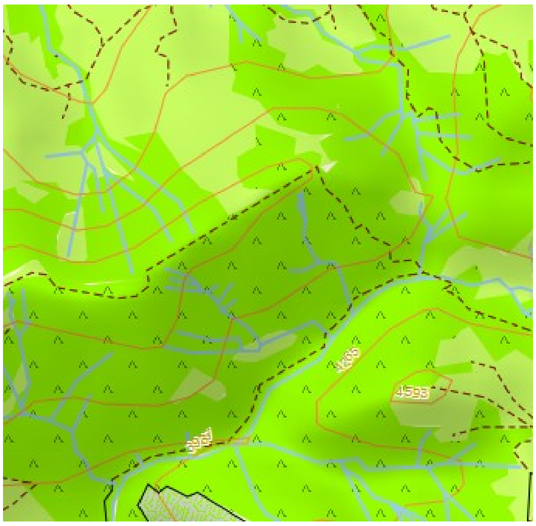
Stand 13.7.2018

Details zum Aufbau und zur Codierung der Garmin-DEM-Daten



Stand 13.7.2018

Einführung

Für den Aufbau eigener Karten sind DEM-Daten nützlich, weil damit z.B. eine „Schummerung“ (Schattierung, hillshading) des Geländes möglich ist. Damit ergibt sich zusätzlich zu den Höhenlinien eine bessere Vorstellung vom Geländeprofil.

Die Garmin-Programme *BaseCamp* und *Mapsource* benötigen DEM-Daten, um ein Höhenprofil eines Tracks darstellen zu können. Die Höhenlinien reichen dafür leider nicht aus.

Ein GPS-Gerät (z.B. Oregon 600) kann damit sogar eine 3-dimensionale Darstellung der Umgebung liefern.

Es wäre deshalb nützlich, z.B. aus den frei verfügbaren Höhendaten der NASA Garmin-DEM-Daten erzeugen zu können. Einige wenige grundlegende Informationen über die Struktur dieser DEM-Daten ist bekannt (siehe https://wiki.openstreetmap.org/wiki/OSM_Map_On_Garmin/DEM_Subfile_Format).

Es ist bekannt, dass Garmin-Karten in einzelnen, unterschiedlich großen „Kacheln“ organisiert sind. Zu jeder Kachel gehören die TRE-, LBL-, RGN- usw. Dateien, die zu IMG-Dateien mit einem eigenen Dateisystem zusammengefasst sein können. Analog kann auch eine DEM-Datei zur Kachel gehören. Diese DEM-Datei organisiert die Daten wiederum in kleinen Kacheln.

Für die Untersuchungen wurde i.W. die „TOPO Deutschland v3“ genauer analysiert und anschließend viele „Trial and Error“-Untersuchungen an einer im Ostseebreich liegenden Kachel durchgeführt. Durch die große Wasserfläche sind hier die meistens Höhendaten konstant.

Stand 13.7.2018

Aufbau der DEM-Datei

Die Beispieldaten sind aus aus der Kachel I0FCC1A3 der „TOPO Deutschland v3“, also der Datei I0FCC1A3.DEM.

Eine DEM-Datei hat zunächst den typischen Garmin-Header:

Adr.	*Format / Länge*	*Inhalt*	*Beispiel*
0x00	UInt16	Länge des **gesamten** Headers (i.A. 0x29)	29 00
0x02	10	„GARMIN DEM“	47 41 52 4D 49 4E 20 44 45 4D
0x0C	1	unbekannt; immer 0x01 (?)	01
0x0D	1	Sperrflag (0x00 oder 0x80)	00
0x0E	7	Datum (2 Byte Jahr, je 1 Byte Monat, Tag, Stunde, Minute, Sekunde)	D9 07 02 13 0F 12 13

Danach folgt der DEM-spezifische Teil des Headers:

Adr.	*Format / Länge*	*Inhalt*	*Beispiel*
0x15	UInt32	Flags; Bit 0 definiert, ob die Zahlenangaben Meter (0) oder Fuß (1) bezeichnen	01 00 00 00
0x19	UInt16	Anzahl der Zoomlevel	02 00
0x1B	4	unbekannt; i. A. 0	00 00 00 00
0x1F	UInt16	Datensatzgröße für die Zoomlevel (immer 0x3C ?)	3C 00 (= 60)
0x21	UInt32	Pointer auf den 1 Zoomlevel-Datensatz	30 9D 0C 00
0x25	4	unbekannt; auch 0x0	01 00 00 00

Die Datensätze der Zoomlevel stehen direkt hintereinander i. A. am Ende der Datei, im Beispiel ab 0xC9D30.

Datensatz für einen Zoomlevel (im Beispiel 1. und 2. Datensatz):

Adr.	*Format / Länge*	*Inhalt*	*Beispiel*
0x00	byte	unbekannt; i.A. 0x00, aber auch 0x01 gesehen	00
0x01	byte	Zoomlevelnummer (0, 1, …)	00 (bzw. 01)
0x02	Int32	waagerechte Pixelanzahl je Kachel (i.A. 0x40)	40 00 00 00 (= 64)
0x06	Int32	senkrechte Pixelanzahl je Kachel (i.A. 0x40)	40 00 00 00 (= 64)
0x0A	UInt32	Höhe – 1 der Nicht-Standard-Kachelzeile am unteren Rand	25 00 00 00
0x0E	UInt32	Breite – 1 der Nicht-Standard-Kachelspalte am rechten Rand	23 00 00 00
0x12	UInt16	Verkleinerungsfaktor für die realen Höhenwerte; i.A. 0x00 (→ 1), aber auch 0x100 (→ 3), 0x200 (→ 5), 0x400 und 0x1400 gesehen	00 00
0x14	UInt32	Anzahl der Standard-Kacheln (64 breit) waagerecht	1E 00 00 00 (= 30)
0x18	UInt32	Anzahl der Standard-Kacheln (64 breit) senkrecht	1C 00 00 00 (= 28)
0x1C	UInt16	Struktur der Kacheldatensätze: Bit 0 und 1: Byteanzahl für den Offset (00 → 1 Byte, 01 → 2 Byte, 10 → 3 Byte und vermutlich 11 → 4 Byte) Bit 2: Byteanzahl der Basis-Höhe (0 → 1 Byte, 1 → 2 Byte) Bit 3: Byteanzahl der Höhendifferenz (0 → 1 Byte, 1 → 2 Byte) Bit 4: bei 1 ex. eine Extrabyte	1E 00 (= 11110) bzw. 06 00 (= 00110)

0x1E	UInt16	Kacheldatensatzgröße	08 00 (bzw. 06 00)
0x20	UInt32	Pointer auf den 1. Kacheldatensatz	29 00 00 00 (bzw. 0E 0D 04 00)
0x24	UInt32	Pointer auf den gesamten Speicherbereich der Höhendaten	41 1C 00 00 (bzw. 20 22 04 00)
0x28	Int32	westliche Grenze der DEM-Datei in Units (360° / 2^{32})	30 3F 7B 09 (= 13,3332°)
0x2C	Int32	nördliche Grenze der DEM-Datei in Units (360° / 2^{32})	50 32 E7 26 (= 54,7075°)
0x30	Int32	Punktabstand senkrecht in Units (360° / 2^{32})	F0 0C 00 00 (= 0,00028°)
0x34	Int32	Punktabstand waagerecht in Units (360° / 2^{32})	F0 0C 00 00 (= 0,00028°)
0x38	Int16	kleinste (Basis-)Höhe	00 00 (bzw. 34 00)
0x3A	Int16	größte Höhe (die Anzeige wird auf diesen Bereich eingegrenzt!)	00 02 (bzw. E1 00)

Zur Umrechnung der Units in Grad kann mit 45 multipliziert und mit 2^{29} = 536870912 dividiert bzw. 0,00000008381903171539306640625 multipliziert werden.

Jeder dieser Datensätze zeigt also auf eine Tabelle von Kacheldatensätzen und auf den Bereich der eigentlichen Höhendaten.

Kacheldatensatz:

<table>
<tr><th>Länge</th><th>Inhalt</th></tr>
<tr><td>1 … 4</td><td>Offset auf die Höhendaten der Kachel (bzgl. des Pointers auf 0x24 des Zoomlevels!); 0, wenn ohne Daten</td></tr>
<tr><td>1 … 2</td><td>Basis-Höhe („signed“, also auch negative Werte möglich)</td></tr>
<tr><td>1 … 2</td><td>max. Höhendifferenz zur Basis-Höhe; 0, wenn ohne Daten (führt allerdings oft zu Darstellungsfehlern!)</td></tr>
<tr><td>0 … 1</td><td>Codierungstyp: 0, …, 6 gesehen
0: größter Wert eingeschlossen
1, 2, 4: größter Wert als „undefiniert“ angezeigt
3, 5, 6: größter Wert und zweitgrößter Wert als „undefiniert“ angezeigt
Bei Länge 0 wird vermutlich für alle Kacheln der Codierungstyp 0 angenommen.

Ein Test für die Typen 1 bis 9 ergab, dass Gebiete ohne Werte grau oder weiß angezeigt werden können.
<table>
<tr><th>Typ</th><th>drittgrößter Wert</th><th>zweitgrößter Wert</th><th>größter Wert</th></tr>
<tr><td>1</td><td></td><td></td><td>grau</td></tr>
<tr><td>2</td><td></td><td></td><td>weiß</td></tr>
<tr><td>3</td><td></td><td>grau</td><td>weiß</td></tr>
<tr><td>4</td><td></td><td></td><td>weiß</td></tr>
<tr><td>5</td><td></td><td>grau</td><td>weiß</td></tr>
<tr><td>6</td><td></td><td>weiß</td><td>weiß</td></tr>
<tr><td>7</td><td>grau</td><td>weiß</td><td>weiß</td></tr>
<tr><td>8</td><td></td><td></td><td>weiß</td></tr>
<tr><td>9</td><td></td><td>grau</td><td>weiß</td></tr>
</table>
Die „weiße“ Anzeige ist möglicherweise für das Hintergrundpolygon gedacht(?).</td></tr>
</table>

Stand 13.7.2018

Für die Beispieldatei ergeben sich folgende Datenbereiche:

0x00	Standard-Header
0x15	DEM-Header
0x29	Tabelle der Kacheldatensätze zum 1. Zoomlevel ((0x1E+1) · (0x1C+1) = 0x383 = 31 * 29 = 899 Datensätze je 8 Byte) z. B. 00 00 00 00 00 00 00 xx → ohne Daten BD 0B 00 27 00 1B 00 00 → Daten auf 0xBBD (+ 0x1C41→ 0x27FE), Basis-Höhe 0x27 (= 39), max. Höhendifferenz 0x1B (= 27), also 39 ...66 50 F0 03 00 00 01 00 02 → Daten auf 0x3F050 (+ 0x1C41→ 0x40C91), Basis-Höhe 0x0, max. Höhendifferenz 0x1
0x1C41	Höhendaten des 1. Zoomlevels
0x40D0E	Tabelle der Kacheldatensätze zum 2. Zoomlevel ((0x1E+1) · (0x1C+1) = 0x383 = 31 * 29 = 899 Datensätze je 6 Byte) z. B. 00 00 00 E1 00 00 → Daten auf 0x0, 0xE1 (= 225), 0x0 → ohne Daten 7A 00 00 5B 00 86 → Daten auf 0x7A (leer), 0x5B (= 91), 0x86 (= 134)
0x42220	Höhendaten des 2. Zoomlevels
0xC9D30	1. Zoomlevel-Datensatz
0xC9D6C	2. Zoomlevel-Datensatz (0xC9D30 + 0x3C)

Die Höhendaten sind als Bit-Stream organisiert, der eine Zahlenmatrix entsprechend der Kachelgröße, also i.A. 64 x 64, beschreibt. Achtung: Bit 7 ist jeweils das 1. Bit, Bit 0 das letzte Bit eines Bytes. Die 12 Bits des Bit-Streams „1 1 0 0 0 0 0 0 1 1 1 1“ ergeben z. B. Die Bytefolge 0x0C 0xF0.

Testkarte und -verfahren

Für die „Trial and Error“-Untersuchungen benötigt man einen hinreichend großen Datenbereich zum „spielen“. Entweder man sucht sich einen passenden Bereich oder man schafft ihn sich selber.

Bei der „TOPO Deutschland v3“ sind die Kacheldaten maximal 3909 Bytes lang. Die größte maximale Differenz für eine Kachel beträgt 0x1972, die größte Basishöhe 0x2119 (außer 0xFFxx bei Basemap), die größte absolute Höhe 9614 bei Basis 0x1EA3 mit maximaler Differenz 0x06EB und Datenlänge 2329 Byte.

93,5% der Kacheln haben als Höhendifferenz-Flag 0x0, der Rest 0x2.

Es gibt 157810 Tiles. Davon enthalten 146938 Daten. Die Summe der Datenlängen beträgt für den Datenbereich 1 165330982 Byte = 157,7MB.

Für die „TOPO Deutschland v3“ sind in der Datei 16564643\I0FCC1A3.DEM die Höhen in Fuß angegeben (1 ft = 30,48 cm). Bei den Untersuchungen sollte deshalb natürlich auch in *Mapsource* die Anzeige in ft erfolgen. Die einzelnen Kacheln sind im Raster 31 x 29 angegeben, also insgesamt 899 Kacheln. Nicht für jede dieser Kacheln liegen auch Daten vor, da einige vollständig über der Ostsee liegen. Jede Kachel ist 64 x 64 Höhendatenpunkte groß und umfasst etwa den Bereich 0,017767° x 0,017767°. Damit wird ein Bereich von 1,1km x 1,5km abgedeckt, die Punktabstände sind also etwa 17,5m bzw. 23,8m.

Die Satzlänge für die Beschreibung der 899 Kacheln beträgt im 1. Datenbereich 8 Byte im 2. 6 Byte. Die Position des jeweiligen Satzes ergibt sich aus

Block1Start + idx · Block1RecordSize

Die Kachel in der 5. Spalte der 10. Zeile hat die Nummer 9 · 31 + 5 = 284. Wegen der Zählung ab 0 ist es die Nummer 283 oder hexadezimal 0x11B. Der Satz für diese Kachel und den 1. Datenbereich steht deshalb auf Position

0x29 + 8 · 0x11B = 0x901

für den 2. Datenbereich auf

0x40d0e + 6 · 0x11B = 0x413B0

Offset(h)	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
00000900	00	58	B6	00	00	00	03	00	00	00	00	00	00	00	00	00
00000910	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00
00000920	00	00	00	00	00	00	00	00	00	64	B6	00	00	00	1A	00
00000930	00	74	B7	00	00	00	AB	00	00	7E	BC	00	37	00	7A	00
00000940	00	B3	C1	00	4D	00	75	00	00	B7	C6	00	73	00	AB	00
00000950	00	E0	CB	00	A1	00	1A	01	00	64	D1	00	21	01	9F	00
00000960	00	F3	D6	00	45	01	80	00	00	B8	DA	00	55	01	87	00

Der Satz für den 1. Datenbereich hat die Bytes 58 B6 00 00 00 03 00 00 (rot markiert). Da die eigentlichen Höhendaten den Offset 0x1c41 haben, ergibt sich für deren Position 0x1c41 + 0xB658 = 0xD299. Die Basishöhe ist 0, die darauf bezogene Maximalhöhe ist 3.

Die folgenden 4 Datensätze (blau markiert) zeigen auf den Datenbereich 0. Sie stehen für die Kacheln in der 6., 7., 8. und 9. Spalte der 9. Zeile.

Füllt man z. B. die folgenden 4 Datensätze (grün markiert) auch noch mit 0, kann deren Datenbereich für Testzwecke mit verwendet werden. Die Daten des 1. Datensatzes nach den „grünen“ beginnen z. B. bei 0xC6B7 + 0x1c41 = 0xE2F8. Damit hätte man also einen Datenbereich von 0xD299 bis 0xE2F8 (0x105F = 4191 Byte) zur freien Verfügung.

Nachdem man sich eine solche Testkarte „gebaut“ hat, können die eigentlichen Versuche beginnen.

Da es um sehr viele Versuche geht, muss man die Untersuchung soweit wie möglich automatisieren. Das betrifft auf jeden Fall die Erzeugung der Testdaten und das Ermitteln der Auswirkung dieser Daten. Für das „Lesen“ der Höhendaten wurde *Mapsource* verwendet. Damit ist naheliegend, dass die gesamte Arbeit zweckmäßigerweise unter Windows erfolgt. Einzelne Schritte werden in Kommandodateien (*.CMD) zusammengefasst.

Es wurden einige „maßgeschneiderte“ Hilfsprogramme in C# geschrieben. Damit ist jederzeit eine leichte Anpassung an neue Erkenntnisse möglich. Außerdem wird das Programm *gmt* (Kommandozeilenversion des GmapTool) verwendet, um die Testkarte aus den Einzeldateien zu erzeugen. Mit dem aus der Unix-Welt bekannten *sed* werden einfache Filteraufgaben erledigt.

I.W. hat man folgenden Ablauf:

Mit dem C#-Programm *Input2* (ja, der Name ist nicht sehr einfallsreich) wird die zu testende Bitfolge in die DEM-Datei geschrieben. Danach wird eine Kommandodatei aufgerufen, z.B.:

```
REM funktioniert nur, wenn Mapsource nicht läuft
del 16564643.img
gmt -j -x -o 16564643.img 16564643\I0FCC1A3.tre 16564643\I0FCC1A3.rgn 16564643\I0FCC1A3.net
           16564643\I0FCC1A3.lbl 16564643\I0FCC1A3.dem
del /Q "%APPDATA%\GARMIN\MapSource\TileCache\*.*"
```

Stand 13.7.2018

```
set MAUSPOS=86 166 1701 166

SimpleProgControl %MAUSPOS% 0 0 64 1 7 tmp 1500 "" "%ProgramFiles(x86)%/Garmin/MapSource.exe"

REM gewünschte Zeilennummern herausfiltern und neu formatieren
sed -n "3,3p" < tmp | sed -e "s/\([0-9]\+\)\t/ : \1 : /" -e s/\t/,/g >> protokoll1.txt
del tmp
```

Die alte 16564643.img wird gelöscht. Mit *gmt* werden die originalen TRE-, RGN-, NET-, LBL- sowie die gepatchten DEM-Daten zusammengefügt. Der Tile-Cache von *Mapsource* sollte **immer** gelöscht werden. Mit dem C#-Programm *SimpleProgControl* wird *Mapsource* gestartet und die Daten werden in die Datei tmp geschrieben. Mit *sed* kann noch eine Filterung erfolgen. Das Ergebnis wird an die Datei protokoll1.txt angehängt.

Das Auslesen der Daten mit *Mapsource* ist etwas schwierig. Es gibt leider nur einen Bereich in der Statuszeile, in der die Höhe des Punktes auf den die Maus zeigt, angezeigt wird.

> Mit spyxx.exe aus „Microsoft Visual Studio“ kann man aber leicht die GUID {7A96B96B-E756-4e42-8274-54CBF24F7944} und den notwendigen Klassennamen „msctls_statusbar32“ ermitteln, um den Text auszulesen. Der Text, z.B. „N54.53305 E13.41344, 17 ft“ gibt dann die Höhe mit einem entsprechenden regulären Ausdruck preis. Nun muss nur noch die Maus schrittweise auf die gewünschten Positionen gesetzt werden.

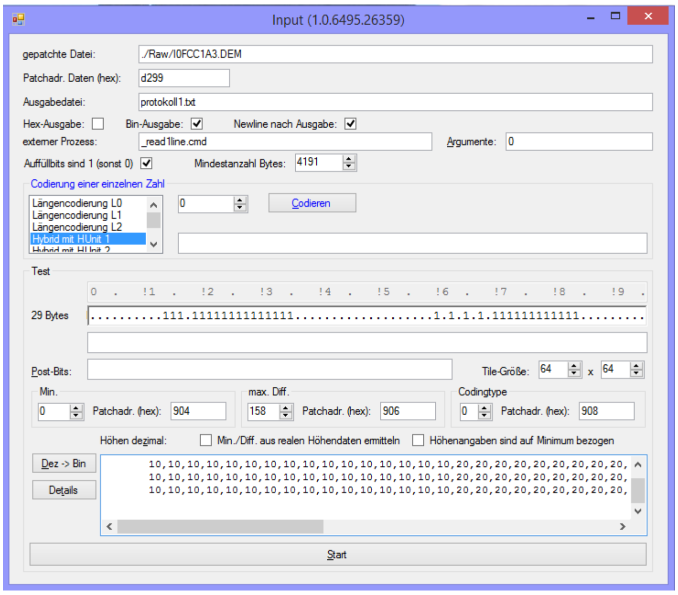
Problematisch ist, dass immer genau der gewünschte Bereich angezeigt werden muss.

SimpleProgControl startet Mapsource. Man kann eine Tastenfolge zur Initialisierung mitschicken z.B. „%AK77777{ENTER}%AzN54.53305 E13.41344{ENTER}“. Damit wird über Alt+A, K der Zoom auf 70m eingestellt und Alt+A, z die Position der Karte festgelegt. Das das Programm im Vollbildmodus startet und wie z.B. die Symbolleisten angeordnet sind muss man aber schon vorher „per Hand“ festlegen.

Mit dem C#-Programm *ShowMousePos* muss man außerdem vorher die Lesepositionen in Bildschirmkoordinaten ermittelt haben. In der Beispiel-Kommandodatei hat die linke obere Mausposition die Koordinaten [86, 166] und die rechte untere [1701, 166]. Aus den folgenden Angaben geht hervor, das die Höhenpunkte von Spalte 0 und Zeile 0 bis Spalte 64 und Zeile 1 zu diesen Mauskoordinaten passen und ausgelesen werden sollen.

Für eine möglichst hohe Genauigkeit des Auslesens sollte *Mapsource* im Vollbildmodus betrieben werden. Für die Kartenanzeige sollte soviel Platz wie möglich vorhanden sein. Die Bildschirmauflösung sollte möglichst hoch sein.

> Bei den Daten der Beispiel-Kommandodatei liegen in O-W-Richtung etwa 25,651 Pixel Abstand zwischen 2 Punkten. Die „Messung“ kann jedoch nur für ganzzahlige Pixelabstände erfolgen. Bei korrekter Rundung liegt der 2. Messpunkt also bei Pixel 26 und damit etwa beim 1,039-fachen der eigentlich gewünschten Entfernung.
> Prinzipiell beträgt der Positions-Fehler ±0,5 Pixel, also etwa ±2%. Bei starken Höhenunterschieden zwischen 2 Messpunkten muss dieser Fehler einkalkuliert werden. Z.B. bei einer Änderung von 50 auf 150 kann der 2. "Messwert" im Bereich 148..152 liegen.



Mit dem Programm *Input2* wird i.W. die DEM-Datei mit einer Bitfolge gepatched, d.h. es werden neue Testdaten eingetragen. Nach dem patchen wird ein weiterer Prozess gestartet, der für das Auslesen der Daten zuständig sein sollte.

Da die Bitfolge i.A. nicht vollständig zum Ausfüllen des letzten Bytes ausreicht, muss die Art der Auffüllbits angegeben werden. Außerdem wird immer eine Mindestanzahl an Bytes geschrieben. Es hat sich bewährt, als Auffüllbit 1 zu verwenden. Die Zusatzbytes sind also 0xFF.

Wegen der besseren Unterscheidbarkeit werden 0-Bits als „.“ geschrieben.

Im Programm ist ein Encoder enthalten, der das aktuelle Wissen über die Codier-Regeln enthält. Deshalb kann man auch direkt Höhenwerte als Dezimalwerte eintragen und codieren lassen.

Wichtig für die Codierung ist die Angabe der maximalen Höhendifferenz.

Stand 13.7.2018

ACHTUNG: Die min. Höhe und die max. Höhendifferenz muss auch im Zoomlevel auf 0x38 und 0x3A eingestellt sein. Andernfalls wird z.B. die dargestellte Höhe auf den Wert auf 0x3A eingegrenzt. Am besten trägt man dort 0x7FFF ein.

Codierung

Komprimierung

Mit einer Speicherung der Höhendaten als 2-Byte-Zahl (16 Bit, also Zahlenbereich 0 ... 65536) könnte man alle Höhen auf der Erde auch in Fuß speichern. Selbst 15 Bit würden ausreichen: 32768 / 3,28084 = 9987m. Bei 64 x 64 Punkten je Kachel wären das $64 \cdot 64 \cdot 16 = 2^{16}$ Bit = 2^{13} Byte = 2^{3} kByte = 8 kB (bzw. 4 kB bei 15-Bit-Zahlen) je Kachel.

Die „TOPO Deutschland v3" hat 157810 Kacheln. Davon enthalten 146938 Daten. Die Summe der Datenlängen beträgt für den Datenbereich 1 165330982 Byte = 157,7 MB. Würde man die oben beschriebene 15-Bit-Speicherung verwenden käme man auf 574 MB. Garmin gelingt also eine Kompression auf etwa 28%.

> Es ist allerdings fraglich, ob es nicht insgesamt effektiver wäre, wenn bei dem heute zur Verfügung stehenden Speichervermögen der SD-Karten eine nur geringe aber einfache Komprimierung verwendet wird, gleichzeitig dadurch aber die Decodierung einfacher und damit die Anzeige schneller wird.

Die anfängliche Vermutung, dass Garmin eines der üblichen und bekannten Kompressionsverfahren verwendet, also Huffman, RLE, Arithmetische Kodierung, LZW usw. usf., hat sich leider nicht bestätigt.

> Liegt das daran, dass diese Verfahren nicht stark genug komprimieren oder wollte man lieber ein „geheimes" Verfahren?

Das verwendete Verfahren ist auf jeden Fall sehr effektiv. Eigentlich sind es mehrere Verfahren. Eines ist besser für größere Zahlen geeignet, ein anderes besser für kleinere. Das Codierverfahren ändert sich nach bestimmten Regeln. Dabei werden jedoch nur die bereits codierten Höhen einbezogen, d.h. es findet keine „Vorausschau" statt.

> Es wäre vermutlich noch effektiver, wenn es optimal auf **alle** aktuell vorhandenen Daten zugeschnitten wäre. Da der Decoder die „zukünftigen" Daten aber noch nicht kennen kann, hätte man zusätzliche Informationen in die codierten Daten aufnehmen müssen, wenn das aktuelle Codierverfahren auf ein anderes „umgeschaltet" wird. Diesen Speicherplatz hat man sich gespart. Das resultierende Verfahren dürfte deshalb weniger effektiv sein, spart aber die „Umschaltinformationen" ein.

Encoder und Decoder müssen natürlich beide nach den gleichen Regeln arbeiten. Um einen Encoder zu schreiben, muss man genau diese Regeln kennen.

Bitstream

Alle Daten werden in Bitfolgen umgewandelt. Diese bilden einen Bitstream der als Byte-Folge, immer beginnend mit dem höchstwertigen Bit, gespeichert wird.

Ist auch nur 1 Bit wegen eines Fehlers im Algorithmus falsch codiert, führt das i. A. dazu, dass der Rest des Bitstreames nicht mehr korrekt decodiert werden kann. Der Rest der DEM-Daten ergibt dann ein völlig unsinniges Bild, oder die Darstellung bricht völlig ab.

Zahlencodierungen

Es gibt 3 grundsätzliche Codierungen: die Längencodierung, die Hybridcodierung und die Binärcodierung. Die Längencodierung ist besonders bei kleinen Zahlen sehr effektiv. Bei größeren Zahlen ist die Hybridcodierung effektiver. Nur für sehr große Zahlen ist die Binärcodierung (deshalb BigBin genannt) notwendig.

Maximallänge einer 0-Bitfolge und maximale Binärbitanzahl

Für die Codierung spielen u.a. auch 0-Bitfolgen eine Rolle. Die Länge dieser Bitfolgen ist jedoch begrenzt und abhängig von der maximalen Höhendifferenz der Kachel.

Für die BigBin-Codierung ist die Anzahl der Binärbits ebenfalls von der maximalen Höhendifferenz der Kachel abhängig.

Für die Plateau-Nachfolger ist die Länge der 0-Bitfolge um die Anzahl der Binärbits + 1 aus der Plateaulänge kürzer!

maximale Höhendifferenz	**maximale Anzahl 0-Bits**	**Wertebereich**						**Binärbits**	**Wertebereich (außer 0) BigBin**
		L0	**L1**	**L2**	**H1**	**H2**	**H3**		
0 .. 1	15	-7 .. 8	-7 .. 8	-8 .. 7	-15 .. 16	-31 .. 32	-47 .. 48		unnötig
2 .. 3	16	-8 .. 8	-7 .. 9	-8 .. 8	-16 .. 17	-33 .. 34	-50 .. 51		unnötig
4 .. 8	17	-8 .. 9	-8 .. 9	-9 .. 8	-17 .. 18	-35 .. 36	-53 .. 54		unnötig
8 .. 15	18	-9 .. 9	-8 .. 10	-9 .. 9	-18 .. 19	-37 .. 38	-56 .. 57	4	-8 .. +8
16 .. 31	19	-9 .. 10	-9 .. 10	-10 .. 9	-19 .. 20	-39 .. 40	-59 .. 60	5	-16 .. +16
32 .. 63	20	-10 .. 10	-9 .. 11	-10 .. 10	-20 .. 21	-41 .. 42	-62 .. 63	6	-32 .. +32
64 .. 127	21	-10 .. 11	-10 .. 11	-11 .. 10	-21 .. 22	-43 .. 44	-65 .. 66	7	-64 .. +64
128 .. 255	22	-11 .. 11	-10 .. 12	-11 .. 11	-22 .. 23	-45 .. 46	-68 .. 69	8	-128 .. +128
256 .. 511	25	-12 .. 13	-12 .. 13	-13 .. 12	-25 .. 26	-51 .. 52	-77 .. 78	9	-256 .. +256
512 .. 1023	28	-14 .. 14	-13 .. 15	-14 .. 14	-28 .. 29	-57 .. 58	-86 .. 87	10	-512 .. +512
1024 .. 2047	31	-15 .. 16	-15 .. 16	-16 .. 15	-31 .. 32	-63 .. 64	-95 .. 96	11	-1024 .. +1024
2048 .. 4095	34	-17 .. 17	-16 .. 18	-17 .. 17	-34 .. 35	-69 .. 70	-104 .. 105	12	-2048 .. +2048
4096 .. 8191	37	-18 .. 19	-18 .. 19	-19 .. 18	-37 .. 38	-75 .. 76	-113 .. 114	13	-4096 .. +4096
8192 .. 16383	40	-20 .. 20	-19 .. 21	-20 .. 20	-40 .. 41	-81 .. 82	-122 .. 123	14	-8192 .. +8192
16384 .. 32767	43	-21 .. 22	-21 .. 22	-22 .. 21	-43 .. 44	-87 .. 88	-131 .. 132	15	-16384 .. +16384

Längencodierung

Prinzipiell wird eine Zahl durch eine 0-Bitfolge und ein abschließendes 1-Bit dargestellt. Es gibt allerdings die 3 verschiedenen Längencodierungen L0, L1 und L2 (siehe Tabelle).

In Abhängigkeit von der maximalen 0-Bitanzahl l ergeben sich folgende Wertebereiche:

$$-l/2 \leq v0 \leq l-l/2$$
$$l/2+1-l \leq v1 \leq l/2+1$$
$$l/2-l \leq v2 \leq l/2$$

Wird der Wertebereich überschritten, muss die passende BigBin-Codierung verwendet werden.

Für Plateau-Nachfolger werden zwei 0-Bit weniger als maximale 0-Bitanzahl verwendet!!

- Man erkennt leicht, dass 3 Zahlen v0, v1 und v2 in den Codierung L0, L1 bzw. L2 die gleiche Länge der 0-Bitfolge haben, wenn folgende Beziehung gilt: $v0=1-v1=-v2$.
- Die Bitlänge l der 0-Bitfolge zur Zahl $v0$ für L0 ist $l=2\cdot|v0|-(sgn(v0)+1)/2$ und für L1 $l=2\cdot|v1-1|+(sgn(v1-1)-1)/2$.
- Aus der Bitlänge l der 0-Bitfolge ergibt sich $v0=(l \bmod 2)\cdot(l+1)/2-((l+1) \bmod 2)\cdot(l/2)$ bzw.

Zahl	Anzahl 0-Bits				0-Bits	Zahl		
	L0	**L1**	**L2**			**L0**	**L1**	**L2**
...					**0**	0	+1	0
+5	9	8	10		**1**	+1	0	-1
+4	7	6	8		**2**	-1	+2	+1
+3	5	4	6		**3**	+2	-1	-2
+2	3	2	4		**4**	-2	+3	+2
+1	1	0	2		**5**	+3	-2	-3
0	0	1	0		**6**	-3	+4	+3
-1	2	3	1		**7**	+4	-3	-4
-2	4	5	3		**8**	-4	+5	+4
-3	6	7	5		**9**	+5	-4	-5
-4	8	9	7		**10**	-5	+6	+5
-5	10	11	9		**11**	+6	-5	-6
...					**12**	-6	+7	+6

Stand 13.7.2018

$$v1=-((l \bmod 2)\cdot(l+1)/2-((l+1) \bmod 2)\cdot(l/2)-1) \quad .$$

Hybride Codierung

Die hybride Codierung ist eine Kombination einer speziellen Längencodierung und einer Binärcodierung. Die Längencodierung verwendet die Einheit *hunit* („Heightunit"). Diese ist immer eine 2er Potenz 2^i und wird nach bestimmten Regeln automatisch angepasst. *hunit* ist gleichzeitig die größte Zahl, die mit dem Binärteil dargestellt werden kann.

Die Codierung erfolgt in der Form $0_{s-1}...0_0\,1\,b_{ld(hunit)-1}...b_0\,v$. Die führenden 0-Bits stehen für die Längencodierung. Dann folgt ein 1-Bit. Danach folgen die Binärbits und zum Abschluss ein Vorzeichenbit (1 für positive, 0 für negative Werte). Die Zahl ergibt sich aus $d=(2\cdot v-1)\cdot\left(s\cdot hunit+\left(\sum_{i=0}^{ld(hunit)-1} b_i\cdot 2^i\right)+v\right)$.

In Abhängigkeit von der maximalen 0-Bitanzahl l ergibt sich folgender Wertebereich:

$$-(l+1)\cdot hunit+1 \;\le\; d \;\le\; (l+1)\cdot hunit \quad .$$

Wird der Wertebereich überschritten, muss die passende BigBin-Codierung verwendet werden.

Für Plateau-Nachfolger wird ein 0-Bit weniger als maximalen 0-Bitanzahl verwendet!!

Beispiele:
hunit=4, „`...11.1`"
3x0-Bits Längencodierung → 3 * 4
1x1-Bit Trennzeichen
1x1-Bit für 2^1
1x0-Bit für 2^0
1x1-Bit Vorzeichen→ positiv
→ (2 * 1 - 1) * (3 * 4 + 1 * 2^1 + 0 * 2^0 + 1) = 14

hunit=1, „`11`"
0x0-Bits Längencodierung → 0
1x1-Bit Trennzeichen
1x1-Bit Vorzeichen→ positiv
→ (2 * 1 - 1) * (0 * 1 + 1) = 1

hunit=1, „`.1.`"
1x0-Bits Längencodierung → 1 * 1
1x1-Bit Trennzeichen
1x0-Bit Vorzeichen → negativ
→ (2 * 0 - 1) * (1 * 1 + 0) = -1

hunit=1, „`1.`"
0x0-Bits Längencodierung → 0
1x1-Bit Trennzeichen
1x0-Bit Vorzeichen → negativ
→ (2 * 0 - 1) * (0 * 1 + 0) = 0

Bis auf 2 Sonderfälle für die Werte 0 und 1 ist die Hybridcodierung immer besser als die einfache L0- bzw. L1-Längencodierung, d.h. es werden weniger Bits für die Codierung benötigt. Vermutlich treten aber gerade diese Sonderfälle in der Praxis häufig genug auf, so dass sich die einfache Längencodierung eben doch lohnt.
Warum wird aber eine Hybridcodierung statt einer reinen Binärcodierung verwendet? Die darin enthaltene Längencodierung ist ja vergleichsweise uneffektiv.
Der Wertebereich bei der Binärcodierung ist immer begrenzt. Z.B. können mit 3 Bit grundsätzlich nur 8 verschiedene Werte codiert werden, mit 8 Bit 256 Werte. Um sicher zu gehen, dass auch eine sehr hohe, steile Klippe codiert werden kann, müssten vermutlich Werte bis ±1000 (Fuß) verfügbar sein. In Binärcodierung müßten für jeden Wert also 11 Bit (±1024) verwendet werden. Am weitaus häufigsten kommen jedoch sehr kleine Werte vor. Die meisten Bits der Binärcodierung wären also verschwendet. Selbst bei *hunit* = 1 können in Hybridcodierung mit 11 Bit schon Werte von -11 bis +12 codiert werden. Bei kleineren Werten ist die Hybridcodierung dann sogar effektiver als die Binärcodierung.
Würde man eine variable Bitanzahl für die Binärcodierung verwenden, müsste vor jedem Binärwert noch eine Festlegung der Bitanzahl erfolgen. Dafür wären aber auch 4 Bit (1 bis 16) nötig. Insgesamt wären dann je Wert 4+2=6 bis 4+11=15 Bit nötig. Mit 8 Bit erreicht man z.B. einen Wertebereich von 2^4=16, bei Hybridcodierung 12, bei 7 Bit 2^3=8 bzw. 10. Bei Werten im Bereich ±5 ist die Hybridcodierung also auch effektiver.
→ Bei häufig relativ kleinen Werten ist die Hybridcodierung effektiver als die reine Binärcodierung.

Binärcodierung für große Zahlen

Diese Binärcodierung ist zwar eine reine Binärcodierung. Da diese Codierung aber nicht auf Grund bestimmter Regeln, sondern aus Sicht des Decodierers „willkürlich" angewendet wird, muss vorher eine Kennung in Form einer „ungültigen" 0-Bitfolge verwendet werden. Diese 0-Bitfolge muss mindestens 1 Bit länger als die maximalen 0-Bitanzahl sein. Danach folgt ein 1-Bit und danach eine Reihe von Binärbits, deren Anzahl ebenfalls von der maximalen Höhendifferenz abhängig ist (siehe Tabelle oben). Der Zahlenwert wird durch die Binärbitfolge $b_n \ldots b_0 b_v$ codiert.

Es gibt prinzipiell die 3 BigBin-Codierungen BigBin, BigBin1 und BigBin2. BigBin wird bei Bedarf an Stelle der L0-Codierung oder der Hybridcodierung, BigBin1 an Stelle der L1-Codierung und BigBin2 an Stelle der L2-Codierung verwendet. Der Wert 0 kann nicht codiert werden, aber dass ist offensichtlich auch nicht nötig.

Die Codierung erfolgt folgendermaßen:

$$d_{H,L0} = -(2 \cdot b_v - 1) \cdot \left(1 + \sum_{i=0}^{n} b_i \cdot 2^i\right)$$

$$d_{L1} = (2 \cdot b_v - 1) \cdot \left(1 + \sum_{i=0}^{n} b_i \cdot 2^i\right) + 1$$

$$d_{L2} = (2 \cdot b_v - 1) \cdot \left(1 + \sum_{i=0}^{n} b_i \cdot 2^i\right)$$

Wie man leicht sieht genügt die Codierung für BigBin, wenn man die zu codierenden Werte bei Bedarf vorher konvertiert. Statt *v1* muss man dann 1 – *v1* und statt *v2* muss man -*v2* verwenden.

Der Wertebereich (außer 0) ergibt sich mit der maximalen Höhendifferenz *max* aus $-2^{integer(ld(max))} \leq d \leq 2^{integer(ld(max))}$.

Für Plateau-Nachfolger wird ein 0-Bit weniger in der Kennung verwendet als für Standardwerte!!

> Für eine Maximalhöhe 158 ergeben sich *integer*(*ld*(158)) = *integer*(7,3) = 7 Bits. Der Wertebereich ist also $-2^7 \leq d \leq 2^7$, also $-128 \leq d \leq 128$.
> Bei einer Maximalhöhe 158 erfolgten Versuche mit 23, 24 und 25 führenden 0-Bits. Die zusätzlichen 1 bzw. 2 0-Bits hatten keinen erkennbaren Einfluss. Nach dem 1-Bit folgte immer die Binärbitfolge mit insgesamt 8 Bit. Insofern ist klar, dass das 1-Bit als Endekennung der 0-Bitfolge nötig ist. Unklar ist jedoch, welche Auswirkungen die scheinbar unnötigen zusätzlichen 0-Bits haben. Es ist kaum zu erwarten, dass sich GARMIN hier nicht etwas gedacht hat. (?)
> Bei einer maximalen Höhendifferenz von ≤ 8 wird mit 17 0-Bits der gesamte Wertebereich abgedeckt. Es ist keine BigBin-Codierung nötig.

Wraparound

In jedem Kacheldatensatz ist mit der maximalen Höhendifferenz die Größe des verwendeten Wertebereichs gespeichert (0 bis maximale Höhendifferenz). Wird bei der Berechnung dieser Wertebereich nach oben oder unten überschritten, erfolgt ein „Wraparound“. Dadurch wird der dieser Wert wieder im gültigen Wertebereich abgebildet:

$$h_w = h_i + n \cdot (max + 1) \qquad n \in G, n \text{ so dass} 0 \leq h_j \leq max$$

Höhe ohne Wrapping (Max. 9)	Höhe mit Wrapping
11	1
10	0
9	9
8	8
1	1
0	0
-1	9
-2	8

Wenn man durch Wraparound einen Datenwert erhält, der mit weniger Bits codiert werden kann weil er kleiner ist oder weil eine BigBin-Codierung vermieden wird, wird immer der Wraparound angewendet.

Damit sich für die Längencodierung ein Wraparound lohnt, muss die Bitanzahl mit Wraparound kleiner sein als ohne.

Bitanzahl bei L0

ohne Wrap: $2\cdot|v|-(sgn(v)+1)/2$

Wrap für pos. Zahl: $2\cdot|v-(max+1)|-(sgn(v-(maxheight+1))+1)/2 \;=\; 2\cdot|v-max-1|-(sgn(v-max-1)+1)/2$

also $2\cdot|v-max-1|-(sgn(v-max-1)+1)/2 \;<\; 2\cdot|v|-(sgn(v)+1)/2$

$\text{wegen}\, 0<v:$

$2\cdot-(v-max-1)-(-1+1)/2 \;<\; 2\cdot v-(1+1)/2$

$-2\cdot v+2\cdot max+2 \;<\; 2\cdot v-1$

$2\cdot max+3 \;<\; 4\cdot v$

$\boldsymbol{(2\cdot max+3)/4<v}$

Wrap für neg. Zahl: $2\cdot|v+(maxheight+1)|-(sgn(v+(max+1))+1)/2 \;=\; 2\cdot|v+max+1|-(sgn(v+max+1)+1)/2$

also $2\cdot|v+max+1|-(sgn(v+max+1)+1)/2 \;<\; 2\cdot|v|-(sgn(v)+1)/2$

$\text{wegen}\, v<0:$

$2\cdot(v+max+1)-(1+1)/2 \;<\; 2\cdot-v-(-1+1)/2$

$2\cdot v+2\cdot max+2-1 \;<\; -2\cdot v$

$4\cdot v \;<\; -(2\cdot max+1)$

$\boldsymbol{v \;<\; -(2\cdot max+1)/4}$

Analog erfolgt die Ableitung für L1 und L2 und man erhält:

L0: $v \;<\; -(2\cdot max+1)/4$ bzw. $(2\cdot max+3)/4 \;<\; v0$

L1: $v \;<\; -(2\cdot max-1)/4$ bzw. $(2\cdot max+5)/4 \;<\; v1$

L2: $v \;<\; -(2\cdot max+3)/4$ bzw. $(2\cdot max+1)/4 \;<\; v2$

Für die Hybridcodierungen gilt: $d \;<\; -(max-1)/2$ oder $(max+1)/2 \;<\; d$

Beispiel: *max* = 255

Damit ergibt sich:

L0: $v \;<\; -127{,}75$ bzw. $128{,}25 \;<\; v0$ also $v\leq-128$ bzw. $129\leq v$

L1: $v \;<\; -127{,}25$ bzw. $128{,}75 \;<\; v1$ also $v\leq-128$ bzw. $129\leq v$

L2: $v \;<\; -128{,}75$ bzw. $127{,}75 \;<\; v2$ also $v\leq-129$ bzw. $128\leq v$

Ein Wert *v* = 240 wird zu -16. Allerdings überschreitet dieser Wert auch die zulässigen Wertebereiche von -11 .. 11, -10 .. 12 und -11 .. 11, so daß in diesem Fall auf die BigBin-Codierungen ausgewichen werden muss.

Ein Wert *v* = 250 wird zu -6 und damit auf jeden Fall im gültigen Wertebereich liegen.

Ein Wert *v* = 129 wird zu -127 und passt damit in den Wertebereich von BigBin.

Beispiel: *max* = 128

Damit ergibt sich:

L0: $v \;<\; -64{,}25$ bzw. $64{,}75 \;<\; v0$ also $v\leq-65$ bzw. $65\leq v$

L1: $v \;<\; -63{,}75$ bzw. $65{,}25 \;<\; v1$ also $v\leq-64$ bzw. $66\leq v$

L2: $v \;<\; -64{,}75$ bzw. $64{,}25 \;<\; v2$ also $v\leq-65$ bzw. $65\leq v$

Ein Wert *v* = 128 wird zu -1 und passt damit im Gegensatz zu 128 in den gültigen Wertebereich.

Ein Wert *v* = 65 wird zu -64 und passt nicht mehr in den gültigen Wertebereich, stellt für BigBin aber kein Problem dar.

Es scheint also sinnvoll zu sein, wenn möglich erst einen Wraparound auszuführen und danach die Grenzen des Wertebereiches zu überprüfen. Unter Umständen kann durch einen Wraparound eine BigBin-Codierung vermieden werden.

Umschaltung der Codierart

Leider kann nicht frei gewählt werden, welche Codierart zu verwenden ist. Die einzige Ausnahme scheinen die BigBin-Codierungen zu sein. Sie sind allerdings sehr uneffektiv.

Für die anderen Codierarten sind genaue Regeln festgelegt, wie aus den bisher codierten bzw. decodierten Werten die Codierart für den nächsten Wert ermittelt wird. Außerdem unterscheiden sich die Regeln für Standardwerte, Plateau-Nachfolger mit $ddiff(n, m) \neq 0$ und Plateau-Nachfolger mit $ddiff(n, m) = 0$. Dabei bilden jede dieser 3 Wertearten eine Wertegruppe für sich.

In der Annahme, dass eine hybride Codierung erfolgt, wird zunächst *hunit* bestimmt. Ist *hunit* kleiner als 1, wird eine Längencodierung verwendet. Es muss dabei aber noch zwischen L0, L1 und L2 entschieden werden.

hunit-Bestimmung

Im Prinzip wird aus den bisherigen Daten auf eine bestimmte Art eine Summe gebildet und diese Summe in Relation zur Anzahl *vg* der bisherigen Werte gesetzt. Für den so erhaltenen Wert kann man aus einer Tabelle *hunit* ablesen. Da für den ersten Wert einer Gruppe noch keine Vorgänger existieren, wird <u>immer</u> die Hybridcodierung verwendet. *hunit* wird aus der maximalen Höhendifferenz der Kachel abgeleitet (siehe Tabelle).

Es wurden 2 Varianten gefunden.

<u>Variante HStd (Hybridcodierung für Standardwerte und Plateau-Nachfolger mit *ddiff* ≠ 0):</u>

hunit / vg	**1**	**2**	**3**	**4**	**5**	**...**	**63**
1	1	2	3	4	5		63
2	3	5	7	9	11		127
4	7	11	15	19	23		255
8	15	23	31	39	47		511
16	31	47	63	79	95		1023
32	63	95	127	159	191		2047
64	127	191	255	319	383		4095
...							

Basistabelle HS der *hunit*-Wechsel (für Start-*hunit* = 1)

Bei der Variante für die Standardwerte wird die Summe aus den Absolutwerten der Daten gebildet.

In der Basistabelle sind einige experimentell gefundene Minimalsummen für die zugehörige *hunit* und Anzahl der bisherigen Datenwerte (Vorgänger-anzahl *vg*) eingetragen. Die Tabellenwerte ergeben sich aus

$$\boxed{minsumHStd_{hunit,\,vg} = (vg+1)\cdot hunit - 1}\ .$$

Sie gelten allerdings nur, wenn die maximale Höhendifferenz der Kachel nicht größer als 158 ist!

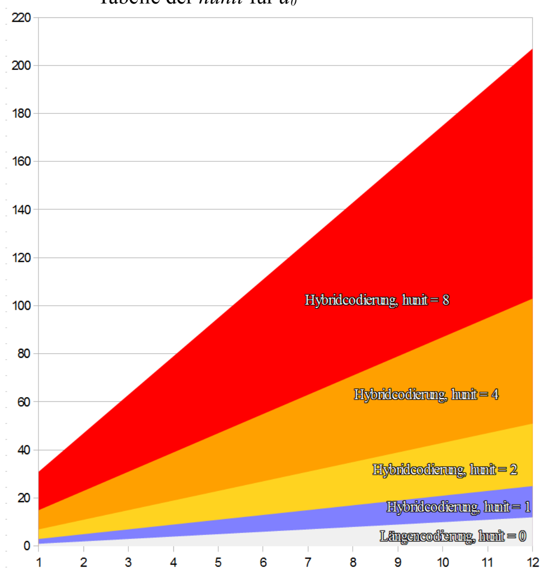
Reicht *minsumHStd* nicht wenigstens zum Erreichen von hunit = 1 aus (also bei $minsumHStd_{vg} < (vg+1)\cdot 1 - 1 \quad\rightarrow\quad minsumHStd_{vg} < vg$), wird der aktuelle Wert mit L0 oder L1 codiert.

maximalen Differenz zum Basiswert		**hunit für d_0**
Hex	**dez**	
… 0x9e	… 158	$2^0 = 1$
0x9f … 0x11e	159 … 286	$2^1 = 2$
0x11f … 0x21e	287 … 542	$2^2 = 4$
0x21f … 0x41e	543 … 1054	$2^3 = 8$
0x41f … 0x81e	1055 … 2078	$2^4 = 16$
0x81f … 0x101e	2079 … 4126	$2^5 = 32$
0x101f … 0x201e	4127 … 8222	$2^6 = 64$
0x201f … 0x401e	8223 … 16414	$2^7 = 128$
0x401f … 0x7fff	16415 … 32767	$2^8 = 256$

Tabelle der *hunit* für d_0

Beispiel:
Für die Datenwerte „1, -3, 2, 4“ ist die Summe der Absolutwerte $asum = 10$ und $vg = 4$.
Für den nächsten Datenwert ergibt sich wegen $9 \leq 10 < 19 \rightarrow hunit = 2$.

hunit scheint auf maximal 256 begrenzt zu sein. Theoretisch ist die Größe von *hunit* aber natürlich nicht begrenzt. Denkbar wäre eine (praktisch wohl unbedeutende) Grenze bei 2 Byte, d.h. 65535. Es ist natürlich auch möglich, dass Garmin einen generellen Grenzwert von 256 definiert hat.



minsumHS in Abhängigkeit von *vg* und *hunit*

Aus der Gleichung für $minsumHS_{hunit,\,vg}$ folgt

$$hunit_{vg} = \frac{minsumHStd_{vg}+1}{vg+1}$$

. Gleichzeitig ist *hunit* aber immer die größte passende 2er Potenz. Deshalb kann *hunit* folgendermaßen berechnet werden:

$$\boxed{hunit = 2^{integer\left(ld\left(\frac{asum_{vg}+1}{vg+1}\right)\right)}}$$

(„integer“ liefert den ganzzahligen Anteil der Zahl).

Beispiele:
Datenwerte „5, 7, 3“ →

$$hunit = 2^{integer\left(ld\left(\frac{5+7+3+1}{3+1}\right)\right)} = 2^{integer(ld(4))} = 2^2 = 4$$

Datenwerte „5, 6, 3“ → $hunit = 2^{integer\left(ld\left(\frac{5+6+3+1}{3+1}\right)\right)} = 2^{integer(ld(3{,}75))} = 2^1 = 2$
Datenwerte „1, 1, -1“ →
$hunit = 2^{integer\left(ld\left(\frac{1+1+1+1}{3+1}\right)\right)} = 2^{integer(ld(1))} = 2^0 = 1$

Wie bereits erwähnt, gilt die Basistabelle nur, wenn die maximalen Höhendifferenz der Kachel nicht größer 158 ist. Bei höheren Werten verringern sich alle Werte der Basistabelle um ein *hunitdelta* dessen Größe von der maximalen Höhendifferenz abhängig ist.

Für Höhendifferenzen im Bereich 0x9f bis 0x11e wird dieser in 2 gleichgroße Bereiche aufgeteilt. Für 0x9f bis 0xde werden alle Werte der Basistabelle um 1 verringert. Für den 2. Bereich von 0xdf bis 0x11e werden alle Werte der Basistabelle um 2 verringert. Das gleiche Prinzip gilt für größere *hunit*. Da mit jeder Verdopplung des *hunit* auch die Gesamtbereichsbreite verdoppelt wird, gleichzeitig aber auch die Anzahl der Teilbereiche verdoppelt wird, bleibt die Breite eines Teilbereiches immer konstant 0x40. Für jeden Teilbereich ab 0x9f werden alle Werte der Basistabelle jeweils um 1 verringert.

Ist die maximale Höhendifferenzen größer oder gleich 158 = 0x9f, ergibt sich *hunitdelta* wie folgt:

$hunitdelta = \text{integer}\left(\frac{\max(0, maxdiff - \text{0x5f})}{\text{0x40}}\right)$ („max“ liefert die größere Zahl → für *maxdiff* < 0x9f wird *hunitdelta* 0).

Damit gilt insgesamt

$$minsumHStd_{hunit,\,vg} = hunit \cdot (vg+1) - 1 - hunitdelta$$

$$minsumHStd_{hunit,\,vg} = hunit \cdot (vg+1) - 1 - \text{integer}\left(\frac{\max(0, maxdiff - \text{0x5f})}{\text{0x40}}\right)$$

bzw. dezimal:

$$minsumHStd_{hunit,\,vg} = hunit \cdot (vg+1) - 1 - \text{integer}\left(\frac{\max(0, maxdiff - 95)}{64}\right)$$

Negative Tabellenwerte bedeuten, dass auf keinen Fall eine Hybridcodierung erfolgt.

Beispiel:
Bei maxdiff = 0x816 = 2070 ist *starthunit* = 16. Für d_2 gilt:

$asum_1 = hunit_2 \cdot 2 - 1 - \text{integer}((\max(95, 2070) - 95) \div 64)$
$asum_1 = hunit_2 \cdot 2 - 1 - \text{integer}(30{,}86)$
$asum_1 = hunit_2 \cdot 2 - 31$

$hunit_2$ = 32 gilt also ab $asum_1$ = 33 (33 = 32 * 2 - 31) und nicht erst ab huv_1 = 63 wie in der Basistabelle.
$hunit_2$ = 16 gilt ab $asum_1$ = 1 (1 = 16 * 2 - 31).
$hunit_2$ = 8 gilt ab $asum_1$ = -15. Da es sich aber um die Summe von Absolutwerten handelt, gilt das nur für 0.
Kleinere *hunit* und damit auch eine Längencodierung sind für d_2 offensichtlich nicht möglich.

Beispiel:
Bei *maxdiff* = 0x21f ist *starthunit* = 8 und *hunitdelta* = 7.
Für i < 4 kann *hunit* = 1 nicht erreicht werden.
Für i = 1 kann *hunit* = 1 und 2 nicht erreicht werden.
Die negativen Werte wurden zwar berechnet und z.T. eingetragen, können aber praktisch nicht erreicht werden. Sie müssten eigentlich durch 0 ersetzt werden.

hunit / i	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	...
1				-3	-2	-1	0	1	2	3	
2		-2	0	2	4	6	8	10	12	14	
4	0	4	8	12	16	20	28	...			
8	8	16	24	32	40	48					
...											

verschobene Basistabelle:
maxdiff = 0x21f → starthunit = 8, hunitdelta = 7

Es läßt sich eine allgemeine Gleichung formulieren, mit der *hunit* aus dem $asum_{vg}$ der Vorgänger und der Anzahl *vg* der Vorgänger ermittelt wird:

$$\boxed{hunit = 2^{integer\left(ld\left(\frac{asum_{vg} + 1 + \text{integer}\left(\frac{\max(0, maxdiff - 95)}{64}\right)}{vg+1}\right)\right)}}$$

Achtung!
Vermutlich verwendet Garmin für $asum_{vg}$ nur eine 2-Byte-Variable. In seltenen Fällen erreicht die Summe die Wertebereichsgrenze. Falls nach der Neuberechnung von $asum_{vg}$ gilt $asum_{vg} + 1 + \text{integer}\left(\frac{\max(0, maxdiff - 95)}{64}\right) \geq 65535$ wird $asum_{vg}$ um 0x10000 = 65536 verringert.

Stand 13.7.2018

Variante HPl0 (Hybridcodierung für Plateau-Nachfolger mit *ddiff* = 0):

Diese Variante funktioniert ähnlich zu HStd. Allerdings ist die Summenbildung geringfügig anders. Bei nichtpositiven Werten, also auch bei 0, wird der Betrag des um 1 verringerten Wertes verwendet. Außerdem wird eine leicht veränderte Tabelle verwendet.

hunit / vg	**1**	**2**	**3**	**4**	**5**	**...**
1	1	3	4	6	7	
2	3	6	8	11	13	
4	7	12	16	21	25	
8	15	24	32	41	49	
16	31	48	64	81	97	
...						

Basistabelle HP0 der *hunit*-Wechsel (für Start-*hunit* = 1)

Wie man leicht sieht, ergeben sich die Tabellenwerte aus der Standardtabelle, wenn man die Werte der Spalten jeweils um 0, 1, 1, 2, 2, 3, … usw. erhöht. Man muss also nur integer(*vg*/2) addieren.

$$\boxed{\begin{aligned} minsumHPl0_{hunit,vg} &= minsumHStd_{hunit,vg} + integer(vg/2) \\ minsumHPl0_{hunit,vg} &= (vg+1)\cdot hunit - 1 + integer(vg/2) \end{aligned}}$$

und *hunit* damit aus

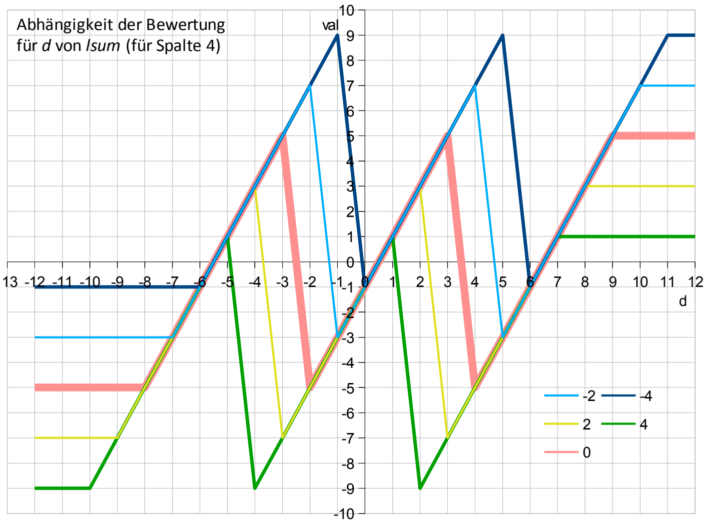
$$hunit = 2^{integer\left(ld\left(\frac{asumspec_{vg}+1-integer\left(\frac{vg}{2}\right)}{vg+1}\right)\right)}$$

Art der Längencodierung

Wie bereits erwähnt, wird die Längencodierung immer dann verwendet wenn bei der spezifischen Summenbildung nicht der notwendige Mindestwert für *hunit* = 1 erreicht wird.

<u>Standardwerte</u>

Für Standardwerte wird eine Summe $valsum_{vg}$ von speziellen „Bewertungen" aller *vg* Vorgänger gebildet. Die Datenwerte werden also zunächst neu „bewertet" und diese „Neubewertungen" werden dann summiert. Ist $valsum_{vg}$ größer als 0, wird die L1-Codierung verwendet, sonst die L0-Codierung.



Das Prinzip der Bewertung wird im Diagramm als Beispiel für den 5. Datenwert, also 4 Vorgänger gezeigt. Für relativ große bzw. kleine *d* ist die Bewertung konstant und hängt nur von $valsum_{vg}$ ab. Für $valsum_{vg} = 4$ ist z. B. die Bewertung für alle $d \geq 7$ konstant 1, für alle $d \leq -10$ konstant -9. Dazwischen gibt es allerdings einen „Zickzack"-Bereich.

Durch viele Versuche wurde folgender Zusammenhang ermittelt:

Region	Bereich für d	Berechnung von *val*
4	$4 - (valsum_{vg} - 3 \cdot vg) / 2 \leq d$	$-valsum_{vg} + (vg + 1)$
3	$2 - (valsum_{vg} - vg) / 2 \leq d < 4 - (valsum_{vg} - 3 \cdot vg) / 2$	$2 \cdot d - 2 \cdot vg - 5$
2	$0 - (valsum_{vg} + vg) / 2 \leq d < 2 - (valsum_{vg} - vg) / 2$	$2 \cdot d - 1$
1	$-2 - (valsum_{vg} + 3 \cdot vg) / 2 \leq d < 0 - (valsum_{vg} + vg) / 2$	$2 \cdot d + 2 \cdot vg + 3$
0	$d < -2 - (valsum_{vg} + 3 \cdot vg) / 2$	$-valsum_{vg} - (vg + 1)$

Der „Abstand" dieser Grenzwerte ist unabhängig von $valsum_{vg}$ und beträgt jeweils 2 + *vg*.

Die scheinbar nicht „in der Reihe" liegenden *val* im obersten und untersten Bereich sind einfach die (konstanten) Werte für das nächstgrößere bzw. nächstkleinere d aus dem aus dem angrenzenden Bereich.

Für betragsmäßig relativ kleine Werte *d* im Bereich $-(valsum_{vg} + vg) / 2 \leq d < 2 - (valsum_{vg} - vg) / 2$ hängt die Bewertung nur von *d* selbst, aber nicht von $valsum_{vg}$ oder *vg* ab.

Im Bereich für die größten *d* ergibt sich als neue *valsum* immer *vg*+1, im Bereich für die kleinsten *d* immer -(*vg*+1). In den 3 Bereichen dazwischen nimmt *valsum* einen Wert im Bereich -(*vg*+1) … (*vg*+1) an. Durch den beschriebenen Algorithmus verläßt *valsum* also niemals diesen Bereich.

ACHTUNG

Immer <u>bevor</u> der 64. Wert bewertet wird (*valsum* ist hier ungerade und entweder *valsum* – 1 oder *valsum* + 1 ist durch 4 teilbar!), wird *d* vor der Bewertung u. U. noch verändert.

Dazu wird zunächst die Region für *d* bestimmt. Der Grenzwert aus der oberen Tabelle zwischen Region 1 und 2 ist dabei allerdings um 1 kleiner als sonst.

> Aus Gründen der „Symmetrie" sollte eigentlich auch der Grenzwert zwischen Region 3 und 4 um 1 geändert sein. Das konnte aber nicht bestätigt werden.

d wird nun folgendermaßen verändert:

Wurde die Region 0, 2 oder 4 ermittelt, wird *d* um 1 vergrößert wenn

- gilt: *valsum* – 1 ist durch 4 teilbar und *d* ist ungerade oder
- gilt: *valsum* + 1 ist durch 4 teilbar und *d* ist gerade

Wurde die Region 1 ermittelt, wird *d* immer um 1 vergrößert und zusätzlich noch ein weiteres mal vergrößert wenn

- gilt: *valsum* – 1 ist durch 4 teilbar und *d* ist ungerade oder
- gilt: *valsum* + 1 ist durch 4 teilbar und *d* ist gerade

Wurde die Region 3 ermittelt, wird *d* um 1 verringert wenn

- gilt: *valsum* – 1 ist durch 4 teilbar und *d* ist gerade oder
- gilt: *valsum* + 1 ist durch 4 teilbar und *d* ist ungerade

Plateau-Nachfolger

Für Plateau-Nachfolger mit $ddiff \neq 0$ wird bei positiven Werten 1, bei negativen Werten -1 summiert. Ist die so erzeugte Summe größer als 0, wird L0 verwendet, sonst L2.

Für Plateau-Nachfolger mit $ddiff = 0$ wird die Summe genauso gebildet. Ist die so erzeugte Summe größer als 0, wird L1 verwendet, sonst L0.

Die Summe ist natürlich immer ungerade, kann also auch nicht 0 werden.

Halbierung der Summen

Bei den Summierungen gibt es eine Besonderheit:

Immer wenn 64 Werte registriert sind, wird die Anzahl der registrierten Werte auf 32 heruntergesetzt, also halbiert.

Die Summe *sumh* für die Hybridcodierung wird verringert auf $sumh_{32} = ((sumh_{64} - hunitdelta) \gg 1) - 1$. Für Plateau-Nachfolger mit $ddiff = 0$ wird diese Summe zusätzlich noch um 1 verringert. Übrigens wird hier nicht einfach durch 2 geteilt, sondern eine Bitschiebe-Operation verwendet. Eine integer-Division würde für positive Zahlen zwar zum gleichen Ergebnis führen, nicht jedoch bei negativen Zahlen!

Die Summe *suml* für die Längencodierung wird halbiert $suml_{32} = integer(suml_{64}/2)$. Ist die neue Summe ungerade, wird sie um 1 vergrößert.

Die Bewertung für die Längencodierung ergibt immer einen ungeraden Wert. Deshalb ist die Summe bei geradem Index immer gerade, bei ungeradem Index immer ungerade. Bei der Halbierung bleibt dieser Zustand erhalten.

Für Plateau-Nachfolger mit $ddiff \neq 0$ wird eine gerader Summe nicht durch Vergrößerung sondern durch Verkleinerung um 1 erzeugt.

Datenberechnung

Eine Kachel (64 x 64) enthält einfach 4096 Höhenwerte. Deshalb hat man zunächst je Kachel eine Zahlen-Matrix von 64 x 64 Zahlen, die die Höhen punktuell angeben.

Diese Höhen werden aber nicht direkt gespeichert. Wenn man möglichst kleine Zahlen mit genau angepassten Codierverfahren speichert, kann man sehr viel Speicherplatz sparen. „Kleine Zahlen“ bedeutet eigentlich „wenige Bits zur Speicherung der Zahlen“, aber eine Voraussetzung ist trotzdem, dass die Zahlen selbst möglich klein sind.

Deshalb werden die realen Höhen zunächst auf eine Basis-Höhe bezogen (gespeichert im Kacheldatensatz). Diese Basis-Höhe ist einfach die kleinste vorhandene Höhe. In Beispiel sei das die Höhe 90. Dann ergibt sich eine neue, normierte Zahlen-Matrix.

Aber auch diese Werte werden nicht gespeichert. Es werden aus diesen Werten normalerweise noch gewisse Differenzen gebildet. In einigen Fällen werden nach bestimmten Regeln auch Bereiche konstanter Höhe verkürzt gespeichert mit der Länge des Bereiches und der nachfolgenden Höhe gespeichert. In der endgültigen Zahlen-Matrix erkennt man z. B., dass in der 1. Zeile einfach nur die Differenzen zum Vorgänger enthalten sind.

95	110	115	110	107	…
110	112	116	119	111	…
115	117	115	113	119	…
120	122	118	116	113	…
125	124	121	119	111	…
…	…	…	…	…	…

Originalhöhen

5	20	25	20	17	…
20	22	26	29	21	…
25	27	25	23	29	…
30	32	28	26	23	…
35	34	31	29	21	…
…	…	…	…	…	…

normierte Höhen

5	15	5	-5	-3	…
15	2	1	8	-5	…
5	0	-6	5	14	…
5	0	-2	0	9	…
5	-3	1	0	-5	…
…	…	…	…	…	…

zu speichernde Datenwerte

Die Höhenwerte werden im Weiteren durch die Angabe der Spalten- und Zeilennummer, beide 0-basiert, angegeben. *h*(0, 0) ist also die Höhe in der linken oberen Ecke, *h*(1, 0) die Höhe rechts daneben und *h*(0, 1) die Höhe darunter in der nächsten Zeile. Analog werden die dazu gehörenden Datenwerte mit *d*(col, line) bezeichnet.

Außerdem fügen wir noch eine virtuelle Zeile über der Zeile 0 und eine virtuelle Spalte vor der Spalte 0 ein. Die virtuelle Zeile soll nur die Höhen 0 enthalten, also *h*(*i*, -1) = 0. Die virtuelle Spalte soll die gleichen Werte wie die erste Spalte, aber um 1 Zeile nach unten verschoben, enthalten, also *h*(-1, *i*) = *h*(0, *i*-1).

An einigen Stellen werden Differenzen zweier benachbarter Werte verwendet. Es gilt dann:

für die „horizontale“ Differenz $hdiff(n,m) = \boldsymbol{h(n,m)} - h(n-1,m)$,
für die „vertikale“ Differenz $vdiff(n,m) = \boldsymbol{h(n,m)} - h(n,m-1)$
und für die „diagonale“ Differenz $ddiff(n,m) = h(n,m-1) - h(n-1,m)$.

h(*n*-1, *m*-1)	*h*(*n*, *m*-1)
h(*n*-1, *m*)	***h*(*n*, *m*)**

Daraus folgt: *hdiff(n, m)* = *ddiff(n, m)* + *vdiff(n, m)*.

	n	0	1	2	3	4	
m		0	0	0	0	0	…
0	0	5	15	5	-5	-3	…
1	5	15	2	1	8	-5	…
2	15	5	0	-6	5	14	…
3	5	5	0	-2	0	9	…
4	5	5	-3	1	0	-5	…
	…	…	…	…	…	…	…

Matrix mit virtueller Spalte und Zeile

Stand 13.7.2018

Standardwerte

Für Standardwerte muss gelten: *ddiff*(*n*, *m*) ≠ 0, also Vorgängerhöhe ≠ Höhe „darüber“. Andernfalls beginnt ein „Plateau“ (s.u.).

> Daraus folgt übrigens, dass die 1. Spalte niemals einen Standardwert enthalten kann und 0-Höhen in der 1. Zeile auch keine Standardwerte sind.

Dann findet noch eine Fallunterscheidung für *hdiff*(*n*, *m*-1) statt:

$$\begin{array}{lll} hdiff(n,m-1) \geq max - h(n-1,m): & d(n,m) = & -sgn(ddiff(n,m)) \cdot (h(n,m)+1) \\ hdiff(n,m-1) \leq -h(n-1,m): & d(n,m) = & -sgn(ddiff(n,m)) \cdot h(n,m) \\ \text{sonst:} & d(n,m) = & -sgn(ddiff(n,m)) \cdot (hdiff(n,m) - hdiff(n,m-1)) \end{array}$$

$hdiff(n,m-1) = max - h(n-1,m) + x$ und $hdiff(n,m-1) \geq 0\,;\, x \geq 0$

$$\begin{aligned}
d_3(n,m) &= -sgn(ddiff)\cdot(hdiff(n,m) - hdiff(n,m-1)) \\
d_3(n,m) &= -sgn(ddiff)\cdot(hdiff(n,m) - hdiff(n,m-1)) \\
d_3(n,m) &= -sgn(ddiff)\cdot(h(n,m) - h(n-1,m) - (max - h(n-1,m) + x)) \\
d_3(n,m) &= -sgn(ddiff)\cdot(h(n,m) - h(n-1,m) - max + h(n-1,m) - x) \\
d_3(n,m) &= -sgn(ddiff)\cdot(h(n,m) - max - x) \quad \text{aber } d_1(n,m) = -sgn(ddiff)\cdot(h(n,m)+1) \\
d_3(n,m) &= -sgn(ddiff)\cdot(-sgn(ddiff)\cdot(d_1(n,m)) + sgn(ddiff) - max - x) \\
d_3(n,m) &= -sgn(ddiff)\cdot - sgn(ddiff)\cdot(d_1(n,m)) - sgn(ddiff)\cdot sgn(ddiff) + sgn(ddiff)\cdot(max + x) \\
d_3(n,m) &= d_1(n,m) - 1 + sgn(ddiff)\cdot(max + x) \\
d_3(n,m) &\leq d_1(n,m) - 1 + max \quad \text{bei } ddiff > 0 \\
d_3(n,m) &\geq d_1(n,m) - 1 - max \quad \text{bei } ddiff > 0
\end{aligned}$$

$$\begin{aligned}
d_3(n,m) &= -sgn(ddiff)\cdot(hdiff(n,m) - hdiff(n,m-1)) \\
d_3(n,m) &= -sgn(ddiff)\cdot(h(n,m) - h(n-1,m) - (-h(n-1,m) - x)) \\
d_3(n,m) &= -sgn(ddiff)\cdot(h(n,m) - h(n-1,m) + h(n-1,m) + x) \\
d_3(n,m) &= -sgn(ddiff)\cdot(h(n,m) + x) \quad \text{aber } d_1(n,m) = -sgn(ddiff)\cdot h(n,m) \quad \text{also} \\
d_3(n,m) &= -sgn(ddiff)\cdot(-sgn(ddiff)\cdot d_1(n,m) + x) \\
d_3(n,m) &= -sgn(ddiff)\cdot - sgn(ddiff)\cdot d_1(n,m) - sgn(ddiff)\cdot x \\
d_3(n,m) &= d_1(n,m) - sgn(ddiff)\cdot x \\
d_3(n,m) &\geq d_1(n,m) \quad \text{bei } ddiff > 0 \\
d_3(n,m) &\leq d_1(n,m) \quad \text{bei } ddiff < 0
\end{aligned}$$

Hinweis:
Für die 1. Zeile gilt wegen der virtuellen Zeile vereinfacht:

$$\begin{aligned} d(n,0) &= -sgn(ddiff(n,0)) \cdot (hdiff(n,0) - 0) \\ d(n,0) &= hdiff(n,0) \end{aligned}$$

Beispiel:
…, 5, 7, 3, 2, …
…, 6, **2**, **5**, **4**, …
h(*n*-1,*m*) < -*hdiff*(*n*, *m*-1)

6 < –(7 – 5)	→ nein	–sgn(7 – 6) · ((2 – 6) – (7 – 5)) = –(0 – 2) = 2
2 < –(3 – 7)	→ ja	–sgn(3 – 2) · 4 = –4
5 < –(2 – 3)	→ nein	–sgn(2 – 5) · ((4 – 5) – (2 – 3)) = (–1 – –1) = 0

Mit der Einführung eines Hilfswertes $\boldsymbol{hook}(n,m) = h(n-1,m) + h(n,m-1) - h(n-1,m-1)$ ändern sich die Gleichungen folgendermaßen:

$$\begin{array}{lll} max_{enc} \leq hook: & d(n,m) = & -sgn(ddiff(n,m)) \cdot (h(n,m)+1) \\ 0 < hook < max_{enc}: & d(n,m) = & -sgn(ddiff(n,m)) \cdot (h(n,m) - hook) \\ hook \leq 0: & d(n,m) = & -sgn(ddiff(n,m)) \cdot h(n,m) \end{array}$$

Für Standardwerte wird die Hybridcodierung HStd bzw. die Längencodierung L0 bzw. L1 verwendet.

Stand 13.7.2018

Plateaus

Ein Plateau besteht aus nebeneinanderliegenden konstanten Höhenwerten. Es liegt auf der Hand, dass in einem solchen Fall nicht jede Höhe einzeln angegeben werden muss. Es genügt, wenn die Länge eines Plateaus angegeben wird. Leider kann nicht willkürlich ein Plateau „gestartet“ werden, sondern für d_p muss die spezielle Bedingung *ddiff* = 0 gelten. Daraus folgt auch, dass immer entweder ein Standardwert oder ein Plateau vorliegt.

Wenn *hdiff* = 0 und *vdiff* = 0, also h_p mit seinem Vorgänger und der Höhe „darüber“ identisch ist, ist die aktuelle Position der Startpunkt eines echten Plateaus mit der aktuellen Höhe (des Vorgängers) und mindestens der Länge 1. Andernfalls handelt es sich um ein unechtes Plateau mit der Länge 0. Dieser Fall kommt vermutlich selten genug vor, so dass sich dieses Verfahren zum Einsparen von Speicherplatz trotzdem lohnt. Auf jeden Fall wird statt eines Datenwertes d_i (für h_i) nur die Länge des Plateaus gespeichert, dass mit h_i beginnt.

Im folgenden Beispiel beginnt nach der 4 ein Plateau. Dessen Länge ist 4 (nicht 5!). der Nachfolgewert ist 5.

…	…	…	…	…	…	…	…	…	…
…	…	…	…	4	…	…	…	…	…
…	2	3	4	4	4	4	4	5	…
…	…	…	…	…	…	…	…	…	…

Bestimmung des Nachfolgewertes

Eine Besonderheit der Plateaucodierung ist die damit verbundene spezielle Bestimmung des direkt nach dem Plateau folgenden Wertes *h*(*f*, *m*). Dafür spielt die Differenz *ddiff* des Nachfolgewertes eine wichtige Rolle.

$$
\begin{aligned}
ddiff(n,m) \neq 0: &\quad d(n,m) = -sgn(ddiff(n,m)) \cdot wrap(vdiff(n,m)) \\
ddiff(n,m) = 0: &\quad d(n,m) = wrap(vdiff(n,m)) - \frac{sgn(wrap(vdiff(n,m))) - 1}{2}
\end{aligned}
$$

Natürlich muss bei *ddiff* = 0 auch v*diff*(*n*, *m*) ≠ 0 gelten, da sonst die Plateaulänge einfach nur zu kurz wäre.

Bei Plateaus mit *ddiff*(*n*, *m*) = 0 wird die Hybridcodierung HPl0 bzw. die Längencodierung L0 oder L1 verwendet.

Bei Plateaus mit *ddiff*(*n*, *m*) ≠ 0 wird die Hybridcodierung HStd bzw. die Längencodierung L0 oder L2 verwendet.

Codierung der Plateaulänge

Wie schon in der Überschrift gesagt, wird diese Codierung ausschließlich für die Längenangabe eines Plateaus verwendet.

Die Längenangabe besteht aus einer Folge von 1-Bits, die durch ein 0-Bit abgeschlossen und eventuell von Binärbits gefolgt wird: $1_s \ldots 1_0 0 b_k \ldots b_0$. Während die Binärbits wie üblich den Wert $\sum_{i=0}^{k} b_i \cdot 2^i$ ergeben, haben die 1-Bits, abhängig von ihrer Position, unterschiedliche Werte. Diese Werte und die Anzahl der Binärbits ergeben sich aus folgender Tabelle:

Startpos.	***0***	***1***	***2***	***3***	***4***	***5***	***6***	***7***	***8***	***9***	***10***	***11***	***12***	***13***	***14***	***15***	***16***	***17***	***18***	***19***	***20***	***21***	***22***
Bitwert	1	1	1	1	2	2	2	2	4	4	4	4	8	8	8	8	16	16	32	32	64	64	128
Binärbits	0	0	0	1	1	1	1	2	2	2	2	3	3	3	3	4	4	5	5	6	6	7	8

Die Plateaulänge ergibt sich aus der der Summe der 1-Bit-Werte und dem Binärwert. Es kann damit jeder beliebige Wert codiert werden.

Beispiele:
Tabellenposition = 0 (1. Plateaulänge)

```
.                  0 + 0 = 0
11.                2 * 1 = 2
1111..             4 * 1 + 0 = 4
11111.1            4 * 1 + 1 * 2 + 1 = 7
1111111111111.1.1  4 * 1 + 4 * 2 + 4 * 4 + 1 * 8 + 5 = 41
1111111..          4 * 1 + 3 * 2 + 0 = 10
```

Die Codierung erfolgt immer soweit wie möglich mit den 1-Bits. Erst wenn die exakte Länge damit nicht codiert werden kann, werden die Binärbits entsprechend gesetzt.

> Eine Plateaulänge 1 die mit der Codierung an Tabellenposition 3 startet, wird nicht als „`.1`“ codiert, sondern als „`1..`“.

Die Anzahl der Binärbits vergrößert sich immer um 1, wenn das nächste 1-Bit einen höheren Wert als das bisherige hat.

> Da die Verwendung der 1-Bits Vorrang vor den Binärbits hat, gibt es ein Problem z. B. beim Codieren der Länge 5. 4x 1-Bit ergibt 4, 5x 1-Bit jedoch schon 6. Deshalb muss beim 4. 1-Bit schon 1 Binärbit vorhanden sein. Analog gilt das für die Positionen 7, 11, 15 usw..

Bevor die nächste Plateaulänge codiert wird, wird die Startposition in der Tabelle zunächst dekrementiert (aber natürlich nicht kleiner als 0). Das 1. Längenbit des neuen Plateaus hat also i.A. die gleiche Startposition in der Tabelle, wie das letzte Längenbit des vorherigen Plateaus. Für jedes verwendete Längenbit im neuen Plateau wird die Startposition inkrementiert. Der Wert jedes Längenbits ergibt sich sich aus seiner Startposition entsprechend der Tabelle. Die Anzahl der Binärbits ergibt sich aus der letztendlich erreichten Startposition.

Hat also das letzte 1-Bit eines Plateaus die Position n in der Tabelle, geht man für das folgende Plateau von der Position $n - 1$ aus. Hat dieses folgende Plateau k 1-Bits, ergibt sich die Anzahl der Binärbits aus der Position $n - 1 + k$. Daraus ergibt sich, dass bei jeder Plateaulänge 0 die Position effektiv um 1 geringer wird.

> Beispiele:
> Tabellenposition = 0 (1. Plateaulänge)
> `1111111..` $4 \cdot 1 + 3 \cdot 2 + 0 = 10$
> dann 2. Plateaulänge mit Tabellenposition = 6
> `1111..1` $2 \cdot 2 + 2 \cdot 4 + 1 = 13$

Tendenziell können bei größer werdender Startposition größere Längen mit weniger Bits codiert werden, da die Binärcodierung für größere Zahlen besser geeignet ist.

Wird mit den Längenbits (!) das Zeilenende erreicht oder überschritten, entfallen das nachfolgende 0-Bit und die eventuell nötigen Binärbits. Es existiert natürlich auch kein Plateau-Nachfolger. Nach den Längenbits folgt sofort die Längenangabe für das Start-Plateau der nächsten Zeile. Eine kleine Besonderheit gibt es noch, wenn mit den Längenbits das Zeilenende genau erreicht wird: die Startposition wird nicht dekrementiert! Dadurch kann mit weniger Bits ein größerer Zeilenbereich umfasst werden.

ACHTUNG

Prinzipiell kann die Tabelle vermutlich auch länger sein. Durch den beschriebenen Algorithmus kann der höchste erreichbare Bitwert nur der nächstgrößere Wert als die Kachelbreite sein. Bei der Standardkachelbreite 64 ist das also 128. Bei einer Kachelbreite von 63 ist das 64. Es wurden schon Kacheln am rechten Rand einer Karte mit einer Breite größer als 64 gefunden, z. B. 73. Daraus hätte man auch eine Standardkachel mit der Breite 64 und eine weitere Spalte mit der Breite 9 machen können. Möglicherweise ist die von GARMIN verwendete Variante aber effektiver.

Größere Breiten als 128 erscheinen aber nicht sinnvoll. Deshalb wurde die Tabelle nicht mit größeren Bitwerten angegeben.

> Beispiel einer Originalkachel mit Breite 64:
> In der DEM-Datei wurde die folgende Bytefolge für eine Kachel mit der Maximalhöhe 3 gefunden:
> FF FF FF FF FF FF FF FF FF FF C0 2E
> Der Bitstream besteht also aus 82 1-Bits, danach folgt „`........1.111.`“
> Damit werden alle Punkte auf 0 bzw. die Basishöhe der Kachel gesetzt. Nur der Punkt in der Ecke links unten ist 3.
> Erklärung:
> Die ersten 17 1-Bits stellen die Plateaulänge 4·1+4·2+4·4+4·8+16 = 76 dar, d.h. die Länge der 1. Zeile wird überschritten. Die nächsten 3 1-Bits stellen die Plateaulänge 16+16+32 = 64 dar. Damit wird die 3. Zeile erreicht. Die nächsten 2 1-Bits stellen die Plateaulänge 32+32 = 64 dar. Damit wird die 4. Zeile erreicht. Die nächsten 60 1-Bits stellen einzeln jeweils 64 dar, d.h. es folgen weitere 20 Zeilenwechsel. Damit befindet man sich nach den 82 1-Bits am Anfang der 64. Zeile. Es folgen 1 Trennbit und 7 Binärbits (Länge 0).
> Es bleiben jetzt noch folgende Bits: „`1.111.`“. Die ersten 2 Bits „`1.`“ stehen bei *hunit*=1 für 0, als Nachfolgewert eines Plateaus bedeutet das die Höhe -1. Wegen des Wraparound steht -1 für die max. Höhe, also 3. Man hätte natürlich auch die 3 direkt codieren können. Bei *hunit*=1 ergibt das „`..11`“, also 4 Bit, d.h. 2 Bit mehr.
> Ähnlich sieht es für die folgenden 2 Bit aus. „`11`“ steht bei *hunit*=1 für 1 und führt zur Höhe 4, die aber wegen des Wraparound für 0 steht. -3 „`...1.`“ führt zum gleichen Ergebnis, benötigt aber 5 statt 2 Bit.
> Die restlichen 2 Bits „`1.`“ stellen eine Plateaulänge für den Rest der Zeile mit der aktuellen Höhe 0 dar. Die Binärbits und der Nachfolger können offensichtlich entfallen.

Verwendung von Shrink-Werten größer als 0 (nur ein Zwischenstand)

Eine Höhenänderung um den Wert 1 steht jetzt nicht mehr für eine Änderung um 1m/1Fuss sondern für einen größeren Wert. Aus dem gespeicherten Wert wird zunächst der eigentliche *Shrink*-Faktor mit $shrink = 2 \cdot value + 1$ berechnet.

Versuche mit verschiedenen Tabellen-Einträgen max_{diff} für die maximale Differenz und verschieden Werten für *shrink* zeigen, dass sich aus diesen beiden Werten ein max_{enc} für den Encoder nach folgender Regel ergibt:

$$max_{enc} = \operatorname{int}\left(\frac{max_{diff} - 1}{shrink}\right) + 1$$

So gilt z.B. $\operatorname{int}\left(\frac{84-1}{13}\right) + 1 = \operatorname{int}(6{,}38) + 1 = 7$ bzw. für max_{diff} im Bereich 79 bis 91 gilt bei *shrink* = 13 der Maximalwert $max_{enc} = 7$.

Das läßt sich leicht über das Wrapping überprüfen, wenn z.B. die Datenwerte 1 und 9 zum gleichen Ergebnis führen. Die Differenz 8 bedeutet, dass die Werte 0 bis 7 verwendet werden können.

Interessant ist, wie Mapsource diese Datenwerte anzeigt. Normalerweise gilt $h_{visible} = min(max_{diff}, h_{encoder} \cdot shrink)$.
Bei $h_{encoder} = max_{enc}$ wird dann immer max_{diff} angezeigt. Die Berechnung ist gewissermaßen auf 0 bezogen (0-aligned). Sie kann aber aber auch auf max_{diff} bezogen sein (top-aligned). Dann gilt
$h_{visible} = max(0, max_{diff} - (h_{encoder} - max_{encoder}) \cdot shrink)$. Im vorherigen Beispiel führt das zu folgender Höhenanzeige:

$h_{encoder}$	0-aligned	top-aligned												
		79	80	81	82	83	84	85	86	87	88	89	90	91
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	13	1	2	3	4	5	6	7	8	9	10	11	12	13
2	26	14	15	16	17	18	19	20	21	22	23	24	25	26
3	39	27	28	29	30	31	32	33	34	35	36	37	38	39
4	52	40	41	42	43	44	45	46	47	48	49	50	51	52
5	65	53	54	55	56	57	58	59	60	61	62	63	64	65
6	78	66	67	68	69	70	71	72	73	74	75	76	77	78
7	max_{diff}	(79)	(80)	(81)	(82)	(83)	(84)	85	86	87	88	89	90	91

Daraus folgt, dass sich die Abbildung der Originalwerte auf die angezeigten Werte an den jeweils verwendeten Modus anpassen sollte. Unabhängig vom Modus sind nur die Anzeige für 0 und den Maximalwert.

Der top-aligned-Modus kann nicht willkürlich ein- bzw. ausgeschaltet werden. Er ergibt sich aus dem Status der „umgebenden" Werte.

Da im folgenden öfter die 3 Werte links, oberhalb und links oberhalb vom aktuellen Wert eine Rolle spielen, also genau die Werte aus denen der Hook berechnet wird, wird für deren Status die Schreibweise TA gefolgt vom Status links, oberhalb und links oberhalb verwendet. Es gibt 8 Varianten. TA001 steht z.B. dafür, dass links und oberhalb der 0-aligned-Modus verwendet wird, links oberhalb aber der top-align-Modus.

Analog wird bei nur 2 relevanten Nachbarn links und oberhalb die Schreibweise TA gefolgt vom Status links und oberhalb verwendet. Es gibt 4 Varianten.

Außer *ddiff* wird jetzt i.A. *ddiffT* unter Berücksichtigung des top-aligned-Modus verwendet. Es gilt i.A. *ddiffT* = *ddiff*. Nur bei TA10 gilt *ddiffT* = *ddiff* + 1, bei TA01 gilt *ddiffT* = *ddiff* - 1.

Auch für *hook* wird die Berechnung etwas modifiziert. Bei TA001 wird er um 1 erhöht, bei TA110 um 1 verringert.

Status

0 immer 0-aligned

max_{enc} immer top-aligned

Status für Standardwerte:

TA000, TA101, TA011, TA110 top-aligned wenn $max_{enc} \leq hook$

TA111, TA010, TA100, TA001 top-aligned wenn $0 < hook$

Status für Plateauwerte:

Alle Werte des Plateaus haben den gleichen Status wie der 1. Wert des Plateaus.

Status für Plateau-Nachfolger:

Bei Plateaus mit $ddiffT(n, m) = 0$ wird der gleiche Status wie für das Plateau verwendet. Bei Plateaus mit $ddiffT(n, m) \neq 0$ wird der Status des Wertes oberhalb verwendet.

Auch das Auftreten von Plateaus ist etwas modifiziert:

I.A. muss gelten *ddiffT* = 0. Bei TA100 und TA011 erweitert sich die Regel jedoch etwas.

Stand 13.7.2018

Bei TA100 muss für ein Plateau *ddiff* = 0 gelten falls für den Wert links (!) $1 \le hook$ und TA001 gilt.

Bei TA011 muss für ein Plateau *ddiff* = 0 gelten falls für den Wert links (!) $1 \le hook$ und TA110 gilt.

Diese Regeln sind noch nicht vollständig erkannt!

Modifikation der Berechnung:

Bei der Berechnung von Standardwerten wird statt sgn(*ddiffT*) bei *ddiffT* = 0 sgn(*ddiff*) verwendet.

Bei manchen Datenwerten für die Erreichung der Höhe 0 bzw. max_{diff} wird genau der entgegengesetzte Wert angezeigt. Der Sinn ist unklar. Diese Sonderfälle treten unter folgenden Umständen auf:

$hook \le 0$		$d(n,m) = -sgn(ddiffT(n,m)) \cdot max_{enc}$
$0 < hook < max_{enc}$	bei TA101, TA100, TA010, TA111	$d(n,m) = sgn(ddiffT(n,m)) \cdot hook$
	bei TA000, TA011	$d(n,m) = sgn(ddiffT(n,m)) \cdot (hook - max_{enc})$
$max_{enc} \le hook$		$d(n,m) = sgn(ddiffT(n,m)) \cdot max_{enc}$

Der Wert d ist dann entweder um den Wert $max_{diff} + 1$ zu erhöhen oder zu verringern, so dass sein Vorzeichen wechselt (wrapping).

Für den Nachfolger des Plateaus gibt es Sonderfälle wie bei den Standardwerten.

Für $ddiffT \neq 0$:

TA11, TA01 und Höhe 0:	$ddiff > 0$ und $d = h(n, m-1)$
	$ddiff < 0$ und $d = -h(n, m-1)$
TA10, TA00 und Höhe max_{enc}:	$ddiff > 0$ und $d = h(n, m-1) - max_{enc}$
	$ddiff < 0$ und $d = -(h(n, m-1) - max_{enc})$

Für $ddiffT = 0$:

Zunächst erfolgt nach der Berechnung d = *vdiff* eine weitere Modifikation:

TA11, TA00:	wenn $d < 0$ dann $d = d + 1$
TA10:	wenn $d > 0$ dann $d = d - 1$
TA01:	wenn $d \ge 0$ dann $d = d + 1$, sonst $d = d + 2$

Die Sonderfälle treten auf bei:

TA11 und Höhe 0:	$d(n,m) = -d(n,m-1)+1$
TA10 und Höhe 0:	$d(n,m) = -d(n,m-1)$
TA00 und Höhe max_{enc}:	$d(n,m) = max_{enc} - d(n,m-1)$
TA01 und Höhe max_{enc}:	$d(n,m) = max_{enc} - d(n,m-1)+1$

Stand 13.7.2018

Für den Encoder sind noch weitere „Feinjustierungen“ gegenüber dem normalen Modus nötig:

a) Die maximale Länge einer 0-Bitfolge richtet sich nach max_{diff} (**nicht** $max_{encoder}$) und nach *Shrink*. Ist sie 1 Bit länger, wird ein BigBin-Wert erwartet.

Die Startwerte eines Bereiches werden zum einen als Zahlenfolge $z_{neu}=2\cdot z_{alt}+shrink-1$ gebildet. „Eingemischt“ werden die Werte der zweiten Zahlenfolge $z_{neu}=2\cdot z_{alt}$. Der Startwert beider Zahlenfolgen ist 1.

Die Glieder dieser Zahlenfolgen können direkt berechnet werden mit $z_n=shrink\cdot(2^n-1)+1$ und $z_n=2^n$. Für die 2. Folge kann die maximale Bitanzahl so berechnet werden:

$$bits_{max}=3\cdot(\mathrm{int}(\mathrm{ld}(max_{diff}))+1)+\mathrm{int}(\mathrm{ld}((shrink-1)/2))$$

Bei einem Wert der 1. Folge verringert sich die Bitanzahl gegenüber der vorherigen um 1.

In der Tabelle sind die Startwerte für Shrink 1, 3, 5 und 7 eingetragen. Die „kleinen“ Bereichgrenzen spielen bei Shrink >= 3 praktisch keine Rolle, weil die maximale 0-Bit-Länge deutlich größer ist.

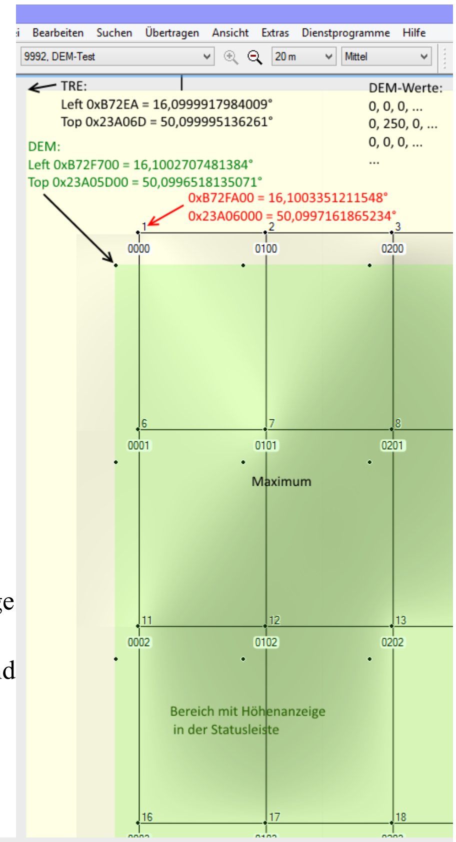
b) Im Standardmodus gibt es keine L1-Codierung mehr.

Je *Shrink*-Stufe werden etwa 10% bis 15% Speicherplatz eingespart.

1		3		5		7	
ab	Bits	ab	Bits	ab	Bits	ab	Bits
0 .. 1	15	1	x	1	x	1	x
2	16	2	x	2	x	2	x
4	17	4	x	4	x	4	x
			x	6	x		x
8	18	8	x	8	x	8	x
		10	x		x		x
16	19	16	x	16	x	20	x
		22	x		x	22	x
32	20	32	x	32	x	32	x
		46	17	36	x	50	x
64	21	64	21	64	x	64	x
		94	20	76	21	106	x
128	22	128	24	128	25	128	25
		190	23	156	24	218	24
256	25	256	27	256	28	256	28
		382	26	316	27	442	27
512	28	512	30	512	31	512	31
		766	29	636	30	890	30
1024	31	1024	33	1024	34	1024	34
		1534	32	1276	33	1786	33
2048	34	2048	36	2048	37	2048	37
		3070	35	2556	36	3578	36
4096	37	4096	39	4096	40	4096	40
		6142	38	5116	39	7162	39
8192	40	8192	42	8192	43	8192	43
		12286	41	10236	42	14330	42
16384	43	16384	45	16384	46	16384	46

Geografische Ausrichtung der DEM-Daten

Testkarte: Das „Raster" in der Karte wurde mit entsprechenden Punkten für Großstädte („1" usw.) und mit „Küstenlinien" direkt in der OSM-Datei definiert. Die Punkte „0000", „0100" usw. wurden in einer GPX-Datei definiert. Als DEM-Daten wurde nur die Höhe 0 verwendet. Nur der Punkt in der 2. Spalte der 2. Zeile wurde auf 250 gesetzt („Bergspitze"). Der Punktabstand für die DEM-Daten ist 0x1200.



Zunächst ist zu beachten, dass die Karten-Ecke links oben in der TRE-Datei nur mit 24 Bit angegeben wird. Um diese Werte mit der Ecke links-oben der DEM-Daten zu vergleichen, ist an die TRE-Werte jeweils ein 0-Byte (8 Bit) anzuhängen.

Merkwürdigerweise existieren für die Anzeige der DEM-Werte in der Statusleiste und für die grafische Anzeige 2 verschiedene Bezugspunkte. Der im Bild grün dargestellte Bereich, in dem eine Anzeige in der Statusleiste erfolgt, bezieht sich exakt auf den Punkt links-oben der in der DEM-Datei angegeben ist (Punkt „0000"). Die Maximalhöhe, d.h. die „Bergspitze", wird im Punkt „0101" angezeigt.

Wenn man von der grafischen Anzeige links von der Linie durch die Punkte „1" und „16" absieht, ist der Bezugspunkt links-oben für die grafische Anzeige der Punkt „1". Die Bergspitze ist im Punkt „7".

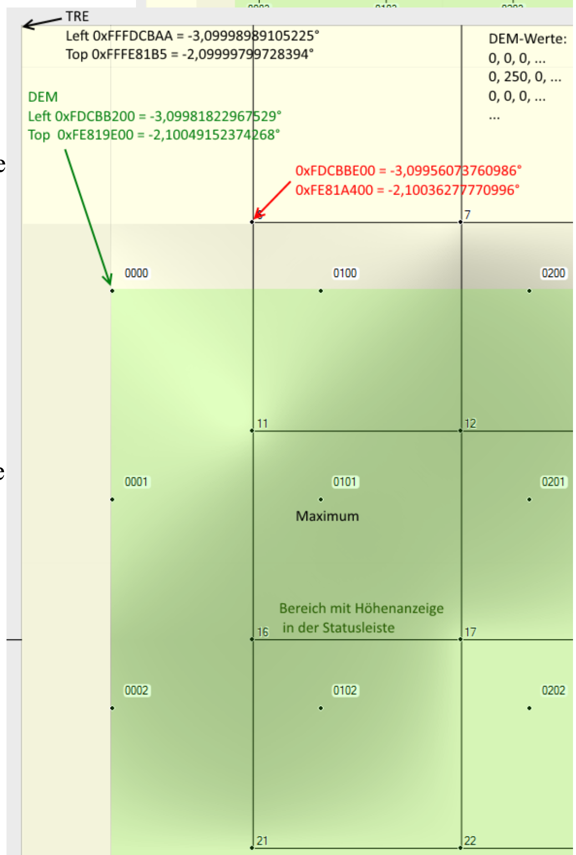
Es sollte natürlich vermieden werden, dass es zwischen grafischer Anzeige und Anzeige in der Statusanzeige eine Differenz gibt. Der Bezugspunkt für die Anzeige in der Statusanzeige läßt sich offensichtlich beliebig als Position in der DEM-Datei definieren. Versuche zeigen, dass das mit der grafischen Anzeige **nicht** möglich ist.

Für den Bezugspunkt gilt immer West % Distance = 0 und North % Distance = 0. Die Position 0xB72FA00 im Beispiel wird eingenommen, wenn 0xB72F100 <= West < 0xB730300 gilt. 0x23A06000 wird eingenommen, wenn 0x23A05700 < North <= 0x23A06900.

Um die beiden Bezugspunkte deckungsgleich zu halten, sollten die DEM-Datei so positioniert werden, dass West % Distance = 0 und North % Distance = 0 gilt. Gleichzeitig sollte natürlich der gesamte Kartenbereich abgedeckt werden. Die DEM-Position sollte also identisch zur TRE-Position sein, oder links-oberhalb davon liegen. Das wäre im Beispiel die Position lat=0xB72E800, lon=0x23A07200.

Ein ähnliches Beispiel wurde mit analogem Ergebnis auch für einen Punktabstand 0xCF0 = 3312 untersucht.

Das untere Bild zeigt ein entsprechendes Beispiel für eine negative Länge und Breite.



Es sollte offensichtlich gelten:

$$lat_{DEM} = floor\left(\frac{lat_{TRE}}{lat_{Distance}}\right)\cdot lat_{Distance}$$

$$lon_{DEM} = ceiling\left(\frac{lon_{TRE}}{lon_{Distance}}\right)\cdot lon_{Distance}$$

Das bedeutet für die beiden Beispiele:

$\begin{aligned} lat_{DEM} &= floor\left(\frac{0xB72EA \ll 8}{0x1200}\right)\cdot 0x1200 \\ &= floor\left(\frac{192080384}{4608}\right)\cdot 4608 \\ &= floor(41684{,}1)\cdot 4608 \\ &= 192079872 \\ &= 0xB72E800 \\ &\rightarrow 16{,}099948883056640625\,° \\ lon_{DEM} &= ceiling\left(\frac{0x23A06D \ll 8}{0x1200}\right)\cdot 0x1200 \\ &= ceiling\left(\frac{597716224}{4608}\right)\cdot 4608 \\ &= ceiling(129712{,}7)\cdot 4608 \\ &= 597717504 \\ &= 0x23A07200 \\ &\rightarrow 50{,}10010242462158203125\,° \end{aligned}$	$\begin{aligned} lat_{DEM} &= floor\left(\frac{0xFFFE81B5 \ll 8}{0x1200}\right)\cdot 0x1200 \\ &= floor\left(\frac{-36984320}{4608}\right)\cdot 4608 \\ &= floor(-8026{,}1)\cdot 4608 \\ &= -36988416 \\ &= 0xFDCB9A00 \\ &\rightarrow -3{,}1003332138061523437 5\,° \\ lon_{DEM} &= ceiling\left(\frac{0xFFFDCBAA \ll 8}{0x1200}\right)\cdot 0x1200 \\ &= ceiling\left(\frac{-25053952}{4608}\right)\cdot 4608 \\ &= ceiling(-5437{,}1)\cdot 4608 \\ &= -25053696 \\ &= 0xFE81B600 \\ &\rightarrow -2{,}0999765396118164062 5\,° \end{aligned}$
Die grafische Anzeige ist mit 0xB72FA00 im Beispiel also 1*0x1200, d. h. 1 Rasterbreite zu weit rechts und mit 0x23A06000 1*0x1200, d. h. 1 Rasterhöhe zu tief.	Die grafische Anzeige ist mit 0xFDCBBE00 im Beispiel also 2*0x1200, d. h. 2 Rasterbreiten zu weit rechts und mit 0xFE81A400 1*0x1200, d. h. 1 Rasterhöhe zu tief.

Die Ausrichtung der grafischen Anzeige bedeutet natürlich auch, dass eine Ausrichtung immer am Punkt mit der Länge und Breite von jeweils 0° erfolgt.

Da die DEM-Daten (wenigstens die SRTM-Daten) immer auf volle Grad, also auch auf 0°, ausgerichtet sind, könnte man im einfachsten Fall auch die Punktdistanz der SRTM-Daten verwenden und als Bezugspunkt für die DEM-Datei die Koordinaten des Punktes der am dichtesten links-oberhalb der linken oberen Ecke der TRE-Datei liegt, verwenden. Man bräuchte dann keine Punkte aus den SRTM-Punkten zu interpolieren.

Eine Garmineinheit ist $\frac{360°}{2^{32}} = \frac{3^2 \cdot 5}{2^{29}}$, d.h. der Wertebereich einer 32-Bit-Zahl wird vollständig ausgenutzt um einen vollen Kreis zu beschreiben. Eine Bogensekunde (1“) ist jedoch $\frac{1°}{60 \cdot 60} = \frac{1}{2^4 \cdot 3^2 \cdot 5^2}$. 1“ in Garmineinheiten umgerechnet ergibt $\frac{1}{2^4 \cdot 3^2 \cdot 5^2} : \frac{3^2 \cdot 5}{2^{29}} = \frac{2^{29}}{2^4 \cdot 3^4 \cdot 5^3} = \frac{2^{25}}{3^4 \cdot 5^3} = \frac{33554432}{10125} = 3314{,}018$, also kein ganzzahliges Vielfaches von Garmineinheiten. Der Punktabstand bei 1“ und 3“ SRTM-Daten läßt sich also nicht korrekt in Garmineinheiten darstellen. Welche Abweichungen sind dabei zu erwarten?
Bei 1“ SRTM-Daten liegt die prozentuale Abweichung bei $\frac{33554432}{10125} : 3314 = 1{,}00000542$, also 0,000542%. Da ein Grad am Äquator etwa 111,2km entspricht, beträgt die Abweichung bei 1° etwa 0,6m. Will man z.B. maximal 10m Abweichung zulassen, könnte eine Teilkarte etwa 16,5° groß sein. Für die nächste Teilkarte wird die DEM-Datei dann wieder neu „justiert“.
Es fällt auf, dass Garmin oft die Distanz 0xCF0 = 3312 für die höchste Zoomstufe verwendet. (Vielleicht sollten die 4 niederwertigsten Bits für eine einfachere Grafikberechnung immer 0 sein?) Damit ergeben sich dann analog zur vorherigen Berechnung $\frac{33554432}{10125} : 3312 = 1{,}00061$, also etwa 0,061% bzw. 68m je 1°. Diese Abweichung ist nicht mehr tolerierbar.
Auch wenn Garmin in seinen eigenen Karten oft relativ kleine Ausdehnungen (0,2°x0,33°, aber auch bis zu 0,5°x0,75°) verwendet, scheint diese Abweichung zu groß zu sein.

Der Punktabstand, den Garmin selbst als kleinsten Wert verwendet (3312), liegt zwar in der Nähe einer Bogensekunde, weicht allerdings zur Verwendung für die SRTM-Daten zu stark ab.

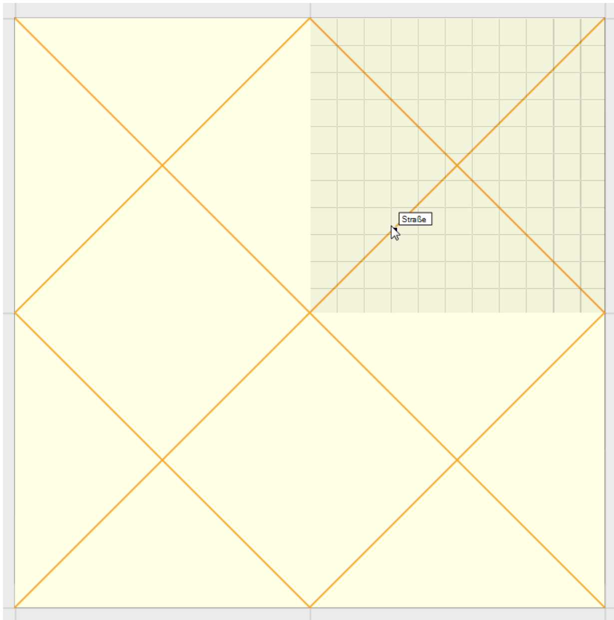
Auch mit dem einer Bogensekunde am nächsten liegenden Punktabstand (3314) sollte die Ausdehnung von Teilkarten begrenzt werden.

Erzeugung einer einfachen Testkarte

Für weitere Experimente empfiehlt sich die Erzeugung einer einfachen Testkarte.

Ich habe z.B. 4 Dateien 99950001.osm, … 99950004.osm mit einem Texteditor erzeugt, die folgende Inhalt haben:

```
<?xml version='1.0' encoding='UTF-8'?>
<osm version='0.6' upload='true' generator='JOSM'>
  <bounds minlat='0.0' minlon='0.0' maxlat='1.0' maxlon='1.0' origin='0.43.1' />
  <node id='1001' timestamp='1969-12-31T23:59:59Z' visible='true' version='1' changeset='1' lat='0.000' lon='0.000' />
  <node id='1002' timestamp='1969-12-31T23:59:59Z' visible='true' version='1' changeset='1' lat='1.000' lon='0.000' />
  <node id='1003' timestamp='1969-12-31T23:59:59Z' visible='true' version='1' changeset='1' lat='1.000' lon='1.000' />
  <node id='1004' timestamp='1969-12-31T23:59:59Z' visible='true' version='1' changeset='1' lat='0.000' lon='1.000' />
  <way id='2001' timestamp='1969-12-31T23:59:59Z' visible='true' version='1' changeset='1'>
    <nd ref='1001' />
    <nd ref='1003' />
    <tag k='highway' v='motorway' />
  </way>
  <way id='2002' timestamp='1969-12-31T23:59:59Z' visible='true' version='1' changeset='1'>
    <nd ref='1002' />
    <nd ref='1004' />
    <tag k='highway' v='motorway' />
  </way>
</osm>
```



Bei den anderen 3 Dateien sind die ID's und Koordinaten jeweils angepasst, so dass 4 einfache Kartenkacheln entstehen, die um den Koordinatenursprung herum angeordnet und jeweils 1 „Quadratgrad“ groß sind.

Mit einer Textdatei „input-files“

```
mapname: 99950001
#description: 99950001
input-file: 99950001.osm

mapname: 99950002
#description: 99950002
input-file: 99950002.osm

mapname: 99950003
#description: 99950003
input-file: 99950003.osm

mapname: 99950004
#description: 99950004
input-file: 99950004.osm
```

erzeugt man mit MKGMAP eine Testkarte:

```
mkgmap --output-dir=map9995 --family-id=9995 --family-name="DEM-Test" --series-name="DEM-Test" --description="Test für DEM" --
mapname=99950001 --overview-mapnumber=99950000 --remove-ovm-work-files --tdbfile -c input-files
```

Diese kann man dann mit einer Kommanddatei

```
set KEY=HKLM\SOFTWARE\Wow6432Node\Garmin\MapSource
if %PROCESSOR_ARCHITECTURE% == AMD64 goto key_ok
set KEY=HKLM\SOFTWARE\Garmin\MapSource
:key_ok
reg ADD %KEY%\Families\FAMILY_9995 /v ID /t REG_BINARY /d 0b27 /f
reg ADD %KEY%\Families\FAMILY_9995\1 /v Loc /t REG_SZ /d "%~dp0\" /f
reg ADD %KEY%\Families\FAMILY_9995\1 /v Bmap /t REG_SZ /d "%~dp0osmmap.img" /f
reg ADD %KEY%\Families\FAMILY_9995\1 /v Tdb /t REG_SZ /d "%~dp0osmmap.tdb" /f
```

in Windows registrieren.

Nachdem man sich vergewissert hat, dass diese einfache Karte in Mapsource angezeigt wird, können die Experimente beginnen. Man zerlegt z.B. die Datei 99950001.img in die 3 Dateien 99950001.TRE, 99950001.RGN und 99950001.LBL. Dann erzeugt man eine DEM-Datei 99950001.DEM und fügt alle 4 99950001-Dateien wieder zusammen. Das Zerlegen und Zusammenführen der IMG-Datei kann z.B. mit gmt erfolgen. Nun muss die Datei OSMMAP.TDB für Mapsource noch angepasst werden. Das Countour- und DEM-Flag muss gesetzt werden und die Datei 99950001.DEM muss registriert werden. (Für alle diese Aufgaben verwende ich mein experimentelles Programm gmtool.)

```
use strict;

# linke obere Ecke
my $left = 0.0;
my $top = 1.0;
# Pixelbreite/-höhe
my $deltalon = 0.000000083819031715393066406 25 * 0xbf4;
my $deltalat = 0.000000083819031715393066406 25 * 0xc6a;
# Tile-Index
my $tile_idxx = 0;
my $tile_idxy = 0;
# Tilebreite in Pixelbreite/-höhe
my $tilewidth = 64;

# "Einrasten"
my $c = int($left / $deltalon) * $deltalon + $deltalon / 2;
while ($c > $left) {
  $c -= $deltalon;
}
$left = $c;

$c = int($top / $deltalat) * $deltalat - $deltalat / 2;
while ($c < $top) {
  $c += $deltalat;
}
$top = $c;

# Ecke links-oben des Tiles
$left += $tile_idxx * 64 * $deltalon;
$top -= $tile_idxy * 64 * $deltalat;

print "<?xml version=\"1.0\" encoding=\"UTF-8\" standalone=\"no\" ?>\n";
print "<gpx xmlns=\"http://www.topografix.com/GPX/1/1\" creator=\"MapSource 6.16.3\" 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\">\n";

for (my $steplat = -1; $steplat < 65; $steplat++) {
  for (my $steplon = -1; $steplon < $tilewidth; $steplon++) {
    my $lat = $top - $steplat * $deltalat;
    my $lon = $left + $steplon * $deltalon;
    my $name = sprintf("%03d%03d", $steplon, $steplat);

    if ($steplat < 0 || 64 <= $steplat ||
       $steplon < 0 || $tilewidth <= $steplon) {
      $name = 'A' . $name;
    }
    SetMarker($name, $lon, $lat);
  }
}
print "</gpx>\n";

sub SetMarker {
my ($name, $lon, $lat) = @_;
  print "<wpt lat=\"$lat\" lon=\"$lon\">";
  print "<name>$name</name>";
  print "<sym>Circle with X</sym>";
  print "</wpt>\n";
}
```

Weitere Infos

Vermutlich müssen die Kacheln den gesamten geografischen Bereich abdecken, der durch das „Hintergrundpolygon“ mit dem Typ 0x4B aus der RGN-Datei beschrieben ist. Meistens dürfte dass der rechteckige Bereich der gesamten Karten-Kachel sein. Speziell am Rand von Karten kann dieses Polygon auch anders geformt sein.

Versuche haben gezeigt, dass bei Nichtabdeckung dieses Polygons merkwürdige Effekte auftreten. Manchmal werden Teilbereiche der Karte nicht vollständig angezeigt, nichtabgedeckte Bereiche z. T. mit (zufälligen ?) Mustern angezeigt oder Mapsource stürzt sogar ab.

Deshalb sollte die linke obere Ecke immer mit der linken oberen Ecke der Karten-Kachel übereinstimmen.

Obwohl die Punktabstände waagerecht und senkrecht einzeln definiert werden, scheinen sie in der Praxis immer identisch (64) zu sein. Das führt natürlich dazu, dass die Höhendaten-Kachel i.A. höher als breit angezeigt wird.

Die unterste Kachelzeile kann vermutlich beliebige Werte von 1 .. 127 annehmen. Die rechte Kachelspalte kann möglicherweise nicht kleiner als 64 sein. Jedenfalls zeigten Versuche dann fehlerhafte Darstellungen. Sie sollte deshalb im Bereich 64 .. 127 sein. Notfalls kann die rechte Spalte und die unterste Zeile auch den benötigten Bereich überschreiten, da die Anzeige dann durch das „Hintergrundpolygon“ mit dem Typ 0x4B aus der RGN-Datei begrenzt wird.

Prinzipiell reicht es (zumindest bei einer einzelnen Karten-Kachel) aus, wenn nur eine einzige Zoomstufe verwendet wird.

Damit die DEM-Daten von Mapsource verwendet werden können, müssen alle DEM-Dateien in der TDB-Datei registriert sein. Außerdem muss das entsprechende Flag der TDB-Datei gesetzt sein. Erst dann wird die Schummerung angezeigt. Zusätzlich muss aber auch das Contour-Flag gesetzt sein, damit auch tatsächlich Höhenwerte angezeigt werden. Es dürfen übrigens auch Kartenkacheln ohne DEM-Datei existieren.

Test der Anzeige mit einer Karte mit 6 Mapleveln + 1 Overviewlevel (24, 23, 22, 21, 20, 18 und 17 Bit):

Die Anzeige der DEM-Zoomlevel ist an die Maplevel der Karte gekoppelt. Fehlt ein passender DEM-Zoomlevel, wird ein anderer verwendet. Deshalb genügt für Mapsource prinzipiell ein einziger DEM- Zoomlevel. Die Overviewmap sollte natürlich auch einen DEM-Zoomlevel haben.

Anzeige in Mapsource bei Kartendetail "mittel":

Maplevel	**Bit**	**„Zoom“**	**DEM-Zoomlevel**
0	24	„Zoom“ ≤ 200m	0
1	23	„Zoom“ = 300m	1
2	22	500m ≤ „Zoom“ ≤ 700m	2
3	21	1km ≤ „Zoom“ ≤ 1,5km	3
4	20	2km ≤ „Zoom“ ≤ 3km	4
5	18	5km ≤ „Zoom“ ≤ 15km	5
6 (Overview)	*17*	*20km ≤ „Zoom“*	*0 (Overviewmap)*

Bei einem GPS-Gerät funktioniert die Anzeige eines DEM-Zoomlevel aber **nur** für den zugehörigen Maplevel. Wenn man für jeden Zoom DEM-Daten anzeigen will, muss auch für jeden Maplevel ein entsprechender DEM-Zoomlevel existieren.

Anzeige mit Oregon 600 bei Kartendetail "normal":

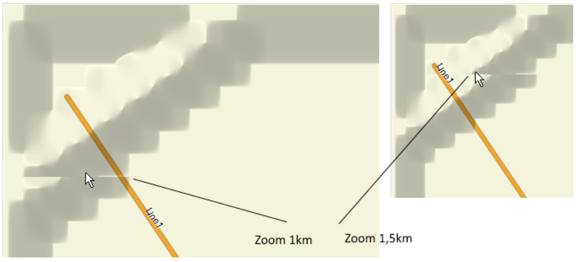
Maplevel	**Bit**	**„Zoom“**	**DEM-Zoomlevel**
0	24	„Zoom“ ≤ 200m	0
1	23	„Zoom“ = 300m	1
2	22	500m ≤ „Zoom“ ≤ 800m	2
3	21	„Zoom“ = 1,2km	3
4	20	2km ≤ „Zoom“ ≤ 3km	4
5	18	5km ≤ „Zoom“ ≤ 80km (?)	5

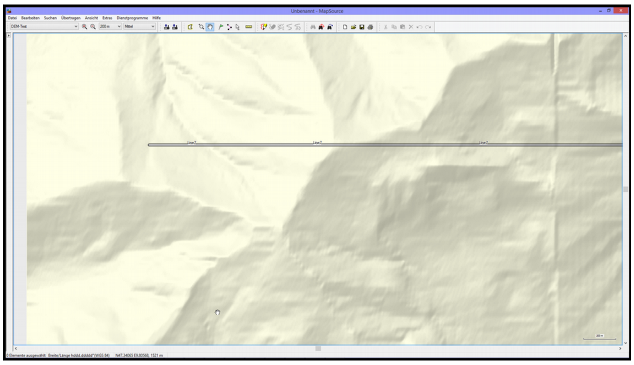
Stand 13.7.2018

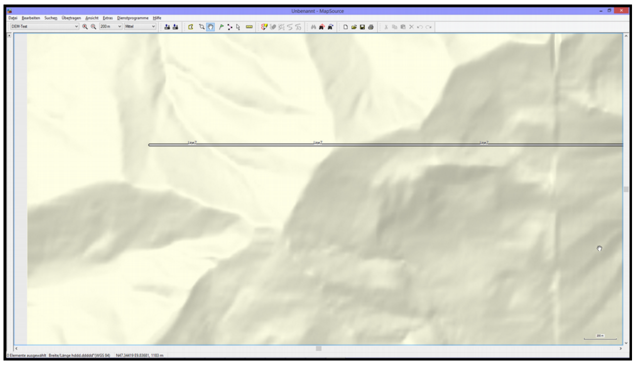
Bei den HGT-Daten ist der Punktabstand für die USA 1“ = 0,000833° und für den Rest der Welt 3“ = 0,0025°. Durch das Interpolationsverfahren beim Einlesen der Daten verringert sich der Abstand zwischen 2 Punkten noch etwas.

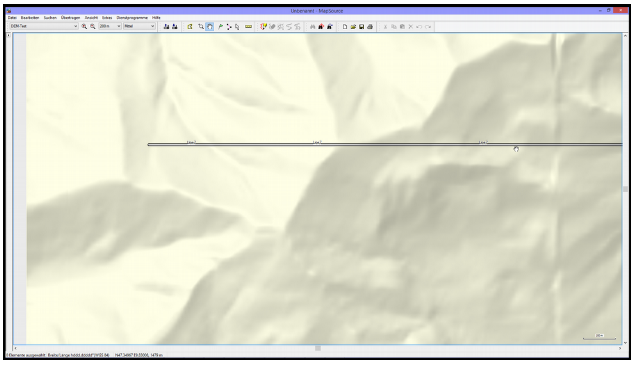
Bei Mapsource/Basecamp wurden z.T. Bildfehler beobachtet. Im folgenden Beispiel wird der gleiche DEM-Zoomlevel für die Anzeige verwendet. Es tritt jeweils ein Fehler auf, jedoch nicht an der gleichen Stelle. Interessanterweise scheint die Anzeige der Höhe an der Cursorposition aber trotzdem korrekt zu sein.

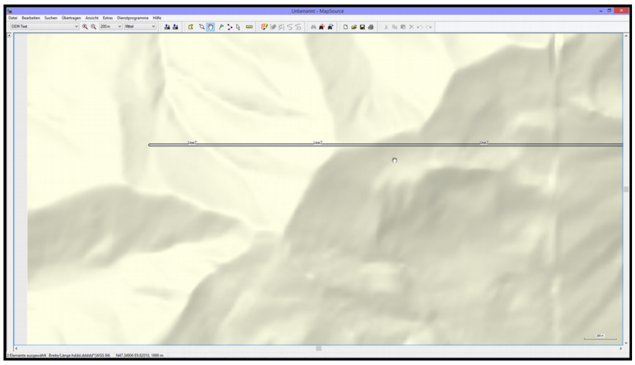
Möglicherweise gibt es auch „bevorzugte“ Punktabstände und andere sollten nicht verwendet werden?

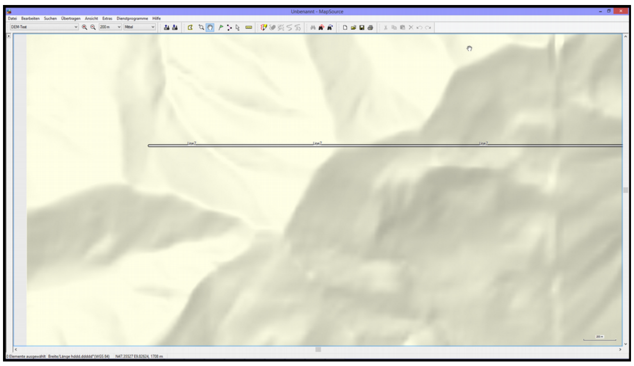


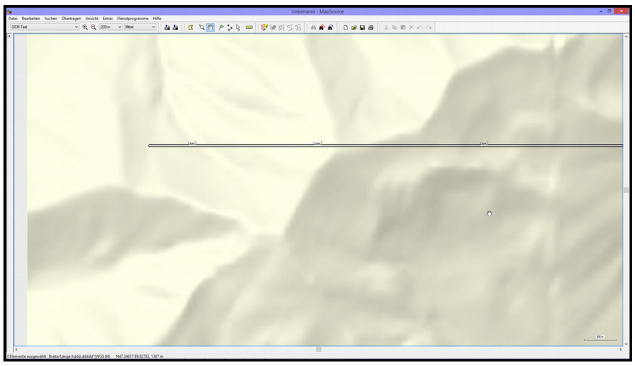

0.0001


0.0002


0.00025


0.00027


0.0003


0.0004

Die Darstellung der Übersichtskarte scheint in Mapsource qualitativ sehr viel schlechter zu sein. Relativ schnell bildet sich ein „Klötzchenmuster“. Deshalb scheint ein Punktabstand unter 0.02 kaum noch sinnvoll zu sein.

Möglichweise reicht für ein GPS-Gerät auch eine allgemeine „Relief-Karte“ ohne anderen Inhalt aus. Diese müsste dann nicht immer neu berechnet werden. Ein Test mit 2 Landeskarten zeigte jedenfalls, dass das über die eine Karte hinausreichende Relief auch für die andere Karte mit angezeigt wird.

Unklares

Header

Die 4 Byte auf Position 0x1B wurden immer nur als 0 gesehen.

Die 4 Byte auf Position 0x25 wurden meist als 1 gesehen, manchmal auch als 0.

Zoomlevel

Es gibt Zoomlevel, bei denen in der Zoomleveltabelle das 1. Byte auf 1 gesetzt und die Nummer gleich der Nummer des vorherigen Zoomlevels ist. Deren Zweck ist völlig unklar. Vielleicht gibt es für spez. Geräte noch ein weiteres Codierformat(?).

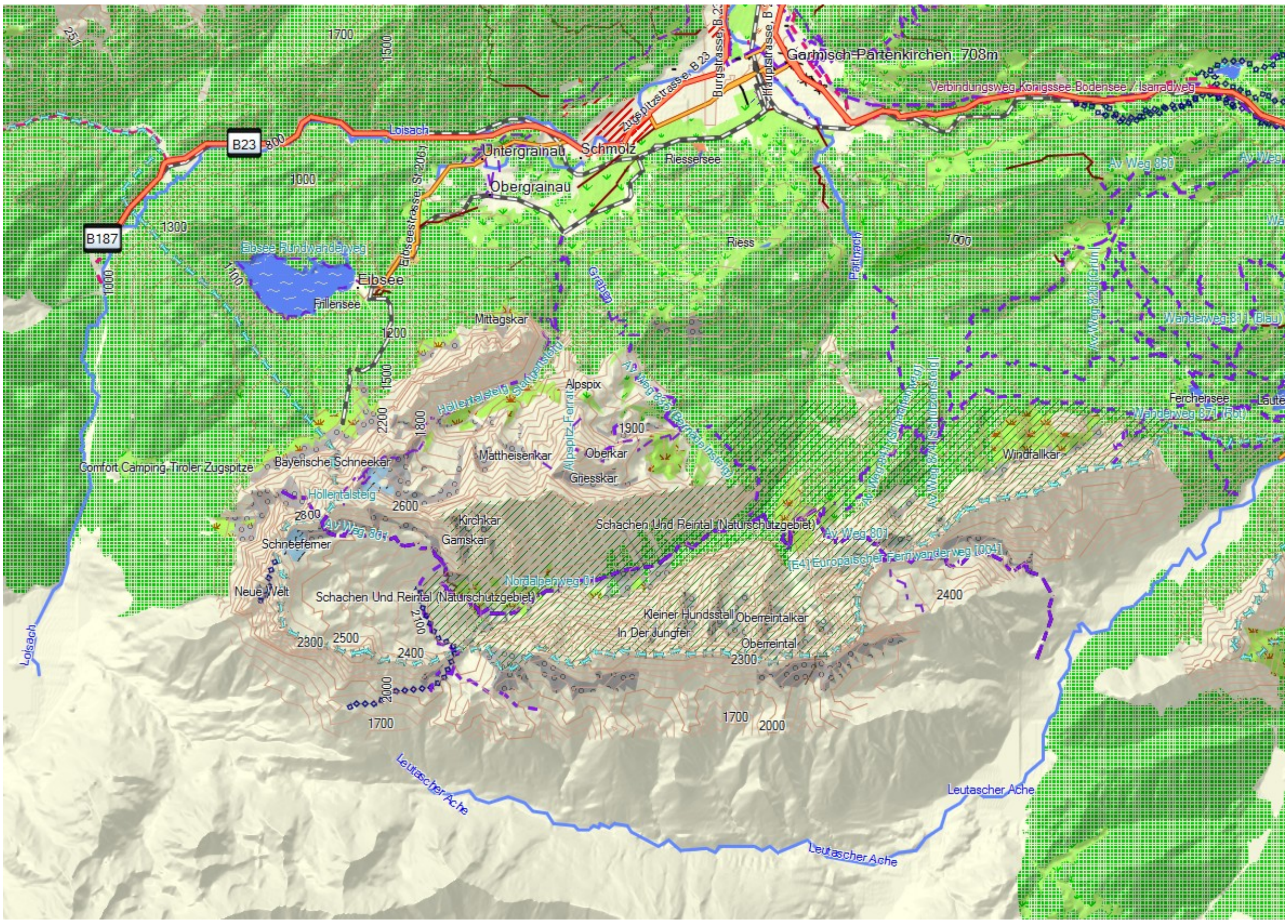
Der Wert auf Position 0x12 ist i.A. 0. Es wurden aber auch die Werte 1, 2, 4 und 20 gesehen. Wenn der Wert 1 ist, scheinen die Höhen immer als das 3fache der Codierung angezeigt zu werden. Außerdem treten die höheren Werte bei größeren Maßstabsfaktoren auf. Es ist zu vermuten, dass damit auf irgendeine Weise größere Werte in kleinere Wertebereiche transformiert werden und damit insgesamt eine bessere Komprimierung bei größeren Maßstabsfaktoren erreicht wird. Die größere Ungenauigkeit spielt dann kaum eine Rolle.

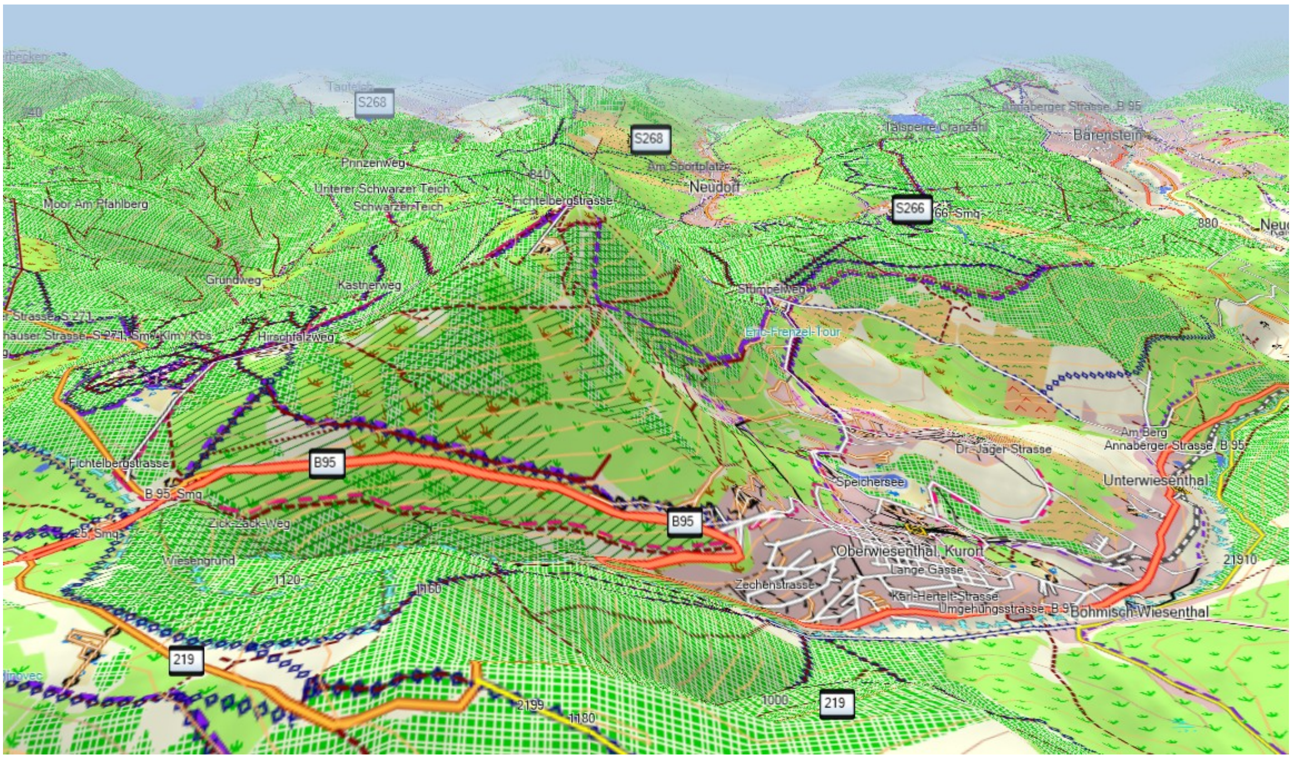
Es wurden keine kleineren Punktabstände als 0xCF0 (3312 → 0,0002776086330413818359375°) gesehen. Das ergibt z.B. für Leipzig (etwa 51,34°N, 12,37°O) etwa 17,5m bzw. 23,8m. Die HGT-Daten haben ein Raster in der Größenordnung von 300m!

Kacheln

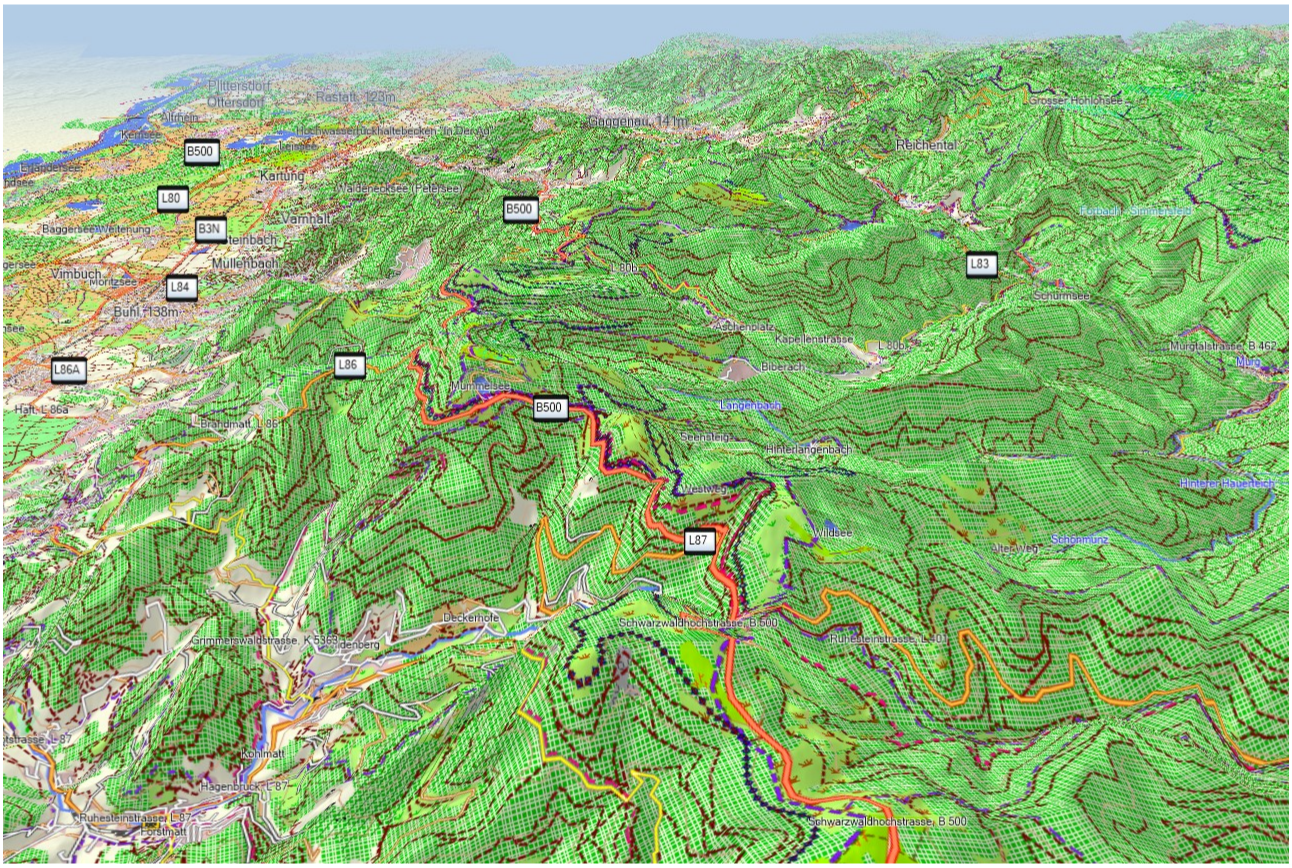
Kacheln mit verringerter Breite scheinen manchmal Probleme bei der Anzeige zu bereiten. Es scheint günstiger zu sein, diese Kacheln auch 64 breit zu erzeugen. Es genügt aber nicht, den „überstehenden“ Teil mit irgendwelchen Werten zu füllen, da nicht vorhersehbar ist, welche IMG-Datei dann die darüberliegende ist. Deshalb sollte auch dieser „überstehenden“ Teil mit echten Werten gefüllt werden.

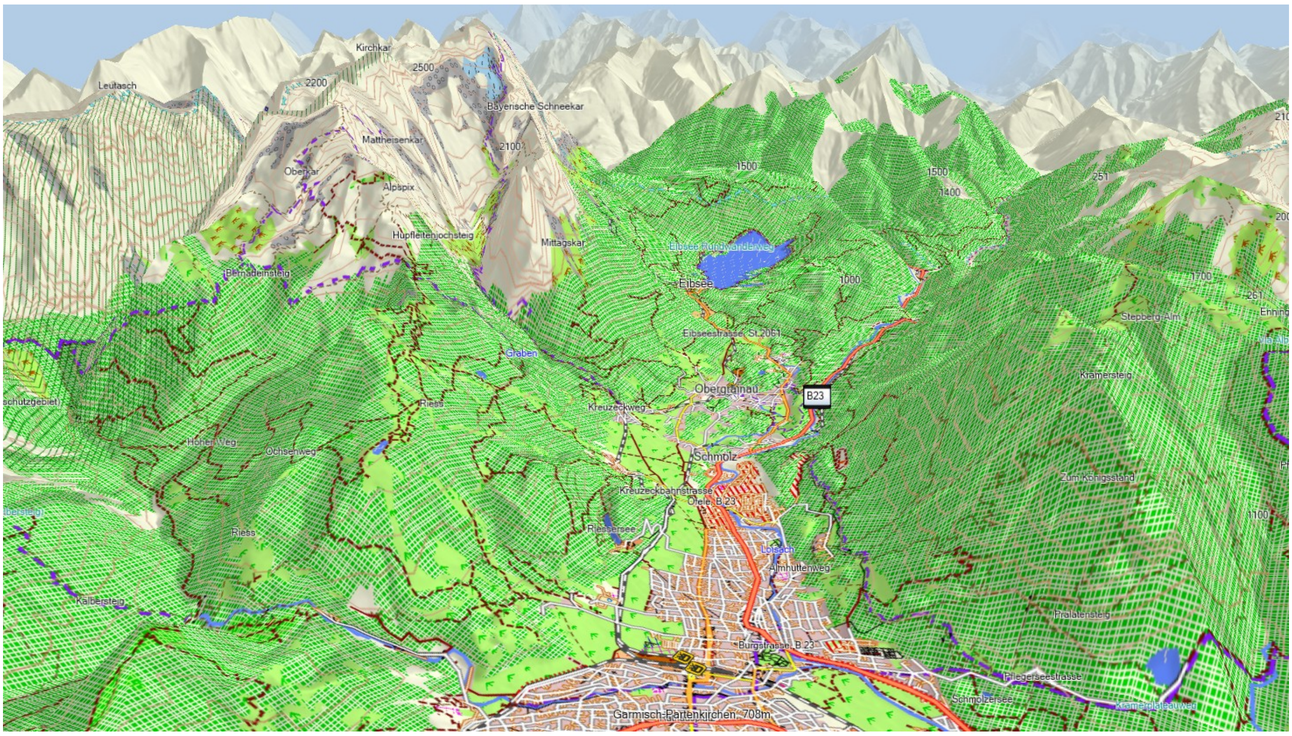
Darstellung mit Basecamp





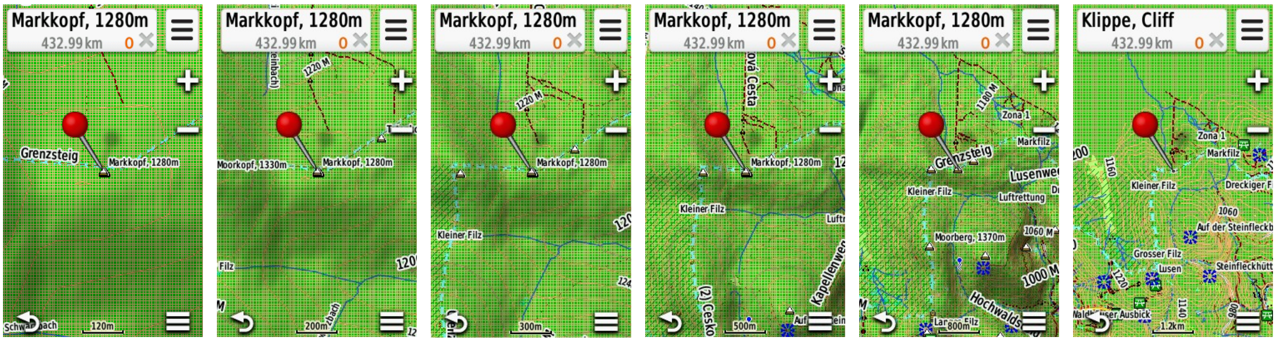
Stand 13.7.2018

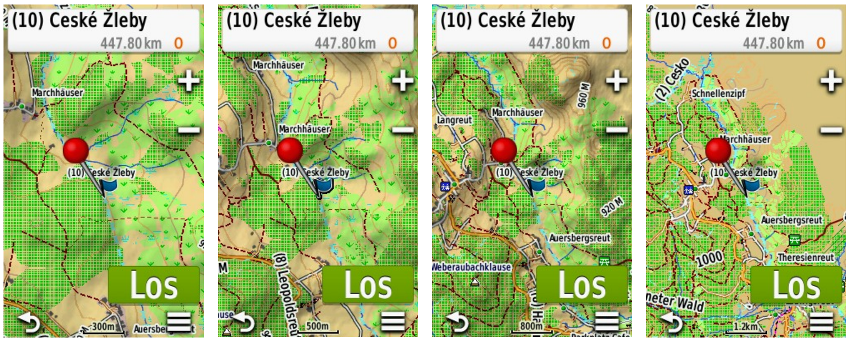




Darstellung mit Oregon 600

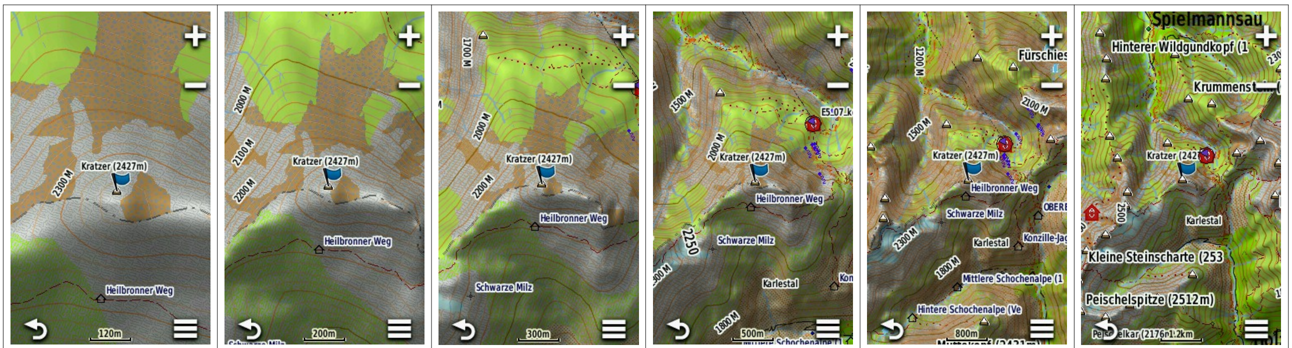
Bei der Einstellung „Karte \ Erweiterte Einstellungen \ Details“ auf „maximal“ und „Karte \ Erweiterte Einstellungen \ Plastische Karte“ auf „zeige wenn verfügbar“ ist das Geländeprofil für den Zoom ≤ 800m sichtbar:

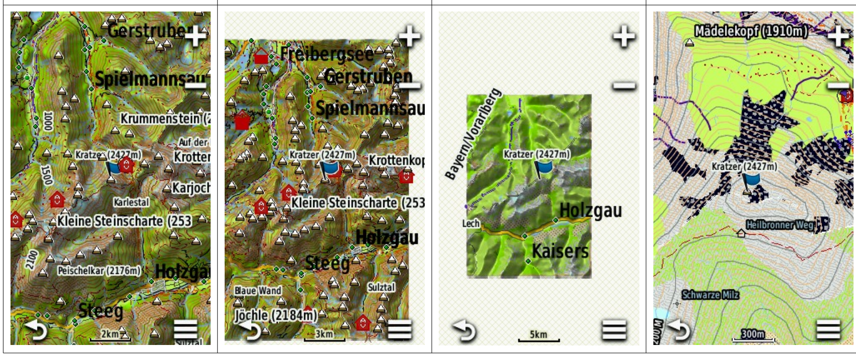




Darstellung „echter“ Garmin-Karten

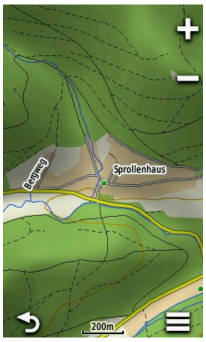
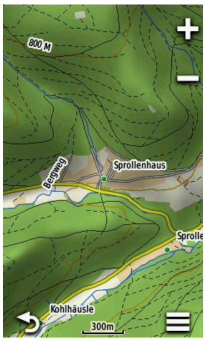
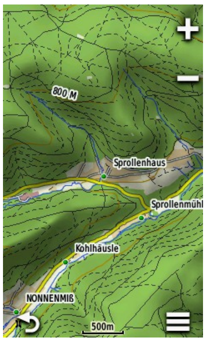
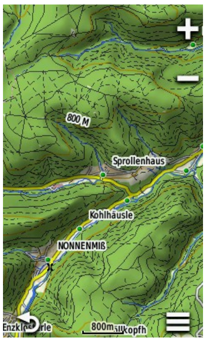
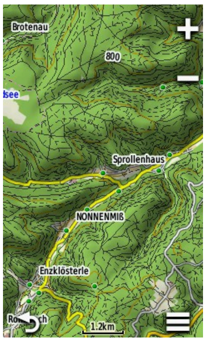
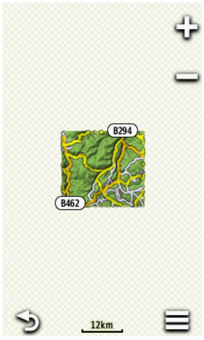
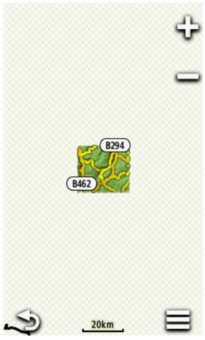
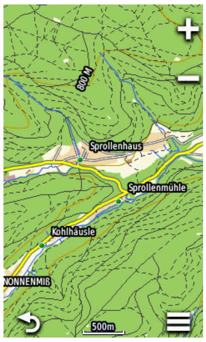
Transalpin





			Nachtmodus		

Topo Light

<table>
<tr>
<td>
</td>
<td>
</td>
<td>
</td>
<td>
</td>
<td>
</td>
<td>
</td>
</tr>
<tr>
<td>
</td>
<td>
</td>
<td>
</td>
<td>
</td>
<td>
</td>
<td>

Nachtmodus</td>
</tr>
<tr>
<td>

Nachtmodus</td>
<td></td><td></td><td></td><td></td><td></td>
</tr>
</table>

Stand 13.7.2018

Garmin-Unit: $1\ GU = 360 / 2^{32}$ 1 Kreisumfang entspricht dem Wert UInt32.Max

Entfernung$_{Latitude}$: $u_{Erde} / 360° = d_{lat} / lat_{delta}$ 1 Erdumfang entspricht 360°

$d_{lat} = lat_{delta} / 360° * u_{erde}$

Entfernung$_{Longitude}$: $u_{Latitude} / 360° = d_{lon} / lon_{delta}$ abh. von der Breite

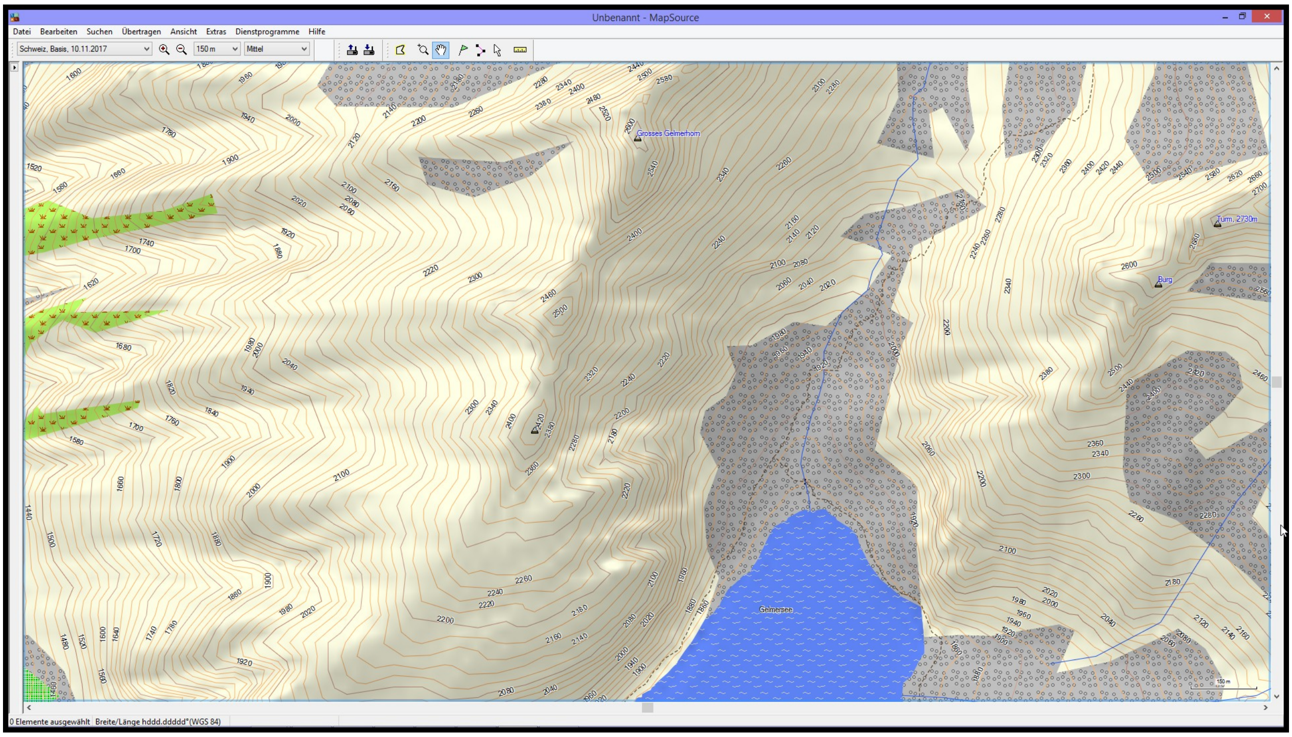
$u_{Erde} * \cos(lon) / 360° = d_{lon} / lon_{delta}$

$d_{lon} = lon_{delta} / 360° * u_{Erde} * \cos(lon)$

Lat in Grad			70	60	50	40	30	20	10	0
Lon_{delta} in Sekunden	in Grad	GU	Entfernung in km							
3600“	1,00000000	11930465	38,0	55,6	71,5	85,2	96,3	104,5	109,5	111,2
1800“	0,50000000	5965232	19,0	27,8	35,7	42,6	48,2	52,3	54,8	55,6
360“	0,10000000	1193046	3,8	5,6	7,1	8,5	9,6	10,5	11,0	11,1
16“	0,00444444	53024	0,169	0,247	0,318	0,379	0,428	0,465	0,487	0,494
8“	0,00222222	26512	0,085	0,124	0,159	0,189	0,214	0,232	0,243	0,247
4“	0,00111111	13256	0,042	0,062	0,079	0,095	0,107	0,116	0,122	0,124
3“	0,00083333	9942	0,032	0,046	0,060	0,071	0,080	0,087	0,091	0,093
1“	0,00027778	3314	0,011	0,015	0,020	0,024	0,027	0,029	0,030	0,031

Stand 13.7.2018

bikubisch interpoliert:



bilinear interpoliert: