Aufbau einer Datenbank unter Matlab zur Verwaltung von Bildsegmenten

Kerstin Herms

TR-IGG-P-2007-05

15. August 2007

Technical Report Nr. 5, 2007

Department of Photogrammetry
Institute of Geodesy and Geoinformation
University of Bonn

Available at
http://www.ipb.uni-bonn.de/papers/

Aufbau einer Datenbank unter Matlab zur Verwaltung von Bildsegmenten

Kerstin Herms

Zusammenfassung

Ein großer Aufgabenbereich der Bildverarbeitung ist die Merkmalsextraktion. Hierbei ist es zunächst erforderlich, die Bilder durch eine Segmentierung in konsistente Landkarten zu überführen. Wir verwenden zur Segmentierung einen Wasserscheidenalgorithmus. Die Verwaltung der Landkarten sollte möglichst effizient erfolgen. Der vorliegende Report erläutert zunächst unterschiedliche Speicherstrukturen und geht auf einen möglichen Ansatz zur konsistenten Umwandlung von Rasterdaten in Vektordaten ein. In einem zweiten Tiel beschäftigen wir uns mit dem Aufbau einer Datenbank zur Verwaltung dieser Vektordaten von Matlab aus.

1 Einleitung

Den Ausgangspunkt dieser Arbeit stellen Bilder dar, die vollständig und überlappungsfrei in Segmente zerlegt wurden. Eine solche Zerlegung bezeichnen wir im Folgenden als Landkarte. Häufig liefert eine Segmentierung Rasterdaten, wie auch die hier vorliegende Segmentierung mit Hilfe eines Wasserscheiden-Algorithmus. Die Speicherung einer solchen Landkarte als Rasterdaten führt jedoch zu einer großen Datenmenge. Zudem werden Informationen über den Rand der Segmente sowie über ihre Beziehung zu den Nachbarsegmenten nicht explizit gespeichert. Um Merkmale zu verwenden, die sich darauf beziehen, ist es sinnvoll diese Informationen dauerhaft zu speichern.
Ziel ist es, eine Datenbank zur Verwaltung der Bildsegmente (hier der Wasserscheidenregionen) zu erstellen. Hierfür werden die Wasserscheiden vektorisiert. Gespeichert wird eine Winged-Edge-Struktur, bestehend aus Maschen, Kanten und Knoten, die eine eindeutige Beschreibung der Segmente erlaubt.
Der vorliegende Bericht erläutert zunächst den Schritt des Vektorisierens (Kapitel 2). Anschließend werden in Kapitel 3 die notwendigen Schritte zum Aufbau der Datenbank von Matlab aus beschrieben, bevor wir uns in Kapitel

4 der speziellen Struktur der Datenbank der Wasserscheidenbilder zuwenden.

2 Vektorisieren der Bilddaten

Gegeben sei die Segmentierung eines Bildes durch Wasserscheiden in Form einer Landkarte. Zur effizienten Speicherung von Landkarten empfiehlt sich eine Vektordarstellung. Wir wollen daher die Darstellung der Wasserscheiden als Rasterdaten in eine Vektordarstellung überführen.
Zunächst werden in diesem Kapitel mögliche Speicherstrukturen vorgestellt. Anschließend beschreiben wir den Schritt des Vektorisierens.

2.1 Vektordatenstrukturen

Zur Repräsentation von Landkarten durch Vektordaten gibt es verschiedene Strukturen [Wor95]. Deren Vor- und Nachteile sollen hier erläutert werden.
Für alle Strukturen gelten die im UML-Diagramm (Abbildung 1) dargestellten Beziehungen. Eine Masche besteht aus mindestens drei Kanten, jede Kante wird von genau zwei Knoten (einem Anfangs- und einem Endknoten) begrenzt und ist zu genau zwei Maschen adjazent, jeder Knoten besitzt einen eindeutigen Satz Koordinaten.

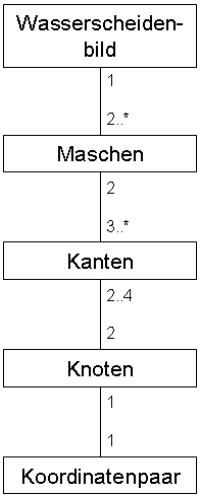


Abbildung 1: UML-Diagramm der Bezeihungen zwischen Knoten, Kanten und Maschen. Eine Masche besteht aus mindestens drei Kanten, jede Kante wird von genau zwei Knoten begrenzt und ist zu genau zwei Maschen adjazent, jeder Knoten besitzt einen eindeutigen Satz Koordinaten.

Zu den Vektordatenstrukturen gehören:

- Spaghetti-Struktur:
  Bei der Spaghetti-Struktur werden zu jeder Masche die Koordinaten der zugehörigen Knoten gespeichert. Man kann somit leicht den Umring einer Masche verfolgen. Leider entstehen bei diese Art der Speicherung Redundanzen, da gleiche Knoten an verschiedenen Stellen gespeichert werden. Zudem erhält man keine expliziten Angaben zur Topologie.

- Knoten-Kanten-Struktur:
  Die Knoten-Kanten-Struktur umgeht das Problem der Redundanz, indem Knoten mit einer eindeutigen Identitätsnummer (ID) versehen werden und ihre Koordinaten in einer eigenen Tabelle gespeichert werden. Zu jeder Kante werden dann Informationen über ihre begrenzenden Knoten sowie adjazenten Maschen gespeichert. Somit enthält diese Darstellung explizite Angaben zur Topologie.
  Dennoch hat diese Speicherstruktur einen Nachteil. Da die Kanten in beliebiger Reihenfolge gespeichert werden, kann man den Kantenumring einer Masche nicht direkt angeben. Hier schafft die Winged-Edge-Struktur Abhilfe.

- Winged-Edge-Struktur:
  Analog zur Knoten-Kanten-Struktur werden zu jeder Kante Informationen über ihre begrenzenden Knoten sowie adjazenten Maschen gespeichert. Zusätzlich werden Angaben zur jeweiligen Vorgänger- und Nachfolgerkante gespeichert, um den Maschenumring direkt angeben zu können. Diese Angaben sind genau dann eindeutig, wenn man sich jeweils auf einen fest vorgegebenen Umring beziehen. Wir verwenden für den Aufbau der Datenbank eine Winged-Edge-Struktur in der folgenden Form: die Vorgängerkante bezieht sich auf den Umring der linken Masche. Die Nachfolgerkante bezieht sich auf den Umring der rechten Masche.

2.2 Bestimmung der Knoten, Kanten und Maschen im Bild

Ausgehend von der Rasterdarstellung der Wasserscheiden, die ein Bild segmentieren, sollen Bildelemente der Winged-Edge-Struktur bestimmt werden. Wir verfolgen das Vorgehen an einem in Abbildung 2 dargestellten Wasserscheidenbild.

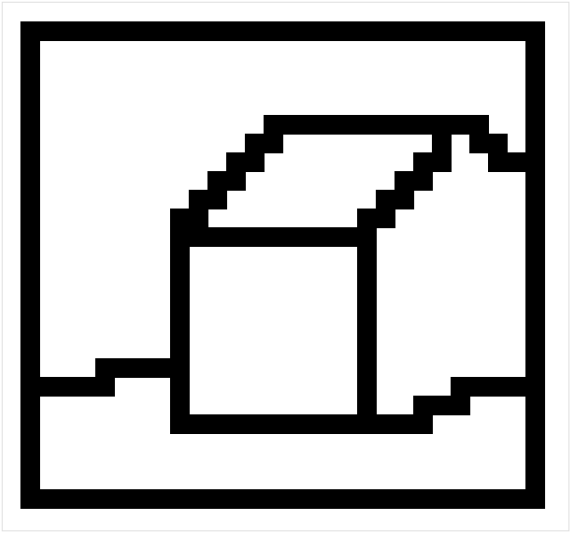


Abbildung 2: Durch die Segmentierung eines Bildes mit Hilfe des Wasserscheidenalgorithmus erhalten wir eine Landkarte im Rasterformat.

Von den Bildelementen Knoten, Kanten und Maschen lassen sich die Maschen am einfachsten bestimmen. Wir verwenden hierfür ein Labeling des Wasserscheidenbildes. Die verwendete Methode stammt aus der Image Processing Toolbox für Matlab (siehe Kapitel 3.1.1).

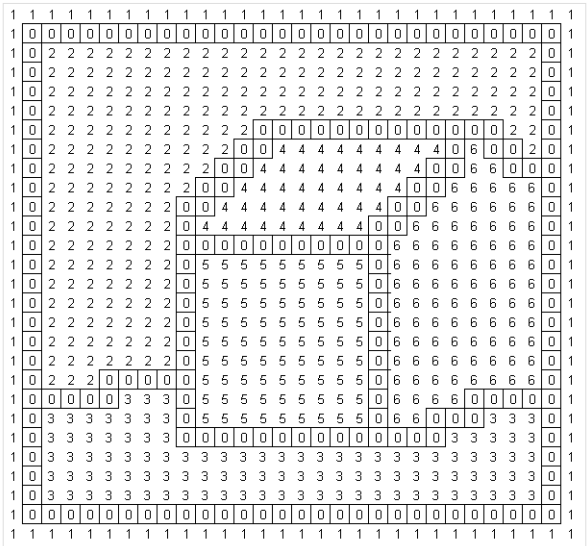


Abbildung 3: Das Labeling der Wasserscheidenregionen liefert eindeutig nummerierte Maschen.

Als Nächstes müssen die Ränder der Maschen bestimmt werden. Da sich die Kanten durch die Angabe von Anfangs- und Endknoten bestimmen lassen, wollen wir alle Knoten bestimmen. Einen Teil der Knotenmenge machen die Kreuzungspunkte aus. Das sind diejenigen Pixel, die in einer Vierernachbarschaft mindestens drei Nachbarn besitzen, die ebenfalls zur Wasserscheide gehören.

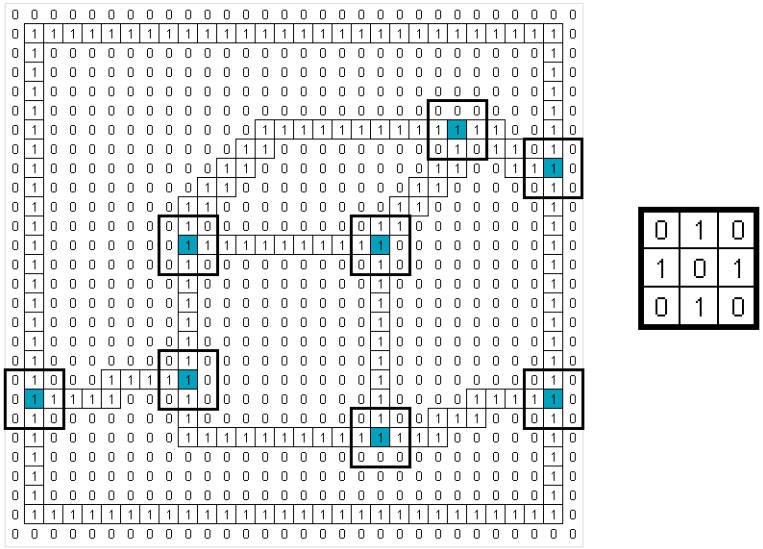
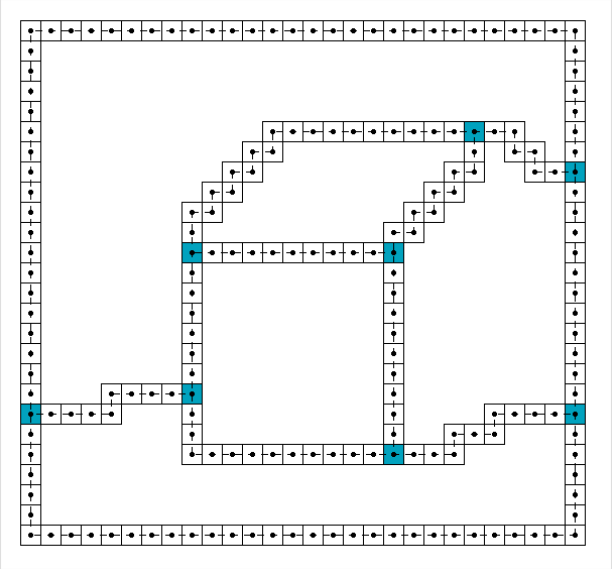
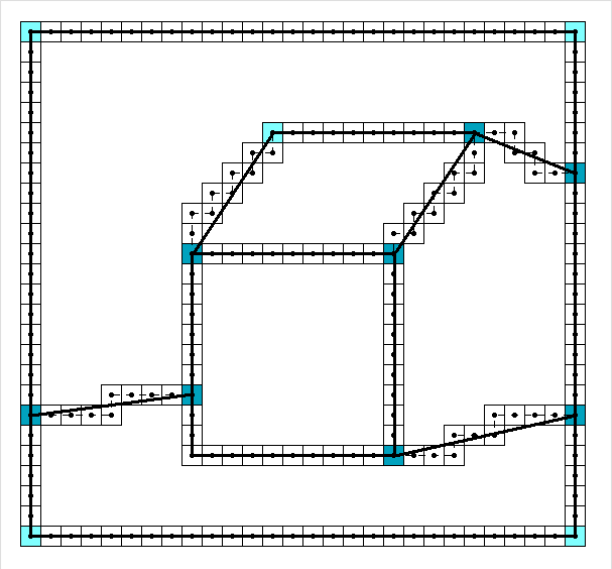


Abbildung 4: Mit Hilfe eines Filters (rechts) werden diejenigen Pixel detektiert, die auf einer Wasserscheide liegen und die in einer Vierernachbarschaft mindestens drei Nachbarn besitzen, die ebenfalls zur Wasserscheide gehören. Diese Pixel bezeichnen wir als Kreuzungspunkte. Sie sind hier farbig markiert.

Über den Befehl *bwtraceboundary* können wir Wasserscheiden zwischen zwei Kreuzungspunkten, die über eine Pixelkette verbunden sind, verfolgen. Aus dem Douglas-Peucker-Algorithmus (Kapitel 2.3) erhalten wir die Länge der einzelnen Kantenstücke (angegeben in Pixeln) und somit auch Positionen für neue Knotenpunkte. Durch die Angabe zweier benachbarter Knoten sowie der angrenzenden Maschen sind letztendlich auch die Kanten definiert.



(a) Eine Verbindung der Wasserscheidenpixel zwischen je zwei Kreuzungspunkten wird mit Hilfe der Matlab-Funktion *bwtraceboundary* gefunden.



(b) Die Anwendung des Douglas-Peucker-Algorithmus führt zu den hellblau dargestellten Bruchpunkten der Wasserscheiden. Diese bilden zusammen mit den Kreuzungspunkten (dunkelblau) die Knotenpunkte der Winged-Edge-Struktur. Durch je zwei benachbarte Knotenpunkte ist eine Kante festgelegt.

Abbildung 5: Dargestellt ist die Gewinnung der Knoten und Kanten der Winged-Edge-Struktur.

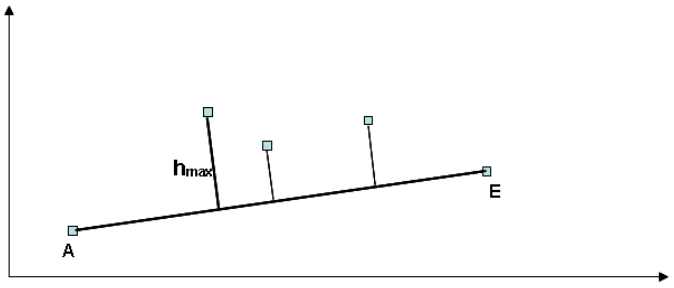
Wenn alle Knoten, Kanten und Maschen bekannt sind, können die zu einer Kante benachbarten Kanten leicht über gemeinsame Knoten und Maschen bestimmt werden.

2.3 Douglas-Peucker-Algorithmus

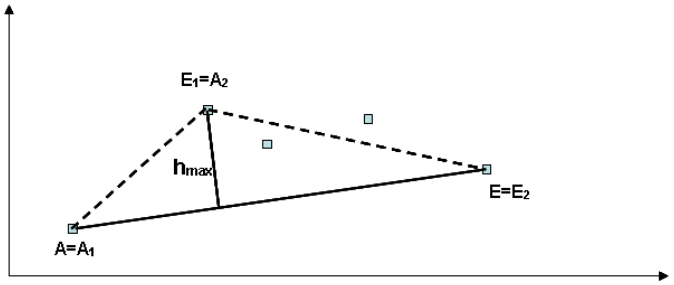
Beim Douglas-Peucker-Algorithmus [DP73] handelt es sich um ein iteratives Verfahren zur Linienglättung.
Allgemein arbeitet der Algorithmus mit einem Polygonzug, der Linien beliebiger Länge mit bekannten Endpunkten enthält. Die Vorgehensweise ist in Abbildung 6 dargestellt. Zunächst werden Anfangspunkt (A) und Endpunkt (E) des Polygonzuges zu einer Basislinie verbunden. Anschließend werden die Höhen der Polygonpunkte über der Basislinie bestimmt.
Der Punkt mit maximaler Höhe (h_{max}) wird zum festen Bestandteil des geglätteten Linienzuges und dient als Anfangs- bwz. Endpunkt der beiden neuen Basislinien $\overline{A_1E_1}$ und $\overline{A_2E_2}$. Für jede Basislinie wird das Verfahren iterativ angewandt, bis entweder kein Punkt mehr zwischen Anfangs- und Endpunkt einer Basislinie liegt oder bis die maximale Höhe eines Punktes über der Basislinie einen vorgegebener Schwellwert unterschreitet.

Speziell auf unsere Anwendung bezogen, können wir alle Pixel einer Pixelkette zwischen zwei Kreuzungspunkten als Endpunkte kleiner Liniensegmente betrachten. Diese werden als Ausgangsdaten in den Douglas-Peucker-Algorithmus gesteckt. Als Ergebnis erhalten wir diejenigen Pixel der Pixelkette, die einen markanten Knickpunkt der Wasserscheiden darstellen. Sie bilden somit neben den Kreuzungspunkten die Knoten der Winged-Edge-Struktur.

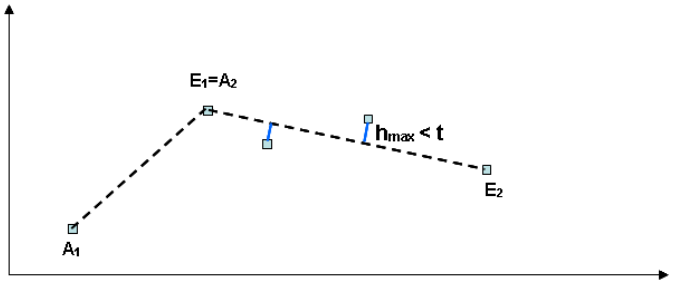
Zu beachten ist, dass der Douglas-Peucker-Algorithmus nicht auf geschlossene Umringe angewendet werden kann. Die Begründung liegt darin, dass die Basislinie dann auf die Länge 0 schrumpfen würde und wir nicht mehr durch ihren Betrag dividieren können.
Es treten jedoch häufig geschlosse Umringe in den Wasserscheidenbildern auf. Wir lösen das Problem, indem wir den Peucker-Algorithmus auf zwei Teilstücke dieser Umringe anwenden.



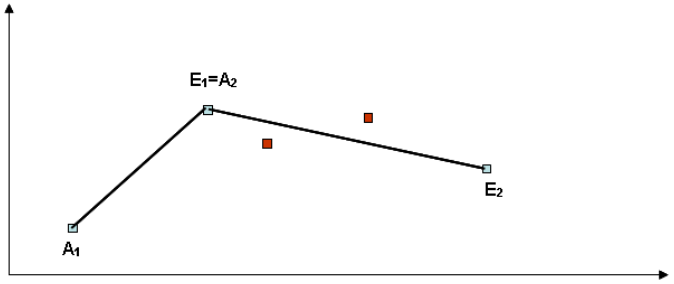
(a) Bestimmung der Punkthöhen über der Basislinie $\overline{AE}$.



(b) Festlegung der neuen Basislinien durch die Punkte A_1 und E_1 sowie durch A_2 und E_2.

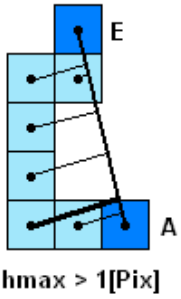


(c) Iteration bis die maximalen Höhen einen Schwellwert t unterschreiten.

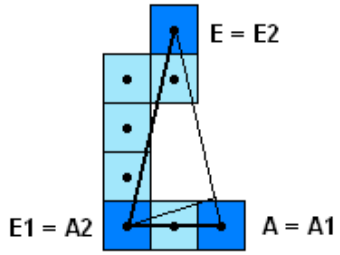


(d) Ergebnis des Peucker-Algorithmus. Rot gekennzeichnete Punkte entfallen nach der Linienglättung.

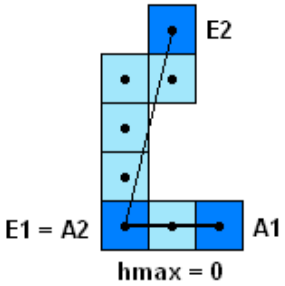
Abbildung 6: Der Douglas-Peucker-Algorithmus.



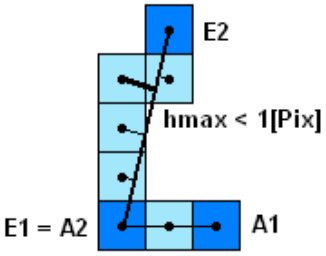
(a) Bestimmung der Höhen der Pixel über der Basislinie $\overline{AE}$. Die maximale Höhe überschreitet den Schellwert von 1 Pixel. Das Pixel mit maximaler Höhe wird somit zum Bruchpunkt des Linienzuges.



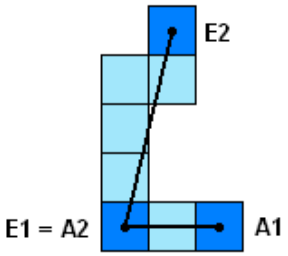
(b) Festlegung der neuen Basislinien $\overline{A_1E_1}$ und $\overline{A_2E_2}$ zwischen dem alten Anfangspunkt ($A = A_1$) beziehungsweise dem alten Endpunkt ($E = E_2$) und dem neu gefundenen Bruchpunkt aus Abbildung 7(a).



(c) Basislinie $\overline{A_1E_1}$: Die maximale Höhe von 0 liegt unterhalb des Schwellwertes von 1 Pixel. An dieser Stelle ist keine weitere Unterteilung vorzunehmen.



(d) Basislinie $\overline{A_2E_2}$: Die maximale Höhe liegt unterhalb des Schwellwertes von 1 Pixel. An dieser Stelle ist keine weitere Unterteilung vorzunehmen.



(e) Ergebnis des Peucker-Algorithmus. Dunkelblau gekennzeichnete Pixel stellen die Endpunkte des geglätteten Linienzuges dar. Die hellen Pixel entfallen nach der Linienglättung.

Abbildung 7: Der Douglas-Peucker-Algorithmus angewandt auf eine Pixelkette.

3 Aufbau einer Datenbank unter MATLAB

Das Ziel ist, vektorisierte Wasserscheiden in einer Datenbank zu speichern. Da für die Vektorisierung relevante Funktionen bereits als Matlab-Programm vorliegen, wollen wir von Matlab aus eine Datenbank aufbauen. Dieses Kapitel beschreibt zunächst, welche Softwarekomponenten verwendet werden und welche Vorbereitungen notwendig sind, um eine Datenbank zu erstellen.
Anschließend gehen wir näher darauf ein, wie man von Matlab aus auf eine bereits existierende Datenbank zugreifen kann. Hierfür bieten sich zwei Möglichkeiten an. Zum einem liefert Matlab bereits eine Schnittstelle zu Java. Diese können wir nutzten, um von Java aus auf SQL zugreifen zu können. Eine Alternative bietet die kostenpflichtige Matlab DatabaseToolbox, die hier ebenfalls vorgestellt werden soll. Beide Möglichkeiten werden auf ihre Eignung für unsere Aufgabenstellung überprüft.

3.1 Vorbereitung

Bevor wir eine Datenbank erstellen können, benötigen wir noch geeignete Software sowie die passenden Schnittstellen. In den drei folgenden Abschnitten wird der Weg des erstmaligen Erstellens einer Datenbank beschrieben.

3.1.1 Software

Installiert wurde Matlab 7.0.4 inklusive der Toolboxen Image Acquisition, Image Processing, LabelMe[1] und AnnotationTool[2]. Die ersten beiden Toolboxen können bei Matlab erworben werden.
Die zusätzliche Installation einer Demo-Version von Matlab 2007a (einschließlich aller verfügbaren Toolboxen) ermöglicht uns den direkten Zugriff auf SQL mit Hilfe der Database Toolbox. Indirekt greifen wir hier auch mittels Java auf SQL zu. Aus diesem Grund wurde zusätzlich noch Java in der Version j2sdk1.4.2_14[3] installiert. Es ist darauf zu achten, dass die gleiche Java-Version installiert wird, die auch Matlab verwendet. Liegt bereits eine andere Java-Version vor, so sind für den Fall der Verwendung von Java unter der Entwicklungsumgebung Eclipse spezielle Einstellungen vorzunehmen. Die genaue Vorgehensweise wird in Anhang A näher beschrieben.
Weiter wurden MySQL 5.0[4] sowie der passende JDBC-Treiber[5] installiert.

[1] http://labelme.csail.mit.edu/LabelMeToolbox/ index.html
[2] http://www.ipb.uni-bonn.de/~filip/annotation-tool/index.html
[3] http://java.sun.com/j2se/1.4.2/docs/api/index.html
[4] dev.mysql.com/downloads/mysql/5.0.html
[5] http://dev.mysql.com/downloads/connector/j/5.0.html

3.1.2 Schnittstellen

Zur Verfügung stehen zwei verschiedene Möglichkeiten für den Zugriff auf SQL:

1. Verbindung zwischen Matlab und SQL über Java:
   Nach der Installation von Java muss der entsprechende Treiber in Matlab geladen werden. Das direkte Laden des Treibers aus Matlab heraus ist jedoch nicht möglich. Aus diesem Grund wurde unter Eclipse eine Java-Klasse erzeugt, die den Treiber lädt.[6] Diese Klasse kann in Matlab mit dem Befehl
   `con=JavaSQLVerbindung()`
   aufgerufen werden. Neben dem Befehl zum Laden des Treibers sind hierin direkt Benutzername und Passwort enthalten, so dass der Anwender diese nicht jedes Mal angeben muss.
   Weiterhin wird ein jar-File des MySQL-Connectors benötigt, um die Verbindung zu MySQL sicherzustellen. Der Pfad zu *mysql-connector-java-5.0.5-bin.jar* wird hierfür in der Datei *classpath.txt* unter Matlab eingetragen.

2. Matlab Database Toolbox für Zugriff auf SQL:
   Analog zur ersten Variante muss der Pfad zu MySQL in die Datei classpath.txt eingetragen werden. Der erste Teil - das Erstellen einer Java-Klasse zum Laden des Treibers - ist jedoch nicht erforderlich. Die Database Toolbox sieht bereits eine Funktion hierfür vor.
   `database`('*databasename*', '*username*', '*password*', *Treiber*, *url*)
   Um die Verbindung zur Datenbank herzustellen, ist
   `con=SQLVerbindung()`
   aufzurufen. Dieser Befehl enthält neben der Funktion zum Laden des Treibers ebenfalls bereits Angaben zu Benutzername und Passwort, so dass der Anwender diese nicht jedes Mal angeben muss.

3.1.3 Einrichten der Datenbank - Anlegen von Benutzer und Passwort

Das Einrichten der Datenbank erfolgt unabhägig von Matlab in der Eingabeaufforderung. Dazu muss man sich als Administrator (`username` = '*root*' und `host` = '*localhost*') einloggen:
`mysql -u root` bzw.
`mysql -u` *root* `-h` *localhost*

[6] Der Quellcode diese Klasse befindet sich in Anhang A.

In der Grundeinstellung ist kein Passwort vergeben. Wir haben an dieser Stelle zusätzlich noch ein Passwort für *'root'* definiert:

`set password = password`(*'password'*) bzw.

`set password for` *username*@*hostname* `= password`(*'password'*)

Zum Nachschlagen sei auf das Handbuch von MySQL verwiesen.[7]

Zusätzlich ist es sinnvoll einen neuen Benutzer mit eigenem Passwort und eingeschränkten Rechten anzulegen, da das Benutzerpasswort beim Einloggen von Matlab aus offen zugänglich ist. Der SQL-Befehl für das Erstellen eines Benutzers lautet

`create user` *username*@*hostname* `identified by` *'password'*

Als Hostname sind die Bezeichnungen '*localhost*' oder '%' (für alle hosts) möglich - im letzten Fall kann man auch einfach @*hostname* weglassen.
Die Zugriffsrechte der einzelnen Benutzer können in der Eingabeaufforderung mit `select * from mysql.user` erfragt werden. Mit dem Befehl `grant` können Benutzerrechte vergeben werden. Für genauere Information hierzu sei ebenfalls auf das oben erwähnte MySQL-Handbuch verwiesen.

Ebenfalls in der Eingabeaufforderung wird die Datenbank erstellt. Dies geschieht mit Hilfe des Befehls:

`create database` *databasename*

Anschließend können wir die Datenbank mit Daten füllen. Bevor wir uns jedoch konkreten Anfragen widmen, wollen wir nochkurz auf den Datenbankzugriff von Außen eingehen.

3.2 Zugriff von Außen

Für den Zugriff auf die Datenbank von außen benötigen wir neben der obengenannten Software noch die IP-Adresse des Rechners auf dem sich die Datenbank befindet. Diese lautet für den Institutsrechner, der als Datenbankserver fungiert:

`131.220.233.172`

Sie ersetzt die Angabe der Hostnamen '*localhost*' sowohl in der Eingabeaufforderung als auch in der Matlab-Funktion `JavaSQLVerbindung`() bzw. `SQLVerbindung()`.
Die Befehlszeile in der Eingabeaufforderung lautet dann:

`mysql -u` *username* `-h` *hostname databasename*

[7] http://dev.mysql.com/doc/refman/5.0/en oder http://dev.mysql.com/doc/refman/5.1/de

3.3 SQL-Anfragen

Die wichtigsten Anfragen an die Datenbank sind das Erstellen von Tabellen sowie das Einfügen und Lesen von Werten. Welche SQL-Befehle mit welchen Matlab-Funktionen aufgerufen werden können, ist der Tabelle 3.3 zu entnehmen. Wir wollen die Umsetzung für die beiden oben beschriebenen Schnittstellen (Kapitel 3.1.2) vergleichen. Im ersten Fall wurden die erforderlichen Funktionen als Java-Anfragen implementiert. In einem zweiten Fall wurde auf die bereits vorhandenen Funktionen der Database Toolbox von Matlab zurückgegriffen. Von den grundlegenden Anfragen an die Datenbank enthält die Matlab DatabaseToolbox jedoch nur solche, die das Lesen und Schreiben der Daten ermöglichen. Es sind keine expliziten Funktionen zum Erstellen oder Löschen von Tabellen vorgesehen. Diese wurden aus den Java-Anfragen übernommen und modifiziert.

SQL Anfrage / Beschreibung	ohne Database Toolbox	mit Database Toolbox
`create table` *Tabellenname* → Erstellen einer Tabelle	`erstelleTabelle`(*con*,*Tabellenname*,*Spalten*, *Art*,*key*)	`erstelleTabelle`(*con*,*Tabellenname*,*Spalten*, *Art*,*key*)
`drop table` *Tabellenname* → Löschen einer Tabelle	`loescheTabelle`(*con*,*Tabellenname*)	`loescheTabelle`(*con*,*Tabellenname*)
`insert into` *Tabellenname* (*Spaltenname1*, ...) `values` (*Wert1*, ...) → Einfügen von Werten in eine vorhandene Tabelle	`SQLEinfuegen`(*con*,*Tabellenname*,*Spalten*, *Werte*)	`fastinsert`(*con*,'*Tabellenname*',{'*Spaltenname1*', ...},*exdata*); → exdata ist eine Matrix oder ein Cell-Array der einzufügenden Elemente
`delete from` *Tabellenname* `where` ... → Löschen von Werten aus einer Tabelle	`loescheEintrag`(*con*,*Tabellenname*, *Bedingung*)	`loescheEintrag`(*con*,*Tabellenname*, *Bedingung*)
`select` ∗ `from` *Tabellenname* `where` ... → Lesen aus einer vorhandenen Tabelle		
- den ersten Wert	`SQLAnfrageErstes`(*con*,*query*,*spalte*)	`curs = exec`(*con*,*query*) `curs = fetch`(*curs*) `k = curs.Data(:,1)`
- alle Werte	`SQLAnfrageAlle`(*con*,*query*,*spalte*)	`curs = exec`(*con*,*query*) `curs = fetch`(*curs*) `k = curs.Data`

3.4 Vergleich zwischen MATLAB-Database-Toolbox und eigener Lösung

In diesem Kapitel wollen wir die beiden SQL-Zugriffsmöglichkeiten (mit bzw. ohne Matlab Database Toolbox) vergleichen. Kriterien hierfür sind Anwenderfreundlichkeit (einschließlich Kosten), Umsetzung und Laufzeit.
In beiden Fällen ist die Installation von Matlab, Java und beispielsweise MySQL erforderlich (siehe Kapitel 3.1.1). Bei der Verwendung der Database Toolbox entfällt lediglich das Erstellen einer Java-Klasse zum Laden des Treibers. Der Eintrag des MySQL-jar-Files in der classpath.txt Datei von Matlab ist in beiden Fällen erforderlich. In der Database Toolbox wird zwar darauf hingewiesen. Diesen Vermerk findet man jedoch erst nach einer 'längeren' Suche.
Ein weiterer Nachteil der Database Toolbox ist das Fehlen einer Methode zum Erstellen, Ändern und Löschen von Tabellen. Man kann eine solche Methode mit den vorhandenen Mitteln jedoch leicht selber schreiben.
Die Umsetzung der Anfragen in der Database Toolbox ist stark an die Formulierung in Java angelehnt. Wer sich jedoch mit SQL-Anfragen in Java auskennt, muss hier umdenken, da die Formulierungen ähnlich aber nicht identisch sind. Beispielsweise lautet der Befehl zum Trennen der Verbindung `con.close()` oder `close(con)`.
Was jedoch positiv auffällt, ist die Ausgabe zu den Anfragen an die Datenbank. Während man sich bei der Anfrage über Java mit `resultset.first()` und `resultset.next()` an der Ergebnismenge entlanghangelt, bekommt man aus der Database Toolbox alle Werte der Anfrage auf einmal. Die Ausgabe kann wahlweise als Matix, Cell-Array oder Strukt erfolgen. Für die Ausgabe als Matrix ist beispielsweise setdbprefs('datareturnformat','numeric') anzugeben.
Für den unerfahrenen Anwender bietet die Database Toolbox zusätzlich noch einen Querybuilder, mit dessen Hilfe sich leicht SQL-Anfragen erstellen lassen.[8] Kennt man sich jedoch mit dem Erstellen von SQL-Anfragen aus, so ist die Verwendung des Querybuilders jedoch eher hinderlich.
Als letztes Kriterium für den Vergleich der beiden Schnittstellen soll ein Laufzeitenvergleich herangezogen werden. In Tabelle 3.4 sind die Ergebnisse dieses Laufzeitenvergleichs dargestellt, einmal unter Verwendung der Database Toolbox und einmal ohne diese Toolbox.

[8] Erläuterungen zur Anwendung des Querybuilders befinden sich im Anhang B.

Funktion	mit DB Tool [sec]	ohne DB Tool [sec]
Verbindung zu Datenbank herstellen	0.011	0.174
Verbindung trennen	0.247	0.004
create table ... (5 Spalten)	0.068	0.065
insert into ...	0.041	0.025
delete from ... where ...	0.025	0.024
drop table ...	0.028	0.026
select *		
1 Knoten (geg. ID)	0.014	0.002
alle Knoten in 1 Bild und 1 Skala	0.251	0.149
1 Kante (geg. 1 Knoten und 1 Masche)	0.184	0.170
alle Kanten in 1 Bild und 1 Skala (erfordert zus. die Abfrage der kleinsten und größten KnotenID im Bild)	1.935	1.738

Auffallend sind die längeren Laufzeiten für fast alle Anfragen. Lediglich das Herstellen der Datenbankverbindung ist unter Verwendung der Database Toolbox schneller. Fasst man jedoch das Herstellen und Trennen der Datenbankverbindung zusammen, schneidet die Database Toolbox auch hier schlecht ab.

4 Die Datenbank der Wasserscheidenbilder

An diese Stelle wollen wir explizit auf die Datenbank der vektorisierten Wasserscheidenbilder eingehen. Wir behandeln neben der Struktur der Datenbank auch Möglichkeiten zur Merkmalsextraktion mit Hilfe der Datenbank. Die Datenbank der Winged-Edge-Struktur der Wasserscheidenbilder trägt den Namen '*maschen*'. In der Eingabeaufforderung kann man sich unter dem Benutzernamen '*wingededge*' und dem gleichnamigen Passwort wie in Kapitel 3.2 beschrieben einloggen:

`mysql -u` *wingededge* `-h` *131.220.233.172* maschen

Beim Zugriff auf die Datenbank von Matlab aus muss das Passwort nicht mehr angegeben werden, da es bereits in der Funktion zum Einbinden von Java enthalten ist.
Um Anfragen an die '*maschen*'-Datenbank stellen zu können, werden wir hier zunächst auf deren Gliederung eingehen. Anschließend werden die für den Zugriff implementierten Funktionen vorgestellt.

4.1 Gliederung der Datenbank

Dei Struktur der Datenbank orientiert sich an der in Kapitel 2.1 beschriebenen Winged-Edge-Struktur. Alle Objekte werden über eine eindeutige Identität (ID) referenziert. Sowohl die Knoten als auch die Maschen enthalten eine eindeutige Angabe darüber, auf welches Bild und welchen Skalenindex sie sich beziehen. Zur Verfügung stehen bereits folgende Tabellen:

KnotenID	Zeile	Spalte	Bild	Skala
integer	*integer*	*integer*	*integer*	*integer*

Zu jedem Knoten werden dessen Koordinaten gespeichert. Das Koordinatensystem hat seinen Urspruing im linken oberen Bildpixel. Wir können die Koordinaten daher wie in einer Matrix zeilen- und spaltenweise abgreifen.

MaschenID	Label	Bild	Skala	Art
integer	*integer*	*integer*	*integer*	*integer*

Pro Bild und Skala gibt es ein eindeutiges Labeling der Maschen. Die Nummerierung beginnt jedes Mal erneut vom Wert 1. Um eine global eindeutige ID zu erhalten, werden alle zuvor vergebenen Label aufsummiert. Die für die Lernphase einer Klassifikation benötigte Information über die Klassenzugehörigkeit von Bildsegmenten wird in der Spalte 'Art' direkt in der Datenbank mitverwaltet.

ElementID	Element
integer	*varchar2*

Die Art der Bildobjekte, die in der Maschen-Tabelle nur als Integer-Wert aufgeführt wird, wird hier näher beschrieben.

KantenID	KnotenVor	KnotenNach	MascheLi	MascheRe	KanteVorLi	KanteNachRe
integer	*integer*	*integer*	*integer*	*integer*	*integer*	*integer*

Hier tritt die eigentliche Winged-Edge-Struktur (wie in Kapitel 2.1 beschrieben) zutage. Die Kanten erhalten ihre eindeutige Zuordnung zu einem Bild und einer Skala über die Knoten bzw. Maschen.

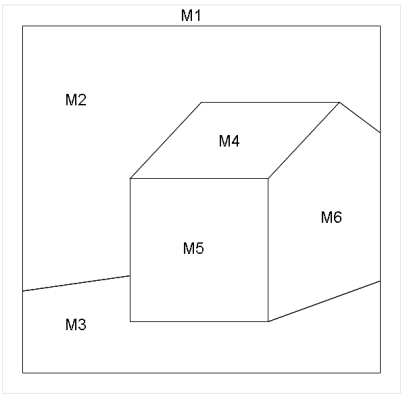
BildID	Bild	Skala	MaschenAnz	KantenAnz	KnotenAnz	ZeilenAnz	SpaltenAnz
integer	*integer*	*integer*	*integer*	*integer*	*integer*	*integer*	*integer*

In dieser Tabelle werden die Bilddimensionen verwaltet. Die Bildgröße wird über die Anzahl der Zeilen und Spalten definiert. Weiter ist die Anzahl der Bildobjekte aufgeführt.

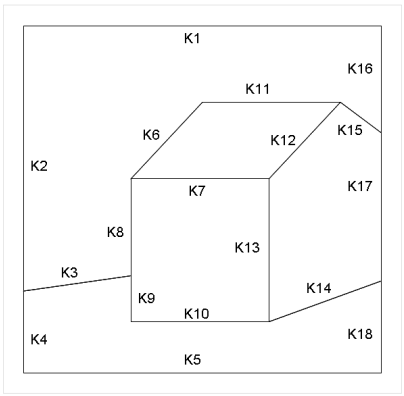
PKantenID	KnotenVor	KnotenNach	Masche
integer	*integer*	*integer*	*integer*

Die Bildstruktur, die in der Tabelle der Kanten gespeichert wird ist zwar konsistent, jedoch für einzelne Arten von Bildmerkmalen ungeeignet, da hier Strukturen in zu kleine Teile zerlegt werden. Aus diesem Grund wurde eine weitere Tabelle angelegt, die Kanten einer Masche unabhängig von Kreuzungspunkten und angrenzenden Maschen speichert. Die hier aufgeführten Kanten wurden mit Hilfe des Douglas-Peucker-Algorithmus weiter zusammengefasst und beschreiben somit größere Strukturen.

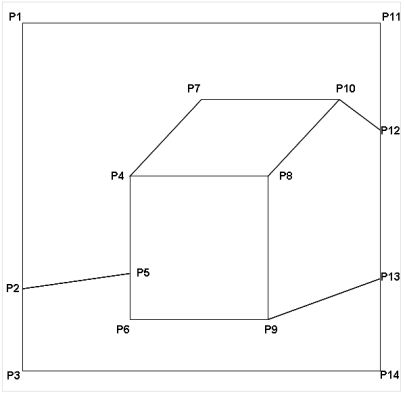
Zur Veranschaulichung greifen wir wieder das Beispiel aus Kapitel 2.2 auf (Abbildung 2). Das Bild liegt nur im Vektorformat vor, wie in Abbildung 8 dargestellt.



(a) Maschen



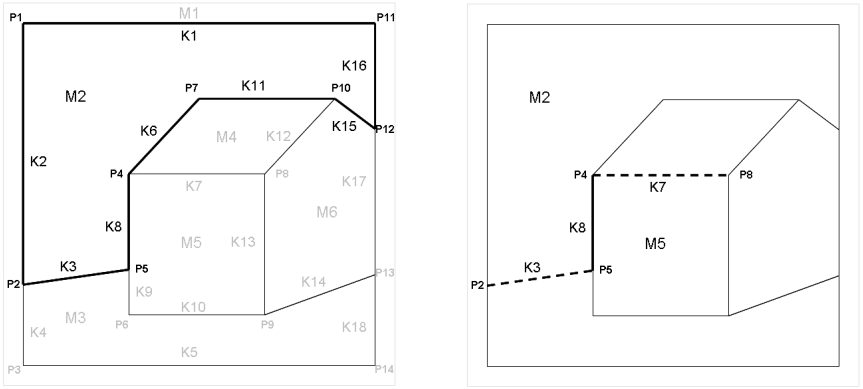
(b) Kanten



(c) Knoten

Abbildung 8: Darstellung der Landkarte in Vektorformat.

Die Struktur der Datenbank wollen wir nun beispielhaft an der Masche mit Label 2 nachvollziehen (Abbildung 9). Diese Masche enthält die Kanten mit den ID's $(1, 2, 3, 6, 8, 11, 15, 16)$. Die Kante mit der ID 8 wird von zwei Knoten begrenzt (Knoten 4 und 5). Neben der Masche 2 ist die Kante 8 noch zur Masche mit der ID 5 adjazent. Des Weiteren gibt es genau eine Vorgängerkante im Umring der linken Masche sowie eine Nachfolgerkante im Umring der rechten Masche. Diese Kanten haben die ID's 3 und 7.



(a) Masche 2 mit allen adjazenten Knoten $(1, 2, 4, 5, 7, 10, 11, 12)$ und Kanten $(1, 2, 3, 6, 8, 11, 15, 16)$.

(b) Kante 8 mit allen adjazenten Knoten und Maschen sowie ihrer Vorgänger- und Nachfolgerkante.

Abbildung 9: Beispiel der Winged-Edge-Struktur.

Die zugehörigen Tabellen haben folgende Form:

KnotenID	Zeile	Spalte	Bild	Skala
1	2	2	1	1
2	21	2	1	1
3	27	2	1	1
4	11	10	1	1
5	20	10	1	1
6	23	10	1	1
7	7	14	1	1
⋮	⋮	⋮	⋮	⋮
14	27	29	1	1

MaschenID	Label	Bild	Skala	Art
1	1	1	1	1
2	2	1	1	4
3	3	1	1	5
4	4	1	1	2
5	5	1	1	2
6	6	1	1	2

ElementID	Element
1	Außen
2	Gebäude
3	Fenster
4	Himmel
5	Vegetation
⋮	⋮

KantenID	KnotenVor	KnotenNach	MascheLi	MascheRe	KanteVorLi	KanteNachRe
⋮	⋮	⋮	⋮	⋮	⋮	⋮
8	5	4	2	5	3	7
⋮	⋮	⋮	⋮	⋮	⋮	⋮

BildID	Bild	Skala	MaschenAnz	KantenAnz	KnotenAnz	ZeilenAnz	SpaltenAnz
1	1	1	6	18	14	28	30
⋮	⋮	⋮	⋮	⋮	⋮	⋮	⋮

4.2 Funktionengruppen

Drei Arten von Funktionen wurden implementiert. Zur ersten Gruppe gehören die Grundfunktionen für die SQL-Anfragen, die bereits in Kapitel 3.3 vorgestellt wurden. Sie erfüllen die Aufgaben vom Erstellen von Tabellen, über das Füllen dieser mit Daten bis zum Auslesen der Daten. Eine zweite Funktionengruppe übernimmt die Aufgabe des Vektorisierens, wie es in Kapitel 2.2 beschrieben wird. Hier werden Knoten, Kanten und Maschen extrahiert und in Form einer Winged-Edge-Struktur in die Datenbank geschrieben. Die zentrale Funktion ist SQLWingedEdge. Hier wird das Vektorisieren und das Befüllen der Datenbank koordiniert. In einer weiteren Gruppe sind diejenigen Funktionen zusammengefasst, die mittels der Daten aus der Datenbank eine Merkmalsextraktion betreiben. Sie sind alle durch den Funktionsnamen 'Abfrage...' gekennzeichnet. Auf die Merkmalsextraktion soll in Kapitel 4.3 eingegangen werden.
Der Matlab-Code zu allen Funktionen ist in der Subversion-Zentrale des Fachbereichs Photogrammetrie zu finden.[9]

4.3 Merkmalsextraktion mit Hilfe der Datenbank

Um Merkmale von Bildelementen (wie beispielsweise Maschen oder Kanten) zu erhalten, können wir diese Objekte nun aus der Datenbank abgreifen. Wir stellen neben den Funktionen zum Abgriff der Daten auch solche zur Verfügung, die daraus Merkmale berechnen.
Behandelte Merkmale beziehen sich vornehmlich auf die Idee, Fenster anhand

[9] http://www.ipb.uni-bonn.de/papers/

ihrer Geometrie aus den Maschen zu detektieren. Der Hauptaugenmerk liegt daher auf der Betrachtung von Kanten.
Zu den bereit gestellten Funktionen gehören:

- `AnfrageMaschenrand`(*con*, *Bildnummer*, *Skala*, *Masche*, *Plot*) - Bestimme alle Knoten (Zeile, Spalte), die zu einer Masche gehören. Die Reihenfolge der Auflistung der Knoten bestimmt die Reihenfolge der Verbindung dieser zu Kanten. Auf Wunsch wird die Ausgabe auch geplottet. In diesem Fall ist Plot auf 1 zu setzen.

- `AbfrageAnzKnoten`(*Bildnummer*, *Skala*, *Bildgröße*, *Knoten*) - Bestimme die Anzahl der Knoten aus einer Masche. Diese Anfrage ist mit `AnfrageMaschenrand`(...) zu kombinieren.

- `AbfrageKantenlaenge`(*Anfangsknoten*, *Endknoten*) - Bestimme den Abstand zwischen allen aufeinanderfolgenden Knoten einer Masche (also den Anfangs- und Endknoten einer Kante).

- `AbfrageUmfang`(*Knoten*) - Bestimme den Umfang einer Masche, wenn deren Knoten bekannt sind. Diese Anfrage ist mit `AnfrageMaschenrand` (...) zu kombinieren. Der Unterschied zu `AbfrageLochrand`(...) besteht darin, dass Verbindungen zwischen Maschen und ihren Löchern nicht berücksichtigt werden.

- `AbfrageAnzLoecher`(*Bildnummer*, *Skala*, *Knoten*) - Bestimme die Anzahl der Maschen, die innerhalb einer Masche liegen. Diese Anfrage ist mit `AnfrageMaschenrand`(...) zu kombinieren.

- `AbfrageLochrand`(*Bildnummer*, *Skala*, *Knoten*) - Bestimme die Randlänge der Löcher in einer Masche im Verhältnis zur Randlänge der Masche. Diese Anfrage ist mit `AnfrageMaschenrand`(...) zu kombinieren.

- `AnfrageNachbarmaschen`(*con*, *Bildnummer*, *Skala*, *Masche*) - Bestimme die Label der an eine Masche angrenzenden Maschen.

- `AbfrageRaenderNachbarmschen`(*con*, *Bildnummer*, *Skala*, *Masche*) - Bestimme die Knoten aller Nachbarmaschen.

- `AbfragePeuckerrand`(*con*, *Bildnummer*, *Skala*, *Masche*, *Plot*) - Bestimme diejenigen Knoten einer Masche, die nach Anwendung des Peuckeralgorithmus nicht wegfallen.

- `AbfrageAnzPeuckerpunkte`(*Bildnummer*, *Skala*, *Bildgröße*, *Knoten*) - Bestimme die Anzahl der Peuckerpunkte einer Masche. Diese Anfrage ist mit `AnfragePeuckerrand`(...) zu kombinieren.

- `AbfrageOrthogonalanteil`(*Bildnummer*, *Skala*, *Knoten*) - Bestimme den Anteil orthogonaler Kanten einer Masche. Diese Anfrage ist mit `AnfrageMaschenrand`(...) oder `AnfragePeuckerrand`(...) zu kombinieren.

- `AbfrageParallelanteil`(*Bildnummer*, *Skala*, *Knoten*) - Bestimme den Anteil paralleler Kanten einer Masche. Diese Anfrage ist mit `Anfrage Maschenrand`(...) oder `AnfragePeuckerrand`(...) zu kombinieren.

5 Ausblick

Einer Überarbeitung bedarf die Form des Datenbankschemas. Zur Zeit werden manche Informationen, wie beispielsweise Bildnummer und Skala, redundant gespeichert. Ziel ist die Anpassung des Datenbankschemas an die Normalform [SS96], das bedeutet eine nicht-redundante Informationsverwaltung.
Weiterhin sind Überlegungen bezüglich der Form der Ausgangsdaten anzustellen. Ausgangspunkt für die Vektorisierung stellen bislang Wasserscheidenbilder (Binärbilder mit einer Pixel-Darstellung der Wasserscheiden) dar. Wir sind zunächst davon ausgegangen, dass der Wasserscheidenalgorithmus ausschließlich Wasserscheidenbilder liefert, die in eine konsistente Winged-Edge-Struktur überführt werden können. Das bedeutet unter anderem, dass die Wasserscheiden nicht ins Leere laufen und nur ein Pixel breit sind. Entgegen dieser Annahme wurde nach neueren Untersuchungen jedoch Gegenteiliges gezeigt.
Dies ändert zwar nichts an der Form der Datenbank, es legt jedoch eine andere Herangehensweise zum Ableiten der Vektordaten nahe. Eine Idee besteht nun darin, von segmentierten Bildern mit 'Crack Edges' auszugehen. Die Segmente werden hierbei nicht mehr von Pixeln begrenzt, sondern stoßen direkt aneinander. 'Crack Edges' lassen sich auch aus einem Wasserscheidenbild herleiten.
Dieser Ansatz soll einen Teil weiterer Untersuchungen darstellen.

A Java-Klasse zum Laden der Treibers

Wir wollen von Matlab aus auf eine SQL-Datenbank zugreifen. Die Verwendung von Matlab ohne die Database Toolbox erfordert dabei zusätzlich das Erstellen einer Java-Klasse, die den Java-Treiber lädt. Nachstehend ist der Quellcode aufgeführt:

```
import java.io.PrintWriter;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class sqlstart {

    public static void ladeTreiber() {
        try {
            Class.forName (,,com.mysql.jdbc.Driver‘‘);
        }
        catch (ClassNotFoundException ex){
            System.out.println (,,ClassNotFound Exception:‘‘
            + ex.getMessage());
        }
    }
}
```

Dieser Quellcode kann beispielsweise mit der Entwicklungsumgebung Eclipse compiliert werden, um so eine '.class'-Datei zu erzeugen, die wiederum von Matlab gelesen werden kann. Hierbei ist darauf zu achten, dass die gleiche Java-Version verwendet wird, die auch Matlab verwendet. Beispielsweise verwendet Matlab 7.0 die Version j2sdk1.4. Ist bereits eine andere Version installiert, so kann man die Eigenschaften (Properties) von Eclipse in zwei Schritten umstellen. Diese Schritte werden im Folgenden beschrieben und illustriert.

Zunächst ist der Eintrag in den Bibliotheken, wie in den Abbildung 10 - 15 dargestellt, zu ändern. Hierzu wählt man in der Menüleiste → File → Properties aus. Man erhält das in Abbildung 10 dargestellte Fenster. Unter 'Java Build Path' findet man die Bibliotheken (Libraries). Mit 'Add Library ...' kann eine neue Bibliothek hinzugefügt werden. Dazu ist die 'JRE System Library' auszuwählen (Abbildung 11(a)), bevor man mit 'Next' zu dem Fenster

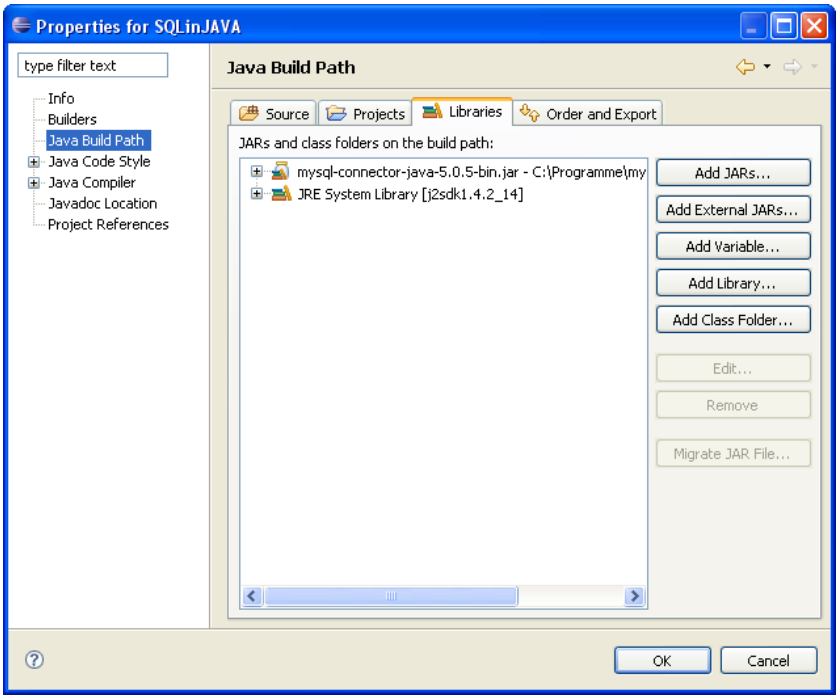
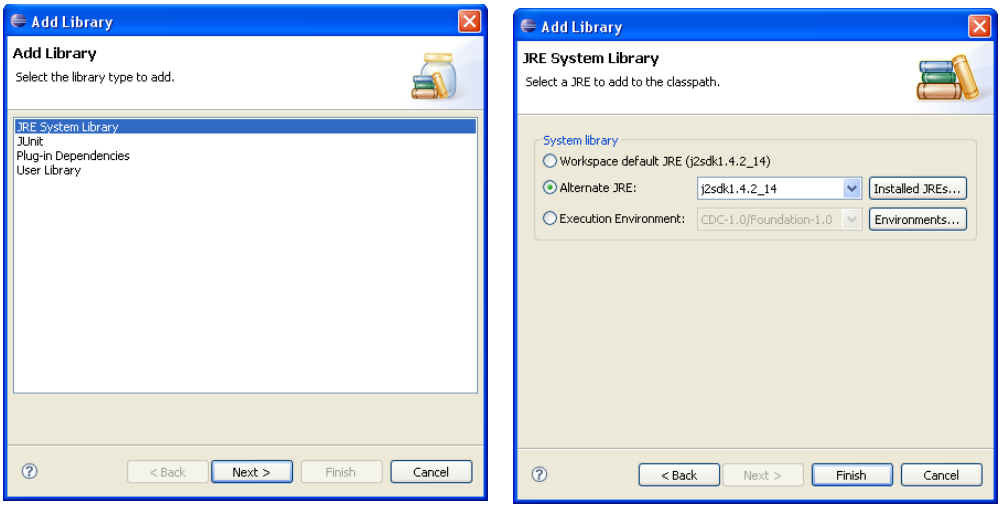


Abbildung 10: Laden der von Matlab verwendeten Java-Bibliotheken in Eclipse. (Schritt 1: Auswahl von 'Java Build Path' im Fenster 'Properties' und Hinzufügen einer neuen Bibliothek mit 'Add Library ...'.)

'Add Library' gelangt (Abbildung 11(b)). Hier ist der Button 'Installed JREs ...' zu drücken. Im Fenster 'Preferences' werden die installierten JRE's dargestellt (Abbildung 12). Um hier den Eintrag der gewünschten Java-Version zu erhalten, ist der Button 'Add ...' zu drücken. Im nächsten Schritt wird im Fenster 'Create JRE' (Abbildung 13 und 14) das Verzeichnis 'JRE home directory' auszuwählen, mit OK zu bestätigen. Dann muss nur noch die richtige Bibliothek mit einem Häkchen versehen (Abbildung 15) und der Vorgang in 'Add Library' mit 'finish' fertig gestellt werden.



(a) Schritt 2: Auswahl von 'JRE System Library' und Bestätigung mit 'Next'.

(b) Schritt 3: 'Installed JREs ...' anzeigen lassen.

Abbildung 11: Laden der von Matlab verwendeten Java-Bibliotheken in Eclipse. (Schritte 2+3)

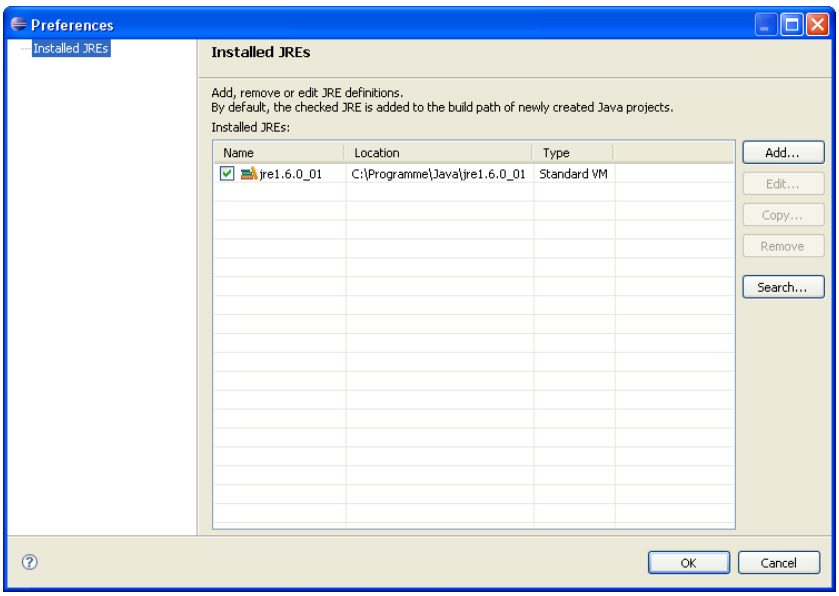


Abbildung 12: Laden der von Matlab verwendeten Java-Bibliotheken in Eclipse. (Schritt 4: Im Fenster 'Preferences' können JRE's durch Drücken des 'Add ...'-Buttons hinzugefügt werden.)

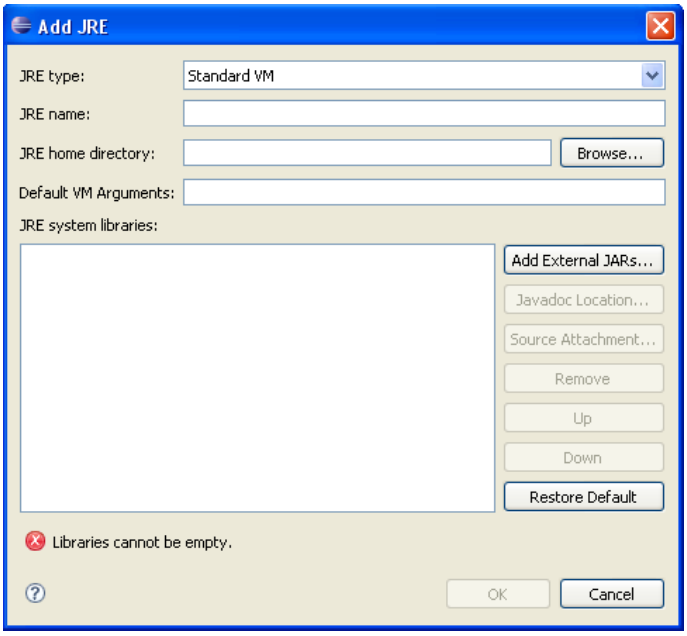


Abbildung 13: Laden der von Matlab verwendeten Java-Bibliotheken in Eclipse. (Schritt 5: Auswahl des Verzeichnis 'JRE home directory' im Fenster 'Add JRE'.)

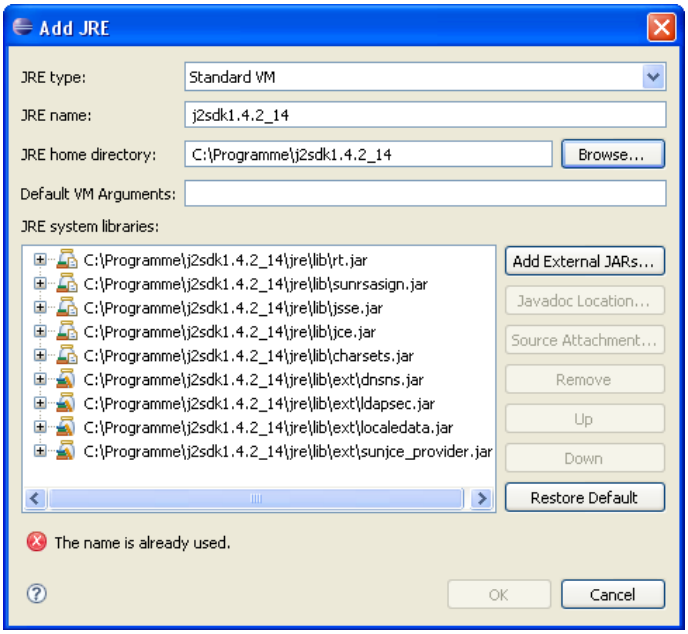


Abbildung 14: Laden der von Matlab verwendeten Java-Bibliotheken in Eclipse. (Schritt 6: Nachdem das Verzeichnis 'JRE home directory' gewählt wurde, wird das Fenster 'Add JRE' durch Drücken der 'Ok'-Taste geschlossen.)

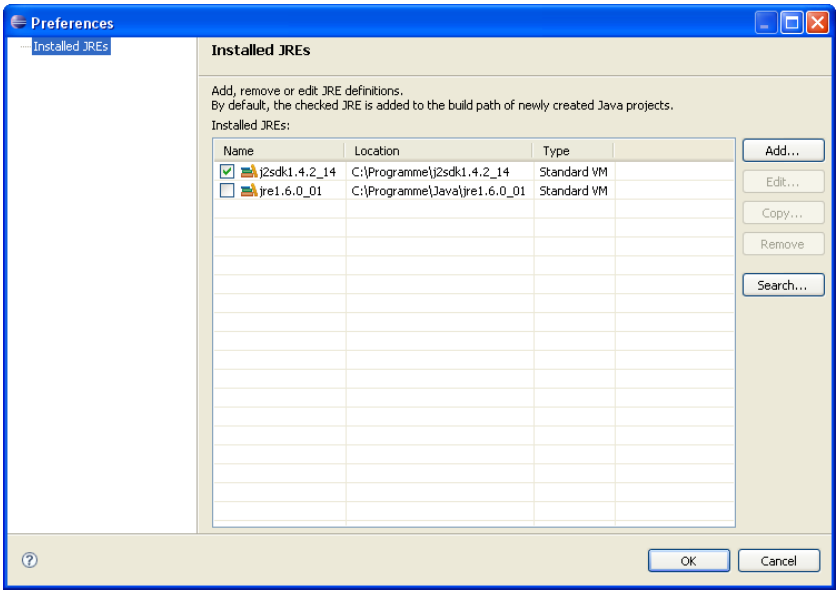


Abbildung 15: Laden der von Matlab verwendeten Java-Bibliotheken in Eclipse. (Schritt 7: Die neu eingetragene Bibliothek mit einem Häkchen versehen.)

Ebenfalls unter Properties ist der richtige Java Compiler (in unserem Fall 1.4) einzustellen. Anschaulich ist dies in Abbildung 16 dargestellt.

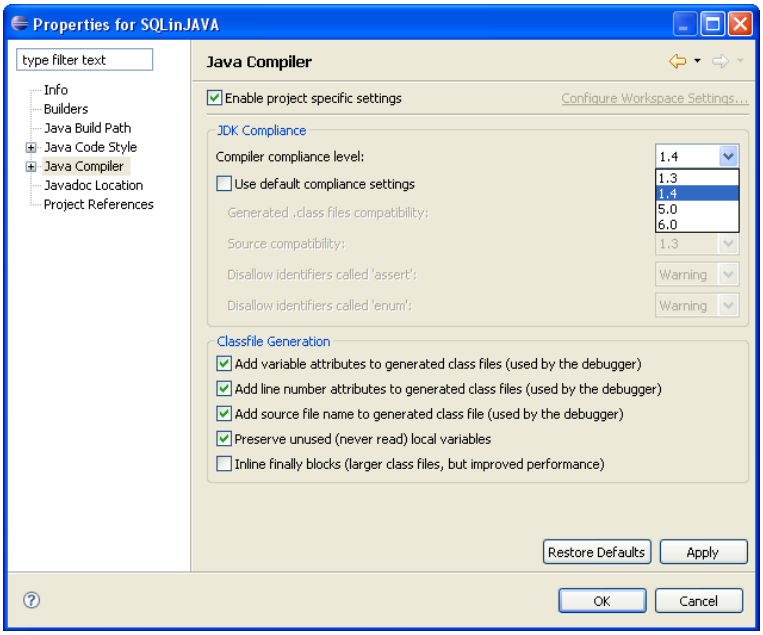


Abbildung 16: Einstellen des richtigen Compilers in Eclipse.

B Visual-Query-Builder der Database Toolbox

Die Matlab Database Toolbox verfügt über einen Querybuilder zum Erstellen von SQL-Anfragen. Dieser ist in Abbildung 17 dargestellt. Seine Bedienung soll im Folgenden kurz erläutert werden.

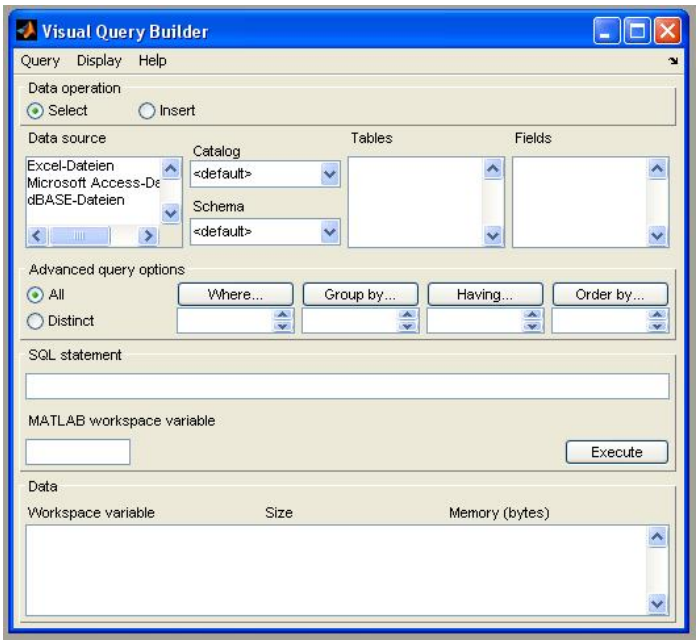


Abbildung 17: Der Querybuilder aus der Matlab Database Toolbox.

Als erster Schritt ist in der Liste 'data source' eine Datenquelle auszuwählen (siehe Abbildung 17). Ist die gewünschte Datenquelle noch nicht vorhanden, so muss sie erst geladen werden. Dies geschieht in der Menüleiste unter der Auswahl 'Query' und 'Define JDBC Data Source ...', wie in Abbildung 18 dargestellt. Es öffnet sich ein Fenster (Abbildung 19), in dem man neue Datenquellen definieren oder bereits definierte Datenquellen auswählen kann. Dies geschieht durch Angabe des Datenbanknamen, des Treibers und der URL-Adresse. Nach Auswahl der Datenbank, erscheint diese nun in der Liste des Ausgangsfensters des Visual-Query-Builder (Abbildung 20(a)).
Wird die neu eingefügte Datenbank ausgewählt, so erscheinen nach Angabe des Benutzernamen und Passwortes alle vorhandenen Tabellen (Abbildung 20(b)). In den Listen 'tables' und 'fields' kann man nun die gewünschten Tabellen und Spalten auswählen. Automatisch wird daraus eine SQL-Anfrage erstellt, die im Feld 'SQL statement' angezeigt wird (Abbildung 20(c)). Zusätzliche Bedingungen können über die Tasten 'Where ...', 'Group by ...',

'Having ...' und 'Order by ...' angegeben werden. Für die Where-Klausel erscheint beispielsweise ein Fenster (Abbildung 21), in dem man Einschränkungen der Ausgabe vornehmen kann.
Das Ausführen der SQL-Anfrage wird (nach Angabe der 'MATLAB workspace variable') mit der Taste 'Execute' vollzogen.

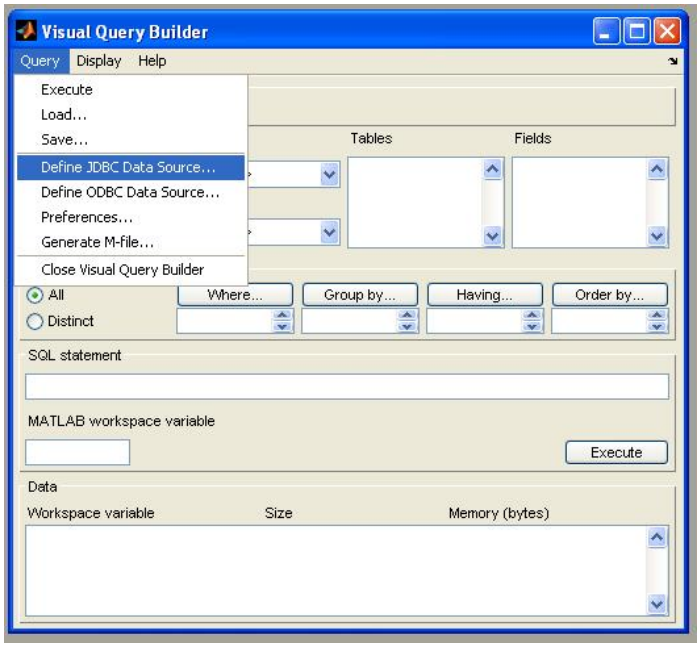


Abbildung 18: Erstellen einer SQL-Anfrage mit Hilfe des Querybuilders aus der Matlab Database Toolbox. (Schritt 1)

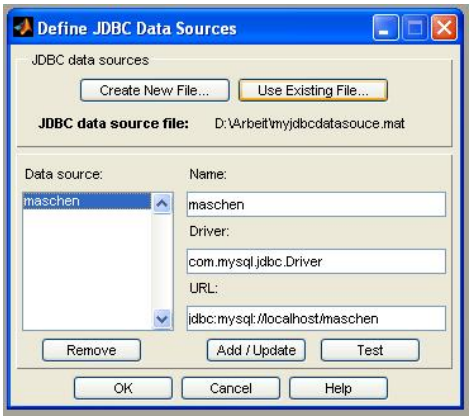


Abbildung 19: Auswahl einer Datenquelle im Querybuilder aus der Matlab Database Toolbox.

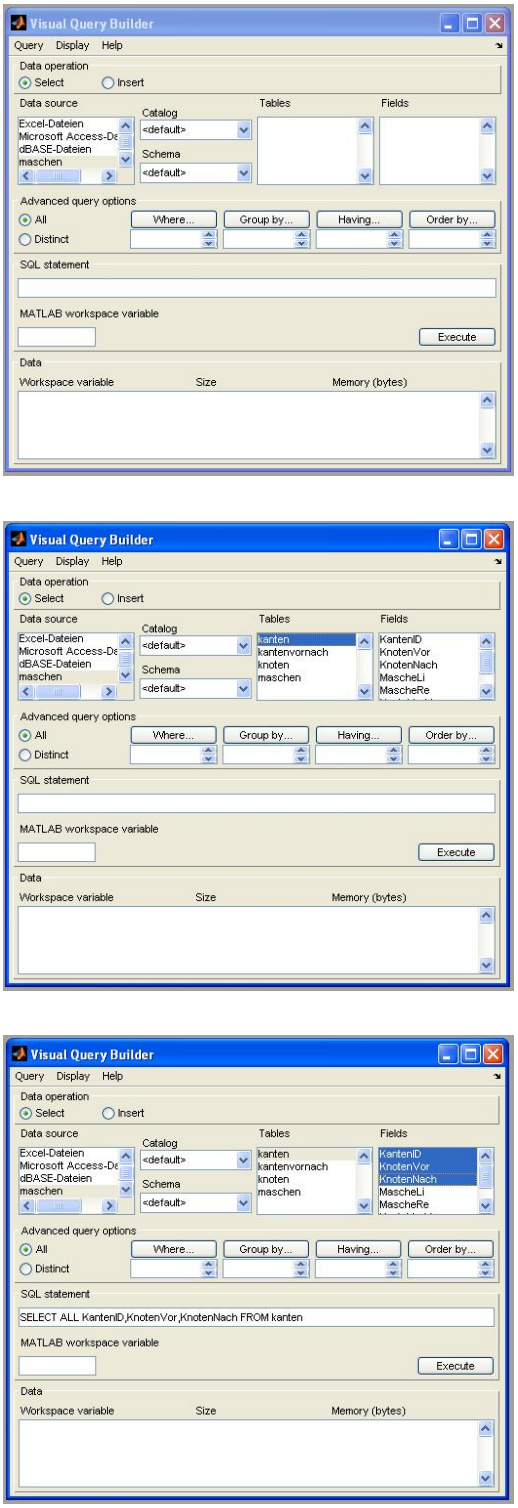


Abbildung 20: Erstellen einer SQL-Anfrage mit Hilfe des Querybuilders aus der Matlab Database Toolbox. (Schritte 3-5: (a) Auswahl der Datenquelle. (b) Auswahl der Tabelle. (c) Auswahl der Spalten einer Tabelle.)

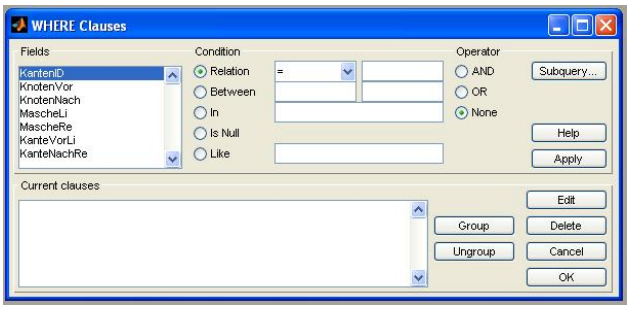


Abbildung 21: Fenster des Querybuilder aus der Matlab Database Toolbox zum Erstellen einer Where-Klausel in einer SQL-Anfrage an die Datenbank.

Literatur

[DP73] David H. Douglas and Thomas K. Peucker. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Canadian Cartographer*, 10(2):112–122, December 1973.

[SS96] Henry F. Silberschatz, Abraham; Korth and S. Sudarshan. *Database System Components*. The McGraw-Hill Companies Inc., 1996.

[Wor95] Michael Worboys. *GIS: A Computing Perspective*. Taylor & Francis Inc., 1995.