Weiterentwicklung des Programmsystems zur digitalen Felsdarstellung

Tobias Dahinden

23.März 2000

Zusammenfassung

Dieser Bericht gibt einen Überblick über die Arbeit zum Vertiefungsblock 99/00 in Kartographie zum Thema der Programmentwicklung zur digitalen Felsdarstellung. Ziel der Arbeit war es einen Algorithmus zur Erzeugung von Füllschraffen und ev. morphologischen Strukturlinien, wie Spalten und Kanten, im Stil der Landeskarte 1:25'000 herzustellen.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Felsdarstellung gehört zu den grossen Künsten der Kartographen. Zum einen hat man es mit fast senkrechten Objekten zu tun, also nicht sehr viel Platz. Zum andern möchte man Felsen so wiedergeben, dass sie möglichst viel Informationen enthalten. Die wichtigsten Felsstrukturen sollten ersichtlich sein, damit sich Wanderer und Kletterer orientieren können.

Die Art, wie die Felsen dargestellt werden sollten, gab immer wieder Anlass zu Streitigkeiten [1][6][7][2]. In einem Streit Anfang der 70er Jahre [7][2] ging es unter anderem darum, mit welcher Äquidistanz die Höhenlinien im Felsgebiet dargestellt werden sollten. Die damaligen Auseinandersetzungen dürfen aber nicht darüber hinwegtäuschen, dass die Kosten für die Neuzeichnung der Felsen in einem Landeskartenblatt in Gebirgsregionen heute mehrere hundertausend Franken betragen würden [4]. Von diesem finanziellen Blickpunkt aus ist der Versuch, eine billigere Methode zur Felsdarstellung zu entwickeln, sicher gerechtfertigt. Verschiedene Varianten der Felsdarstellung für interaktive Graphiksysteme wurden deshalb bereits in [5] aufgezeigt.

Ein erster Schritt zur Digitalisierung und Automatisierung der Felszeichnung unternahm Lorenz Hurni im Rahmen seiner Dissertation [4]. Er programmierte eine Funktion, die es ermöglichte, auf einfache Weise Gerippelinien von Felsen zu zeichnen. Ein Beispiel davon findet sich u.a. in [3].

Versuche zur digitalen Darstellung von Felsschraffen unternahm auch die Universität der Bundeswehr München. Die Bemühungen bestanden aber darin, die Felsschraffen in einem Rasterprogramm von Hand zu zeichnen. Das ermöglicht zwar, Schraffen mit dem Computer darzustellen, ist aber eben so aufwendig, wie wenn die Schraffen von Hand gezeichnet werden.

In der Landeskarte 1:25'000 werden die Felsgebiete mit Schraffen gefüllt. An diese wird der Anspruch gestellt, dass es kleine unregelmässige, verzitterte Linien sind. Im Gegensatz dazu sind Computer im wesentlichen darauf ausgelegt, möglichst monotone regelmässige Prozesse auszuführen.

Kapitel 2

Ausgangslage

2.1 Material der Landestopographie

Eine Grundlage zur Herstellung des Felsschraffenprogrammes waren die Lehrlingsunterlagen der Landestopographie. Die Angaben, wie der Fels gezeichnet werden soll, sind leider ziemlich dürftig. Es bestehen zwar viele Angaben darüber, dass das Felsbild plastisch wirken sollte, oder dass Striche dicker oder dünner gezeichnet werden können. Eine kochrezeptartige Angabe, wie der Felsen gezeichnet werden soll, fehlt allerdings (leider).

2.2 Das Programm von Lorenz Hurni

Im Rahmen seiner Dissertation programmierte Lorenz Hurni einen Algorithmus zur Gerippeliniendarstellung. Mit dem Programm können auf einfache Weise die Gerippelinien von Felsbändern (Abb. 2.1)gezeichnet werden. Aus der Arbeit von Hurni wird ersichtlich, welche Faktoren (Felsneigung, Ausrichtung, Kavität) bei der Felsdarstellung eine Rolle spielen. Es wird auch aus-

Abbildung 2.1: Vergrösserte Gerippelinien nach Lorenz Hurni.

Abbildung 2.2: Gerippelinien in FreeHand.

führlich darauf eingegangen, wie die einzelnen Linien zugespitzt (verjüngt) sein sollten.

Das Programm wurde ursprünglich für ein Rasterprogramm hergestellt. Das Verzittern der Linien war dementsprechend auf dieses Programm abgestimmt. Inzwischen wurde das Programm auch ins FreeHand (vektorbasiert) übernommen. Es zeigt sich aber, dass dadurch die Gerippelinien keine starke Verzitterung mehr aufweisen (Abb. 2.2), falls der Massstab zu gross gewählt wurde.

2.3 Landeskarten

Die Felsdarstellungen in der Landeskarte bildeten eine weitere Grundlage, für das Felsschraffenprogramm. Im Wesentlichen versuchte ich, die Felsen in möglichst grosse Gebiete mit dem selben Charakter aufzuteilen. Daraus resultierte eine Dreiteilung der Felsengebiete in a) Strukturlinien b) kleinen Schraffenflächen c) Felsbändern.

Höhenlinien im Felsgebiet können mit den entsprechenden Aussparungen über die Felsschraffen hinüber gezeichnet werden. Darum wird an dieser Stelle nicht weiter darauf eingegangen.

Kapitel 3

Anforderungen an das Programm

3.1 Darstellung der Schraffen

Die mit dem Computer hergestellten Felsschraffen sollten möglichst ähnlich wie jene der Landeskarte 1:25'000 sein. Dazu werden zu den einzelnen Felspartien Informationen über Richtung des Lichteinfalles (Abb. 3.1), über die Neigung des Felsens (steil, flach; Abb. 3.2) und über die Kavität (Abb. 3.3) benötigt. Wegen der Vielzahl der benötigten Informationen müssen diese über ein Dialogfenster eingegeben werden. Die Randpunkte der einzelnen Felspartien werden dann mit der Maus gezeichnet. Die Richtung der einzelnen Schraffenlinien ist dann je nach Neigung entweder parallel oder orthogonal zu der ersten Linie, die vom Benutzer gezeichnet wird.

Es ist auch denkbar, dass die Felsgebiete vorgängig vektorisiert werden. Die Eigenschaften der einzelnen Felsteilstücke müssen zusammen mit den Koordinaten abgespeichert sein.

Damit eine allfällige spätere Änderung der Datenübergabe möglich ist, müssen alle Parameter möglichst am selben Ort (zu Beginn des Programms) definiert sein oder von aussen übergeben werden. Das Programm sollte darum auch "Open-soucre" [1] sein.

3.2 Flexibilität

Das Programm kann durchaus als Schritt zu weiteren Algorithmen zur Felsdarstellung angesehen werden. Darum sollte das Programm ohne grossen Auf-

[1] im Sinne der GNU General Public License, die von der Free Software Foundation, Inc., 675 Mass Ave., Cambridge, MA 02139, USA, publiziert werden,

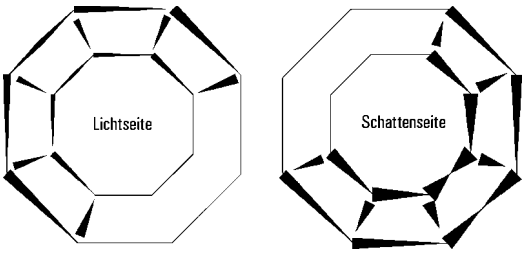

Abbildung 3.1: Lichteinfall bei Felsen.

Abbildung 3.2: Neigung des Felsens.

Abbildung 3.3: Leicht gekrümmte Felsschraffen.

wand abgeändert werden können. Es sollte auch (im Hinblick auf Vektor25) auf verschiedene Situationen flexibel reagieren können. Damit ist gemeint, dass es nicht stur an eine einzige Möglichkeit der Eingabe gebunden ist.

Das Programm sollte so in Klassen unterteilt werden, dass sich ein anderer Entwickler möglichst nur mit einer (kleinen) Klasse befassen muss.

3.3 Bezug zu Vektor25

Es war nie das Ziel dieser Arbeit, die Probleme der Felsdarstellung in Vektor25 zu lösen. Es ist jedoch folgende Situation denkbar: Die Felsen der Landeskarte müssten in kleine Einzelflächen unterteilt werden. Diese werden getrennt nach Ausrichtung und Neigung in verschiedenen Layern gespeichert. Danach wird das Programm auf die einzelnen Layer angewandt. Die Schraffen werden so aus Vektordaten wiedergewonnen. Es kann eine Felsdarstellung entstehen, die jener der Landeskarte ähnlich sieht.

Im Moment (26. September 2000) zeichnet das Programm nur Füllschraffen und einzelne Schraffen. Strukturlinien müssten "von Hand" gezeichnet werden. Es sollte meines Erachtens aber möglich sein, aus den Eingabeinformationen und den Schraffen noch Strukturlinien herzuleiten und zu zeichnen.

Kapitel 4

Entwicklung, Vorgehen

Die Aufgabenstellung war sehr offen definiert. In einer ersten Phase versuchte ich darum herauszufinden, was überhaupt gemacht werden kann. Da für den Vertiefungsblock normalerweise nur etwa 200 Stunden zu Verfügung stehen, musste ich relativ schnell und ohne gutes Konzept mit programmieren beginnen. Dabei legte ich in der ersten Phase das Schwergewicht auf die Bestimmung von Schraffenanfang und Schraffenende in einer Schraffenfläche.

Als unglücklichen Umstand muss ich im Nachhin die Tatsache bewerten, dass mir der Compiler erst ab Dezember zur Verfügung stand[1]. Mein Programm konnte ich bis Weihnachten soweit bearbeiten, dass der Compiler keine Fehler mehr fand. Riesig war hingegen die Anzahl der "Bugs"[2]. Die Zeit ab Januar benötigte ich darum nur zur Suche nach Fehlern. Diesen Zeitaufwand hatte ich gewaltig unterschätzt. Verstärkt wurde dieses Problem indem ich lange Zeit nicht wusste, mit welchen Input/Output-Mechanismen FreeHand arbeitet. Weiter wusste ich lange auch nicht, dass es in FreeHand keine globalen Variablen und "Enumerations" geben kann. Dadurch wurde die Arbeit noch weiter in die Länge gezogen.

Parallel zu den ersten Programmierarbeiten studierte ich die Felsschraffuren in den Landeskarten und das Programm von Lorenz Hurni. Leider konnte ich das Zeichnen der einzelnen Schraffen nicht wie erhofft von Lorenz Hurni übernehmen. Der Aufwand um den Zeichnungalgorithmus zu übernehmen wäre zum Einen sehr gross gewesen. Zum Zweiten war es nur schwierig die Verzitterung der Schraffen dem Massstab anzupassen. Darum entschloss ich mich das Verzittern der Linien neu zu programmieren.

[1] Das Programmieren mit Papier und Bleistift war nichts ungewöhnliches für mich. Es war aber das erste grössere Programmierprojekt, das ich ausführte.

[2] Bei Syntaxfehler handelt es sich um reine Schreibfehler, z.B. wenn "Murst" statt "Wurst" geschrieben wird. Bei Bugs handelt um Unhaltliche Fehler. Das Programm macht zwar etwas, was aber nicht sinnvoll ist, z.B. "Erhitze Milch auf 3000 Grad".

Das Programm besteht aus vielen einzelnen Funktionen. In typisch mathematischer Manier versuchte ich immer meine Probleme auf schon gelöste Probleme zurückzuführen. Das bewirkt, dass die Funktionen in nur wenigen Klassen zusammengefasst werden. Diese Klassen (ausser der Hauptklasse "cTopSchraffen") können nur mühsam weiter aufgeteilt werden.

Da ich "einfach drauflos" programmierte, habe ich einige Funktionen mehrmals in ähnlicher Form geschrieben. Ein Beispiel dafür ist das Löschen der verschiedenen Pointer. Es hätte sich geradezu aufgedrängt, dafür Schablonen zu programmieren. Dieser Umstand wurde mir erst nach der Programmierarbeit ersichtlich.

Kapitel 5

Programmbeschrieb

5.1 Arbeitsweise des Programmes

Das Programm ist als Tool in FreeHand realisiert. Der Benutzer zeichnet als erstes eine Figur mit dem Zeichenstift, die mit Schraffen gefüllt werden soll. Dabei kann es sich um eine einzelne Linie handeln, die als Schraffe (genannt Einzelschraffe) dargestellt werden soll oder eine Fläche (Polygonzug; genannt Schraffenfläche) oder ein Felsband (zwei Polygone, gleichviele Unter- und Oberkanten, wie im Tool Felsdarstellungen; genannt Schraffenband), die mit Schraffen gefüllt werden sollen. Danach wird in den Xtras die Funktion Schraffendarstellung aufgerufen. Als erstes erscheint dann ein Dialogfenster, in dem die notwendigen Parameter eingestellt werden. Der Dialog wurde teilweise von Ernst Hutzler erstellt. Er muss aber noch aktualisiert werden.

Mit "OK" im Dialog wird der Hauptteil des Programmes aufgerufen. Mit "Reset" besteht die Möglichkeit, die Eingabeparameter auf vorgegebene Werte zu setzen. Mit "Help" sollten helfende Angaben zum Programm aufgerufen werden können. Diese Informationen konnte ich aus Zeitgründen aber noch nicht erstellen.

Der Dialog lässt keine ungültigen Eingaben zu. Es ist jedoch möglich unvernüftige Felsflächen zu zeichnen.

Als erstes untersucht das Programm ob es sich um eine Einzelschraffe, eine Schraffenfläche oder ein Schraffenband handelt. Ein Schraffenband wird dann in einzelne Schraffenflächen aufgeteilt. Bei den Schraffenflächen wird dann der Ort der einzelnen Schraffen bestimmt (Anfangs- und Endpunkt). Danach werden alle Einzelschraffen in viele kleine Linien unterteilt und jeder Linie eine Strichdicke und eine Auslenkung zur Verzitterung zugeordnet. Die Anfangs- und Endpunkte werden zufällig nach links oder rechts ausgelenkt. Diese kleinen Linien werden dann an dem Zeichnungsprogramm (FreeHand)

Abbildung 5.1: Mögliches Endresultat.

zurückgegeben und auf dem Bildschirm gezeichnet. Die Schraffen sehen dann etwa wie in Abb. 5.1 aus.

Die Unterteilung der Schraffenflächen in Einzelschraffen bildete dabei die grösste Herausforderung. Schliesslich besitzen die Einzelschraffen einen fast regelmässigen Abstand, dürfen sich nicht kreuzen, müssen alle in die selbe Richtung zeigen und dürfen sich, auch bei beliebigen Geometrien, nicht ausserhalb der Schraffenfläche befinden.

Da ich längere Zeit nicht wusste, wie die Datenübergabe von und nach FreeHand funktionieren soll, wurden alle Zwischenspeicherungen der Daten mit Pointern realisiert. Diese sind zwar syntaxfehleranfällig[1], erlauben aber eine offene und trotzdem speicherplatzgünstige Benutzung des Programmes. Es sollte so auch möglich sein, das Programm ohne grösseren Aufwand in eine andere Umgebung einzupassen.

5.2 Numerische Probleme

Alle Funktionen des Programmes wurden nach Möglichkeit mit linearen Prozessen versehen. Das Programm beinhaltet insbesondere keine Rekursionen. Diese numerische Optimierung dürfte sich jedoch nur bei grösseren Felsflächen bezahlt machen.

Sämtliche Winkelberechnungen laufen über eine Arcus-Tanges-Funktion von “math.h”. Dadurch kann einigen unangenehmen numerischen Problemen

[1]Die Pointer verursachten u.a. zwei Fehler, die ich mehr als 4 Tage lang suchen musste, bis ich sie gefunden hatte!

ausgewichen werden[2]. Diese Funktion sollte genügend ausgetestet sein. Der Output sollte sich zum Input linear verhalten.

5.3 Kurzer Quellcodebeschrieb

Das Programm wurde in C++ verfasst. Im Moment besteht es aus vier Klassen: cTopSchraffen, Referenzpunkte, Schraffenpunkte und Geometrie. Die Klasse cTopSchraffen könnte/sollte noch weiter unterteilt werden. Vorallem sollten die Funktionen, die spezielle kartographische Eigenschaften verwalten (Anzahl Strichdicken, Strichdicken zuordnen, Verzittern der Schraffen, etc.) als eigene Klasse abgesondert werden. Damit wird eine spätere Änderung vereinfacht.

Von den Klassen eignet sich vorallem die Klasse Geometrie zur Wiederverwendung in einem anderen Programm. Sie verwaltet eine Vielzahl von geometrischen Funktionen (Zwischenwinkel zwischen zwei Vektoren berechnen, Komponenten eines Vektor bestimmen, Orthogonalprojektionen, etc.). Dazu gehören auch noch einige Strukturen (Vektoren, Geraden, Linien) die Pointereigenschaften besitzen[3]. Die Anderen Klassen sind eher programm spezifisch und können darum auch weniger leicht verändert und übernommen werden.

5.4 Funktionstüchtigkeit

Das Programm liefert seit dem 17. März 2000 brauchbare Resultate, was die Berechnungen anbelangt. Viele Flächen werden problemlos mit Felsschraffen gefüllt. Es treten aber immer noch vereinzelte Fälle auf, in denen das Programm versagt und keine Schraffen gezeichnet werden. Weiter gibt es noch Fälle, in den Schraffen gezeichnet werden, aber die Ausrichtung noch nicht klappt.

Nur mangelhaft sind die Parameter Dicke, Vezitterung und Schraffenabstand eingestellt. Der Grund liegt daran, dass ich aus Zeitgründen bis jetzt fast keine Bemühungen unternommen habe, diese Parameter einzustellen. Im

[2]Die Probleme entstanden unter anderem dadurch, das FreeHand mit relativ grossen Koordinaten arbeitet. Einige Probleme machen sich noch bei Längenberechnungen bemerkbar, sollten aber keinen Einfluss mehr auf die Lauffähigkeit des Programmes haben. Weiter machte sich Auslöschung bei der Berechnung eines Winkels über das Standardskalarprodunkt bemerkbar. Mit der Arcus-Tangens-Funktion sollten das aber behoben sein.

[3]Ich bin sicher, das solche Funktionen schon vielfach programmiert wurden. Während der Entwicklung stand mir aber leider keine solche Klasse zur Verfügung.

Moment ist es auch noch so, dass jedesmal wenn eine dieser Einstellungen geändert wird, das Programm neu compiliert werden muss.

Um die Fähigkeiten des Programmes aufzuzeigen, werden zwei Testbeispiele gezeigt (Abb. 5.2 und Abb. 5.3). In Abb. 5.2 werden verschiedene mögliche Füllungen der selben Schraffenfläche gezeigt. Es handelt sich allerdings nur um eine Auswahl. In Abb. 5.3 versuchte ich das Felsbild eines Ausschnittes einer Landeskarte 1:25'000 (Blatt Amsteg) nachzuzeichnen. Während einer halben Stunde versuchte ich, die Felsschraffen anhand des bestehenden Kartenbildes zu vektorisieren (Abb. 5.4). Zum Vergleich sieht man das Original in Abb. 5.5.

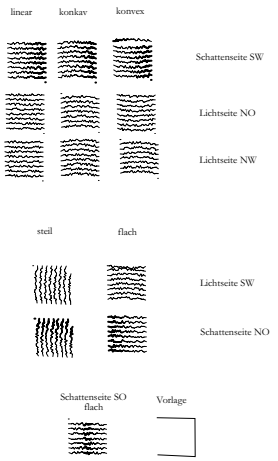

Abbildung 5.2: Testbeispiele einiger Schraffenflächen.

5.5 Beurteilung des Programmes

Nach ersten Versuchen mit dem Programm zeigt sich folgendes: Mit dem Programm können schnell und einfach Felsschraffen gezeichnet werden. Die einzelnen Schraffen sind schon heute ziemlich schön. Ihr Aussehen kann noch optimiert werden. Ob aber der Fels vernünftig in einzelne Flächen aufgeteilt

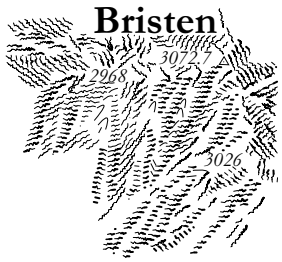

Abbildung 5.3: Neuzeichnung des Bristen.

wird und diese Flächen gut nach Licht und Schatten, Neigung und Kavität eingeteilt werden liegt aber nach wie vor im Geschick des Zeichners.

Dem Aspekt nach einem brauchbaren Kartenbild muss also in Zukunft wohl eher noch vermehrt Rechnung getragen werden. In diesem Zusammenhang erlaube ich mir folgendes zu erwähnen: Mehrere meiner Bekannten (nicht Kartographen), denen ich meine Felsschraffen im Vergleich zu bisherigen Schraffuren zeigte, äusserten den Wunsch, Felsgebiet nicht mit Informationen zu überfüllen. Das Felsgebiet solle übersichtlich wirken.

Abbildung 5.4: Bristen: alt und neu.

Abbildung 5.5: Ausschnitt des Bristen aus LK 1:25'000 (Blatt Amsteg).

Kapitel 6

Mögliches weiteres Vorgehen

In einer ersten Stufe muss das Programm so zum Laufen gebracht werden, dass keine Abstürze mehr vorkommen, die durch Geometrien der Schraffenflächen hervorgerufen werden. Zudem muss die Funktion der Schraffenausrichtung (Verjüngung) lauffähig werden.

In einem nächsten Arbeitsgang kann ein Dialog erstellt werden, in dem die Dicke und die Verzitterung der Schraffen variiert werden können. Im gleichen Schritt sollte auch das unnötige Resultatfenster, das im Moment noch erscheint, entfernt werden. Das schon bestehende Dialogfenster muss um den Wert Neigung und einem Verweis zum neuen, zweiten Dialogfenster ergänzt werden.

Wenn das Dialogfenster für die Strichdicke und Verzitterung einmal besteht, sollten vernünftige Defaultwerte für die Schraffen festgelegt werden. Im Rahmen dieser Arbeit könnte auch Überlegungen zu allfälligen Änderungen im Felsbild gemacht werden. Weiter stellt sich die Frage, ob die Einteilung nach Lichteinfall noch verfeinert werden soll. Dabei ist auch ein Modell denkbar, indem die Strichstärken kontinuierlich nach Lichteinfall zu oder abnehmen.

Es scheint mir sinnvoll, wenn in meinem Programm ebenfalls die Möglichkeit der Ausmaskierung von Flächen besteht, wie sie im Programm "Felsdarstellung" vorhanden ist. Diese Funktion könnte noch eingebaut werden.

Weiter wird das Resultat meines Programmes noch nicht in einen von der Eingabe unabhängigen Layer gezeichnet. Damit wird das Arbeiten mit dem Programm recht mühsam. Die Möglichkeit des Layertrennen sollte unbedingt noch ausgeschöpft werden.[1]

[1]Diese Abänderungen des Programms hatte ich ursprünglich mit Ernst Hutzler bereits vereinbart. Einer Ausführung sollte spätestens nach der Genesung von Ernst Hutzler nichts im Wege stehen.

Eine weitere Aufgabe besteht darin, Fehlermeldungen des Programms zu ermöglichen. Im jetzigen Quellcode sind schon Ansätze dazu vorhanden[2].

Um die Schraffen ohne grossen Aufwand verändern zu können besteht noch die Möglichkeit einer Zoomfunktion. Diese muss noch ins Programm eingebunden werden.

Der Quellcode muss noch so aufgearbeitet werden, dass ohne grössere Schwierigkeiten weitere Funktionen von Nichtprogrammkennern eingebaut werden können. Gemeint sind damit u.a. Erweiterungen wie das Berechnen von Strukturlinien oder die automatische Berücksichtigung eines Höhenmodells beim Zeichnen. Die Einstellungen zu den Ausrichtungen, den Anzahl Strichdicken und den Zufallsfunktionen sollten auch noch leichter auffindbar sein.

[2]Eigentlich wollte ich bereits von Beginn weg Fehler mit der Funktion “cerr” von ISO/ANSI Standard C++ auffangen. Diese Funktion steht aber bei FreeHand und Visual-C++ nicht zur Verfügung. Im Programm sind diese Funktionen darum oft als Bemerkung vorhanden.

Literaturverzeichnis

[1] Bertschmann, S.: *Probleme der Gebirgsblätter der Landeskarte 1:25'000*, Schweiz. Bauzeitung, Zürich 1953, Seiten 3-5.

[2] Brandstätter, Leonhard: *Neue Gedanken zur topographischen Kartographie des Hochgebirges*, Kartographische Nachrichten 5/70, Gütersloh 1970, Seiten 167-178.

[3] Hurni, Lorenz; Naumann, Andreas: *Digitale Felsdarstellung für topographische Gebirgskarten*, Kartographische Nachrichten 1/99, Bonn 1999, Seiten 16-22.

[4] Hurni, Lorenz: *Modellhafte Arbeitsabläufe zur digitalen Erstellung von topographischen undgeologischen Karten und dreidimensionalen Visualisierungen*, Institut für Kartographie ETHZ, Zürich 1995, 190 Seiten.

[5] Hurni, Lorenz: *Verschiedene Felsdarstellungsarten für Gebirgskarten 1:25'000*, Institut für Kartographie ETHZ, Zürich 1989, 23 Seiten.

[6] Jegher, W.: *Neues von den Landeskarten der Schweiz*, Schweiz. Bauzeitung, Zürich 1953, Seiten 5-7.

[7] Spiess, Ernst: *Zur Felsdarstellung in unseren Gebirgsblättern*, Der UTO 48 (1970) 10, Zürich 1970, Seiten 138-144.

Anhang A

Detaillierterer Programmbeschrieb, Stand: 26. September 2000

Zuerst muss in FreeHand der Rand der zu schraffierenden Fläche mit dem Zeichenstift gezeichnet werden. Wie oben bereits erwähnt, wird dann das Programm mit der Xtra-Funktion "Schraffendarstellung" gestartet.

Es erscheint ein Dialog, in dem die Felsparameter Neigung (im Moment via "Maske"), Ausrichtung und Kavität eingegeben werden sollten. Dabei werden die letzten benutzen Parameter angezeigt. Es besteht die Möglichkeit die Ausrichtung automatisch errechnen zu lassen. Es sei aber darauf hingewiesen, dass es sich nur um eine Schätzung der Ausrichtung anhand der eingegebenen Eckpunkte handelt.

Nach der Parameterabfrage (Neigung, Ausrichtung) wird zuerst mit der Funktion $check-was$ untersucht, ob eine Einzelschraffen, eine Schraffenfläche oder ein Schraffenband gezeichnet werden muss.

Falls es sich um eine Schraffenfläche oder Bandschraffen handelt, werden zuerst die Orte bestimmt, wo die einzelnen Schraffen gezeichnet werden. Die Bandschraffen werden hier in einzele Schraffenflächen zerlegt. Dazu wird anhand einer Grundlinie (erste gezeichnete Linie) eine Referenzlinie bestimmt, die je nach Neigung parallel oder orthogonal zur Grundlinie liegt ($bestimme - Referenzlinie$). Die Länge dieser Linie ist länger, als die längeste Diagonale der Schraffenfläche ($finde - Diagonalenlaenge$). Zusätzlich wird zur Referenzlinie eine Art "Suchbaum" erzeugt, indem die Schraffen (Anfang und Ende) geordnet gespeichert werden können. Das ist die Aufgabe der Klassen Referenzpunkte und Schraffenpunkte. Der Ort, wo sich ein Schraffenanfangs- oder -endpunkt befindet, wird so bestimmt, indem der Computer der Reihe nach die einzelnen Seiten der Fläche auf die Referenzlinie

projiziert und in 3tel Millimeterschritten[1] einen Punkt festlegt. Der Punkt wird dann am Suchbaum befestigt. Ist der Punkt ein Eckpunkt der Fläche, wird er zweimal befestigt. Danach sind am Suchbaum eine gerade Anzahl Punkte und es folgt immer ein Anfangspunkt auf einen Endpunkt einer Schraffe. Jetzt muss lediglich wegen der Verjüngung der Schraffen noch untersucht werden, ob die Schraffen alle in die selbe Richtung schauen ($ordne$).

Nun müssen all die Einzelschraffen noch verzittert werden ($unterteile - Schraffen$). Das Programm beginnt beim Anfangspunkt der Schraffe, und geht einen (fast) zufällig langen Schritt entlang der Mittellinie oder eines Kreises mit gegebenem Radius. Dann wird die Linie orthogonal zur Linie bzw. zum Kreis ausgelenkt (wieder zufällig). Dieser Punkt wird nun als Endpunkt einer neuen, kurzen Linie gespeichert und ist zugleich Anfangspunkt der nächsten neuen kurzen Linie. Das Programm bestimmt nun den nächsten Enpunkt, indem es vom nicht ausgelennkten Ort den ersten Schritt wiederholt, bis die ganze Linie unterteilt ist.

Als letzten Schritt werden den kleinen Linien noch unterschiedliche Strichdicken zugeordnet ($zuordne - Dicke$). Im Moment handelt es sich um eine Unterteilung in fünf Strichstärken. Der Grund liegt darin, dass Lorenz Hurni in seinem Felsgerippeprogramm gute Erfahrungen mit dieser Zahl gemacht hat [4]. Diese kurzen Linien werden nun im FreeHand auf den Bildschirm gezeichnet.

Es ist möglich mit der “Undo”-Funktion von FreeHand die gezeichneten Schraffen sofort wieder zu löschen und neue zu zeichen.

[1] Default-Wert, veränderbar

Anhang B

Quellcode der Headerdatei

Um einen Eindruck über die Programmierarbeit zu vermitteln wird hier die Headerdatei (Version: 26. September 2000) wiedergegeben.

```
/*
 *      FILE:           Sektor_Dcrl.h
 */


/*
 *      sentry
 */
#ifndef _H_Sektor_Dcrl
#define _H_Sektor_Dcrl


/*
 *      include files
 */
#include "moaxtra.h"
#include "moastdif.h"
#include "Schraffen_xtra.h"
#include "Schraffen_butn.h"

/*
 *      special (required for 68k)
 */
#if defined(__MWERKS__) && !defined(powerc)
```

```
#pragma push
#pragma pointers_in_D0
#endif



/*
 *      special (required for 68k)
 */
#if defined(__MWERKS__) && !defined(powerc)
#pragma pop
#endif


/*
 *      sentry
 */
#endif



/*                                                                  */
/*                                                                  */
/*              Headerfile "schraffen.h"                            */
/*                                                                  */
/*              gehört zu schraffen.cpp                             */
/*                                                                  */
/*                                                                  */

typedef MoaLong Koordinate;
typedef MoaDouble Double;

/******************************************************************/
/*                                                                */
/*              Type definitons                                   */
/*                                                                */
/******************************************************************/

struct Line
{
```

```
        int Ax;
        int Ay;
        int Ex;
        int Ey;
        float Dicke;
        Line* next;
};

/*                                                              */
/*              Struktur "Punkt"                                */
/*                                                              */
/*              Aufgabe:                                        */
/*              Punkt als Vektor darstellen                     */
/*              Heapfähig                                       */
/*                                                              */


struct Punkt
{
        Punkt();
        Koordinate a;
        Koordinate b;
        Punkt* next;
};


/*                                                              */
/*              Struktur "Gerade"                               */
/*                                                              */
/*              Aufgabe:                                        */
/*              Linie mit Anfangs- und Endpunkt darstellen      */
/*              heapfähig                                       */
/*                                                              */


struct Gerade
{
        Gerade();
        Punkt c;
        Punkt d;
        Gerade* next;
```

```
};


/*                                                              */
/*              Struktur "Linie"                                */
/*                                                              */
/*              Aufgabe:                                        */
/*              enthält Attribute eines Linienelementes         */
/*                                                              */


struct Linie:Gerade
{
        Double Dicke;
        int Farbe;
        Linie* plus;
};


/*                                                              */
/*              Struktur "Vektor"                               */
/*                                                              */
/*              Aufgabe:                                        */
/*              Stellt einen Vektor vom Origin aus dar          */
/*                                                              */


struct Vektor
{
        Koordinate a;
        Koordinate b;
};


/*                                                              */
/*              Struktur "Element"                              */
/*                                                              */
/*              Aufgabe:                                        */
/*              wird als Node bei "Referenzpunkte" benötigt     */
/*                                                              */
```

```
struct Element
{
        Element();
        Punkt* Schraffen;
        signed int Nummer;
        Element* next;
};

/*                                                                  */
/*              Klasse "Geometrie"                                  */
/*                                                                  */
/*              Aufgabe:                                            */
/*              Die wichtigsten geometrischen Funktionen            */
/*                                                                  */

class Geometrie
{
        public:
        Double laenge(Gerade);
        Vektor Komponenten(Gerade);
        MoaLong laenge(Punkt, Punkt);
        Double Zwischenwinkel(Gerade, Gerade);
        Punkt Mitte(Gerade);
        Punkt berechne_Schnittpunkt(Gerade, Gerade);
        Gerade projeziere_auf_Linie(Gerade, Gerade);
        Punkt projeziere_Punkt(Punkt, Gerade, Gerade);
        Double sin_Zwischenwinkel(Gerade, Gerade);
        Double cos_Zwischenwinkel(Gerade, Gerade);
        Double laenge(Vektor x);
        float bestimme_Azimut(Punkt Anfangspunkt, Punkt Endpunkt);
        float bestimme_Azimut(Gerade g);
 };


/*                                                                  */
/*              Klasse "cTopSchraffen"                              */
/*                                                                  */
/*              Aufgabe:                                            */
/*              Die Klasse beihaltet alle Funktionen und            */
/*              Variablen, die                                      */
```

```
/*              normalerweise global sind                          */
/*                                                                 */


class cTopSchraffen:public Geometrie, Element
{
        public:
        cTopSchraffen(CSchraffen_Button * pObj, MoaDouble, MoaDouble,
MoaDouble, MoaDouble, unsigned int, MoaLong[], MoaLong[],
unsigned int, MoaLong[], MoaLong[], MoaDouble, MoaLong,
MoaLong, MoaLong, MoaLong[], MoaLong, Line*);
        cTopSchraffen();
        MoaDouble check_was(unsigned int, unsigned int);
        void einzelschraffe(MoaLong[], MoaLong[]);
        void bandschraffen(unsigned int, MoaLong[], MoaLong[],
MoaLong[], MoaLong[]);
        void schraffenflaechen(unsigned int,MoaLong[], MoaLong[]);
        void schraffiere();

        void berechne_Referenzlinie(Double, int);
        Double finde_Diagonalenlaenge();
        void ordne(int);
        void unterteile_Schraffe();
        void zuordne_Dicke();
        void anhaenge(Punkt, Punkt);
        int loesche();
        unsigned int zaehle_Geraden();
        float bestimme_Dicke(int, int, int);
        Double zufallslaenge();
        Double zufallsbreite();
        Double Korrektur(Double, Double, int);

        Punkt berechne_Mittelpunkt();
        void schraffe();
        int bestimme_ausrichtung(int, int);
        void anfuegen();
        int bestimme_Richtung( Punkt, Gerade, int);
        void aufraeumen(Line*);
        unsigned int zaehle_Linien();
        void Fehler(int);
```

```
         MoaLong Schraffenabstand;
         int n;
         MoaDouble Wert;
         MoaLong einteilungen;
         MoaLong bereichsbreite;
         Koordinate abschnittslaenge;
         MoaLong Breite[10];
         int kavitaet;
         int ausrichtung;
         MoaLong Felsneigung;
         int autoAuswahl;
         MoaLong zoomFaktor;
         MoaLong was;

        Punkt* pPunktBeginn;
        Gerade* pGeradeAnfang;
        Gerade* pGeradeAnfang_zwei;
        Gerade* pGeradeReferenzlinie;
        Linie* pLinieVorne;
        Linie* pLinieStop;
        Linie* pLinieGo;
};


/*                                                              */
/*              Klasse "Referenzpunkte"                         */
/*                                                              */
/*              Aufgabe:                                        */
/*              Die Struktur soll die Möglichkeit schaffen      */
/*              die Schraffenanfangs- und -endpunkte geordnet   */
/*              zu speichern                                    */
/*                                                              */


class Referenzpunkte:public Geometrie
{
        public:
        Referenzpunkte(Gerade, MoaLong);
        ~Referenzpunkte();
        Element* Anfang;
        Gerade Referenz;
```

```
        protected:
        Element* Ende;
        int loeschen();
        void anhaengen(const int);
        void generiere();

        MoaLong Schraffenabstand;
//?? MoaLong zoomFaktor;
};


/*                                                                  */
/*              Klasse "Schraffenpunkte"                            */
/*                                                                  */
/*              Aufgabe:                                            */
/*              Den Ort von sich feststellen, sich speichern        */
/*                                                                  */


class Schraffenpunkte:public Referenzpunkte
{
        public:
        Schraffenpunkte(Gerade, Gerade*, Punkt*, MoaLong);
        ~Schraffenpunkte();
        void zuordne_Schraffen();
        private:
        void berechne_Schraffenpunkte();
        int loesche_Schraffenpunkte();
        void Punktanhaengen(Punkt*, int);
        Element* Aktuell;
        Punkt* aktuell;

        Gerade* pGeradeAnfang;
        Punkt* pPunktBeginn;

};


/*****************************************************************/
/*                                                               */
/*              Include directives                               */
```

```
/*                                                                    */
/********************************************************************/


#include<math.h>
#include<stdlib.h>



/*
 *      external prototypes
 */
 static MoaError SchraffenBerechnung(
        CSchraffen_Button * pObj,
        MoaDouble schraffen_Auswahl, MoaDouble schraffen_Kavitaet,
MoaDouble schraffen_LS, MoaDouble schraffen_ZF,
MoaDouble schraffen_Maske,
        MoaLong n1, MoaLong x1[LINELENGTH],MoaLong y1[LINELENGTH],
        MoaLong n2, MoaLong x2[LINELENGTH],MoaLong y2[LINELENGTH],
        MoaDouble wert, MoaLong einteilungen, MoaLong bereichsbreite,
        MoaLong abschnittslaenge, MoaLong Breite[10],
MoaLong Schraffenabstand, Line *pLine
);
```