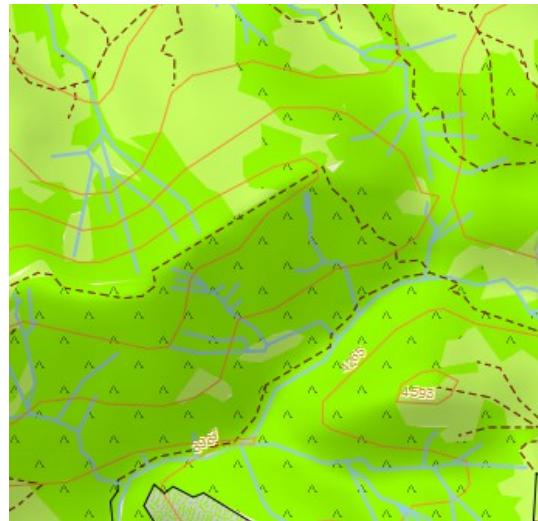


Details zum Aufbau und zur Codierung der Garmin-DEM-Daten



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.

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	Satznummer (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	unbekannt; i.A. 0x00, aber auch 0x100, 0x200, 0x400 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) 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 ($360^\circ / 2^{32}$)	30 3F 7B 09 (= 13,3332°)
0x2C	Int32	nördliche Grenze der DEM-Datei in Units ($360^\circ / 2^{32}$)	50 32 E7 26 (= 54,7075°)
0x30	Int32	Punktabstand senkrecht in Units ($360^\circ / 2^{32}$)	F0 0C 00 00 (= 0,00028°)
0x34	Int32	Punktabstand waagerecht in Units ($360^\circ / 2^{32}$)	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:

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

Für die Beispieldatei ergeben sich folgende Datenbereiche:

0x00	Standard-Header
0x15	DEM-Header
0x29	Tabelle der Kacheldatensätze zum 1. Zoomlevel $((0x1E+1) \cdot (0x1C+1) = 0x383 = 31 * 29 = 899$ Datensätze je 8 Byte) z. B. 00 00 00 00 00 00 xx → ohne Daten BD 0B 00 27 00 1B 00 00 → Daten auf 0xBB (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 $((0x1E+1) \cdot (0x1C+1) = 0x383 = 31 * 29 = 899$ 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

$$\text{Block1Start} + \text{idx} \cdot \text{Block1RecordSize}$$

Die Kachel in der 5. Spalte der 10. Zeile hat die Nummer $9 \cdot 31 + 5 = 284$. 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

$$0x29 + 8 \cdot 0x11B = 0x901$$

für den 2. Datenbereich auf

$$0x40d0e + 6 \cdot 0x11B = 0x413B0$$

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.

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

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 дем
del /Q "%APPDATA%\GARMIN\MapSource\TileCache\*.*"

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
```

Stand 24.10.2017

```
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 $\pm 0,5$ Pixel, also etwa $\pm 2\%$. 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.

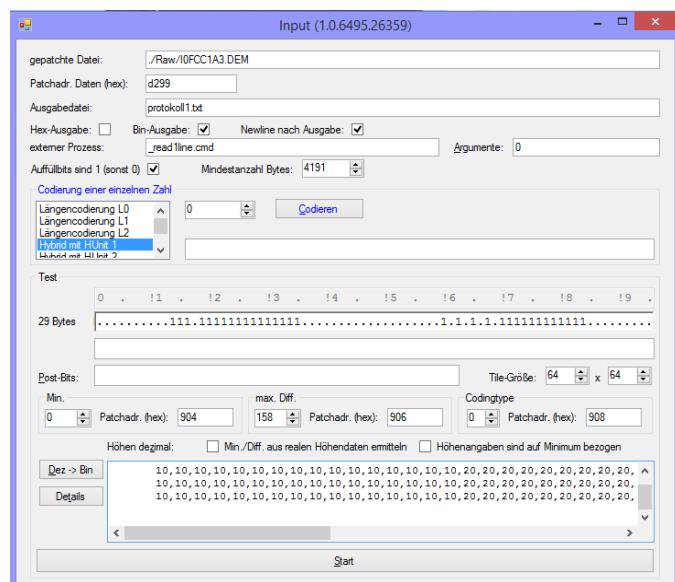
Da die Bitfolge i.A. nicht vollständig zum Ausfüllen des letzten Bytes ausreicht, muss die Art der Auffüllbits angegeben werden. Außerdem wird immer eine Mindestanzahl an Bytes geschrieben. Es hat sich bewährt, als Auffüllbit 1 zu verwenden. Die Zusatzbytes sind also 0xFF.

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.

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 eingesenkt. Am besten trägt man dort 0xFFFF 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: $32768 / 3,28084 = 9987\text{m}$. Bei 64×64 Punkten je Kachel wären das $64 \cdot 64 \cdot 16 = 2^{16}$ Bit = 2^{13} Byte = 2^3 kB = 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 Bitstreams 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	Binär-bits	Wertebereich (außer 0)
0 .. 1	15		unnötig
2 .. 3	16		unnötig
4 .. 8	17		unnötig
8 .. 15	18	4	-8 .. +8
16 .. 31	19	5	-16 .. +16
32 .. 63	20	6	-32 .. +32
64 .. 127	21	7	-64 .. +64
128 .. 255	22	8	-128 .. +128
256 .. 511	25	9	-256 .. +256
512 .. 1023	28	10	-512 .. +512
1024 .. 2047	31	11	-1024 .. +1024
2048 .. 4095	34	12	-2048 .. +2048
4096 .. 8191	37	13	-4096 .. +4096
8192 .. 16383	40	14	-8192 .. +8192
16384 .. 32767	43	15	-16384 .. +16384

Längencodierung

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

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

$$\begin{aligned} -l/2 &\leq v_0 \leq l-l/2 \\ l/2+1-l &\leq v_1 \leq l/2+1 \\ l/2-l &\leq v_2 \leq l/2 \end{aligned}$$

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 v_0 , v_1 und v_2 in den Codierung L0, L1 bzw. L2 die gleiche Länge der 0-Bitfolge haben, wenn folgende Beziehung gilt: $v_0 = 1 - v_1 = -v_2$.
- Die Bitlänge l der 0-Bitfolge zur Zahl v_0 für L0 ist

$$l = 2 \cdot |v_0| - (\text{sgn}(v_0) + 1)/2$$
 und für L1

$$l = 2 \cdot |v_1| + (\text{sgn}(v_1) - 1)/2$$
.
- Aus der Bitlänge l der 0-Bitfolge ergibt sich

$$v_0 = (l \bmod 2) \cdot (l+1)/2 - ((l+1) \bmod 2) \cdot (l/2) \quad \text{bzw.} \quad v_1 = -((l \bmod 2) \cdot (l+1)/2 - ((l+1) \bmod 2) \cdot (l/2) - 1)$$
.

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

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 $0_{s-1} \dots 0_0 1 b_{ld(hunit)-1} \dots b_0 v$. 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).

$$\text{Werte). Die Zahl ergibt sich aus } d = (2 \cdot v - 1) \cdot (s \cdot hunit + \left(\sum_{i=0}^{ld(hunit)-1} b_i \cdot 2^i \right) + v) .$$

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

$$-(l+1) \cdot hunit + 1 \leq d \leq (l+1) \cdot hunit .$$

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 $\rightarrow 3 * 4$

1x1-Bit Trennzeichen

1x1-Bit für 2^1

1x0-Bit für 2^0

1x1-Bit Vorzeichen \rightarrow positiv

$$\rightarrow (2 * 1 - 1) * (3 * 4 + 1 * 2^1 + 0 * 2^0 + 1) = 14$$

$hunit=1$, „11“

0x0-Bits Längencodierung $\rightarrow 0$

1x1-Bit Trennzeichen

1x1-Bit Vorzeichen \rightarrow positiv

$$\rightarrow (2 * 1 - 1) * (0 * 1 + 1) = 1$$

$hunit=1$, „1.“

1x0-Bits Längencodierung $\rightarrow 1 * 1$

1x1-Bit Trennzeichen

1x0-Bit Vorzeichen \rightarrow negativ

$$\rightarrow (2 * 0 - 1) * (1 * 1 + 0) = -1$$

$hunit=1$, „1.“

0x0-Bits Längencodierung $\rightarrow 0$

1x1-Bit Trennzeichen

1x0-Bit Vorzeichen \rightarrow negativ

$$\rightarrow (2 * 0 - 1) * (0 * 1 + 0) = 0$$

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.
 \rightarrow 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 $b_n \dots b_0 b_v$ 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

$$-2^{\text{integer}(ld(max))} \leq d \leq 2^{\text{integer}(ld(max))}$$

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

Für eine Maximalhöhe 158 ergeben sich $\text{integer}(ld(158)) = \text{integer}(7,3) = 7$ 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:

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

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.

Die Grenzwerte, ab denen ein Wraparound sinnvoll ist, sind von der maximalen Höhendifferenz max abhängig. Für die Längencodierungen gilt:

max gerade:	$v0 < -max/2$	oder	$max/2 < v0$
	$v1 < -max/2$	oder	$(max+2)/2 < v1$
	$v2 < -max/2$	oder	$max/2 < v2$
max ungerade:	$v0 < -(max-1)/2$	oder	$(max+1)/2 < v0$
	$v1 < -(max-1)/2$	oder	$(max+1)/2 < v1$
	$v2 < -(max+1)/2$	oder	$(max-1)/2 < v2$

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

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

Wraparound wird immer verwendet, wenn dadurch betragsmäßig kleinere Zahlen entstehen. Das ist wichtig bei der $hunit$ -Berechnung.

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 immer die Hybridecodierung verwendet. $hunit$ wird aus der maximalen Höhendifferenz der Kachel abgeleitet (siehe Tabelle).

Es wurden 2 Varianten gefunden.

Variante HStd (Hybridcodierung für Standardwerte und Plateau-Nachfolger mit $ddiff \neq 0$):

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 (Vorgängeranzahl vg) eingetragen. Die Tabellenwerte ergeben sich aus

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

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 $minsumHStd_{vg} < (vg + 1) \cdot 1 - 1 \rightarrow minsumHStd_{vg} < vg$), wird der aktuelle Wert mit L0 oder L1 codiert.

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.

Aus der Gleichung für $minsumHStd_{hunit, vg}$ folgt

$$hunit_{vg} = \frac{minsumHStd_{vg} + 1}{vg + 1} . \text{ Gleichzeitig ist } hunit \text{ aber immer}$$

die größte passende 2er Potenz. Deshalb kann $hunit$ folgendermaßen berechnet werden:

$$hunit = 2^{\text{integer}\left(\text{ld}\left(\frac{asum_{vg} + 1}{vg + 1}\right)\right)} \quad (\text{„integer“ liefert den ganzzahligen Anteil der Zahl.})$$

Beispiele:

Datenwerte „5, 7, 3“ →
 $hunit = 2^{\text{integer}\left(\text{ld}\left(\frac{5+7+3+1}{3+1}\right)\right)} = 2^{\text{integer}\left(\text{ld}(4)\right)} = 2^2 = 4$

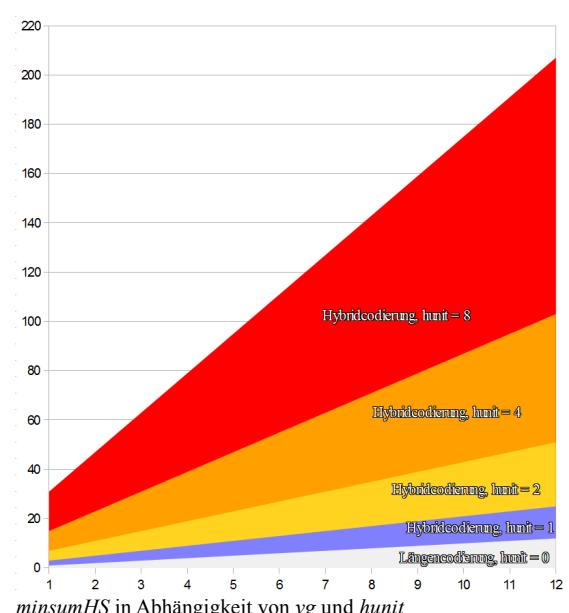
Datenwerte „5, 6, 3“ →
 $hunit = 2^{\text{integer}\left(\text{ld}\left(\frac{5+6+3+1}{3+1}\right)\right)} = 2^{\text{integer}\left(\text{ld}(3,75)\right)} = 2^1 = 2$

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- $hunit = 1$)

maximalen Differenz zum Basiswert	hunit für d_0
Hex	dez
... 0x9e	... 158
0x9f ... 0x11e	159 ... 286
0x11f ... 0x21e	287 ... 542
0x21f ... 0x41e	543 ... 1054
0x41f ... 0x81e	1055 ... 2078
0x81f ... 0x101e	2079 ... 4126
0x101f ... 0x201e	4127 ... 8222
0x201f ... 0x401e	8223 ... 16414
0x401f ... 0x7fff	16415 ... 32767
	$2^8 = 256$

Tabelle der $hunit$ für d_0



Datenwerte „1, 1, -1“ →

$$hunit = 2^{\text{integer}(\text{ld}(\frac{1+1+1+1}{3+1}))} = 2^{\text{integer}(\text{ld}(1))} = 2^0 = 1$$

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 Gesamtbreite 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:

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

Damit gilt insgesamt

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

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:

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

$hunit_2 = 32$ 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:

$maxdiff = 0x21f \rightarrow starthunit = 8, hunitdelta = 7$

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

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

Variante HPI0 (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.

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 $\text{integer}(\text{vg}/2)$ addieren.

$$\begin{aligned} \text{minsumHPl0}_{\text{hunit}, \text{vg}} &= \text{minsumHStd}_{\text{hunit}, \text{vg}} + \text{integer}(\text{vg}/2) \\ \text{minsumHPl0}_{\text{hunit}, \text{vg}} &= (\text{vg}+1) \cdot \text{hunit} - 1 + \text{integer}(\text{vg}/2) \end{aligned}$$

und hunit damit aus

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

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- $\text{hunit} = 1$)

Art der Längencodierung

Wie bereits erwähnt, wird die Längencodierung immer dann verwendet wenn bei der spezifischen Summenbildung nicht der notwendige Mindestwert für $h_{unit} = 1$ erreicht wird.

Standardwerte

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:

$$\begin{array}{lll}
 4 - (valsum_{vg} - 3 \cdot vg) / 2 & \leq d & val = 1 - valsum_{vg} + vg \\
 2 - (valsum_{vg} - vg) / 2 & \leq d < 4 - (valsum_{vg} - 3 \cdot vg) / 2 & val = 2 \cdot (d - vg) - 5 \\
 0 - (valsum_{vg} + vg) / 2 & \leq d < 2 - (valsum_{vg} - vg) / 2 & val = 2 \cdot d - 1 \\
 -2 - (valsum_{vg} + 3 \cdot vg) / 2 & \leq d < 0 - (valsum_{vg} + vg) / 2 & val = 2 \cdot (d + vg) + 3 \\
 d < -2 - (valsum_{vg} + 3 \cdot vg) / 2 & & val = -1 - valsum_{vg} - vg
 \end{array}$$

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

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.

Für betragsmäßig relativ große Werte d hängt die Bewertung dagegen nur noch von $valsum_{vg}$ und 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 $-(vg+1) \dots (vg+1)$ an. Durch den beschriebenen Algorithmus verlässt $valsum$ also niemals diesen Bereich.

ACHTUNG

Immer bevor der 64. Wert bewertet wird, wird er unter den folgenden Bedingungen zunächst verändert:

Ist die Summe bisher positiv und

- die Summe + 1 ist durch 4 teilbar und der Wert ist ungerade oder
- die Summe + 1 ist nicht durch 4 teilbar und der Wert ist gerade

wird der Wert um 1 verkleinert.

Ist die Summe bisher negativ und

- die Summe - 1 ist durch 4 teilbar und der Wert ist ungerade oder
- die Summe - 1 ist nicht durch 4 teilbar und der Wert ist gerade

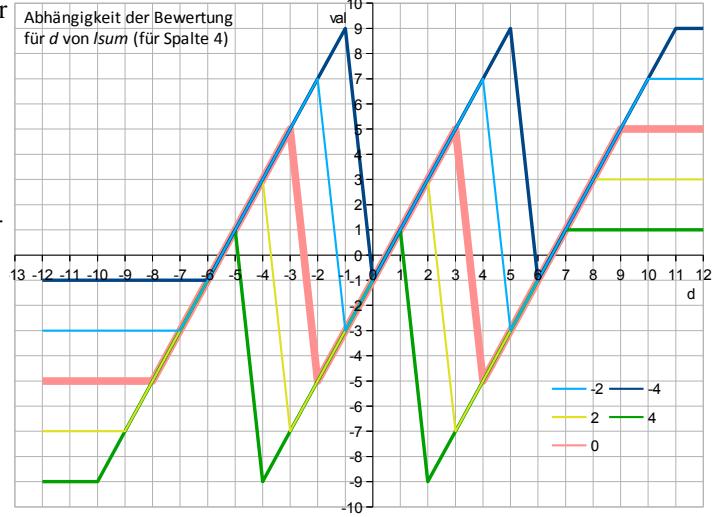
wird der Wert um 1 vergrößert.

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 $suml_{32} = \text{integer}(suml_{64}/2)$. 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 $d(\text{col}, \text{line})$ 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 $hdif(n, m) = h(n, m) - h(n-1, m)$,

für die „vertikale“ Differenz $vdif(n, m) = h(n, m) - h(n, m-1)$

und für die „diagonale“ Differenz $ddif(n, m) = h(n, m-1) - h(n-1, m)$ (immer „aufwärts diagonal“).

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

Daraus folgt: $hdif(n, m) = ddif(n, m) + vdif(n, m)$.

n	0	1	2	3	4	m
0	0	5	15	5	-5	...
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

Standardwerte

Für Standardwerte muss gelten: $ddiff(n, m) \neq 0$, also Vorgängerhöhe \neq 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:

$hdiff(n, m-1) \geq max - h(n-1, m)$:	$d(n, m) = -sgn(ddiff(n, m)) \cdot (h(n, m) + 1)$
$hdiff(n, m-1) \leq -h(n-1, m)$:	$d(n, m) = -sgn(ddiff(n, m)) \cdot h(n, m)$
sonst:	$d(n, m) = -sgn(ddiff(n, m)) \cdot (hdiff(n, m) - hdif(n, m-1))$

$$hdiff(n, m-1) = max - h(n-1, m) + x \quad \text{und } hdif(n, m-1) \geq 0; x \geq 0$$

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

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

Hinweis:

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

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

Beispiel:

$$\begin{aligned} \dots, 5, 7, 3, 2, \dots \\ \dots, 6, 2, 5, 4, \dots \\ h(n-1, m) < -hdif(n, m-1) \\ 6 < -(7 - 5) &\rightarrow \text{nein} & -\text{sgn}(7 - 6) \cdot ((2 - 6) - (7 - 5)) = -(0 - 2) = 2 \\ 2 < -(3 - 7) &\rightarrow \text{ja} & -\text{sgn}(3 - 2) \cdot 4 = -4 \\ 5 < -(2 - 3) &\rightarrow \text{nein} & -\text{sgn}(2 - 5) \cdot ((4 - 5) - (2 - 3)) = (-1 - -1) = 0 \end{aligned}$$

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

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
...

Achtung: Plateaus können auch über mehrere Zeilen reichen.

Da in der 1. Spalte wegen der virtuellen Spalte immer $ddiff = 0$ gilt, beginnt jede Zeile mit einem Plateau, wenn sich dort nicht schon ein „überlanges“ Plateau aus der vorherigen Zeile befindet.

In der 1. Zeile beginnt wegen der virtuellen Zeile auch nach jeder Höhe 0 ein Plateau.

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.

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

Natürlich muss bei $ddiff = 0$ auch $vdiff(n, m) \neq 0$ gelten, da sonst die Plateaulänge einfach nur zu kurz wäre.

Bei Plateaus mit $ddiff(n, m) = 0$ wird die Hybridcodierung HPI0 bzw. die Längencodierung L0 oder L1 verwendet.

Bei Plateaus mit $ddiff(n, m) \neq 0$ 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: $1_s \dots 1_0 b_k \dots b_0$. 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	7

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

Beispiele:

Tabellenposition = 0 (1. Plateaulänge)

$$0 + 0 = 0$$

.

$$2 * 1 = 2$$

11.

$$4 * 1 + 0 = 4$$

11111..

$$4 * 1 + 1 * 2 + 1 = 7$$

111111.1

Stand 24.10.2017

$$\begin{array}{ll} 1111111111111.1.1 & 4 * 1 + 4 * 2 + 4 * 4 + 1 * 8 + 5 = 41 \\ 1111111.. & 4 * 1 + 3 * 2 + 0 = 10 \end{array}$$

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 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:

$$\begin{array}{ll} \text{Tabellenposition } = 0 \text{ (1. Plateaulänge)} & \\ 1111111.. & 4 \cdot 1 + 3 \cdot 2 + 0 = 10 \\ \text{dann 2. Plateaulänge mit Tabellenposition } = 6 & \\ 1111..1 & 2 \cdot 2 + 2 \cdot 4 + 1 = 13 \end{array}$$

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 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 $4 \cdot 1 + 4 \cdot 2 + 4 \cdot 4 + 4 \cdot 8 + 16 = 76$ 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.

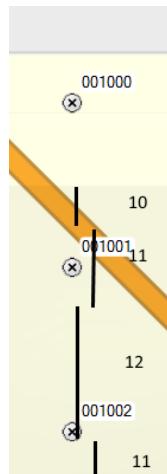
Die restlichen 2 Bits „1.“ stellen eine Plateaulänge für den Rest der Zeile mit der aktuellen Höhe 0 dar. Die Binärbits und der Nachfolger können offensichtlich entfallen.

Geografische Ausrichtung der DEM-Daten

Bei einer Testkarte wurde versucht, die DEM-Daten auf verschiedene Art „einzupassen“. Die nördliche Grenze der TRE-Daten ist 0xB60B. Diese 24-Bit-Zahl steht für $0xB60B \cdot 360,0^\circ / 0x1000000 = 0,999991893768310546875^\circ$. Die geografischen DEM-Grenzen werden als 32-Bit Zahl gespeichert. Der analoge Wert ist $0xB60B \cdot 0x100 = 0xB60B00$. Als DEM-Pixelhöhe wurde 0xBA6 verwendet. Als nördliche Grenze der DEM-Daten wurde $0xB601C0 + \text{delta}$ verwendet.

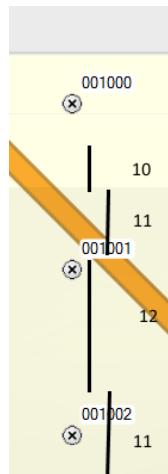
$$\text{delta} = 0$$

$$0xB601C0$$



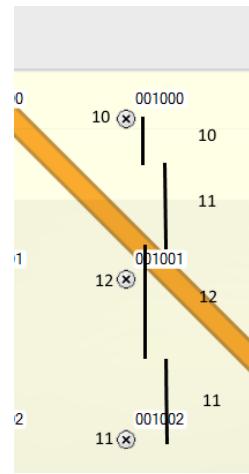
$$\text{delta} = \frac{1}{4} \cdot 0xBA6$$

$$0xB604A9$$



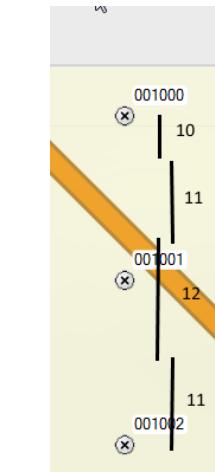
$$\text{delta} = \frac{1}{2} \cdot 0xBA6$$

$$0xB60793$$



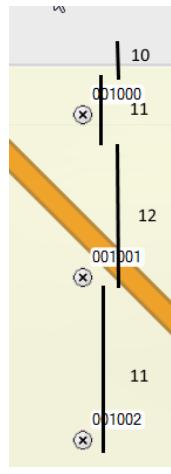
$$\text{delta} = \frac{1}{2} \cdot 0xBA6 + 1$$

$$0xB60794$$



$$\text{delta} = \frac{3}{4} \cdot 0xBA6$$

$$0xB60D67$$



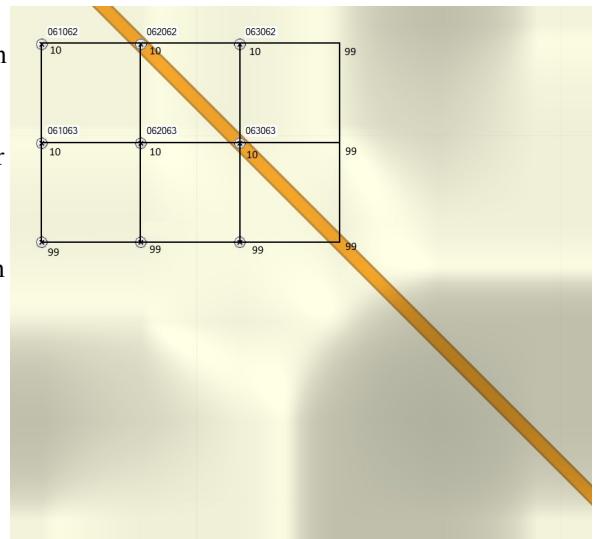
Bei der Codierung wurde der oberste Punkt (1. Zeile) auf 10, der mittlere (2. Zeile) auf 12 und der unterste auf 11 gesetzt. Skizziert wurde jeweils die angezeigte Höhe. Über dem 10er-Bereich erfolgt **keine** Anzeige mehr. Diese Grenze ist nicht (!) identisch mit dem Bereich mit der Höhenschattierung. Die Anzeige der Höhen bleibt, abgesehen von der vertikalen Verschiebung, konstant. Die DEM-Daten passen am besten mit dem TRE-Bereich zusammen, wenn sie die (hier nicht dargestellte) gleiche nördliche Grenze wie die TRE-Daten haben, also 0xB60B00. Interessant ist aber auch, dass die Höhenangaben der 1. Zeile der DEM-Daten offensichtlich direkt auf dem oberen Rand des DEM-Bereiches liegen. Analog gilt das für den linken Rand.

Stellt man sich ein DEM-Pixel als Rechteck vor, so gilt dessen Höhe offensichtlich für die linke obere Ecke dieses Pixels und nicht etwa für dessen Mittelpunkt. Das muss bei der Ermittlung der Länge und Breite der Höhe eines Pixels berücksichtigt werden.

Beim Überschreiten von $\frac{1}{2}$ der Pixelhöhe für delta „springt“ der Bereich mit der Höhenschattierung um 1 Pixelhöhe nach oben.

Im nebenstehenden Bild ist die rechte untere Ecke einer DEM-Kachel dargestellt, deren Höhe konstant 10 ist. Die angrenzenden Kacheln haben auch die Höhe 10, aber jeweils einen 1 Pixel breiten Rand mit 99.

In der Fläche der letzten Pixelspalte und -zeile erfolgt in der 10er Kachel eine Interpolation zwischen 10 und 99.



- Die DEM-Daten sollten die gleichen Koordinaten für den linken und oberen Rand erhalten wie die TRE-Daten.
- Länge und Breite für die Höhenbestimmung des DEM-Pixels links oben sind identisch mit der Ecke links oben der DEM-Daten.
- Alle weiteren Längen und Breiten ergeben sich durch die Breite und Höhe der DEM-Pixel.

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.

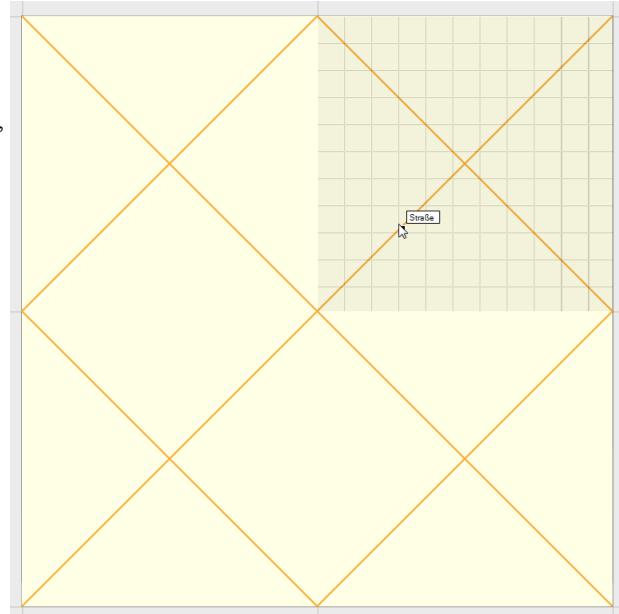
Mit einer Textdatei „input-files“

```
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 Kommandodatei

```
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.)

Stand 24.10.2017

```
use strict;

# linke obere Ecke
my $left = 0.0;
my $top = 1.0;
# Pixelbreite/-höhe
my $deltalon = 0.00000008381903171539306640625 * 0xbff4;
my $deltalat = 0.00000008381903171539306640625 * 0xc6a;
# Tile-Index
my $tile_idxxx = 0;
my $tile_idxxy = 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_idxxx * 64 * $deltalon;
$top -= $tile_idxxy * 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 Kachelpalte 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.

Es ist vielleicht nicht der effektivste Weg, aber in der Praxis scheint für Mapsource/Basecamp ein einzelner Zoomlevel zu genügen. Für jeden beliebigen Zoom wird die Darstellung aus dem 1. Zoomlevel abgeleitet. Damit beim Übergang zur Overviewmap auch ein Relief angezeigt wird, muss diese IMG-Datei auch DEM-Daten erhalten. Der Punktabstand kann aber wegen des viel größeren Maßstabs auch entsprechend groß gewählt werden.

Für ein GPS-Gerät (getestet mit „Oregon 600“) sind jedoch mehrere Zoomlevel nötig. Mit der Einstellung „Karte \ Erweiterte Einstellungen \ Details“ auf „maximal“ und „Karte \ Erweiterte Einstellungen \ Plastische Karte“ auf „zeige wenn verfügbar“ ist das Geländeprofil bei folgenden Punktabständen für den folgenden Zoom sichtbar:

Anzeige 1. Zoomlevel bis 800m	Punktabstand: 0,000277609°
Anzeige 2. Zoomlevel von 1,2 bis 3km	Punktabstand: 0,001110435°
Anzeige 3. Zoomlevel von 3 bis 12km	Punktabstand: 0,002222210°
Anzeige 4. Zoomlevel von 12 bis min. 80km	Punktabstand: 0,004444420°

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!

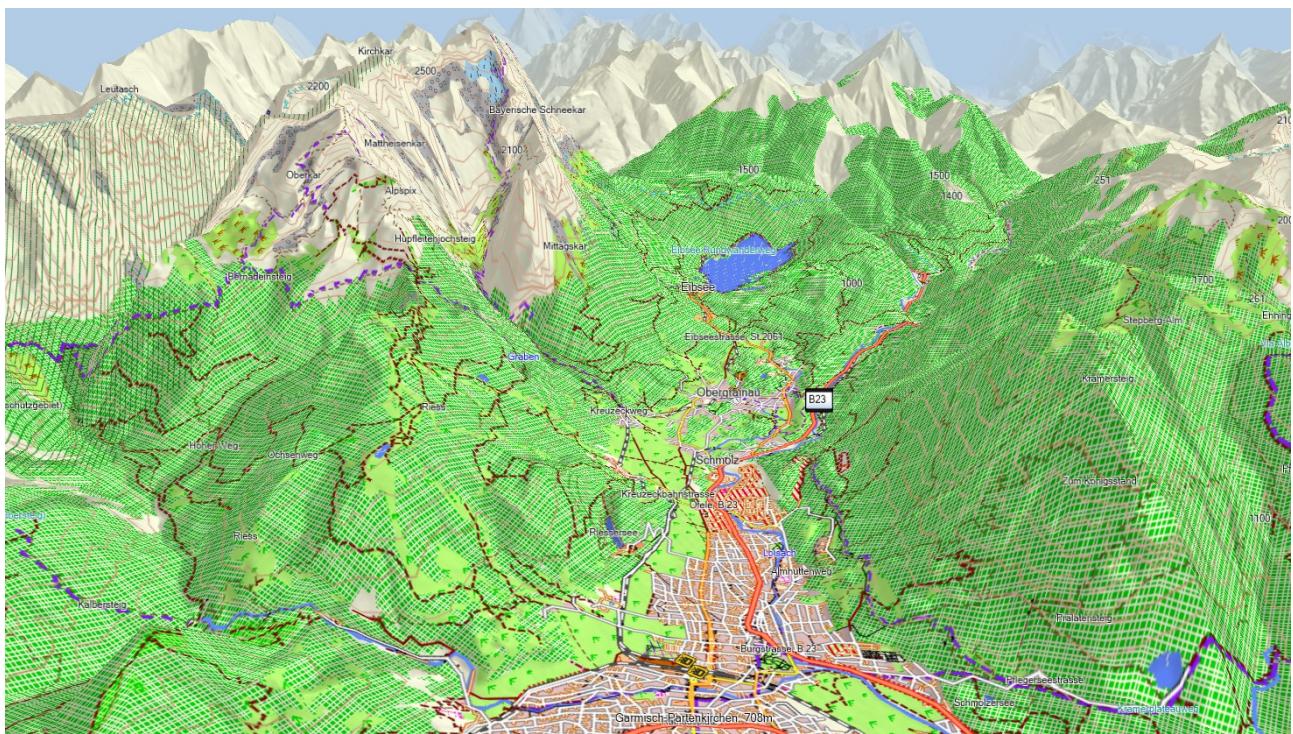
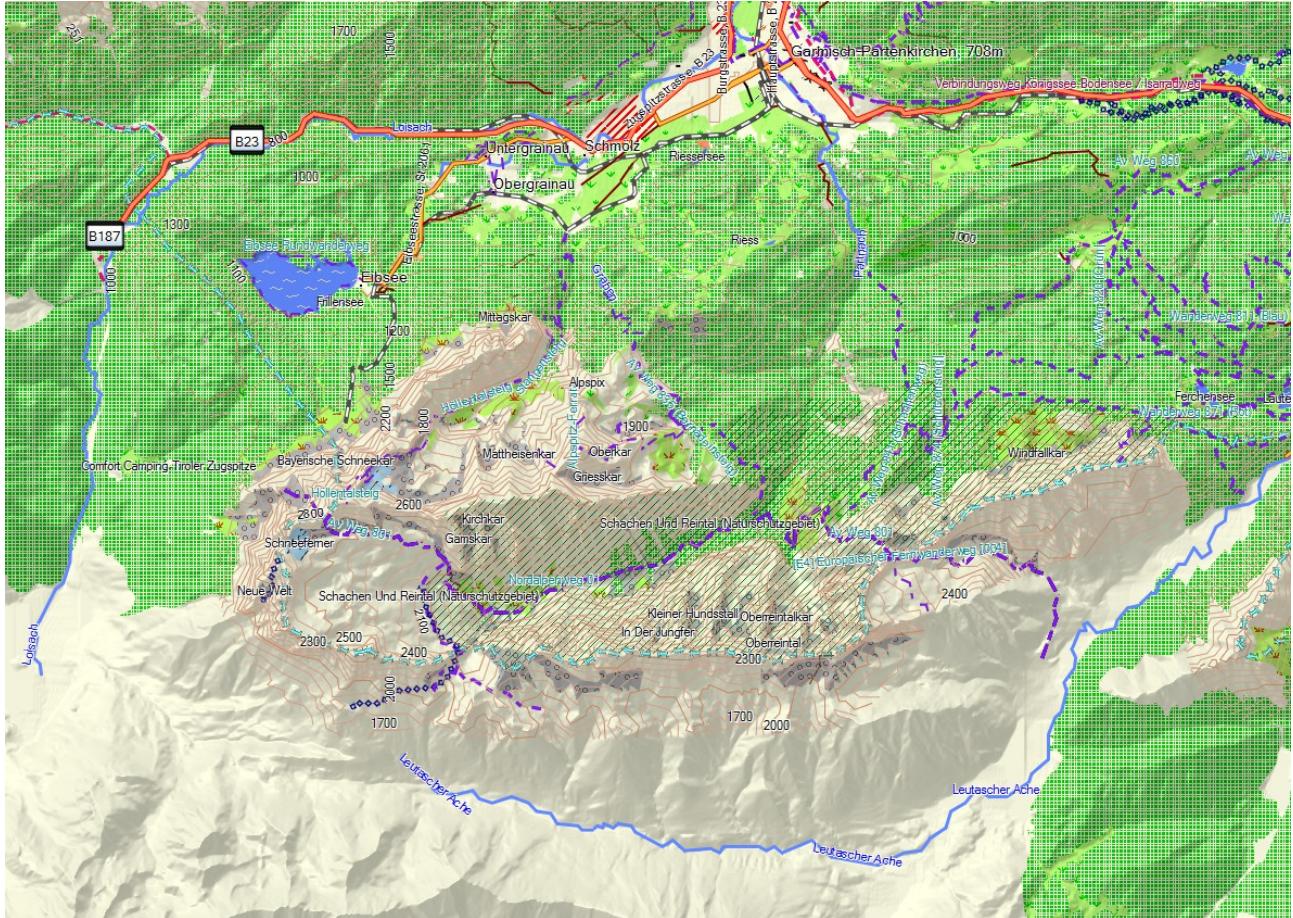
Stand 24.10.2017

Kacheln

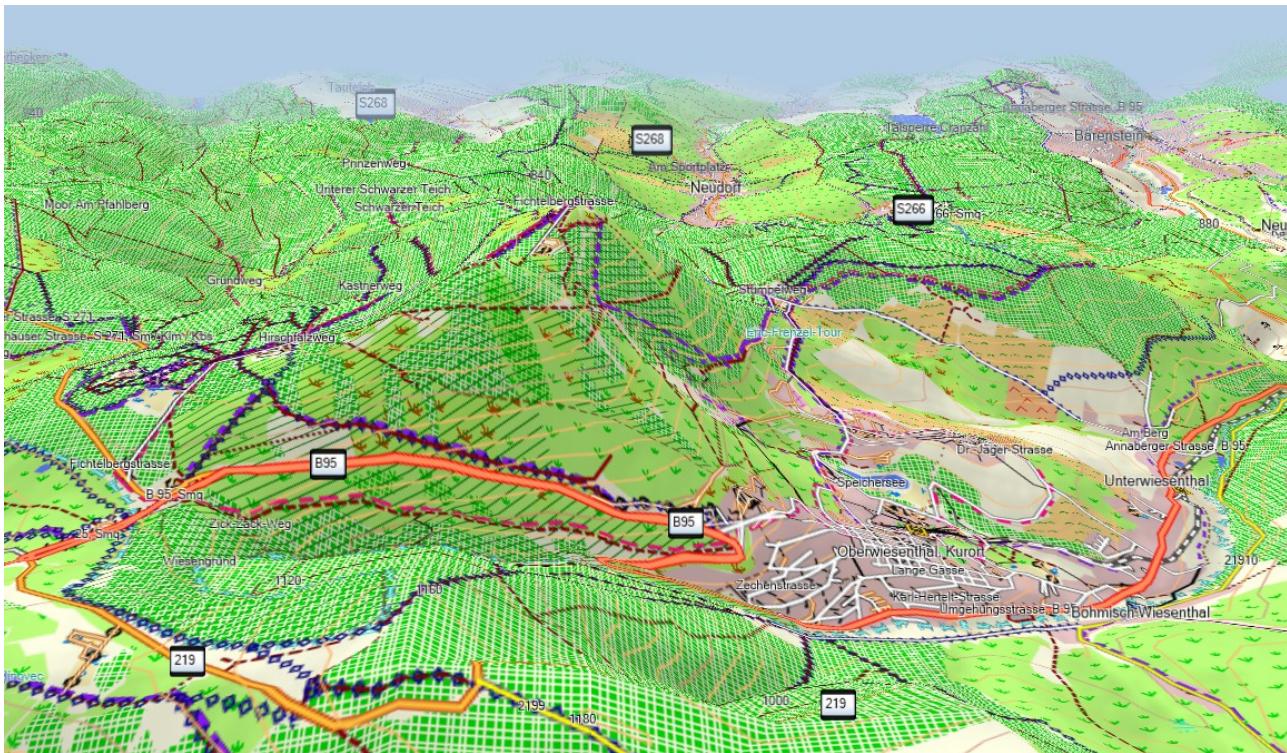
Kacheln mit verringelter Breite scheinen manchmal Probleme bei der Anzeige zu bereiten. Es scheint günstiger zu sein, diese Kacheln auch 64 breit zu erzeugen. Es genügt aber nicht, den „überstehenden“ Teil mit irgendwelchen Werten zu füllen, da nicht vorhersehbar ist, welche IMG-Datei dann die darüberliegende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

Stand 24.10.2017

Darstellung mit Basecamp



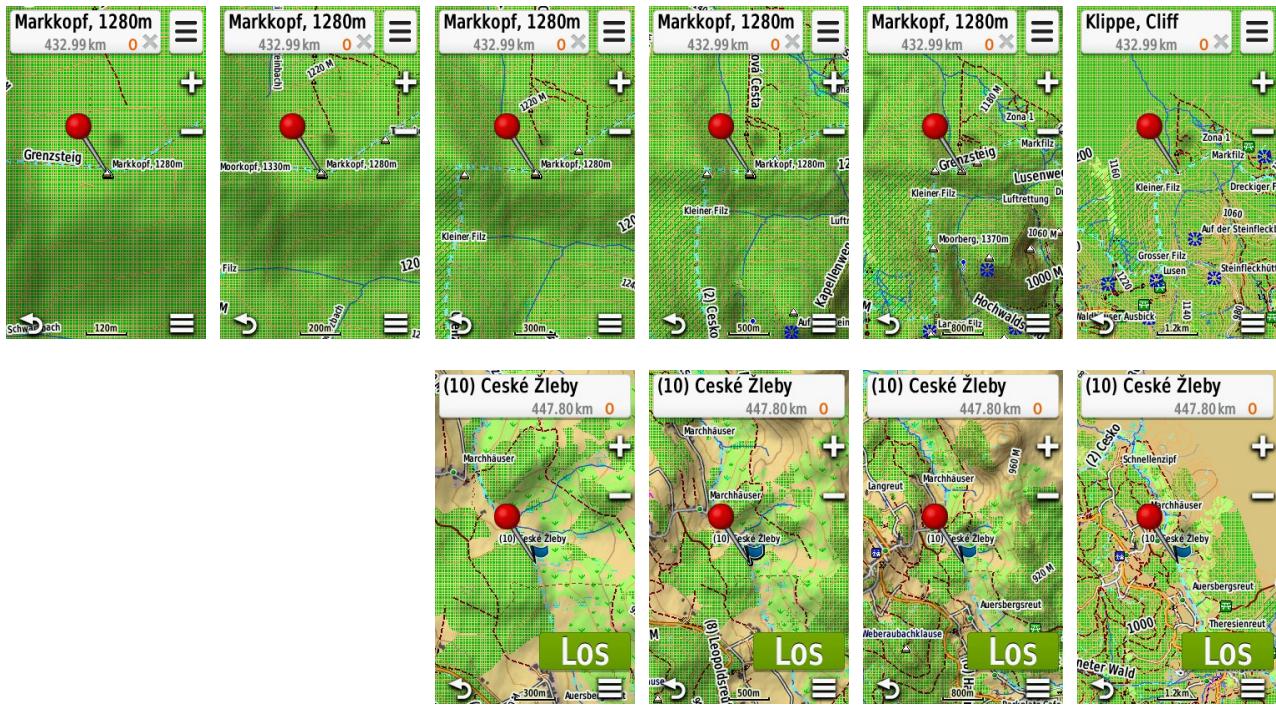
Stand 24.10.2017



Stand 24.10.2017

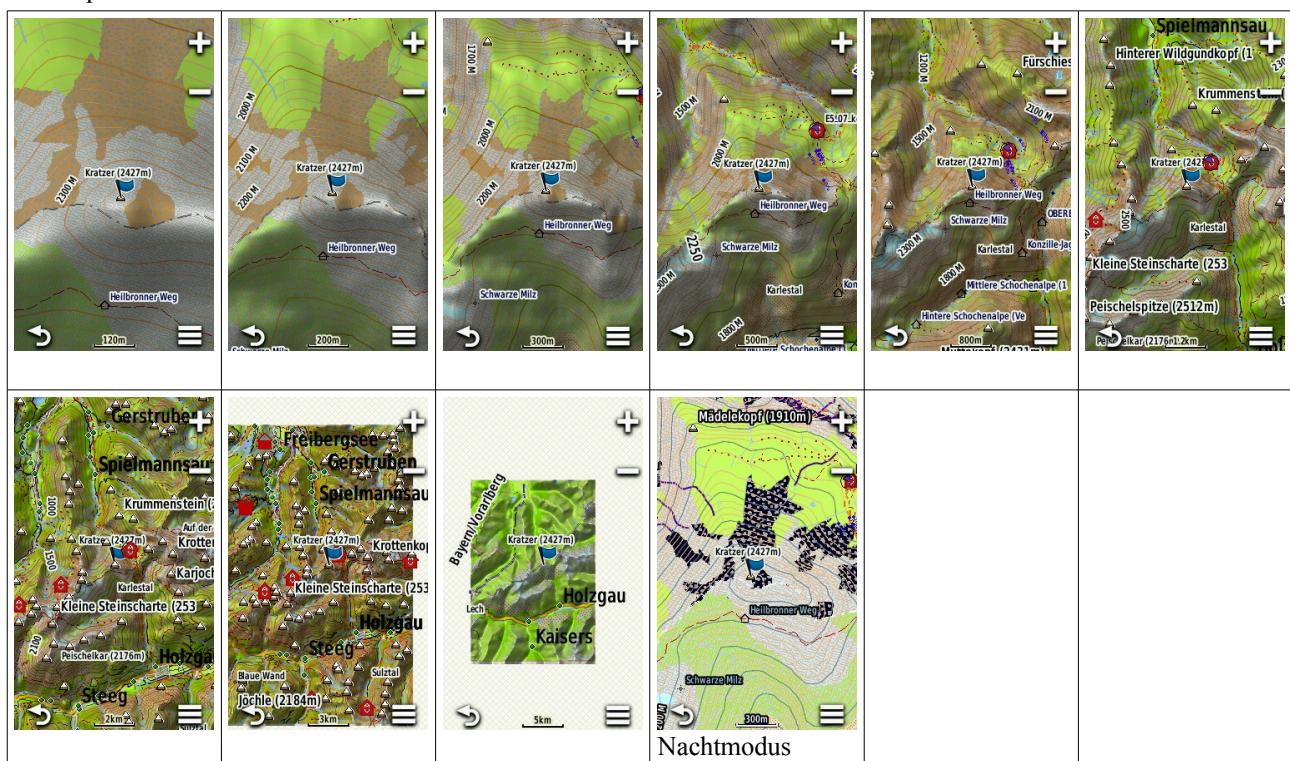
Darstellung mit Oregon 600

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 $\leq 800\text{m}$ sichtbar:



Darstellung „echter“ Garmin-Karten

Transalpin



Stand 24.10.2017

Topo Light

