Robert Pfeffer

Molehills in QGIS

Relief Representation in Molehill Style
Based on Digital Elevation Data

'Molehills' are an ancient type of relief depiction commonly found in maps of the 16th to 18th centuries and, inspired by these, in maps of the fantasy genre. Molehills lend themselves where a certain antique aesthetic is desired, while geographical accuracy is not a priority. This tutorial explains how hills prepared as SVG graphics can be automatically placed on the map using downloaded elevation data with a little extra work in QGIS.[1]

This is essentially done in six steps:

  I.   Download of the elevation data,

  II. Conversion of the elevation map into a steepness map,

  III. Derivation of medium and high mountain areas from the steepness map,

  IV. Creating source layers and target layer for the hill points,

  V.   Generating points for different scales and

 VI. Representation of the points by hill graphics.

---

[1] It has been tested in QGIS 3.28 and 3.34. — I did not programme the Python scripts used here on my own, as I am not familiar with it. ChatGPT was a great help!

I. Download of the Elevation Data

First, we need an elevation map for the area in question. For this we can use the *OpenTopography DEM Downloader* plugin and download Copernicus 90 data, for example. As the server only ever releases limited volumes, the data may have to be downloaded in chunks; in the example below, this is done in 5° wide strips:
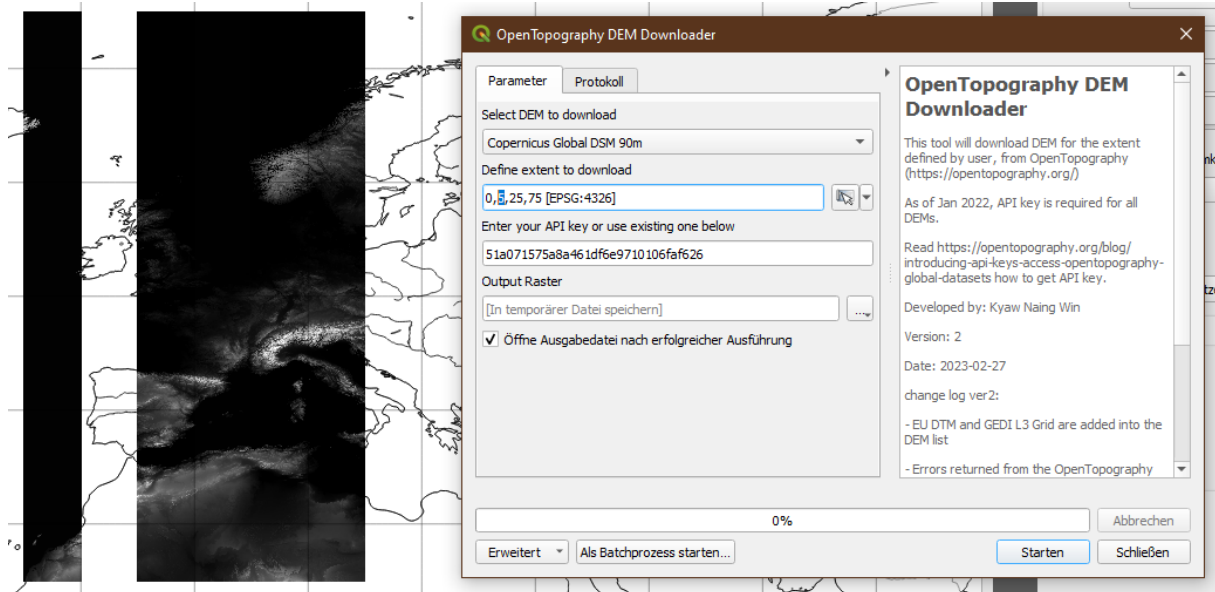

The many different layers arising from this should be merged into one layer before further processing. If QGIS reaches its performance limits due to the large amount of data, you may have to do with several, as few layers as possible. The function for merging can be found in the menu under *Raster —> Miscellaneous —> Merge:*
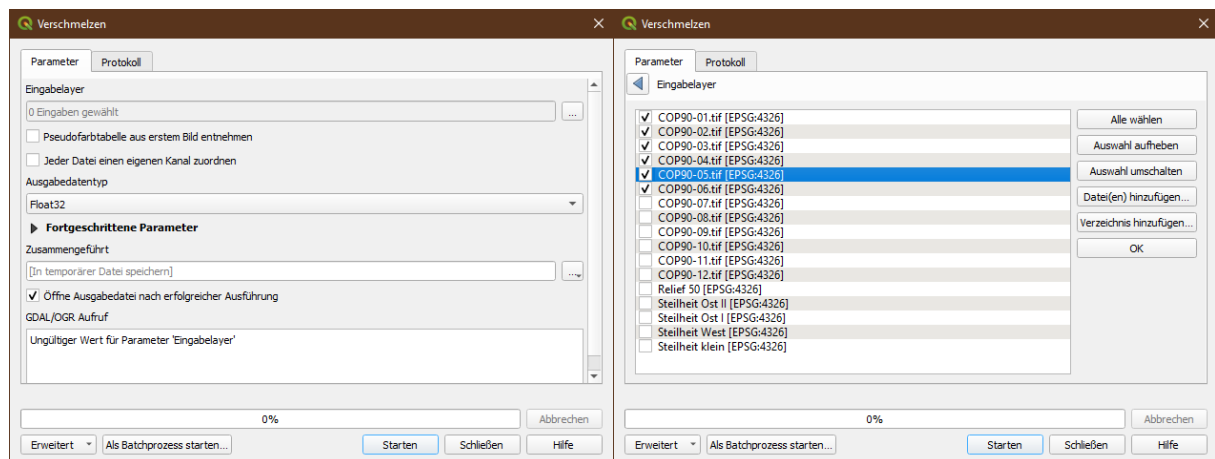
## II. Conversion of the Elevation Map into a Steepness Map

Now we have a nice elevation map, which, however, isn't much use—as we don't wish to seed a plain with hills just because it's a high plain, whereas conversely we want to depict coastal mountains even if they don't rise much above zero. So what we need instead is a map showing not the terrain's elevation, but its steepness.

To do this, we use the GRASS GIS function *r.slope.aspect*. Under *Advanced parameters* in the function's window, only the tick behind 'Slope' must remain, all others can be removed. I have selected 4 as the multiplier; other values are certainly just as possible.
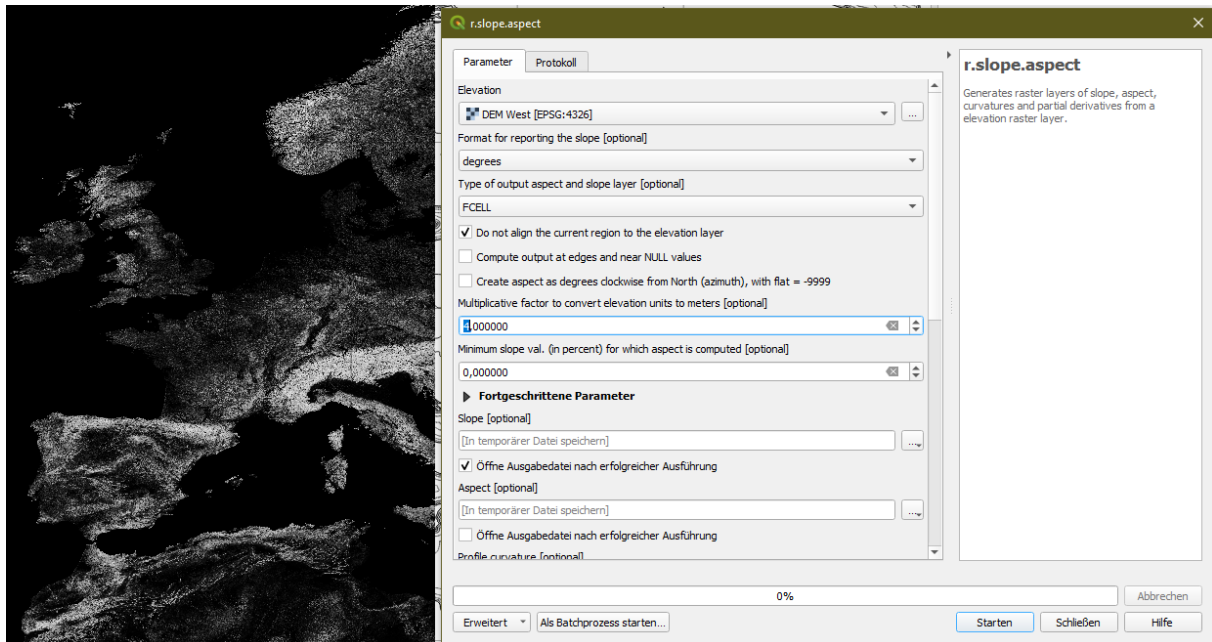

This results in a map no longer showing how high the terrain is, but how steeply rugged it is. Note the high plains of Spain and Morocco, for example: they are no longer grey (for being high), but black (for being plain).

To ease further processing, we first reduce the graphic to one twentieth of its hitherto size by right-clicking on the layer, followed by —> *Export* —> *Save as…*. In the window that opens now, we remove one zero from each row and column and divide the remainder by two. Under *File name* we select a storage location and save the graphic as a new layer (see right).
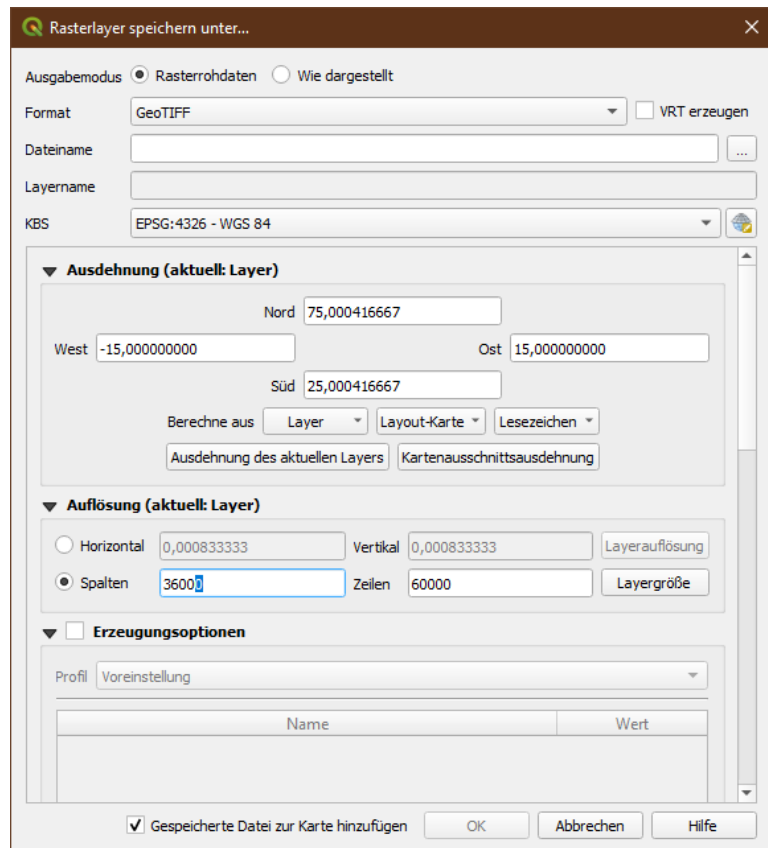
However, this reduced map is still too fine-grained for further use, i.e. vectorisation. We therefore apply a Gaussian blur to it by using the *Gaussian filter* from the SAGA tools[2]; in this example, a radius of 15 seemed appropriate (see next page).

___

[2] QGIS 3.28 seems to be the last version still supporting SAGA tools out of the box. QGIS 3.34 already requires the "Processing Saga NextGen Provider" plugin and its being set up under *Settings —> Options —> Processing —> Provider*.
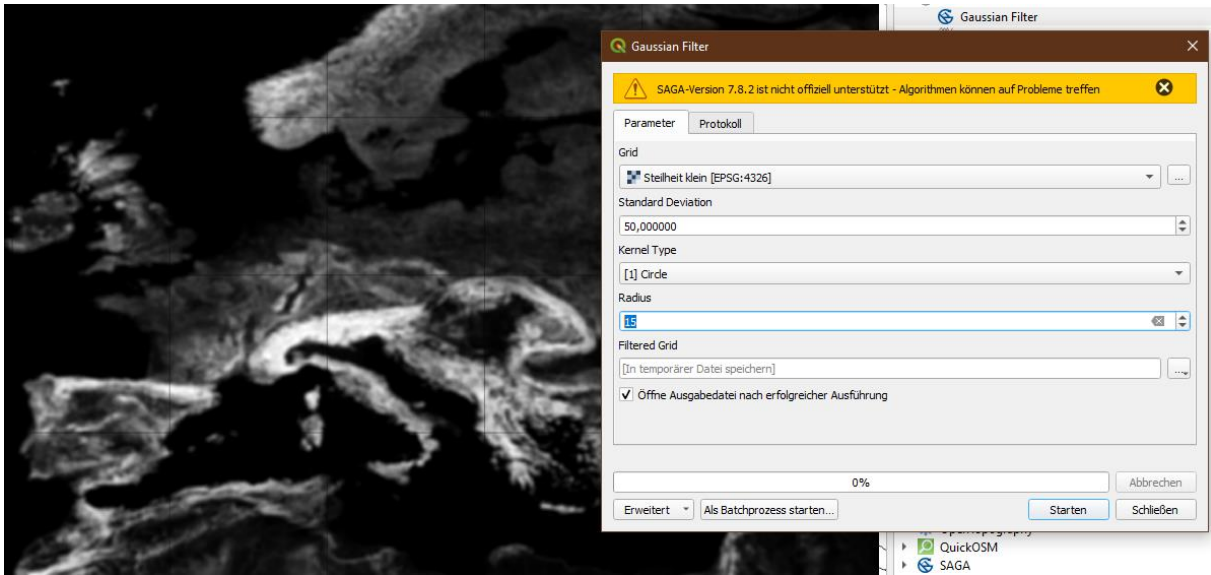
III. Derivation of Medium and High Mountain Areas from the Steepness Map

Now the areas covered by low and high mountains must be found and extracted as polygons. To do so, we use the GDAL function *Contour polygons*, which is genuinely intended for contour lines. In this example, an interval value of 22 leads to suitable results:
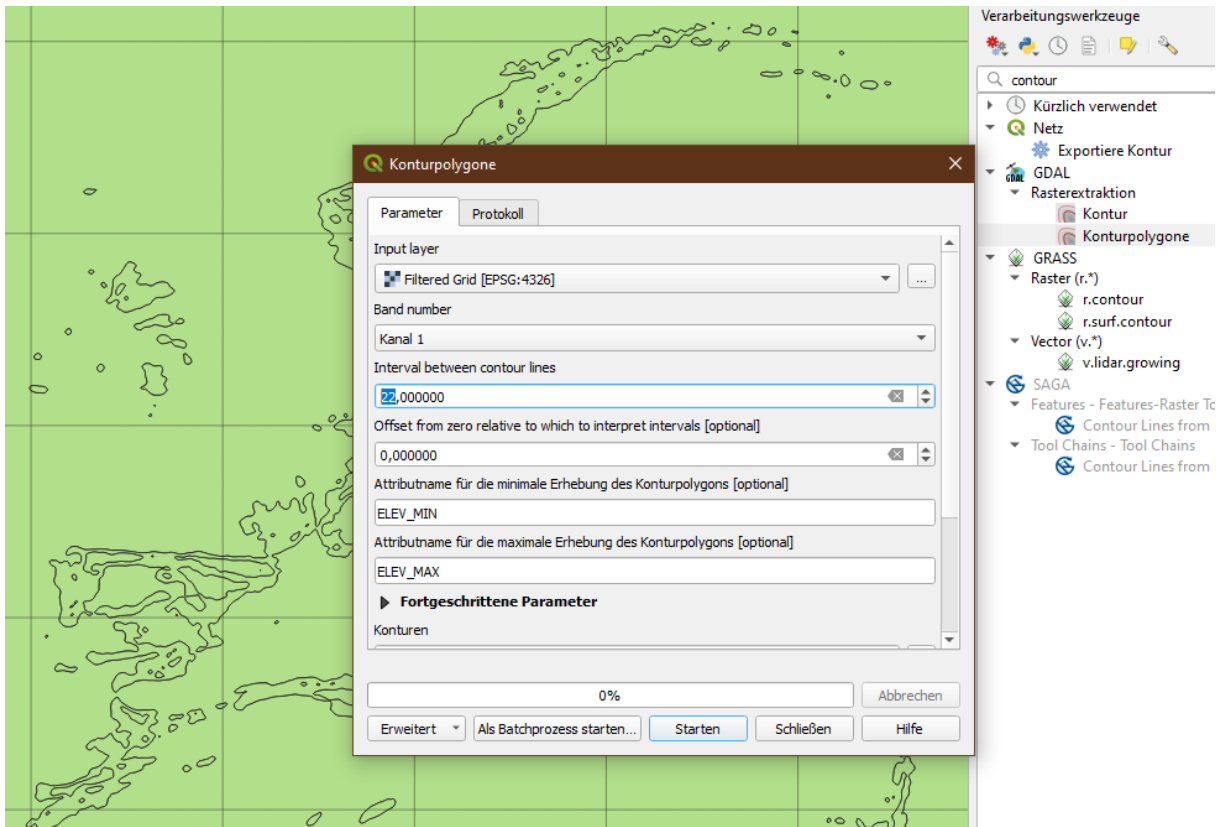
Now we have three polygons on the newly created layer: one extends up to the map's edges and covers all relatively flat areas including the seas; we can delete this one. What remains are the two polygons for medium and high mountain areas. If there is an additional one for the steepest peaks, we can also delete it if it's negligibly small, or we can start the procedure again with a value higher than 22. (If there are even more polygons, the value was too low anyway.)
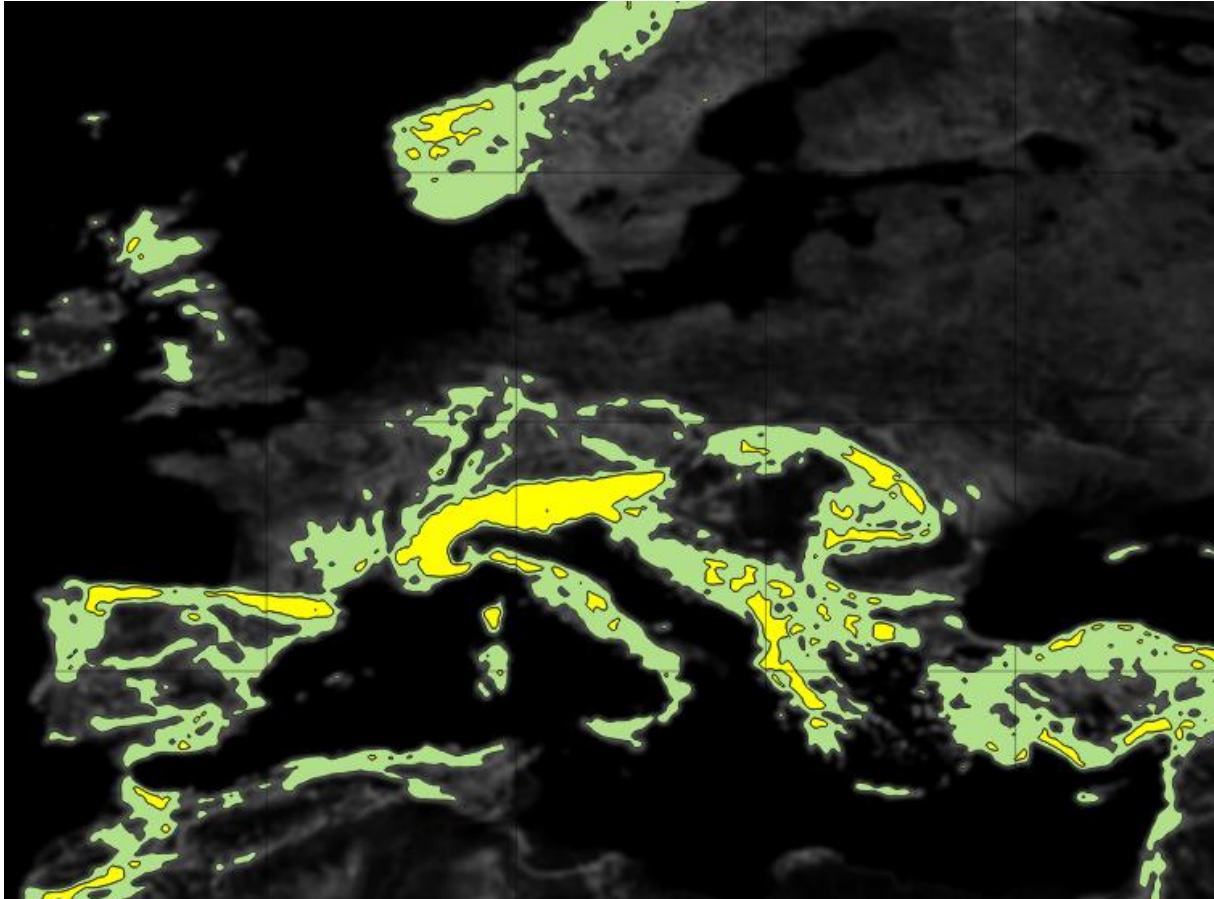
IV. Creating Source Layers and Target Layer for the Hill Points

We allocate the two remaining polygons to two layers, which we call "Mittelgebirge" ('medium mountains') and "Hochgebirge" ('high mountains')[3]. In addition, we create another vector plane called "Gebirgspunkte" ('mountain points'), whose geometry consists of points. To this layer the following fields need to be added, either at the layer's creation or later in its properties (except the field "id" being already existent, but useless):

Q Layereigenschaften - Gebirgspunkte — Felder

Id ▲	Name	Alias	Typ	Typname	Länge	Genauigkeit
123 0	id		Ganzzahl (Integer 64 bit)	Integer64	10	0
abc 1	SVG		Text (string)	String	10	0
123 2	Massstab		Ganzzahl (Integer 64 bit)	Integer64	10	0
abc 3	Kategorie		Text (string)	String	10	0
123 4	Versatz		Ganzzahl (Integer 64 bit)	Integer64	10	0
1.2 5	Skalierung		Dezimal (double)	Real	10	3

Sidebar items: Beschriftungen, Masken, 3D-Ansicht, Diagramme, Felder, Attributformular

---

[3] As I originally puzzled out this method and wrote these scripts just for myself, most of the terms used are in German. You can of course translate them, but will then have to change the functions in their corresponding places, too.

In the following, the aim is to fill the areas of the two polygons with randomly arrayed points, which in turn are represented by randomly selected SVG graphics. The first two layers serve as source layers from which the points are generated, and the third one is the target layer in which they are saved.

This is done using a Python function[4], which is provided in two variants—one for the medium and one for the high mountains (see appendix)—and which essentially does the following:
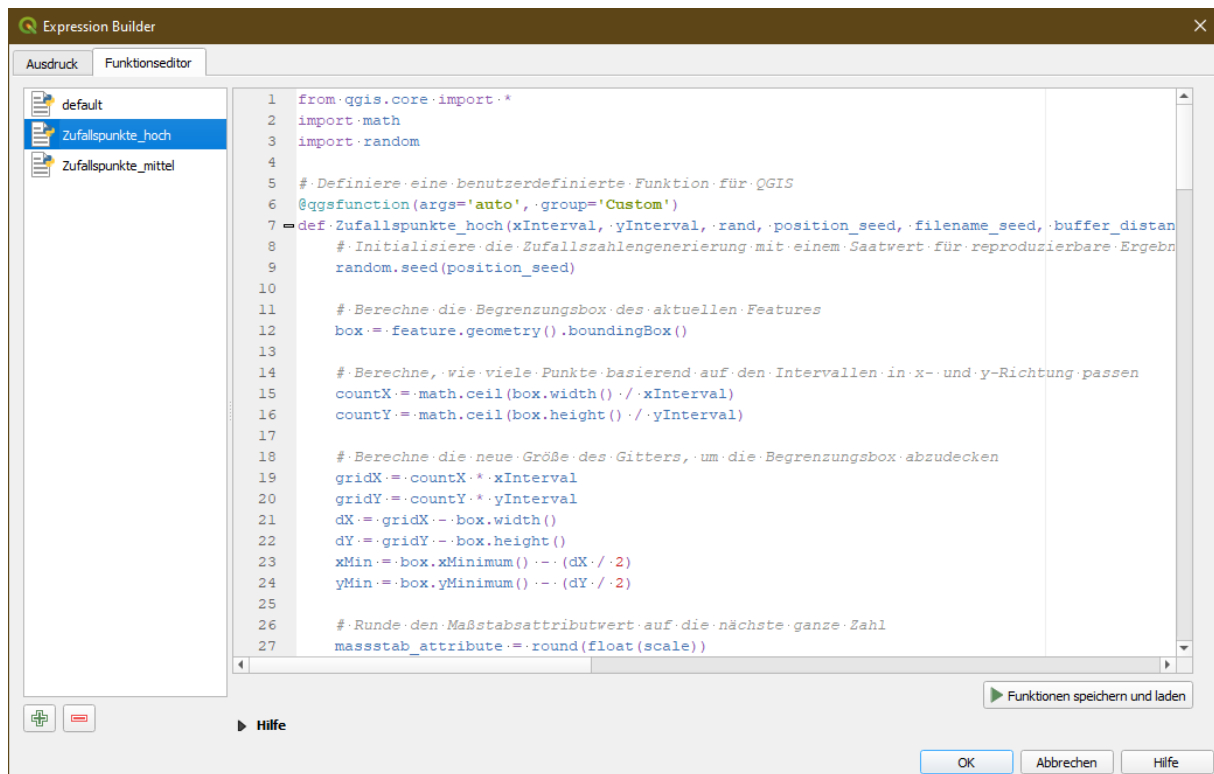
1. draw a grid over the entire extent of the polygon,

2. randomly place one point in each box of the grid,

3. delete all points that do not end up inside the polygon,

4. delete all points that are too close to the geometry of certain other layers from which they should keep away (e.g. rivers, coastlines, …),

5. assign an attribute value of +1 or -1 to all points that are still too close to the right or left of that geometry so that they can be moved aside later (see below for the sense and nonsense of this duplication),

6. randomly assign each point a file name from "MH01.svg" to "MH63.svg" (these are the 63 molehill graphics from the prepared set being enclosed) and

7. assign a random scaling factor of 0.9 to 1.1 to each point in order to add some additional variety to the map design.

The purpose of points 4 and 5 is to prevent the hills from coming too close to the geometry of certain other layers (which must be entered by name in the function, see appendix). In point 4, a buffer is drawn around that geometry in order to then delete all points that lie within it. As our hills are more wide than high, it would actually be nice to have a buffer that is also more wide than high. However, QGIS doesn't seem to provide for this. Moreover, if degrees are used as map units (as is the case here), these are more high than wide in European latitudes, and so the buffer is exactly the opposite of what it should be. The only solution I could think of is to check again in a further step 5 which points would still be in the buffer if they were a little further to the right or to the left (by half a buffer width). A positive or negative attribute value is then assigned to these points, which later can serve as a reference for a lateral marker offset[5]. (This method has one advantage: points that would otherwise be deleted are retained and are only shifted a little towards the others.)

In the *Layer Styling* of both source layers (high and medium mountains), we now select *Geometry generator* as the fill, *Points* as the geometry type and *Millimetres* as the units (in this example). Next to the expression field, we press the ε button to open the Expression Dialogue. There we first select the Function Editor, create two new function files called "Random points high" and "Random points medium" (or any other name) and insert the code of the two functions for high and medium mountains into these files (see appendix). We can do this for both layers in one, as the Function Editor's content is available in both layers.

---

[4] The 'historic nucleus' of this function is a code published by Rob Jones at https://impermanent.io/2017/05/05/generative-pseudo-random-polygon-fill-patterns-in-qgis/ (last accessed on 02/02/2014). I'd like to take this opportunity to credit him for this.

[5] See VI.4 below.

```
Q Expression Builder                                                                          ×
Ausdruck    Funktionseditor
┌─────────────────────┐  1  from·qgis.core·import·*
│ 📄 default          │  2  import·math
│ 📄 Zufallspunkte_hoch│ 3  import·random
│ 📄 Zufallspunkte_mittel│4
│                     │  5  #·Definiere·eine·benutzerdefinierte·Funktion·für·QGIS
│                     │  6  @qgsfunction(args='auto',·group='Custom')
│                     │  7 ▭def·Zufallspunkte_hoch(xInterval,·yInterval,·rand,·position_seed,·filename_seed,·buffer_distan
│                     │  8      #·Initialisiere·die·Zufallszahlengenerierung·mit·einem·Saatwert·für·reproduzierbare·Ergebn
│                     │  9      random.seed(position_seed)
│                     │ 10
│                     │ 11      #·Berechne·die·Begrenzungsbox·des·aktuellen·Features
│                     │ 12      box·=·feature.geometry().boundingBox()
│                     │ 13
│                     │ 14      #·Berechne,·wie·viele·Punkte·basierend·auf·den·Intervallen·in·x-·und·y-Richtung·passen
│                     │ 15      countX·=·math.ceil(box.width()·/·xInterval)
│                     │ 16      countY·=·math.ceil(box.height()·/·yInterval)
│                     │ 17
│                     │ 18      #·Berechne·die·neue·Größe·des·Gitters,·um·die·Begrenzungsbox·abzudecken
│                     │ 19      gridX·=·countX·*·xInterval
│                     │ 20      gridY·=·countY·*·yInterval
│                     │ 21      dX·=·gridX·-·box.width()
│                     │ 22      dY·=·gridY·-·box.height()
│                     │ 23      xMin·=·box.xMinimum()·-·(dX·/·2)
│                     │ 24      yMin·=·box.yMinimum()·-·(dY·/·2)
│                     │ 25
│                     │ 26      #·Runde·den·Maßstabsattributwert·auf·die·nächste·ganze·Zahl
│                     │ 27      massstab_attribute·=·round(float(scale))
│                     │                                              ▶ Funktionen speichern und laden
│ ➕  ➖              │   ▶ Hilfe
└─────────────────────┘                                    OK      Abbrechen      Hilfe
```

Now we switch to the *Expression* tab, and now it becomes layer-specific: Here we insert the expression that will call the function as soon as the layer is made visible. For now, the following expressions may suffice to create an appealing point pattern at a scale of 1 : 15 000 000:

Zufallspunkte_hoch(0.7, 0.35, 0.35, 33, 42, 0.25, $scale)

Zufallspunkte_mittel(0.53, 0.3, 0.48, 33, 42, 0.25, $scale)

Please note that "Zufallspunkte_hoch" and "Zufallspunkte _mittel" ('Random points high' and 'Random points medium') relate to the function names defined in the code (see appendix) — so if you prefer to alter them, you'll have to alter them there as well. The parameters in the brackets have the following meaning in the upper of the two examples (see also the comments at the beginning of both functions in the appendix):

– "0.7" and "0.35" are the width and height of the individual grid boxes, measured in map units, in this case degrees. If you take twice the height for the width, this results in an approximately square box in European latitudes.

– Another "0.35" determines the degree of random scatter with which the individual point is placed in the respective box.

– "33" and "42" are arbitrarily selectable values as random seeds for the point scattering and for the assignment of the SVG file names. As long as this seed remains the same, the random pattern will also remain the same. So if you don't like the hill display, you can vary it at these points without basically changing the points' density.

– "0.25" determines the width of the buffer that is drawn around the geometry of the other layers mentioned, i.e. the zone to be kept free of points.

– "$scale" passes the reciprocal value of the current scale (i.e. "15 000 000" for a scale of 1 : 15 000 000). This allows the above expressions to be refined later so that they take into account the respective scale (see below).

V. Generating Points for Different Scales

The points are now created simply by making one of the source layers visible and allowing QGIS to execute the function. This may take quite a while, and QGIS may well crash! With a bit of luck, however, the points will then already have been generated and saved.

Although the points are generated from the source layers using the functions stored therein, they are saved in the "Gebirgspunkte" target layer. The source layers must and should therefore only be made visible for the purpose of generating points and then immediately made invisible again.

Both functions are programmed to only generate points if none are present for the current scale, yet. They are also preset to not generate points for any crooked scale, but only for scales whose reciprocal value is divisible by 250 000, which applies, for example, to the initial scale of 1 : 15 000 000 selected here as well as to many of its halves. (This can of course be changed in the functions if required, see appendix).

The points are thus created for different scales by making the source layer visible in different scales or by scrolling through different scales when the source layer is visible. This immediately reveals the problem that the distances passed as parameters are absolute, so that the grid boxes and thus also the points have the same geographical distance in each scale. If you want the points to be twice as dense at double the scale so that the visual distance remains the same, only half the value may be passed as a parameter at double the scale. This is achieved by inserting a fraction using "$scale" in the numerator instead of an absolute value. (For precisely this purpose, "$scale" is passed to the function as the last parameter). So if, for example, a horizontal distance of 0.7 seemed appropriate at a scale of 1 : 15 000 000—like this:

<p style="text-align:center">Zufallspunkte_hoch(0.7, …,</p>

then you can replace this 0.7 with a fraction that gives the same result as 0.7 on the scale mentioned—namely 15 000 000 / 21 428 571.4—and which, using "$scale" instead of "15 000 000" and being rounded a little, reads like this:

<p style="text-align:center">Zufallspunkte_hoch($scale/21400000, ….</p>

If the scale is doubled, "$scale" is then only half as large, which would halve the geographical distance so that visual distance remains the same. If you now want the visual distance not to exactly remain the same at double the scale, but to become a little bit larger again, you can add a small summand to the parameter, which has the exact opposite effect, e.g:

<p style="text-align:center">Zufallspunkte_hoch($scale/21400000+1200000/$scale, ….</p>

You can proceed in the same way with all other parameters that denote distances. I ended up with the following expressions:

<p style="text-align:center">Zufallspunkte_hoch($scale/25000000+1200000/$scale,
$scale/50000000+600000/$scale, $scale/70000000+2000000/$scale, 33, 42,
$scale/70000000+500000/$scale, $scale)</p>

<p style="text-align:center">Zufallspunkte_mittel($scale/33000000+1200000/$scale,
$scale/65000000+1000000/$scale, $scale/60000000+3500000/$scale, 33, 42,
$scale/70000000+500000/$scale, $scale)</p>

## VI. Representation of the Points by Hill Graphics

All generated points are now registered in the attribute table of the 'Gebirgspunkte' layer, where they differ in their respective attributes: The "SVG" column specifies which of the 63 molehill files are to be used. The "Kategorie" column shows whether the point is for high or medium mountain ranges. The "Massstab" ('Scale') column tells you for which scale the point was entered—the point should only be visible for this scale later on. "Versatz" ('Offset') indicates a possible offset to the east or west for some of the points: For example, the points labelled "-1" above are too close to the west bank of a river and should therefore be shifted still a little further to the west. Finally, the last column provides a scaling factor which is "0.5" for points that are 'sandwiched' between rivers or coastlines (see appendix), "0.8" for points to be displaced from rivers or coastlines (see "Versatz" column) and a random value of 0.9 to 1.1 for all other points (in tenths of a millimetre increments). If necessary, this can be changed in the function code (see appendix).

1. Scale-Dependent Visibility of the Individual Points

So let's first make sure that each point is only displayed for the scale it has been created for or, more precisely, for a range from this scale to the next. *Rule-based symbology* is ideal for this:
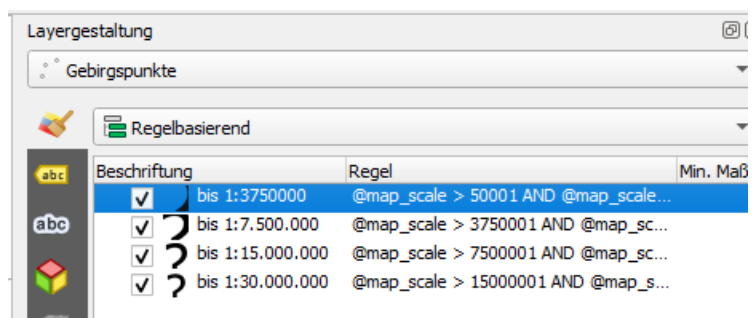
In our example, points were generated for the four scales of 1:3 750 000, 1:7 500 000, 1:15 000 000 and 1:30 000 000. Accordingly, we define four display rules for the ranges from one scale to the next, i.e. initially from an arbitrary lower limit of 50 001 to the scale-reciprocal of 3 750 000, then from 3 750 000 to 7 500 000 and so on.
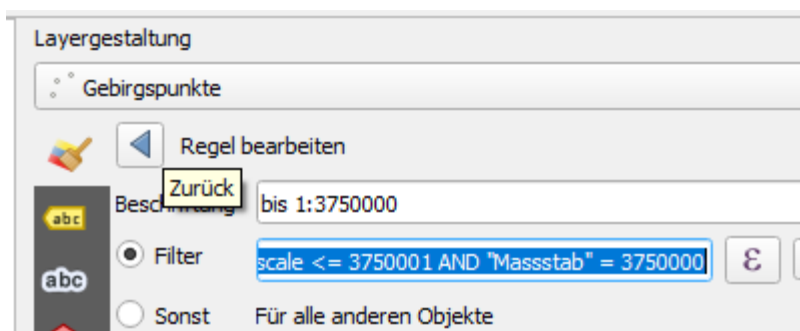
For each of these rules, an expression is entered under *Filter* that regulates the visibility of the points. For the rule "bis 1:3750000" ('up to 1:3750000'), for example, it reads:

@map_scale > 50001 AND @map_scale <= 3750000 AND "Massstab" = 3750000

This means: If the current scale's reciprocal value is greater than 50001 and at the same time less than or equal to 3750000, and if the column "Massstab" ('Scale') shows a value of "3750000" for the respective point, then this point will be displayed; if not, then not.

The rules for the other scale areas are to be formulated accordingly.

2. Assigning the SVG Files

If we double-click on the individual rule and then click again on *SVG Marker*, we will find a field at the bottom where a path to an SVG file can be entered. However, as we don't want to use one and the same hill graphic for all of the points, but rather assign each point the graphic that is specified for it in the attribute table, we must instead press the ε button to the right of the field in order to enter an expression such as the following:

'd:/Kartografie/Signaturen/
Maulwurfshügel/' || "SVG"

It first returns the path to the SVG file, followed by what is shown in the "SVG" column, i.e. the file name, and hence describes a complete path such as "D:\Cartography\Signatures\Molehills\MH31.svg".
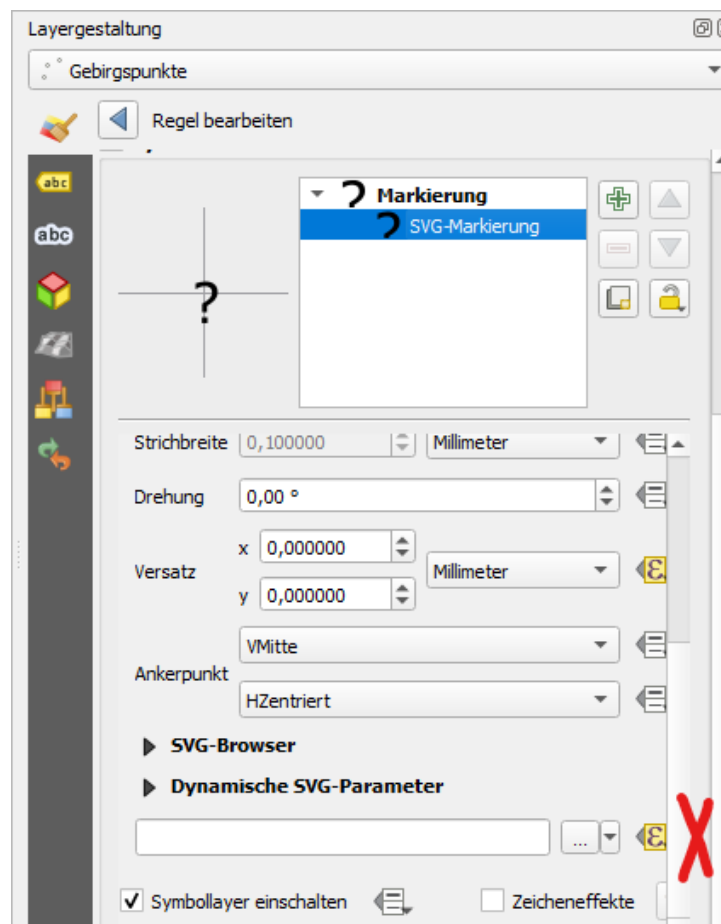
3. Scaling the Hill Graphics Depending on the Scale—and the Latitude

After double-clicking on one of the rules and single-clicking on *SVG marker,* we find below the fields to set the size of the hill graphics. Here we should first set a suitable hill size for the nominal scale of the respective rule. (We should therefore make sure that the map currently is at this scale instead of somewhere between this scale and the next). The specified size refers to the invisible circle that lies in the background of each hill graphic.

Once we have found an appealing map appearance, we should ensure that it is maintained even when the map is scaled up into the area between this and the next scale. The hills should also be scaled up in order to maintain their relative size in proportion to the rest of the map. We can achieve this by klicking the ε button and choosing *Edit* to open the Expression Dialogue, where the following expression can be entered:

```
CASE
   WHEN "Kategorie" = 'hoch' THEN 8 * "Skalierung" * 7500000/$scale * (-0.021 * $y +2.1)
   WHEN "Kategorie" = 'mittel' THEN 7.5* "Skalierung" * 7500000/$scale * (-0.021 * $y +2.1)
   ELSE 1
END
```

Its first lines mean: In the Case when the category is "high", then the point has a size of 8 mm being scaled by the factor from the "Skalierung" column (so that the hill pattern is more varied), then by a scale-dependent factor, which is exactly 1 in the nominal scale, but increases along with the scale, and finally by the bracketed factor to the rear, which takes into account the y-value on the map, i.e. the latitude. Since the coordinate reference systems of these layers have degrees as map units, the distance between the hills is also given in degrees—being a unit which becomes the longer the closer you get to the equator. Thus the map appearance becomes somewhat more attractive if the hillsize increases to the same extent as do the degrees southwards. In the coordinate reference system EPSG:3034 used here, with its conical Lambert projection, the meridians diverge linearly to the south, which makes it easy to find a formula for an equally linear enlargement of the hills (namely "-0.021 * $y +2.1". The absolute member was chosen so that the hills have their original size at about 50 degrees north and from there become smaller northwards and larger southwards.)

The third line is the same for the medium mountains; they are given a basic size of 7.5 mm. The default value of 1 in the penultimate line is only relevant if nothing is entered under "Kategorie" (i.e. never).

4. Distancing the Hills from Rivers, Coastlines, etc.

The *Offset* section can also be found in the place already mentioned several times, where instead of a fixed offset for the hill graphics, you can also set one that is based on the entries in the "Versatz" ('Offset') attribute column. To do this, we press the ε button again, go to *Edit* and enter into the expression editor for example:

<div align="center">to_string("Versatz"*0.7) || ',' || 0</div>

This expression returns an offset in the format "x,y". The entry from the "Versatz" ('Offset') column is taken as the x-value and is multiplied by 0.7 (a factor that can be freely selected). The result is converted into a string in order to be digestible by QGIS. The concatenation continues with a comma as separator and a zero as y-value.

5. Adjusting the Drawing Order for the Current Coordinate Reference System

To give the hill display a perspective impression, the two functions are already formulated such that the hills are sorted by their y-value and that the more southerly hills are only drawn after the more northerly ones, having the front hills cover the rear ones and not the other way round (see appendix). This makes the map already look reasonably neat. However, the y-value used for this is the geographic north/south value, which leads to inadequate results where north is not (or not quite) at the top of the map: namely in the eastern and western peripheral areas, where the meridians run diagonally towards the North Pole, and even more so in maps that are not north-orientated at all, but are, for example, east-oriented (as was not uncommon in the olden days).

To get a really neat map display, we need to ensure that the hills are really rendered from the top to the bottom of the maß. We can do this by ticking the *Control feature rendering order* box at the bottom of the Layer Styling and clicking the button to the right of it. The below window will now open, in which we can enter the following expression:

y(transform($geometry, 'EPSG:4326', 'EPSG:3024'))

It causes the points to be drawn in the order of their y-value, but only after the coordinates used in the layer (EPSG:4326) have been transformed into the coordinates on the screen (EPSG:3024).
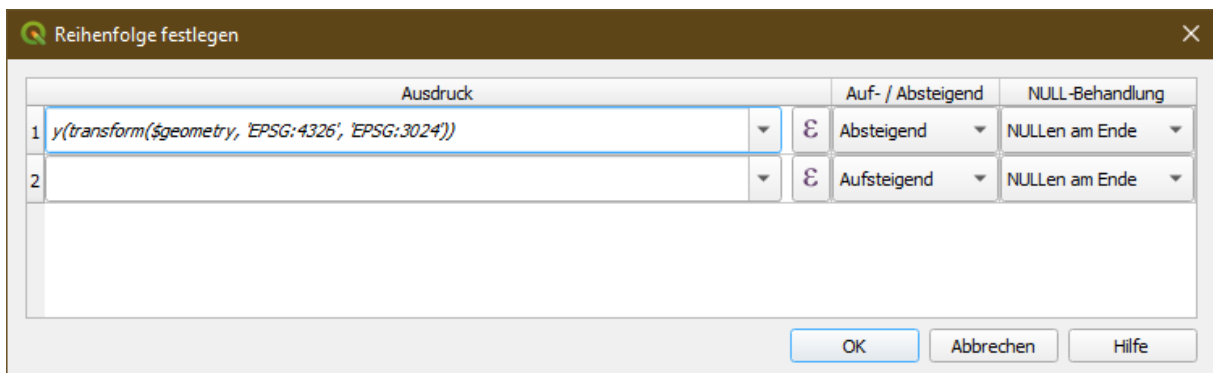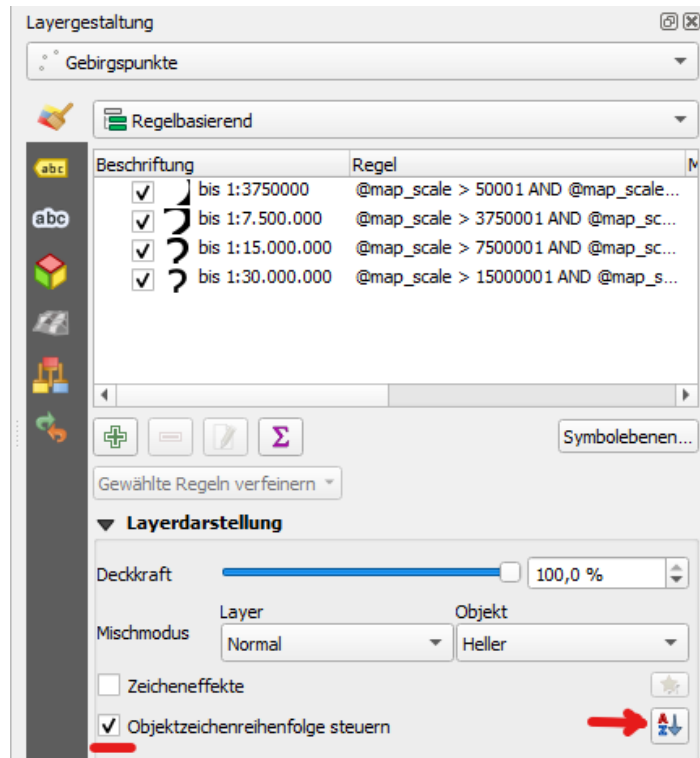
## 6. Setting the Colour and Line Width of the Hills

The final step is to set the colours and line widths of the hill graphics. These are not fixed in the SVG files, but are left open as variables (parameters) whose value can be determined by QGIS (so-called 'parameterised' SVG files). This means that we can adapt the appearance of the hill graphics in QGIS to the requirements of our map.

In order to activate the necessary fields hitherto greyed out in QGIS, we have to pretend we still want to use one and the same graphic for all of the hills and enter the path to a specific hill file into the field marked red on page 10 above:
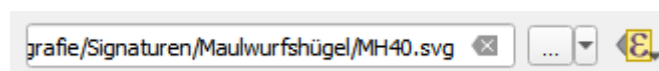
(Which file we choose doesn't matter, as it has no effect, but is still subject to data-defined override.)

Colours and line widths can now be set up using the following fields:
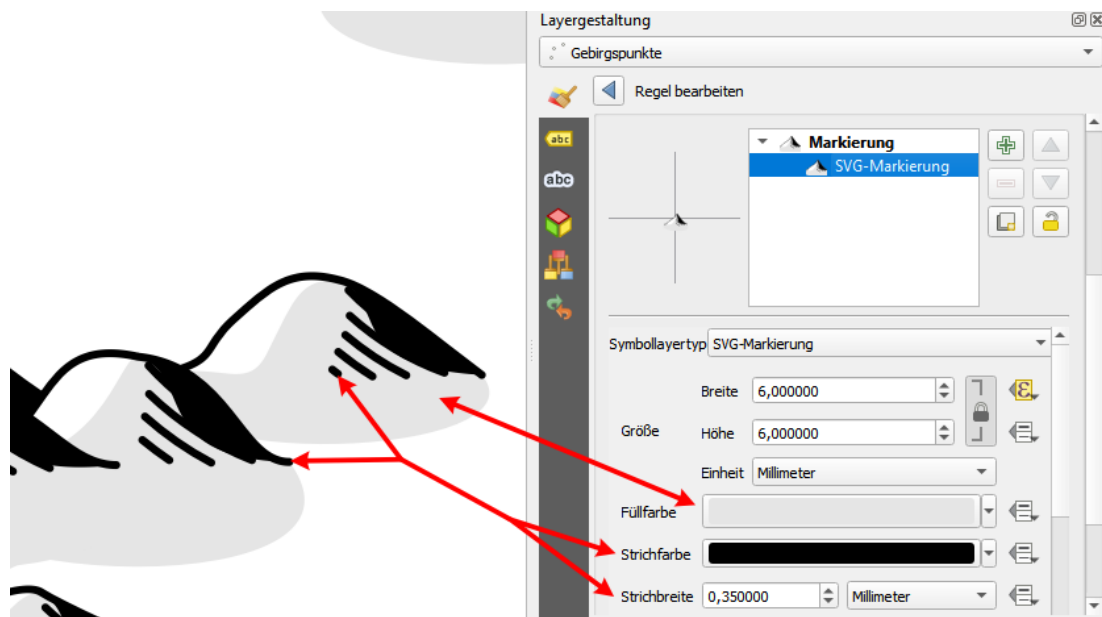
If you click on the *Fill colour* field, a more detailed settings panel opens up, including the colour's opacity. It should be noted here that QGIS thinks it is setting the opacity of the hill colour, whereas in reality this is affecting the black shadow! This misappropriation was necessary because QGIS' use of parameterised SVG files is still somewhat deficient at present.