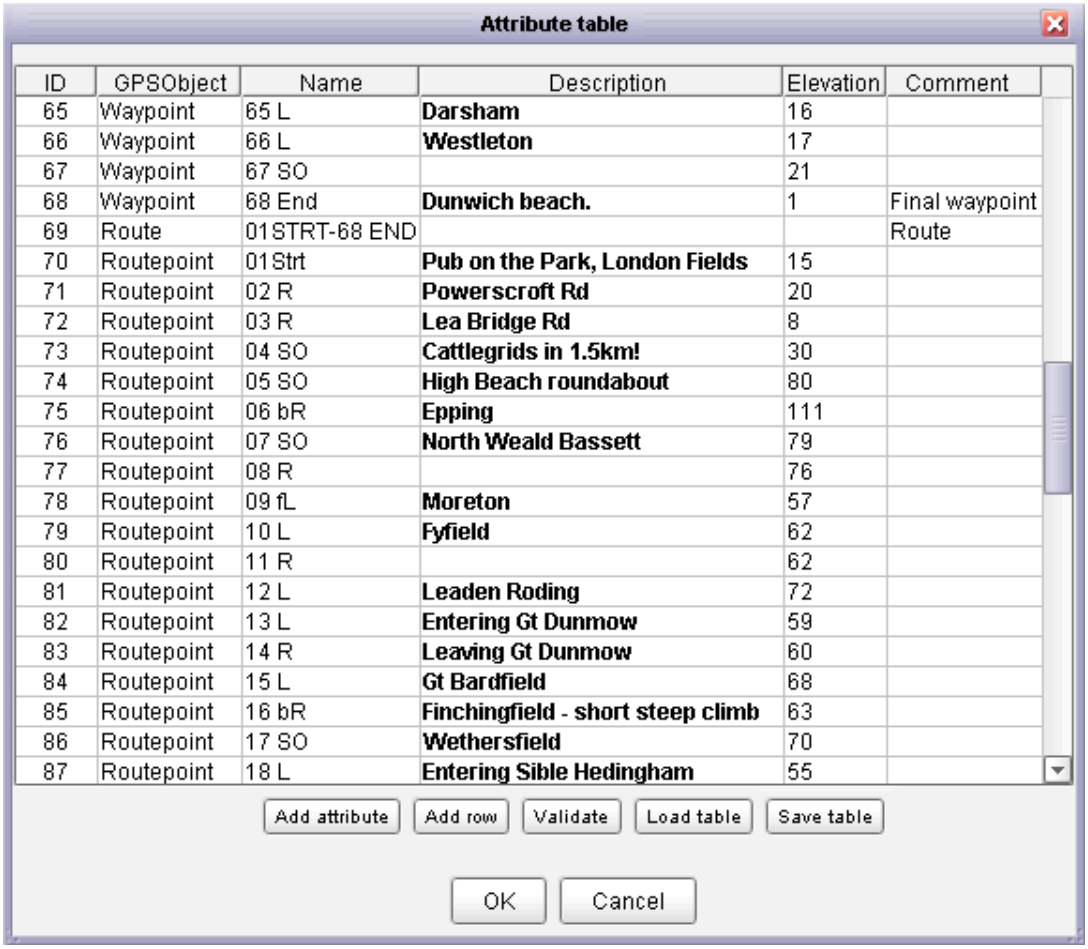
Figure 2.11: GPS waypoint export.

GPX Import and Export

Data can also be input/output with a GPS by using GPX interchange files. Most software that handles GPS data can read and write GPX files. Importing a GPX file into LandSerf is simple and is achieved by selecting `GPS interchange file (.gpx)` from the file dialogue when opening a file. Vector maps imported this way will always use the *Lat/Long (WGS84)* coordinate system. They can be reprojected into other coordinate systems using `Transform→Reproject...`.

Exporting a vector map into GPX format requires the exportable vector to be in some defined map projection. By default, all points within a vector map are assumed to represent GPS

waypoints and all lines are assumed to represent GPS *tracks*. Area objects are not exported as part of the GPX file. Creating an attribute table (see Creating editing and transforming data and Figure 2.12) allows more precise control over the GPX export.



ID	GPSObject	Name	Description	Elevation	Comment
65	Waypoint	65 L	**Darsham**	16	
66	Waypoint	66 L	**Westleton**	17	
67	Waypoint	67 SO		21	
68	Waypoint	68 End	**Dunwich beach.**	1	Final waypoint
69	Route	01STRT-68 END			Route
70	Routepoint	01Strt	**Pub on the Park, London Fields**	15	
71	Routepoint	02 R	**Powerscroft Rd**	20	
72	Routepoint	03 R	**Lea Bridge Rd**	8	
73	Routepoint	04 SO	**Cattlegrids in 1.5km!**	30	
74	Routepoint	05 SO	**High Beach roundabout**	80	
75	Routepoint	06 bR	**Epping**	111	
76	Routepoint	07 SO	**North Weald Bassett**	79	
77	Routepoint	08 R		76	
78	Routepoint	09 fL	**Moreton**	57	
79	Routepoint	10 L	**Fyfield**	62	
80	Routepoint	11 R		62	
81	Routepoint	12 L	**Leaden Roding**	72	
82	Routepoint	13 L	**Entering Gt Dunmow**	59	
83	Routepoint	14 R	**Leaving Gt Dunmow**	60	
84	Routepoint	15 L	**Gt Bardfield**	68	
85	Routepoint	16 bR	**Finchingfield - short steep climb**	63	
86	Routepoint	17 SO	**Wethersfield**	70	
87	Routepoint	18 L	**Entering Sible Hedingham**	55	

Figure 2.12: Attributes used by GPX export.

To control GPX export, the attribute table must have a column header called `GPSObject`. This defines the the type of object to be associated with a point or line of the given `ID`. Attributes in this column must be one of `Waypoint`, `Routepoint` or `Trackpoint` (for points) or `Route`, `Track` or `Track segment` (for lines). If any of the column names include `Name`, `Description`, `Elevation` or `Comment`, the values in these columns will be transferred to the GPX file.

2.3 Creating, Editing and Transforming Data

2.3.1 Creating New Raster Data

It is possible to create new raster surfaces by selecting `New Raster...` from the `File` menu, by pressing the  button, or by using the shortcut key `Ctrl-N`. You will be presented with a dialogue window allowing the bounds, resolution and map projection of the new raster to be changed (see Figure 2.13). The default settings create a raster with 100 rows and 100 columns. The new raster can be either a polynomial expression or a fractal surface with a user-defined fractal dimension.

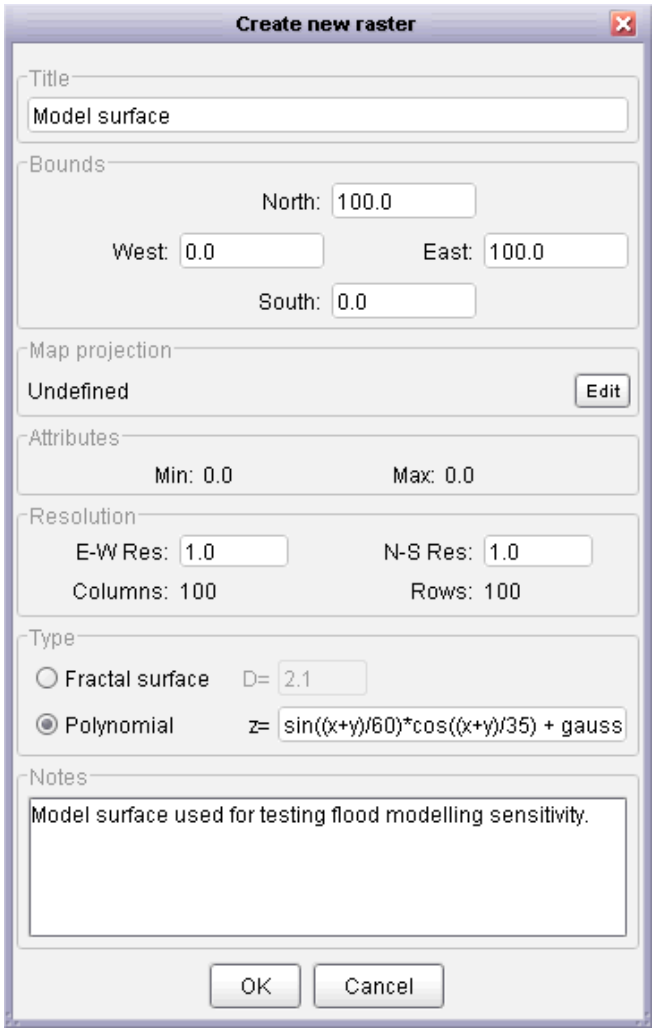


Figure 2.13: New raster window using polynomial expression and metadata.

For modelling and simulation purposes it is sometimes useful to create artificial surfaces with known characteristics. By selecting `Polynomial` from the `Create New Raster` window, you can define surfaces using variables and functions. Some of the more common are identified below. For a full list see the LandScript function reference.

Expression	**Explanation**
`x`	the x position of each raster cell. x is scaled to be +- nCols/2, where nCols is the number of columns in the raster
`y`	the y position of each raster cell. y is scaled to be +- nRows/2, where nRows is the number of rows in the raster
`z1`	the value of any given cell in the primary raster.
`z2`	the value of any given cell in the secondary raster. This will only be meaningful if a secondary raster has been selected; if it does not, z2 will always return 0.
`pi()`, `e()`	The constants PI (3.1459) and *e* (2.7183)
`+`, `-`, `*`, `/`, `^`	Addition, subtraction, multiplication, division and power operators.
`cos()`, `sin()`, `tan()`	trigonometrical functions. Expects angles to be given in radians (to convert from degrees into radians multiply by pi()/180 or 0.01745329). Values of infinity (e.g. `tan(pi/2)`) are returned as 0.
`acos()`, `asin()`, `atan()`, `atan2()`	inverse trigonometrical functions. Returns values in radians (to convert from radians into degrees multiply by 180/pi() or 57.2957795). Values outside of the range of +-1 supplied to these functions will return a value of 0.
`sqrt()`	Square root. If the value given to the function is negative, the function will return a 0.
`ln()`	Natural logarithm (to base *e*). If the value given to the function is <=0, the function will return a zero.
`rand()`	Random value with 'rectangular' distribution between 0 and 1.
`gauss()`	Random value with Gaussian (normal) distribution with mean of 0 and standard deviation of 1.
`ifelse(`*`condition`*`,` *`value_if_true`*`,` *`value_if_false`*`)`	Evaluates the expression condition. If the condition is true or evaluates to a non-zero value, returns *`value_if_true`* otherwise returns *`value_if_false`*.

Example	**Explanation**
`x+y`	Creates a plane sloping from bottom right to top left.
`0 - (x^2 + y^2)`	Creates a convex-up dome.
`z1-z2`	Creates a difference map of the differences between the primary and secondary rasters.
`100 - sqrt((x*y*sin(x*y/200) * cos(y/30))+(x^2+y^2))`	Complex polynomial (central peak with surrounding valleys).
`z1 + (gauss()*10)`	Adds a random gaussian value with mean of 0 and standard deviation of 10 to each cell in the primary raster.
`ifelse(z1<10,10,z1)`	Creates a 'flooded' version of the primary raster with the 'water level' set to 10 vertical units.

Expressions that are incorrectly specified (e.g. using unknown functions, or failure to close brackets) are highlighted in the edit window and prevent it from being closed.

By selecting `Fractal` from the `Create New Raster` window, more realistic terrain surfaces may be created for modelling and simulation purposes. The roughness of such surfaces can be controlled by entering a fractal dimension between 2.0 (smooth) and 3.0 (very rough). Typical landscapes have fractal dimensions of around 2.1.

These may also be combined with smoother polynomial surfaces, as shown in Figure 2.14. This can be achieved by creating a fractal surface and a separate a polynomial surface and finally creating a new polynomial as `z1+z2`, thus combining the two simulated fields.

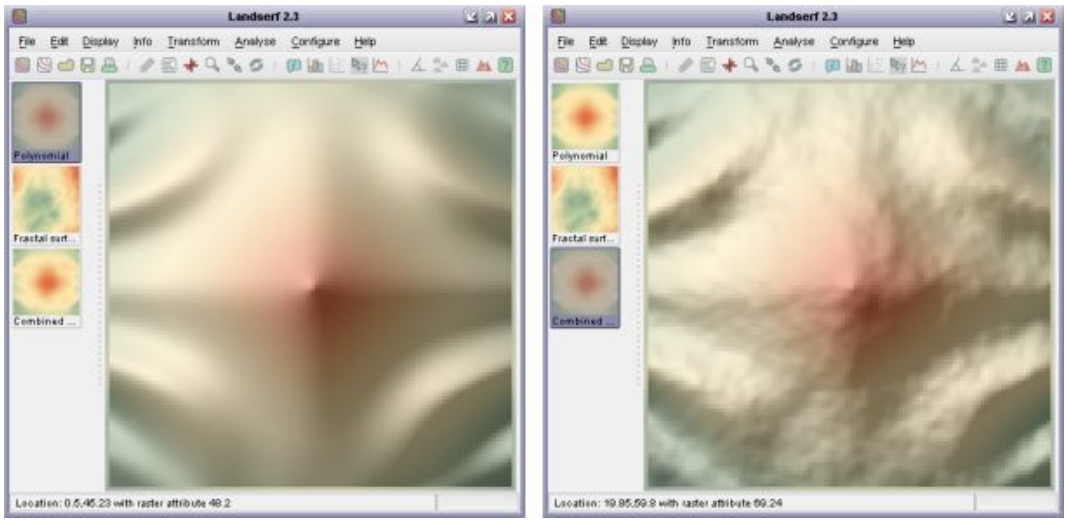


Figure 2.14: Polynomial and combined polynomial and fractal surfaces.

2.3.2 Creating New Vector Data

New vector maps can be created either by selecting `New Vector...` from the `File` menu or by pressing the button. You will be presented with a dialogue window allowing the bounds of the new map to be changed. Initially this vector map will be 'empty', but it is possible to add vector objects through on-screen digitizing.

To digitize new data, make sure a vector map is selected (either an empty vector map as described above, or an existing one to which manually digitized objects are to be added), then select either the `Edit`→`Digitize Mode` menu option or the button.

You will then be presented with a small window (see Figure 2.15) that provides you with the option of creating vector points, lines or areas. By clicking in the main LandSerf window with the left mouse button, coordinates can be added to a new vector object. The numeric attribute to be associated with that object can also be defined at this stage. If the `auto-increment` option is selected, the numeric attribute associated with the object will be increased by one every time an object is digitized. This can be particularly useful if you wish to digitize a series of waypoints. To store the digitized points, press the `Store object` button. Pressing `Clear`

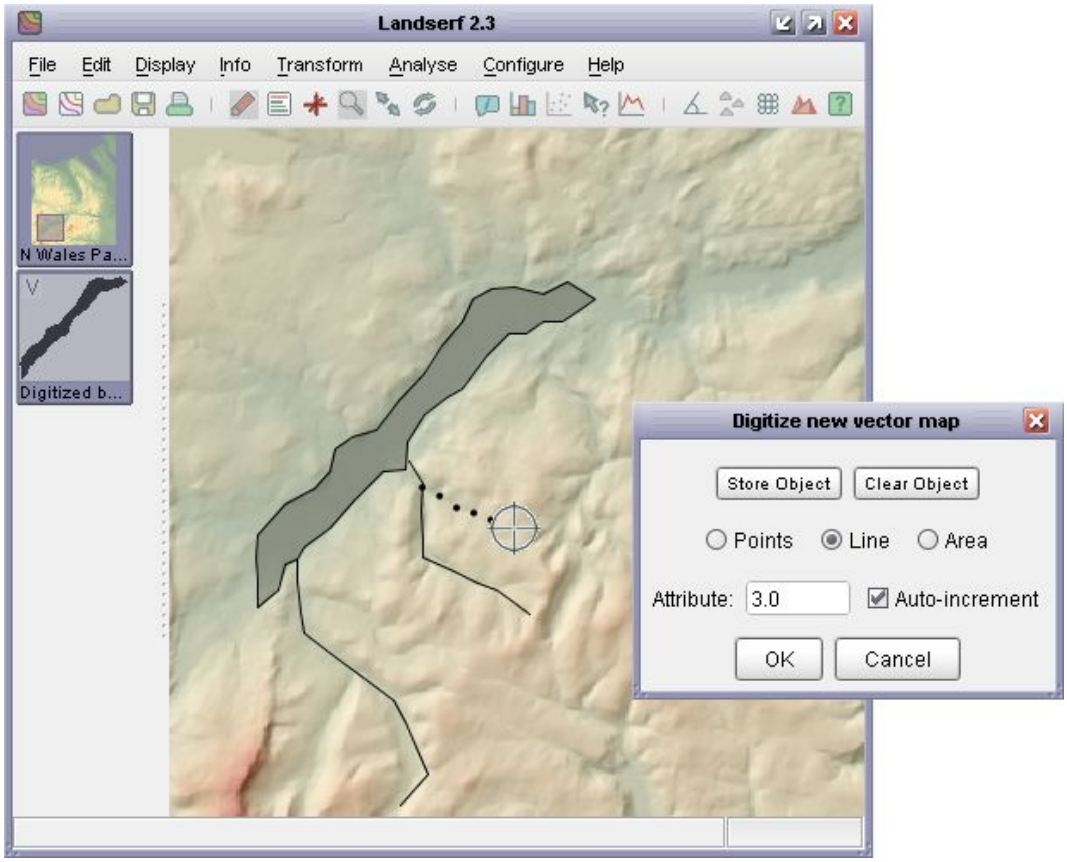


Figure 2.15: Digitizing new vector objects.

`object` will clear the points in the currently digitized object (but not the vector map as a whole). Pressing `Cancel` will quit from digitize mode without saving any of the newly digitized objects.

To make the process of digitizing easier and more precise, any raster data can be displayed on screen while digitizing, and the main display can be panned with the right mouse button (or <shift> left-click). If you wish to zoom in or out, you can use the mouse wheel or temporarily toggle between *digitize mode* and *zoom mode* by selecting the and buttons or the appropriate menu options.

Complex polygons with holes and islands can also be digitized. To do this, digitize the outer boundary first, making sure it is stored as an area object by pressing the `Store Object` button. To cut a hole in the polygon just created, ensuring that the attribute matches that of the existing polygon, digitize the boundary of the hole, but with the `Control` key pressed down. Once this hole has been created, pressing the `Store Object` should create the hole (see Figure 2.16).

It is often easier to digitize if before selecting the digitize option, you set the `Display`→`Vector appearance...`to display point labels, a point size of about 3.0, and line width of about 2.0. This will make it easier to see the objects you have already digitized along with their attribute values.

Figure 2.16: Digitizing complex polygons with holes and islands.

2.3.3 Combining Data

Combing Raster Data

It is possible to combine the primary and secondary raster into a single raster layer by selecting the `Edit→Combine rasters...` menu option. This allows rasters that do not have the same spatial extent to be merged into a single object. This can be particularly useful for assembling 'tiled' datasets. Assuming a primary and a secondary raster have been selected in the thumbnail view, selecting the `Combine Rasters` option will bring up a window similar to that on the left of Figure 2.17. If the spatial extents of both rasters overlap but are not identical, you will have the option of either selecting the *intersection* of the two layers, or their *union*. For cell locations that are present in both rasters, you can select the output value for those locations to be the primary raster cell value, the secondary raster cell value, or the average of the two. Additionally, you can choose to override this selection for cells where the chosen raster value is *null* but would be a numeric value if selected from the other raster.

Combing Vector Data

Vector maps can be combined in much the same way as raster data. Select the two maps to be combined as primary and secondary vector maps, then choose the `Edit→Combine vector maps...` menu option. As with raster combination, if the two vector map areas overlap you will have the option of either selecting the *intersection* of the two maps, or their *union*. You will also have the option of using the intersection/union of the bounding rectangle of the entire map, or the combination of the vector objects (see right of Figure 2.17). This second option is useful if you wish to select all objects that fall within a given set of polygons for example.

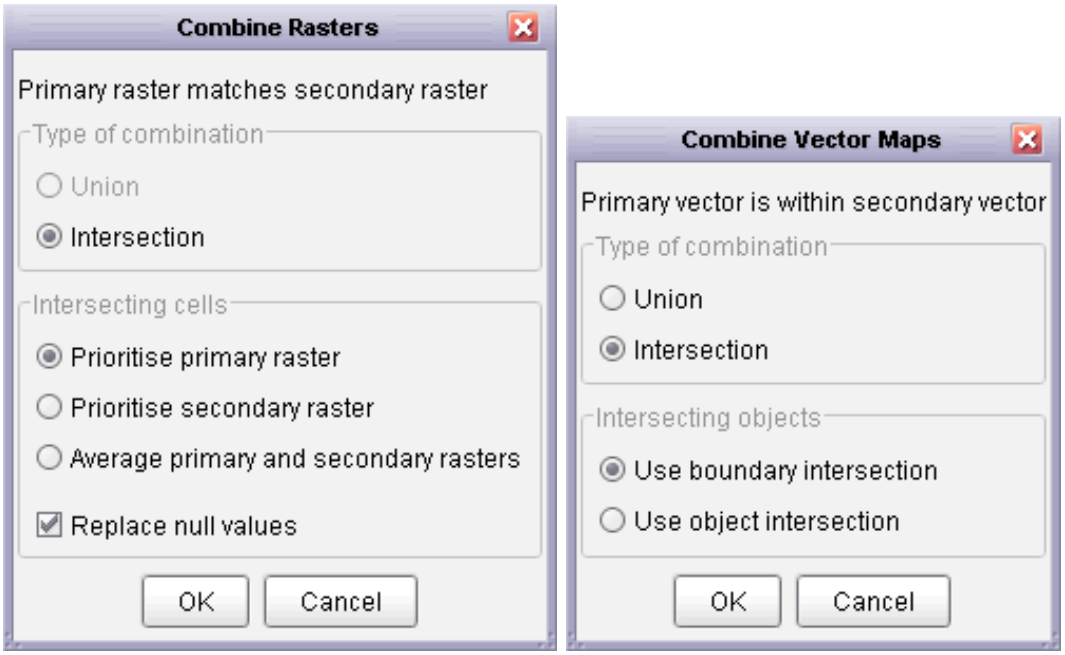


Figure 2.17: Raster (left) and vector (right) combination options.

'Cookie-cutting' Rasters With Vector Maps

The spatial extent of vector maps or the objects within them can be used to cut out selected portions of raster data. Parts of the the raster map that fall outside the vectors are replaced with null values and those parts within are retained. To do this, make sure you have a primary raster and primary vector map displayed, then select `Edit`→`Combine raster and vector map...`. By selecting `Intersection` as the type of combination and `Use object intersection` as the type, LandSerf will create a new raster consisting of any of the primary raster's cells that fall within any area objects in the primary vector (see Figure 2.18). If `Use boundary intersection` is selected, the enclosing rectangle of the vector map is used as the basis for raster selection.

Copying Selected Vector Objects

Choosing `Edit`→`Select vector objects...` allows a copy of a vector map to be created containing only objects with the given set of attributes. In the window that appears, enter numeric attribute IDs in the `Attributes to select` field. These should be comma separated and can contain ranges of values separated by '`to`'. To copy null values, '`n`' can be entered. The following are therefore all examples of valid entries:

```
1
10,6,250
1 to 99,-999
0,n
```

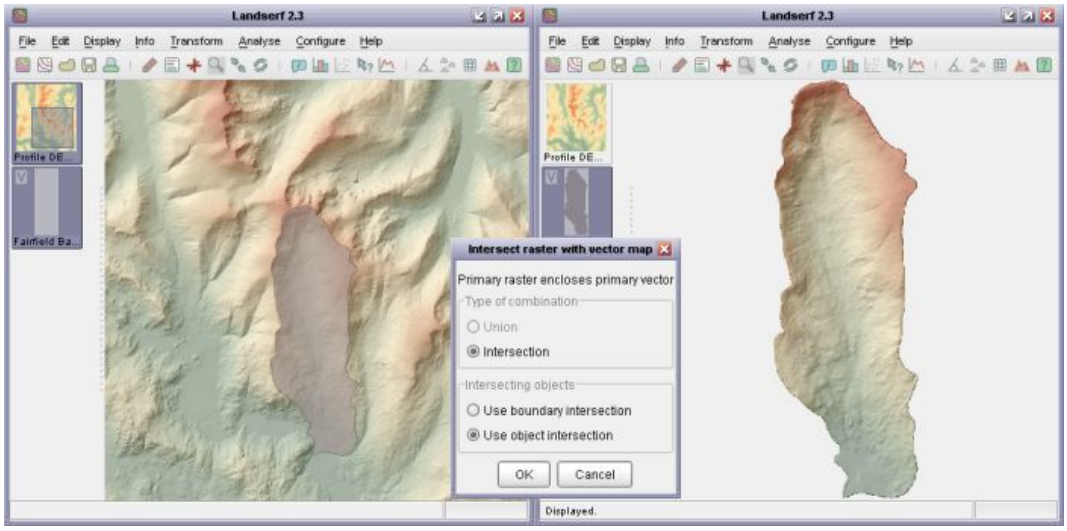


Figure 2.18: Cookie cutting a raster with a vector object.

Transferring Raster Values to Vector Points

The final way in which spatial objects may be combined requires a primary vector map containing point objects and a primary raster map. By selecting `Edit→Combine points with raster`, a new attribute is associated with each point object. That attribute is based on the raster cell value at that point. This provides a convenient way of, for example, using a DEM to create spot heights for selected locations.

2.3.4 Creating New Attribute Data

Both raster and vector data are associated with numeric attribute values. However, this can be limiting for data that have non-numeric or multiple attributes. In order to overcome this, any raster or vector map in LandSerf can be linked to an *attribute table* that defines the relations between numeric attributes (the 'ID' or 'key') and any other numeric or textual attribute values.

To create an attribute table, select either `Edit Raster` or `Edit Vector` as appropriate and then click the `Edit` button in the `Attributes` area (adjacent to the minimum/maximum attribute values). This should open a new attribute table editor that allows you to add rows and columns to the attribute table as well as load or save them from/to a file. Any attribute can be edited by clicking on the appropriate cell in the table. Attribute type names (column headings) can be changed by clicking the relevant column header.

Attributes can be either numeric or textual. Textual attributes have the advantage of allowing meaningful descriptions to be attached to objects, while numeric attributes can be manipulated analytically, for example, by calculating their average. By default, if you add a new attribute to a table, it is assumed to be textual. If all rows in the table of a textual attribute happen to be numbers, they can be converted to numeric attributes by pressing the `Validate` button.

All attribute tables have an *active attribute* associated with them. This attribute is indicated in bold in the editor window (see Figure 2.19 for an example). The active attribute is the one that

is displayed when querying a raster or vector. To make any attribute active, simply click one of the values in the relevant table column.

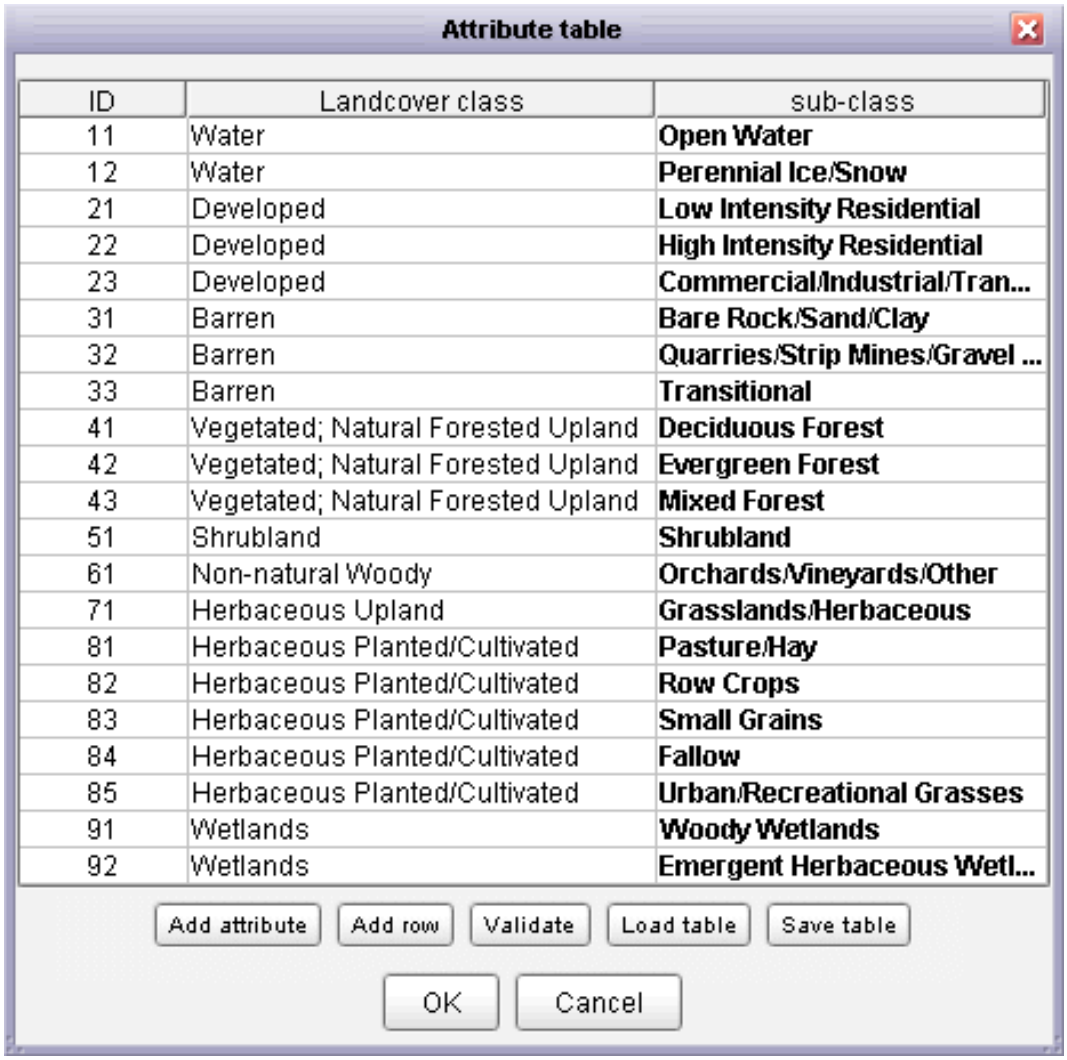


Figure 2.19: LandSerf attribute table.

2.3.5 Editing Spatial Objects

All spatial objects (raster and vectors) can be associated with metadata describing the spatial properties of the object (bounds, resolution and map projection). They can also be classified into different *object types* (e.g. elevation, slope, features, fuzzy membership, image, contours, TINs etc.) that determine the type of operations that can be performed on them and the way in which they are displayed.

To edit either a raster or vector object, select it then choose the `Edit→Edit Raster` or `Edit→Edit Vector` menu item. Bounds, resolution (raster only), map projection, type, colour table and supplementary notes can all be changed from this window. Bounds can be changed

either by typing new coordinates for each of the four edges of the object, or by dragging the mouse in the main display window to identify a sub-area of the object. To 'cut out' a subset of an object, make sure the `Extract subset` option is selected. To change the resolution of a raster object, ensure the `Interpolate to new resolution` option is selected and appropriate values are entered in the `N-S Res` and `E-W Res` fields. This technique can be used to remove 'steps' in a coarse DEM, or to provide an optimal sampling strategy for rasters that are too large for effective processing.

If you know the map projection system used to represent a spatial object, this can be set by clicking the `Edit` button in the `Map projection` section of the edit window to create a window as shown in Figure 2.20. You can choose between a limited set of projections and ellipsoids. Note that changing the projection information here does not reproject the spatial data. To transform the data from one projection system to another, select an appropriate option from the `Transform`→`Reproject` menu (see Section 'Coordinate transformations' below).

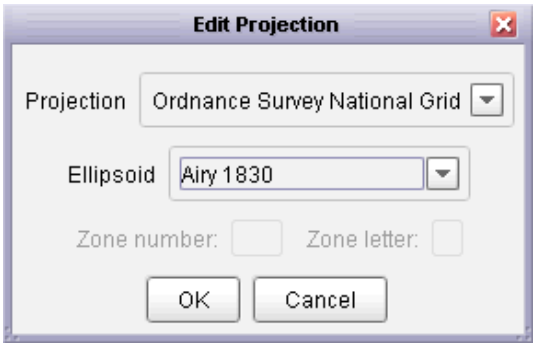


Figure 2.20: Projection information.

To commit changes made in the edit window, press the `OK` button. If there is an inconsistency between the new raster bounds and the resolution and neither the resampling or subset options are selected, the relevant fields will be highlighted when you attempt to exit the window.

2.3.6 Removing Data

Rasters and vectors can be removed by selecting the relevant thumbnail view and then selecting either the `File`→`Close raster` or `File`→`Close vector` menu option. You can also close all spatial objects by selecting the `File`→`Close all...` option.

Simplifying Linear Data

Some vector objects may contain detail that is not required for analysis or display. Detailed vector boundaries can slow down performance and consume unnecessary resources. Selecting `Edit`→`Simplify vector map...` will allow you to produce a new version of a vector map containing fewer points. The degree of simplification is controlled by specifying a *tolerance distance* that corresponds approximately to the length of the largest features in a boundary that will be removed after simplification. That distance is measured in the map units of the vector map being simplified. So for example, UTM and National Grid coordinate systems would require

the tolerance distance to be specified in metres. Figure 2.21 shows the effect of simplifying the British Isles coastline with a tolerance of 10km.

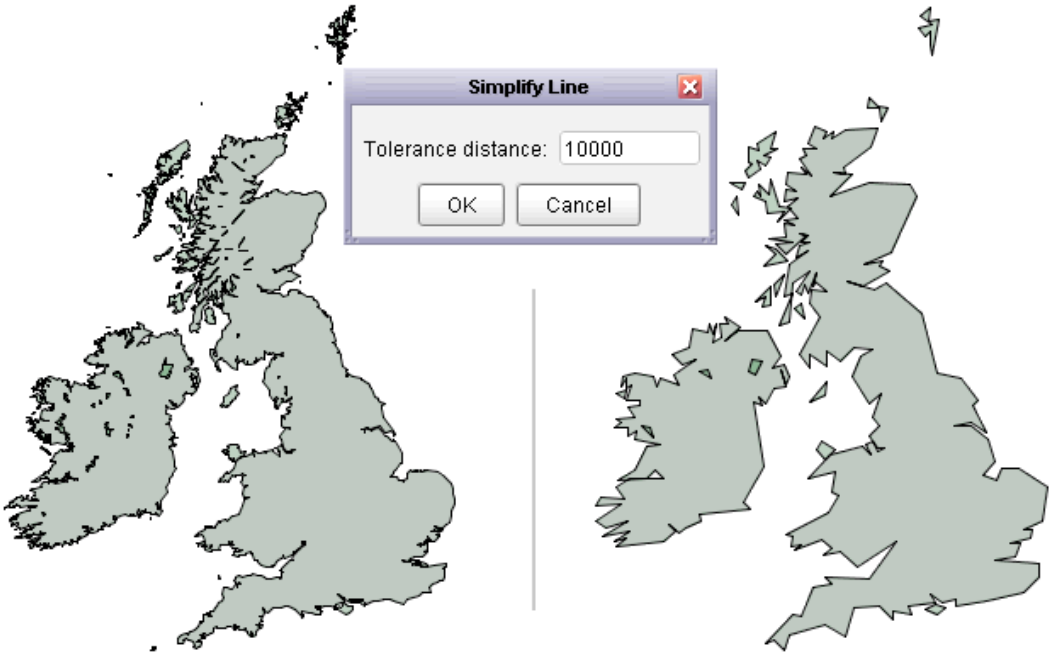


Figure 2.21: British Isles coastline before and after simplification.

The process uses the Douglas-Peucker line simplification algorithm which is applied to any line or area boundaries. Lines or areas whose maximum length are within the tolerance distance are removed entirely. Point features in a vector map are preserved.

Polygon Centroids

It is sometimes useful to identify the centroids of polygons in a vector map, for example when creating polygon labels or calculating density and proximity measures. To create a new vector map containing the centroids of all polyons in a given vector map, select `Transform→Polygons to centroids`. If necessary, the new centroid vector map can be combined with the original vector map to create single layer (see 'Combining Data' above).

Joining Linear Data

Linear data produced by a GPS track can often be fragmented into smaller line objects separated by periods where the device received no signal. Selecting `Edit→Join vector lines` will create a new vector map that joins up such fragmented line elements. The result will be a single line object with no 'gaps'. This can be useful for querying the total length of a GPS track or vector line (see 'Getting Information from Spatial Objects').

2.3.7 Transforming Data

LandSerf allows you to perform three types of transformation. *Coordinate transformations* change the locational representation of rasters and vectors and include map projection and rectification. *Model transformations* include the conversion of vector to raster models, and the generation of contours and TINs (Triangulated Irregular Networks) from raster surfaces. *Raster value transformations* allow the values of individual raster cells to be changed through scaling, reclassification, translation etc. All are accessible through the `Transform` menu.

Coordinate Transformations

Map Projections Both raster and vector maps may be reprojected from and to global latitude/longitude to and from the following projection types:

- Universal Transverse Mercator (UTM)
- Ordnance Survey National Grid (OSGB)
- French NTF National Grid
- Swiss National Grid
- United States Albers
- Conterminious United States Albers
- British Columbia Albers

All reprojections are selected from the `Transform`→`Reproject` menu item. Note that in order to reproject a raster or vector map, it must first have some defined map projection information. If necessary this can be identified by editing the spatial object.

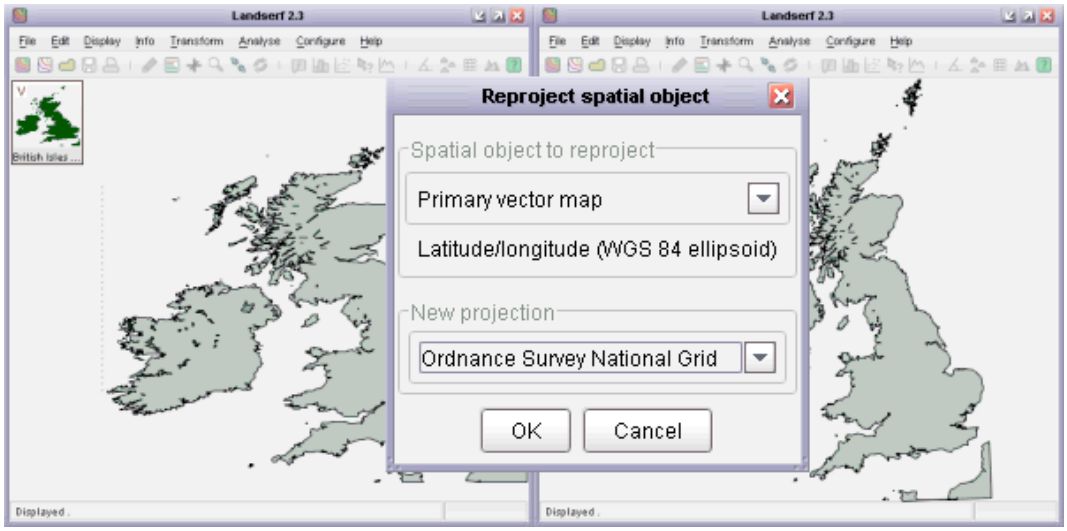


Figure 2.22: Lat/long (left) and OSGB (right) coordinate projections.

When rasters are reprojected you are given the option of setting the new resolution and bounds of the projected raster. This allows allows square pixels to be created by setting identical x and y resolutions. You are also given the option to control whether or not to *interpolate* new values.

Interpolation is suitable for continuous measurement data such as elevation. For categorical data, make sure this option is not selected so that new raster values are *resampled* from the original raster.

Image Rectification Many raster objects that need to be referenced to some spatial units can be georeferenced simply by editing the bounds of the four edges. However for some data, more sophisticated transformations are required. Rectification allows a number of 'before' and 'after' points (known as *transformation control points*) to be defined. LandSerf will then attempt to find an appropriate transformation that will convert the georeferenced locations of the 'before' points into appropriate 'after' locations. This transformation can then be applied to the entire raster.

To perform image rectification, display the raster that will be transformed in the main LandSerf window and then select the `Transform→Rectify...` menu option. A new window similar to that shown in Figure 2.23 will appear. Making sure the `Add points` option is selected, click on the main raster display at a point at which you know the true georeferenced location. The untransformed raster coordinates will appear in the rectification window in columns 2 and 3. Enter the georeferenced coordinates in columns 4 and 5. Repeat this process for other points on the raster. You should aim to create a spread of typically 10-20 points over the entire raster. To make this process easier, you may also use *zoom mode* to zoom into particular parts of the raster (making sure the `Add points` is *not* selected when using the mouse to zoom and pan).

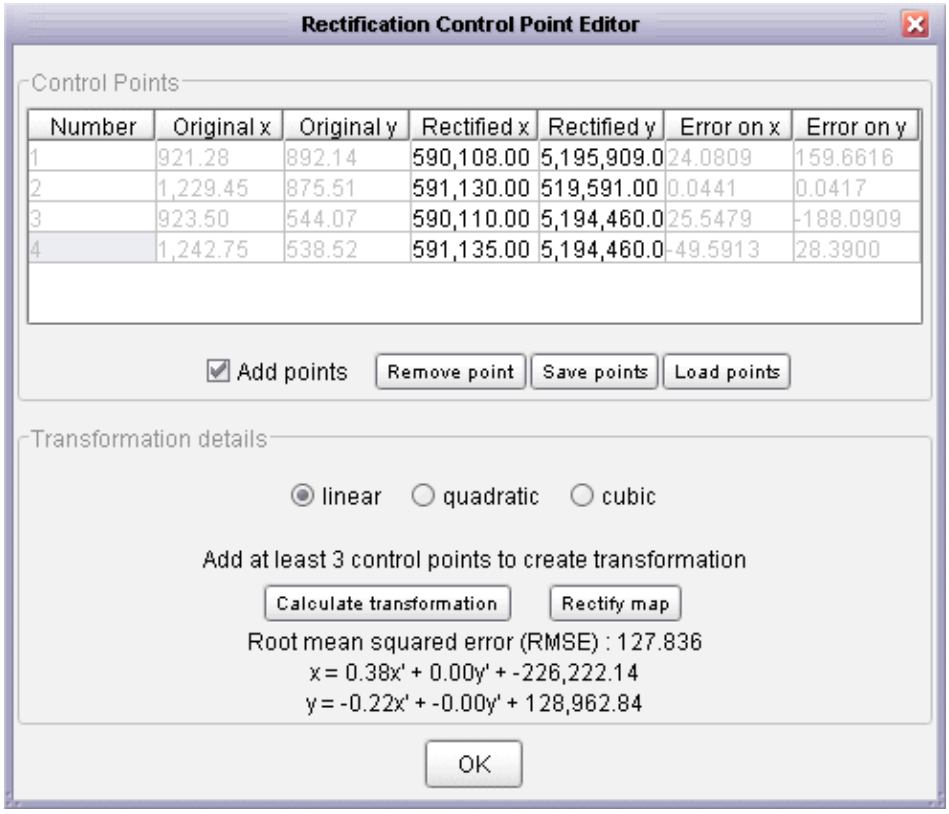


Figure 2.23: Rectification control point editor.

You can control the complexity of the transformation to be applied by selecting one of the `linear`, `quadratic` or `cubic` transformation options. To calculate the transformation, press the `Calculate transformation` button. This will display the coefficients of the transform along with a measure of *Root Mean Squared Error* (RMSE) which should be as small as possible. The error associated each point is also displayed, and this can give a clue as to which of the control points should be changed in order to reduce the error. It is often worth trying different linear/quadratic/cubic options and examining their effect on the overall RMSE. Figure 2.24 shows the degree of flexibility each of these has on the transformation process. Higher order transformations require progressively more points in order to model the transformation (linear - 3, quadratic - 6 and cubic - 10).

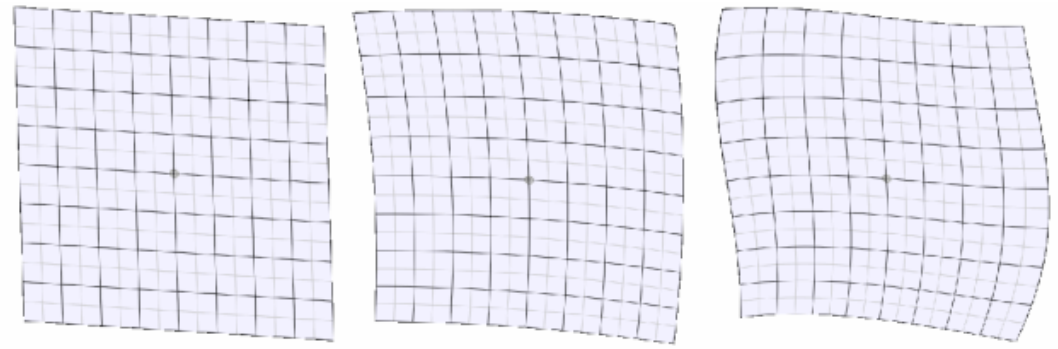


Figure 2.24: Linear (left), quadratic (middle) and cubic (right) rectification.

When you are happy with an appropriate set of control points and transformation type, these can be saved as a separate file for later use using the `Save points` button. To transform the raster itself, press the `Rectify map` button. You will then be presented with the option of changing the new bounds and resolution of the new raster before applying the rectification.

Model Transformations

A raster DEM can be converted into a vector Triangulated Irregular Network (TIN) by selecting the `Transform→DEM to TIN...` menu item. You will then be presented with a dialogue window asking for various triangulation options (see Figure 2.25). To control the number of triangles produced by the process, either specify the number explicitly and select the relevant check box, or specify some error criterion. The error (either average as specified by the Root Mean Squared Error, or the maximum error) represents the difference in elevation between any point on the original DEM and its elevation in the TIN. The smaller the error specified, the greater the number of triangles required in the network. A representation of the spatial pattern of errors can be produced by selecting `Create Error Surface` from the dialogue box.

In common with other LandSerf processing options, the DEM to TIN transformation can be interrupted at any stage by clicking on the progress bar in the bottom-right corner. Once interrupted, LandSerf will create a new TIN based on the number of triangles created at the point of interruption.

The triangulation process works by successively adding triangles to the network until either any of the selected error criteria are met, or the maximum number of triangles is reached (if specified).

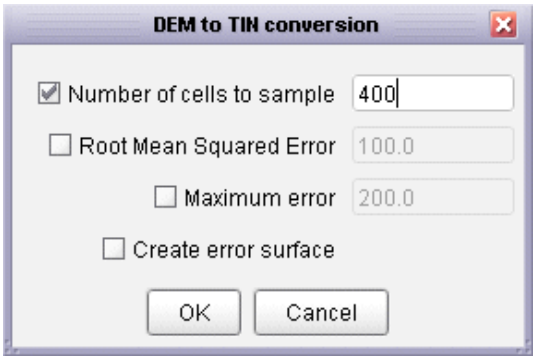


Figure 2.25: TIN creation options.

Alternatively, any set of point values can be triangulated by selecting `Points to TIN` from the `Transform` menu. This allow TINs to be created from points sets such as surface features (pits, passes and peaks) or spot heights.

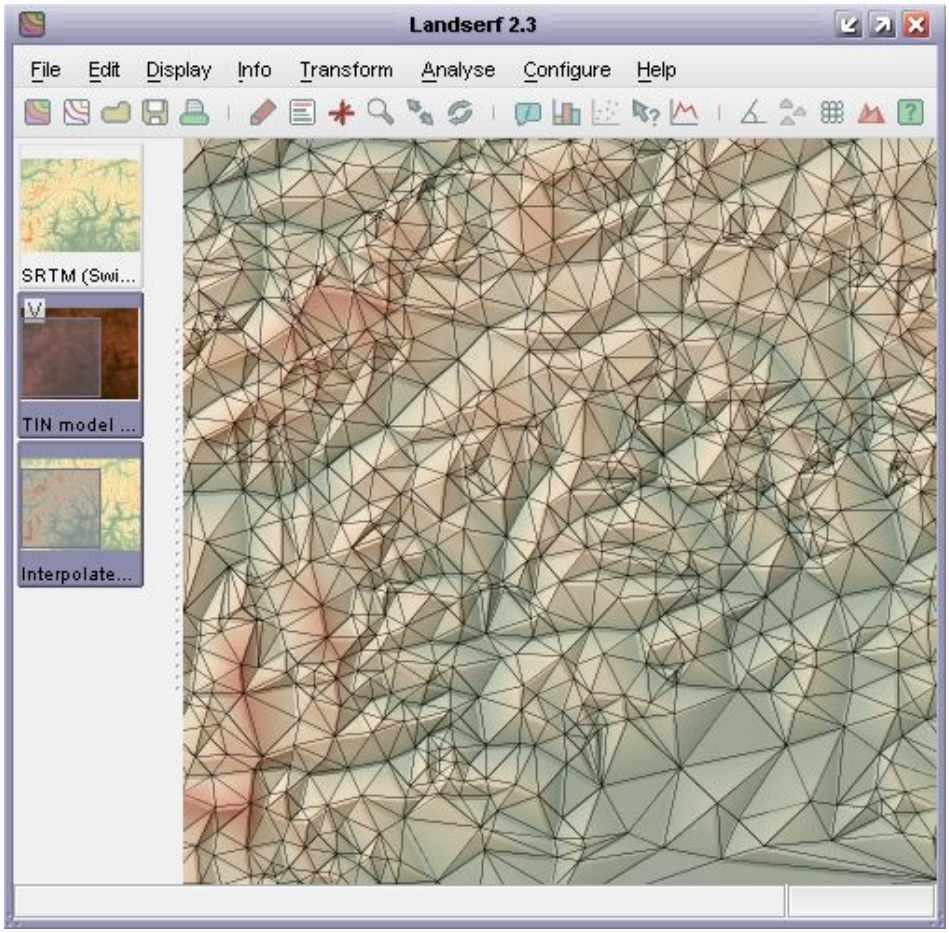


Figure 2.26: TIN over 'TIN to DEM' surface.

It is also possible to convert a given TIN back into a DEM. This is achieved by selecting the

`Transform→TIN to DEM...` menu option. LandSerf will apply a planar interpolation of the triangle network that tends to produce a surface composed of flat facets (see Figure 2.26). If necessary, these can be smoothed using *Quadratic Interpolation*.

To create a contour representation of a given DEM, select the `Transform→DEM to contours...` menu option. Once selected, you can control the contour interval and the elevation of the lowest contour (see Figure 2.27). For DEMs with large flat areas (e.g. sea around an island), more useful results are produced when a contours do not coincide with the elevation of the extended flat region. Alternatively, flat areas can be reclassified as *null* values and be excluded from the transformation. The `Grid width` option controls the resolution at which DEM cells are sampled in order to thread contour lines. A value of 1 will sample every DEM cell and produce the most detailed contour lines; larger values will be faster and will generalise the resulting contours.

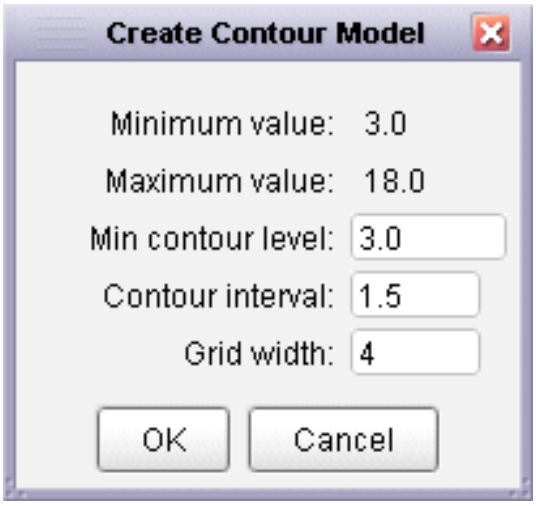


Figure 2.27: Contour creation options.

By default, generated contour vectors are given the same colour scheme as the DEM from which they were derived. This can make them difficult to see when overlaid on the same DEM, so it can useful to change the colour scheme of one of the two models (e.g. grey contours shown in Figure 2.28).

More general vector to raster transformations can be made by selecting the `Transform→Vector to raster...` menu option. This will attempt to rasterize the currently selected vector map to a given resolution specified. Point data can be rasterized in this way, or they can be converted rasters of point density by selecting `Transform→Point to density surface...`. On selecting this menu option, a new dialogue window is shown requesting the raster resolution required and the size of the local window used to calculate density values. This can range from 1 to the size of the raster, but should be an odd number. The larger the number, the smoother and more generalised the resulting density surface will be. The transformation uses Cressman interpolation to create the surface where each cell in the new raster is the average density of neighbouring vector points weighted according to

$w_ij = = (s - d_ij) / (s + d_ij)$

where s is the selected window size and d_ij is the distance from the centre of the raster cell to each neighbouring point within the window.

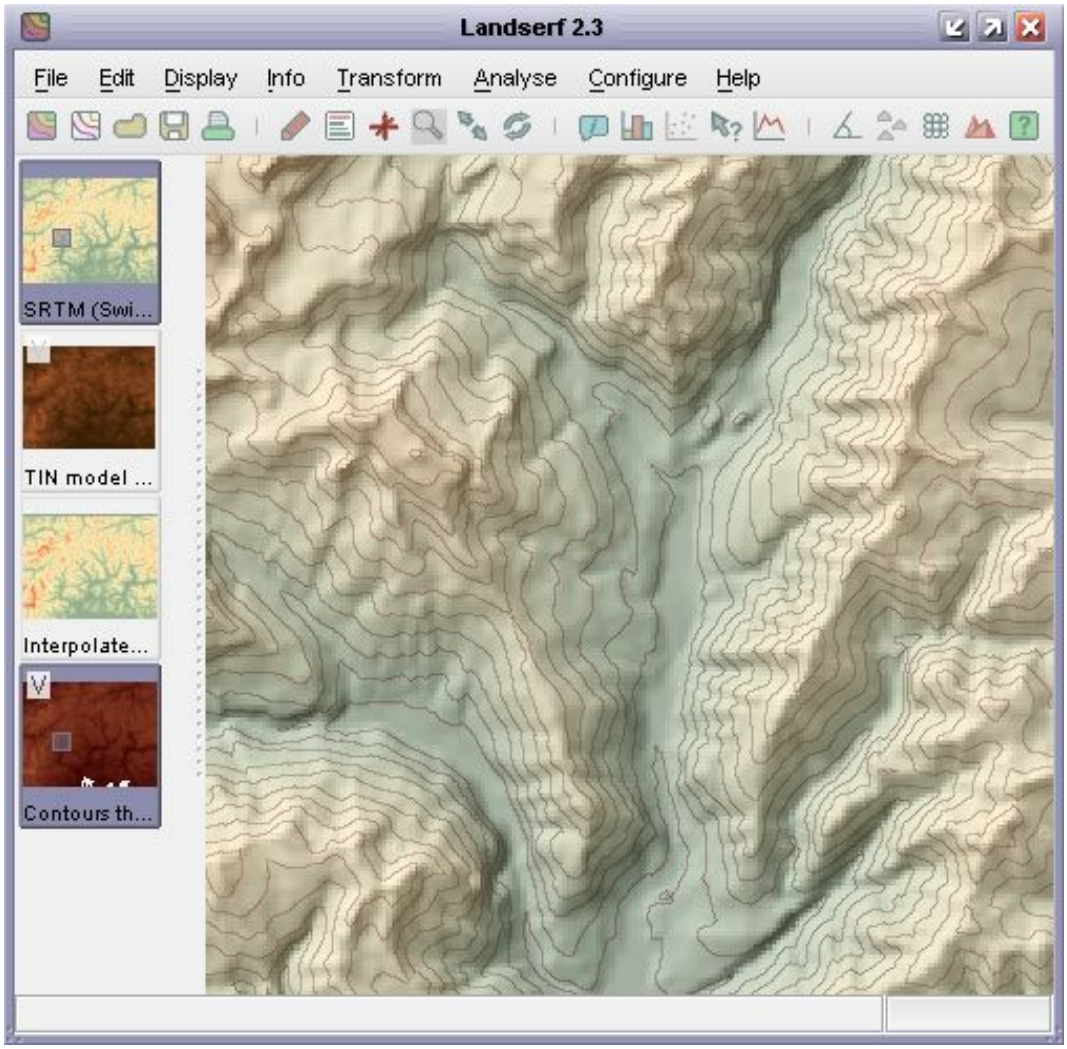


Figure 2.28: DEM with contour overlay

Transforming Raster Values

A series of simple transformations can be applied to the values of a raster's cells by selecting the `Transform→Raster Values...` menu item. Raster values are scaled by the value in the `Scale` field, translated 'up' or 'down' by the `Translate` value or rounded to the nearest `Round` value. The elevation model can also be 'flooded' such that all values below that in the `Flood` field will be assigned that value. Transformations are only applied if the relevant check box is selected.

Additionally, you can replace given raster values by entering appropriate figures in the `Replace ... with` fields. If the two numeric values here are different, then all occurrences of the number in the first field will be replaced with that in the second. This can be useful for simple reclassifications of categorical information. Rasters can also contain *null* values that are not displayed and are removed from processing operations. This is particularly useful for representing missing data, for water bodies and for excluding raster edges from processing. To replace numeric

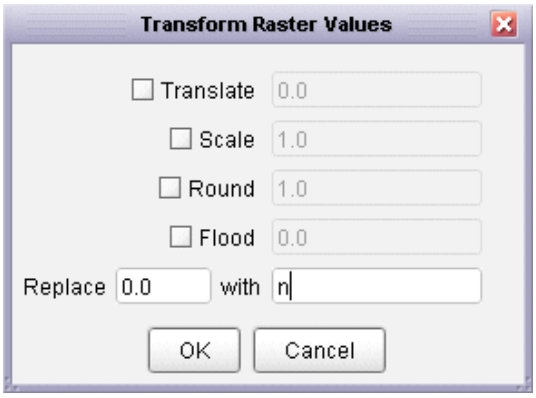


Figure 2.29: Raster value transformation replacing zeros with null values.

values with a null, or a null value with a numeric equivalent, enter an `n` in the relevant field (see Figure 2.29).

2.4 Visualising Spatial Data

2.4.1 Displaying Raster Data

To display raster surface models, select either `Raster` or `Relief` from the `Display` menu. The former will display the primary raster based on its own *colour table* (see Editing Colours below). Raster values that fall between those with defined colours are given a linearly interpolated colour value. This gives a continuous range of colours for most DEMs. *Relief* presentation will combine these same colour rules with a shaded relief calculation simulating local texture and shadow (see Figure 2.30). If a *secondary* raster is selected, the colour rules of this raster will be combined with shaded relief calculated from the *primary raster*.

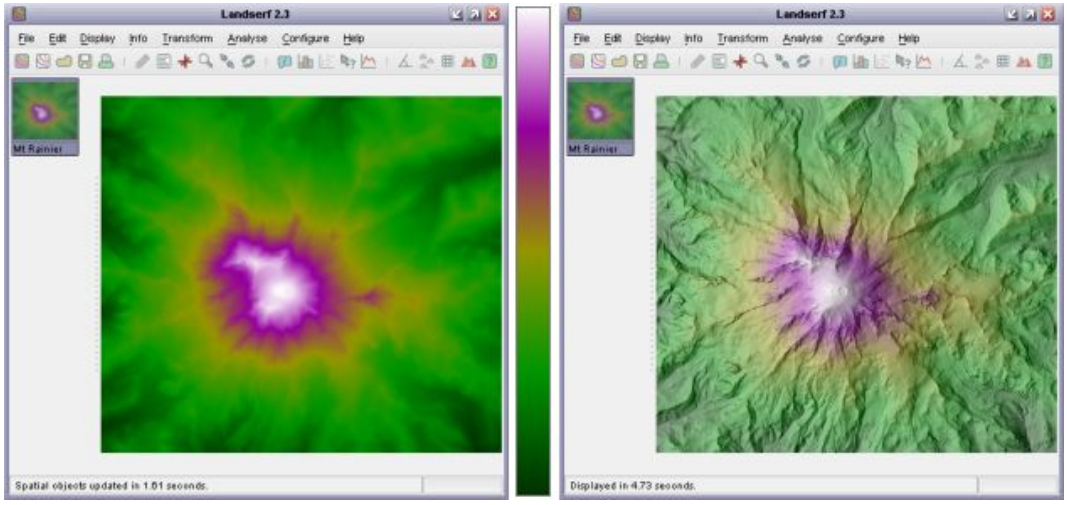


Figure 2.30: Raster and shaded relief display with default colour table.

Various settings for the shaded relief calculation may be set from the `configure`→`Shaded relief...` menu option or the button. Each of the parameters can be changed in realtime by moving the relevant slider. The small thumbnail image shows the effect of these changing parameters on the surface.

The position of the 'sun' illuminating the surface is controlled by the `azimuth` and `elevation` sliders. Lower sun angles give more pronounced shadowing effects, while azimuths from a southerly or easterly direction give the impression of inverted topography. The relative balance of the colour and shading can be controlled using the `Percent shading` slider. `Vertical exaggeration` can be used to control the degree of shadow throughout the whole image and is particularly useful for larger DEMs representing non-mountainous terrain. The `Aspect bias` slider controls the degree to which slope direction (aspect) rather than slope steepness controls the amount of shadow. Higher values will tend to emphasise local detail in the surface and is particularly useful for detecting error artifacts in elevation models. Lower values will tend to produce more realistic shading.

Selecting `Show advanced` reveals a second set of shaded relief parameters that are used when creating a shaded relief surface as a *surface parameter*. These parameters control the *Phong lighting model*, their effects being shown on the grey sphere as the sliders are moved. Note that

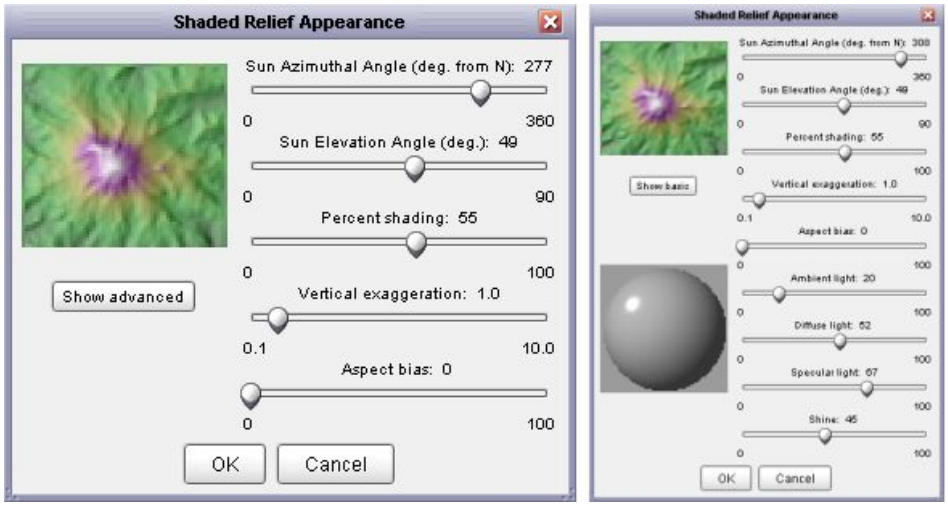


Figure 2.31: Basic (left) and advanced (right) shaded relief parameters.

none of these advanced parameters have any effect on the relief shading produced by selecting `Display→Relief`.

After pressing `OK` in this window, the main raster display will be updated with the new settings.

2.4.2 Displaying Vector Data

Vector maps are displayed by selecting the `Display→Vector` menu. This menu item is toggled on and off with further selection. Colours are assigned to vector values using interpolation of colour rules in an identical way to rasters. Both primary and secondary vector maps can be displayed simultaneously if selected.

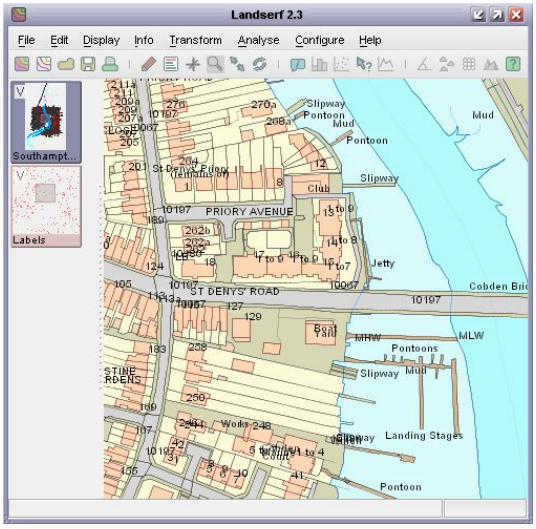


Figure 2.32: Vector map with points, lines, polygons and text labels.

The appearance of vector maps can be controlled through the editing of its colour table (see below) and by altering their appearance using the `Display→Vector appearance` menu item. This brings up a window as shown in Figure 2.33 that allows the width of vector points and lines to be set, as well as the transparency associated with vector polygons. Width values can be less than 1 in order to create thin lines and small points. Setting line width to 0.2 often results in attractive unobtrusive linear boundaries.Transparent polygons allow multiple features to be shown overlapping the same area on the ground. Polygon boundaries can be turned on or off, and their colour can be set by clicking on the small square representing the current boundary colour. Setting points or lines to `Surround` will draw boundaries *around* the feature rather than at the feature location. This can be useful for drawing parallel lines along roads or for circling points of interest.

Labels associated with point objects in the vector map can be optionally displayed by ticking the `Show labels` box. The appearance of these labels can be controlled by selecting an appropriate foreground and background colour (click the coloured square to change it). Particularly useful is the ability to change the transparency of the background. By using semi-transparent background colour, sufficient contrast with the foreground lettering can be used without obscuring any underlying map data (see Figure 2.33). The size of the labels can controlled with the slider. The position of the label relative to its point location can be controlled by selecting the relevant button under `Text Alignment`. When displaying labels is often useful to select an appropriate textual attribute from the attribute table.

The vector appearance dialogue can also be used to set higher quality rendering of vector objects (`Optimize for quality`), or faster rendering (`Optimize for speed`).

Appearance settings may be saved as a file by pressing the `Save Styles` button for later retrieval via the `Load Styles` button. Vector draw style files should have the extension `.vst`.

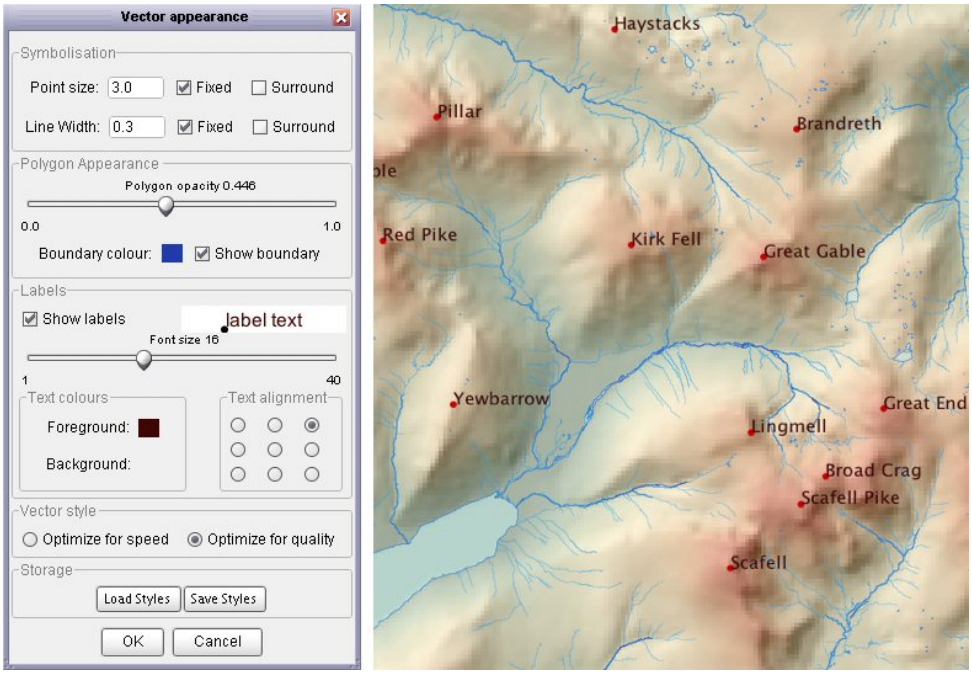


Figure 2.33: Vector display properties and sample output.

2.4.3 Zooming and Panning

Any vector or raster map can be enlarged by placing LandSerf into *zoom mode*. This is achieved either by selecting the `Display`→`Zoom mode` menu option or by toggling the button. When in zoom mode, dragging the mouse up and down over the main display area with the left mouse button pressed will zoom in and out of the displayed image. Zooming can also be achieved by using the mouse wheel if present. Dragging the mouse with the right button pressed (or <shift> left button) will allow a zoomed image to be panned in any direction. When zoomed in to a part of a map, the area displayed in the main window is highlighted on the primary and secondary thumbnails to the left (see Figure 2.34). To reset a view back to its full extent select either the `Display`→`Full image` menu option or the button.

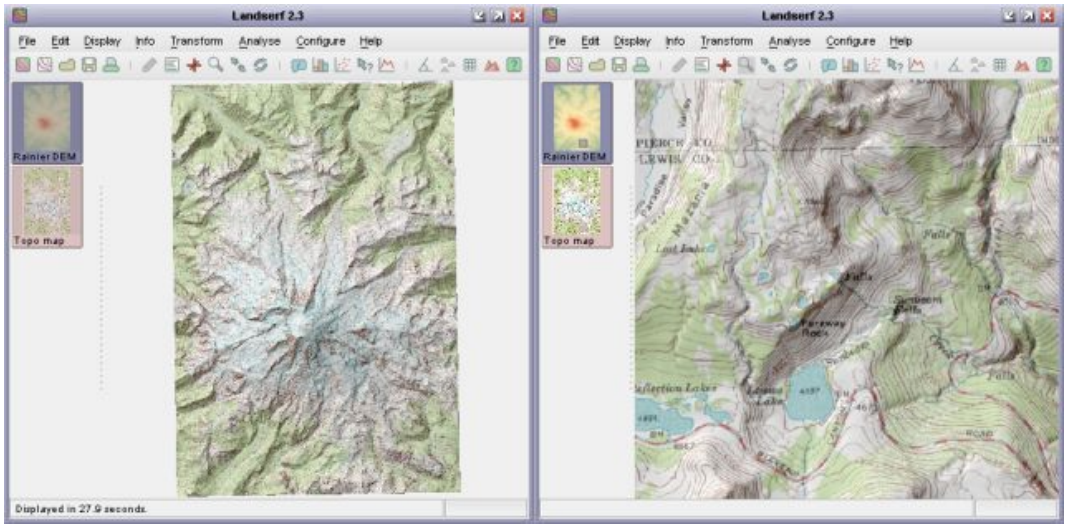


Figure 2.34: Surface with full extent (left) and zoomed-in area (right). Note small highlighted area on thumbnail in zoomed version.

Displaying Combined Models

Two raster maps may be combined in a single display in a number of ways. In addition to the shaded relief mapping, three other forms of display combination can be selected from the `Display` menu.

Selecting `Blend` will take the average RGB values of the primary and secondary rasters to produce a new colour map (see Figure 2.35). A window will appear requesting the the percentage weighting given to the primary raster. The closer to 100%, the closer the blended image will resemble the colours of the primary raster. If the resolutions of the two rasters are different, the blended map will always use the resolution of the primary raster, so for the highest quality results, the raster with the higher resolution should be selected as the primary raster.

Selecting `Hue-Intensity` will use the hues ('colour') of the primary raster combined with the intensity (degree of black/white) of the secondary (see Figure 2.36). This can be useful for producing slightly bolder shaded relief maps (see Figure 2.36) as well as for combining coloured with grey-scaled maps.

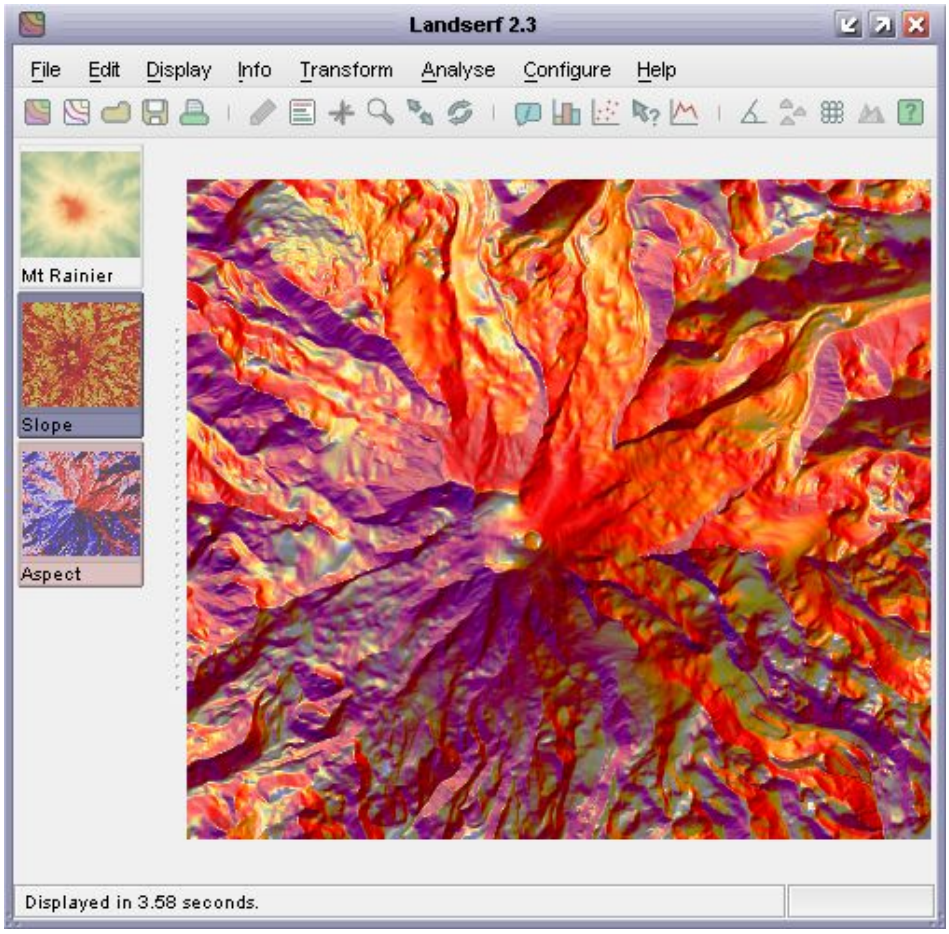


Figure 2.35: RGB blended slope and aspect map (50% slope, 50% aspect).

Selecting `Hue-Saturation` has a related effect to hue-intensity mapping, but in this case the intensity of the secondary map is used to control the saturation (greyness/boldness) of the combined images. Low intensities in the secondary raster result in washed out low saturation colours while brighter secondary raster values produce bolder colours (see Figure 2.37). Such combinations can be particularly useful for representing uncertainty, with stronger saturation indicating greater certainty.

Editing Colours

The colour tables associated with either vector or raster maps may be modified by double-clicking on the thumbnail of the object to edit (or by selecting `Edit`→`Edit Vector` / `Edit`→`Edit Raster`) and then clicking the `Edit` button adjacent to the displayed colour bar. Colour tables are defined by a series of *colour rules* that associate one or more attribute values with one or more colours. *Discrete colour rules* link a single attribute with its own colour while *continuous colour rules* are used to interpolate a range of attributes with a range of colours. Both discrete and continuous colour rules can be defined either graphically or numerically (see Figure 2.38).

The colour editor window will allow you to select one of several preset colour tables useful for

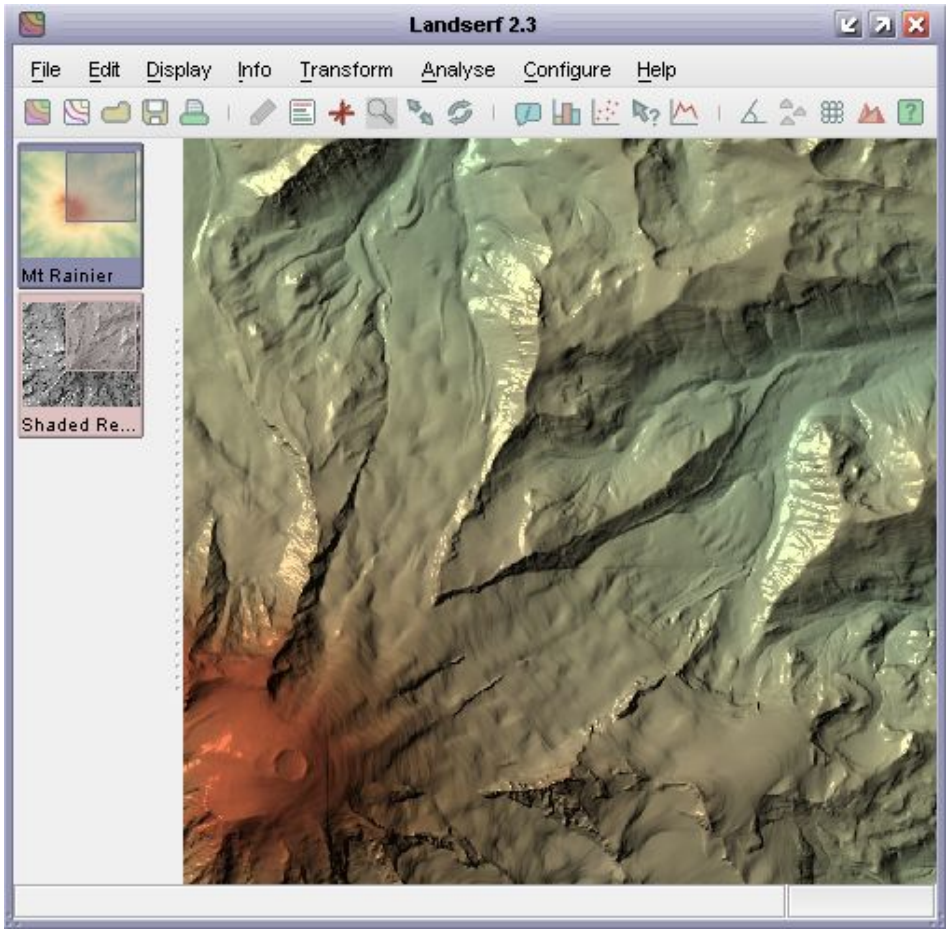


Figure 2.36: Hue-intensity map combining DEM (hue) and Phong shaded relief (intensity).

mapping onto continuous attributes such as elevation. The colour table is selected by choosing an appropriate colour scheme from the drop-down menu towards the bottom of the colour editor window. By default, each is scaled between the minimum and maximum values in the spatial object being displayed.

The 'land' and 'sea' colour tables follow guidance by the Swiss cartographer Eduard Imhoff and are particularly suitable for continuous data that you wish to display with a landscape-like appearance. The 'random' colour table is the only preset table that is specifically designed for discrete data. When selected, a random set of colours will be given to each numeric attribute in the object being displayed. Once set, the colours associated with each value will not change. The 'exp1' colour table is suitable for exponentially scaled data where most colour variation occurs near the minimum data value. The diverging schemes are suitable for data that vary around a central data point. This is assumed to be 0 if the spatial object being displayed contains both positive and negative values.

Once defined, colour rules can be modified by dragging the small vertical tic marks below the colour bar. These increase or decrease the attribute values associated with a given colour rule. A single colour can be changed graphically by double-clicking on one of the tic marks to bring up a colour chooser (see Figure 2.39 right).

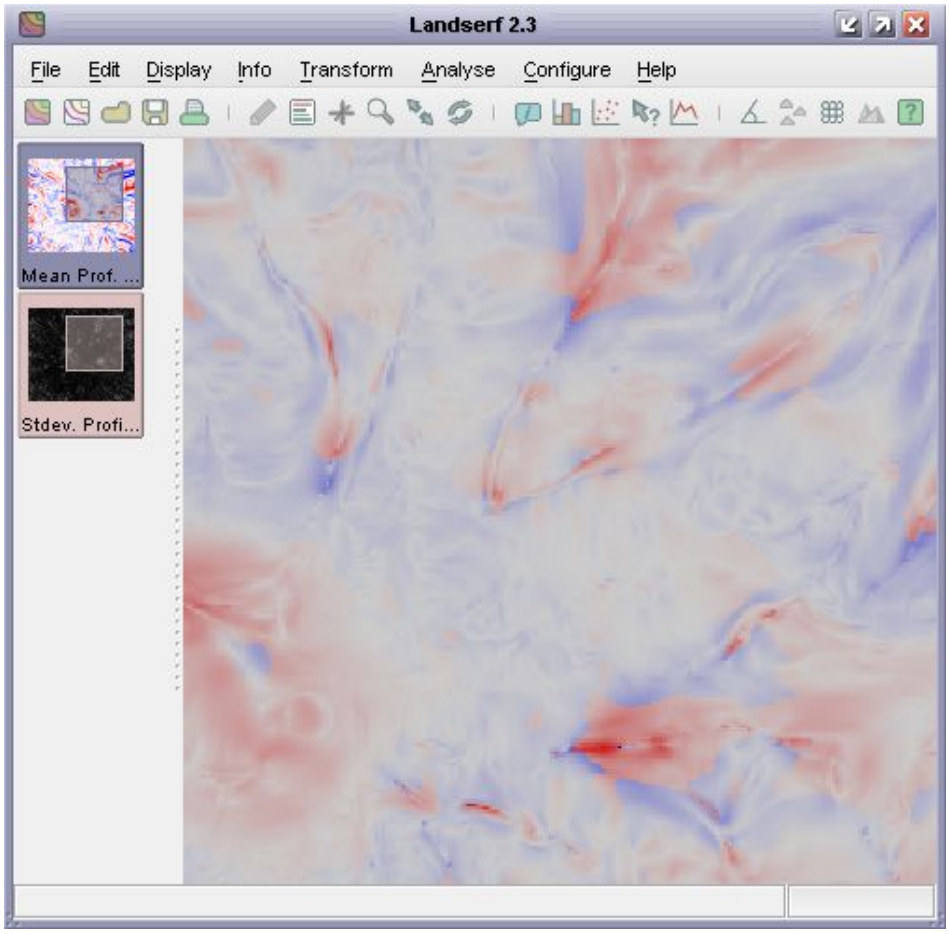


Figure 2.37: Hue-saturation map combining mean profile curvature (hue) local standard deviation of curvature measures (saturation).

For greater precision, colours tables can be edited numerically by selecting the `Numeric` tab at the top of the colour editor window. Each rule is defined by 5 numbers. The first is the attribute associated with a given colour. The remaining 4 numbers represent the *red*, *green*, *blue* and *alpha* (transparency) components of the colour. For convenience if the alpha value is omitted, it is assumed to be opaque (255). Likewise grey scale colours can be defined with a single number that will be applied to red, green and blue components.

Colour tables may be saved for later use or shared use between spatial objects. This can be particularly useful for frequently used datasets such as landcover maps that use particular IDs to represent particular categories of information. The `data` sub-directory of LandSerf contains an example colour table file for representing USGS landcover maps (see the LandSerf tutorial for an example of its use).

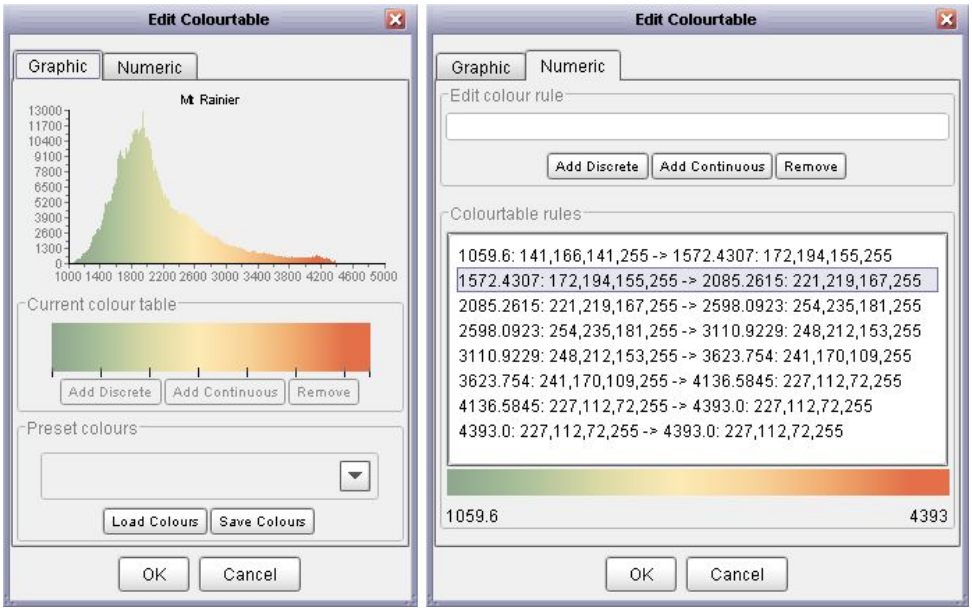


Figure 2.38: Graphical (left) and numerical (right) colour editor.

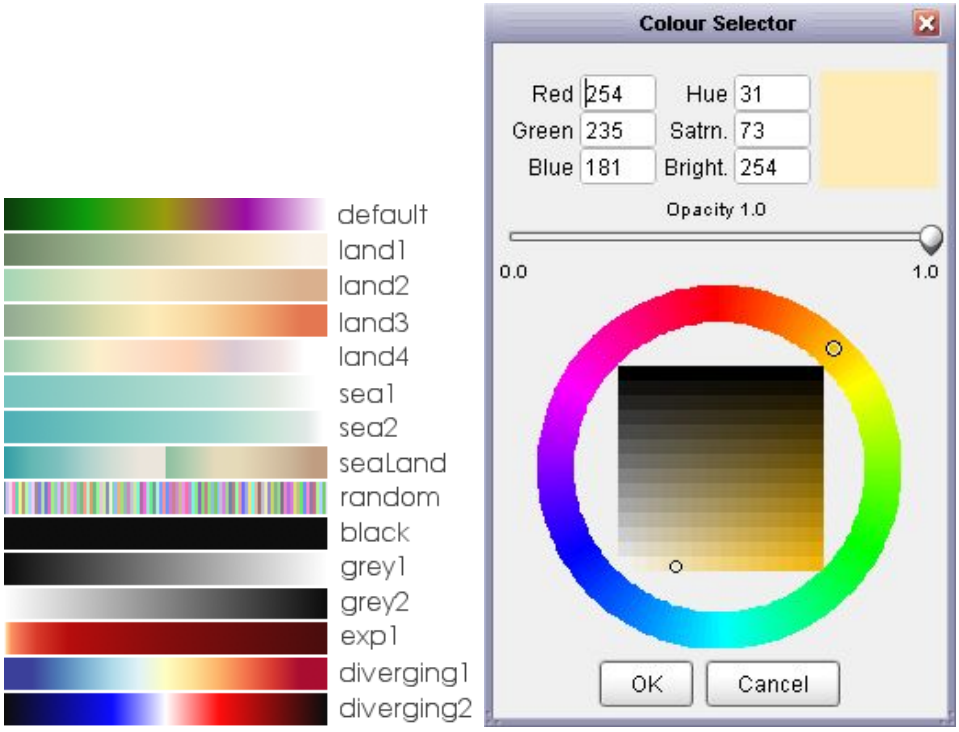


Figure 2.39: Preset colour tables (left) and graphical colour selector (right).

2.5 Viewing and Navigating in Three Dimensions

Machines with graphics cards capable of accelerating 3D graphics through OpenGL can take advantage of LandSerf's 3D viewing options. Non-accelerated graphics rendering is still possible, but in many cases will be too slow to allow much interactive visualisation.

To start the 3D viewer, a DEM must be selected as the primary raster. Optionally, a separate drape and vector map can be incorporated into the display if present. Selecting either the `Display→3D view` menu option or the button will start the 3d viewer. After a few seconds this should display a window similar to Figure 2.40 showing the primary raster draped with whatever was shown in the main LandSerf window.

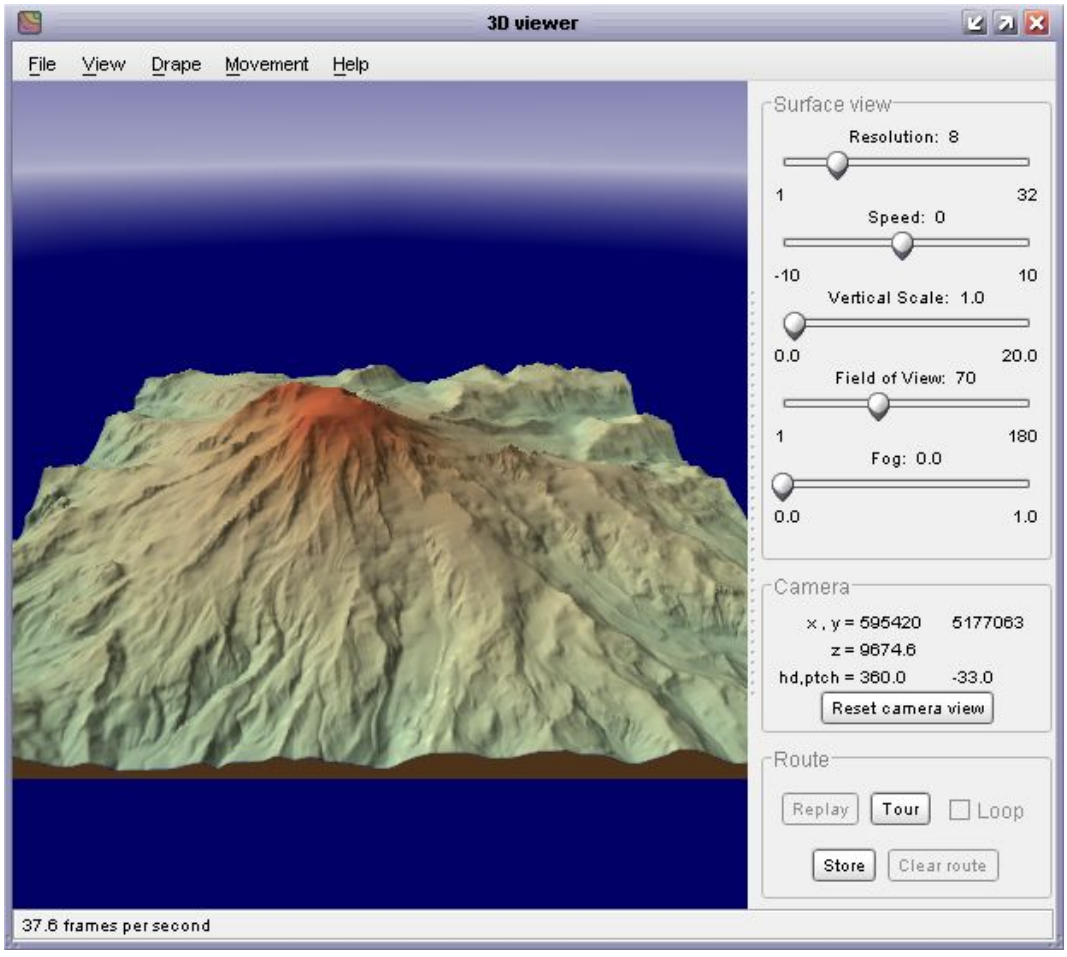


Figure 2.40: Initial 3D view.

Note that on some systems, especially Linux distributions, starting the 3d viewer can cause an error that quits LandSerf. This is usually due to some incompatibilities between the OpenGL drivers on that platform and the low-level graphics code in LandSerf. The most likely remedy is to ensure that you have up-to-date OpenGL drivers on your system.

2.5.1 Navigating in 3 Dimensions

To change the viewing position, use the mouse to control the location of the 'camera'. Navigation in three dimensions takes a little while to get used to, but the general rule for moving the camera is to think of the camera as being set in the nose of an aeroplane pointing at the centre of the viewing window.

Holding the left mouse button down while dragging the mouse up and down controls the pitch of the aeroplane. Dragging the mouse left or right controls the horizontal tilt ('yaw') and roll of the aeroplane.

To move towards or away from the viewpoint at the centre of the screen, either drag the mouse with the right button up or down, or hold the a shift key down while dragging the left mouse button. Dragging the mouse left or right in this way will move the camera left or right relative to your current view. If you have a mouse wheel between the two buttons, it can be used to raise or lower the elevation of the camera position. If you get lost at any stage, you can reset your position and viewing direction by pressing the `Reset camera view` button towards the bottom of the control panel.

When you have got used to these controls, the plane can be made to 'fly' or 'drive' by selecting the appropriate option from the `Movement` menu. Driving will keep the camera at ground level while flying will allow the camera to move anywhere above the surface. The speed of the movement can be controlled either with the mouse or the `Speed`slider control to the right of the view. To stop the camera from moving select wither `Hover` or `Walk` from the `Movement` menu.

Normally, the camera is kept above the surface during movement. The minimum height at which the camera can be placed over the surface can be controlled using the `Movement`→`Control Movement` menu item. This option also allows the amount of rolling associated with an azimuthal change in direction to be controlled. Setting a larger rolling component can be used to simulate an aeroplane banking movement, while a value of 0 simulates a 'helicopter turn' without rolling. The maximum animation speed can be set from the `Control Movement` option, which can be useful for creating synchronised screen grabbing animation files.

As you navigate over a surface, it is possible to record the route you take. This can be stored and later replayed. To store a route, navigate to an appropriate starting point and press the `Store` button towards the bottom of the control panel. If you navigate to a second position over the surface and press the button again, LandSerf will store the route between the two points. Further points can be added to build up a complete route over the surface.

To 'play back' the route you have created, press the `Replay` button. The camera will be moved from the first through each of the subsequent stored points to the last recorded location. If the `loop` tick box is set, the route will replay indefinitely. This is particularly appropriate for circular routes where the first and last points in the route are at similar locations. The speed of the replay can be controlled by `Speed` slider on the control panel. The playback can be stopped at any time by pressing the `Stop` button and the route can be cleared from memory by pressing the `Clear route` button.

Routes can be saved and loaded to /from disk by selecting the appropriate 'route' item from the `File` menu. A saved route will not only store the camera trajectory, but also the other viewing parameters such as field of view, fog setting, resolution and background colours (see below for details of these options). If you do not wish to define or load your own route, a default circular

'tour' around the currently displayed surface can be created by pressing the `Tour` button in the control panel.

2.5.2 Controlling Surface, Vector and Drape Detail

The resolution of the underlying triangular mesh can be controlled using the `resolution` slider to the right of the view. Generally this should be set to quite a coarse value (e.g. 16 or above) while you are positioning the view, as a fine resolution slows down the rendering considerably. If you have created a TIN in the main LandSerf window, this can be used to represent the surface in the 3d view by selecting `View→Use TIN` (see Figure 2.41).

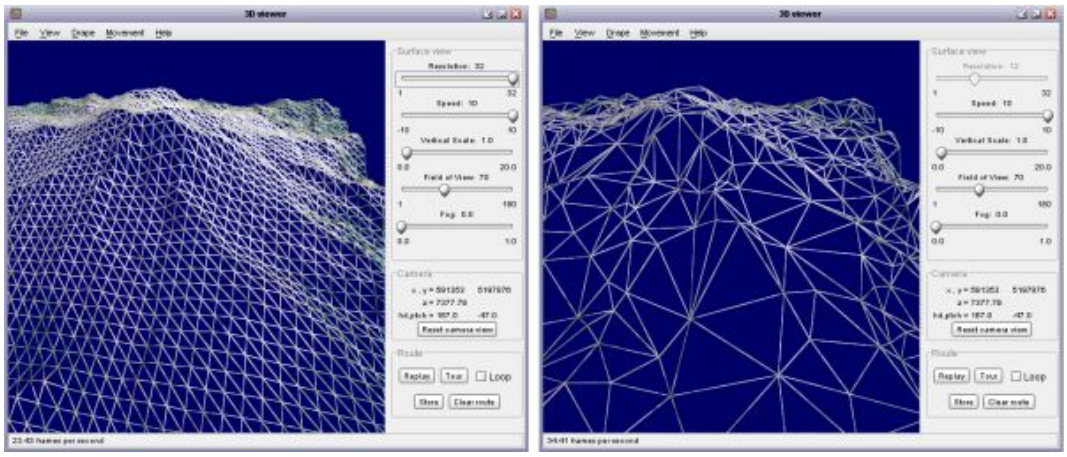


Figure 2.41: Mesh (left) and TIN (right) representation of surface shape.

If a primary vector map is selected in the main LandSerf window, this will be available for display in the 3d view (see Figure 2.42). Simply select `View→Show Vector` to toggle its display on or off. The appearance of the vectors can be controlled through the `View→Vector appearance` item. This allows the width, height and quality of the vector lines and points to be altered. Sometimes it is possible that parts of a vector object are obscured by the underlying surface. To avoid this, the `Vector appearance` box allows you to control the `clipping depth` effectively allowing even parts of the vector that are 'under' the surface to be displayed. You should experiment with the two clipping sliders to produce the best quality display.

If you wish to change the vector map displayed in the 3d view, select the appropriate vector in the main LandSerf window and then select `View→Refresh Vector` in the 3d view.

The raster to be draped over the surface is, by default, the image that was last displayed in the main LandSerf window. Alternatively a graphics file (GIF, PNG or JPEG) can be loaded from the `File` menu. The quality of the drape, from whatever source, is controlled from the `Drape→Quality` menu. The drape quality would not normally be set beyond the 'very high' level, which creates a textured surface of 1024x1024 pixels. Graphics cards with larger memories (typically 128 megabytes or greater) can also show the maximum texture size of 2048x2048 pixels (e.g. Figure 2.43).

The image draped over the surface can be updated at any time by selecting the `Drape→Refresh` menu item. Whatever is displayed in the main LandSerf window will be transferred to the 3d

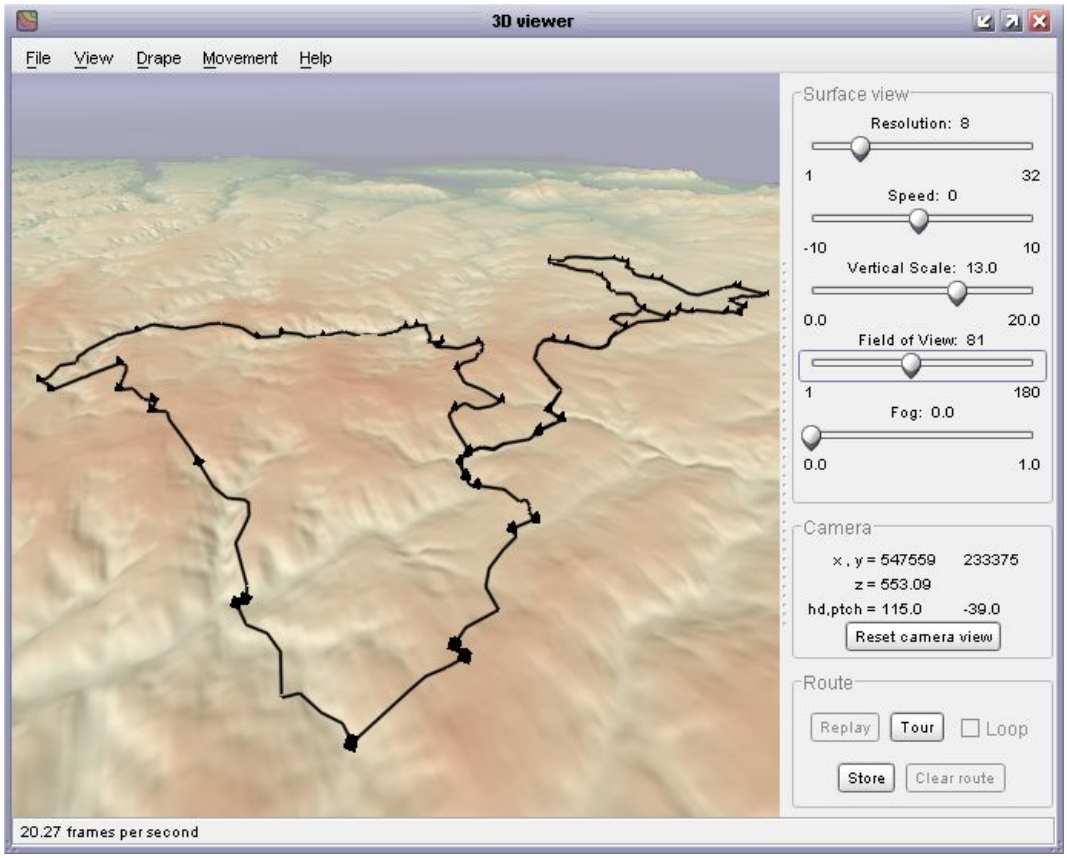


Figure 2.42: Vector route draped over DEM in 3D view.

view. For more sophisticated control over the drape, selecting `Drape→Distance drapes...` allows several independent images to be draped simultaneously. If your graphics card supports *mipmapping*, a different drape is displayed according to the distance between the 'camera' and surface. Larger resolutions correspond to closer views, smaller resolutions to more distant views. This technique ('*geomipmapping*') can be used to view multi-scale behaviour in a dynamic 3d environment.

One further surface texture may be added to the 3d view. Local detail is displayed when the camera is close to a portion of the surface model. By adding sub-cell detail here, the 'pixelising' effect of viewing DEM cells at close range can be reduced. To change the properties of the local detail, select the `Drape→Detail...` menu option, which should bring up a window similar to Figure 2.44.

From this window, a number of preset textures including *ripples*, *stones*, *marble* and *grids* can be selected. Their size and transparency can be controlled using the sliders. Alternatively, your own local texture image can be loaded and displayed by pressing the `Load` button. When viewed from a distance, tiled local textures can dominate the surface view, so by default the local texture fades with distance (see Figure 2.45). This effect can be turned on and off using the `Fade` option.

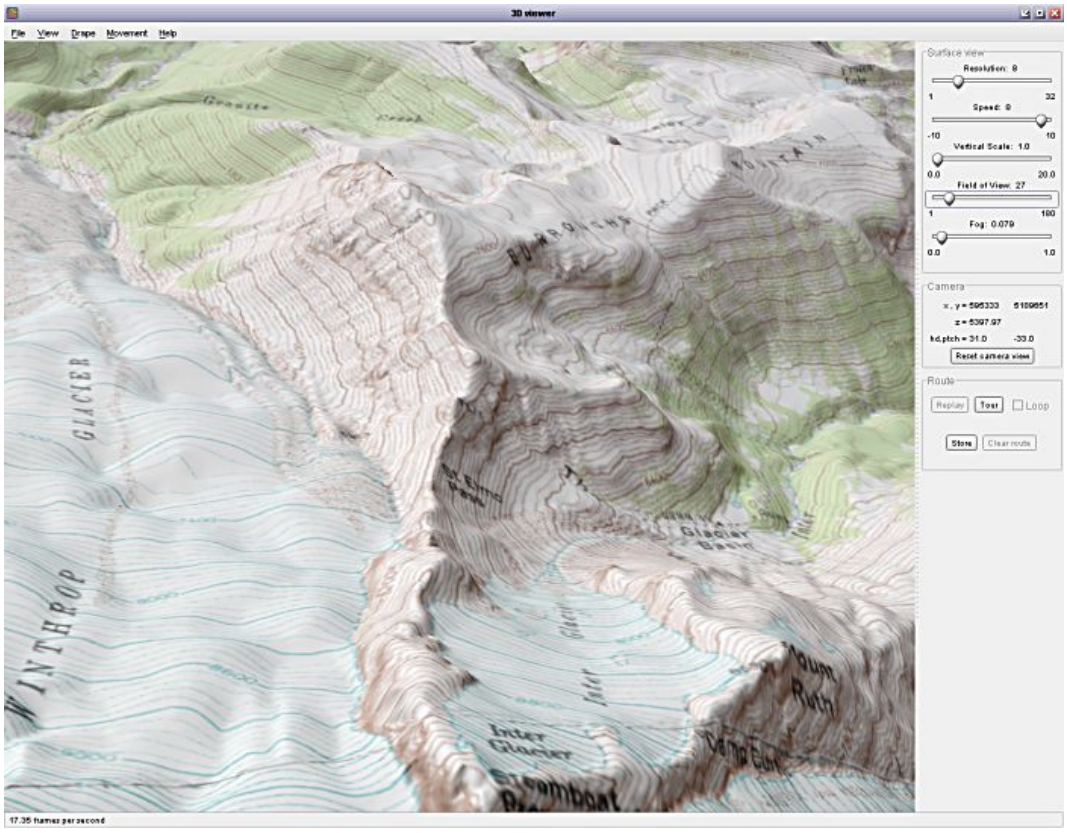


Figure 2.43: High quality drape over DEM in 3D view.

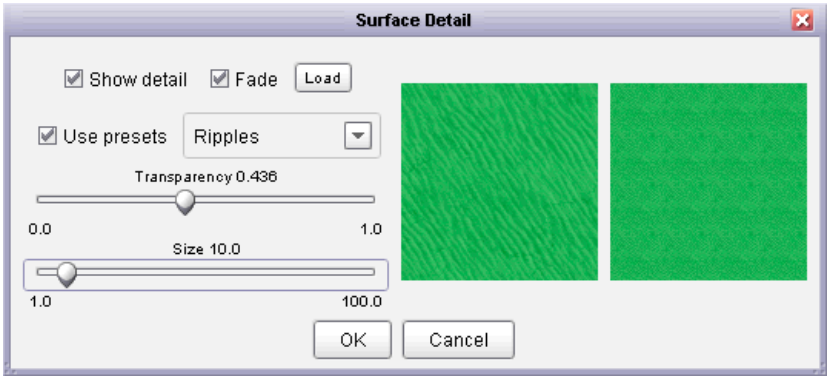


Figure 2.44: Local detail editor.

Environmental Settings

A number of viewing options control the appearance of the environment in which the surface is viewed. The camera `Field of view` slider on the main control panel can be used to set the degree of zooming of the camera lens. Wider fields of view taken with the camera close to the surface tend to imply immersion in a large terrain, while narrow fields of view ('zoom lens') suggest a smaller physical model of a surface viewed in greater detail.

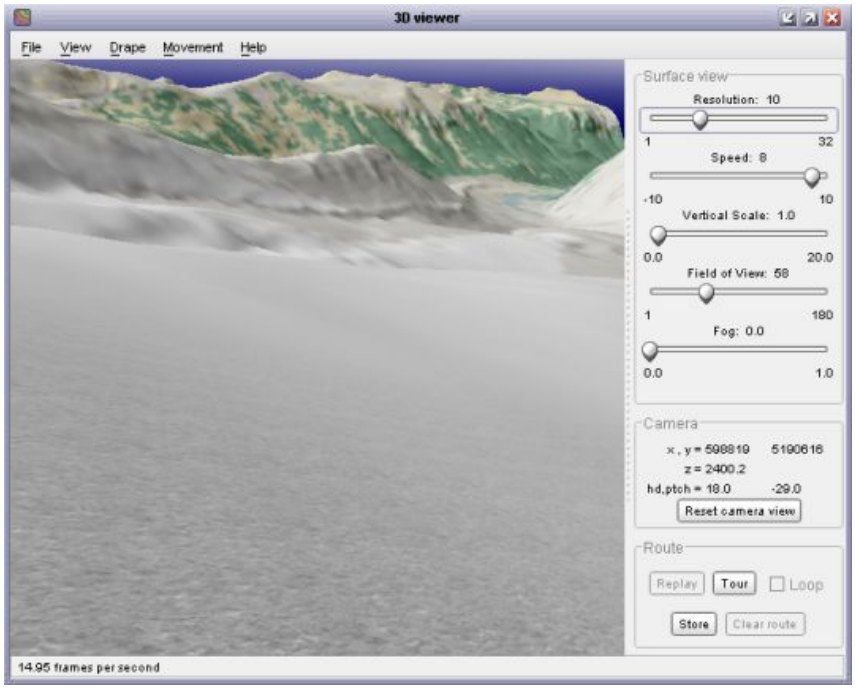


Figure 2.45: Surface with local 'stone' detail fading with distance.

Fog can be controlled with the `Fog` slider, which influences the degree to which the view becomes hazy with distance. Higher fog settings tend to reinforce immersion within the viewed scene.

The colours of the 'sea' surrounding the surface, the sky and the fog can all be altered interactively by selecting the appropriate item from the `View`→`Colours` menu. For added realism, it can be effective to set the sea colour to one similar to the dominant colour of the surface. This tends to blur the boundaries at the edge of the surface model, and especially effective for island models (see Figure 2.46).

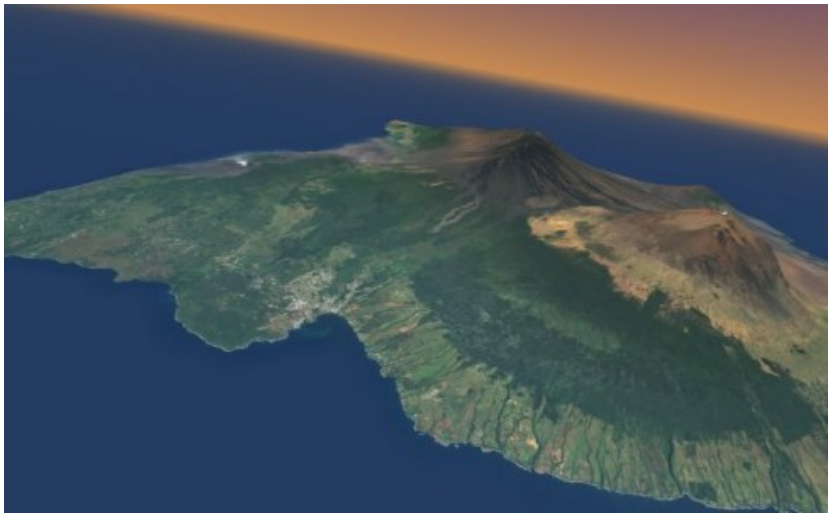


Figure 2.46: Aerial photograph over DEM in 3D view.

Image Output

Images from the 3d viewer can either by directly printed or they can be captured as image files for further manipulation. To print a 3d view select `File→Print preview` to preview the printed image (see Figure 2.47), or `File→Print...` to send to the printer.

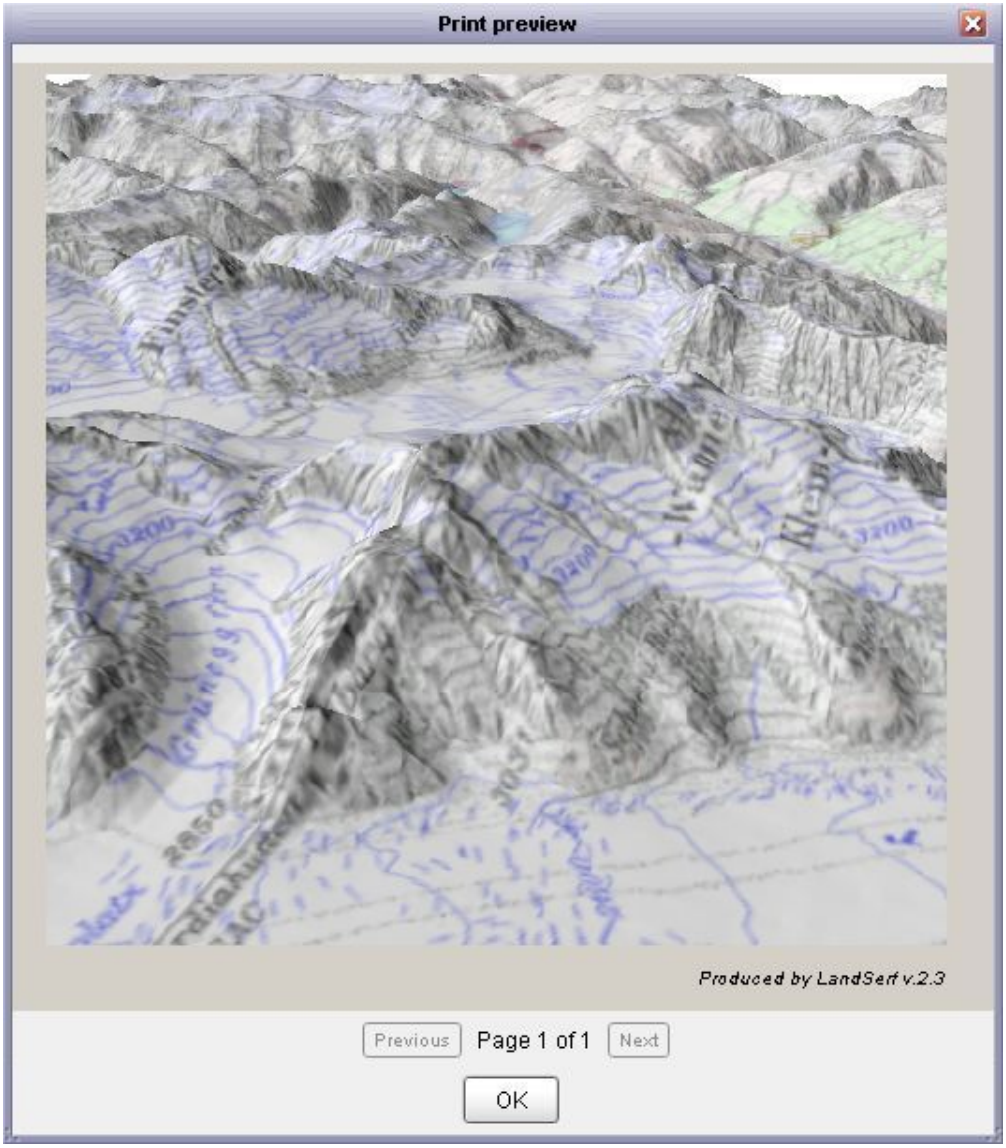


Figure 2.47: Print preview of 3d view.

You may wish to capture output from the 3d viewer as an image or sequence of images. This is particularly useful if you wish to create a stand-alone animation file to distribute to others without LandSerf. To configure image output, select the `File→Image Output...` menu item. This should display a window similar to Figure 2.48.

Clicking the `Snapshot` button will produce a single image identical to the current 3d display. Clicking the `Record` button will take a continuous sequence of images from the 3d viewer. This

Figure 2.48: Image capture options.

can be used to build up an animation sequence. Note that while recording in this way, rendering to screen is likely to be much slower than usual. Recording can be stopped at any point by pressing the `Pause` button.

The name and location of snapshot and animation files is determined by the `folder` and `base` options. The folder name can be entered directly or selected with the `browse` button. The 'base' name is that used to name the output file. It will be appended with the number of the file being written. The format of the image file can be selected using the drop-down menu. After changing the folder, base or format, make sure you press the `Set output` button to confirm your changes. So, for example, if the folder was set to `c:\anim`, the base to `image` and the format to `JPEG`, the first 3 files written would be `c:\anim\image0001.jpg`, `c:\anim\image0002.jpg`, `c:\anim\image0003.jpg`. The name of the last file written is displayed at the bottom of the image recorder window. To create a stand-alone animation file from a sequence of images, you will need to use third-party software such as Virtual Dub to assemble the individual images into a single animation file.

2.6 Getting Information From Spatial Objects

2.6.1 Text and Numeric Output

All spatial objects used by LandSerf are fully georeferenced. This allows rasters and vectors to be co-registered when overlaying one on the other. To display this information, select the relevant objects from the thumbnail view and select either `Info→Summary Info` or the button. This displays the title, bounds, map projection information, notes and colour table associated with the spatial object (see Figure 2.49). This information may be changed by selecting the relevant item from the `Edit` menu.

Figure 2.49: Spatial object summary information.

Univariate statistics for a raster can be calculated by selecting the `Info→Statistical summary` menu item. This will calculate measures of average, dispersion, spatial autocorrelation (local roughness) and fractal dimension (see Figure 2.50 left). You also have the option of displaying a variogram that indicates how roughness changes with scale (see Figure 2.50 right). The data points that make up this variogram can be exported as a simple text file for use in spreadsheets etc. by clicking the `Export variogram` button.

More detailed information about raster and vector attributes can be found by querying the spatial object interactively with the mouse. To do this, place LandSerf in *Query Mode* by either selecting

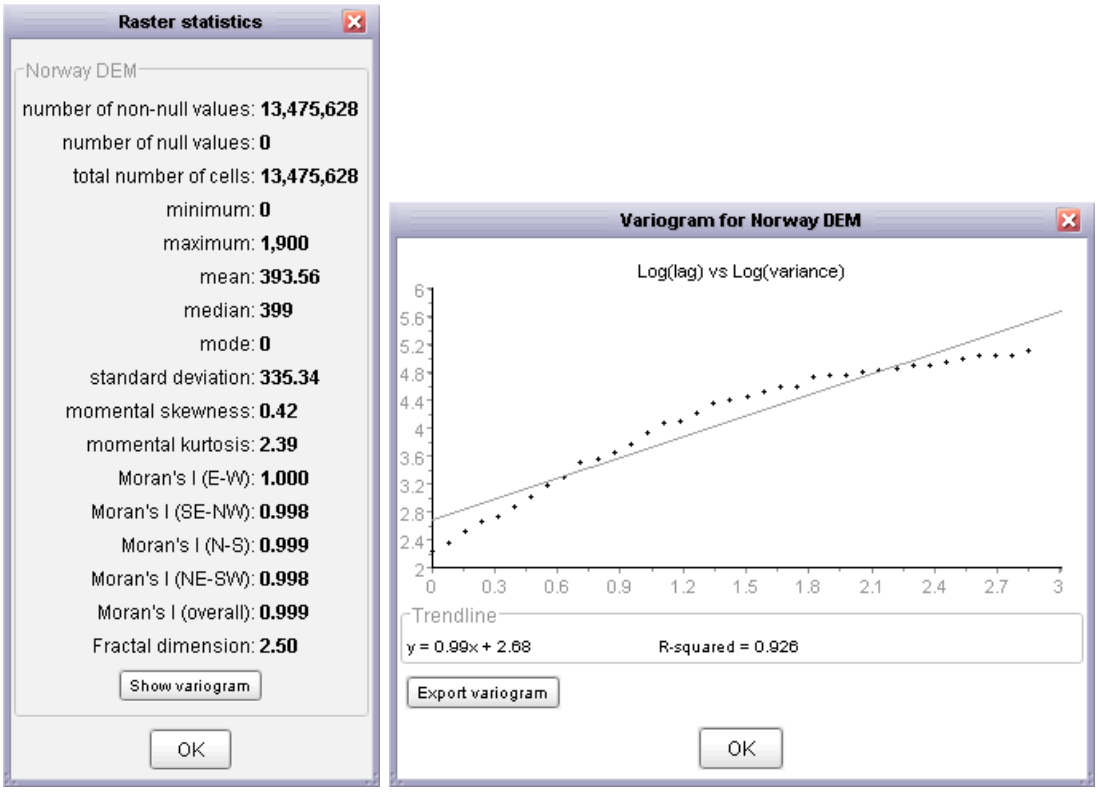


Figure 2.50: Raster statistical summary and log-log variogram.

the `Info→Query Map` menu item or by toggling the button. By moving the mouse over the main LandSerf display, location and attribute associated with the current mouse position will be displayed at the bottom of the main window. A permanent record of these query results are displayed in the *LandSerf console* and recorded in the LandSerf *log file*. If a spatial object is associated with an attribute table, the value displayed will be determined by the *active attribute* selected from that table. This allows textual as well numerical values to be displayed. If a secondary raster or primary vector is selected, clicking on a location while in query mode will display both these attributes while highlighting the selected vector object on screen.

2.6.2 Graphical Output

Displaying Raster Values

In addition to query with a mouse, the numerical value of each raster cell can be displayed directly in the mainLandSerf window. To activate numerical display, select `Numeric raster values` in the `Display`menu. On displaying the raster, numeric values will only appear if individual cells are large enough to display them. You may therefore have to zoom in to see them displayed. For floating point values, the number of decimal places displayed will depend on the size of each raster cell in the display window (see Figure 2.51).

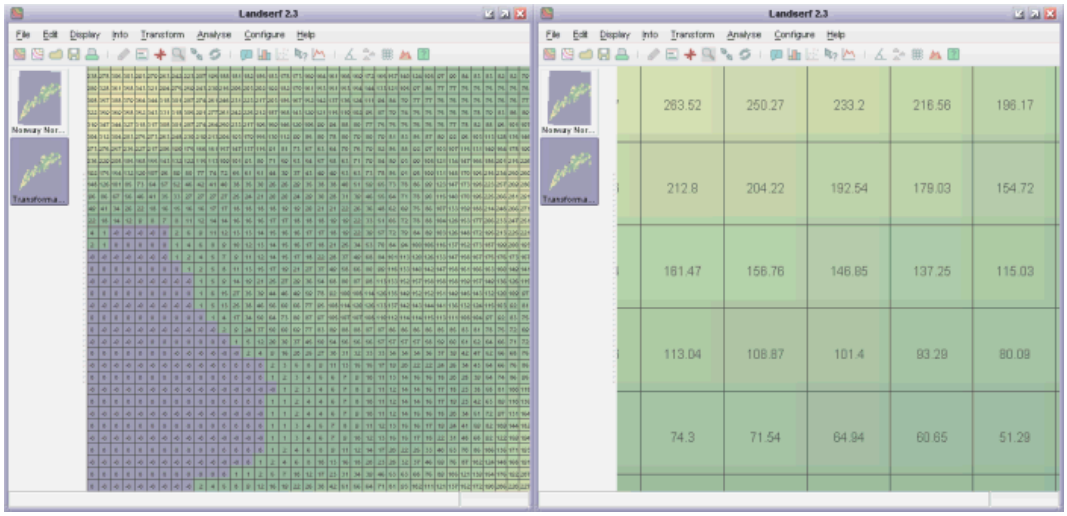


Figure 2.51: Numeric raster display showing higher precision when zoomed in (right).

Surface Profiles

To query more than one raster cell value at a time, cross-sectional profiles can be displayed by selecting either the `Info→Profile` menu item or the button. By clicking somewhere on the main raster display and dragging the mouse to another location, linear cross-sections are displayed in the profile window. The labels along the X-axis give the distance from the first point in the profile in ground units. The number of sample points along the profile can be controlled by the slider at the bottom of the profile window (see Figure 2.52).

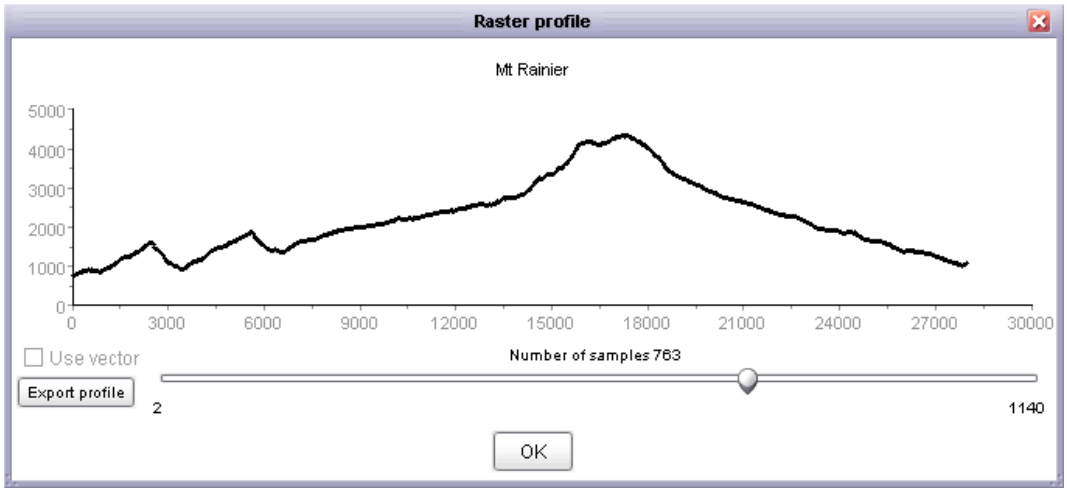


Figure 2.52: Interactive elevation profile output.

If you have a vector map selected as a primary vector in addition to a raster elevation model, you have the option of viewing the profile along the selected vector. To do this, select the `Use vector` option then click on the chosen vector in the main LandSerf window. This vector will be highlighted on screen and the profile updated accordingly.

As with all graph output, you can export the distance/elevation values as a simple text file for handling by other software (e.g. spreadsheets) by clicking the `Export profile` button.

Surface Parameters

It is possible to query various surface parameter values interactively using the mouse. To do this, make sure a DEM is displayed and select the `Info`→`Multi-scale query...` from the Info menu. This opens a dialogue box asking for the window scale and parameter type to be selected (see Figure 2.53 and *Performing Analysis on Surfaces* for more details on these parameters).

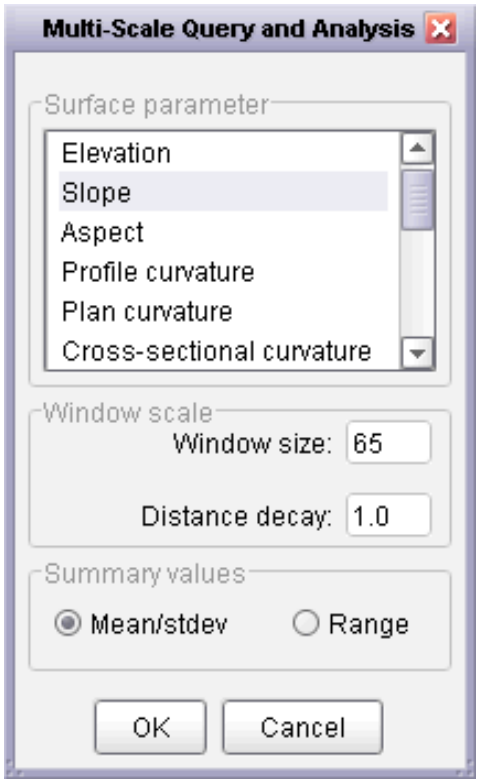


Figure 2.53: Multi-scale query options.

By dragging the mouse over the main display, output similar to Figure 2.54 is produced. The graph shows the value of the selected parameter (slope, curvature, feature type etc.) on the vertical axis, and the spatial extent over which the parameter was measured on the horizontal axis. Thus the curve produced shows how the given parameter varies with scale.

The numeric values at the bottom of the window represent the location of the point being queried and either the mean and standard deviation of the queried parameter over all scales, or its minimum and maximum vales, depending on what was selected in the `Query Options` window. If aspect is selected as the parameter to query, the circular mean and standard deviation are displayed (see Figure 2.55 right). If categorical parameters such as feature type are selected, the mode and entropy are displayed (see Figure 2.55 left). The higher the standard deviation or entropy, the greater the scale dependency of the measure.

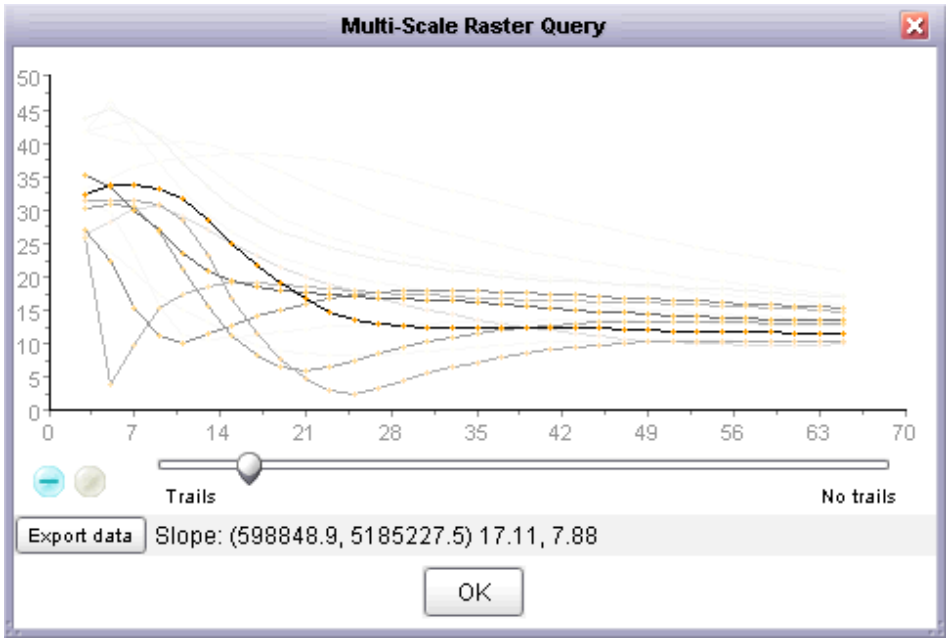


Figure 2.54: Interactive multi-scale query output with negative curve accumulation.

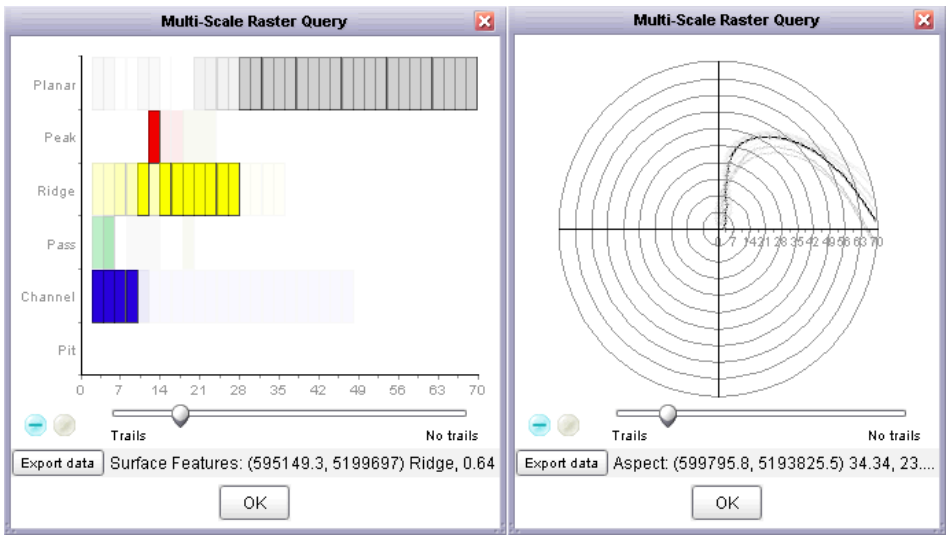


Figure 2.55: Interactive multi-scale output of feature type (left) and aspect (right) queries.

Curves are updated dynamically as the mouse is dragged over the surface. Old curves can be left on the display by moving the slider towards the `Trails` end. Leaving old curves on the graph allows the spatial variation in scale dependency to be shown as a query area is moved over a surface. They way in which old query curves are displayed can be controlled by toggling between the and buttons. Positive accumulation means that as successive lines are drawn over the same location, their representation becomes darker. Negative accumulation means that as curves are replaced with newer ones, they become progressively lighter. Pressing the button will clear the current graph display.

Scatterplots

To compare the values of two rasters, select the `Info→Scatterplot` menu item or the  button. This will plot the currently selected primary raster as the independent variable on the horizontal axis, and the secondary surface as the dependent variable on the Y-axis axis. The number of samples taken from the two rasters can be controlled with the slider (see Figure 2.56). The best-fit straight line along with its equation and R^2 value are also given, and will change dynamically as the number of samples is changed.

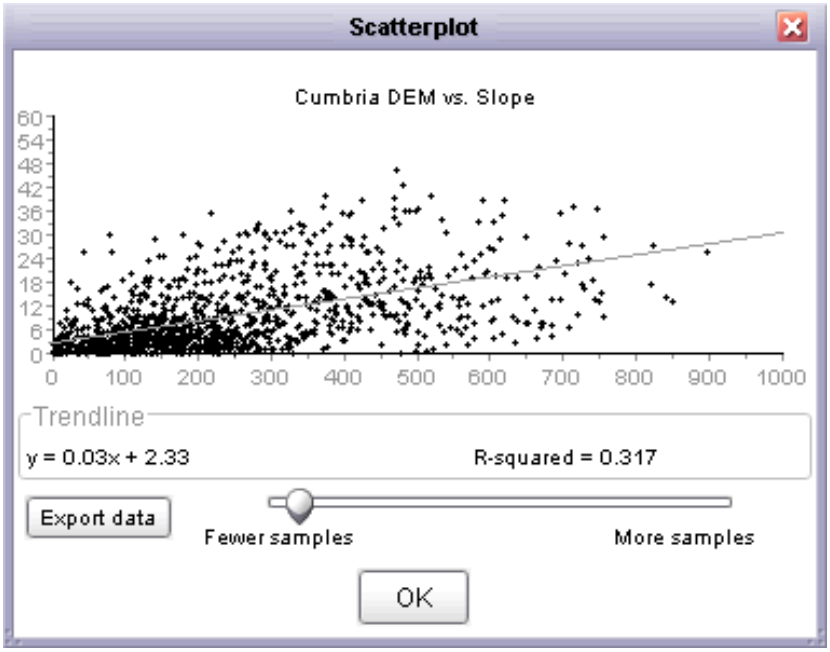


Figure 2.56: Scatterplot output comparing elevation (X) and slope (Y).

Frequency Histograms

To examine the frequency distribution of a raster, select either the `Info→Histogram` menu item or the  button. This plots a frequency histogram of the primary raster using its colour table. The class width of the histogram can be controlled via the slider at the bottom of the graph (see Figure 2.57). Moving the slider to the right increases the width of each class and therefore decreases the total number of classes. Surfaces that contain large flat areas such as lakes or sea can produce histograms dominated by the elevation of the flat region. Such areas can be removed from analysis by ticking the `Ignore values at` box and supplying an appropriate value or values. If more than one value is to be ignored, they should be separated by commas.

Many DEMs exhibit artifacts of the original contour lines that were interpolated to create the surface model. This can sometimes be detected by examining the frequency distribution of elevation values. For example, a DEM derived from 10m contour lines may show higher frequencies of 10m, 20m, 30m... elevations than 5m, 15m, 25m... elevations. This effect can be visualised by plotting the frequency histogram not of the elevation values directly, but the modulus (remainder) to the base of the suspected contour interval (see Figure 2.58). To do this, select

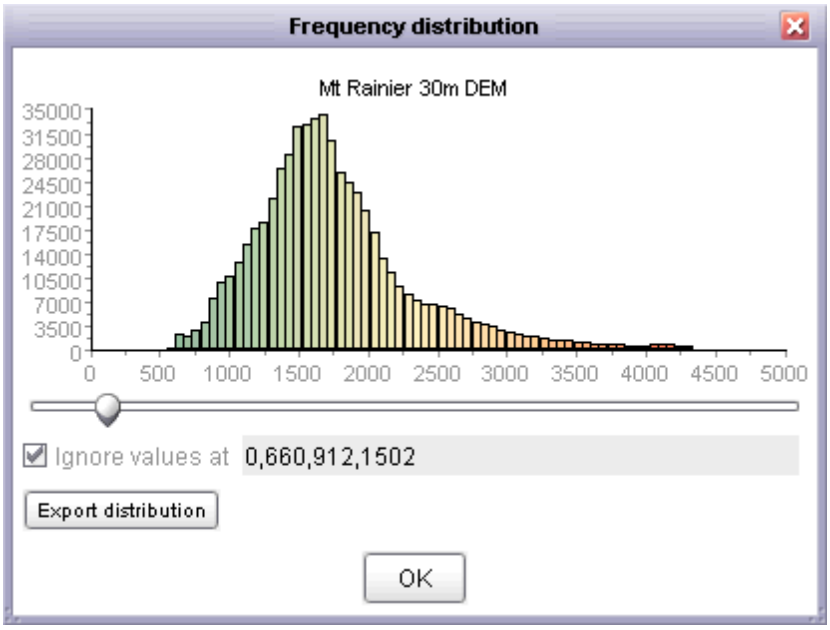


Figure 2.57: Frequency histogram of elevation surface.

the `Info→Hammock plot` menu item. DEMs that show this effect will result in a 'U' shaped hammock plot. DEMs that do not show this effect will typically show a rectangular distribution.

Note that large flat areas (e.g. lakes or sea) will significantly affect the hammock distribution, so as with the frequency histogram, user-defined values can be ignored in the calculation. The modulus value used for calculation of the plot defaults to 10, but can altered by moving the slider.

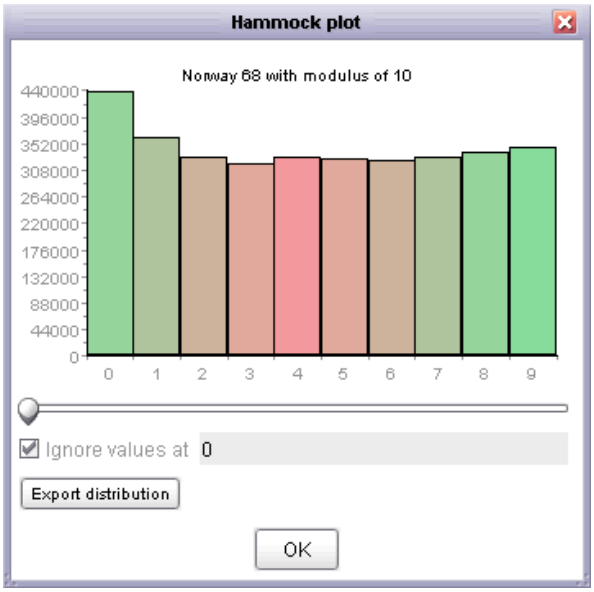


Figure 2.58: Hammock plot of elevation surface showing bias towards multiples of 10.

2.7 Performing Analysis on Surfaces

2.7.1 Surface Parameter Measurement

Once a raster DEM has been selected as a primary raster, it is possible to calculate a range of surface parameters at any scale. The scale of analysis is set by selecting either the `Configure`→ `Window scale...` menu option or the button. The window (or kernel) size is set as the number of cells along one side, so a value of 3 gives 9 cells, 4 gives 16 cells, 5 gives 25 cells etc. The value must be odd and less than the size of the smallest side of the raster. In addition, a distance decay exponent can be set that determines how important cells nearer the centre of the window are in comparison with those towards the edge. A value of 0 gives equal importance to all cells, a value of 1, an inverse linear distance decay, and a value of 2, an inverse squared distance decay. Negative and non-integer values are also permitted.

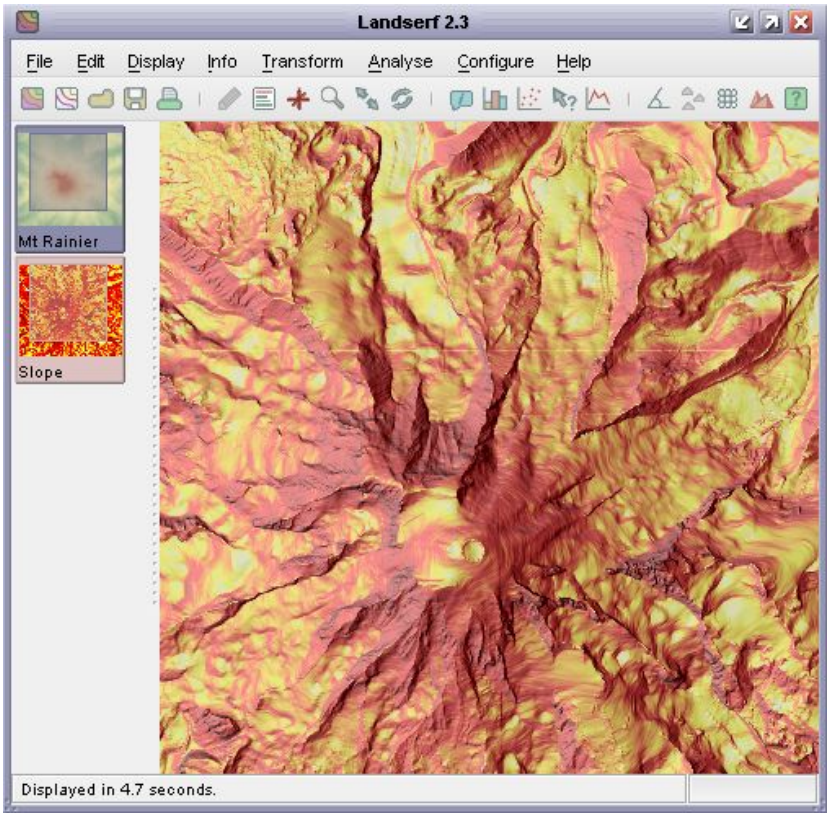


Figure 2.59: Slope map draped over a DEM.

The parameter to be calculated is chosen by selecting the `Analyse`→`Surface parameter...` menu item or the button (see Figure 2.59 for an example of the slope parameter). This opens a dialogue box allowing a choice to be made from a range of surface parameters (elevation, slope, aspect, 7 measures of curvature etc.). Calculation of these parameters at broader scales (i.e. using larger window sizes) can take some time, depending on the size of the DEM to process.

2.7.2 Surface Feature Classification

Surface features detection can be applied in much the same way as surface parameters. The result of this process is a classification of a surface into 6 categories - pits, channels, passes, ridges peaks and planar regions (see Figure 2.60).

Figure 2.60: Surface feature types.

The processing scale is again set by selecting `Configure→Window Scale...` option. Choosing `Configure→Feature extraction...` from the same menu brings up a dialogue box asking for two tolerance values. The *slope tolerance* value determines how steep the surface can be while still being classified as part of a pit, pass or peak feature. Larger values of this parameter tend to increase the number of point features detected, The *curvature tolerance* value determines how convex/concave ('sharp') a feature must be before it can be considered part of any feature. Curvature is recorded as a dimensionless ratio, with typical tolerance values ranging from 0.1 to 0.5. Larger values tend to increase the proportion of the surface classified as planar, leaving only the sharpest features identified. To classify features select `Feature extraction` from the list of options after choosing the `Surface parameter` ( ) item (see Figure 2.61).

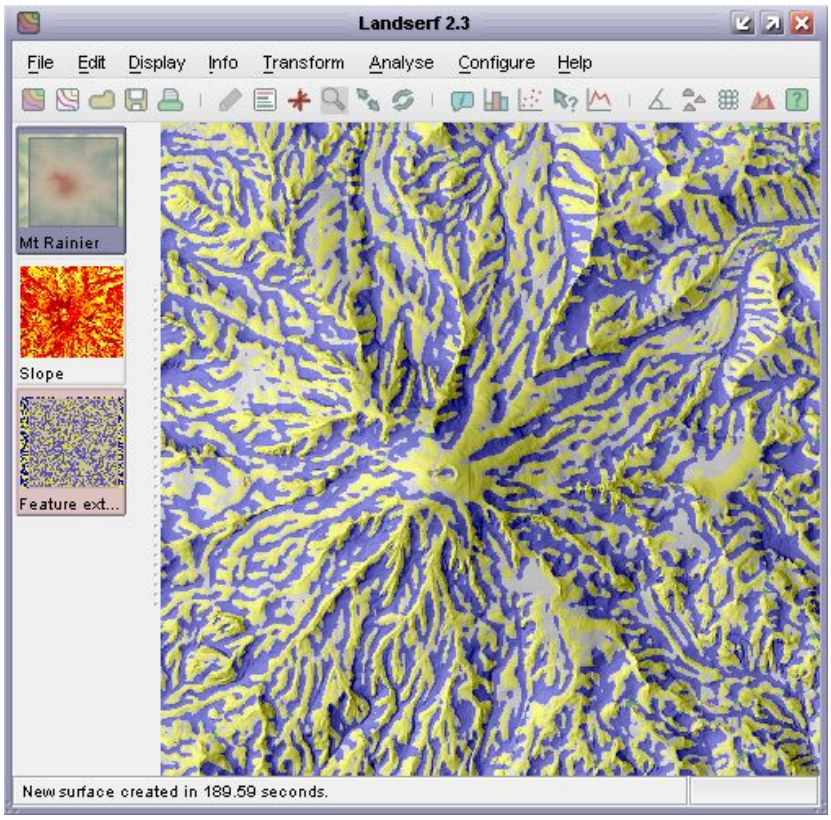


Figure 2.61: Surface feature classification draped over a DEM.

Alternatively, it may be desirable to ensure that features detected are filtered through a particular scale to ensure that channels and ridges remain as linear as possible. This may be achieved by selecting `Feature network extraction` from the `Surface Property Selection` window. This is the first stage required to produce a *Vector feature network*. If a DEM is selected as the primary raster and a surface feature layer selected as the secondary raster, the `Analyse`→`Vector feature network` () will become available. This will attempt to produced a connected set of linear ridges and channels that intersect as pass points on the surface (see Figure 2.62).

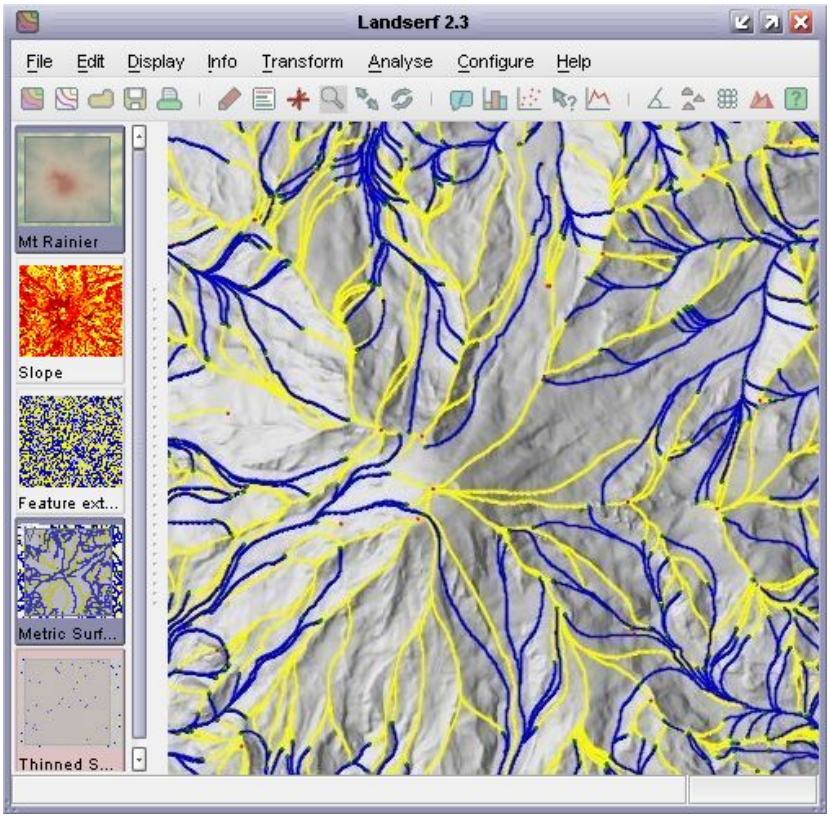


Figure 2.62: Vector surface feature network draped over a DEM.

A variation of the channel/ridge network is to calculate the 'flow accumulation' surface of a DEM (`Analyse`→`Flow accumulation...`) (see Figure 2.63). This process estimates the amount of water that would flow over each cell in the DEM assuming that each cell received a fixed amount of 'rain' plus any water that might have flowed into the cell from higher neighbours. It can be a useful way of estimating the hydrological characteristics of a surface and the likely channelized flow of water.

The generation of the flow accumulation layer is best performed on DEMs with pits removed in order to stop 'sinks' from disrupting the flow network. If a vector channel layer is to be generated, a threshold value of flow accumulation can be chosen below which no channel is assumed to be formed (see Figure 2.64). The higher the value of the threshold, the fewer the channels that are generated.

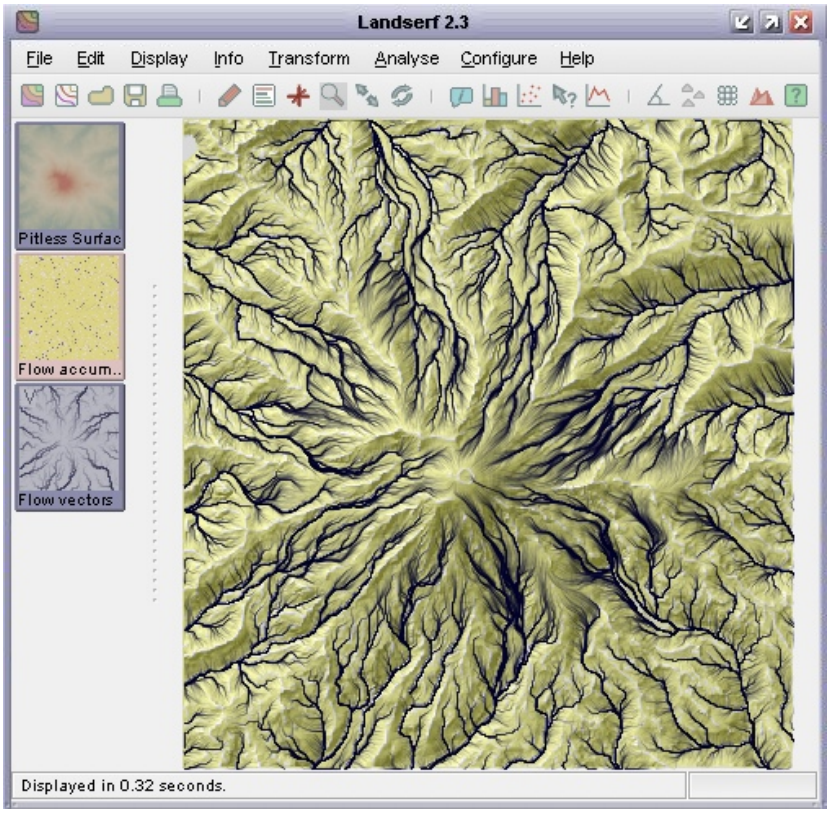


Figure 2.63: Flow accumulation surface with vector channel overlay.

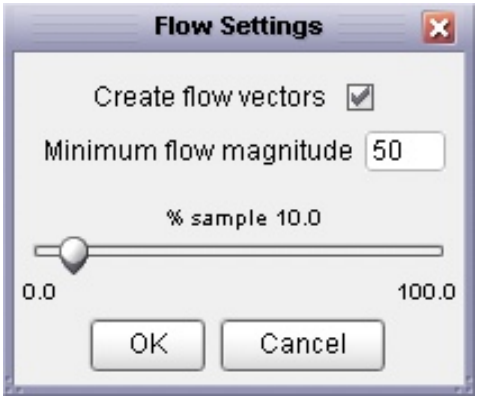


Figure 2.64: Flow accumulation options.

2.7.3 Multi-scale Analysis

It is possible to calculate any surface feature/parameter over a range of window sizes, and examine their dependency on scale. By selecting the `Analyse→Multi Scale parameter...` menu item any parameter can be calculated at all scales up to a specified maximum window size. The results of this process is the creation of two surfaces, one containing a measure of the average parameter value, the other a measure of its variation with scale. The exact

form of these measures depends on the type of parameter/feature being analysed. Details of the measures are given in the 'notes' section of the surfaces produced. Note that due to the computational intensity involved in deriving these measures, processing may take some time. Progress is indicated in the bottom right hand corner as a percentage complete bar and can be stopped at any time by clicking on it.

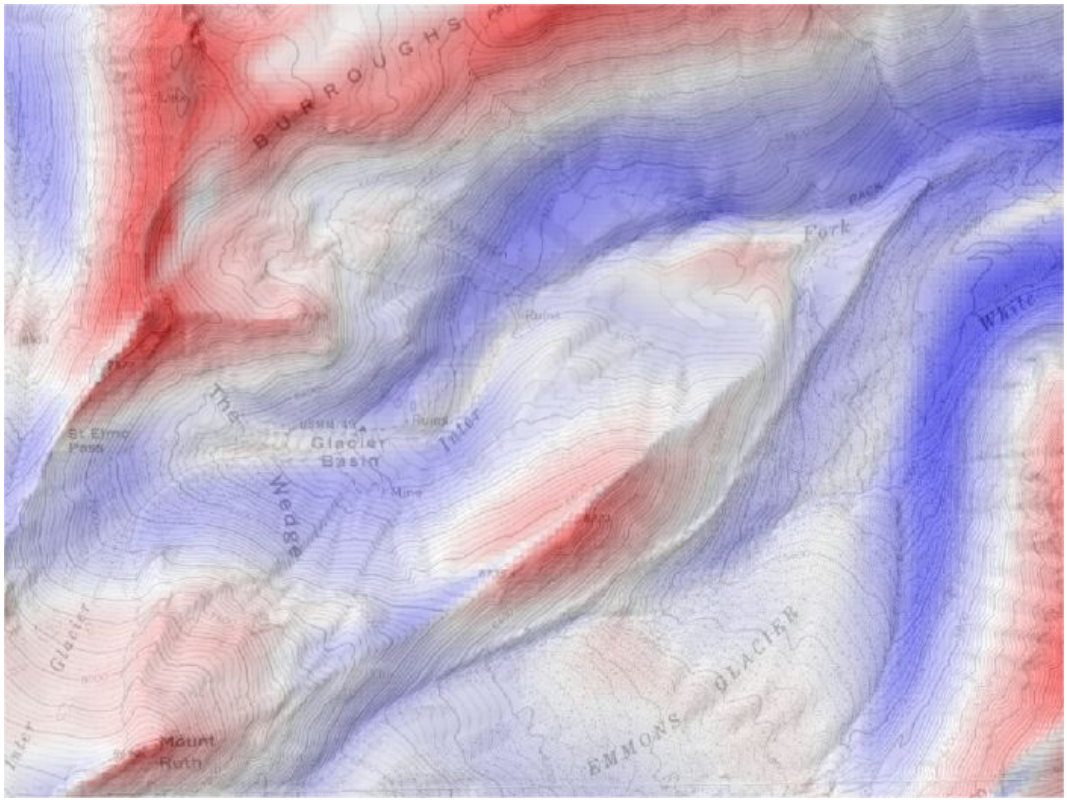


Figure 2.65: Multi-scale profile curvature mean.

The detection of features (peaks, channels, pits etc.) at a range of scales can result in the same location on the ground being classified into a number of different features at different scales. The most frequently classified feature type for any given location can be identified with using the muti-scale parameter as described above. However it can be useful to use this information to create *fuzzy feature maps* that indicate the degree to which a location can be regarded as a pit, peak, channel etc. Selecting the `Analyse`→`Fuzzy feature classification` menu item will produce a set of 6 raster maps, each indicating the membership of one of the feature types when classified using windows from 3x3 to the currently defined window size (see Figure 2.66). Each raster cell is scaled from 0 (indicating the cell is never classified as a given feature) to 1 (indicating the cell is classified as the given feature at all window scales). The results of applying this process are shown in Figure 2.66. Note that this can be a computationally intensive process and can take several hours for large surfaces with large maximum window sizes).

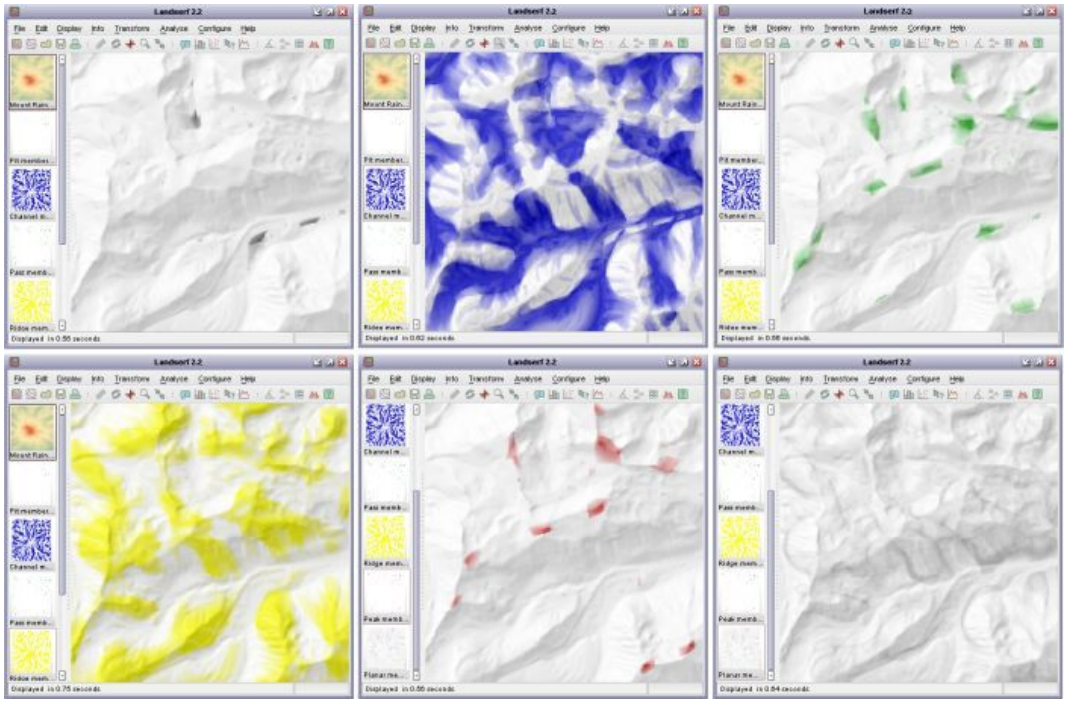


Figure 2.66: Multi-scale fuzzy features (pits, channels, passes, ridges, peaks, planar).

2.7.4 Other forms of Feature Detection

LandSerf offers an alternative approach to identifying certain types of surface feature. Rather than attempt to classify by measuring morphometry (shape) as described above, it is possible to detect pits and peaks by comparing the relative heights of candidate points. For example, a peak can be defined as a location that is entirely surrounded by other locations that are lower by a given amount.

Identification of pits and their subsequent removal from a DEM can be useful as the first stage of hydrological analysis where it is assumed that water can only flow from one DEM cell to another that is lower or of the same elevation. Selecting `Analyse→Pit removal` will identify pits in a DEM and 'flood' them such that there are no locations on the surface entirely surrounded by higher DEM cells. The location and depth of all flooded pits are also produced as a separate output raster (see Figure 2.67).

A similar approach can be taken to the identification of summits and peaks in a landscape. Selecting the `Analyse→Peak classification...` menu option will display a dialogue window asking for two identification criteria to be identified. `Minimum height of peak` allows an elevation threshold to be identified below which no peaks can be classified. This simple thresholding allows portions of a landscape equivalent to the Scottish *Munros* (peaks over 3000 ft in height) to be identified. However, this criterion alone is not sufficient to separate portions of a surface that do not dip below that threshold. A second criterion, the `Minimum drop surrounding peak` can be entered that ensures that a peak must be entirely surrounded by elevations that are lower by the given amount. This allows for more refined definition of distinct peaks. Either or both criteria can be selected in LandSerf.

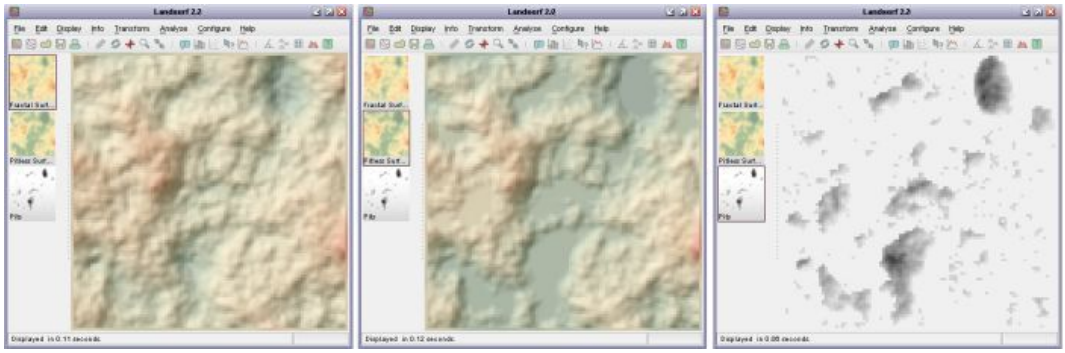


Figure 2.67: Pit removal applied to fractal surface. Original surface (left), flooded surface (middle) and pits (right).

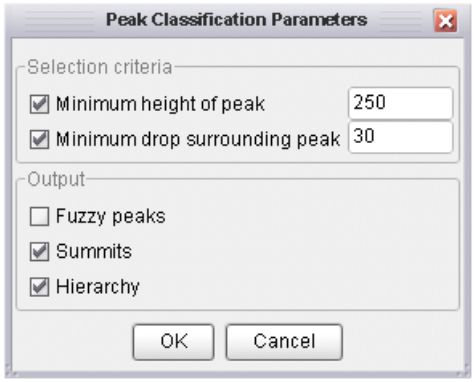


Figure 2.68: Peak classification options.

Output is a map of summits (highest point on a peak) in red and peak extents in yellow. Note that the extent is defined solely by the supplied criteria. So for example, if a minimum drop of 100m is specified, the extent of the peak will be all parts of the surface that are vertically within 100m of the summit elevation (see Figure 2.69).

More sophisticated peak classification can be chosen by selecting `Fuzzy peaks`, `Summits` and `Hierarchy` from the peak classification window. Fuzzy peak selection produces a raster of the degree of peakedness of every cell in the DEM. Summit points of peaks that have a drop of at least that defined by the minimum drop criterion will have a membership of 1. This decreases to 0 down the side of the peak as it approaches 'minimum drop' vertical units below the summit.

Selecting the `Summit network` option will generate a vector map containing the summits of all peaks that meet the selection criteria. The attribute table associated with the map also contains the relative drop of each summit, its connected pass and the topological ridge connecting each summit to its pass. Selecting the `Hierarchy` option additionally identifies 'peaks within peaks'. The catchment area of each of these sub-peaks is shown in Figure 2.70, along with the relative drop of each summit. If `Hierarchy` is selected, the `Summits` option will additionally produce the vector network that connects each summit to its parent summit.

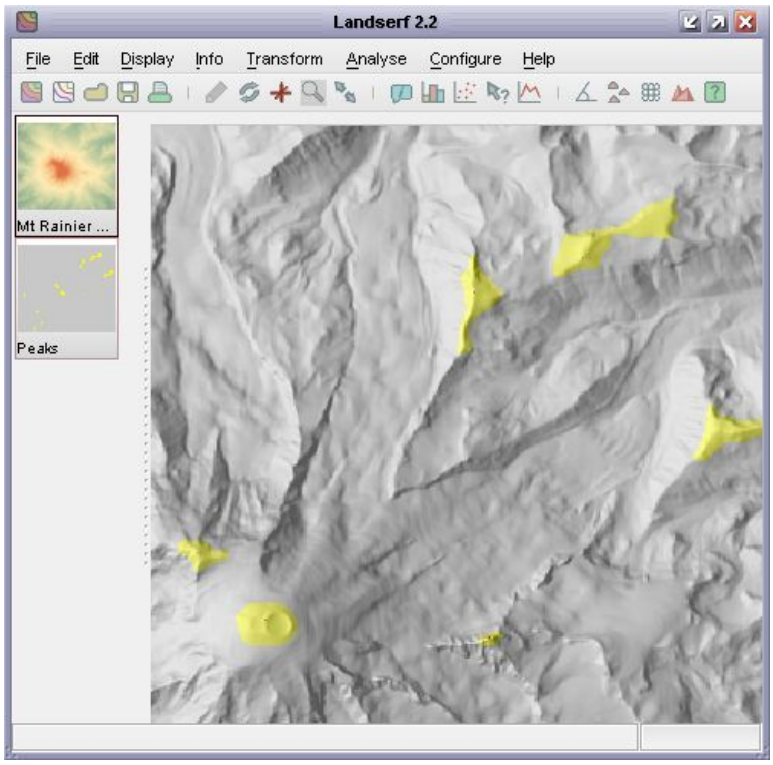


Figure 2.69: Detection of all peaks with 100m relative drop.

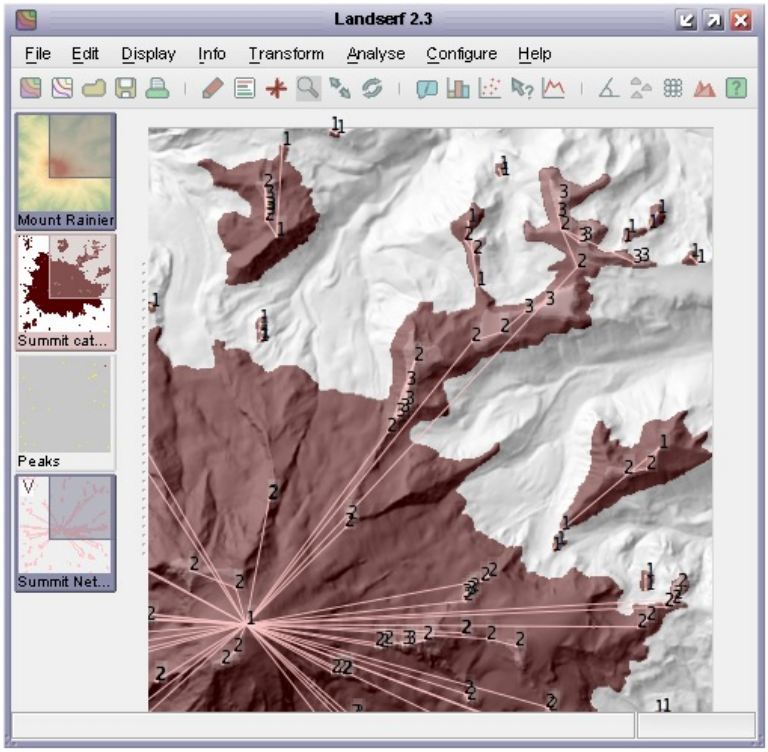


Figure 2.70: Sub-peaks and summit network with at least 20m relative drop.

3 'How to' Guides

3.1 Importing Data into LandSerf

The way in which you import elevation models into LandSerf will depend on the location and resolution of the surface you wish to model.

3.1.1 Global Datasets

One of the most useful global coverages of elevation data is provided by the *Shuttle Radar Topography Mission* (SRTM) dataset (see www2.jpl.nasa.gov/srtm). Elevation data at 3 arc-second resolution (about 90m E-W at the equator) are available for the Americas, Europe, parts of Asia, Africa and Australia from dds.cr.usgs.gov/srtm/version2_1/. These can be imported directly into LandSerf by selecting the SRTM format when opening a raster. Note that as with other global datasets, after import, the data should be reprojected into some planar projection such as UTM (select the appropriate option from the `Transform→Reproject` menu). Additionally, SRTM data are likely to contain several 'voids' where the sensor was unable to record an accurate elevation. These can be removed by selecting `Void removal` from the `Transform` menu. See the guide how to import SRTM data for further details.

In June 2009, the ASTER global elevation data at an approximate resolution of 30m was released by NASA/METI. While a significant improvement over SRTM in terms of the precision and coverage of the dataset (it spans latitudes between 83 degrees north and south), the data quality of the first release limits its use. Initial assessments suggest significant pre-processing is required to remove artifacts before detailed hydrological and morphometric analysis can be performed. The data can be imported directly into LandSerf as GeoTiff format files (LandSerf 2.3.1 or above). Access to the data is free, but requires users to log in and agree to a licence on use and distribution. See www.gdem.aster.ersdac.or.jp/.

Coarser resolution global elevation at approximately 1km resolution are available from the freely available GTOPO30 dataset. These data are compiled from a variety of sources and include the entire global landmass (bathymetric data are not included in this dataset, but see below for alternatives). To import GTOPO30 data, extract the files from the compressed tar file

(Windows users make sure that if you are using WinZip to do this, that 'TAR file smart CR/LF conversion' is *not* set). Select `GTOPO30` as the file format in LandSerf to import the .dem file.

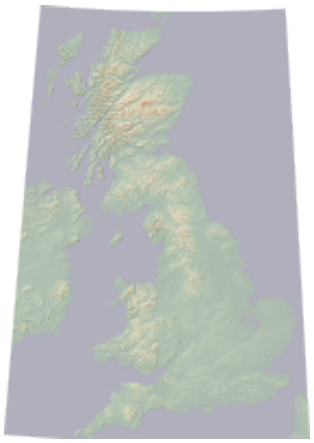
Bathymetric (see floor) data at a 2 minute resolution are freely available in the form of the ETOPO2 V.2 dataset. Data are distributed in ArcGIS ASCII grid format so can be directly imported into LandSerf.

The SRTM30Plus global dataset provides an integration of SRTM data covering global landmasses, bathymetic data for sea floor surfaces and GTOPO30 data for latitudes above +-80 degrees. Data are tiled into 33 regions of the globe at a resolution of 30 arc-seconds (~1km) and are stored in the same format as SRTM data so can be imported directly into LandSerf.

A number of freely available sources of landcover and remote sensing are available for much of the globe. For both, http://glcfapp.umiacs.umd.edu:8080/esdi provides a good start and can be used to provide data as georeferenced GeoTiff files that can be imported directly into LandSerf.

Data for other areas of the world are likely to be provided in a range of formats. You may wish to consult the details on file formats to see how they may be imported. Another useful source of information is the Virtual Terrain Project site, which includes a list of international elevation model sources.

3.1.2 United Kingdom Elevation Data



In the UK, the most widely used DEM coverage is provided by the Ordnance Survey (www.ordnancesurvey.gov.uk). They provide DEMs at 50m ('Panorama'), 10m ('Profile') and 2m ('Profile-plus') resolutions for Britain. DEMs from the Ordnance Survey are usually provided in *National Transfer Format* (NTF V.2). Data in this format can be immediately imported into LandSerf by selecting the `Ordnance Survey NTF Raster DEM` format from the `Open` file dialogue. Contour data are also distributed by the Ordnance Survey derived from their 1:50k maps ('panorama') and 1:10k maps ('profile'). These are also distributed in NTF format and may be imported into LandSerf by selecting the `Ordnance Survey NTF Vector file` format.

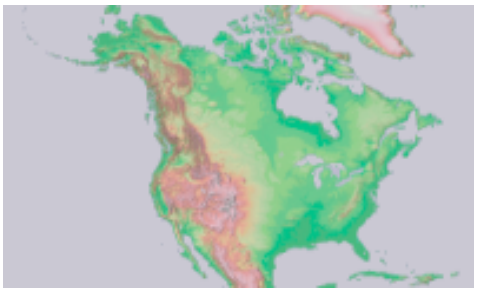
The UK Environment Agency (www.environment-agency.gov.uk) provide 2m resolution LiDAR DEMs for many coastal and inland areas in England and Wales, especially for areas susceptible to flooding. These are usually provided in ArcGIS ASCII raster format and so can be directly imported into LandSerf.

The LandMap project, (www.landmap.ac.uk) provides a 25m DEM of the British Isles to UK higher educational institutions along with some selected LiDAR DEMs. The 25m DEM is based on multiple passes of the ERS SAR sensor. Data can be downloaded in ArcGIS ASCII raster format, so can therefore be imported directly into LandSerf.

There are also various providers of global elevation data including the UK (see below). In many cases these data will be referenced using global latitude/longitude coordinates. Such data should be reprojected onto the UK National Grid by selecting the appropriate option from from Land-

Serf's `Transform→Reproject` menu.

3.1.3 United States Elevation Data



The main provider of elevation data for the US is the *United States Geological Survey* (USGS), who provide DEMs (and other digital spatial data) at a variety of scales and resolutions. The USGS now distribute their National Elevation Dataset (NED) - a continually updated seamless 30m DEM of the United States. Details can also be found in an article in the January 2002 edition of Photogrammetric Engineering and Remote Sensing.

Additionally available from the USGS are parts of the global *Shuttle Radar Topography Mission* (SRTM) dataset (see srtm.usgs.gov and www2.jpl.nasa.gov/srtm). Elevation products from this mission include 90m resolution DEM of the American continent and a 30m resolution coverage of the US.

The easiest way to download these data is to use the USGS's seamless.usgs.gov website that allows arbitrary regions to be selected from the US and some other global sites. Also available from the same site are selected landcover, vector and orthophoto coverages of the US. To import raster data from this site into LandSerf, select the BIL (for integer data), GridFloat (for floating point data) or TIFF (for images) format from the USGS pages and import using the relevant format in LandSerf. Note that all data are georeferenced using global latitude/longitude coordinates, and should therefore be reprojected onto a planar UTM coordinate system using the appropriate option from the `Transform→Reproject` menu.

The USGS also provided their older tiled '7.5 minute' DEMs in SDTS format. These can be found by selecting the '1:24k DEM' option from the Geodata web site. If you download data in this format you will next need to convert these files into a form readable by LandSerf. The easiest way to do this is to use one of the utilities provided by the USGS to convert SDTS into ArcGIS or 'USGS Native DEM' format (e.g. see www.cs.arizona.edu/topovista/sdts2dem/ for some downloadable converters.

The principal advantage of these older data is that being tiled they correspond to other datasets produced by the USGS such as their Digital Raster Scanned topographic maps.

3.2 Importing Shuttle Radar Topography Mission (SRTM) elevation data

The Shuttle Radar Topography Mission is a collaborative project between NASA, the National Imagery and Mapping Agency (NIMA), and the German and Italian space agencies to map the elevation of a large part of the globe. Using Synthetic Aperture Radar (SAR) attached to the Endeavor space shuttle, the mission was flown for 11 days between 11th and 22nd of February 2000. Subsequently, the data collected were processed and have been released as DEMs for the majority of the earth's landmass. See www2.jpl.nasa.gov/srtm for details on the background to the project.

3.2.1 Step 1. Downloading .hgt files

SRTM DEMs at a 3 arc-second resolution (~90m at the equator) are freely available for most of the globe. Additionally, 1 arc second (~30m at the equator) SRTM DEMs are available in various formats for the continental United States and a combined SRTM, GTOPO30 and bathymetry dataset at 30 arc-seconds, known as srtm30Plus is available for the entire globe. SRTM data are often distributed in '.hgt' format as described in the SRTM documentation. Data for the United States are also distributed in other raster formats from the USGS seamless data server.

Note that there are currently 3 versions of SRTM currently being distributed. 'Version 1' SRTM data are the raw elevation values sampled and smoothed at either 1 or 3 arc second resolution and include voids and erroneous data over water bodies. 'Version 2' data have been post-processed in an attempt to remove voids, mask out sea areas and generally remove any known erroneous spikes in the data. SRTM data are also available on DVD from the EROS data center. These data have been post-processed but have been sampled without smoothing from higher resolution SRTM data. Consequently the data will appear slightly rougher compared with those downloaded from the servers described below.

Version 2 HGT format data for most of the globe between 60^{o}N and 60^{o}S (see Figure 3.1) and can be found at dds.cr.usgs.gov/srtm/version2_1/ organised by continent.

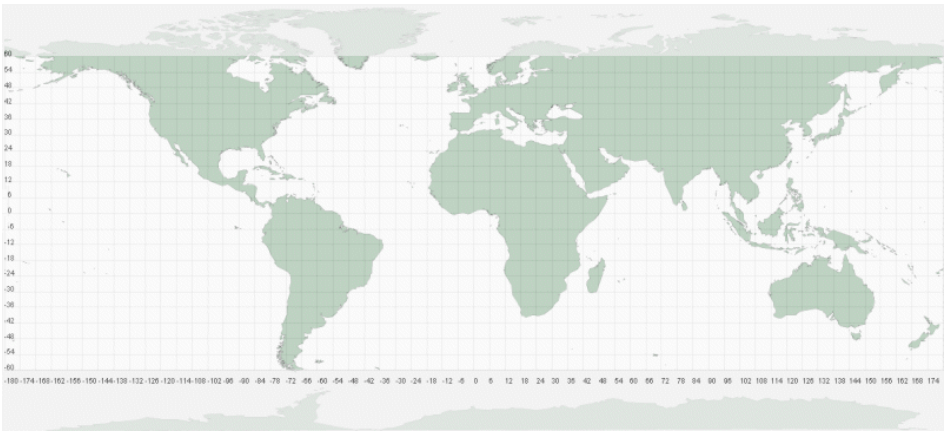


Figure 3.1: SRTM coverage. Each grid square represents 6 degrees, covering 36 SRTM tiles.

Each file covers a 1 degree by 1 degree section of the earth's surface and is named according to the latitude and longitude of its bottom left corner. So for example, the city of Bristol in the UK located at $51^o28'$ N, $2^o35'$ W can be found in the tile `N51W003.hgt` (see Figure 3.2. Tiles east of the Greenwich Meridian will be named with an 'E' rather than 'W', and those south of the equator with an 'S' rather than 'N'. It is important to retain the name of the tile when downloading data as this is used by LandSerf to attach the correct spatial metadata to the DEM.

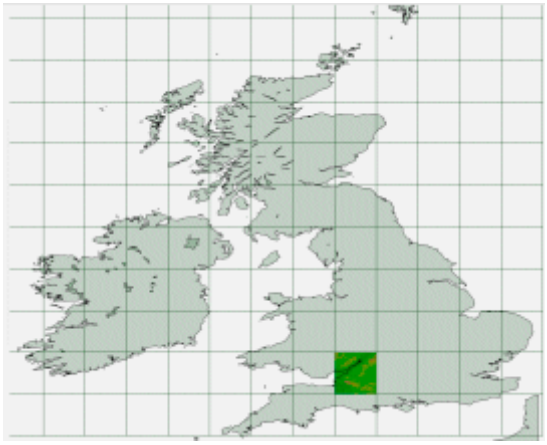


Figure 3.2: UK and Ireland tile locations with N51W003.hgt highlighted.

When wishing to download several tiles covering a continuous area of the globe, it is useful to identify the names of the tiles you wish to use in advance, since the download sites usually list them in alphabetical order and not necessarily in terms of an obvious geographical pattern.

Once downloaded, you should 'unzip' the files in a directory on your machine. You should then be able to import the `.hgt` files directly into LandSerf by selecting `Shuttle Radar Topography Mission DEM (.hgt)` file type from the file dialogue displayed by LandSerf's `File`→`Open` menu.

Note: If you are importing srtm30Plus files, the same guidelines above apply except that the file will have the extension `.srtm` rather than `.hgt`, and each tile is 40 degrees E-W by 50 degrees N-S.

3.2.2 Step 2. Removing Voids (*Version 1 data only*)

Note that the following step is unnecessary if you are importing srtm30Plus or Version 2 HGT files as they have already been processed to remove any voids.

Due to the nature of deriving elevation from Synthetic Aperture Radar, most Version 1 SRTM DEMs will contain 'voids' or cells where no elevation value could be found. This is especially the case for some water surfaces and patches of very rough terrain. In LandSerf voids will appear as light-grey gaps in the DEM. Usually, you will want to remove these void cells before processing further. You can do this either by combining the DEMs with other DEMS of the same area for which you have elevation data (e.g. GTOPO30 data), or more simply by selecting `Void removal` from the `Transform` menu. After selecting this option, select `Refresh` (or `Ctrl-R`) from the `Display` menu to see the effect of the void removal (see Figure 3.3).

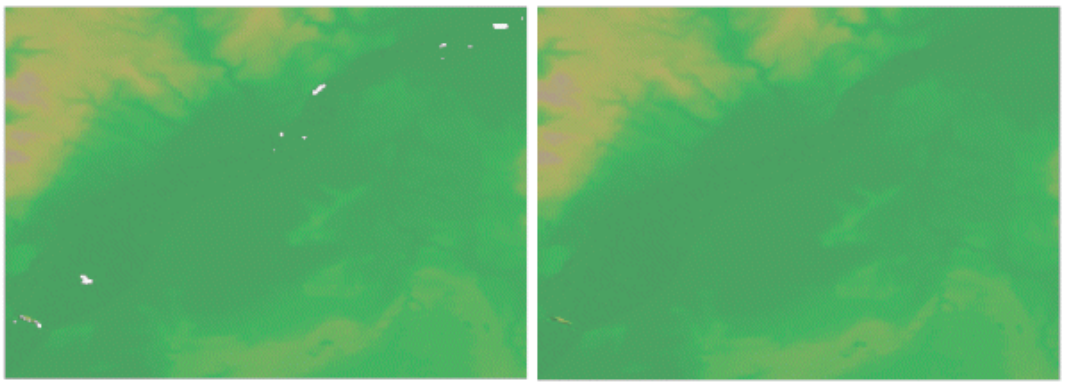


Figure 3.3: DEM before and after void removal.

Void removal works by replacing the outer 'ring' of a void with the average of its non-void neighbours, repeating the process until no voids remain. For most voids, which are usually small holes, this results in a pretty reliable interpolation. For larger patches, this can leave minor artifacts in the interpolated area that look like radiating 'spokes' from the centre of the void patch.

3.2.3 Step 3. Combining Tiles

The following step is unlikely to be necessary if you are importing srtm30Plus files. In fact, you are more likely to want to subset a smaller region from the large continental-scale tiles. To do this select `Edit`→`Edit raster` *then mark an area to subset with the mouse before selecting the* `Extract subset` *box and pressing* `OK`.

In many cases, you will wish to create a DEM that is made up of several contiguous SRTM tiles. To do this, you should import and process the first two tiles as described above. When you have two void-free tiles in LandSerf, make sure one is selected as the primary raster (left-click its thumbnail view) and the other selected as the secondary raster (right-click its thumbnail view). Choose `Combine rasters...` from the `Edit` menu.

Combining rasters allows you to merge the contents of two existing raster maps to create a new one. The new raster can be made up of the intersection of two overlapping rasters, or the union. In this case you should select the `Union` option so that the entire contents of both primary and secondary rasters are combined to create the new one (see Figure 3.4).

If the primary and secondary rasters overlap you can tell LandSerf which raster to prioritise for these cells. Since the SRTM data should be the same for any overlapping edge cells, this should not make any difference. The final option to '`Replace null values`' should also make no difference in this case, since we have removed all voids in the previous step.

After successfully combining two adjacent SRTM rasters you should have something similar to the Figure 3.5. To save resources, you can now delete the original primary and secondary rasters, and save the new combined one to disk. To build up the contiguous region further, you should repeat steps 1 to 3, but this time using the new combined raster as one of your input raster maps.

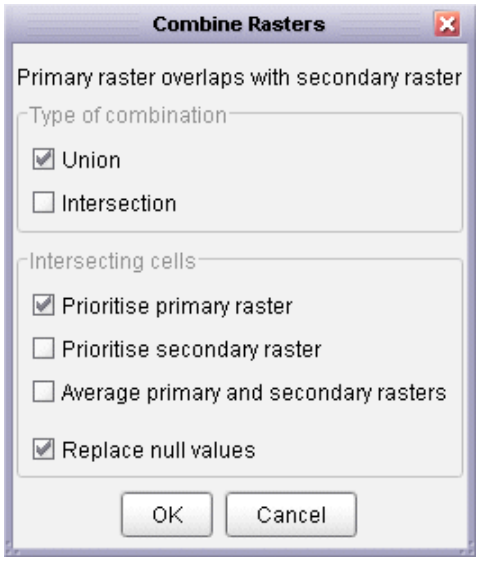


Figure 3.4: Raster combination options.

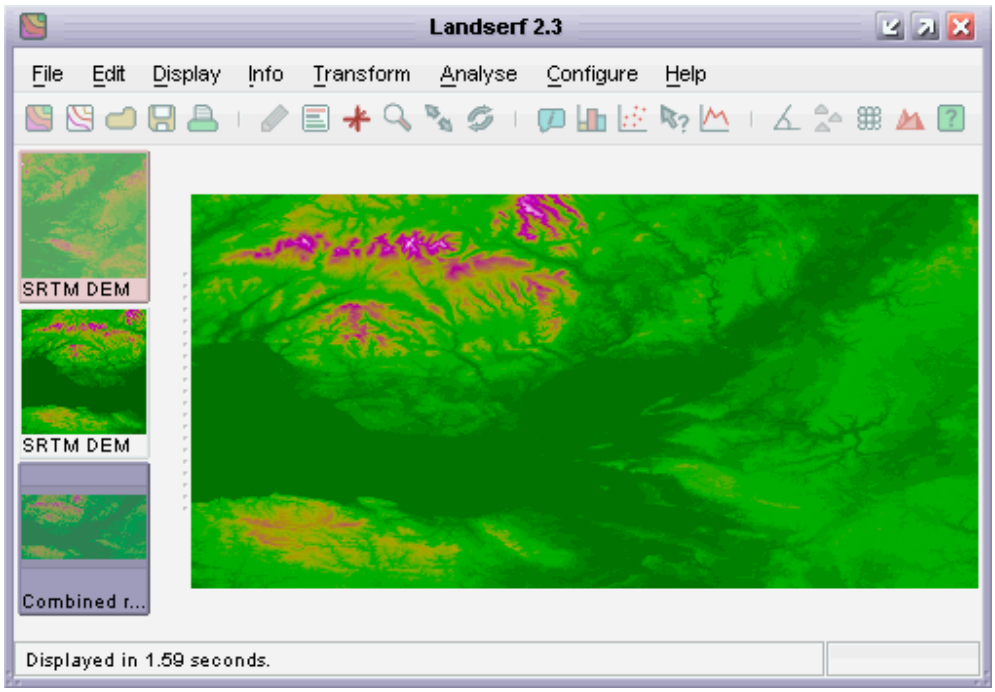


Figure 3.5: Combined rasters.

If you need to combine many tiles into a single DEM, you may find it easier to create a *LandScript* file containing the combination instructions (see Chapter 4 for details on creating and runningLandScript files). An example script is given below, which can be modified to use the names of the files you wish to combine.

```
# Combines SRTM tiles into a single raster.
version(1.0);

baseDir = "/users/jowood/data/srtm/";
raster = open(baseDir & "N51E001.hgt");

raster = combine(raster_,open(baseDir & "N51E000.hgt"));
raster = combine(raster_,open(baseDir & "N51W001.hgt"));
raster = combine(raster_,open(baseDir & "N51W002.hgt"));
raster = combine(raster_,open(baseDir & "N52E001.hgt"));
raster = combine(raster_,open(baseDir & "N52E000.hgt"));
raster = combine(raster_,open(baseDir & "N52W001.hgt"));
raster = combine(raster_,open(baseDir & "N52W002.hgt"));

save(raster,baseDir & "EastEngland.srf");
```

3.2.4 Step 4. Masking sea areas

The following step is not necessary if you are importing srtm30Plus (sea areas contain bathymetric sea floor data) or using 'Version 2' HGT files (sea already masked and set to 0m elevation).

If the DEM area you have selected contains sea in addition to land, it is sometimes convenient to mask out the non-land areas from analysis and display. This can be done in two ways. If you have a raster containing the land/sea boundary for the area of interest, you can set the land area to `null` and combine the raster with your DEM, making sure you select the '`replace null values`' tick box. In this way, only the land raster cells will be transferred to the new raster.

A simpler option which works reasonably well is to 'flood fill' all areas less than 0m elevation, and then classify all elevations of 0 as `null`. To do this, select your combined DEM as the primary raster and then select the `Transform`→`Raster values...` menu item. Select the `Flood` tickbox, leave the default value of 0, and press `OK`. This will produce a new DEM with all values that were originally less than 0 now classified as 0. Selecting that new raster, choose the `Transform`→`Raster values...` menu item again, this time selecting `Replace 0 with n` on the bottom line. Pressing `OK` should reclassify the DEM with all values that were 0 (sea) now classed as null. An example is shown in Figure 3.6

This is not a perfect method, as the SRTM data may well have identified some of the sea area as having an elevation of above 0m (small dots in the figure above). These can be masked by overlaying patches of null valued rasters in areas known to be sea, or by using *LandScript* to apply a median filter to cells surrounded by null values.

3.2.5 Step 5. Reprojecting the DEM

As SRTM data are provided in geographical (latitude/longitude) coordinates, they are generally unsuitable for most analysis unless the are projected onto a planar coordinate system where the (x,y) units are the same as the elevation units. LandSerf allows reprojection into a variety of coordinate systems including UTM, Ordnance Survey, French and Swiss national grids, all of which use coordinate systems in metre units.

To reproject, firstly check that the existing combined raster has the relevant projection metadata attached by selecting the `Edit`→`Edit raster...` menu option and pressing the `Edit` button

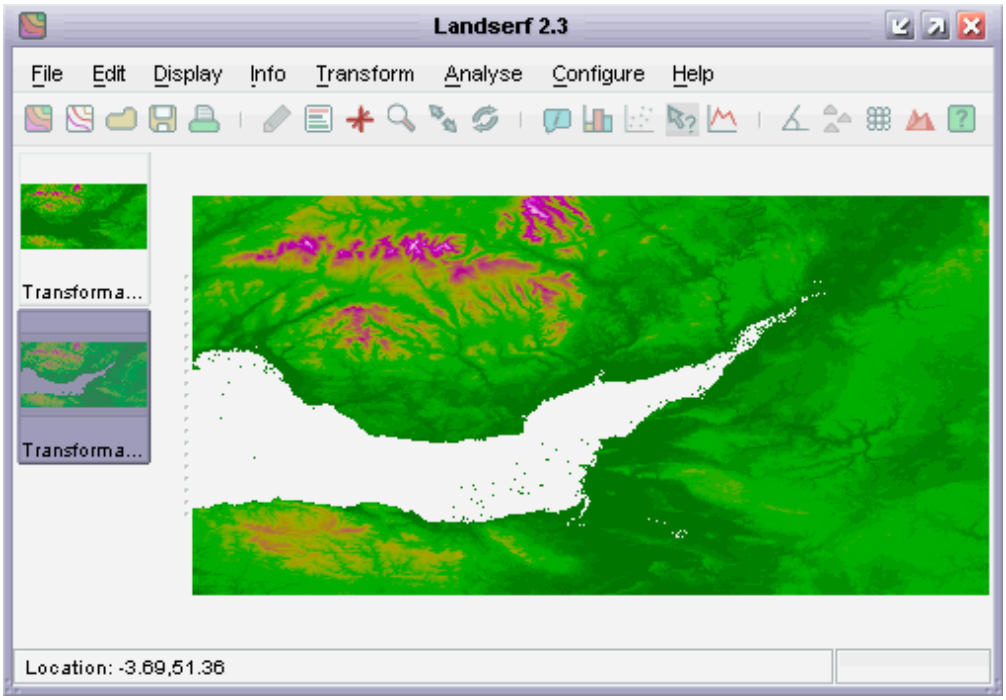


Figure 3.6: Combined rasters with water masked.

in the `Map projection` area. The map projection should be set to `Latitude/longitude` projection and `WGS84` ellipsoid. Press the `OK` buttons to confirm these values. To reproject the DEM, select the `Transform→Reproject...` menu, and choose an appropriate new projection from the drop down menu. Pressing `OK` will display a new dialogue window with the option for you to change any of the bounding coordinates or resolution of the new raster. Generally, you can accept the default values for the bounding area, but you should change those for the E-W and N-S resolution. The resolution of the unprojected DEM is about 90m at the equator, but decreases to about 60m at the northern and southern extremities of the mapped area. You should select the same resolution for N-S and E-W values so that the resulting DEM has square pixels. Typically, this should be in the range of 60-90m for the 3 arc-second data, 20-30m for the 1 arc-second data, or ~1000m for the 30 arc-second data, but can be finer or coarser than that if required. Pressing the `OK` button should perform the reprojection, producing a DEM ready for analysis.

If you are reprojecting srtm30Plus data (see Figure 3.8), beware of attempting to reproject data at very high latitudes (beyond +-80 degrees) to UTM. To avoid undue distortion, if possible, subset the data to be within at most +-80 degrees before attempting a reprojection.

3.2.6 LandScript Import

An example script that will import eight SRTM files, remove their voids, flood all elevations below 0 to sea level and add appropriate colour and title metadata is shown below. This can be adapted for importing and tiling any set of SRTM data by changing the appropriate file names and base directory, and possibly the new projection to use.

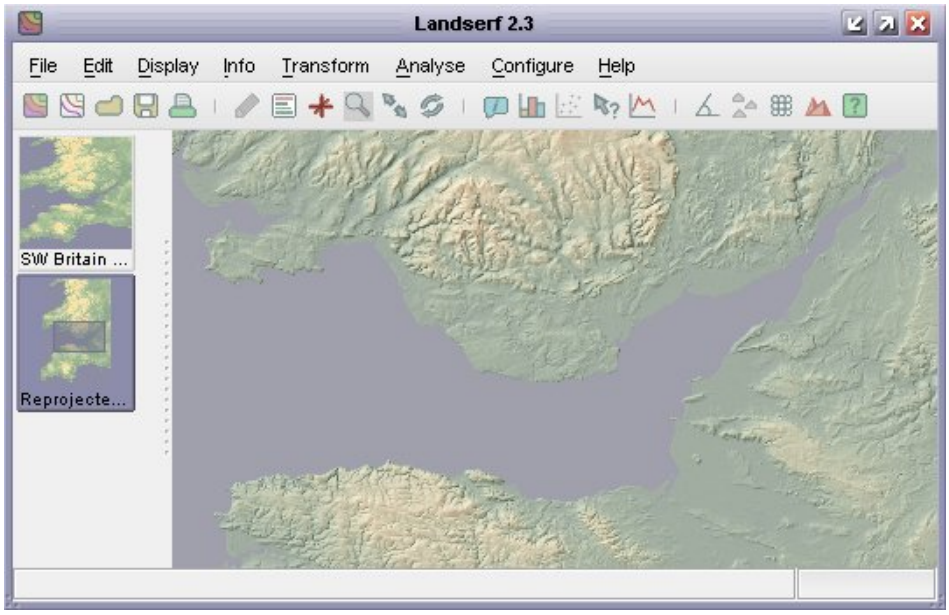


Figure 3.7: Reprojected DEM ready for analysis.

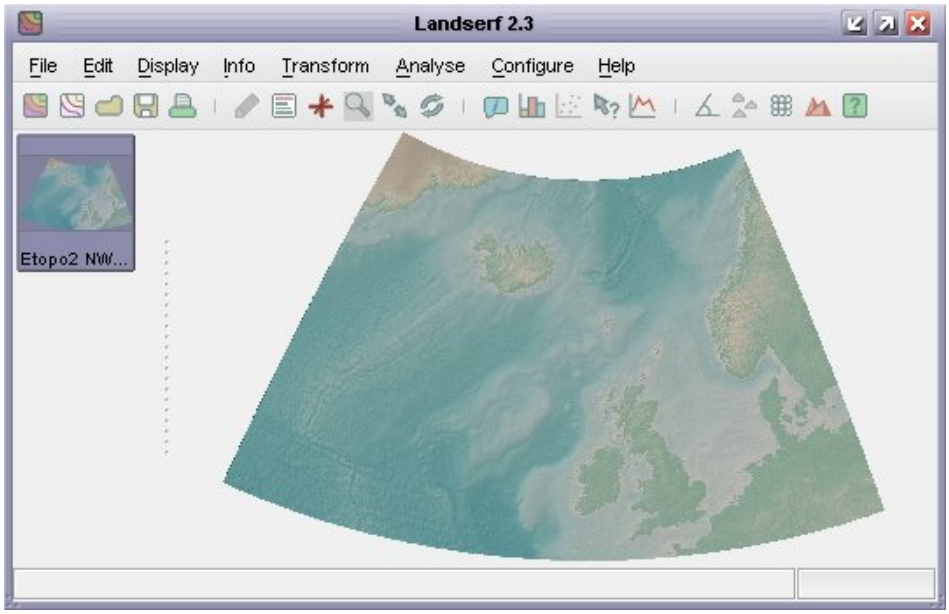


Figure 3.8: Reprojected srtm30plus DEM and bathymetry.

```
1 # Combines SRTM tiles into a single raster.
2 version(1.0);
3
4 baseDir = "/users/jowood/data/srtm/";
5 raster = open(baseDir & "N51E001.hgt");
6
7 raster = combine(raster_,open(baseDir & "N51E000.hgt"));
```

```
raster = combine(raster_,open(baseDir & "N51W001.hgt"));
raster = combine(raster_,open(baseDir & "N51W002.hgt"));
raster = combine(raster_,open(baseDir & "N52E001.hgt"));
raster = combine(raster_,open(baseDir & "N52E000.hgt"));
raster = combine(raster_,open(baseDir & "N52W001.hgt"));
raster = combine(raster_,open(baseDir & "N52W002.hgt"));

# Remove any voids in the raster.
raster = removevoids(raster_);

# Flood any values less than 0
raster = ifelse(raster<0,0,raster);

# Reproject data to Ordnance Survey National Grid.
raster = reproject(raster_,"OSGB","true",90,90);

# Create new colour table with blue for sea level
colouredit(raster,"land3");
colouredit(raster,"addrule","0 141 141 166 (D)");
colouredit(raster,"addrule","1 141 166 141");

# Add metadata to new raster
edit(raster,"title","SRTM 2 South East England");

# Save the combined and reprojected raster.
save(raster,baseDir & "SouthEastEnglandOSGB.srf");
```

3.3 Converting file formats using LandSerf

Format	**Extensions**	**Import**	**Export**	**LandScript ID**
ArcGIS raster text files.	`.grd .asc`	x	x	ArcGridText
ArcGIS Binary Image files.	`.bil`	x	x	ArcBIL
ArcGIS 'GridFloat' text files.	`.flt`	x	x	ArcGridFloat
ArcGIS 'ungenerate' vector text files.	`.gen .poi .lin .pol`	x	x	ArcGenerate
ArcGIS shapefile binary vector files.	`.shp`	x	x	Shapefile
Attribute table.	`.atr`	x	x	Attributes
BT binary raster files (Virtual Terrain Project).	`.bt`	x	x	BT
Colour table files.	`.ctb`	x	x	Colourtable
DTED Digital Terrain Elevation Data files.	`.dt0 .dt1 .dt2`	x		DTED
EDX raster text files.	`.asc`	x		EDX
Generic binary raster files.	`.bin`	x	x	BinRaster
Generic point vector text files.	`.txt`	x	x	Points
Generic raster text files.	`.txt`	x	x	TextRaster
Generic vector text files.	`.txt`	x	x	TextVector
Global Hierarchical, High-resolution Shoreline files.	`.ghs`	x		GHSS
GPS exchange (GPX) files.	`.gpx`	x	x	GPX
GTOPO30 global raster DEM files.	`.dem`	x		GTOPO30
Graphic image files (including GeoTIFF).	`.jpg .gif .png .tif .bmp`	x	x	Image
GRASS raster text files.	`.txt`	x	x	GrassRasterText
GRASS vector text files.	`.txt`	x	x	GrassVectorText
KML (Google Earth) raster files.	`.kmz`	x		KMZ
KML (Google Earth) vector files.	`.kml`	x		KML
LandScript files.	`.lsc`	x	x	
LandSerf native raster files.	`.srf`	x	x	RasterMap
LandSerf native vector files.	`.vec`	x	x	VectorMap
Mastermap (GB Ordnance Survey) vector text files.	`.gz .xml`	x		Mastermap
NTF (GB Ordnance Survey) raster text files.	`.ntf`	x		NTFRaster
NTF (GB Ordnance Survey) vector text files.	`.ntf`	x		NTFVector
Scalable Vector Graphics (SVG) file.	`.svg .svgz`		x	SVG
Shuttle Radar Topography Mission (SRTM) DEM files.	`.hgt .srtm`	x		SRTM
Terragen binary raster files.	`.raw`	x	x	TerragenBin
Terragen 'native' DEM files.	`.ter`	x	x	TerragenNative
3D route file.	`.r3d .xml`	x	x	Route
USGS Native DEM text files.	`.dem`	x		USGSNative
Vector drawing style files.	`.vst`	x	x	Vectorstyles
VistaPro binary raster files.	`.bin`	x	x	VistaProBin
VistaPro raster text files.	`.txt`	x	x	VistaProText
VRML raster 'world' files.	`.wrl .wrz .vrml`		x	VRML

LandSerf can import and export spatial data in a large range of commonly used data formats. The table above summarises the formats available, whether a file in a given format can be imported into LandSerf or exported from it, and the ID used to identify the format when handling the file type with *LandScript*. Details of each format are given below.

3.3.1 ArcGIS Raster Text File

Default file extensions: .grd or .asc

Example

```
ncols 321
nrows 468
xllcorner 387570.0
yllcorner 5289240.0
cellsize 30
nodata_value -32766
-32766 393 393 393 393 ..etc
-32766 395 396 396 394 ..etc
:
etc.
```

Description

ASCII file for storing raster data. Header information consists of at least the number of rows/columns,the geographical origin and the grid cell resolution. Any other header items are permitted but ignored.Note that this format only allows square pixels to be written (single cell size value). On output fromLandSerf, if the raster cells are not square, their dimension is given as the average of the two sides.

Raster data are stored in *nrows* rows from north to south, each row consisting of *ncols*values from west to east.

Raster values can be floating point or integer numbers and are separated by any number of white space characters.

When output from LandSerf, null values are given the numeric value -32766. If you wish to use another number to represent null on output, the raster should be transformed in LandSerf ('Transform raster values').

3.3.2 ArcGIS Binary Image (BIL) File

Default file extension: .bil (with .hdr and .blw files containing metadata)

Example

.hdr file:

```
BYTEORDER M
NROWS 468
NCOLS 321
NBITS 16
NBANDS 1
```

.blw file:

```
30.0
0.0000000
0.0000000
-30.0
387570.0
5289240.0
```

.bil file - binary equivalent of:

```
386 393 393 393 393 ..etc
387 395 396 396 394 ..etc
:
etc.
```

Description

Binary integer file with two separate header files containing image metadata and georeferencing. *fileName*.bil contains the 16 bit or 32 bit integers in row prime order from north to south.

fileName.hdr contains image metadata including the number of rows, columns and bits per cell. Byte order is 'big-endian' if BYTEORDER is set to M (Motorola), or 'little-endian' if set to I (Intel). LandSerf assumes only one interleaved layer.

fileName.blw contains the georeferencing information including grid resolution, and coordinates of the centre of the upper-left pixel of the image. The six values represent the affine transformation from image to georeferencedcoordinates. If the order of the 6 values are represented by A-F, the transformation is

x' = Ax + By + E
y' = Cx + Dy + F

Therefore, values (A,D) give (xRes,-yRes), (E,F) give (xOrigin,yOrigin) and (B,C) define the rotation(usually 0). Note that D is negative since the image origin is at the top, but georeferenced origin will be at the bottom.

3.3.3 ArcGIS 'GridFloat' text file.

Default file extension: .flt (with .hdr file containing metadata)

Example

.hdr file:

```
ncols 321
nrows 468
xllcorner 387570.0
yllcorner 5289240.0
cellsize 30.0
nodata_value -32767
byteorder LSBFIRST
```

.flt file - binary equivalent of:

```
386.1 393.0 393.4 392.8 392.6 ..etc
387.0 394.5 396.4 396.3 393.9 ..etc
:
etc.
```

Description

Binary floating point file with separate header file containing raster metadata and georeferencing. *fileName*.flt contains the 32 floating point values in row prime order from north to south.

fileName.hdr contains image metadata including the number of rows and columns, cell size and location of lower-left corner. Note that only one cel lsize value is permitted, so this format can only store square pixel values.On output from LandSerf, if the raster cells are not square, their dimension is given as the average of the two sides.Byte order is 'big-endian' if byteorderis set to MSBFIRST, or 'little-endian' if set to LSBFIRST. Note also that this '.hdr file' is not in the same format as the .hdr files used by the BILformat.

3.3.4 ArcGIS 'ungenerate' Vector Text File

Default file extensions: .poi, .lin, .pol or .gen (geometry) and .atr (attributes)

Example

```
1
320550.0 425320.0
320633.0 425320.0
320720.0 425300.0
320701.0 425295.0
END
2
360297.0 425310.0
361050.0 425305.0
361078.0 425300.0
END
3 361455.0 424987.0
4 361632.0 425001.0
5 361760.0 424977.0
:
etc.
END
```

Description

ASCII file for storing vector data. Points are stored in ID, x, y order on a single line.Separators can be spaces, tabs or commas. Vector objects consisting of lines or areas comprise an ID on its own line followed by space, comma or tab separated x and y coordinates, one pair per line. The end of each object's geometry is marked by the 'END' keyword. A final 'END' terminates the file. The file extension is usually used to distinguish lines from areas (.lin or .pol). If extension is not one of these then only repeated first and last coordinates can be used to infer polygon-line distinction.Attributes are stored in a separate file, each line comprising the object ID, a comma, and the object's attributes.

IDs should be integers, geometry should be floating point numbers. Attributes can be numbers or text.

3.3.5 ArcGIS Shapefile Binary Vector File

Default file extensions: .shp (with .shx, and .dbf files containing metadata and attributes)

Example

3 binary files containing geometry, attributes and indexing information.

Description

Shapefiles contain geometric vector object information without topological structure (for example, the common boundary of shared polygons will be recorded twice, once for each polygon). A given shapefile can store only one type of object, either points, lines or areas. It is therefore common to import/export 3 sets of shapefiles representing a vector map containing points, lines and areas.The geometry is stored in a *fileName*.shp file with each item referenced by a unique ID. IDs can be related to multiple attributes in a dBase III format *fileName*.dbf file. The *fileName*.shx file contains indexing information and is not used by LandSerf when importing data, but is created on output.

For detailed information about shapefile formats, see www.esri.com/library/whitepapers/pdfs/shapefile.pdf

3.3.6 Attribute Table

Default file extension: .atr

Example

```
1 35.4 "Sheffield"
2 17.6 "Nottingham"
3 11.5 "Birmingham"
4 5.0 "Liverpool"
:
etc.
```

Description

ASCII file representing an attribute table. Columns are space, tab or comma separated. The first column is the primary ID, the remaining columns are the attributes associated with that ID. The primary ID must be integer or a floating point number, the remaining attributes can be numbers or text. Text attributes containing spaces,tabs or commas must be enclosed in double quotes. There must be an equal number of columns in every row.

3.3.7 BT Binary Raster File (Virtual Terrain Project)

Default file extension: .bt

Byte Offset	Length	Contents	Description
0	10	"binterr1.3"	A marker which indicates that this is a BT 1.3 file
10	4 (int)	Columns	Width (east-west) dimension of the height grid.
14	4 (int)	Rows	Height (north-south) dimension of the height grid.
18	2 (short)	Data size	Bytes per elevation grid point, either 2 or 4.
20	2 (short)	Floating-point flag	If 1, the data consists of floating point values (float), otherwise they are integers.
22	2 (short)	Projection	0: Geographic; 1: metres or Universal Transverse Mercator (UTM); 2: feet (international foot = .3048 meters); 3: feet (U.S. survey foot = 1200/3937 meters)
24	2 (short)	UTM zone	Indicates the UTM zone (1-60) if the file is in UTM. The usual convention is negative zone numbers for the southern hemisphere.
26	2 (short)	Datum	Indicates the Datum (ignored).
28	8 (double)	Left extent	The extents are specified in the coordinate space specified by the UTM flag/zone fields. If UTM is false, they are ordinary geographic (latitude-longitude) values.
36	8 (double)	Right extent	
44	8 (double)	Bottom extent	
52	8 (double)	Top extent	
60	2 (short)	External projection	0: Projection is fully described by this header; 1: Projection is specified in a external **.prj** file
62	4 (float)	Vertical scale	Scaling factor for vertical units. If set to 0 (for backward compatibility), assumed to represent a scaling of 1
66-255	190	unused	Bytes of value 0 are used to pad the rest of the header.

Followed by binary equivalent of the following...

```
390.0 392.0 393.0 393.0 393.0 ..etc
391.0 391.0 392.0 392.0 393.0 ..etc
:  etc.
```

Raster data are stored in *nrows* rows from south to north, each row consisting of *ncols* values from west to east. Raster values should be 'little-endian' values either 32-bit float or 16-bit integer according to the header value.

3.3.8 Colour Table

Default file extension: .ctb

Example

```
<?xml version="1.0" ?>
<!--Colour table rules-->
<!--Generated by LandSerf 2.3-->
<colourTable>
 <rule value="0" type="discrete">255,255,255,255</rule>
 <rule value="1">98,123,92,255</rule>
```

```
 <rule value="25">130,152,117,255</rule>
 <rule value="50">196,197,160,255</rule>
 <rule value="75">244,232,195,255</rule>
 <rule value="100">249,242,230,255</rule>
</colourTable>
```

Description

XML file representing a set of colour table rules. Each rule should be associated with a numeric value and should define a colour using *R*,*G*,*B*,*opacity* values each scaled between 0 and 255. Colours for raster or vector maps are then interpolated between these rules. If a colour rule is not to be interpolated, but only associated with a particular numeric value, it should have its typeset to `discrete`.

If a spatial object's numeric value defines its colour directly (as a 32 bit ARGB integer), its associated colour table should be set to 'raw' as follows:

```
<?xml version="1.0" ?>
<colourTable raw="true" />
```

3.3.9 Digital Terrain Elevation Data (DTED) Files

Default file extensions: .dt0, .dt1, or .dt2

Example

Binary integer elevations with origin defined as lat/long coordinates.

Description

Binary format file storing elevation values and associated metadata, originally created by theUS Defence Mapping Agency (DMA) but now used by several national mapping agencies as well as the format for processed SRTM data supplied on DVD. Available at a number of resolutions - level 0 at 30 arc seconds, level 1 at 3 arc seconds and level 2 at 1 arc second. For details on the file format, see www.nga.mil/ast/fm/acq/89020B.pdf.

3.3.10 EDX Raster Text File

Default file extension: .asc

Example

```
Copyright 1994.  EDX Engineering, Inc.  All Rights Reserved.  5 280000.0 80000.0 280000.0
100000.0 300000.0 100000.0 300000.0 80000.0 50.0 50.0 401 401 .0 -2.00000 49.00000 6377563.4
.00334091 .99960000 -100000.0 400000.0 .0 323.0
392 393 393 393 393 ..etc
393 394 393 395 394 ..etc
:
etc.
```

Description

ASCII file for storing elevation data. Header information consists of a single first line with whitespace characters separating the following information `copyright`, `ktype`, `xc(1)`, `yc(1)`, `xc(2)`, `yc(2)`, `xc(3)`, `yc(3)`, `xc(4)`, `yc(4)`, `xinc_ns`, `xinc_ew`, `ny`, `nx`, `cell_offset`, `xlno`, `xlto`, `rad`, `f`, `xko`, `false_north`, `false_east`, `min_elev`, `max_elev`. LandSerf uses the copyright field, the south-

west [xc(1), yc(1)] and north-east [xc(3), yc(3)] corners and the resolution fields [xinc_ns, xinc_ew] only, but all other values should be included in the same order.

Raster data are stored in *ncols* rows from west to east, each row consisting of *nrows* values from south to north.

Raster values should be integer values only and are separated by any number of whitespace characters.

3.3.11 Generic Binary Raster File

Default file extension: .bin

Example

Binary equivalent of the following...

```
390 392 393 393 393 ..etc
391 391 392 392 393 ..etc
:
etc.
```

Description

Binary file for storing raster data. No header information is used, by default, when importing generic binary rasters, the grid resolution is assumed to be 50x the vertical units.

Raster data are stored in *nrows* rows from north to south, each row consisting of *ncols* values from west to east.

On saving, raster values are stored as integers and are stored as Intel ordered words (low byte first). To store non-integer values, rasters should first be transformed (e.g. by multiplying by 100) into suitable positive integers. Any values outside the range of the word (i.e., 0-255 for 8 bits, 0-65535 for 16 bits etc.), will be truncated around the minimum and maximum values. On import, files can be 8, 16 or 32 bit words and use either Intel or Motorola byte ordering.

3.3.12 Generic Point Vector Text File

Default file extension: .txt

Example

```
340530.0 448650.0 390.0 "Church"
340580.0 448650.0 392.0 "Car park"
350550.0 441250.0 493.0 "Unclassified"
:
etc.
```

Description

Text file for storing point data. No header information is required although comment lines starting with # are ignored and permitted at any point in the file.

Point data are stored in *xy* [*att1*] [*att2*] order, one point per line and can be space, tab or comma separated. All values can be in either integer or floating point formats. Attributes are optional. If only one attribute per point is provided and it is numeric (e.g. height value), it is stored as the primary attribute. If more than 1attribute is provided, or the first attribute is non-numeric, they are stored in a separate attribute table with an automatically computed point ID. Non-numeric attributes that consist of more than one word should be enclosed in double quotation marks.

3.3.13 Generic Raster Text File

Default file extension: .txt

Example

```
390 392 393 393 393 ..etc
391 391 392 392 393 ..etc
:
etc.
```

Description

ASCII file for storing raster data. No header information is used, by default the grid resolution is assumed to be 50x the vertical units.

Raster data are stored in *nrows* rows from north to south, each row consisting of *ncols* values from west to east.

Raster values can be floating point or integer numbers and are separated by any number of whitespace characters (including new lines) or a comma. Lines starting with # are assumed to be comments and are ignored.

When output from LandSerf, null values are given the numeric value -32766. If you wish to use another number to represent null on output, the raster should be transformed in LandSerf ('Transform raster values').

3.3.14 Generic Vector Text File

Default file extension: .txt

Example

```
P340000.0 483520.0 14.165
P345000.0 47905013.015
L27
366000.0 489530.0 14.5
375210.0 488100.0 10
A39
366000.0 489550.0 14.5
368100.0 488780.0 10.0
368200.0 488180.0 12.0
:
etc.
```

Description

Text file for storing point, line, flow and area vector data. No header information is necessary although lines starting with # are assumed to be comments and are ignored. x,y and z values all assumed to be in the same units.

Feature type is identified by a P, L or A prefix representing point, line and area features respectively. Point data are stored in *xy* [*z*] *Attrib* order, one point per line. z values are optional. Line and area data have an initial line containing the number of coordinates and an optional feature attribute. Coordinates follow in *xy* [*z*] order where the z value is optional.

All coordinates and attributes can be in either integer or floating point formats. If multiple attributes are

required, they should be stored in a separate attribute table.

3.3.15 Global Self-consistent, Hierarchical, High-resolution Shoreline Files

Default file extension: .ghs

Example Binary floating point lat/long coordinates.

Description

Binary format file holding global shorelines, rivers and lakes as closed areas. Available at a number of generalisation levels from www.ngdc.noaa.gov/mgg/shorelines/gshhs.html. This site includes more detailed information on the file format.

3.3.16 GPS Exchange (GPX) Files

Default file extension: .gpx Example

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<gpx xmlns="http://www.topografix.com/GPX/1/1" creator="MapSource 6.5" version="1.1"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.topografix.com/GPX/1/1
      http://www.topografix.com/GPX/1/1/gpx.xsd">

  <metadata>
    <link href="http://www.garmin.com">
      <text>Garmin International</text>
    </link>
    <time>2005-07-24T22:20:35Z</time>
    <bounds maxlat="52.292004" maxlon="1.632300" minlat="51.541886" minlon="-0.058193"/>
  </metadata>

  <wpt lat="51.542000" lon="-0.058100">
    <ele>15.000000</ele>
    <name>01Strt</name>
    <cmt>Pub on the Park, London Fields</cmt>
    <desc>Pub on the Park, London Fields</desc>
    <sym>Waypoint</sym>
  </wpt>
</gpx>
```

Description XML file for storing GPS waypoints, tracks and routes. A *de facto* standard for sharing GPS data between applications. Can include a range of metadata in addition to geometric data. For full details of the format see www.topografix.com/gpx.asp. Note that all georeferencing uses lat/long coordinates and WGS84.

Note: To have full control over GPX attributes when saving from LandSerf, the vector map should have an attribute table with at least the headings `ID`, `GPSObject` and optionally `Name`, `Description`, `Comment` and `Elevation`. Valid attributes for the `GPSObject` column are `Waypoint`,`Routepoint` for point objects and `Track` for line objects. If an appropriate attribute table is not found when saving as a GPS file, LandSerf defaults to saving all point objects as waypoints and all line objects as tracks.

3.3.17 GTOPO30 DEM Files

Default file extension: .dem (with .hdr, .dmw, .prj files containing metadata)

Example

.hdr file:

```
BYTEORDER M
LAYOUT BIL
NROWS 6000
NCOLS 4800
NBANDS 1
NBITS 16
BANDROWBYTES 9600
TOTALROWBYTES 9600
BANDGAPBYTES 0
NODATA -9999
ULXMAP -99.99583333333334
ULYMAP 39.99583333333333
XDIM 0.00833333333333
YDIM 0.00833333333333
```

.dmw file:

```
30.0
0.00833333333333
0.00000000000000
0.00000000000000
-0.00833333333333
-99.99583333333334
39.99583333333333
```

.prj file:

```
Projection GEOGRAPHIC
Datum WGS84
Zunits METERS
Units DD
Spheroid WGS84
Xshift 0.0000000000
Yshift 0.0000000000
```

.dem file - binary equivalent of:

```
386 393 393 393 393 ..etc
387 395 396 396 394 ..etc
:
etc.
```

Description

Global DEM files distributed as a min binary integer .DEM file with a collection of metadata files. Those used by LandSerf include the .hdr file that contains file format metadata, a .dmw that contains spatial transformation metadata and a .prj file that contains global map projection information. All spatial metadata are recorded as decimal degrees. Note that this collection is a specific case of the more general Binary Image file distribution.

fileName.dem contains the raster values in row-wise order in the format indicated by the associated .hdr file.

fileName.hdr contains image metadata including the number of rows, columns and bits per cell. Byte order is 'big-endian' if BYTEORDER is set to M (Motorola), or 'little-endian' if set to I (Intel). LandSerf assumes only 1 interleaved layer.

fileName.dmw contains the georeferencing information including grid resolution, and coordinates of the centre of the upper-left pixel of the image. The six values represent the affine transformation from image to georeferenced coordinates. If the order of the 6 values are represented by A-F, the transformation is

x' = Ax + By + E
y' = Cx + Dy + F

Therefore, values (A,D) give (xRes,-yRes), (E,F) give (xOrigin,yOrigin) and (B,C) define the rotation(always 0 for GTOPO30). Note that D is negative since the image origin is at the top, but georeferenced origin will be at the bottom.

fileName.prj contains global map projection information such as the spheroid used and projection type. It can also contain projection parameters. All GTOPO30 data use WGS84 geographic (lat/long) coordinates with the addition that an antarctic DEM is provided using a polar stereographic projection.

For more details on the format and distribution of GTOPO30 files, see
http://edcdaac.usgs.gov/gtopo30/README.asp.

3.3.18 Graphics Image File

Default file extensions: .jpg, .gif, .png, .tif, or .bmp

Example

GIF, TIF, JPEG or PNG file.

Description

Binary image file. Export is in either GIF, JPEG or PNG format (plus .bmp format from 3d viewer). Import can read GIF, JPEG, PNG or TIF (including GeoTIFF) format. Currently LZW compressed TIF files and CMYK colour scales cannot be read. Since most graphics images formats do not generally store geographical limits, they require some geographical information to be defined.

Import allows image files to be stretched over primary raster. If an ArcGIS 'world' transformation file is present with same name and '.w' extension (i.e. .GFW, .JGW, .PGW or .TFW), the image will be georeferenced on import.

Exported images are given the dimensions of LandSerf's graphical display at the point at which the image is saved.

3.3.19 GRASS Raster Text File

Default file extension: .txt

Example

```
north:  517000
south:  506000
east:  350000
west:  333000
```

```
rows:  220
cols:  340
null:  -32766
type:  float
multiplier:  10

647 662 680 695 709 ..etc
644 660 680 694 707 ..etc
:
etc.
```

Description

ASCII file for storing raster data. Header information consists of at the outer bounds of the raster and the number of rows/columns (in any order). It can optionally include the value representing null cells, the type of cell (float or int), and the value by which cells are multiplied.

Raster data are stored in *nrows* rows from north to south, each row consisting of *ncols* values from west to east. Raster values can be floating point or integer numbers and are separated by any number of whitespace characters.

3.3.20 GRASS Vector Text File

Default file extension: .txt

Example

Geometry file...

```
ORGANIZATION: Jo Wood
DIGIT DATE: 2.5.03
DIGIT NAME:
MAP NAME: contours from lakes.dem
MAP DATE:
MAP SCALE: 25000
OTHER INFO:
ZONE: 0
WEST EDGE: 333000
EAST EDGE: 350000
SOUTH EDGE: 506000
NORTH EDGE: 517000
MAP THRESH: 0
VERTI:
L 3
506025 336625
506025 336625
506025 336625
:
etc.
```

Attribute file...

```
L 336625.0000 506025.0000 50
L 333165.2545 507005.2545 100
L 334774.6173 506041.1564 100
```

```
:
etc.
```

Category file...

```
# 2 categories
0.00 0.00 0.00 0.00
50:  low risk
100:  high risk
```

Description

Text files for storing point, line, and area vector data. Geometry, primary and an optional secondary attribute set are stored in separate files in subdirectories `dig_ascii`, `dig_att` and `dig_cats` respectively.

The geometry file contains header information in the first 14 lines. The bounds of the vector map are used to set the edge values in the header, the remaining items are either ignored by LandSerf or stored as 'notes' when importing.

Feature type is identified by a P, L or A prefix representing point, line and area features respectively. Each new feature contains the type and number of coordinates to follow. Note that coordinate data are stored in *yx* order, one pair per line in the geometry file, but *xy* order in the attribute file.

The primary attributes are stored in a separate file with the same name in the subdirectory `dig_att`. Attributes are linked to geometry by identifying a coordinate pair in common (the first pair) followed by the attribute (one per line).

If secondary attributes (i.e. the attribute selected in an attribute table that is not a primary ID)are present, they are stored in a separate file with the same name in the subdirectory `dig_cats`

All coordinates and primary attributes can be in either integer or floating point formats.Secondary attributes (category labels in GRASS) can be numeric or textual.

3.3.21 KML (Google Earth) Raster File

Default file extension: .kmz

Example

Compressed binary format containing 'ground overlay' raster(s).

Description

KMZ files can be imported directly into Google Earth. See http://code.google.com/apis/kml/documentation for more details. LandSerf outputs the current display as a georeferenced raster. If called via LandScript, output is the selected raster.

3.3.22 KML (Google Earth) Vector File

Default file extension: .kml

Example

```
<?xml version="1.0" encoding="UTF-8" ?>
<!--KML file created by LandSerf 2.3-->
<kml xmlns="http://earth.google.com/kml/2.1">
 <Document xmlns:xlink="http://www.w3/org/1999/xlink">
```

```
  <Folder>
   <name>Vector map title</name>
   <Placemark>
    <name>1.0</name>
    <Point>
     <coordinates>-1.681592,53.186565,0</coordinates>
    </Point>
   </Placemark>
   <Style id="ls3.0">
   <LineStyle>
    <width>3</width>
    <color>ff000000</color>
   </LineStyle>
  </Style>
 </Document>
</kml>
```

Description

KML files can be imported directly into Google Earth. See http://code.google.com/apis/kml/documentation for more details. LandSerf outputs the primary vector map when saving in this format.

3.3.23 LandScript File

Default file extension: .lsc

Example

```
# Script to contour a surface
baseDir = "c:\data\";
surf = open(baseDir & "fractal1.srf");
contours = contour(surf,0,5,1);
save(contours,baseDir & "contours.vec");
```

Description

Text file storing *LandScript* commands. See the LandScript user's guide for details.

3.3.24 LandSerf Native Raster File

Default file extension: .srf

Example

Compressed binary format.

Description

This is the default format in which LandSerf raster maps are stored. In addition to raster values, full metadata, colour tables and attributes are stored in the single zip compressed file.

3.3.25 LandSerf Native Vector File

Default file extension: .vec

Example

Compressed binary format.

Description

This is the default format in which LandSerf vector maps are stored. In addition to vector geometry, full metadata, colour tables and attributes are stored in the single zip compressed file.

3.3.26 MasterMap Vector Text File

Default file extensions: .gz or .xml

Example

```
<?xml version='1.0' encoding='UTF-8'?>
<osgb:FeatureCollection
xmlns:gml='http://www.opengis.net/gml'
xmlns:xlink='http://www.w3.org/1999/xlink'
xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance'
fid='GDS-2183-1'>
:
etc.
```

Description

GML format vector of GB Ordnance Survey's 'MasterMap' large scale spatial database. See www.ordnancesurvey.co.uk/oswebsite/products/osmastermap/ for more details.

3.3.27 NTF Raster Text File

Default file extension: .ntf

Example

```
07 Section header record (raster bounds)
50 Grid header record (DEM range)
51 Grid record (DEM values)
:
etc.
```

Description

ASCII file for storing geographical data. This is a complex transfer format used largely by the GB Ordnance Survey for distributing their pre-MasterMap digital data. It is also referred to as British Standard BS 7567 'Electronic Transfer of Geographic Information'. For details see e.g. http://www.bsi-global.com/ (select a catalogue search for BS 7567). Header information is stored in section and grid header records, the elevation values in a grid record.

3.3.28 NTF Vector Text File

Default file extension: .ntf

Example

02 Database header record (product type)
07 Section header record (vector bounds)
05 Feature class record (attribute labels)
15 Point record
23 Line record
21 Geometry record (coordinates)
:
etc.

Description

ASCII file for storing geographical data. This is a complex transfer format used largely by the GB Ordnance Survey for distributing their pre-MasterMap digital data. It is also referred to as British Standard BS 7567 'Electronic Transfer of Geographic Information'. For details see e.g. http://www.bsi-global.com/ (select a cataloguesearch for BS 7567) or gdal.velocet.ca/~warmerda/projects/ntf. LandSerf can read Profile and Panorama contour and form-line files, LandLine (large scale), Meridian (medium scale) and Strategi (small scale) vector data in this format.

3.3.29 Scalable Vector Graphics (SVG) File

Default file extensions: .svg or .svgz

Example

```
<?xml version="1.0" encoding="iso-8859-1"?>
<svg xmlns="http://www.w3.org/2000/svg" viewBox="0 0 1020.0 1020.0">
<!-- Format definitions -->
<defs>
 <style type="text/css"><![CDATA[
  polyline { stroke:black; stroke-linejoin:round; stroke-linecap:round;
  stroke-width:17.435898; stroke-opacity:1; fill:none; }
  path { stroke:#ffffff; stroke-linejoin:round; stroke-width:17.435898;
  stroke-opacity:0.19607843; fill-opacity:0.75; }
  circle { stroke:none; fill-opacity:1; }
  text { stroke:none; fill-opacity:1.0; fill:#000000; font-family:sans-serif}
 ]]></style>
</defs>

<!-- Object metadata -->
<title>Combined vector</title>
<desc>Union of Simple shapes and Simple shapes</desc>

<!-- Geometry -->
<g id="spatialObject" transform="translate(10.0,1010.0) scale(1,-1)">
 <path style="fill:#cb6b66; " d="M300.0,300.00006 L100.0,0 500.0,0 300.0,300.00006 Z "/>
 <path style="fill:#e488ed; " d="M150.0,1000.0 L0.0,700.0 300.0,700.0 150.0,1000.0 Z "/>
 <circle style="fill:#cb6b66;stroke-opacity:1.0; " cx="300.0" cy="100.0" r="0.2"/>
 <circle style="fill:#e488ed;stroke-opacity:1.0; " cx="150.0" cy="800.0" r="0.2"/>
</g>

<!-- Point labels -->
```

```
<text x="313.05127" y="919.1539" font-size="30" text-anchor="middle" >Thing one</text>
<text x="163.05128" y="219.15387" font-size="30" text-anchor="middle" >Thing two</text>
</svg>
```

Description

Scalable vector graphics viewable over the web with any SVG plugin. If file has extension .svgz, then it will be compressed using GZIP deflation. Most SVG plugins should be able to decompress .svgz files on the fly. The geometry is mirrored in the X axis since the default SVG origin is the top-left corner. Styles are defined separately allowing easy user customisation of vector appearance. Polygons should have repeated first and last coordinates to ensure closing boundaries.

3.3.30 Shuttle Radar Topography Mission (SRTM) DEM

Default file extensions: .hgt or .srtm

Example

```
Binary equivalent of the following...
218 218 219 219 219 ..etc
217 217 218 218 219 ..etc
:
etc.
```

Description

Binary file for storing global elevation data, sampled at 1 arc-second, 3 arc-second or 30 arc-second resolution. No header information is included, but the file name indicates the geographic origin of the DEM. See www2.jpl.nasa.gov/srtm/ for details. LandSerf can also read 'srtm30Plus' files that comprises combined SRTM, GTOPO30 and bathymetric data at 30 arc-second resolution. For more details seehttp://topex.ucsd.edu/WWW_html/srtm30_plus.html.

Raster data are stored in *nrows* rows from north to south, each row consisting of *ncols* values from west to east. Raster will consist of 1201x1201 cells (3 arc-second DEM), 3601x3601 cells(1 arc-second DEM),48000x6000 cells (30 arc-second srtm30plus), or 7200x3600 cells (30 arc-second antarctic srtm30plus).

3.3.31 Terragen Raster Binary File

Default file extension: .raw

Example

```
Binary equivalent of the following...
218 218 219 219 219 ..etc
217 217 218 218 219 ..etc
:
etc.
```

Description

Binary file for storing raster data. No header information is included in the file.

Raster data are stored in *nrows* rows from south to north, each row consisting of *ncols* values from west to east.

On export from LandSerf raster values will be a square of side $2^n + 1$, for example, 257x257, 513x513, 1025x1025 depending on the original raster size. Attribute values will be 8 bit integers. LandSerf will automatically scale any raster to these dimensions with a range of 0-255. On import a choice of 8, 16 or 32 bit Intel (low byte first) ordered integers can made.

3.3.32 Terragen Native Binary File

Default file extension: .ter

Example

```
Binary equivalent of the following...
218 218 219 219 219 ..etc
217 217 218 218 219 ..etc
:
etc.
```

Description

Binary file for storing terrain data. Unlike Terragen's 'raw' binary file format, this contains header information and can be used to export non-square terrain data without resampling of elevation values.

For details of the format, see www.planetside.co.uk/terragen/dev/tgterrain.html.

3.3.33 3D Route File

Default file extensions: .r3d .xml

Example

```
<?xml version="1.0" ?>
<!--3d route and viewing parameters generated by LandSerf 2.3-->
<fieldtrip>
<title>3d route generated by LandSerf 2.3</title>
<environment fogdensity="0.1" sky="true" skycolour="#ff1a801f" seacolour="#ffad69aa"
   fogcolour="#ffccb4b3" wallcolour="#ffdcdef5"></environment>
<surface resolution="4" vscale="1.9" smooth="true" grid="false"></surface>
<route>
<waypoint name="waypoint1">
<view heading="320.0" pitch="-33.0" fov="70.0">
<spatial type="point" x="50.0" y="-25.0" z="51.5"></spatial>
</view>
</waypoint>
<waypoint name="waypoint2">
<view heading="328.0" pitch="-23.0" fov="30.8">
<spatial type="point" x="89.2" y="6.8" z="17.4"></spatial>
</view>
</waypoint>
</route>
</fieldtrip>
```

Description

XML file for storing 3d viewing parameters. The optional <environment> tag stores parameters describing the surrounding space. The optional <surface> tag contains parameters describing the surface appearance. The <waypoint> tags describe a sequence of camera positions and views. These are used to construct 'flythroughs' in the 3d viewer, but can also be imported as 2d vectors in LandSerf.

3.3.34 USGS Native DEM Text File

Default file extension: .dem

Example

```
MOUNT RAINIER WEST, WA-24000 LAT::  46 45 0.0000 N LONG::  -121 45 0.0000 W SCALE::  24000
...etc...  392 393 393 393 393 ...etc
```

Description

ASCII file for storing elevation data. Header information contains spatial extent, resolution,geometry, projection and error information in 1024 byte blocks. Structure is relatively complex and is documented at rockyweb.cr.usgs.gov/nmpstds/demstds.html. See also www.cs.arizona.edu/topovista/sdts2dem/ for an error-corrected and updated version of the **sdts2dem** converter.

Raster data are stored in columns rows from west to east, each row consisting of values from south to north. Because of the projection or potentially large areas into planar coordinates, each S-N profile can be of a different length. Raster values are integers but can be scaled by a multiplier stored in the header file.They are stored in fixed-width 1024 byte records, although LandSerf will use whitespace delimiters to read in the raster body.

3.3.35 Vector Drawing Styles

Default file extension: .vst

Example

```
<?xml version="1.0" ?>
<!--Vector drawing styles-->
<vectorstyles>
 <point fixed="true" size="2" surround="false"></point>
 <line fixed="true" size="1" surround="false"></line>
 <polygon boundary="true" boundarycolour="#000000ff"
  opacity="0.25"></polygon>
 <label align="northeast" background="#ffffff80"
  foreground="#000000ff" size="10" visible="false">
 </label>
 <rendering style="quality"></rendering>
</vectorstyles>
```

Description

XML file that defines the drawing styles for vector maps. Point, line, polygon and label styles can be defined along with rendering style (`speed` or `quality`). Each attribute corresponds to a setting shown in LandSerf's 'Vector appearance' window. Colours are defined using 8 digit hex numbers in the form `#RRGGBBAA` (red, green, blue and opacity).

3.3.36 VistaPro Raster Binary File

Default file extension: .bin

Example

```
Binary equivalent of the following...
390 392 393 393 393 ..etc
391 391 392 392 393 ..etc
:
etc.
```

Description

Binary file for storing raster data. No header information is used, by default the grid resolution is assumed to be 50x the vertical units. Raster data are stored in *nrows* rows from south to north, each row consisting of *ncols* values from west to east. Raster values should be 16 bit Intel (low byte first) integers. If floating point values are required, scale up to largest possible integer value (65535) and rescale after file has been transferred.

3.3.37 VistaPro Raster Text File

Default file extension: .txt

Example

```
390 392 393 393 393 ..etc
391 391 392 392 393 ..etc
:
etc.
```

Description

ASCII file for storing raster data. No header information is used, by default the grid resolution is assumed to be 50x the vertical units. Raster data are stored in *nrows* rows from south to north, each row consisting of *ncols* values from west to east. Raster values should be integers and are separated by any number of whitespace characters (including new lines).

3.3.38 VRML Raster 'World'

Default file extensions: .wrl, .wrz or .vrml

Example

```
#VRML V2.0 utf8#
# Converted from LandSerf file: Soil
# SW corner was originally: 320000.0,435020.0
# Scaled so centre is at 0,0,0 with largest side 10 units long
Background {
   skyColor [0.0 0.2 0.7, 0.0 0.5 1.0, 1.0 1.0 1.0 ]
   skyAngle [ 1.309, 1.571 ]
   groundColor [0.1 0.10 0.0,0.4 0.25 0.2, 0.6 0.60 0.6, ]
   groundAngle [ 1.309, 1.571 ] }
Transform {
```

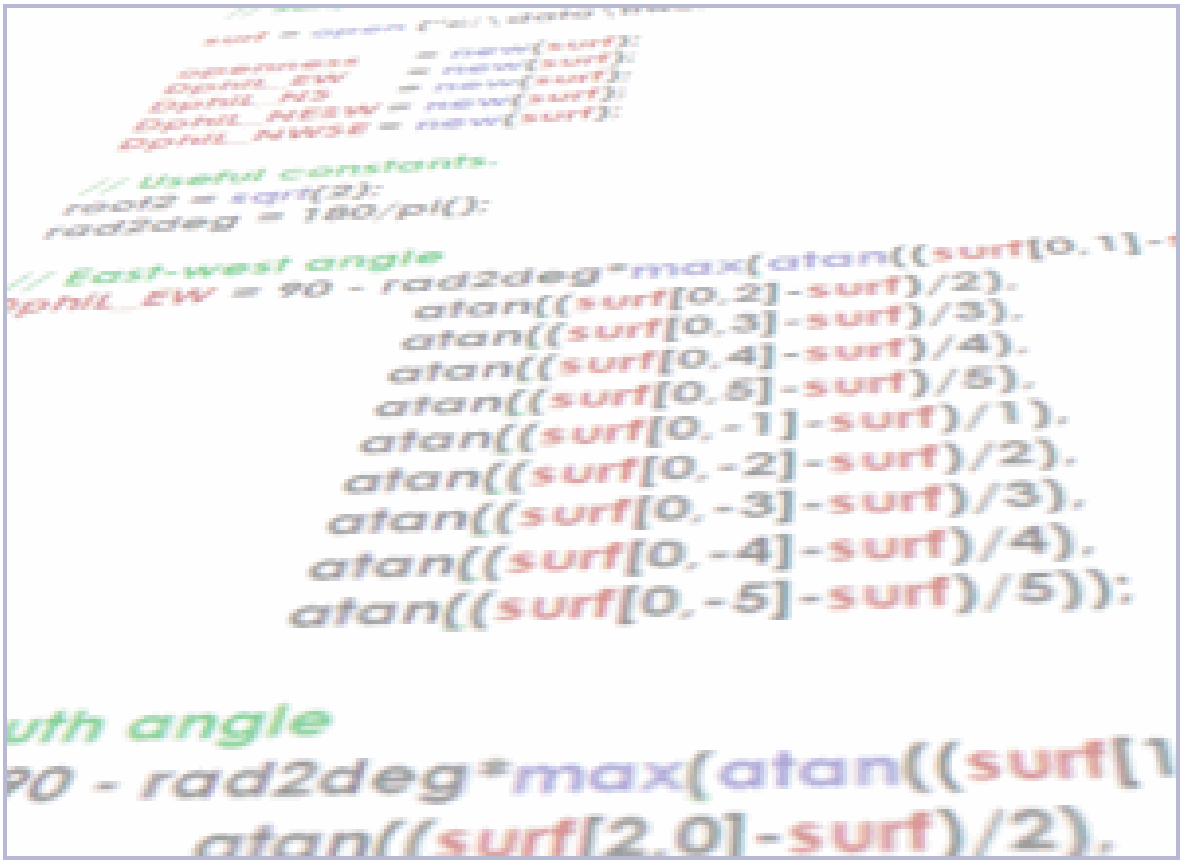
```
   translation -5, -4, -10
   scale 0.1,0.1,0.1
   children
   [
     Shape {
      appearance Appearance { material Material { } }
      geometry ElevationGrid {
         xDimension 100
         zDimension 100
         xSpacing 1.0
         zSpacing 1.0
         solid FALSE
         creaseAngle 3
         height [
        2.5961194, 2.5735216, etc.
        2.5961135, 2.5735227, etc.
        etc.
        ]
       colorPerVertex TRUE
       color Color {
       color [
         0.553 0.163 0.463, etc.
         0.553 0.121 0.495, etc.
        etc.
] } } } ] }
```

Description

ASCII file for defining Virtual Reality worlds. If the extension is .wrz, file is GZIP compressed.Raster maps saved in this format consist of raster elevations values in a VRML97 heightfield and vertex colouring as RGB triplets for each elevation. Vector TINs can be saved as VRML Indexed Face Sets which generally produce smaller and more quickly rendered VRML worlds.

All geometry is scaled to form a 20x20 unit grid centred at local coordinates 0,0,0. Null values are represented by the minimum elevation value. You may wish to consider resampling DEM to a reasonably coarse resolution (100x100 or so) or creating a TIN before output to keep the VRML world a manageable size. Includes a simple ground plane and sky to provide background context. Elevation values are floating point and are separated by any number of whitespace characters. Output has been tested with the Cortona VRML client compatible with most web browsers (IE, Opera, Firefox, Mozilla).

4 LandScript



4.1 LandScript - Controlling LandSerf by scripting

LandScript allows LandSerf to perform operations on spatial data by issuing commands within a script. This has the advantage of allowing repeated tasks to be carried out easily such as combining many rasters together. By issuing commands from within a script, you can document, reuse and share sequences of operations using LandSerf.

Most of the operations available via menus in LandSerf can be executed using LandScript. In addition,

numbers, strings of text and entire spatial objects can be stored and manipulated in variables and actions repeated in loops. This allows sophisticated map algebra operations to be built up without the need for knowledge of programming in Java. Since scripting involves no graphical interaction, operations can be faster and will use up less memory than their equivalent from within LandSerf.

4.1.1 Creating and Editing Scripts

Scripts can be edited and run by using the *LandScript Editor* (see Figure 4.1), either started directly or selected from the `File` menu within LandSerf. Alternatively, they can be created in any text editor and run directly from the command line. All saved scripts should have the extension `.lsc` to identify them as LandScript files. They can be loaded into the editor using the button or the `File`→`Open` menu.

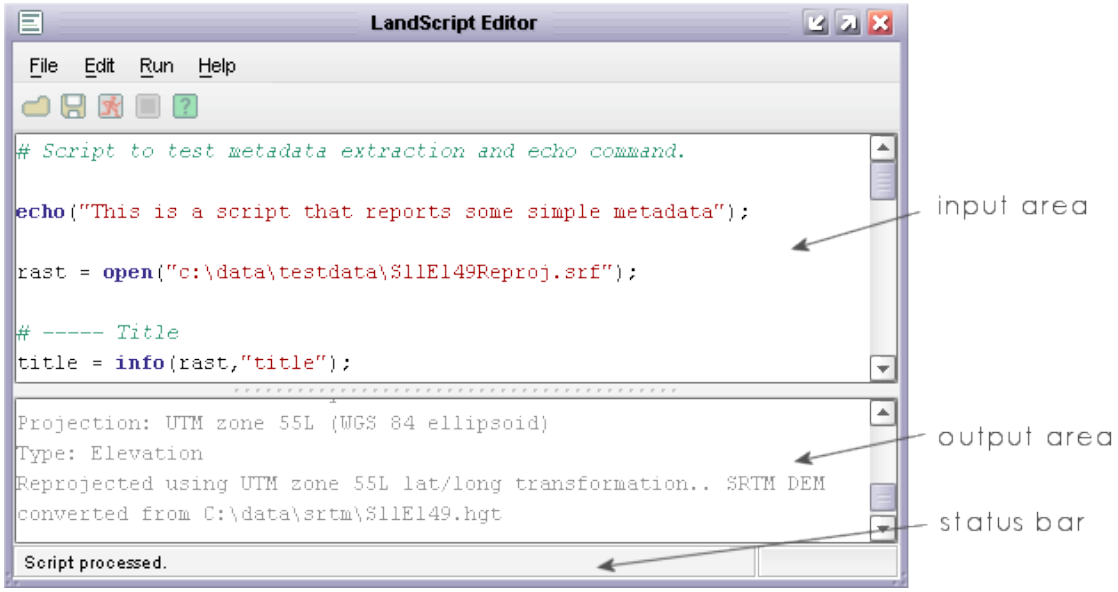


Figure 4.1: The LandScript editor

The advantages of using the LandScript Editor include coloured syntax highlighting, the ability to validate code before it is run, the separation of error reporting from standard output and the ability to stop scripts running mid-program.

LandScript can be laid out as you like, as long as each command line is separated by a semicolon. The LandScript editor will colour the elements of the script according to the following rules

- *`comments`* are coloured in green, should have a # symbol at the start of the line and are ignored by the LandScript interpreter.
- `commands`, `functions`, `keywords` are coloured in blue, and are special words reserved by LandSerf (see below for details).
- `"text strings"` are coloured in red and should be enclosed in quotes.
- `global variables` are coloured mid-grey.
- `local variables`, `numbers` and other characters are coloured in black.

When you run a script (by pressing the  button or selecting the `Run→Run` menu), any text output is displayed in the output area of the editor. Normal text is displayed in grey, and any errors in red. The status bar of the window will summarise the part of the script that may be causing problems should there be any errors in the interpreted script. A running script can be stopped at any point by pressing the  button (or the `Run→Stop` menu). When writing LandScript that involves much processing it can be useful to test the code before running it. Selecting the `Run→Validate` menu item will parse the code looking for syntax, command or typecasting errors without actually issuing the commands themselves.

All of the controls above as well as standard editing operations have keyboard shortcuts to speed up the write-test-debug-run cycle:

Control	**Shortcut***
Run	`Ctrl-R`
Stop	`Ctrl-P`
Validate	`Ctrl-D`
Copy selected text in editor	`Ctrl-C`
Cut selected text from editor	`Ctrl-X`
Paste text into editor	`Ctrl-V`
Undo last text edit	`Ctrl-Z`
Redo last undone text edit	`Ctrl-Y`

**On the Mac, use the `Command` ('Apple') key with the relevant shortcut letter.*

4.2 LandScript - Language Basics

4.2.1 Language Syntax

LandScript is made up of *commands* and *functions* that operate on *numbers*, *text* and *spatial objects*. The layout of script on the page is flexible and is designed to make reading script as easy as possible. Script can be spread over multiple lines with each instruction being terminated with a semicolon ';'.

All non-blank lines in a script file will be interpreted by LandSerf's *script engine* except those that start with the hash symbol '#'. This symbol can be used to identify *comment lines* that help to explain the meaning of the script.

So, for example, the following two scripts are treated in exactly the same way even though their layout and comments differ:

```
# Simple script to display a message on screen.
version(1.0);

echo("Hello world");
```

```
version(1.0);
echo
# This is a comment line so it is ignored.
  ("Hello world");
# Layout is flexible, but each command line
# must be terminated with a semicolon.
```

LandScript is a case sensitive language, and by convention, all variables, functions and commands are given lower case letters (an exception is the intercapping of variable names described below).

It is possible that commands, functions or even syntax may change in future versions of LandScript. It is therefore good practice to include the `version()` command at the start of a script. This will ensure that the LandScript interpreter will always process the script in the way it was originally intended.

4.2.2 Variables

Variables can be used to store items of information. There are 3 types of information that can be stored - *numbers*, *text* and *spatial objects*. LandScript is not a strictly typed language, which means that you do not need to declare what type of information is to be stored in a variable, and the type of information it does store can change while a script is running.

Information is placed in variables using the assignment operator '=', where the code on the left of the = should be name of the variable to store the information, and the code on the right should be an expression that defines what is to be stored within it.

```
# Script to show multiple variable types.
version(1.0);

message = "Hello World";
echo(message);

message = 1+2+3;
echo(message);

myRaster = newraster(0,0,10,10,100,100);
```

In the example above, the variable `message` firstly stores a text string (`"Hello World"`). Text strings can be indicated with pairs of quotation marks (`"`) surrounding the text. The next line in the example replaces the contents of the same variable with the result of evaluating the expression `1+2+3` (the number 6). Finally a separate variable `myRaster` is used to store an entire raster map created by the command `newraster()`.

Variable names must start with alphabetic characters but can contain numbers (e.g. `answer1`, `map2009` etc.). They cannot contain spaces or non-alphanumeric characters. By convention, variable names should be lowercase except when they consist of compound words, in which case intercapping (capitalising of the first letter of subsequent words) is encouraged. For example `myRaster`, `forestMap`.

Variables can also appear on the right hand side of the assignment operator, but only if they have been previously defined:

```
# The following two lines are valid
total = 0;
total = total+1;

# But the following would generate the error "'newTotal' not found".
newTotal = newTotal + 1;
```

The *scope* of a variable describes the parts of code in which its contents can be read or changed. Scope is limited to the *block* of code in which it is first assigned. A block (see section 3 below), is usually defined as any code enclosed between braces `{` and `}` including any nested sub-blocks within it.

```
# var1 is first defined in the global block so is visible to the whole script.
var1 = "Global variable";

{
  # This is a sub-block so var2 is only visible within this block
  # Note also that variables defined in the 'parent' block(s) of
  # a sub-block are also visible.
  var2 = "Local variable";
  echo ("Var1 is " & var1 & " Var2 is " & var2);

  # This will change the contents of the global variable originally defined outside this block.
  var1 = "A new value";
}

 # The following line would now generate an error since var2 is not visible.
 echo ("Var1 is " & var1 & " Var2 is " & var2);
```

Variables that are defined in the outermost block (as `var1` in the example above) are effectively global variables as their values can be read and changed in all parts of the script. The LandScript *ScriptEditor* identifies such global variables by colouring them mid-grey. It is generally good practice to make variables as local as possible within a script to avoid the possibility of accidentally changing variable contents and to limit naming conflicts.

4.2.3 Expressions

An *expression* is any collection of quantities (numbers, text or spatial objects) and operators that can be evaluated by the LandScript interpreter to produce a new quantity. The following are all examples of expressions:

```
numeric1 = 1 + (2/3) * 4;
numeric2 = sin(pi()/3);
numeric3 = numeric1 + 2*numeric2;
text1    = "First pair. " & "Second pair.";
spatial1 = newraster(5,5,15,15,100,100);
spatial1 = spatial1 + 5;
```

Line 1 would be evaluated as 3.6666666666666665 (note rounding error in last digit).
Line 2 would be evaluated as 0.8660254037844386
Line 3 would be evaluated as 5.398717474235544
Line 4 would be evaluated as the text `First pair.  Second pair.`
Line 5 would be evaluated as a raster of 100x100 cells each containing the value 0
Line 6 would be evaluated as a raster of 100x100 cells each containing the value 5.

4.2.4 Loops and Program Control

When LandScript code is interpreted, it is normally done line-by-line starting from the top of the script. However, it is possible to modify the flow of program control using *loops*, *conditional blocks* and the `return` command.

Loops

Loops are created using the command `while` followed by a *condition* in brackets and terminated by a semicolon. The *condition* can be any expression that can be evaluated to be true/non-zero or false/zero. The code to be repeated in the loop is then enclosed in braces following the `while` line. Because the looped code is defined in its own block, any variables defined for the first time inside the loop, are only visible within it.

```
version(1.0);

tableRow = 1;

while (tableRow <= 10);
{
  echo(tableRow & " x 7 = " & tableRow*7);
  tableRow = tableRow + 1;
}
```

The script above left generates times-table output shown below:

```
1 x 7 = 7
2 x 7 = 14
3 x 7 = 21
4 x 7 = 28
5 x 7 = 35
6 x 7 = 42
7 x 7 = 49
8 x 7 = 56
9 x 7 = 63
10 x 7 = 70
```

You must take care to ensure that it is possible for the loop to terminate in a finite amount of time. In the example above this is ensured by adding 1 to the value of `tableRow` each time control loops round the code in braces.

Note that if you are used to programming in languages such as C++ or Java, the `while` line must always be terminated with a semicolon and the looped code itself must always be enclosed in braces even if it only contains a single line. Note also that there is no 'for' loop in LandScript. This can be represented instead with a `while` loop and a counter variable as shown in the times-table example above.

Conditional Blocks and the `return` statement

A conditional block is identified with the `if` keyword followed by a *condition* in brackets and terminated by a semicolon. It works in much the same way as a while loop, except that the code within braces is only run once if the condition is evaluated to be true or non-zero. As with any block of script, the scope of any variables defined for the first time within a conditional block is limited to that block.

```
# Script to toss a coin.
version(1.0);

randomNum = rand();

if (randomNum < 0.5);
{
   echo("Heads");
}

if (randomNum >= 0.5);
{
   echo("Tails");
}
```

In the simple example above, two conditional blocks are used to simulate the outcome of a randomly tossed coin. Note that there is no 'else' construction in LandScript, so if mutually exclusive actions have to be modelled (as in the coin example above), you must explicitly state the mutually exclusive conditions.

It is possible exit a script early using the `return` keyword. When not used inside a script function (see section 5 below), calling `return` will quit the script, so it is most likely to be used inside a conditional block. For example:

```
# Creates a raster containing the z scores of another raster.
version(1.0);

baseDir = "/data/testdata/";
surf =open(baseDir&"nyquist.srf");
zScores = new(surf);

mean = info(surf,"mean");
stdev = info(surf,"stdev");

if (stdev == 0);
{
  echo ("Raster has no variation, so cannot calculate z-scores.");
  return;
}

zScores = (surf-mean)/stdev;
```

```
18
19 colouredit(zScores,"diverging1");
20 save(zScores, baseDir&"zScores.srf");
```

The conditional block from lines 11-15 checks to fory variation in the attribute values contained in the raster. If there is no variation (i.e. the standard deviation is zero), it reports an error message and exits the script at line 14. If there is variation, control passes to line 17 and beyond.

4.2.5 Script Functions

You can create your own functions (also known as methods or subroutines) in LandScript in order to group together the lines of code required to perform common tasks. User-defined functions should be indicated with the `function` keyword followed by the name of the function you wish to define and any parameters it requires. Following this line should be a block of code in braces that defines what the function actually does.

Functions can optionally return a value that can be used by the main body of the script in further processing. If the value is numeric, this can be combined with other built in functions to form more complex expressions.

```
1  # Script to show how a simple function works
2  version(1.0);
3
4  echo ("The sum of the numbers is "&sum(3,4));
5
6  # Function to add two numbers together.
7  # Params: num1 First number to add.
8  #         num2 Second number to add.
9  # Return: Sum of the two given numbers.
10 function sum(num1, num2)
11 {
12     return num1+num2;
13 }
```

Functions that do not return a value, but instead perform some specific task can also be useful, For example, the following displays the metadata associated with a raster map:

```
1  # Script to display the metadata associated with a raster.
2  version(1.0);
3
4  baseDir = "/data/testdata/";
5
6  newRaster = open(baseDir & "xslope.srf");
7  ans = displayInfo (newRaster);
8
9
10 # Function to display raster metadata.
11 # Params: rast Raster whose metadata is to be reported.
12 function displayInfo(rast)
13 {
14   echo("Metadata for "&info(rast,"title"));
15
16   echo("Bounds:");
17   echo("    "&info(rast,"N"));
18   echo(info(rast,"W")&" "&info(rast,"E"));
19   echo("    "&info(rast,"S"));
```

```
20   echo("Minimum: "&info(rast,"min")&" Maximum: "&info(rast,"max"));
21
22   echo("Resolution - x: "&info(rast,"xRes")&" y: "&info(rast,"yRes"));
23   echo("Projection: "&info(rast,"projection"));
24   echo("Type: " & info(rast,"type"));
25   echo(info(rast,"notes"));
26 }
```

Functions can also recursively call themselves to provide very powerful processing constructs. In the example below, a simple recursive function calculates the factorial of any given number (the factorial of 6 is 6*5*4*3*2=720).

```
1  # Script to calculate factorials using a recursive function.
2  version(1.0);
3
4  number = 6;
5  answer = factorial(number);
6  echo ("Factorial of " & number & " is " & answer);
7
8  # Function that recursively calculates the given number's factorial.
9  # Params: num Number to calculate
10 # Return: The given number's factorial.
11 function factorial(num)
12 {
13     if (num <= 1);
14     {
15         return 1;
16     }
17
18     if (num > 1);
19     {
20         ans = factorial(num-1);
21         return ans*num;
22     }
23 }
```

Note how line 20 inside the `factorial()` function recursively calls itself substituting `num-1` for the original value of `num`. Care should always be given when writing recursive functions that they always have a 'get-out clause' enabling control to be returned from the function.

4.2.6 LandScript Programming Styles

LandScript as a Macro Language

Probably the simplest approach to using LandScript is as a way of repeating a sequence of actions that would otherwise be achieved using menu selection in LandSerf. This has the advantage of be reproducible and sharable between users, although it does not take advantage of some of the more sophisticated functionality of the language.

When used as a macro language, most of the LandScript code will be calls to the LandScript commands providing the values that would otherwise be entered via the LandSerf Graphical User Interface. So for example, to load a file, change its projection and colour table and save a copy of the edited object, the following commands could be issued:

```
# Script to demonstrate simple macro processing.
version(1.0);

baseDir = "/data/testdata/";

# Open file.
rast = open(baseDir&"rast.srf");

# Add projection metadata and reproject.
edit(rast,"projection","LatLong");
edit(rast,"ellipsoid","WGS 84");
newRast = reproject(rast,"UTM","true",30,30);

# Provide new colour table
colouredit(newRast,"land3");

# Save the new file.
save(newRast,baseDir&"rastUTM.srf");
```

LandScript as a Map Algebra Calculator

Map algebra is an attempt to formalise the way we can combine and process spatial data. The concept comes from the work of Dana Tomlin (Tomlin 1983, 1990) who proposed a platform independent way of recording spatial processing. The concept is a simple one: represent spatial objects as algebraic terms and allow those terms to be manipulated using mathematical and logical operators.

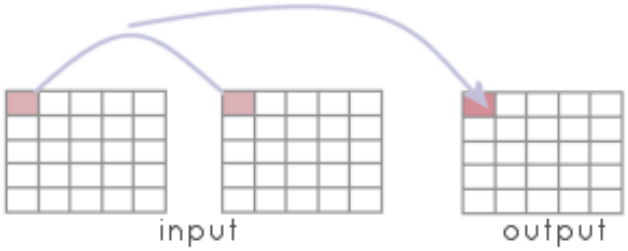
For example, suppose we had two elevation models of the same area, one generated via remote sensing that represents the upper surface of a region (which we will call `DSM`), the other generated via the interpolation of contours (which we will call `DGM`). The difference in height between these two models would give us the height of all the features, such as trees and hedges, that sit on top of DGM and have been recorded in the DSM. In map algebra terms, this can simply be represented as

All map algebra operations can be expressed in the form

newObject = f([Object1], [Object2], [Object3]...)

In other words new spatial objects are created as a function of existing objects. These objects may be raster or vector maps. Depending on what is used as input to a map algebra operation, two broad categories of function can be considered.

Local operations usually take input from at least two spatial objects. The output for any location is a function of the input objects at that same location.

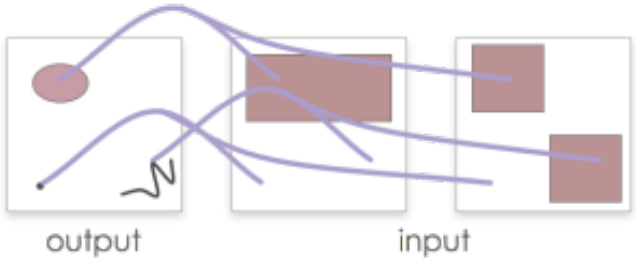


An example of a LandScript local operation applied to a pair of raster maps might be:

```
errorMap = sqrt((dem1-dem2)^2);
```

which creates a raster containing the root of the squared difference between two elevation models for each raster cell.

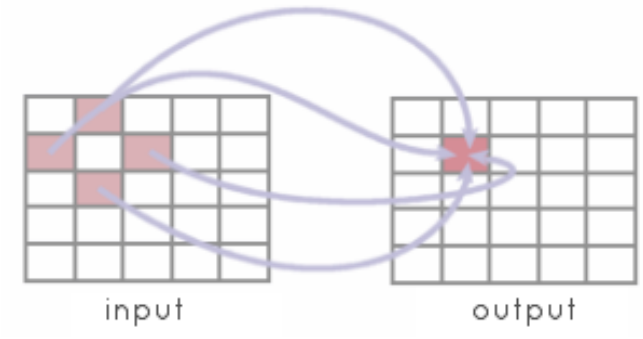
Applying local map algebra to vector maps works in a similar way except that the output map must first be created so that the vector objects that are to have new attributes may be defined. In other words, applying map algebra to vector objects has no effect on their *geometry*, but their *attributes* are a function of the input objects that share the same spatial location as the output objects. In LandScript, the centroid of each output object is used to define their locations.



Both raster and vector maps may be combined in a single local map algebra expression. So for example, the following expression effectively creates a rasterized version of combined vector maps summing their contents where the vector maps overlap.

```
raster = vect1 + vect2;
```

Focal operations usually take input from several locations in a single spatial object. The output for any location is a function of the input object at points surrounding the output location. Such functions are often referred to as neighbourhood operations since they process the neighbourhood of location in order to generate output.



LandScript can only apply focal operations to rasters and allows neighbouring raster cells to be identified using a focal modifier in square brackets containing row and column offsets. For example:

```
smoothedDEM = (dem[-1,-1] + dem[-1,0] + dem[1,0] +
               dem[-1,0]  + dem[0,0]  + dem[1,0] +
               dem[1,-1]  + dem[1,0]  + dem[1,1])/9;
```

creates a raster where each cell is the average of an input raster's immediate neighbourhood.

More sophisticated use of focal modifiers can be made by combining them with conditional operations. In the example below, the nested `ifselse` commands in lines 22-26 copy all the non-null raster cells from one raster map to another as well as those that immediately border non-null cells. The result is a that islands are 'grown' by one cell. By using a local map algebra operation (line 30) to find the difference between the original and expanded rasters, an additional 'coastline' map containing only the new border cells can be created.

```
1  # Creates some fractal islands, then grows the coastlines by 1 pixel.
2  version(1.0);
3
4  baseDir = "/data/testdata/";
5
6  # Create empty raster.
7  fracSurf = newraster(1000,1000,10,10,250,250,"Fractal");
8
9  # Fill raster with fractal surface.
10 fractal(fracSurf,"2.01");
11 islands = new(fracSurf);
12
13 # Use data range to get 'sea level';
14 maxi = info(fracSurf,"max");
15 mini = info(fracSurf,"min");
16 midPoint = mini + (maxi-mini)/2;
17
18 # Flood all values below 'sea level'
19 fracSurf = ifelse(fracSurf < midPoint, null(), fracSurf);
20
21 # Grow coastline
22 islands = ifelse(fracSurf != null(), fracSurf,
23             ifelse(fracSurf[0,-1] != null(), fracSurf[0,-1],
24                ifelse(fracSurf[0,1] != null(), fracSurf[0,1],
25                   ifelse(fracSurf[-1,0] != null(), fracSurf[-1,0],
26                      ifelse(fracSurf[1,0] != null(), fracSurf[1,0],null())))));
27
28 # Find coastline by taking difference between original and grown islands.
29 coastline = new(fracSurf);
30 coastline = islands-fracSurf;
31
32 # Save original and expanded islands.
33 save(islands,baseDir & "islands.srf");
34 save(coastline,baseDir & "coastline.srf");
```

When creating a local map algebra expression, it is sometimes useful to be able to identify a complete spatial object rather than a particular cell value. This is especially so when including LandScript commands in an expression. For example, suppose we wished to add mean value of a raster to each cell in a raster. If we tried to do this with the following expression:

```
1 dem = open("myRaster.srf");
2 dem = dem + info(dem,"mean");
```

LandScript would generate the error: `Expecting spatial object as parameter 1 in info but number provided`. This is because second occurrence of `dem` on the right hand side of the expression in line 2 is interpreted as being part of a local operator that would take each cell in turn and process its numeric value. What we actually intended was for the `info()` command to treat that particular instance of `dem` as a single raster rather than process each of its cells. To force this to happen, simply place an underscore character (_) after the name of the raster:

```
1 dem = open("myRaster.srf");
2 dem = dem + info(\textbf{dem_},"mean");
```

In this particular example, it would have been more efficient to store the mean of the `dem` values once in its own variable rather than calculate the same mean for every cell in the raster:

```
1 dem = open("myRaster.srf");
2 mean = info(dem,"mean");
3 dem = dem + mean;
```

Keeping code efficient in map algebra expressions is important if you wish your scripts to execute in a reasonable length of time. But there are some circumstances when the use of the underscore character to stop local map algebra operations being performed is necessary. For example, calling a LandScript command whose output is to replace one of the input parameters in that expression:

```
1 raster = reproject(raster_,"UTM","true",50,50);
```

Without the underscore, LandScript would incorrectly interpret this expression as a local operator as it contains a spatial object variable (`raster`) on the left and right hand side of the equals symbol.

Finally, the special variables `easting`, `northing`, `row` and `col` can be used in map algebra expressions to substitute the location coordinates of each cell in a spatial object into the expression itself. So in the example below, a mathematical function based on the (x,y) coordinates of each cell (line 7) is used to create a new surface (see Figure 4.2):

```
1  # Script to create a Nyquist sampling function.
2  version(1.0);
3
4  baseDir = "/data/testdata/";
5
6  nyquist = newraster(-100,-100,1,1,201,201);
7  nyquist = 10* (sin(easting/10) / (easting/10)) * (sin(northing/10) /(northing/10));
8
9  # Add metadata.
10 edit(nyquist,"title","Nyquist");
11 edit(nyquist,"notes",
12             "Nyquist sampling model based on the 2-d sinC function "&
13             "((sin PI x / PI x)(sin PI y / PI y) )");
14 edit(nyquist,"type","elevation");
15 colouredit(nyquist,"land4");
16
17 # Save the new raster map.
18 save(nyquist,baseDir&"nyquist.srf");
```

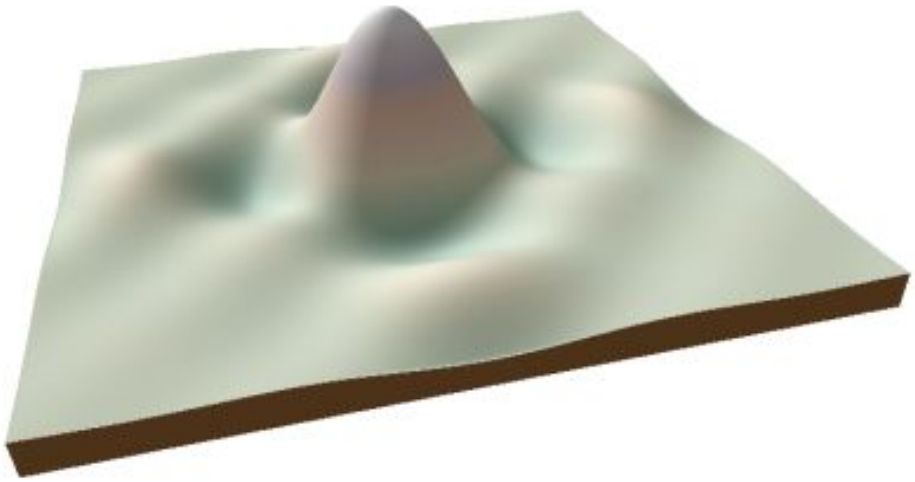


Figure 4.2: Nyquist sampling function produced by landscript.

LandScript as a Lower Level Programming Language

There may be some special cases where map algebra expressions do not provide sufficient control over operations to produce the desired output. In such circumstances, it is possible to use LandScript looping and conditional constructions with LandScript commands to process spatial objects in a low level fashion.

The disadvantage of this approach is that it can be considerably slower than using map algebra, and certainly much slower than using the LandSerf API to code a routine in Java or some other low level language.

Most lower level processing of a raster object will need to use two nested loops to select each raster cell in turn. The example below shows how this can be achieved in LandScript. Note that this particular example simply does what would more readily be achieved with the single map algebra expression `newRast = rast1 + rast2;`

```
# Script to add the contents of two rasters.
version(1.0);

baseDir = "/Users/jwo/data/testData/";

# Create the two input rasters and one output raster.
rast1 = newraster(0,0,1,1,100,100,"Fractal 1");
fractal(rast1,"2.01");

rast2 = newraster(0,0,1,1,100,100,"Fractal 2");
fractal(rast2,"2.01");

newRast = new (rast1);
edit(newRast,"title","Sum of rast1 and rast2");

# Store dimensions of the rasters
numRows = info(newRast,"numRows");
numCols = info(newRast,"numCols");

# Process each raster cell individually.
r=0;
```

```
22 while (r < numRows);
23 {
24   c=0;
25   while (c < numCols);
26   {
27     z1 = rvalueat(rast1,r,c);
28     z2 = rvalueat(rast2,r,c);
29     rvalueat(newRast,r,c,z1+z2);
30     c = c+1;
31   }
32   r = r +1;
33 }
34
35 # Save new raster.
36 colouredit(newRast,"grey1");
37 save(newRast,baseDir&"newRast.srf");
```

The key to being able to extract individual raster values is the LandScript command `rvalueat()` which is able to extract the raster value at a given row and column coordinate (lines 27 and 28 above) as well as set a new raster value for a given cell (line 29 above). By calling this command inside two nested loops that process each column in the rasters row by row, all the raster cells may be processed.

The code above is not particularly useful in that is simply replicates the process more easily achieved with map algebra. However, for more sophisticated processing, such as recursive traversal of a raster, this approach can be useful:

```
1  # Script to calculate flow magnitude of a DEM.
2  version(1.0);
3  basedir = "/data/testdata/";
4
5  # Initialise flow magnitude and drainage basin rasters.
6  dem = open(baseDir&"newRast.srf");
7  aspect = surfparam(dem,"aspect",3);
8  flowMag = new(dem);
9  basins = new(dem);
10 basinID = 0;
11 edit(flowMag,"title","Flow magnitude");
12 edit(flowMag,"type","other");
13 edit(basins,"title","Drainage basins");
14 edit(basins,"type","other");
15
16 numRows = info(dem,"numRows");
17 numCols = info(dem,"numCols");
18 r=1;
19
20 while (r < numRows);
21 {
22   echo("Processing row " & r);
23   c=1;
24   while (c < numCols);
25   {
26     basinID = r*numCols+c;
27     localFlow = calcFlowMag(r,c);
28     rvalueat(flowMag,r,c,localFlow);
29     c = c +1;
30   }
31   r = r +1;
32 }
33
34 # Provide logarithmic colour scale for flow magnitude.
35 colouredit(flowMag,"exp1");
```

```

# And a random colour table for drainage basins.
colouredit(basins,"random");

# Save flow magnitude and drainage basins.
save(flowMag,baseDir&"demFlowMag.srf");
save(basins,baseDir&"demBasins.srf");

endTime = datetime();
echo ("Completed in "&(endTime-startTime)/10 &"seconds.");

# Recursive flow magnitude function.
function calcFlowMag(r,c)
{
  # Check we haven't been here before
  visitedCell = rvalueat(basins,r,c);
  if (visitedCell == basinID);
  {
    # We have already visited this cell during this pass.
    return 0;
  }
  flow = 1;

  # Log this cell as belonging to the drainage basin.
  rvalueat(basins,r,c,basinID);

  # Stop if we have reached the edge.
  if ((r==0) or (c == 0) or (r >= numRows-1) or (c >= numCols-1));
  {
    return flow;
  }

  # Look for neighbours that might flow into this cell.
  aspVal = rvalueat(aspect,r-1,c);

  if ((aspVal >135) and (aspVal <=215));
  {
     fl = calcFlowMag(r-1,c);
     flow = flow +fl;
  }

  aspVal = rvalueat(aspect,r+1,c);
  if ((aspVal >305) or ((aspVal <=45) and (aspVal != null())));
  {
     fl = calcFlowMag(r+1,c);
     flow = flow +fl;
  }

  aspVal = rvalueat(aspect,r,c-1);
  if ((aspVal >45) and (aspVal <= 135));
  {
     fl = calcFlowMag(r,c-1);
     flow = flow +fl;
  }

  aspVal = rvalueat(aspect,r,c+1);
  if ((aspVal >215) and (aspVal <= 305));
  {
     fl = calcFlowMag(r,c+1);
     flow = flow +fl;
  }
  return flow;
}
```

4.3 LandScript - Tutorial

This tutorial will show you how to use LandScript to perform tasks ranging from opening and saving files to sophisticated spatial analysis using map algebra. It is highly recommended that you first familiarlise yourself with LandSerf before completing this tutorial, perhaps by completing the introductory LandSerf tutorial.

For the moment, each tutorial step corresponds approximately to a stage in the LandSerf tutorial. The scripts are commented and should be self-explanatory.

4.3.1 Step 1 - Opening, reprojecting and saving a raster map.

Explanation pending (see Step 1 of the Introductory LandSerf tutorial for details).

```
# Script to convert binary image file into a UTM LandSerf file.
# v1.2 Jo Wood 21st April, 2009.
# ---------------------------------------------------------------
version(1.0);

# Store the base directory in a variable to allow easy changing.
baseDir = "c:/Program Files/LandSerf/data/";

# Open the .bil file and store the raster as a variable.
dem = open(baseDir & "lincolnNED.bil");

# Add the projection metadata to the DEM.
edit(dem,"projection","LatLong");
edit(dem,"ellipsoid","wgs 84");

# Create a new Universal Transverse Mercator (UTM) reprojected DEM.
dem = reproject(dem_,"UTM","true",30,30);

# Save the edited raster as a LandSerf file.
save(dem, baseDir & "landscriptTutorial/step1.srf");
```

4.3.2 Step 2 - Editing raster metadata.

Explanation pending (see Step 2 of the Introductory LandSerf tutorial for details).

```
# Script to edit the metadata of a spatial object.
# v1.2 Jo Wood 21st April, 2009.
# ---------------------------------------------------------------
version(1.0);

# Store the base directory in a variable to allow easy changing.
baseDir = "c:/Program Files/LandSerf/data/landscriptTutorial/";

# Open the file from the previous step and store as a variable.
dem = open(baseDir & "step1.srf");

# Give the raster a new colour table.
colouredit(dem,"land3");

# Guess at the water level by finding modal elevation
maxFreq = info(dem,"modef");
```

```
if (maxFreq > 1000);
{
  # If there are more than 1000 cells of same height, colour them blue
  waterLevel = info(dem,"mode");
  colouredit(dem,"addRule",waterLevel & " 141 141 166 (D)");
}

# Update title and notes.
edit(dem,"title","Lincoln 30m DEM");
edit(dem,"notes","Elevation data from USGS National Elevation Dataset (NED) available from http://
    seamless.usgs.gov");

# Save the edited raster as a LandSerf file.
save(dem, baseDir & "step2.srf");
```

4.3.3 Step 3 - Processing a landcover raster.

Explanation pending (see Step 3 of the Introductory LandSerf tutorial for details).

```
# Script to process a landcover raster.
# v1.2 Jo Wood 21st April, 2009.
# ------------------------------------------------------------
version(1.0);

# Store the base directory in a variable to allow easy changing.
baseDir = "c:/Program Files/LandSerf/data/";

# Open the landcover file, reproject it to UTM.
landcover = open(baseDir & "lincolnLandcover.bil");
edit(landcover,"projection","LatLong");
edit(landcover,"ellipsoid","wgs 84");
landcover = reproject(landcover_,"UTM","false",30,30);

# Update the landcover metadata.
edit(landcover,"title","Lincoln landcover");
colouredit(landcover,"file",baseDir & "landcover.ctb");
edit(landcover,"attributes",baseDir & "landcover.atr");
edit(landcover,"attCol",3);
edit(landcover,"type","other");

# Find average height of all forested areas.
dem = open(baseDir & "step2.srf");
forestHeight = new(landcover);
forestHeight = ifelse((landcover >= 40) and (landcover <=49),dem,null());
echo("Average height of forest is " & round(info(forestHeight,"mean")) & "m.");

# Find average height of scrubland areas.
dem = open(baseDir & "step2.srf");
scrubHeight = new(landcover);
scrubHeight = ifelse(landcover == 51,dem,null());
echo("Average height of scrubland is " & round(info(scrubHeight,"mean")) & "m.");

# Save the edited landcover as a LandSerf file.
save(landcover, baseDir & "landscriptTutorial/step3.srf");
```

4.3.4 Step 4 - Simple morphometric analysis.

Explanation pending (see Step 4 of the Introductory LandSerf tutorial for details).

```
# Script to perform some simple morphometric analysis on a DEM.
# v1.2 Jo Wood 21st April, 2009.
# ------------------------------------------------------------
version(1.0);

# Store the base directory in a variable to allow easy changing.
baseDir = "c:/Program Files/LandSerf/data/landscriptTutorial/";

# Open the DEM and landcover raster from the previous steps.
dem = open(baseDir & "step2.srf");
landcover = open(baseDir & "step3.srf");

# Create a shaded relief image saved as a file.
edit(dem,"vExag",3);
draw(baseDir & "relief.jpg",dem,landcover,"null","null","relief");

# Output some relevant DEM metadata
showMetadata(dem,"elevation");

# Calculate and output slope of DEM
slope = surfparam(dem,"slope");
draw(baseDir & "slope.jpg",dem,slope,"null","null","relief");
showMetadata(slope,"slope");

# Create images of the other surface parameters.
draw(baseDir & "aspect.jpg",dem,surfparam(dem,"aspect"),"null","null","relief");
draw(baseDir & "profc.jpg", dem,surfparam(dem,"profc"),"null","null","relief");
draw(baseDir & "planc.jpg", dem,surfparam(dem,"planc"),"null","null","relief");

# Finally create a surface feature map and save as a LandSerf file.
features = surfparam(dem,"feature");
save(features, baseDir & "LincolnFeatures.srf");


# Function to display some spatial metadata.
# Params: raster Raster from which to extract metadata.
#         name Name of raster.
function showMetadata(raster,name)
{
  echo("\n" & name);
  echo("East-west extent: "& (info(raster,"E")-info(raster,"W"))/1000 & "km.");
  echo("North-south extent: "& (info(raster,"N")-info(raster,"S"))/1000 & "km.");
  echo(name&" range from "& info(raster,"min")&" to "&info(raster,"max")&".");
  echo("Mean "&name&": "& info(raster,"mean"));
  echo("Modal "&name&" value: "& info(raster,"mode"));
  echo("Standard deviation of "&name&": "& info(raster,"stdev"));
  echo("Skewness of "&name&": "& info(raster,"skew") & "\n");
}
```

4.3.5 Step 5 - Advanced morphometric analysis.

Explanation pending (see Step 5 of the Introductory LandSerf tutorial for details).

```
# Script to perform more advanced morphometric analysis on a DEM.
# v1.2 Jo Wood 21st April, 2009.
# ---------------------------------------------------------------
version(1.0);

# Store the base directory in a variable to allow easy changing.
baseDir = "c:/Program Files/LandSerf/data/landscriptTutorial/";

# Open the DEM from the earlier step.
dem = open(baseDir & "step2.srf");

# Calculate 11x11 cell smothed elevation and draw output.
winSize = 11*info(dem,"xRes");
echo("Smoothing elevation using "&winSize&"m x "&winSize&"m window");
draw(baseDir & "elev11.jpg", surfparam(dem,"elev",11),"null","null","null","relief");

# Calculate 65x65 cell smothed elevation and draw and save output.
winSize = 65*info(dem,"xRes")/1000;
echo("Smoothing elevation using "&winSize&"km x "&winSize&"km window");
elev65 = surfparam(dem,"elev",65);
draw(baseDir & "elev65.jpg",elev65,"null","null","null","relief");
save(elev65,baseDir&"elev65.srf");

# Create a 5x5 feature network and metric surface network
feat5 = surfparam(dem,"netfeature",5,1);
msn = surfnetwork(dem,feat5);
draw(baseDir & "featnetwork.jpg",dem,"null",msn,"null","relief");
save(msn,baseDir&"msn.vec");
```

4.4 LandScript - Functions and Operators

The following can be used to create expressions. The can be simple scalar expression or map algebra expressions that manipulate entire spatial objects. Functions differ from commands in that they do not directly replicate LandSerf operations that would otherwise be accessed via the LandSerf menus. Instead, functions can be used to build complex expressions to manipulate the contents of variables in a script.

4.4.1 Arithmetic Operators

These can be used in any expression that involves scalar quantities (simple numbers and variables) or spatial objects (raster and vector maps).

+, -, *, /	Addition, subtraction, multiplication and division operators.
^	Power operator (e.g. 5^2 is 25,7^(1/3) is the cube root of 7).
%	Modulus (remainder) operator (e.g. 15%6 is 3 since 15/6 is 2 with remainder 3).
()	Brackets to control the order of evaluation.

4.4.2 String Operators

Applied to quantities representing text.

"	Identifies a string (e.g. `title = "Elevation model";`)
&	String concatenation (e.g. `title = "Elevation" & " model";` results in a single string `Elevation model`).
`compare(`*string1*`,`*string2*`)`	Compares two strings alphabetically. If *string1* comes alphabetically before *string2*, this function returns -1; if it comes alphabetically afterwards, returns 1; otherwise returns 0 if the strings are identical.
`quote(`*arg*`)`	Creates a new string that wraps the given *arg* inside "quotes". The argument can be a number or a string, but the result is always a string. This can be useful when needing to create text with quotation marks.
`strlen(`*arg*`)`	Returns the number of characters in string *arg2*.
`substring(`*string*`,`*arg1*`,`*arg2*`)`	Creates a new string that is part or all of *string*. The substring starts from position *arg1* where 0 is the first letter of the string, and continues until position just less than *arg2*. If *arg2* is larger than the length of the string, the substring from *arg1* to the end is returned.

4.4.3 Mathematical Functions and Constants

Applied to scalar numeric quantities.

`abs(arg)`	Calculates the absolute value of *arg* (strips it of any minus sign if it exists).
`acos(arg)`, `asin(arg)`, `atan(arg)`, `atan2(arg1,arg2)`	Inverse trigonometrical functions. Returns values in radians. Values outside the range of +-1 supplied to these functions will return a value of 0.
`cos(arg)`, `sin(arg)`, `tan(arg)`	Trigonometrical functions. Expects *arg* to be an angle expressed in radians . Values of infinity (e.g. `tan(pi()/2)`) are returned as 0.
`e()`	Provides Euler's number *e* (2.7182818).
`gauss()`	Random value with Gaussian (normal) distribution with mean of 0 and standard deviation of 1.
`ln(arg)`	Natural logarithm (to base *e*) of *arg*. If the value given to the function is <=0, the function will return a zero.
`null()`	Provides the constant representing a null value.
`pi()`	Provides the constant *pi* (3.14159).
`round(arg)`	Rounds *arg* to the nearest whole number.
`rand()`	Random value with 'rectangular' distribution between 0 and 1.
`sqrt(arg)`	Square root of *arg*. If the value given to the function is negative, the function will return a 0.

4.4.4 Statistical Functions

Applied to a collection of scalar numeric quantities.

`max(arg1,arg2,arg3...)`	Returns the maximum of the given list of values.
`median(arg1,arg2,arg3...)`	Returns the median value of the given list of values. If an even number of arguments specified, the mean of the two middle values returned.
`min(arg1,arg2,arg3...)`	Returns the minimum of the given list of values.
`mode(arg1,arg2,arg3...)`	Returns the mode of the given list of values. If more than one modal value is found, returns the right-most modal value in the parameter list.
`percentile(perc,arg2,arg3...)`	Returns the value corresponding to the given percentile from the given list of values. The percentile (`perc`) should be a number between 0-100. If there is no value in the list corresponding to the precise percentile figure, the weighted mean of the two nearest ordered values is returned.
`rank(arg1,arg2,arg3...)`	Returns the rank (position in ordered list) of the given value (`arg1`) in the given list of values.

4.4.5 Colour Functions

Used to maipulate images where each raster value is an integer representing a red, green, blue triplet.

`blue(`*`arg`*`)`	Returns the blue component of the colour integer represented by `arg`. This is scaled between 0-255.
`green(`*`arg`*`)`	Returns the green component of the colour integer represented by `arg`. This is scaled between 0-255.
`red(`*`arg`*`)`	Returns the red component of the colour integer represented by `arg`. This is scaled between 0-255.
`rgb(`*`arg1`*`,`*`arg2`*`,`*`arg3`*`)`	Returns a single colour value composed of the red, green and blue colour components represented by `arg1`, `arg2` and `arg3` respectively. Each colour component should be a whole number between 0-255.

4.4.6 Logical Operators and Functions

Used to test boolean conditions that evaluate to true (non-zero) or false (zero).

`and`, `or`, `not`	Boolean AND, Boolean OR, Boolean NOT
`==`, `!=`	Equal to, not equal to
`>`, `<`, `>=`, `<=`	Greater than, less than, greater than or equals, less than or equals
`ifelse(`*`condition`*`, `*`value_if_true`*`, `*`value_if_false`*`)`	Evaluates the expression *condition*. If the condition is true or evaluates to a non-zero value, returns *value_if_true* otherwise returns *value_if_false*. Returned values can be numbers, strings or spatial objects.

4.4.7 Focal Operators

The are applied to entire spatial objects. They cannot be applied to scalar quantities.

`col`	Extracts column coordinate from raster. Allows spatial coordinates to be used in map algebra calculations.
`easting`	Extracts easting from a spatial object. Allows spatial coordinates to be used in map algebra calculations.
`northing`	Extracts northing from a spatial object. Allows spatial coordinates to be used in map algebra calculations.
`row`	Extracts row coordinate from raster. Allows spatial coordinates to be used in map algebra calculations.
`[`*`row_offset`*`,col_offset]`	Extracts raster value with the given coordinate offset. Allows focal neighbourhoods to be processed
`_` (*underscore*)	When appended to a spatial object name inside a map algebra expression, forces it to treated as an entire spatial object rather than scalar value. This can be useful when passing spatial objects to commands inside a map algebra expression.

4.4.8 Flow Control

Used to control program flow of a script.

`if (condition); {action}`	Processes the code in *action* only if *condition* is true or evaluates to a non-zero value.
`while (condition); {action}`	Repeatedly processes the code in *action* while *condition* is true or it evaluates to a non-zero value.
`returnvalue`	Leaves the the current user-defined function or script, optionally returning a value. If placed within a user-defined function, control is returned to the line of code that called the function. If placed elsewhere, the script will quit.

4.4.9 Miscellaneous

Other functions and reserved words.

`clear variable`	Clears the contents of the given variable. This can be used to free memory in scripts that contain variables holding very large spatial objects.
`functionfunction_name(arg1,arg2,arg3...)`	
`{function_code}`	Declares a user-defined function that can be called from within a script. The function can take optional parameters which become local to the code enclosed in braces.
`help`	If used as a parameter for any function or command, provides a brief message explaining how to use the function or command.

4.5 LandScript Command Reference

colouredit	Edits the colour table associated with a spatial object. Can use either preset or user-defined colour rules.
combine	Combines two spatial objects to form a new spatial object.
contour	Creates a contour map of a raster surface.
datetime	Retrieves the current date or time.
density	Creates a density surface from a set of point values.
draw	Creates a graphics file representing spatial objects.
echo	Displays text on screen or writes it to a file.
edit	Adds a metadata item to a spatial object (raster or vector map).
fractal	Fills the given raster with a fractal surface.
freqdist	Calculates a frequency distribution of a raster.
info	Extracts a metadata item from a spatial object (raster or vector map).
joinlines	Joins any line objects in the given vector map.
new	Creates a new spatial object (raster or vector map).
newraster	Creates a new raster map with the given bounds and metadata.
newvector	Creates a new vector map with the given bounds and metadata.
open	Opens a file representing a spatial object.
peakclass	Identifies peaks and summits from a DEM.
rcolourat	Extracts a colour component from a raster map using row and column coordinates.
rectify	Produces a rectified raster map from a set of control points.
removepits	Removes pits from a DEM. Creates a new pitless surface based on the given input surface.
removevoids	Removes voids from a raster by interpolating neighbours.
reproject	Reprojects a spatial object into a new coordinate system.
rvalueat	Extracts or assigns a single attribute associated with a raster map using row and column coordinates.
save	Saves a spatial object as a file.
simplify	Simplifies any line or area objects in the given vector map.
surfnetwork	Calculates the metric surface network from a DEM and surface feature map.
surfparam	Calculates a raster map of surface parameter values such as slope or aspect.
tocentroids	Finds centroids of any polygons in the given vector map.
toraster	Converts a TIN or vector map into a raster map.
triangulate	Creates a TIN from a raster surface or a Delaunay triangulation of points in a vector map.
valueat	Extracts or sets a single attribute associated with a spatial object (raster or vector map).
vectorstyle	Sets the vector appearance when drawn in a graphics file.
version	Identifies or queries the LandScript version to be used when interpreting this script.

4.5.1 colouredit() - Edits the colour table associated with a spatial object. Can use either preset or user-defined colour rules.

Does not return a value.

*Usage:***colouredit** (< `spatial_object`>**spObj**, [< `string`>**type**], [< `string`>**rules**]);

where

- **spObj**: Spatial Object to have new colour table.
- **type**: Type of colour table . Should be one of 'default', 'land1', 'land2', 'land3', 'land4', 'sea1', 'sea2', 'seaLand', 'random', 'grey1', 'grey2', 'exp1', 'diverging1', 'diverging2', 'raw', 'file', 'rules', 'addRule'. Default is 'default'.
- **rules**: Name of file containing colour table, or set of numeric rules, or 'null' if not used. Default is 'null'.

All preset colour tables are scaled between the minimum and maximum values of the spatial object with which it is associated unless the `rules` parameter is provided (see below). If the `type` is set to ' `raw`', the object associated with the colour table is assumed to contain raw 32-bit colour values.

If `type` is set to one of the preset colour tables, `rules` can optionally provide minimum and maximum values between which the colour table is scaled. This should be provided as a whitespace separated pair of values.

If `type` is set to ' `file`', ' `rules`' or ' `addRule`', user-defined colour rules should be provided. They should either be stored in a standard ' `.ctb`' (`file`) colourtable file, or provided as a single string of comma separated rules (`rules` and `addRule`). If type is set to ' `file`' or ' `rules`' the new rules replace any that might exist in the colour table. If it is set to ' `addRule`', the given rules are added to any which exist in the colour table. Each rule consists of a *value colour* sequence, where *value* represents the attribute with which to associate a colour and *colour* is either single greyscale number between 0-255, or an `rgb` triplet or and `rgba` quad. An optional `(D)` value can also be provided to indicate discrete colour rule. If `(D)` not provided colour rules are assumed to be continuous interpolations.

4.5.2 combine() - Combines two spatial objects to form a new spatial object.

Returns a value of type < `spatial_object`>.

*Usage:***combine** (< `spatial_object`>**first**, < `spatial_object`>**second**, [< `string`>**type**], [< `string`>**rule**], [< `boolean`>**replaceNull**]);

where

- **first**: first spatial object to combine.
- **second**: second spatial object to combine.
- **type**: type of combination (intersection or union). Default is 'union'.
- **rule**: raster object to prioritise ('first', 'second' or 'average') or type of vector intersection ('object', 'map' or 'points'). Default is 'first'.
- **replaceNull**: Indicates whether null values are replaced where possible. Default is 'true'.

The type of combination will depend on the types of spatial objects provided and the `rule` parameter. If two raster maps are provided, the combination rule determines whether the attributes of the first, second or average of the two are used to create the attributes of the combined raster. In the case of raster cells with a *null* value, this rule can be superseded if `replaceNull` is true. For two intersecting vector maps `rule` determines if the intersection is based on the bounding rectangle of each vector (' `map`'), or the intersection of any area objects within each map (' `object`)'.

If the two supplied spatial objects are of different types, the type of combination is determined by the

order of objects provided to the command. If ' `first`' is a raster map, it will be 'cookie cut' according to the objects or bounding rectangle of the vector map, again depending on the supplied `rule`. If ' `first`' is a vector map containing points, a new attribute will be added to the vector map's attribute table that will be contain the raster at each point location.

4.5.3 contour() - Creates a contour map of a raster surface.

Returns a value of type < `vector_map`*>.*

*Usage:***contour** (<`raster_map`>**surface**, <`number`>**min**, <`number`>**interval**, [<`number`>**gridWidth**]);

where

- **surface**: Raster surface to contour.
- **min**: minimum contour value.
- **interval**: contour interval.
- **gridWidth**: Grid cell spacing of sample points. Default is '4'.

4.5.4 datetime() - Retrieves the current date or time.

Returns a value of type < `string`*>.*

*Usage:***datetime** ([< `string`>**format**]);

where

- **format**: Format of date or time to report. Should be one of 'full', 'date', 'time', 'milliseconds' Default is 'milliseconds'.

Allows the current date and/or time to be retrieved. If the `format` is set to ' `full`', text representing the current date and time is returned. If ' `date`' or ' `time`' is requested, text representing the current date or current time is returned. All of these can be useful for adding metadata descriptions to newly created objects. The default value of ' `milliseconds`' returns a number representing the number of milliseconds that have elapsed since some arbitrary start time. This can be useful for benchmarking or providing real-time information on how long a script has been running for.

4.5.5 density() - Creates a density surface from a set of point values.

Returns a value of type < `raster_map`*>.*

*Usage:***density** (< `vector_map`>**vector**, < `number`>**windowSize**, < `number`>**xRes**, [< `number`>**yRes**]);

where

- **vector**: vector map storing points from which to calculate density.
- **windowSize**: window size used to calculate density. Should be odd and at least 1.
- **xRes**: resolution of resulting density surface in x direction.
- **yRes**: resolution of resulting density surface in y direction or 'null' if same as xRes. Default is 'null'.

4.5.6 draw() - Creates a graphics file representing spatial objects.

Does not return a value.

*Usage:***draw** (< `string`>**fileName**, [< `raster_map`>**raster1**], [< `raster_map`>**raster2**], [< `vector_map`>**vector1**], [< `vector_map`>**vector2**], [< `string`>**type**], [< `number`>**blendWeight**]);

where

- **fileName**: Name of graphics file to create. Should have a .png, .jpg or .gif extension.
- **raster1**: Primary raster map to draw. Default is 'null'.
- **raster2**: Secondary raster map to draw. Default is 'null'.
- **vector1**: Primary vector map to overlay. Default is 'null'.
- **vector2**: Secondary vector map to overlay. Default is 'null'.
- **type**: Type of image to draw. Should be one of 'raster','relief','hueInt', 'hueSat','blend' or 'add'. Default is 'raster'.
- **blendWeight**: The blending weight given to the primary raster. This only applies if the 'blend' type is selected. Default is '50'.

This command can be used for more sophisticated graphical output from a script. For simple display of a single raster or vector map, consider using `save()` instead, selecting 'image' as the output type.

The `type` options ' `hueInt`' (hue-intensity image), ' `hueSat`' (hue-saturation image), ' `blend`' (blended image) and ' `add`' (image addition) require both a primary and secondary raster to be supplied. The type ' `relief` will use the shaded relief of the primary raster and the colours of the secondary raster is it is provided. If ' `blend`' is requested, the optional parameter `blendWeight` can be used to determine the weighting given to the primary and secondary rasters.

4.5.7 echo() - Displays text on screen or writes it to a file.

Does not return a value.

*Usage:***echo** ([< `string`>**text**], [< `string`>**fileName**]);

where

- **text**: Text, variable or expression to display. Default is an empty string.
- **fileName**: File to store output Default is an empty string.

The text to display can be a combination of literal strings enclosed in "quotes", variables and expressions. Can also include spectial control characters ' `\n`' to indicate a new line and ' `\t`' to indicate a tab stop. Literal text is separated from variables and expressions using the `&` operator. For example

```
mapNum = 5;
echo("Map number "&mapNum&" contains "&240*240&" pixels.");
```

would display the following output:

```
Map number 5 contains 57600 pixels.
```

If the optional `fileName` parameter is given, the text is written to the given file instead.

4.5.8 edit() - Adds a metadata item to a spatial object (raster or vector map).

Does not return a value.

*Usage:***edit** (< `spatial_object`>**spObj**, < `string`>**item**, < `string`>**value**);

where

- **spObj**: Spatial object to edit.
- **item**: Type of metadata to edit. Should be one of 'N', 'S', 'E', 'W', 'xRes', 'yRes', 'projection', 'ellipsoid', 'type', 'attributes', 'attCol', 'title', 'notes', 'sunElev', 'sunAzim', 'pShade', 'vExag', 'aBias', 'ambient', 'shine', 'diffuse' or 'specular'.
- **value**: Value of to be associated with the metadata item.

Metadata items can be provided as strings or numeric values. In the case of raster maps, if any of the bounding values or resolution are changed, the change must not affect the number of rows or columns in the raster. Note that changing the bounding values (N, S, E, W) of a map will not cookie-cut it. To subset a map, create a new empty object (e.g. `newraster` or `newvector`) then intersect it with the map using `combine`.

For valid values of ' `type`', ' `projection`' and ' `ellipsoid`', see `newRaster()` or `newVector()`.

The value associated with the ' `attributes`' type should be an `.atr` file containing an attribute table or 'noTable' if an object should not have an attribute table. If an attribute table exists, the *active attribute* can be set by setting `attCol` to a number representing the column of the active attribute.

The lighting parameters `sunElev`, `sunAzim` determine sun direction when calculating shaded relief and should be expressed in degrees. `pShade` and `aBias` should be percentages and represent the proportion of greyscale shading and the aspect bias respectively when calculating shaded relief. `vExag` determines the vertical exaggeration used when calculating shaded relief. Finally, the light parameters `shine`, `ambient`, `diffuse`, and `specular` are all expressed as percentages and are used when calculating shaded relief as a surface parameter (e.g. with the command `surfparam()`)

4.5.9 fractal() - Fills the given raster with a fractal surface.

Does not return a value.

*Usage:***fractal** (< `raster_map`>**raster**, [< `number`>**dimension**]);

where

- **raster**: Raster to contain the fractal surface
- **dimension**: Fractal dimension of surface to create. Default is '2.1'.

4.5.10 freqdist() - Calculates a frequency distribution of a raster.

Does not return a value.

*Usage:***freqdist** (< `raster_map`>**raster**, < `string`>**fileName**, [< `number`>**min**], [< `number`>**binWidth**], [< `string`>**ignoreVals**], [< `boolean`>**calcHammock**]);

where

- **raster**: Raster map from which to measure frequency distribution.
- **fileName**: Name of file to store frequency distribution values.
- **min**: Lower bound of minimum frequency class. If null, raster's minimum value used. Default is 'null'.
- **binWidth**: Width of each frequency class or modulus if hammock distribution is to be calculated. If not specified, frequency distribution bin width will be 1, and hammock modulus will be 10. Default is 'null'.
- **ignoreVals**: Optional list of values to be ignored when calculating distribution. If more than one value is to be ignored, a space-separated list of numbers should be provided in quotes. Default is 'null'.
- **calcHammock**: If true, a hammock distribution will be calculated instead of frequency distribution. Default is 'false'.

The raster's frequency distribution is calculated in classes ranging from its minimum value to maximum values in widths defined by the `binWidth` parameter. The `ignoreVals` parameter can be used to exclude certain values from contributing to the distribution, for example sea areas with value of 0. If more than one value is to be ignored, a string of space separated values should be provided.

If `calcHammock` is `true`, the frequency distribution of the modulus of raster values is found instead. When generating a hammock distribution, the modulus is taken from the `binWidth` parameter. For example, a `binWidth` of 10 would find the distribution of raster value remainders after dividing each cell by 10. This allows a comparison of those values ending in 0 with those ending in digits 1-9 and can be useful for detecting contour and rounding artifacts in interpolated DEMs.

In all cases a text file is created with one category per line consisting of a comma-separated lower bin value and frequency.

4.5.11 info() - Extracts a metadata item from a spatial object (raster or vector map).

Returns a value of type < *`string`*>.

*Usage:***info** (< `spatial_object`>**spObj**, < `string`>**item**);

where

- **spObj**: Spatial object from which to extract information.
- **item**: Type of metadata to extract. Should be one of 'N', 'S', 'E', 'W', 'xRes', 'yRes', 'numRows', 'numCols', 'projection', 'min', 'max', 'mean', 'median', 'percentileN', 'mode', 'modef', 'sum', 'numVals', 'stdev', 'skew', 'kurtosis', 'moran', 'fracD', 'type', title', 'notes', 'sunElev', 'sunAzim', 'pShade', 'vExag', 'aBias', 'ambient', 'shine', 'diffuse' or 'specular'.

The result of calling this command can be either text or a number depending on the type of metadata selected. Statistical information that can be extracted from raster maps includes number of non-null cells (`numVals`), sum of all cells (`sum`), measures of average - `mean`, `median`, `mode`, and frequency of modal

value (`modef`); measures of dispersion - `stdev`, `skew` and `kurtosis`, `percentile`*N* (where *N* is any number between 0-100); measures of spatial autocorrelation (`moran`) and fractal dimension (`fracD`).

4.5.12 joinlines() - Joins any line objects in the given vector map.

Returns a value of type < `vector_map`>.

*Usage:***joinlines** (< `vector_map`>**vector**);

where

- **vector**: Vector map containing lines to join.

The result of this operation is a new vector map identical to the original except that any linear features will be joined as a single line. If the original lines have different attributes, the new joined line will take the attribute of the first one to be joined. Uses for this operation include the defragmenting a GPS track with patchy signal, the joining of a multi-session line digitization, the rejoining of lines truncated during vector map tiling.

4.5.13 new() - Creates a new spatial object (raster or vector map).

Returns a value of type < `spatial_object`>.

*Usage:***new** (< `spatial_object`>**spObj**, [< `boolean`>**copyContents**]);

where

- **spObj**: Spatial object upon which to base the new object.
- **copyContents**: If true contents and metadata are copied into new object, otherwise only the metadata are copied. Default is 'false'.

The result of this command, which is always a spatial object of some kind, is normally placed in a variable.

For example,

```
myRaster = new(existingRaster);
```

will create a new empty raster map called `myRaster` that contains the same metadata as those in `existingRaster`.

```
myRaster = new(existingRaster,"true");
```

will create a duplicate of `existingRaster` including data and metadata.

4.5.14 newraster() - Creates a new raster map with the given bounds and metadata.

Returns a value of type < `raster_map`>.

Usage:**newraster** (< `number`>**xOrigin**, < `number`>**yOrigin**, < `number`>**xRes**, < `number`>**yRes**, < `number`>**numRows**, < `number`>**numCols**, [< `string`>**title**], [< `string`>**notes**], [< `string`>**projection**], [< `string`>**ellipsoid**], [< `string`>**type**]);

where

- **xOrigin**: Western boundary of raster
- **yOrigin**: Southern boundary of raster
- **xRes**: E-W width of single raster cell
- **yRes**: N-S height of single raster cell
- **numRows**: Number of rows in raster
- **numCols**: Number of columns in raster
- **title**: Title of new raster. Default is 'New raster map'.
- **notes**: Notes associated with new raster. Default is 'Raster created by LandScript'.
- **projection**: Map projection used by new raster (e.g. 'AlbersBC', 'FrenchNTF', 'LatLong', 'OSGB', 'SwissNG', 'UTM 12 a', 'undefined'). Default is 'undefined'.
- **ellipsoid**: Projection ellipsoid used by new raster (e.g. 'Airy 1830', 'Australian National', 'Bessel 1841', 'Bessel 1841 (Namibia)', 'Clarke 1866', 'Clarke 1880', 'Everest', 'Fischer 1960', 'Fischer 1968', 'GRS 1967', 'GRS 1980', 'Helmert 1906', 'Hough', 'International', 'Krassovsky', 'Modified Airy', 'Modified Everest', 'Modified Fischer 1960', 'South American', 'WGS 60', 'WGS 66', 'WGS 72', 'WGS 84', 'undefined'). Default is 'undefined'.
- **type**: Type of raster (e.g. 'Elevation', 'Slope', 'Image', 'Fuzzy' , 'Other'). Default is 'other'.

The result of this command, which is always a raster map, is normally placed in a variable. Each new declaration must include at least enough metadata to define the origin and dimensions of the new raster map to be created. The origin is defined relative to the bottom left corner, and the number of rows/-columns along with the resolution in the x and y directions implicitly define the top-right corner. These bounding values should represent the *outer edges* of the corner cells in the raster map.

For example,

```
myRaster = newraster(100,100, 50, 50, 800,1000);
```

will create a new empty raster with a bottom left corner at (100,100) and top right corner at (50100,40100). Other metadata can be optionally specified. For example,

```
myRaster = newraster(100,100, 50, 50, 800,1000,"My title",
"Raster created by landscript command","OSGB","Airy 1830","Elevation");
```

4.5.15 newvector() - Creates a new vector map with the given bounds and metadata.

Returns a value of type < `vector_map`>.

Usage:**newvector** (< `number`>**west**, < `number`>**south**, < `number`>**east**, < `number`>**north**, [< `string`>**title**], [< `string`>**notes**], [< `string`>**projection**], [< `string`>**ellipsoid**]);

where

- **west**: Western boundary of vector map
- **south**: Southern boundary of vector map
- **east**: Eastern boundary of vector map
- **north**: Northern boundary of vector map
- **title**: Title of new vector map. Default is 'New vector map'.
- **notes**: Notes associated with new vector. Default is 'Vector map created by LandScript'.
- **projection**: Map projection used by new raster (e.g. 'AlbersBC', 'FrenchNTF', 'LatLong', 'OSGB', 'SwissNG', 'UTM 12 a', 'undefined'). Default is 'undefined'.
- **ellipsoid**: Projection ellipsoid used by new raster (e.g. 'Airy 1830', 'Australian National', 'Bessel 1841', 'Bessel 1841 (Namibia)', 'Clarke 1866', 'Clarke 1880', 'Everest', 'Fischer 1960', 'Fischer 1968', 'GRS 1967', 'GRS 1980', 'Helmert 1906', 'Hough', 'International', 'Krassovsky', 'Modified Airy', 'Modified Everest', 'Modified Fischer 1960', 'South American', 'WGS 60', 'WGS 66', 'WGS 72', 'WGS 84', 'undefined'). Default is 'undefined'.

The result of this command, which is always a vector map, is normally placed in a variable. Each new declaration must include at least enough metadata to define the bottom-left and top-right of the map.

For example,

```
myVectorMap = newvector(302100,450000, 303100, 451000);
```

will create a new empty vector map with a bottom left corner at (302100,450000) and top right corner at (303100,451000). Other metadata can be optionally specified. For example,

```
myVectorMap = new(302100,450000, 303100, 451000,"My title",
"Vector map created by landscript command","OSGB","Airy 1830");.
```

4.5.16 open() - Opens a file representing a spatial object.

Returns a value of type < `spatial_object`>.

*Usage:***open** (< `string`>**file**, [< `string`>**type**]);

where

- **file**: full pathname of file to open.
- **type**: type of file to open. For list of available types, see the LandSerf 'Howto' document on file formats. Specifying 'guess' will try to guess the file type from its extension. Default is 'guess'.

4.5.17 peakclass() - Identifies peaks and summits from a DEM.

Returns a value of type < `raster_map`>.

*Usage:***peakclass** (< `raster_map`>**dem**, < `number`>**minDrop**, [< `number`>**minElev**], [< `raster_map`>**fuzzyPeakRaster**], [< `vector_map`>**summitNetwork**], [< `raster_map`>**hierarchyRaster**]);

where

- **dem**: DEM from which to identify peaks and summits.
- **minDrop**: Minimum vertical drop defining summit.
- **minElev**: Minimum elevation for any summit. If null, minimum elevation value is not used to define summit. Default is 'null'.
- **fuzzyPeakRaster**: Raster into which fuzzy peak values are placed. If null, fuzzy peaks are not calculated. Default is 'null'.
- **summitNetwork**: Vector map into which summit network is placed. If null, summit network is not calculated. Default is 'null'.
- **hierarchyRaster**: Raster into which peak hierarchy values are placed. If null, peak hierarchy is not calculated. Default is 'null'.

Uses 'relative drop' to identify summits and their catchments from the given DEM. The `minDrop` parameter sets the minimum vertical drop required for a path from a summit to a higher neighbour. The larger the value the fewer the summits, but the larger in scale. If `minElev` is also defined, an additional constraint can be applied to summit definition. Only locations with at least this height will be considered as candidate summits. In all cases, a raster containing summit points and their contributing areas is generated. Additionally, a number of optional output maps can also be generated. If a `fuzzyPeakRaster` is given, this will be filled with values indicating peak membership ranging from 1 at the summit point to 0 at the boundary of its contributing area. If a `summitNetwork` vector map is given, this will be filled with a set of points representing summits, their passes and connecting topological ridge lines. If a `hierarchyRaster` is supplied, this will be populated with each peak's nested hierarchy value.

4.5.18 rcolourat() - Extracts a colour component from a raster map using row and column coordinates.

Returns a value of type < `number`>.

*Usage:***rcolourat** (< `raster_map`>**raster**, < `number`>**row**, < `number`>**col**, < `string`>**component**);

where

- **raster**: Raster map from which to query colour.
- **row**: Row coordinate of raster to query. This should be between 0 and (numRows-1) of the raster.
- **col**: Column coordinate of raster to query. This should be between 0 and (numCols-1) of the raster.
- **component**: Colour component to extract. Should be one of 'red', 'green', 'blue', or 'alpha'

The raster's row and column origin is assumed to be at the top left. Note also that the order of coordinates is `row` followed by `col`. The red, green, blue or alpha component is scaled between 0 and 255. If the query location is outside the raster's bounds, a warning message is reported and 0 is returned.

See also `rgb()`, `red()`, `green()` and `blue()` that can convert to and from colour integers and red, green and blue values. These functions can be used in local map algebra operations and will be much quicker than `rcolourat()` for processing entire rasters.

4.5.19 rectify() - Produces a rectified raster map from a set of control points.

Returns a value of type < `raster_map`>.

*Usage:***rectify** (< `raster_map`>**raster**, < `string`>**ctrlPoints**, < `string`>**order**, < `boolean`>**interpolate**,

< number>**xRes**, [< number>**yRes**], [< number>**north**], [< number>**south**], [< number>**east**], [< number>**west**]);

where

- **raster**: Raster to rectify.
- **ctrlPoints**: File holding the control points used to create the transformation.
- **order**: type of transformation (one of 'linear', 'quadratic' or 'cubic').
- **interpolate**: Interpolates between raster cells if true, otherwise selected nearest neighbour.
- **xRes**: resolution of resulting raster in x direction.
- **yRes**: resolution of resulting raster in y direction or 'null' if same as xRes. Default is 'null'.
- **north**: Northern boundary of rectified raster or 'null' if defined by control points. Default is 'null'.
- **south**: Southern boundary of rectified raster or 'null' if defined by control points. Default is 'null'.
- **east**: Eastern boundary of rectified raster or 'null' if defined by control points. Default is 'null'.
- **west**: Western boundary of rectified raster or 'null' if defined by control points. Default is 'null'.

4.5.20 removepits() - Removes pits from a DEM. Creates a new pitless surface based on the given input surface.

Returns a value of type < *raster_map*>.

*Usage:***removepits** (< raster_map>**dem**, [< string>**type**]);

where

- **dem**: Surface from which to remove pits.
- **type**: Type of pit processing. 'infill' will flood pits to their spill points; 'channel' will create an outflow channel from a pit to a lower point. Default is 'infill'.

The most appropriate type of pit processing will depend on the application and nature of the DEM being processed. Infilling is best suited to the removal of small isolated pits in noisy data. It has the advantage of being quick to compute. Channelling outflow from pits is better suited to hydrological applications that require all non-edge points to flow off the DEM. Channelling is not suited to surfaces with large or deep pits.

The changed elevation values, whether through infilling or channelling, can be found by taking the difference between the original surface and the one returned by this command.

4.5.21 removevoids() - Removes voids from a raster by interpolating neighbours.

Returns a value of type < *raster_map*>.

*Usage:***removevoids** (< raster_map>**raster**);

where

- **raster**: Raster map containing data with voids.

4.5.22 reproject() - Reprojects a spatial object into a new coordinate system.

Returns a value of type < `spatial_object`>.

*Usage:***reproject** (< `spatial_object`>**spObj**, < `string`>**type**, [< `boolean`>**interpolate**], [< `number`>**xRes**], [< `number`>**yRes**], [< `number`>**north**], [< `number`>**south**], [< `number`>**east**], [< `number`>**west**]);

where

- **spObj**: Spatial object to reproject.
- **type**: Type of reprojection to perform. Options include 'AlbersBC', 'AlbersUS', 'AlbersUSAll', 'FrenchNTF', 'LatLong', 'OSGB', 'SwissNG', and 'UTM'. Note that not all reprojection combinations are available.
- **interpolate**: Interpolates between raster cells if true, otherwise selected nearest neighbour. This parameter is ignored for reprojection of vector maps. Default is 'true'.
- **xRes**: resolution of resulting raster in x direction or 'null' if calculated automatically/not relevant. Default is 'null'.
- **yRes**: resolution of resulting raster in y direction or 'null' if same as xRes. Default is 'null'.
- **north**: Northern boundary of reprojected spatial object or 'null' if calculated automatically. Default is 'null'.
- **south**: Southern boundary of reprojected spatial object or 'null' if calculated automatically. Default is 'null'.
- **east**: Eastern boundary of reprojected spatial object or 'null' if calculated automatically. Default is 'null'.
- **west**: Western boundary of reprojected spatial object or 'null' if calculated automatically. Default is 'null'.

4.5.23 rvalueat() - Extracts or assigns a single attribute associated with a raster map using row and column coordinates.

Returns a value of type < `string`>.

*Usage:***rvalueat** (< `raster_map`>**raster**, < `number`>**row**, < `number`>**col**, [< `number`>**att**]);

where

- **raster**: Raster map from which to query or assign attribute.
- **row**: Row coordinate of raster to query or assign attribute. This should be between 0 and (numRows-1) of the raster.
- **col**: Column coordinate of raster to query or assign attribute. This should be between 0 and (numCols-1) of the raster.
- **att**: Optional attribute to assign to the given location. If not specified, this will query the attribute at the given location. Default is 'null'.

The raster's row and column origin is assumed to be at the top left. Note also that the order of coordinates is `row` followed by `col`, in contrast to `valueat()` which requires coordinates in `easting,northing` order.

The value to be associated with the given location can be assigned if the `att` parameter is set. If `att` not set this command will perform a raster query, the result of which can be either text or a number

depending on the attribute selected. If the raster map has an attribute table associated with it, the *active attribute* will determine which column from the table is used to return a value. If the object has no attribute table, the numeric attribute is returned. If the query location is outside the raster's bounds, a warning message is reported and a null value is returned.

4.5.24 save() - Saves a spatial object as a file.

Does not return a value.

*Usage:***save** (< `spatial_object`>**spObj**, < `string`>**file**, [< `string`>**type**], [< `string`>**byteOrder**], [< `number`>**wordSize**]);

where

- **spObj**: Spatial object to save.
- **file**: Full pathname of file to save.
- **type**: Type of file format to save. For list of available types, see the LandSerf 'Howto' document on file formats. Specifying 'native' will save the object in LandSerf's native raster map or vector map format. Default is 'native'.
- **byteOrder**: Byte order. This only applies to the generic binary raster format. If supplied it should be one of 'bigEndian' (Motorola) or 'littleEndian' (Intel). Default is 'littleEndian'.
- **wordSize**: Number of bytes per raster cell. This only applies to the generic binary raster format. If supplied it should be one either 1, 2 or 4. Default is '4'.

4.5.25 simplify() - Simplifies any line or area objects in the given vector map.

Returns a value of type < `vector_map`>.

*Usage:***simplify** (< `vector_map`>**vector**, < `number`>**distance**);

where

- **vector**: Vector map to simplify.
- **distance**: Tolerance distance that controls the degree of simplification.

The result of this operation is a new vector map identical to the original except that any linear features or area boundaries will be simplified. Uses the Douglas-Peucker line simplification algorithm.

4.5.26 surfnetwork() - Calculates the metric surface network from a DEM and surface feature map.

Returns a value of type < `vector_map`>.

*Usage:***surfnetwork** (< `raster_map`>**surf**, < `raster_map`>**features**);

where

- **surf**: Surface over which to calculate network.
- **features**: Surface features on which to base network.

A surface feature network consists of points representing *pits*, *passes* and *peaks* joined by lines repsenting *channels* (from pits to passes) and *ridges* (from passes to peaks). Input must consist of a Digital Elevation Model and a raster representing the (possibly fragmented) surface features.

4.5.27 surfparam() - Calculates a raster map of surface parameter values such as slope or aspect.

Returns a value of type < `raster_map`>.

*Usage:***surfparam** (< `raster_map`>**surf**, < `string`>**type**, [< `number`>**wSize**], [< `number`>**decay**], [< `number`>**slopetol**], [< `number`>**curvetol**], [< `string`>**scaleSummary**]);

where

- **surf**: Surface from which to calculate parameter.
- **type**: Type of surface parameter. Options include 'elev', 'slope', 'aspect', 'profc', 'planc', 'crossc', 'longc', 'minc', 'maxc', 'meanc', 'feature', 'netfeature', 'residual', 'meanresidual, 'shade', 'fractal', 'coeffA', 'coeffB', 'coeffC', 'coeffD', 'coeffE', 'coeffF'.
- **wSize**: Window size used for calculation. If a multiscale summary is selected (see below), wSize should be the maximum window size over which to measure. In all cases, wSize should be at least 3 and an odd number. If 'null', the window size defined by 'surf' is used. Default is 'null'.
- **decay**: Distance decay used for calculation. If 'null', the decay exponent defined in 'surf' is used. Default is 'null'.
- **slopetol**: Slope tolerance used for feature classification (ignored by other parameters). If 'null', the tolerance value defined in 'surf' is used. Default is 'null'.
- **curvetol**: Curvature tolerance used for feature classification (ignored by other parameters). If 'null', the tolerance value defined in 'surf' is used. Default is 'null'.
- **scaleSummary**: Multiscale summary surface to create. Valid values are 'average', 'variation', 'min', 'max' and 'null'. If 'null', the parameter is calculated at a single window size only. Default is 'null'.

The type of surface measurement is determined by the `type` parameter at the scale determined by `size`. The `decay` parameter determines the weighting given to neighbouring cells as a function of distance from the centre of the cell to measure. A value of 0 indicates equal weight to all cells, 1 a linear distance decay, 2 a squared distance decay etc. Most parameter types should be self-explanatory. The values ' `coeffA`', ' `coeffB`' etc. allow the quadratic coefficients a-f of the best fit surface $z = ax^2 + by^2 + cxy + dx + ey + f$ to be retrieved.

Parameters can be calculated over a range of window sizes and summarised by including a non-null value for `scaleSummary`. In which case, `wSize`, which must be at least 5, represents the largest window size, and the smallest window size is always 3x3.

4.5.28 tocentroids() - Finds centroids of any polygons in the given vector map.

Returns a value of type < `vector_map`>.

*Usage:***tocentroids** (< `vector_map`>**vector**);

where

- **vector**: Vector map to in which to find centroids.

The result of this operation is a new vector map containing only points representing the centroids of each polygon in the original vector map. The centroids share the same attributes and colours as the polygons they represent.

4.5.29 toraster() - Converts a TIN or vector map into a raster map.

Returns a value of type < *`raster_map`*>.

*Usage:***toraster** (< `vector_map`>**vector**, < `number`>**xRes**, [< `number`>**yRes**]);

where

- **vector**: TIN or vector map to convert.
- **xRes**: resolution of resulting raster in x direction.
- **yRes**: resolution of resulting raster in y direction or 'null' if same as xRes. Default is 'null'.

4.5.30 triangulate() - Creates a TIN from a raster surface or a Delaunay triangulation of points in a vector map.

Returns a value of type < *`vector_map`*>.

*Usage:***triangulate** (< `spatial_object`>**spObj**, [< `number`>**numSamples**], [< `number`>**rmse**], [< `number`>**maxError**], [< `raster_map`>**errorSurface**]);

where

- **spObj**: Raster or points to triangulate.
- **numSamples**: maximum number of points to sample or 'null' if not relevant. Default is 'null'.
- **rmse**: maximum root mean squared error allowed or 'null' if not relevant. Default is 'null'.
- **maxError**: maximum error allowed or 'null' if not relevant. Default is 'null'.
- **errorSurface**: raster to hold error surface. Default is 'null'.

4.5.31 valueat() - Extracts or sets a single attribute associated with a spatial object (raster or vector map).

Returns a value of type < *`string`*>.

*Usage:***valueat** (< `spatial_object`>**spObj**, < `number`>**easting**, < `number`>**northing**, [< `number`>**att**]);

where

- **spObj**: Spatial object from which to query or assign attribute.
- **easting**: Easting of location to query or assign. This should be in the geographic units used by the spatial object.
- **northing**: Northing of location to query or assign. This should be in the geographic units used by the spatial object.
- **att**: Optional attribute to assign to the given location. If not specified, this will query the attribute at the given location. Default is 'null'.

The origin of the spatial object is assumed to be at the bottom left, and the coordinate system used is defined by the spatial object itself. Both raster and vector maps can be queried or set using this command, but to query or assign raster map attributes using row and column coordinates, use `rvalueat()` instead.

The value to be associated with the given location can be assigned if the `att` parameter is set. If it is not, the result of calling this command can be either text or a number depending on the attribute selected. If the spatial object has an attribute table associated with it, the *active attribute* will determine which column from the table is used to return a value. If the object has no attribute table, the numeric attribute is returned. If the query location is outside the spatial object's bounds, a warning message is reported and null is returned.

4.5.32 vectorstyle() - Sets the vector appearance when drawn in a graphics file.

Does not return a value.

*Usage:***vectorstyle** (< `string`>**action**, < `string`>**value**);

where

- **action**: Style action to perform. Should be one of 'open', 'save', 'boundarycolour', 'fixedline', 'fixedpoint', 'labelbackground', 'labelsize', 'labelforeground', 'labelposition', 'linewidth', 'pointsize', 'polygonopacity', 'render', 'showboundary', 'showlabels', 'surroundline' or 'surroundpoint'.
- **value**: Style setting or name of style file to load or save.

Allows the vector appearance to be changed when creating graphics files with the `draw()` or `save()` commands. The options available correspond to those available in LandSerf's 'Vector Appearance' window. Alternatively, settings can be saved or loaded with the `save` and `open` options.

The size of point and line rendering can be set with `linewidth` and `pointsize`. In both cases, the *value* parameter should be a number of pixels. This need not be a whole number, for example a `linewidth` of 0.3 gives faint lines useful for contour rendering or polygon boundaries.

Point and line style can also be controlled with the `surroundpoint` and `surroundline` options, which should be set to `true` or `false`. If true, points or lines are drawn with a surrounding line. This can be useful for highlighting underlying raster objects without obscuring them.

Polygon appearance can be controlled with the `polygonopacity` option that should have a value ranging from 0-1. Non-opaque polygons are useful for showing overlapping areal features or for combining with an underlying raster. Polygon boundary appearance is controlled with the `boundarycolour` option that takes a colour value defined by a series of space-separated colour values scaled between 0-255. Grey levels

are specified with a single number; grey levels with transparency by a pair of numbers; red-green-blue values by a triplet and red-green-blue-transparency with a set of four numbers. Polygon boundaries can be turned on and off by setting `showboundary` to `true` or `false`.

Point label colours are set with `labelbackground` and `labelforeground` each taking a colour value as described above. Labels can be turned on or off with `showLabels` that should be set either to `true` or `false`. Label size is controlled with the `labelsize` option requiring a numeric value corresponding to the height in pixels of the label. Its position is controlled with `labelposition` which should have a value corresponding to one of eight compass directions (`north`, `northeast`, `east` etc.) or `centre` to centre the label on the point it represents.

The type of rendering can be set with `render` that can take the value `speed` for faster, low quality rendering or `quality` for higher quality anti-aliased rendering.

4.5.33 version() - Identifies or queries the LandScript version to be used when interpreting this script.

Returns a value of type < `number`*>.*

*Usage:***version** ([< `number`>**versionNum**], [< `boolean`>**showDetail**]);

where

- **versionNum**: LandScript version number Default is '-1'.
- **showDetail**: If true, a full description of the LandScript version is given; if false, just a number representing the current version is reported. Default is 'false'.

The LandScript version should be identified at the top of each script to ensure compatibility with future enhancements to the language. If a value of less than 0 is supplied, the current Landscript version is assumed and displayed. If a version more recent (larger) than the current version is supplied, a warning message is displayed.

This command will return the version of LandScript that will be used to interpret the script, which was either set by this command, or is the current version if out of range or not supplied. The result can be stored in a variable and used to perform sections of script conditionally dependent on the version number.

If `showDetail` is true more detailed information is displayed including the version of Java used to run LandScript.

4.6 Creating New Raster and Vector Maps

Some example scripts to help you get started with creating new spatial objects with LandScript.

4.6.1 Creating and saving a new raster containing a fractal surface

```
# Creates a fractal surface of 250x250 cells and fractal dimension of 2.01
version(1.0);

# Change the directory below to correspond to your file location.
baseDir = "c:/Program Files/LandSerf/data/landscriptExamples/";

fracSurf = newraster(1000,1000,10,10,250,250);
fractal(fracSurf, 2.01);
save(fracSurf,baseDir&"fractal.srf");
```

4.6.2 Creating and saving a new raster with customised metadata

```
# Creates a fractal surface of 250x250 cells and fractal dimension of 2.01
version(1.0);

# Change the directory below to correspond to your file location.
baseDir = "c:/Program Files/LandSerf/data/landscriptExamples/";

fracSurf = newraster(1000,1000,10,10,250,250,"A simulated surface");
fractal(fracSurf, 2.01);

# Customise the new surface's metadata
colouredit(fracSurf,"land1");
edit(fracSurf,"notes","Fractal surface generated by spectral synthesis with d=2.01");

# Save the new surface.
save(fracSurf,baseDir&"fractal.srf");
```

4.6.3 Creating a new blank vector map

```
# Creates an empty vector map.
version(1.0);

# Change the directory below to correspond to your file location.
baseDir = "c:/Program Files/LandSerf/data/landscriptExamples/";

# Boundaries of the new vector map.
west  = 123000;
south = 456000;
east  = 623000;
north = 823000;
myVect = newvector(west,south,east,north,"Blank","An empty vector map");

# Set the default colour table and save the new vector.
colouredit(myVect,"grey1");
save(myVect,baseDir&"blank.vec");
```

4.6.4 Creating a 2d mathematical function surface

```
# Creates a raster based on a 2d mathematical function
version(1.0);

# Change the directory below to correspond to your file location.
baseDir = "c:/Program Files/LandSerf/data/landscriptExamples/";

nyquist = newraster(-100,-100,1,1,201,201);
nyquist = 10* (sin(easting/10) / (easting/10)) * (sin(northing/10) /(northing/10));

# Add metadata.
edit(nyquist,"title","Nyquist");
edit(nyquist,"notes",
            "Nyquist sampling model based on the 2d sinC function "&
            "((sin PI x / PI x)(sin PI y / PI y) )");
edit(nyquist,"type","elevation");
colouredit(nyquist,"land4");

# Save the new raster map.
save(nyquist,baseDir&"nyquist.srf");
```

4.7 File Format Conversion

Some example scripts to show how to use LandScript to convert between different file formats. This can be useful for batch processing of large numbers of files to convert.

4.7.1 Converting a USGS National Elevation Dataset (NED) file to LandSerf format

In this example, the original file is stored in ArcGIS 'Binary Interleaved Layers' (BIL) format as downloaded from seamless.usgs.gov.

```
# Converts a ArcGIS Binary Interleaved File (.bil) to LandSerf format
version(1.0);

# Change the directory and filename to correspond to your file location.
baseDir = "c:/Program Files/LandSerf/data/";

# Open the BIL file.
dem = open(baseDir & "lincolnNED.bil");

# Add the metadata to the DEM.
edit(dem,"title","Lincoln DEM");
edit(dem,"projection","LatLong");
edit(dem,"ellipsoid","wgs 84");
colouredit(dem,"land3");

# Save the edited raster as a LandSerf file.
save(dem, baseDir & "landscriptExamples/LincolnNED.srf");
```

4.7.2 Creating text files representing objects and their metadata

The function `saveAsText()` can be used as a general purpose function for saving any raster or vector map as a series of text files. These files contain all the spatial data as well as associated metadata such as attribute tables, colour tables and projection information.

```
# Saves a raster and vector map and all their metadata as a collection of text files.
version(1.0);

# Change the directory and filename to correspond to your file location.
baseDir = "c:/Program Files/LandSerf/data/landscriptExamples/";

# Convert a raster and a vector map.
saveAsText("dem","raster");
saveAsText("shapes","vector");


# --------
# Function to save a spatial object as a set of text files.
function saveAsText(baseName, type)
{
    spObj = null();

    if (compare(type,"raster")==0);
    {
```

```
        spObj = open(baseDir&baseName&".srf");
        save(spObj,baseDir&baseName&"_ascii.txt","ArcGridText");
    }
    if (compare(type,"vector")==0);
    {
        spObj = open(baseDir&baseName&".vec");
        save(spObj,baseDir&baseName&"_ascii.txt","TextVector");
    }

    if ((compare(type,"raster")==0) or (compare(type,"vector")==0));
    {
        save(spObj,baseDir&baseName&"_ascii.ctb","Colourtable");
        save(spObj,baseDir&baseName&"_ascii.atr","Attributes");
        metadata = "Title:\t\t"&info(spObj,"title")&"\n"&
                   "Projection:\t"&info(spObj,"projection")&"\n"&
                   "Notes:\t\t"&info(spObj,"notes");
        echo(metadata,baseDir&baseName&"_metadata.txt");
    }
}
```

4.8 Combining the contents of two objects

4.8.1 Adding two rasters

```
1  # Adds the contents of two rasters together.
2  version(1.0);
3
4  # Change the directory below to correspond to your file location.
5  baseDir = "c:/Program Files/LandSerf/data/landscriptExamples/";
6
7  # Open elevation model and vegetation layer.
8  dem = open(baseDir&"dem.srf");
9  veg = open(baseDir&"vegetation.srf");
10
11 # Create combined surface
12 dsm = new(dem);
13 dsm = dem + veg;
14
15 # Add metadata and save combined raster.
16 edit(dsm,"title","Digital Surface Model");
17 edit(dsm,"notes","Combined DEM ground surface and vegetation layer");
18 save(dsm,baseDir&"dsm.srf");
```

4.8.2 Adding two vectors

In this example, the vector maps that are added have their own attribute tables. The numeric attribute that is used when adding values together can be set by changing column of the active attribute (lines 16 and 17). If the active attribute contains non-numeric values, the vector's ID is used instead. After addition, it no longer makes sense for the the new vector map (`output`) to retain its original attribute table, so it is removed (line 13).

```
1  # Adds the contents of two vector maps with attribute tables.
2  version(1.0);
3
4  # Change the directory below to correspond to your file location.
5  baseDir = "c:/Program Files/LandSerf/data/landscriptExamples/";
6
7  # Open a point map and an area map
8  points = open(baseDir&"points.vec");
9  areas  = open(baseDir&"shapes.vec");
10
11 # Use the point map to define the output vector
12 output = new(points,"true");
13 edit(output,"attributes","noTable");
14
15 # Define the attribute table columns to act as input
16 edit(areas,"attCol",3);
17 edit(points,"attCol",2);
18
19 # Add the active attributes of points and area maps
20 output = areas+points;
21
22 # Set metadata and save the combined vector map.
23 colouredit(output,"default");
24 save(output,baseDir&"points2.vec");
```

4.8.3 Adding the contents of a raster to objects in a vector map

The raster (a fractal surface in this example) is queried at the centroid of each vector object in the vector map. Each object then takes the value of that queried raster cell. If the vector object is outside the bounds of the raster, its value is set to `null` and is therefore not displayed.

```
# Adds the contents of a raster map to objects in a vector map
version(1.0);

# Change the directory below to correspond to your file location.
baseDir = "c:/Program Files/LandSerf/data/landscriptExamples/";

# Create a fractal raster map
fracSurf = newraster(400,400,1,1,800,800);
fractal(fracSurf, 2.01);

# Open a point map
points = open(baseDir&"points.vec");

# Set the value of each vector object to the raster cell at its location.
points = fracSurf;
edit(points,"attributes","noTable");

# Set metadata and save the new vector map.
colouredit(points,"default");
save(points,baseDir&"points2.vec");
```

4.9 Tiling Objects

4.9.1 Splitting a raster into 4 quadrants

```
# Splits a raster into 4 equally sized tiles
version(1.0);

# Change the directory below to correspond to your file location.
baseDir = "c:/Program Files/LandSerf/data/landscriptExamples/";

# Open a raster map to split.
raster = open(baseDir&"topoMap.srf");

# Create four new empty rasters.
midEasting = info(raster,"w")+(info(raster,"e")-info(raster,"w"))/2;
midNorthing = info(raster,"s")+(info(raster,"n")-info(raster,"s"))/2;

nw = newraster(info(raster,"w"),midNorthing,
              info(raster,"xRes"),info(raster,"yRes"),
              info(raster,"numRows")/2,info(raster,"numCols")/2);
ne = newraster(midEasting,midNorthing,
              info(raster,"xRes"),info(raster,"yRes"),
              info(raster,"numRows")/2,info(raster,"numCols")/2);
sw = newraster(info(raster,"w"),info(raster,"s"),
              info(raster,"xRes"),info(raster,"yRes"),
              info(raster,"numRows")/2,info(raster,"numCols")/2);
se = newraster(midEasting,info(raster,"s"),
              info(raster,"xRes"),info(raster,"yRes"),
              info(raster,"numRows")/2,info(raster,"numCols")/2);

# Cookie-cut the original raster with the new quadrant rasters.
nw = combine(raster_,nw_,"intersection");
edit(nw,"title","NW quadrant of "&info(raster,"title"));
ne = combine(raster_,ne_,"intersection");
edit(ne,"title","NE quadrant of "&info(raster,"title"));
sw = combine(raster_,sw_,"intersection");
edit(sw,"title","SW quadrant of "&info(raster,"title"));
se = combine(raster_,se_,"intersection");
edit(se,"title","SE quadrant of "&info(raster,"title"));

# Save the new quadrants
save(nw,baseDir&"topoMapNW.srf");
save(ne,baseDir&"topoMapNE.srf");
save(sw,baseDir&"topoMapSW.srf");
save(se,baseDir&"topoMapSE.srf");
```

4.9.2 Combining 4 quadrants into a new raster

```
# Combines 4 tiled rasters into a single raster map.
version(1.0);

# Change the directory below to correspond to your file location.
baseDir = "c:/Program Files/LandSerf/data/landscriptExamples/";

# Start with one of the tiled rasters.
bigRaster = open(baseDir&"topoMapNW.srf");

# Combine each of the tiled rasters with the main raster.
tile = open(baseDir&"topoMapNE.srf");
```

```
bigRaster = combine(bigRaster_,tile_);

tile = open(baseDir&"topoMapSE.srf");
bigRaster = combine(bigRaster_,tile_);

tile = open(baseDir&"topoMapSW.srf");
bigRaster = combine(bigRaster_,tile_);

# Save the newly combined raster
save(bigRaster,baseDir&"topoMapCombined.srf");
```

4.10 Transforming Between Elevation Models

4.10.1 Contouring a raster DEM

```
# Contours a DEM with a 5m vertical interval at the finest resolution of the raster.
version(1.0);

# Change the directory below to correspond to your file location.
baseDir = "c:/Program Files/LandSerf/data/landscriptExamples/";

# Open a surface raster and contour it at 50m vertical intervals.
dem = open(baseDir&"dem.srf");

minHeight = round(info(dem,"min")/10)*10;
contours = contour(dem,minHeight,50,1);

# Colour all contours brown and add metadata
colouredit(contours,"rules","0 50 10 0");
edit(contours,"notes","Contours through "&info(dem,"title")&" at 50m vertical interval");

# Save the new contour vector map.
save(contours,baseDir&"contours.vec");
```

4.10.2 Creating a TIN from a DEM

```
# Creates a TIN from a DEM with average error of 5% of DEM's standard deviation
version(1.0);

# Change the directory below to correspond to your file location.
baseDir = "c:/Program Files/LandSerf/data/landscriptExamples/";

# Open a surface raster.
dem = open(baseDir&"dem.srf");

# Used to define precision of triangulation
avError = round(info(dem,"stdev")*0.05);

# Perform the triangulation
tin = triangulate(dem,"null",avError,"null");

# Save the new TIN as a vector map.
edit(tin,"notes","TIN derived from "&info(dem,"title")&" with max error of "&avError);
save(tin,baseDir&"tin.vec");
```

4.11 Transforming Raster Values

4.11.1 Rescaling raster values to range 0-255

```
# Rescales a raster so all values are integers between 0-255
version(1.0);

# Change the directory below to correspond to your file location.
baseDir = "c:/Program Files/LandSerf/data/landscriptExamples/";

# Open the raster
dem = open(baseDir&"dem.srf");

# Create rescaled raster.
rescaledDEM = new(dem);
minZ = info(dem,"min");
maxZ = info(dem,"max");
range = maxZ-minZ;
rescaledDEM = round(255*(dem-minZ)/range);

# Give rescaled raster a grey scale and save it.
colouredit(rescaledDEM,"rules","0 0, 255 255");
save(rescaledDEM,baseDir&"rescaledDEM.srf");
```

4.11.2 Growing raster cells

This example uses a series of nested focal operations to grow the boundaries of non-null areas in a raster by one pixel. The newly grown cells are saved in their own raster (`coastline`).

```
# Creates some fractal islands, then grows the coastlines by 1 pixel.
version(1.0);

# Change the directory below to correspond to your file location.
baseDir = "c:/Program Files/LandSerf/data/landscriptExamples/";

# Create empty raster.
fracSurf = newraster(1000,1000,10,10,250,250,"Fractal");

# Fill raster with fractal surface.
fractal(fracSurf,"2.01");
islands = new(fracSurf);

# Use data range to get 'sea level';
midPoint = info(fracSurf,"median");

# Flood all values below 'sea level'
fracSurf = ifelse(fracSurf < midPoint, null(), fracSurf);

# Grow coastline
islands = ifelse(fracSurf != null(), fracSurf,
            ifelse(fracSurf[0,-1] != null(), fracSurf[0,-1],
               ifelse(fracSurf[0,1] != null(), fracSurf[0,1],
                  ifelse(fracSurf[-1,0] != null(), fracSurf[-1,0],
                     ifelse(fracSurf[1,0] != null(), fracSurf[1,0],null())))));

# Find coastline by taking difference between original and grown islands.
coastline = new(fracSurf);
coastline = islands-fracSurf;
```

```
30
31 # Save original and expanded islands.
32 colouredit(islands,"land1");
33 edit(islands,"title","Fractal islands");
34 save(islands,baseDir & "islands.srf");
35
36 colouredit(coastline,"rules","0 255, 1 180 180 0");
37 edit(coastline,"title","Fractal coastline");
38 save(coastline,baseDir & "coastline.srf");
```

4.12 Drawing Vector Maps

4.12.1 Creating an image from a vector map

By default, only point features can display labels. Therefore to show the labels associated with area or line objects, first find their centroids and store these new point values in a separate vector map. The `draw()` command can then draw both the original vector map (primary) and the labels vector map (secondary).

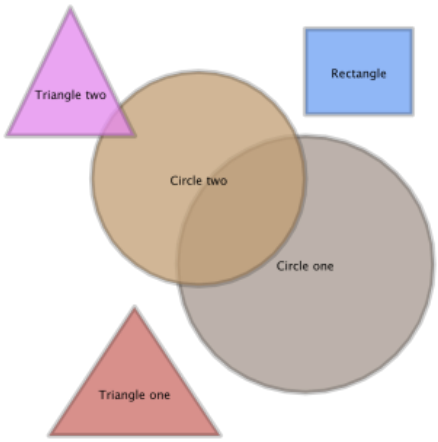


Figure 4.3: shapes.png.

```
# Draws a vector map of area objects with labels.
version(1.0);

# Change the directory below to correspond to your file location.
baseDir = "c:/Program Files/LandSerf/data/landscriptExamples/";

# Open vector map containing area objects.
vectMap = open(baseDir&"shapes.vec");

# Create a new vector map containing shape centroids to use as labels
centroids = tocentroids(vectMap);

# Add a small border (1% of width) around vector map for display
border = (info(vectMap,"e")-info(vectMap,"w"))*0.01;
edit(vectMap,"s",info(vectMap,"s")-border);
edit(vectMap,"n",info(vectMap,"n")+border);
edit(vectMap,"w",info(vectMap,"w")-border);
edit(vectMap,"e",info(vectMap,"e")+border);

# Set drawing styles
vectorstyle("pointsize",0.1);
vectorstyle("showlabels","true");
vectorstyle("labelbackground","0 0 0 0");
vectorstyle("labelsize",28);
vectorstyle("labelposition","centre");
vectorstyle("linewidth",12);
vectorstyle("boundarycolour","0 0 0 50");
vectorstyle("polygonopacity",0.75);

# Draw the vector map with labels and save to file
draw(baseDir&"shapes.png","null","null",vectMap,centroids);
```

4.12.2 Creating a high quality SVG image from a vector map

SVG (Scalable Vector Graphics) images preserve the high quality appearance of vector maps. Similar to the previous example, but this time, the centroid labels are combined with the original vector map so that a single map can be saved in SVG format.

```
# Saves a vector map of area objects with labels as an SVG file.
version(1.0);

# Change the directory below to correspond to your file location.
baseDir = "c:/Program Files/LandSerf/data/landscriptExamples/";

# Open vector map containing area objects.
vectMap = open(baseDir&"shapes.vec");

# Create a new vector map containing shape centroids to use as labels
centroids = tocentroids(vectMap);

# Combine the two vector maps
vectMap = combine(vectMap_,centroids_);

# Add a small border (1% of width) around vector map for display
border = (info(vectMap,"e")-info(vectMap,"w"))*0.01;
edit(vectMap,"s",info(vectMap,"s")-border);
edit(vectMap,"n",info(vectMap,"n")+border);
edit(vectMap,"w",info(vectMap,"w")-border);
edit(vectMap,"e",info(vectMap,"e")+border);

# Set drawing styles
vectorstyle("pointsize",0.1);
vectorstyle("showlabels","true");
vectorstyle("labelbackground","0 0 0 0");
vectorstyle("labelsize",14);
vectorstyle("labelposition","centre");
vectorstyle("linewidth",8);
vectorstyle("boundarycolour","255 255 255 50");
vectorstyle("polygonopacity",0.75);

# Save the combined map as an SVG file
save(vectMap,baseDir&"shapes.svg","svg");
```

4.13 Cartographic Shaded Relief

4.13.1 Banded hypsometric tinting

This example colours elevation in 7 discrete colour bands with a fine contour line between each band.

```
# Draws a shaded relief map with banded hypsometric colours and contours
version(1.0);

# Change the directory below to correspond to your file location.
baseDir = "c:/Program Files/LandSerf/data/landscriptExamples/";

# Open raster DEM and create seven elevation bands.
dem = open(baseDir&"dem.srf");
minHeight = info(dem,"min");
range = info(dem,"max")-minHeight;
inc = range/7;

# Provide a discrete colour for each band.
colouredit(dem,"rules", minHeight &   " 141 166 141 (D)," &
                       minHeight+ 1*inc & " 172 194 155 (D)," &
                       minHeight+ 2*inc & " 221 219 167 (D)," &
                       minHeight+ 3*inc & " 254 235 181 (D)," &
                       minHeight+ 4*inc & " 248 212 153 (D)," &
                       minHeight+ 5*inc & " 241 170 109 (D)," &
                       minHeight+ 6*inc & " 227 112 72 (D)");

# Create contours at boundries of the bands.
contours = contour(dem,minHeight+(inc/2),inc,1);
colouredit(contours,"rules","0 50 10 0");
vectorstyle("linewidth",0.1);

# Combine contours, relief and hypsometric tinting.
draw(baseDir&"dem.png",dem,dem,contours,"null","relief");
```

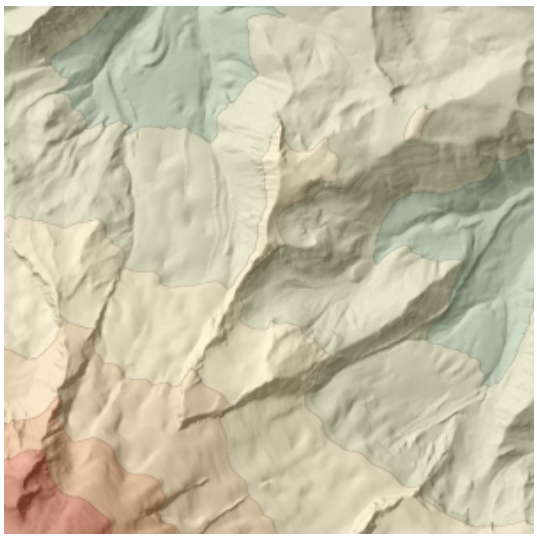


Figure 4.4: dem.png.

4.13.2 Cold and warm shaded relief

This script uses Phong shaded relief to create two relief maps of a DEM with a blue (cold) light slightly clockwise of NW and a yellow (warm) light slightly anticlockwise. The 'warm' relief is calculated at a coarse scale to emphasise the larger scale features in the DEM (lines 22-30). The two images are then blended and saved as file (line 33).

Contours are threaded through the DEM at 100m vertical interval (line 37) and coloured a transparent brown (line38).

Finally, the relief map is blended with the original coloured DEM with the contours overlaid to produce the final output.

```
1  # Draws cartographic shaded relief map of a DEM with contours.
2  version(1.0);
3
4  # Change the directory below to correspond to your file location.
5  baseDir = "c:/Program Files/LandSerf/data/landscriptExamples/";
6
7  # Open raster DEM.
8  dem = open(baseDir&"dem.srf");
9
10 # Set phong shading parameters
11 edit(dem,"specular",70);
12 edit(dem,"diffuse",50);
13 edit(dem,"ambient",30);
14
15 # Sun shining between W and NW
16 # with a 30 deg. divergence between warm and cold light directions.
17 sunAzim = 292.5;
18 sunDeviation = 30;
19 coldSize = 3;
20 warmSize = 25;
21
22 # Cold light clockwise of main sun azimuth
23 edit(dem,"sunAzim",(sunAzim+sunDeviation)%360);
24 relief1 = surfparam(dem,"shade",coldSize,1);
25 colouredit(relief1,"rules","0 0 0 0, 1 127 127 255");
26
27 # Warm light anticlockwise of main sun azimuth
28 edit(dem,"sunAzim",(sunAzim-sunDeviation)%360);
29 relief2 = surfparam(dem,"shade",warmSize,1);
30 colouredit(relief2,"rules","0 0 0 0, 1 255 255 127");
31
32 # Blend the warm and cold images
33 draw(baseDir&"relief.png",relief1,relief2,"null","null","blend",50);
34
35 # Create contours from original DEM
36 minHeight = round(info(dem,"min")/10)*10;
37 contours = contour(dem,minHeight,100,1);
38 colouredit(contours,"rules","0 50 10 0 100");
39 vectorstyle("linewidth",0.1);
40
41 # Combine contours, relief and hypsometric tinting.
42 relief = open(baseDir&"relief.png","image");
43 draw(baseDir&"dem.png",dem,relief,contours,"null","blend",30);
```

Figure 4.5: `relief.png` and `dem.png`.

4.14 Processing Colour Components

4.14.1 Splitting a raster into 3 colour components

```
# Extracts red, green and blue components from a raster
version(1.0);

# Change the directory below to correspond to your file location.
baseDir = "c:/Program Files/LandSerf/data/landscriptExamples/";

# Open image file.
image = open(baseDir&"photo.srf");

# Create a new raster to store the colour components
colourComponent = new(image);
edit(colourComponent,"type","other");

# Extract the three colour components and save the rasters.
colourComponent = red(image);
colouredit(colourComponent,"rules","0 0, 255 255 0 0");
save(colourComponent,baseDir&"redImage.srf");

colourComponent = green(image);
colouredit(colourComponent,"rules","0 0, 255 0 255 0");
save(colourComponent,baseDir&"greenImage.srf");

colourComponent = blue(image);
colouredit(colourComponent,"rules","0 0, 255 0 0 255");
save(colourComponent,baseDir&"blueImage.srf");
```

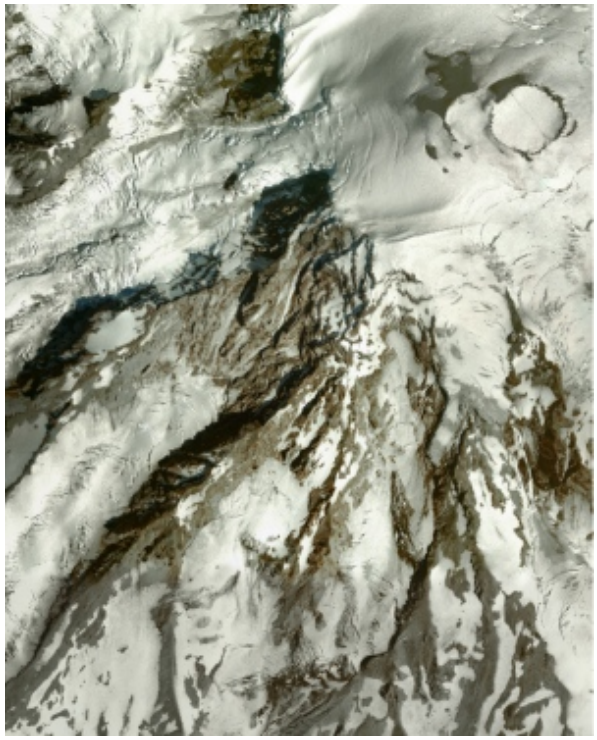


Figure 4.6: `photo.srf`.

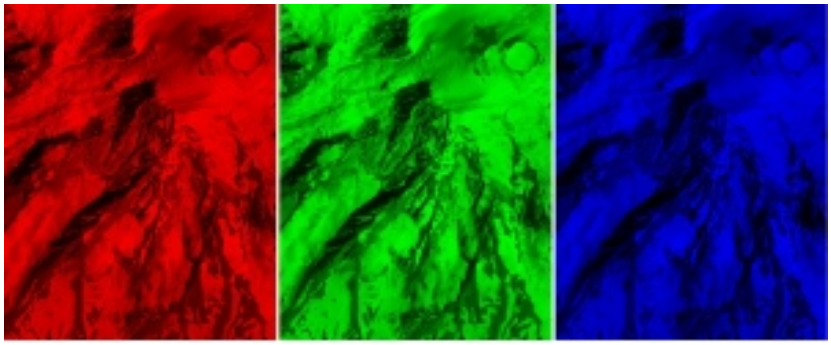


Figure 4.7: `redImage.srf`, `greenImage.srf`, `blueImage.srf`.

4.14.2 Creating a colour composite

```
# Combines three colour component rasters to create a single colour composite.
version(1.0);

# Change the directory below to correspond to your file location.
baseDir = "c:/Program Files/LandSerf/data/landscriptExamples/";

# Open image file.
redImage = open(baseDir&"redImage.srf");
grnImage = open(baseDir&"greenImage.srf");
bluImage = open(baseDir&"blueImage.srf");

# Create and save a new raster storing colour composite
composite = new(redImage);
edit(composite,"type","image");
colouredit(composite,"raw");

composite = rgb(redImage,grnImage,bluImage);
save(composite,baseDir&"compositeImage.srf");
```

4.14.3 Converting a colour raster to greyscale

```
# Creates a greyscale version of a raster using the formula
# greyLev = 0.3R + 0.59G + 0.11B
version(1.0);

# Change the directory below to correspond to your file location.
baseDir = "c:/Program Files/LandSerf/data/landscriptExamples/";

# Open image file.
image = open(baseDir&"photo.srf");

# Create a new raster to store the grey level values
greyImage = new(image);
edit(greyImage,"type","other");
colouredit(greyImage,"rules","0 0, 255 255");

# Do the conversion and save the raster.
greyImage = 0.30*red(image) + 0.59*green(image) + 0.11*blue(image);
save(greyImage,baseDir&"greyImage.srf");
```

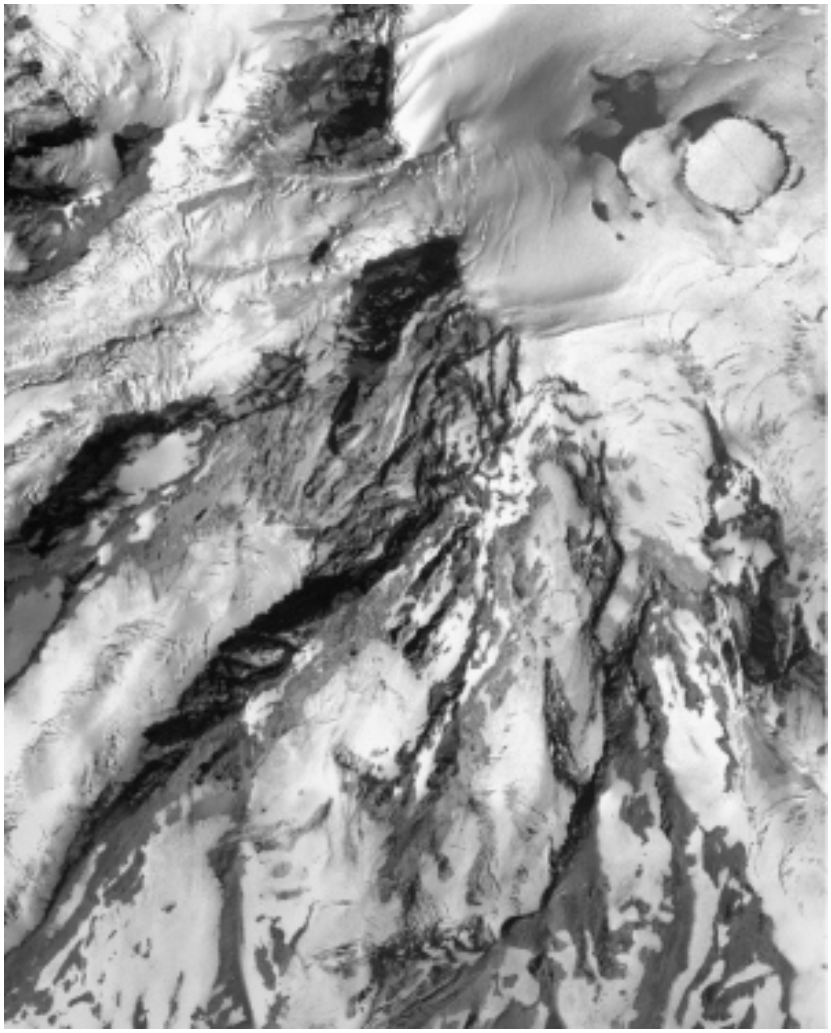


Figure 4.8: `greyImage.srf`.

4.14.4 Extracting green pixels from a scanned raster map

This example looks for greenish pixels by comparing the R,G and B values for each raster cell. If a pixel is found to be green it is labeled as vegetation (1), otherwise it is labeled as white background (0).

```
# Extracts green components from an image
version(1.0);

# Change the directory below to correspond to your file location.
baseDir = "c:/Program Files/LandSerf/data/landscriptExamples/";

# Open image file.
image = open(baseDir&"topoMap.srf");

# Extract the greenish pixels
image = ifelse((green(image) > blue(image)) and
             (green(image) > red(image)) and
             (green(image)/(red(image)+blue(image)) > 0.6),1,0);
```

```
14
15 # Pass a 5x5 median filter over surface to remove isolated patches.
16 veg = new(image);
17 veg = median(image[-2,-2],image[-2,-1],image[-2,0],image[-2,1],image[-2,2],
18             image[-1,-2],image[-1,-1],image[-1,0],image[-1,1],image[-1,2],
19             image[ 0,-2],image[ 0,-1],image[ 0,0],image[ 0,1],image[ 0,2],
20             image[ 1,-2],image[ 1,-1],image[ 1,0],image[ 1,1],image[ 1,2],
21             image[ 2,-2],image[ 2,-1],image[ 2,0],image[ 2,1],image[ 2,2]);
22
23 # Add metadata and save the green raster.
24 colouredit(veg,"rules","0 255 255 255, 1 0 200 0");
25 edit(veg,"type","other");
26 save(veg,baseDir&"veg.srf");
```

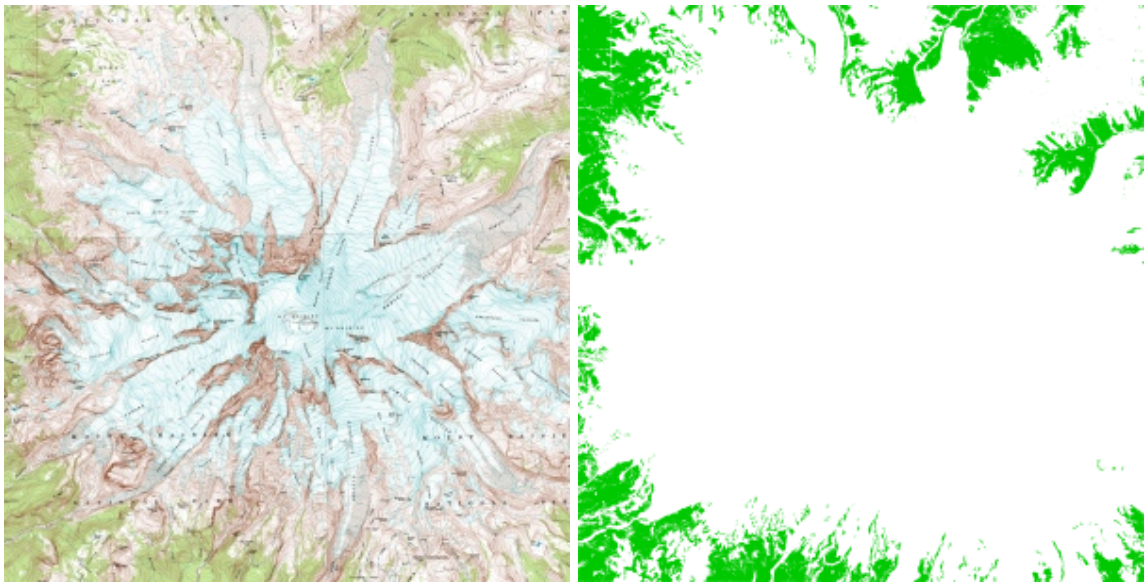


Figure 4.9: `topoMap.srf` (left) and `veg.srf` (right).

4.15 Image Filtering

The following examples assume the input raster contains numeric greyscale values. If a raw colour image is used, the filters below will need to be applied separately to the red, green and blue components before recombining in a composite image.

4.15.1 A 3x3 mean filter

```
# Passes a 3x3 mean filter over a raster.
version(1.0);

# Change the directory below to correspond to your file location.
baseDir = "/Users/jwo/landscript/examples/data/";

# Open image file.
image = open(baseDir&"greyImage.srf");
filteredImage = new(image);

# Do the filtering and save the raster.
filteredImage = (image[-1,-1] + image[-1,0] + image[1,0] +
                 image[-1,0] + image[0,0] + image[1,0] +
                 image[1,-1] + image[1,0] + image[1,1])/9;
save(filteredImage,baseDir&"filteredImage.srf");
```

4.15.2 A 5x5 median filter

```
# Passes a 5x5 median filter over a raster. Useful for removing noise from images.
version(1.0);

# Change the directory below to correspond to your file location.
baseDir = "/Users/jwo/landscript/examples/data/";

# Open image file.
image = open(baseDir&"greyImage.srf");
filteredImage = new(image);

# Do the filtering and save the raster.
filteredImage = median(image[-2,-2],image[-2,-1],image[-2,0],image[-2,1],image[-2,2],
                       image[-1,-2],image[-1,-1],image[-1,0],image[-1,1],image[-1,2],
                       image[ 0,-2],image[ 0,-1],image[ 0,0],image[ 0,1],image[ 0,2],
                       image[ 1,-2],image[ 1,-1],image[ 1,0],image[ 1,1],image[ 1,2],
                       image[ 2,-2],image[ 2,-1],image[ 2,0],image[ 2,1],image[ 2,2]);
save(filteredImage,baseDir&"filteredImage.srf");
```

4.15.3 A 3x3 edge detection filter

```
# Passes a 3x3 edge detection filter over a raster.
version(1.0);

# Change the directory below to correspond to your file location.
baseDir = "/Users/jwo/landscript/examples/data/";

# Open image file.
image = open(baseDir&"greyImage.srf");
```

```
filteredImage = new(image);

# Do the filtering and save the raster.
filteredImage = abs(-image[-1,-1] - image[-1,0] - image[1,0]
                  -image[-1,0] +8*image[0,0] - image[1,0]
                  -image[1,-1] - image[1,0] - image[1,1]);

filteredImage = ifelse(filteredImage >255,255,filteredImage);

save(filteredImage,baseDir&"filteredImage.srf");
```

4.16 Detecting Peaks and Summits

The following two examples show approaches to feature detection from a surface.

4.16.1 Detecting fuzzy peak membership using morphometry

The example below mimics LandSerf's 'Fuzzy Feature Classification' by identifying peak features over a range of scales. It has slightly greater flexibility than LandSerf's menu option in that both the minimum and maximum window sizes can be specified. It would also be possible to modify this code in line 28 so that a window sizes were selected in a non linear way (e.g. by doubling the window size on each iteration through the loop.

```
1
2  # Script to find fuzzy feature memberships.
3  version(1.0);
4
5  # Change the directory below to correspond to your file location.
6  baseDir = "c:/Program Files/LandSerf/data/landscriptExamples/";
7
8  # Change the values below to define the feature to explore
9  # and the scale range over which that feature is to be extracted.
10 featureName   = "peak";
11 decayExp    = 0.0;
12 slopeTol    = 20;
13 curveTol    = 0.2;
14 minWinSize   = 5;
15 maxWinSize   = 19;
16
17 dem = open(baseDir & "dem.srf");
18 fuzzyFeature = new(dem);
19 winSize = minWinSize;
20 numScales = 0;
21 featureID = getFeatureID(featureName);
22
23 while (winSize <= maxWinSize);
24 {
25     echo("Processing with "&winSize&"x"&winSize&" window.");
26     numScales = numScales+1;
27     features = surfparam(dem_,"feature",winSize,decayExp,slopeTol,curveTol);
28     fuzzyFeature = ifelse(features==featureID,fuzzyFeature+1,fuzzyFeature);
29     winSize = winSize + 2;
30 }
31
32 fuzzyFeature = fuzzyFeature/numScales;
33
34 # Add metadata and save
35 edit(fuzzyFeature,"title","Fuzzy "&featureName&" membership");
36 if (featureID == 1); { colouredit(fuzzyFeature,"rules","0 255 255 255, 1 0 0 0");}
37 if (featureID == 2); { colouredit(fuzzyFeature,"rules","0 255 255 255, 1 0 0 200");}
38 if (featureID == 3); { colouredit(fuzzyFeature,"rules","0 255 255 255, 1 0 150 0");}
39 if (featureID == 4); { colouredit(fuzzyFeature,"rules","0 255 255 255, 1 250 250 0");}
40 if (featureID == 5); { colouredit(fuzzyFeature,"rules","0 255 255 255, 1 200 0 0");}
41 if (featureID == 6); { colouredit(fuzzyFeature,"rules","0 255 255 255, 1 200 200 200");}
42
43 save(fuzzyFeature,baseDir&"fuzzy"&featureName&".srf");
44
45
46 # --------
```

```
# Function to report the numeric feature id corresponding to the given feature.
function getFeatureID(featName)
{
    if (compare(featName,"pit") == 0);     {return 1;}
    if (compare(featName,"channel") == 0); {return 2;}
    if (compare(featName,"pass") == 0);    {return 3;}
    if (compare(featName,"ridge") == 0); {return 4;}
    if (compare(featName,"peak") == 0);    {return 5;}
    if (compare(featName,"planar") == 0); {return 6;}

    echo("Unknown feature type '"&featName&"'. Assuming 'planar'.");
    return 6;
}
```

4.16.2 Detecting peaks using relative drop

```
# Identifies peaks and summits on a DEM
version(1.0);

# Change the directory below to correspond to your file location.
baseDir = "c:/Program Files/LandSerf/data/landscriptExamples/";

# Open raster DEM.
dem = open(baseDir&"dem.srf");

# Perform the peak identification.
fuzzyPeaks = new(dem);
summits = newvector(0,0,0,0);
hierarchy = new(dem);
peaks = peakclass(dem,50,"null",fuzzyPeaks,summits,hierarchy);

# Save the new peak and summit maps.
save(peaks,baseDir&"peaks.srf");
save(fuzzyPeaks,baseDir&"fuzzyPeaks.srf");
save(summits,baseDir&"summits.vec");
save(hierarchy,baseDir&"peakHierarchy.srf");
```

4.17 Measuring Characteristic Scales

4.17.1 Finding scales of most extreme parameter values

This example measures the given parameter (e.g. slope, profile curvature etc.) over a range of scales, and finds the scale at which that parameter is most extreme (either positive or negative). Output consists of two rasters, one containing the measured parameter, the other the window size at which the parameter is most extreme. Can be used to explore scale sensitivity of a surface.

```
# Script to measure surface parameter at characteristic scales.
version(1.0);

# Change the directory below to correspond to your file location.
baseDir = "c:/Program Files/LandSerf/data/landscriptExamples/";

# Change the three values below to define the parameter to explore and the
# scale range over which that parameter is to be measured.
param = "planc";
maxWinSize = 35;
minWinSize = 3;

# Open the dem to analyse and create the rasters to store parameter and scale.
dem = open(baseDir & "dem.srf");
scaleMax = new(dem);
scaleMax = minWinSize;
maxParamSurf = surfparam(dem,param,minWinSize,1.0);
winSize = minWinSize+2;
maxParam = 0;

# Loop through each window size comparing parameter with previous extreme values.
while (winSize <= maxWinSize);
{
    echo("Calculating "&param&" using "&winSize&"x"&winSize&" window.");
    paramSurf = surfparam(dem_, param, winSize, 1.0);
    scaleMax = ifelse(abs(paramSurf) > abs(maxParamSurf), winSize, scaleMax);
    maxParamSurf=ifelse(abs(paramSurf) > abs(maxParamSurf), paramSurf, maxParamSurf);
    winSize = winSize + 2;
}

# Add metadata to the two new rasters.
colouredit(scaleMax,"rules",minWinSize&" 255 255 255, "& maxWinSize&" 0 0 0");
edit(scaleMax,"title",param&" characteristic scale");
edit(scaleMax,"notes","Window size at which "&param&" values are most extreme.");

edit(maxParamSurf,"title","Extreme "&param);
edit(maxParamSurf,"notes",param&" calculated with window sizes from "&
                        minWinSize&" to "&maxWinSize&". Contains the largest "&
                        "absolute value of the parameter over the range of scales.");

# Save the two new rasters
save(maxParamSurf,baseDir&"max"&param&".srf");
save(scaleMax,baseDir&"charScale"&param&".srf");
```

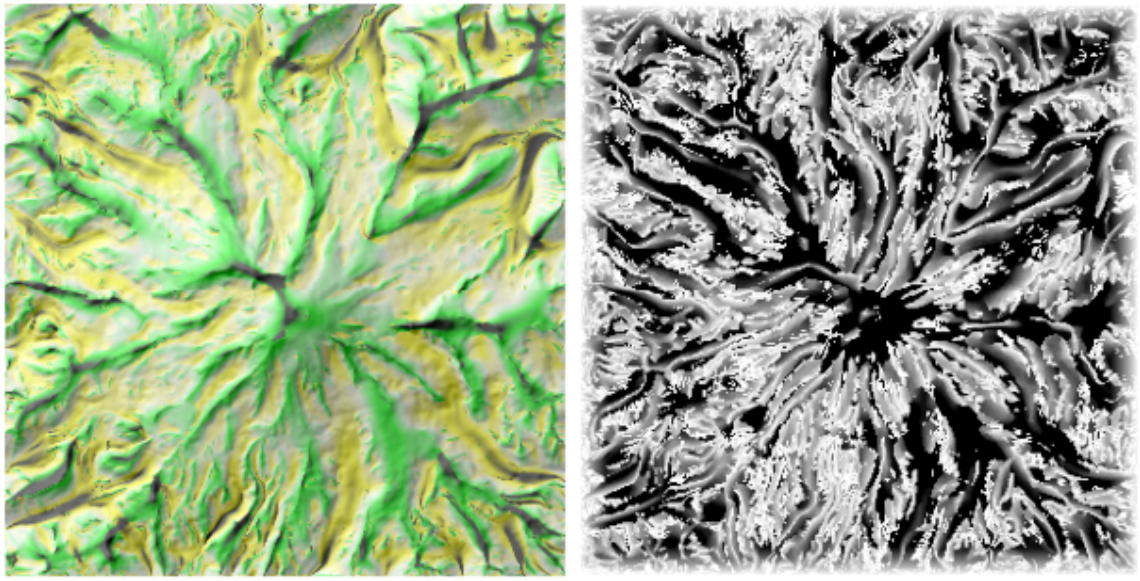


Figure 4.10: `maxplanc.srf` (left) and `charScaleplanc.srf` (right).

5 Programming with Java

5.1 Introduction to Java Programming with LandSerf

This chapter covers four aspects of Java programming using the LandSerf libraries and API.

- Getting Started
  Linking your Java classes with the LandSerf libraries; compiling and running a 'Hello World' application; creating spatial object metadata from within a Java program.

- Manipulating raster maps
  Creating new raster maps; creating and editing raster metadata; processing raster values.

- Manipulating vector maps
  Handling vector metadata; how vector maps are organised; creating and processing vector maps.

- Using the `jwo.landserf.process` classes
  Storing spatial objects in a `GISFrame`; handling threaded processes; where input and output is stored.

To follow this guide and program with the LandSerf classes you will need:

- LandSerf 2.3 installed on your computer;

- the Java 1.4 SE (or higher) Software Development Kit installed on your computer;

- access to the LandSerf API documentation

You may also find it useful to have available:

- An *Integrated Development Environment* such as Eclipse;

- the book Java Programming for Spatial Sciences, which covers much of the background and design behind the development of the LandSerf classes;

- the LandSerf discussion forum covering programming issues.

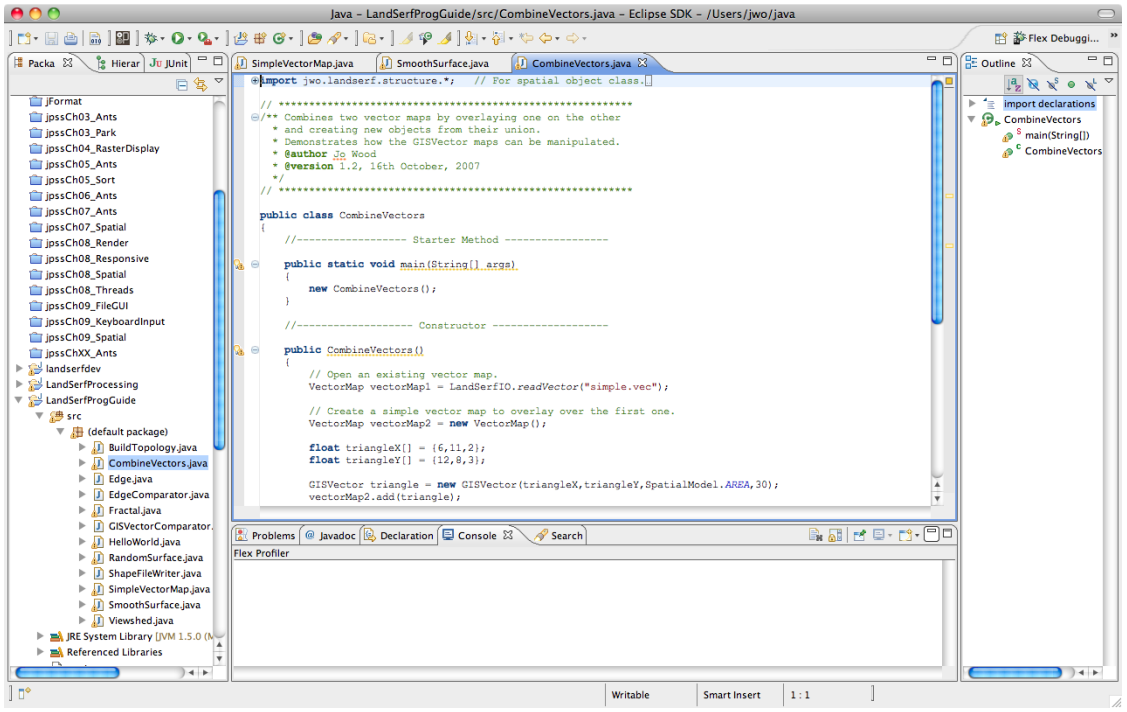


Figure 5.1: Sample Java code in the Eclipse environment.

5.2 Getting Started

By the end of this section you will be able to:

- Link your Java programming environment to the relevant LandSerf class libraries.
- Compile a class to create and save a simple raster with its own metadata.
- Use the LandSerf API documentation to explore the LandSerf Java class libraries.
- Create a standalone script to run your Java class.

5.2.1 Linking your Java code with the LandSerf Class Libraries

In order to take advantage of the several hundred classes that make up the LandSerf class libraries, you will first need to identify their location in your Java development environment. The libraries are stored in two *.jar* files in the directory in which you installed LandSerf. `landserf230.jar` contains LandSerf-specific classes and `utils230.jar` contains a set of more general utility classes.

If you are compiling your own Java code from the command line, you will need to point to these two jar files by using the `-classpath` option of `javac`. For example, in Windows you might use the following:

```
javac -classpath .;"c:\Program Files\LandSerf\landserf230.jar";
"c:\Program Files\LandSerf\utils230.jar" myClass.java
```

In Linux, the equivalent would be the following:

```
javac -classpath .:~/LandSerf/landserf230.jar:~/LandSerf/utils230.jar
myClass.java
```

and in MacOSX:

```
javac -classpath
.:/Applications/LandSerf/LandSerf.app/Contents/Resources/Java/landserf230.jar:
/Applications/LandSerf/LandSerf.app/Contents/Resources/Java/utils230.jar
myClass.java
```

If you are using an IDE, you will need to make an equivalent link with the two jar files. For example, in Eclipse, create a new project and select `Properties`→`Java Build Path`→`Libraries` and then click the `Add External JARs...` button.

5.2.2 Creating A 'Hello World' Raster Map

In time honoured fashion, we will start by writting a simple 'hello world' program that will create a simple LandSerf Raster Map, add some metadata, and save it to disk. Consider the following code:

```
1  import jwo.landserf.structure.*; // For spatial object class.
2  import jwo.landserf.process.io.*; // For file saving.
3
4  // *********************************************************
5  /** Creates a simple raster map and with metadata.
6    * @author Jo Wood
7    * @version 1.1, 15th October, 2007
8    */
9  // *********************************************************
10 public class HelloWorld
11 {
12     //------------------ Starter Method -----------------
13
14     public static void main(String[] args)
15     {
16         new HelloWorld();
17     }
18
19     //------------------- Constructor -------------------
20
21     public HelloWorld()
22     {
23         // Create a blank raster of 100 x 100 cells.
24         RasterMap raster = new RasterMap(100,100);
25
26         // Add some simple metadata
27         Header header = new Header("My first raster");
28         header.setNotes("Hello world");
29         raster.setHeader(header);
30         raster.setDefaultColours();
31
32         // Write new raster to file.
33         LandSerfIO.write(raster,"hello.srf");
34     }
35 }
```

Most of the spatial data structures such as raster maps, vector maps, TINs etc. are represented by classes in the `jwo.landserf.structure` package. Therefore to use them, this package is `imported` in line 1 of the code. Since this first example will save a raster map as a file, we also need to import the `jwo.landserf.process.io` package that contains most of the file import/export functionality.

The constructor (line 24) creates an empty raster map containing 100 rows and 100 columns of data. The RasterMap API documentation provides details of other ways to construct raster maps.

The next four lines of code (27-30) create some simple metadata to attach to the raster map. Most metadata are stored in the jwo.landserf.structure.Header class. This class, which is attached to all SpatialObjects, allows a title, author, copyright information and supplementary notes to be associated with the raster. In this example, just the title (set via its constructor) and notes are set. Finally, a default ColourTable is attached to the raster map.

The last line of the `HelloWorld` constructor (line 33) saves the raster we have just created to disk with the name `hello.srf`. Most operations for reading and writing LandSerf files are represented as static methods in the classes in the jwo.landserf.process.io package. Methods for reading and writing in the default LandSerf file format are found in the LandSerfIO class in that package.

5.2.3 Running the 'HelloWorld' Class

To run a compiled class simply call the java interpreter making sure the relevant jar files are linked to your code. This may be achieved directly from with your IDE, or by setting your machine's `CLASSPATH` variable, or by explicitly linking the jar files from the command line:

Windows:
```
java -classpath .;"c:Program Files\LandSerf\landserf230.jar" HelloWorld
```

Linux:
```
java -classpath .:~/LandSerf/landserf230.jar HelloWorld
```

MacOSX:
```
java -classpath
.:/Applications/LandSerf/LandSerf.app/Contents/Resources/Java/landserf230.jar
HelloWorld
```

The command line options can be used to create a simple `.bat` (*Windows*) or `.sh` (Unix/Linux/MacOSX) file that will allow your program to be conveniently run without an IDE.

You can test the output of the program by starting LandSerf and opening the file `hello.srf`. Selecting `Info`→`Summary Info` should produce output similar to Figure 5.2.

Exercises

1. If you have not done so already, create a `HelloWorld` project in your favourite Java IDE (or use the command line if you prefer). Create a new class `HelloWorld` and copy the code above into it. After making sure you have linked your code to the `landserf230.jar` file, compile and run the class. Check that it has behaved as expected by starting LandSerf and loading the new raster file `hello.srf`.

2. Using the API documentation as a guide, try adapting the code to add the following metadata to the raster:

   Resolution:50
   Northern Boundary:10000
   Southern Boundary:5000
   Western Boundary:20000
   Eastern Boundary:25000
   Raster type:*Other*

   Check your results by examining the metadata within LandSerf.

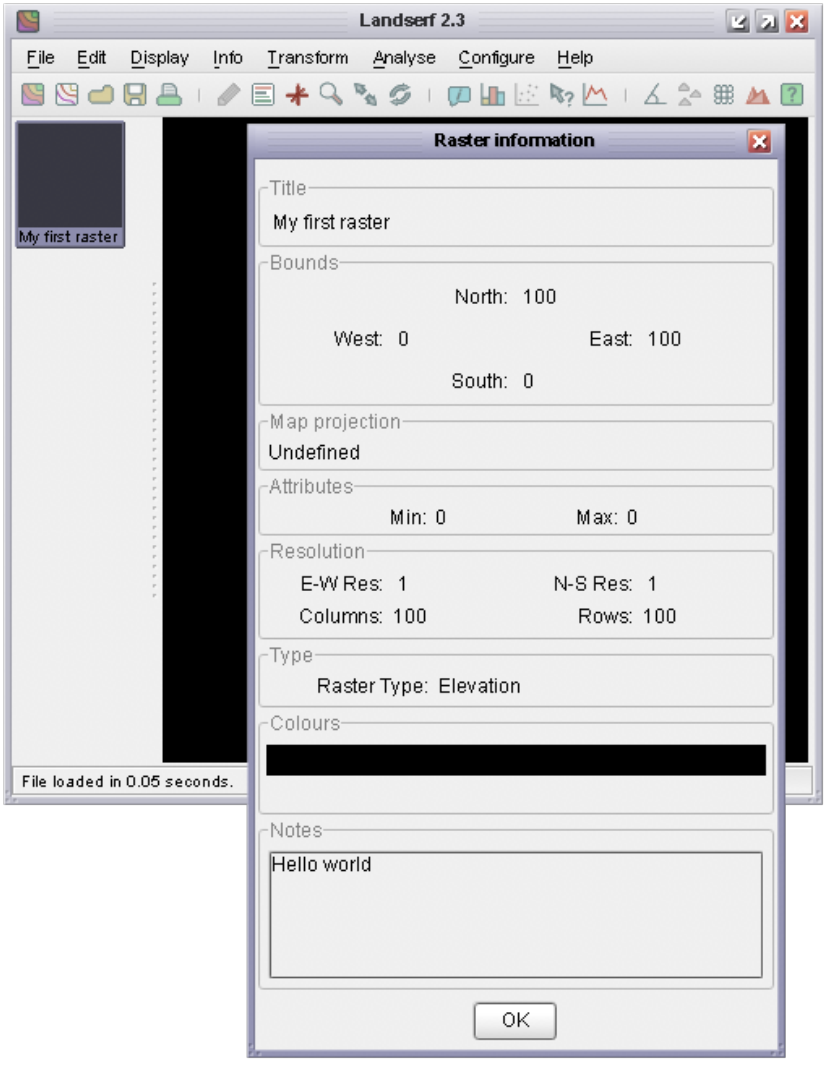


Figure 5.2: Metadata from simple raster displayed in LandSerf.

5.3 Manipulating Raster Maps

By the end of this section you will be able to:

- Create a raster map of arbitrary dimensions, resolution and location.
- Read individual cell values from a raster.
- Write individual cell values to a raster.
- Extract metadata from a raster.
- Set the colour table for a raster.

5.3.1 Creating Raster Maps

As you will have seen in the previous section, creating a new raster map is as straightforward as calling one of the `RasterMap` constructors. Each raster map will have a fixed number of rows and columns, some metadata (title, map projection, notes etc.), a location and resolution. Depending on how much of this information is known at compile-time, one of 7 different constructors can be called.

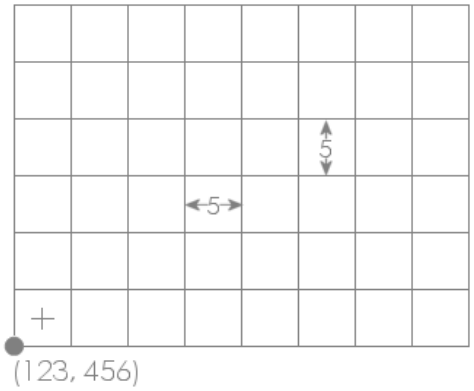


Figure 5.3: Example raster location, resolution and dimensions.

If you know the spatial properties of the raster you are about to create (location, x and y resolution and number of rows/columns), the most useful constructor is probably `RasterMap(int numRows, int numCols, Footprint fp)`. The `Footprint` class is a general one for storing the rectangular size and location of any spatial object. Its own constructor can take the coordinates of its bottom left corner, its width and height (see Chapter 5 of Java Programming for Spatial Sciences for more details of this class). When used as part of the `RasterMap` constructor, this footprint is used to define the location and size of the bottom left pixel in the raster. Thus it defines both the location of the entire raster as well as its resolution. For example, to create a raster map with the properties shown in the figure to the left, you would call the following constructor:

```
new RasterMap(6,8, new Footprint(123,456,5,5)
```

This implicitly sets the top-right location to (163,486) (8 columns of 5 units = 40 and 6 rows of 5 units = 30).

Note that the location set by the origin is of the *outer* boundary of the bottom-left raster cell. The centre of that cell would be equal to the origin plus half the resolution in x and y directions (125.5, 458.5 in this example).

Other useful `RasterMap` constructors include two that take other spatial objects as arguments. `RasterMap(RasterMap oldRaster)` will create a new raster that is a copy (clone) of `oldRaster`. This can be useful if you wish to create a new raster map that will have identical spatial properties to an existing one, but perhaps modified contents. In a similar way you can create a new raster map based on the spatial properties (but not contents) of a vector map (`RasterMap(VectorMap oldVectorMap)`).

5.3.2 Manipulating Raster Cells

Reading and writing the contents of individual raster cells is very straightforward once you have created a `RasterMap` object. Cells can either be addressed by their row and column values (where row 0, column 0 is the top left cell), or by their absolute geographic location. Row/column manipulation is generally easier when the contents of an entire raster are to be processed and geographic location more useful when location of individual cells are to be processed (for example as a result of user input).

All raster values are store as `float` values and can therefore store any type of number with a precision of up to about 7 decimal places. To set an arbitrary raster value use either `setAttribute(int row, int col, float newValue)` (row/column addressing)

or

`setAttribute(new Footprint(float easting, float northing), float newValue)` (geographic addressing)

Likewise, raster values can be retrieved using equivalent 'get' methods:

`float rasterValue = getAttribute(int row, int col)` (row/column addressing)

or

`float rasterValue = getAttribute(new Footprint(float easting, float northing))` (geographic addressing)

5.3.3 Handling Raster Metadata

In common with other `SpatialObjects`, the boundary information and metadata associated with a `RasterMap` can be found easily using a number of 'get' methods. In particular, to retrieve the bounds of a raster (or any `SpatialObject`, the method `getBounds()` will return a `Footprint` representing the origin, width and height of the raster in geographic coordinates. Unlike other spatial objects, the number of rows and columns in a raster can also be retrieved by calling `getNumRows()` or `getNumColumns()`. The range of attribute values with a raster can be found with `getMinAttribute()` and `getMaxAttribute()`.

Other non-spatial metadata can be retrieved using methods such as `getHeader()` (title, notes, owner information) and `getColourTable()`. Because these methods retrieve the metadata by reference, it is possible to edit metadata without calling the equivalent 'set' method. For example to change the title of `myRaster`, you could do the following:

```
myRaster.getHeader().setTitle("A new title");
```

Map projection information can also be retrieved and changed using `getProjection()` and `setProjection()`. The information itself is stored in the `Projection` class. Note however, that changing the map projection metadata does not itself change the projection of the raster. To reproject a spatial object you would need to call `ProjectionThread` or provide your own reprojection code.

5.3.4 Examples

The class below shows a simple example of manipulating a raster. It creates a new raster object of arbitrary dimensions and then allocates a random number between 0 and 100 to each cell in that raster. Finally it finds the range of values in that raster and allocates one of the preset colour tables stretched across that range.

```
import jwo.landserf.structure.*; // For spatial object class.
import jwo.landserf.process.io.*; // For file handling.

// *******************************************************
/** Creates a simple raster map containing random values
  * values. Demonstrates how raster values can be written.
  * @author Jo Wood
  * @version 1.1, 15th October, 2007
  */
// *******************************************************

public class RandomSurface
{
    //------------------ Starter Method -----------------

    public static void main(String[] args)
    {
        new RandomSurface();
    }

    //------------------- Constructor -------------------

    public RandomSurface()
    {
        // Create a raster of 400 x 400 random cells with a 50 unit grid spacing
        // and origin at (450000,280000)
        RasterMap raster = new RasterMap(400,400,new Footprint(450000,280000,50,50));

        for (int row=0; row<raster.getNumRows();row++)
        {
            for (int col=0; col<raster.getNumCols();col++)
            {
                raster.setAttribute(row,col,(float)Math.random()*100);
            }
        }

        // Add some simple metadata.
        Header header = new Header("Random surface");
        header.setNotes("Uncorrelated random noise");
        raster.setHeader(header);

        // Find range of values and create a colour table between them.
        float min = raster.getMinAttribute();
        float max = raster.getMaxAttribute();
        raster.setColourTable(ColourTable.getPresetColourTable(ColourTable.IMHOF_L3,min,max));

        // Write new raster to file.
        LandSerfIO.write(raster,"random.srf");
    }
}
```

The raster produced by this code is shown in Figure 5.4.

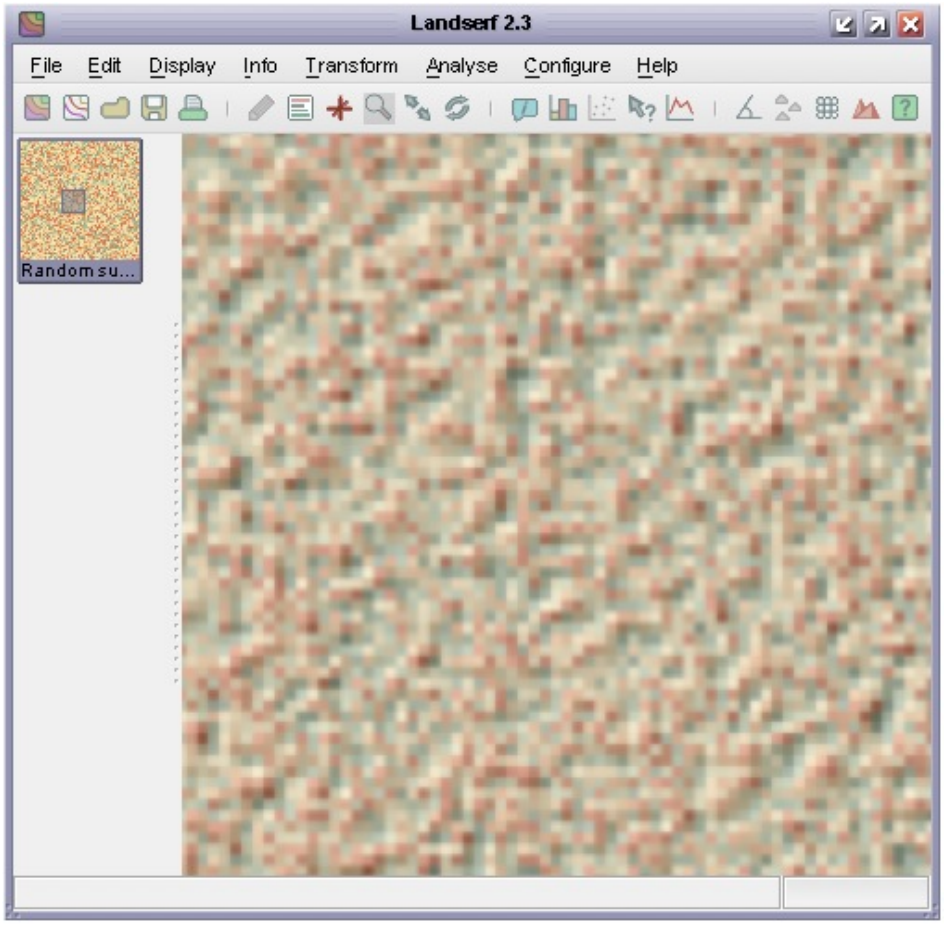


Figure 5.4: Output from CreateRandomSurface() showing uncorrelated random noise and 'Imhof' colour scheme.

This second example creates another raster, but this time based on reading the values of an existing one. It first creates a clone of the raster stored in the file `random.srf` (produced by the code above). It then replaces each cell (except the first row and column) with the average of its neighbouring cells.

```
1  import jwo.landserf.structure.*; // For spatial object class.
2  import jwo.landserf.process.io.*; // For file handling.
3
4  // *********************************************************
5  /** Opens a raster map and smoothes its contents by averaging
6    * local cells. Demonstrates how raster values can be read
7    * and written.
8    * @author Jo Wood
9    * @version 1.1, 15th October, 2007
10   */
11 // *********************************************************
12
13 public class SmoothSurface
14 {
15    //------------------ Starter Method -----------------
16
17    public static void main(String[] args)
```

```
18      {
19          new SmoothSurface();
20      }
21
22      //------------------- Constructor -------------------
23
24      public SmoothSurface()
25      {
26          // Read in a raster from a file.
27          RasterMap raster = LandSerfIO.readRaster("random.srf");
28
29          if (raster == null)     // Problem reading file
30          {
31              System.err.println("Problem reading raster");
32              System.exit(-1);
33          }
34
35          // Make a copy of the raster which will contain smoothed values.
36          RasterMap smoothedRaster = new RasterMap(raster);
37
38          // Smooth raster cells by taking average with cells to N and W.
39          for (int row=1; row<smoothedRaster.getNumRows();row++)
40          {
41              for (int col=1; col<smoothedRaster.getNumCols();col++)
42              {
43                  smoothedRaster.setAttribute(row,col,(raster.getAttribute(row-1,col-1)+
44                                                       raster.getAttribute(row, col-1)+
45                                                       raster.getAttribute(row, col))/3);
46              }
47          }
48
49          // Add some simple metadata based on original raster.
50          Header header = new Header("Smoothed "+raster.getHeader().getTitle());
51          header.setNotes("Smoothed "+raster.getHeader().getNotes());
52          smoothedRaster.setHeader(header);
53
54          // Write new raster to file using non-factory method.
55          LandSerfIO.write(smoothedRaster,"smoothed.srf");
56      }
57  }
```

The raster produced by this code is shown in Figure 5.5.

Exercises

1. Write a class that reads in two rasters of identical size and creates a new raster where each cell is based on the average of each pair of cells in the equivalent position in the two input rasters.
2. Modify the class you have just written to allow rasters of possibly different sizes and locations to be combined in this way. The new raster should be the intersection of the two input rasters. Your program should display a warning if the two input rasters do not intersect at all.

   Hint: You can use the method `getIntersectionMER()` to calculate the boundary of the intersecting region. You will also need to identify cells using absolute geographic referencing rather than row/column values.

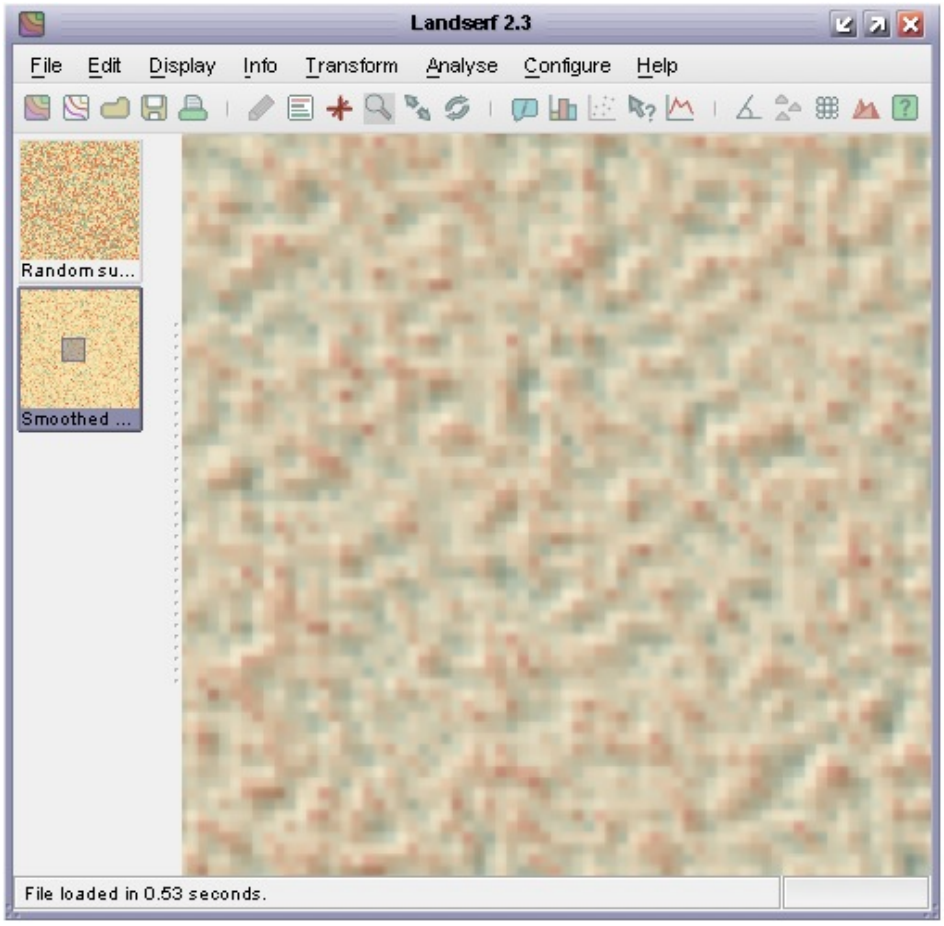


Figure 5.5: Output from `SmoothSurface()` showing smoothed random noise and 'Imhof' colour scheme.

5.4 Manipulating Vector Maps

By the end of this section you will be able to:

- Understand the relationship between a `VectorMap` and `GISVector`.
- Create `GISVectors` from sets of geometric coordinates.
- Create a vector map, add `GISVectors` to it and associate it with metadata
- Perform simple processing operations on vector maps

5.4.1 The structure of a vector map

In contrast with raster maps, vector maps are used to represent collections discrete spatial objects with known boundaries. Those objects may be points in space, lines, or areas defined by linear boundaries. A vector map may contain any number of these objects that may be separate from each other, joined, or overlapping.

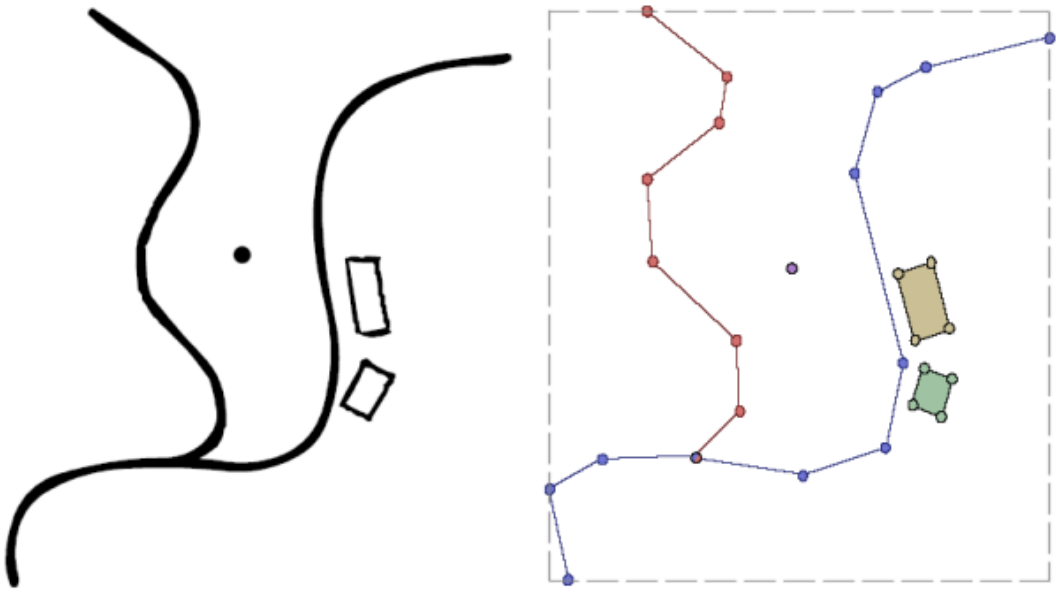


Figure 5.6: Simple point, line and area features and their vector map representation.

An individual vector object (such as the red line, blue line, yellow or green rectangle or point above) is represented by the class `GISVector`. This stores the geometry of the object as a series of (x,y) coordinate pairs. Implicit in this model is that lines and area boundaries are defined by joining consecutive pairs with straight line segments. A `GISVector` also inherits `SpatialObject` just as the `RasterMap` class does. It therefore has methods for defining its bounds, performing simple spatial comparisons, and manipulating its attribute(s). The `VectorMap` class simply assembles a collection of the `GISVectors` into a single object that itself has bounds and other metadata attached to it.

5.4.2 Creating a simple vector map

Creating a vector map is a simple process and just requires calling one of `VectorMap`'s constructors. Like the `RasterMap` class, you can define the bounds of the object at the point of creation, copy the contents of an existing spatial object, or simply create an empty vector map. Unlike rasters, it is more common for vector maps to grow and shrink as objects are added and removed, so it is quite usual initially to create an empty vector map object:

```
VectorMap myVectorMap = new VectorMap();
```

The geometry of an individual object within a vector map is stored in a `GISVector` object. The object also stores its type – one of `GISVector.POINT`, `GISVector.LINE` or `GISVector.AREA`, as well as its attribute value. These can all be set in `GISVector`'s constructor, or added later using an appropriate method.

The class below shows how some simple vector objects may be created and added to a vector map which is then saved to disk.

```
1  import jwo.landserf.structure.*; // For spatial objects.
2  import jwo.landserf.process.io.*; // For file handling.
3
4  // *********************************************************
5  /** Creates a simple vector map containing some GISVectors.
6    * Demonstrates how vector maps can be created and written.
7    * @author Jo Wood
8    * @version 1.1, 16th October, 2007
9    */
10 // *********************************************************
11
12 public class SimpleVectorMap
13 {
14    //------------------ Starter Method -----------------
15
16    public static void main(String[] args)
17    {
18       new SimpleVectorMap();
19    }
20
21    //------------------- Constructor -------------------
22
23    public SimpleVectorMap()
24    {
25       // Create an empty vector map, then place some GISVectors within it.
26       VectorMap vectorMap = new VectorMap();
27
28       float riverX[] = {2,1,4,7,10,12,11};
29       float riverY[] = {2,4,5,3, 4, 7,11};
30
31       GISVector road = new GISVector(riverX,riverY,SpatialModel.LINE,10);
32       vectorMap.add(road);
33
34       float buildingX[] = {6,10,11,10,9,7};
35       float buildingY[] = {8,10, 7, 5,7,6};
36
37       GISVector building = new GISVector(buildingX,buildingY,SpatialModel.AREA,20);
38       vectorMap.add(building);
39
40       GISVector pump = new GISVector(8,5,30);
41       vectorMap.add(pump);
42
```

```
43          // Add some simple metadata.
44          Header header = new Header("Simple vector map");
45          header.setNotes("Simple vector map containing a river, building and pump location.");
46          vectorMap.setHeader(header);
47
48          // Find range of values and create a random colour table.
49          float min = vectorMap.getMinAttribute();
50          float max = vectorMap.getMaxAttribute();
51          vectorMap.setColourTable(ColourTable.getPresetColourTable(ColourTable.RANDOM,min,max));
52
53          // Write new vector map to file.
54          LandSerfIO.write(vectorMap,"simple.vec");
55      }
56  }
```

Note the similarity with the way in which raster map metadata are added and saved to disk. The vector map that is created as a result of this class is shown in Figure 5.7.

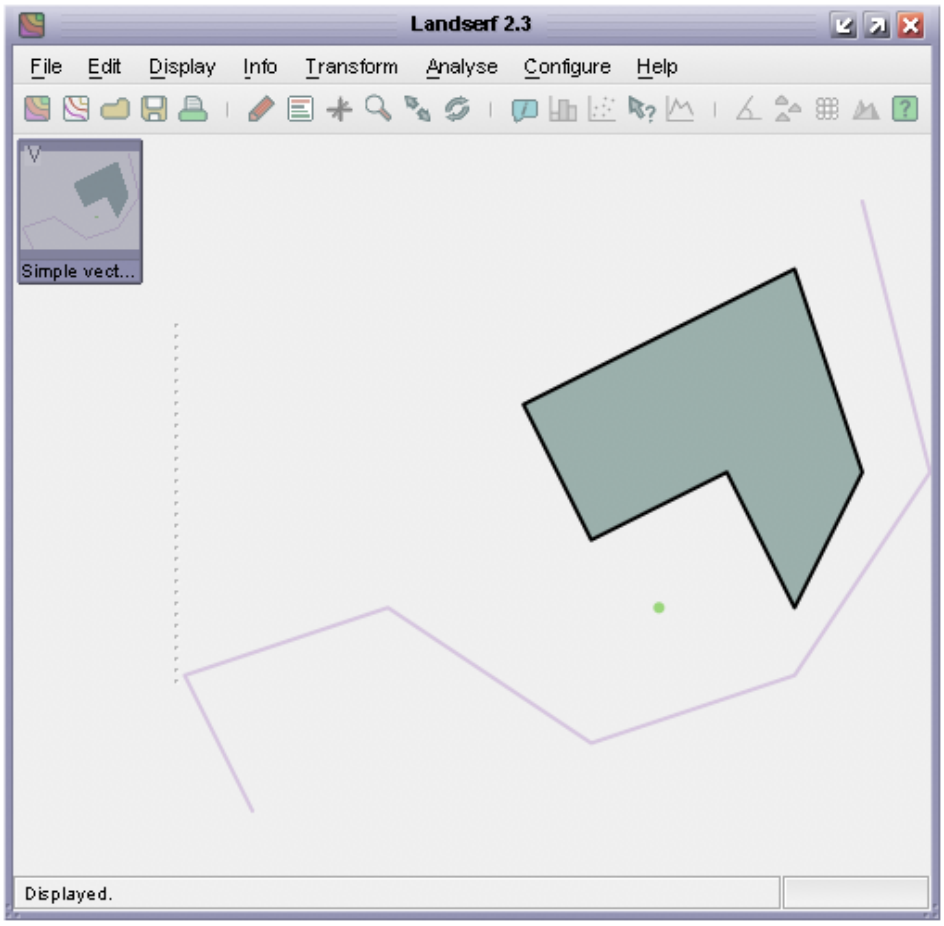


Figure 5.7: VectorMap output from the SimpleVectorMap class.

5.4.3 Manipulating Vector Maps

Vector maps can be read much as raster maps using the static method `LandSerfIO.readVector(`*`fileName`*`)`. Altering the metadata associated with a vector map is identical to that of a raster. To manipulate an individual `GISVector` within a vector map, the set of `GISVectors` contained can be iteratively extracted using the method `getGISVectors()`. This method (slightly unfortunately for naming purposes) returns a Java `Vector` of `GISVectors`. To process each object in turn, it would be normal to create an iterator:

```
Iterator i = myVectorMap.getGISVectors().iterator();

while (i.hasNext())
{
    GISVector gisVect = (GISVector)i.next();

    // Do something with each GISVector here.
}
```

Note that LandSerf 2.3 does not use generics to organise collections, so Java 1.5 'for-each' loops with generics cannot be used. Future releases of LandSerf will require Java 1.5 or above and will make use of generics for all collections.

Depending on what you wish to do, it is sometime possible to avoid manipulating GISVector objects directly by calling appropriate methods from the `VectorMap` object. The example below shows how two vector maps can be combined with simple `clone()` and `add()` methods.

```
1  import jwo.landserf.structure.*; // For spatial object class.
2  import jwo.landserf.process.io.*; // For file handling.
3
4  // *********************************************************
5  /** Combines two vector maps by overlaying one on the other
6    * and creating new objects from their union.
7    * Demonstrates how the GISVector maps can be manipulated.
8    * @author Jo Wood
9    * @version 1.2, 16th October, 2007
10   */
11 // *********************************************************
12
13 public class CombineVectors
14 {
15    //------------------ Starter Method -----------------
16
17    public static void main(String[] args)
18    {
19       new CombineVectors();
20    }
21
22    //------------------- Constructor -------------------
23
24    public CombineVectors()
25    {
26       // Open an existing vector map.
27       VectorMap vectorMap1 = LandSerfIO.readVector("simple.vec");
28
29       // Create a simple vector map to overlay over the first one.
30       VectorMap vectorMap2 = new VectorMap();
31
32       float triangleX[] = {6,11,2};
```

```
33        float triangleY[] = {12,8,3};
34
35        GISVector triangle = new GISVector(triangleX,triangleY,SpatialModel.AREA,30);
36        vectorMap2.add(triangle);
37
38        // Add some simple metadata.
39        Header header = new Header("Triangular vector map");
40        header.setNotes("Simple vector map containing a triangular GISVector.");
41        vectorMap2.setHeader(header);
42
43        // Clone the first vector map and add the second to it.
44        VectorMap vectorMap3 = (VectorMap)vectorMap1.clone();
45        vectorMap3.add(vectorMap2);
46
47        vectorMap3.getHeader().setTitle("Combined vector map");
48        vectorMap3.getHeader().setNotes("Union of "+vectorMap1.getHeader().getTitle()+
49                                        " and "+vectorMap2.getHeader().getTitle());
50
51        float min = vectorMap3.getMinAttribute();
52        float max = vectorMap3.getMaxAttribute();
53        vectorMap3.setColourTable(ColourTable.getPresetColourTable(ColourTable.RANDOM,min,max));
54
55        // Write new vector map to file.
56        LandSerfIO.write(vectorMap3,"overlay.vec");
57    }
58 }
```

The output from this class is shown in LandSerf in Figure 5.8:

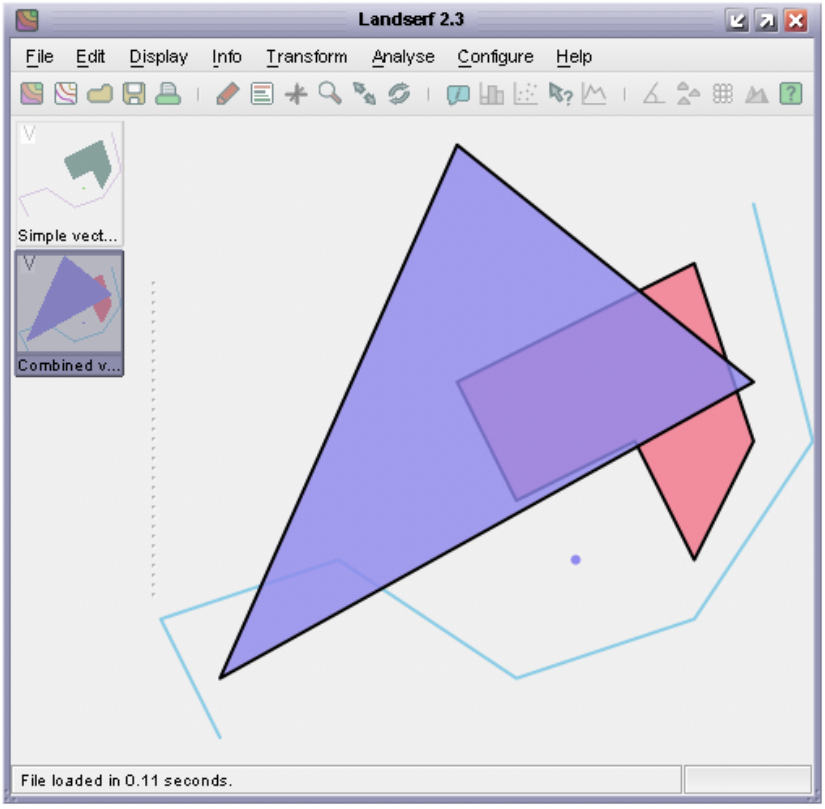


Figure 5.8: VectorMap output from the CombineVectors class.

Exercises

1. Write a class that reads in a vector map and separates the map into 'layers' of vector maps, each contining objects with the same attribute. To do this, you will need to iterate through each `GISVector` contained within the input map, identify its attribute, and then create a new vector map or add it to an existing one.
2. Modify the `CombineVectors` class shown above so that it uses the existing colour tables of the combined vector maps (rather than creating a random one as above). You will have to consider what to do if the two vector maps have different colours assigned to the same attribute value.

5.5 Using the `jwo.landserf.process` classes

By the end of this section you will be able to:

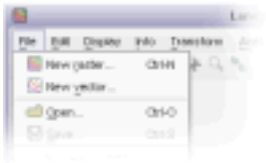
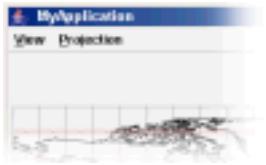
- Store spatial objects in an object that implements the `GISFrame` interface.
- Understand the difference between the `GISFrame`, `GISFrameAdapter`, `GUIFrame` and `SimpleGISFrame` classes.
- Create an object from the `jwo.landserf.process` package and start it as a threaded process.
- Provide input and extract output from a `GISFrame` used by a processing class.

5.5.1 Collecting input and output in a `GISFrame`

We have already seen how to read, create and write raster and vector spatial objects using Java and the LandSerf classes. When writing your own programs it is likely that you are doing so because you wish to perform some processing of those spatial objects between the reading and writing stages. You can take advantage of much of the processing functionality of LandSerf by creating objects from the `jwo.landserf.process` package. Classes in this package handle tasks that involve some significant (often time consuming) analysis and processing operations.

Different processing operations will require a range of spatial objects for input, and may create many different output spatial objects. For example, surface peak detection requires a DEM as input, and creates up to three raster maps and one vector map as output. Other routines such as fractal surface generation require no input spatial objects at all. To handle the varying input and output demands of different processes, all the classes in this package use a `jwo.landserf.gui.GISFrame` to hold the potential collection of input and output spatial objects. The `GISFrame` is an interface that defines the minimum requirements for assembling spatial objects, often in the form of a graphical user interface. For example, `jwo.landserf.gui.GUIFrame` is the main GUI window that implements a `GISFrame` and is used when you normally start LandSerf.

As a programmer, you must decide whether to create a graphical user interface for your program or whether it is to run from the command line. If you do create a GUI, should it use the 'normal' LandSerf GUI or do you wish to create your own GUI? Depending on what you choose, there are various incarnations of `GISFrame` you can use:

Type of user interface	**GISFrame to use**
 'LandSerf' GUI	`jwo.landserf.gui.GUIFrame`
 Customised GUI	Inherit `jwo.landserf.gui.SimpleGISFrame` or `jwo.landserf.gui.GISFrameAdapter`
 Command line interface	Inherit `jwo.landserf.gui.SimpleGISFrame`

The easiest to use is probably the `SimpleGISFrame` that implements the spatial object storage methods allowing any number of raster maps or vector maps to be added and removed. We will use this class in the discussion below. The `GISFrameAdapter` is simply an 'empty' implementation of the `GISFrame` that contains no functionality, but speeds up the process of creating a class that implements the interface. This is only really necessary if you are creating a customised user interface that behaves in a very different way to the one used by LandSerf.

Spatial objects can be added to a `GISFrame` by calling the `addRaster()` or `addVectorMap()` methods. Raster and vector maps can be added as *primary* (default), *secondary* or *unselected* objects. This distinction is important in determining which objects are selected by the processing classes. Most operator on the currently selected primary object, while some require both a primary and secondary selected object. Output from the processing classes can be as new primary, secondary or unselected objects stored in the `GISFrame`.

You can consult the Landserf 2.3 API documentation to find out what input is required and output produced by each of the processing classes. You may also find the table below a useful summary.

Processing class	Input objects	Output objects
`BasinThread`	Primary raster (surface) and primary vector (seed points)	Secondary raster (drainage basins)
`CentroidThread`	Primary vector (polygons)	Primary vector (polygon centroids)
`CombinePointsWithRasterThread`	Primary raster and primary vector (points)	Primary vector (points with new raster attributes)
`CombineThread`	Primary and secondary raster *or* primary and secondary vector	Unselected raster *or* vector
`ContourThread`	Primary raster (surface)	Primary vector (contours)
`DeleteThread`	Objects provided to constructor	None
`DemToTinThread`	Primary raster (surface)	Primary vector (TIN) and optional secondary raster (error map)
`DispThread`	Primary spatial objects and optional secondary objects	None
`FlowThread`	Primary raster (surface)	Secondary raster (flow magnitudes)
`FracSurfaceThread`	Empty primary raster (for raster dimensions)	Primary raster (fractal surface)
`FuzzyFeatureThread`	Primary raster (surface)	6 unselected rasters (pits, channels, passes, ridges, peaks, planes)
`JoinVectorThread`	Primary vector (fragmented lines)	Primary vector (joined lines)
`OpenThread`	Objects provided to constructor	Primary spatial object(s) and optional unselected objects (if more than one raster or vector provided)
`PeakClassificationThread`	Primary raster (surface)	Secondary raster (peaks) and optional primary vector (summits) and optional unselected rasters (fuzzy peaks and peak hierarchy)
`PitRemovalThread`	Primary raster (surface)	Secondary raster (pitless surface) and unselected raster (pits map)
`PointDensityThread`	Empty raster (for dimensions of density surface) and (points) vector provided to the constructor	Primary raster (density surface)
`PolySurfaceThread`	Primary raster (for dimensions)	Primary raster (polynomial surface)
`ProjectionThread`	Primary raster *or* primary vector	Primary raster *or* primary vector (re-projected map)
`RectifyThread`	Raster provided to the constructor (unrectified)	Primary raster (rectified raster)
`SaveThread`	Primary raster *or* primary vector	None
`ScaleParamThread`	Primary raster (surface)	Secondary raster (average surface parameter values), unselected raster (variation in surface parameter values)
`SelectVectorThread`	Primary vector (source map)	Primary vector (selected objects)
`SimplifyThread`	Primary vector (source map)	Primary vector (simplified objects)
`SurfNetworkThread`	Primary raster (surface) and secondary raster (surface features)	Primary vector (surface network) and secondary raster (thinned surface network)
`SurfParamThread`	Primary raster (surface)	Secondary raster (surface parameter)
`TinPointsThread`	Primary vector (points or TIN)	Primary vector (TIN or points)
`TinToDemThread`	Primary vector (TIN)	Primary raster (surface)
`TransRastThread`	Primary raster (source)	Primary raster (transformed raster)
`UpdateThread`	None	None (updates display)
`VectorToRasterThread`	Vector map (source) and empty raster (for dimensions) provided to the constructor	Primary raster (rasterized vector)
`View3dThread`	Primary raster (surface), optional primary vector (overlay) and active `GraphicsArea` (display)	None
`VoidRemovalThread`	Primary raster (with voids)	Primary raster (without voids)

5.5.2 Threaded Processing

All the processing classes mentioned above inherit `LSThread` which handles messages by the processor and allows the class to be threaded. This can be useful for constructing GUIs, as many of the processing tasks can take some time to complete. By threading their processing, other GUI activities can continue while processing takes place. To execute one of the processing classes as a threaded process, simply create an object from it and call its `start()` method. For example,

```
SurfParamThread processor = new SurfParamThread(gisFrame, mySurfParam);
processor.start();
```

This will call the `processor`'s `doProcessing()` method automatically, and inform the parent GUI (`gisFrame`) when it is complete. Calling the `start()` method is preferred to calling `doProcessing()` directly, since the former also initialises the message communication between the processing class as the `gisFrame`.

If you are creating a command line interface where GUI activities are not going to be taking place while the process runs, it may make more sense not to run a process in parallel with other activities. In particular, it may be necessary to know when the process has completed before, for example, saving an output file to disk. The easiest way to do this is probably to let your class 'join' the thread (in other words, halt further execution until the thread has completed). For example,

```
SurfParamThread processor = new SurfParamThread(gisFrame, mySurfParam);
processor.start();
processor.join();
// This point is reached when the processor has completed its task.
```

Alternatively, you can poll the thread repeatedly until it has completed by calling the thread's `isAlive()` method. An example illustrating these ideas showing how processing classes may be used is shown below:

```
1  import jwo.landserf.structure.*; // For spatial object class.
2  import jwo.landserf.gui.*;       // For GISFrame.
3  import jwo.landserf.process.*;   // For processing classes.
4  import jwo.landserf.process.io.*; // For file handling.
5
6  // ********************************************************
7  /** Creates a fractal surface and contours it. Demonstrates
8    * how processing classes can be used.
9    * @author Jo Wood
10   * @version 1.1, 16th October, 2007
11   */
12 // ********************************************************
13
14 public class Fractal
15 {
16    //------------------ Starter Method -----------------
17
18    public static void main(String[] args)
19    {
20       new Fractal();
21    }
22
23    //------------------- Constructor -------------------
24
25    public Fractal()
26    {
27       // Create a GISFrame that will store raster and vector maps.
28       GISFrame gisFrame = new SimpleGISFrame();
29
30       // Add a blank raster to the GISFrame.
31       RasterMap fracRaster = new RasterMap(400,400,new Footprint(450000,280000,50,50));
32       fracRaster.getHeader().setTitle("Fractal surface");
33       gisFrame.addRaster(fracRaster,GISFrame.PRIMARY);
34
35       // Create and start threaded process to generate fractal surface.
36       FracSurfaceThread fracThread = new FracSurfaceThread(gisFrame,2.2f);
37       fracThread.start();
38       try
39       {
40          fracThread.join();  // Join thread (i.e. wait until it is complete).
41       }
42       catch (InterruptedException e)
43       {
44          System.err.println("Error: Fractal generation thread interrupted.");
45          return;
```

```
46          }
47          System.out.println("Fractal surface created.");
48
49          // Create and start a threaded process to contour the fractal.
50          float minVal = fracRaster.getMinAttribute();
51          float range = fracRaster.getMaxAttribute()-minVal;
52          ContourThread contourThread = new ContourThread(gisFrame,minVal,range/30f,4);
53          contourThread.start();
54          try
55          {
56              contourThread.join();  // Join thread (i.e. wait until it is complete).
57          }
58          catch (InterruptedException e)
59          {
60              System.err.println("Error: Contouring thread interrupted.");
61              return;
62          }
63          System.out.println("Fractal surface contoured.");
64
65          // Write new raster and vector maps to file.
66          LandSerfIO.write(fracRaster,"fractal.srf");
67          LandSerfIO.write(gisFrame.getVectorMap1(),"contours.vec");
68      }
69  }
```

Running the class above produces two new spatial objects - the raster map `fractal.srf` and the vector map `contours.vec`. They are shown in Figure 5.9 (since the fractal is randomly generated each time, the form of the surface will vary each time this program is run).

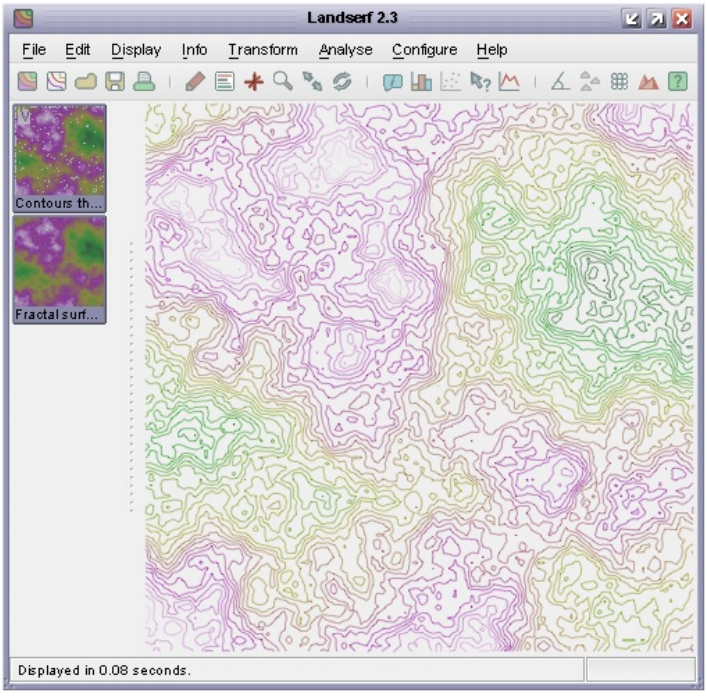


Figure 5.9: Fractal surface and contour output from `Fractal`.

Exercises

1. Write a program that takes the contour output from `Fractal` above and creates a new vector map containing only those contour lines above the midpoint between the highest and lowest elevations. *Hint:* You can achieve this using the `SelectVectorThread` to perform the contour selection process.
2. Write a program that converts a surface (e.g. the fractal surface generated above) into a TIN, and then converts the TIN back to a surface again. Create and save a new raster that is the difference between the original surface and the version produced via the TIN round-trip.
   Hint: You can use processing classes to perform the TIN conversions, but you will need to manipulate the individual raster cell values in order to construct the difference map.