ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

INSTITUTE of CARTOGRAPHY

Vertiefungsblock SS 2004

Frühe Vermessungen

von

Urs Aebi und Hannes Eugster

Juni 2004

Zusammenfassung

Bei der Erforschung nach vorchristlichen Raumstrukturen in der Schweiz wurden sogenannte Kultlinien gefunden. Kultachsen oder Kultlinien sind heutzutage ersichtlich durch alte christliche Kirchen, die zusammen eine Gerade definieren. Beispiele solcher Kultlinien finden sich im Raum Basel, im Tessin oder in der Surselva (Kanton Graubünden). Erklären lassen sich diese Linien durch Absteckungen von Kultplätzen in vorrömischer, keltischer Zeit. Abgesteckt wurden astronomisch/kalendarische Ereignisse. Mit der Christianisierung wurden diese Kultplätze durch Kirchen, Kapellen oder Wegkreuze besetzt. Die Suche nach solchen Linien erfolgte bis anhin manuell mit Landkarten und Massstab.

Die Erforschung solcher räumlichen Strukturen sollte nun durch moderne Informatikmittel unterstützt werden. Nach der Evaluation von verschiedenen Möglichkeiten wurde eine GIS-Applikation entwickelt, basierend auf der ESRI Produktefamilie ArcGIS. Die neue Applikation erlaubt nun mit einem eigens entwickelten Algorithmus eine grossräumige automatische Suche von Kultlinien mit verschiedenen Eingabeparametern. Detektierte Linien werden mit einer Ausgleichung optimal in die dazugehörenden Punkte eingerechnet. Eine weitere zusätzliche Erweiterung ermöglicht die Ausgleichung einzelner Linien, sei es eine Neuberechnung bestehender Linien mit neuen Parametern oder das Generieren einer neuen Linie durch Eingabe von beliebigen Punkten. Die Resultate werden in einer Datenbank gespeichert und können mit der gesamten GIS Funktionalität weiter interpretiert, analysiert und verwaltet werden.

Inhaltsverzeichnis

1. Einleitung

Verschiedene Autoren, darunter Professor Ammann, welcher diese Vertiefungsarbeit initiiert hat, haben in der Schweiz Kultachsen mit Kartenstudien entdeckt. Kultachsen oder Kultlinien sind heutzutage ersichtlich durch alte christliche Kirchen, welche in einer Geraden aufgereiht sind. Beispiele solcher Kultlinien finden sich im Raum Basel, im Tessin und in der Surselva (Kanton Graubünden). Die Kirchen befinden sich mit einer bemerkenswerten Genauigkeit auf einer gemeinsamen Linie (auf einer Geraden), so dass die Lage der Kirchen nicht als zufällig bezeichnet werden kann. Erklären lassen sich diese Linien durch Absteckungen von Kultplätzen in vorrömischer, keltischer Zeit. Abgesteckt wurden astronomisch/kalendarische Ereignisse. Plausibel und nachvollziehbar sind dabei Absteckungen von Richtungen der Kalenderastronomie im lokalen Horizontsystem, das sind in erster Linie die Sonnen- oder Mondwenden. Die Sonnenwenden bezeichnen den nördlichsten Aufgangs- oder Unterganspunkt der Sonne am Horizont am 21. Juni und die südlichsten Punkte im Winter am 21. Dezember.

Mit der Christianisierung wurden diese Kultplätze mit Kirchen, Kapellen oder Wegkreuzen überbaut. Die Umwidmung älterer, sakraler Orte durch das Christentum war offizielle Kirchenpolitik des Papstes Gregor des Grossen. Die Standorte dieser Kirchen weisen nun indirekt auf eine Vermessung in keltischer Zeit hin. Namen wie Höllboden oder Teufelsstein sind ebenfalls Indizien für alte Kultplätze, mit diesen Bezeichnungen wurden diese Orte tabuisiert.

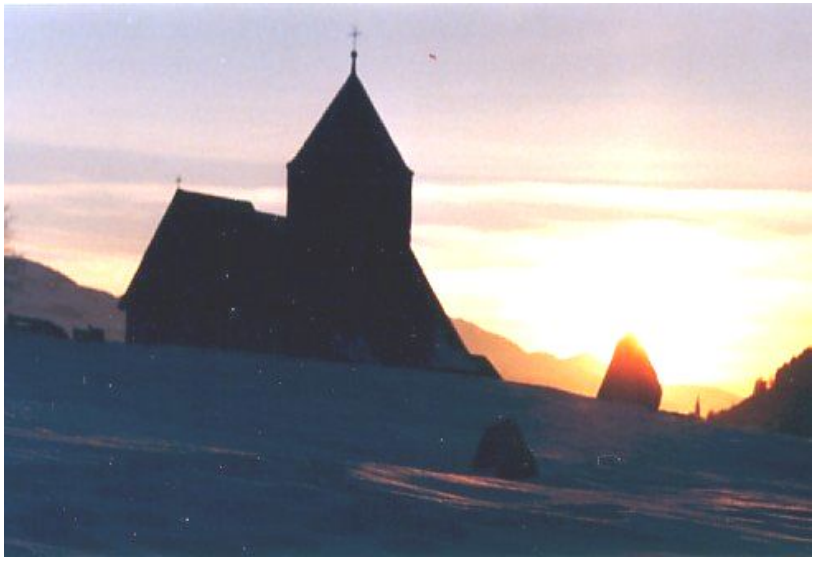


Abbildung 1-1: Kirche Falera

1.1 Bisherige Forschungen und Verfahren

Professor Ammann hat seit 1985 verschiedene Kultlinien mit Kartenstudien gefunden. Markant sind zwei Linien in der Surselva, von denen eine Linie über die Kirchen Schnaus, Ruschein, Ladir, Falera und Laax verläuft. In Falera befinden sich in der gleichen Linie mit dem Azimut von 62° eine Steinreihe[1], sogenannte Menhire, aus der Bronzezeit. Das Azimut von 62° entspricht dem Sonnenaufgang einen Monat vor der Sommersonnenwende. Die Kirche Ladir befindet sich in einer weiteren genau West-Ost ausgerichteten Linie über die Kirchen von Siat, Ladir, Schluein, Sagogn und Valendas.

Ausgeprägt ist auch ein weiteres ganzes System im Tessin, es handelt sich dabei um mehrere Kultlinien, welche sich in einem Pol bei der Kirche San Zenone schneiden. Vermutlich frühgeschichtliche Schalen neben der Kirche von San Zenone sind hier ebenfalls ein Hinweis auf einen Kultplatz. Die einzelnen Richtungen der Kultlinien korrespondieren ebenfalls mit einer astronomischen Bedeutung. Detaillierte Angaben und weitere Beispiele in den Regionen Basel und im Mittelland sind im VPK 7/02 [2] beschrieben.

1.2 Ziel dieses Vertiefungsblocks

Das Ziel dieser Arbeit ist es, eine GIS- Erweiterung für die Suche von Kultlinien in Karten zu entwickeln. Die Erweiterung soll von selektierten Objekten automatisch Kultlinien suchen, Geraden berechnen (Regression) und die Resultate anschliessend graphisch darstellen. Zudem soll eine Datenbank aufgebaut werden, in welcher die Resultate verwaltet werden und zusätzliche Analysen durchgeführt werden können.

[1] Neben der hier erwähnten Steinreihe gibt es in Falera noch weitere Steinsetzungen. Vermessen und dokumentiert wurden die Steinreihe von Ulrich und Greti Büchi. [1]

1.3 Definition verwendeter Begriffe im Bericht

In diesem Bericht werden die Begriffe Linie, Kultlinie, Gerade und Einzellinie verwendet. Hier soll nun eine kurze Definition dieser Begriffe, in welchem Kontext diese zu verstehen sind, folgen. Unter einer Kultlinie wird eine Gerade verstanden, auf welcher sich mit einer gewissen Toleranz mindestens drei Punkte befinden. Der Begriff Linie wird ähnlich verwendet, wichtig für den Begriff Linie ist, dass dieser eine Gerade beschreibt, welche durch Anfangs und Endpunkt begrenzt ist. Mit dem Begriff Gerade ist häufig der Begriff Ausgleichsgerade oder Regressionsgerade verknüpft. Damit ist die Gerade gemeint, welche durch beste Anpassung aus mehreren Punkten bestimmt ist. Der Begriff Einzellinie wird hingegen als direkte Verbindung zweier Punkte verwendet.

2. Anforderungen

Für die GIS- Applikation "Frühe Vermessungen" spezifizierten wir Anforderungen an das zu verwendende Datenschema und die benötigte Funktionalität. Die Anforderungen wurden aufgrund einer vorgängig durchgeführten Problemanalyse erarbeitet. Bei der Problemanalyse wurde die bis anhin angewandte manuelle Forschungsmethode auf Basis der Landeskarten analysiert. Ausgehend von dieser Analyse erarbeiteten wir auf Basis eines Geographischen Informationssystems Konzepte, wie die Forschung nach früheren Raumstrukturen vereinfacht und effizienter gestaltet werden kann. Unsere erarbeiteten Konzepte kombinierten wir mit den Zielen und Ideen von Herrn K. Ammann und definierten anschliessend ein Anforderungskatalog an die GIS- Applikation.
Aufgrund der kurzen Projektdauer, wurde der Projektfokus auf die automatische Detektion von Linien und Polarsystemen gelegt. Die Erforschung der römischen Limitation mit Hilfe eines GIS ist aus zeitlichen Gründen in den Projektanforderungen nicht berücksichtigt. Die beiden nächsten Unterkapitel enthalten die festgelegten Anforderungen an die GIS- Applikation.

2.1 Anforderungen an das Datenschema

Die Daten in dieser Applikation sollen in einem festgelegten Datenschema verwaltet werden. Dabei sollen die Grundlagedaten für die Forschung und Analyse in bestehender Form als Datenbasis der GIS- Applikation zur Verfügung stehen. Das Datenschema für die Verwaltung der Kultlinien und der Polarsysteme, muss den nachfolgenden Kriterien entsprechen. Ein Pol ist als Punktobjekt zu verwalten. Dabei muss ein Pol mindestens zwei Kultlinien referenzieren. Eine Kultlinie ist als Linienobjekt zu speichern. Dabei muss eine Beziehung zu den zur Kultlinie gehörenden Punkte bestehen. Zu einer Kultlinie müssen mindestens die Eigenschaften mittlere Abweichung zu den Punkten, Anzahl der auf der Linie liegenden Punkte, Gesamtlänge und Azimut verwaltet werden. Die einzelnen Punkte welche zu einer Kultlinie gehören sind als Punktobjekt zu speichern. Zu den Punkten, müssen mindestens folgende Attribute verwaltet werden. Die Lagekoordinatengenauigkeit des Punktes sowie der Abstand zur dazugehörenden Kultlinie. Weiter soll die Beziehung und der Abstand zum nachfolgenden Punkt auf derselben Kultlinie abbildbar sein. Zusätzlich soll eine Möglichkeit geschaffen werden,

damit sämtliche Sachdaten, welche das Punktobjekt in den Grundlagedaten besitzt, referenziert werden können.

2.2 Anforderungen an die GIS Funktionalität

(1) Allgemeine Funktionen für die Darstellung von räumlichen Daten

(2) Einbindung und Darstellung von beliebigen Raster- und Vektordatenbeständen. Insbesondere soll die Möglichkeit bestehen Landeskartenausschnitte als Hintergrundinformation effizient anzuzeigen.

(3) Möglichkeit zur Durchführung einfacher Datenanalysen und zur Bearbeitung der Grundlagedaten.

(4) Georeferenzierung von Rasterdaten und anschliessender Möglichkeit zur Digitalisierung von Geometrieobjekten. Durch diese Funktion können noch nicht georefernzierte Karten (z.B. Siegfriedkarte) referenziert werden und interessierende Punkte digitalisiert werden.

(5) Funktion für die automatische Detektion von Kultlinien aus einem beliebig grossen Punkthaufen. Dabei definiert eine Line eine Gerade auf welcher mindestens drei Punkte in einer vorgegebenen Toleranz liegen. Aus den detektierten Punkten einer Linie soll anschliessend mit der Methode der kleinsten Quadrate eine Regressionsgerade bestimmt werden. Dabei soll jeder Inputpunkt in der Ausgleichung mit einer Lagegenauigkeit modelliert werden können. Als Toleranzkriterium soll ein distanzunabhängiger- und abhängiger Anteil definierbar sein. Zusätzlich soll eine maximale Distanz zum nachfolgenden Punkt auf einer Linie bestimmt werden können. Die resultierenden Geraden und dazugehörenden Punkte sollen nach den Anforderungen des Datenschemas gespeichert werden. Als weiteres Feature dieser Funktion sollte der Benutzer einen Punkt auswählen können, von welchem alle Kultlinien gesucht werden, die diesen Punkt enthalten.

(6) Damit eine gefundene Kultlinie auf deren mögliche Bedeutung hin untersucht werden kann, ist eine Funktion wünschenswert, welche das Azimut der Kultlinie auf eine allfällige Solstizialrichtung überprüft. Die zu prüfende Richtung ist immer auf den Auf- und Untergang zu beziehen. Dabei sind die zu prüfenden Richtungen um den Einfluss der Topographie (Horizont), von einem Punkt aus gesehen, zu korrigieren. Die Funktion soll mindestens Sommersonnenwende, Wintersonnenwende, Tag- und Nachtgleiche und kleines und grosses Mondextrem unterstützen. In der Berechnung sollen geographische Azimute verwendet werden.

(7) Eine bestehende Kultlinie soll editierbar bleiben. Das Hinzufügen, sowie das Löschen einzelner Punkte ist zu unterstützen. Dabei muss die Regressionsgerade neu bestimmt werden, und die betroffenen Attribute des Datenschemas sind nachzuführen.

(8) Bei der Bestimmung der Regressionsgerade, muss die Möglichkeit bestehen, die Punkte als Festpunkte oder Massenpunkte in die Ausgleichung einzuführen. Dabei üben Massenpunkte keinen Einfluss auf die Parameter der Ausgleichsgerade aus.

(9) Aus einem Höhenmodell und einer bestehenden Kultlinie muss ein Höhenprofil abgeleitet werden können.

(10) Bestimmung der orthogonalen Komponenten Abszisse und Ordinate eines Punktes bezüglich einer Kultlinie.

3. Aufbau GIS- Applikation

Wie aus den Anforderungen zu entnehmen ist, soll die Forschung früher Raumstrukturen in der Schweiz mit Hilfe von Informatikmitteln bestmöglich unterstützt werden. Da dieses Thema stark durch die räumliche Komponente beeinflusst ist, ist es naheliegend für die Forschung ein Geographisches Informationssystem (GIS) mit angepassten Erweiterungen einzusetzen. Auf diese Weise lassen sich effizient die Grundlagedaten sowie die ermittelten Resultate verwalten. Durch die zur Verfügung stehende Grundfunktionalität eines GIS und den Applikationsspezifischen Erweiterungen, lassen sich auf einfache Weise verschiedenste Analysen und Auswertungen durchführen.

Eine GIS- Applikation oder ein GIS- Projekt kann durch verschiedenste Softwarekomponenten realisiert werden. Dieses Kapitel soll einen Überblick vermitteln über das gewählte GIS für unser Projekt und soll den Aufbau der einzelnen eingesetzten GIS- Komponenten darstellen.

3.1 GIS Systemevaluation

Für unsere GIS- Applikation mussten zuerst verschiedene GIS- Varianten untersucht und mit den durch das Projekt definierten Anforderungen beurteilt werden. Dazu wurden vier verschiedene mögliche GIS Software- Architekturen aufbauend auf ESRI Komponenten für unsere Applikation geprüft. Für die ESRI Produktefamilie entschieden wir uns, da diese weltweit eingesetzt wird und ESRI derzeit der Marktführer für Geographische Informationssystem ist.

3.1.1 Variante 1: Funktionserweiterung mit Java- Applikation

Die Idee dieser Variante ist, dass die benötigten zusätzlichen Funktionen in einer unabhängigen Java- Applikation umgesetzt werden. Die benötigten Grundlagedaten werden hierbei von der Java- Applikation aus einem ESRI- Shapefile gelesen und das Resultat anschliessen wieder in ein Shapefile geschrieben. Die Grundlagedaten und die daraus generierten Resultatdaten könnten von jedem GIS betrachtet oder analysiert werden, welche das ESRI- Shapefile Format unterstützen. So könnte als GIS Desktop Programm der kostenlose ArcExplorer von ESRI eingesetzt werden.

Die Vorteile dieser Variante liegen eindeutig bei der Unabhängigkeit von der Betriebssystemplattform und der hierbei zusätzlich benötigten GIS- Software. Weiter kann das von der Java- Applikation verwendete ESRI- Shapefile Format als

Quasistandard in der GIS- Welt betrachtet werden. Nachteil dieser Variante ist, dass die Funktionserweiterung nicht direkt auf einer bestehenden GIS- Software aufsetzt, wodurch die Nutzung sehr umständlich wird.

3.1.2 Variante 2: ESRI MapObjects – Java Edition

MapObjects ist eine Java basierte API[1], welche leistungsstarke Komponenten zur Anzeige, Abfrage, Analyse und Organisation von GIS Daten zur Verfügung stellt. Mit Hilfe dieser Klassenbibliothek, kann auf einfache Weise ein eigenes GIS- System mit den benötigten Grundfunktionen und spezifischen Erweiterungen aufgebaut werden.
Mit dieser Variante könnte eine genau nach den spezifizierten Anforderungen entsprechende GIS- Applikation entwickelt werden. Dies hat den Vorteil, dass keine überflüssigen Funktionen zur Verfügung stehen und dadurch die Handhabung für den Benutzer optimal ist. Da man aber bei dieser Variante die gesamte Benutzeroberfläche und Grundfunktionen aus den bestehenden Komponenten der API zusammenbauen muss, ist der Entwicklungsaufwand für die kurze Zeit des Vertiefungsblocks zu hoch.

3.1.3 Variante 3: ESRI ArcView 3.x

Bei dieser Variante wird ESRI ArcView 3.x als GIS- Software eingesetzt. Die zusätzlich benötigte Funktionalität für unsere GIS- Applikation müsste mit der ArcView eigenen Scriptsprache Avenue implementiert werden.
Vorteile dieser Variante ist, dass die gesamte Funktionalität von ArcView zur Verfügung steht und nur projektspezifische Funktionen entwickelt werden müssten. Nachteil dieser Variante ist, dass ArcView 3.x durch ArcGIS 8 abgelöst wird und die Scriptsprache Avenue in der neuen Version nicht mehr unterstützt wird.

3.1.4 Variante 4: ESRI ArcGIS 8

ArcGIS 8 ist die neue GIS Produkte- Familie der Firma ESRI. Dies Produkte- Familie besteht aus verschiedensten Komponenten, welche nach den Anforderungen des GIS- Projektes nach belieben kombiniert werden können. Für spezifische Funktionserweiterungen steht dem Entwickler das ArcObjects Objektmodell bestehend aus COM- Objekten zur Verfügung. Das Objektmodell deckt sämtliche Funktionen der ArcGIS- Produkte Familie ab. Die COM[2]- Objekte werden von sämtlichen Windows Programmiersprachen wie Visual C++, Visual Basic, .Net oder

[1] API: Applikation Programming Interface (siehe Kapitel 4.1.1)
[2] COM: Component Object Model (siehe Kapitel 4.1.2)

Delphi unterstützt. Zusätzlich wird die Scriptsprache VBA (Visual Basic for Application) unterstützt.

Mit dieser Variante kann die gesamte bestehende Funktionalität der ArcGIS 8 Produkte Palette verwendet werden. Allfällige Zusatzentwicklungen können mit verschiedenen Programmiersprachen realisiert werden. Mit dem ArcObject COM-Objektmodell können beliebig komplexe Funktionen in eine ArcGIS- Komponente implementiert werden. Nachteil dieser Variante ist, dass sie das kostspielige Softwarepaket ArcGIS voraussetzt.

3.1.5 GIS Software- Architektur Variantenwahl

Aus folgenden Gründen realisieren wir unsere GIS- Applikation mit Variante 4. Die Variante 1 und 2 setzten einen hohen Entwicklungsaufwand voraus, da bei beiden Varianten die GIS Grundfunktionalität nur in beschränktem Umfang zur Verfügung steht. Mit den Varianten 3 und 4 müssen lediglich die applikationsspezifischen Erweiterungen implementiert werden. Den Entscheid zwischen Variante 3 und 4 begründen wir mit folgenden Argumenten:

Grundsätzlich sollte eine Neuentwicklung immer auf der neusten Produktversion aufbauen, damit möglichst aktuelle Informatikkomponenten genutzt werden können.

Da ArcObjects auf dem bewährten Windows COM- Objektmodell aufbaut, hat der Entwickler die Freiheit die Programmiersprache zu wählen. Zudem garantiert das in Windows verbreitete COM- Objektmodell, dass die implementierten Funktionen auch in Zukunft verwendbar bleiben und in anderen Windows basierenden GIS allenfalls eingesetzt werden könnten. (z.B Geomedia)

3.2 Architektur GIS- Applikation "Frühe Vermessungen"

3.2.1 Architektur ArcGIS

Für unsere GIS Applikation "Frühe Vermessungen" verwenden wir ESRI ArcGIS 8.3 als GIS- Software. ArcGIS beinhaltet folgende drei Desktop Anwendungen:

- **ArcMap:** ArcMap ist die zentrale Anwendung innerhalb ArcGIS Desktop. Mit ArcMap können sämtliche interaktiven Arbeiten mit Karten, wie Datenerfassung, Analyse oder Datenausgabe durchgeführt werden.
- **ArcCatalog:** ArcCatalog unterstützt den Nutzer bei der Verwaltung sämtlicher Geodatenquellen.
- **ArcToolbox:** In ArcToolbox können sämtliche Funktionen angezeigt werden, welche dem System ArcGIS zur Verfügung stehen. Diese möglichen Funktionen hängen von den installierten Klientenkomponenten und den Erweiterungen ab.

Auf ArcGIS Desktop bauen nun funktional abgestufte Klientenanwendungen auf

- **ArcView:** Stellt GIS Basisfunktionalität zur Verfügung, wie Mapping, Visualisierung, Datenausgabe, Datenanalyse und Datenerfassung. Für einen grossteil der GIS- Applikationen genügt diese Klientenkomponente.
- **ArcEditor:** Erweitert ArcView mit der vollen Unterstützung zur Datenmodellierung, Mehrbenutzerfähigkeit und Topologiebildung.
- **ArcInfo:** Dieser Klient beinhaltet die gesamte Funktionalität, und wird für High-End Anwendungen eingesetzt.

Zu diesen drei funktional abgestuften Klientenanwendungen stellt ESRI weitere Funktionserweiterungspakete (Extensions) dem Nutzer zur Verfügung. Die GIS-Applikation "Frühe Vermessungen" ist auf diese Weise aufgesetzt, dass lediglich die Klientenkomponente ArcView benötigt wird. Mit den beiden Komponenten ArcEditor oder ArcInfo stehen dem Benutzer weitere hilfreiche Funktionen zur Verfügung, welche aber für die Umsetzung unserer Anforderungen nicht zwingend benötigt werden. Damit alle Anforderungen an unsere GIS- Applikation erfüllt werden können, ist eine spezifische Funktionserweiterung zu implementieren. Diese Funktionserweiterung soll in ArcMap integriert werden und wird nachfolgend als "Limes" bezeichnet.

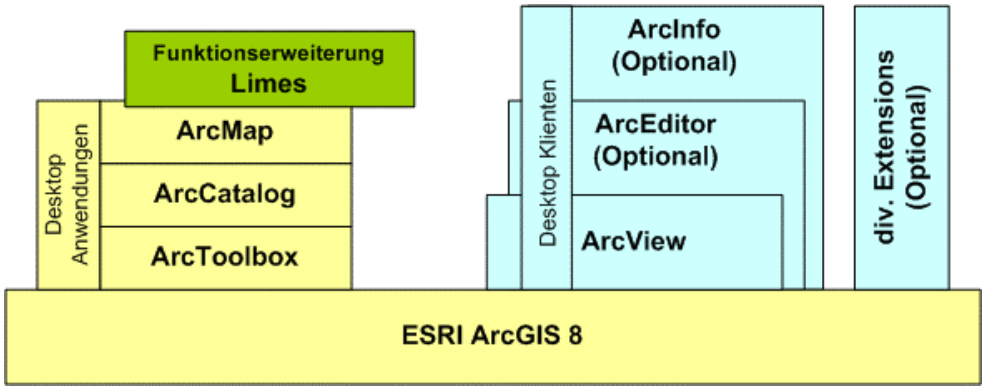


Abbildung 3-1: Komponenten Architektur ArcGIS 8

3.2.2 Aufbau GIS- Applikation "Frühe Vermessungen"

Folgende Abbildung zeigt den Aufbau der GIS- Applikation. Als Software wird ArcGIS 8.3 mit ArcView verwendet. Dabei wird die bestehende ArcView Funktionalität durch die projektspezifische Funktionserweiterung "Limes" ergänzt. Details zu den zusätzlich implementierten Funktionen werden im Kapitel 4 dokumentiert. Beim Konzept der Datenhaltung wird zwischen applikationsunabhängigen und applikationsspezifischen Datenquellen unterschieden. Die applikationsspezifische Datenquelle enthält ein festgelegtes Datenschema, welches von den zusätzlichen Funktionen zwingend benötigt wird. Für unsere Applikation setzen wir für diese Datenquelle eine relationale Microsoft Access Datenbank (Limes.mdb) ein. Grundsätzlich werden in dieser Datenquelle die Resultate der Auswertungen und Analysen gespeichert. Das verwendete Datenschema wird im folgenden Kapitel 3.3 näher beschrieben. Applikationsunabhängige Datenquellen sind hingegen Datenbestände, welche als Grundlagedaten für die Auswertung und Analyse dienen oder als zusätzliche Daten die Interpretation der Resultate unterstützen. Die zusätzlichen Funktionen können nun diese Datenquellen für Auswertungen als Inputdaten verwenden. Grundsätzlich können alle von ArcGIS unterstützen Datenquellen eingesetzt werden wie z. B. Access Datenbank, Esri Shapefile oder Shapefile Coverages.

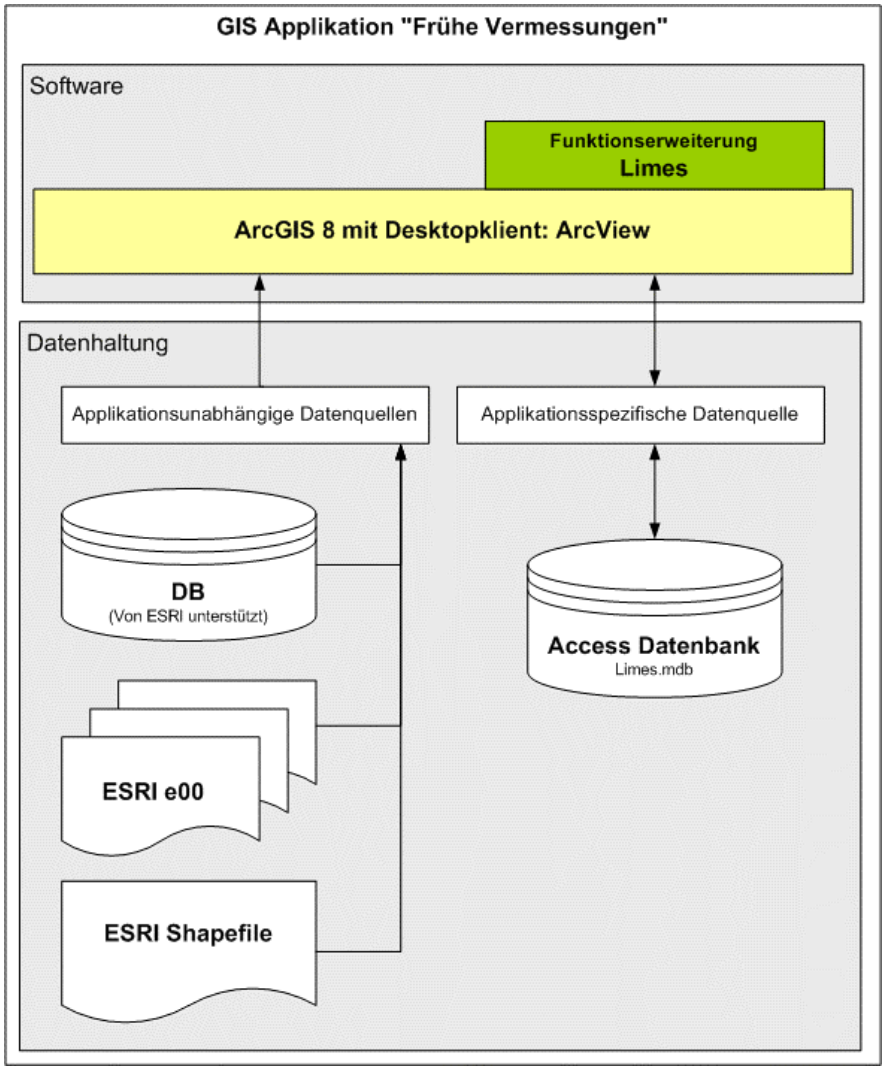


Abbildung 3-2: Aufbau GIS- Applikation

3.3 Datenschema

Die Applikation basiert auf einem festgelegten Datenschema, wie bereits in Kapitel 3.2.2 erklärt. Dieses Datenschema wird in der applikationsspezifischen Datenquelle (Access Datenbank limes.mdb) zur Verfügung gestellt und enthält im derzeitigen Entwicklungsstand einen Topic Polarsysteme. Das detaillierte Datenschema ist in der folgenden Abbildung dargestellt. Die Datentypen beziehen sich auf Microsoft Access und die kursive Schreibweise definiert eine optionale Entität oder optionales Attribut. Nachfolgend sollen lediglich die wesentlichen Details des Datenschemas erklärt werden.

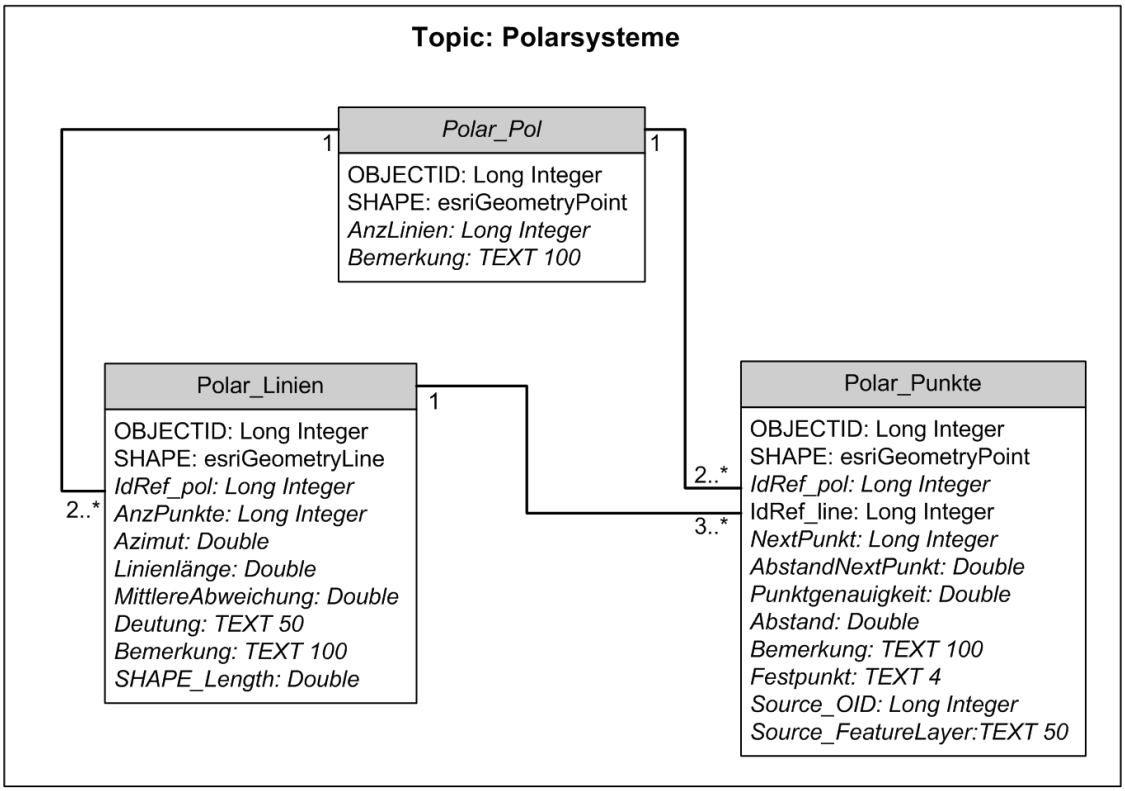


Abbildung 3-3: ERM- Diagramm Topic Polarsysteme

Das Attribut *OBJECTID* definiert den eindeutigen Identifikator der einzelnen Relationen. Der Topic Polarsysteme enthält die drei Entitäten *Polar_Pol, Polar_Linien* und *Polar_Punkte*, wobei die Entität *Polar_Pol* optional ist. Die Entität *Polar_Linien* beschreibt Linien, welche durch mindestens drei Punkte gebildet werden. Durch eine Fremdschlüsselbeziehung lassen sich die darauf liegenden Punkte in der Entität *Polar_Punkte* referenzieren. Die in dieser Tabelle gespeicherten Objekte (Linien), werden mittels eines Ausgleichs aus mindestens drei Punkten erzeugt. Somit beschreibt das Attribut *MittlereAbweichung* das gewogene Mittel aller zur Gerade normalen Punktabstände. Im Attribut *AnzPunkte* wird die Anzahl Punkte gespeichert, aus wie vielen Punkten eine Gerade gebildet wird. Mit dem Attribut *Deutung* erhält der Anwender die Möglichkeit zu einer Line mit speziellem Azimut dessen Bedeutung zu speichern. Die Entität *Polar_Punkte* beschreibt nun einen Einzelpunkt, der zu einer Gerade gehört. Im Attribut *IdRef_line* wird die Referenz auf ein Objekt der Relation Polar_Linien abgebildet. Das Attribut *NextPunkt* verweist auf den jeweils nächsten Punkt auf derselben Gerade. Dabei wird im letzten Punkt der Gerade der Wert 0 gespeichert. Zu diesem Attribut gibt es das Attribut *AbstandNextPunkt*, welches die Distanz auf der Geraden vom aktuellen zum nächsten Punkt definiert. Anhand des Attributs *Punktgenauigkeit* lässt sich einen mittleren Fehler der Lagekoordinaten modellieren, welcher bei der Bestimmung der Ausgleichsgerade berücksichtigt wird.

Das Attribut *Abstand* speichert den Abstand vom Punkt normal zur Ausgleichsgerade. Mit dem Attribut *Festpunkt* wird festgelegt, ob dieser Punkt als Fest- oder Massenpunkt in die Ausgleichung eingeführt wurde. Die beiden Attribute *Source_OID* und *Source_Layer* verweisen auf das Inputpunktobjekt im ArcMap Projekt. Die untenstehende Abbildung soll die wichtigsten Elemente des Datenschemas anhand eines einfachen Beispiels illustrieren.

Polar_Punkte			
OBJECTID	IdRef_line	NextPunkt	AbstandNextPunkt
1022	152	1023	1250
1023	152	1025	1300
1025	152	1024	1350
1024	152	1026	1400
1026	152	0	0

Polar_Linien
OBJECTID
152

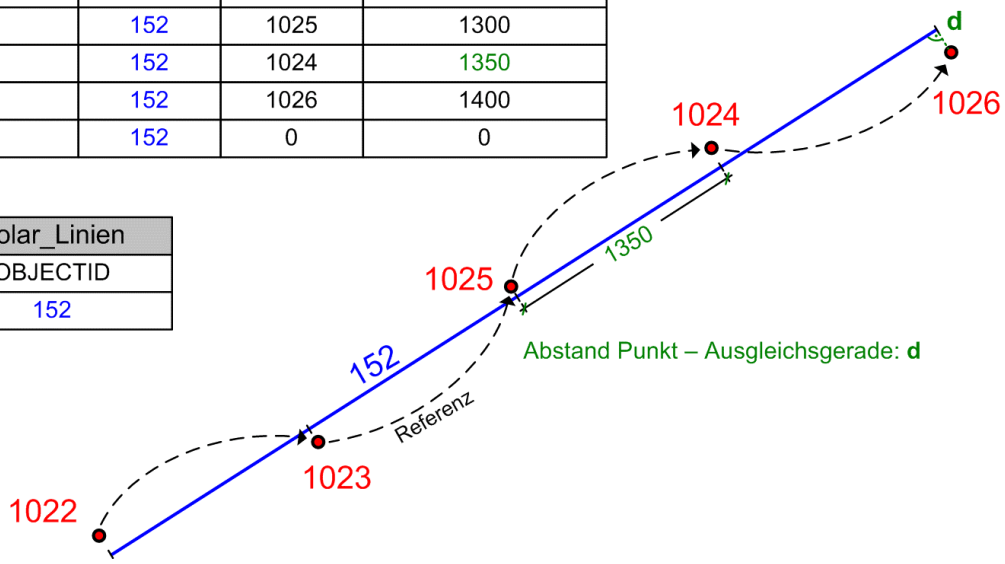


Abbildung 3-4: Bsp. Speicherung Kultlinie im Datenschema

4. Softwareentwicklung

Damit sämtliche Anforderungen an die GIS- Applikation umgesetzt werden können, muss die eingesetzte GIS- Software mit zusätzlichen Funktionen erweitert werden. Für solche Zwecke stellt ESRI die Klassenbibliothek ArcObjects zur Verfügung. Die Klassenbibliothek ArcObjects enthält sämtliche Klassen aus der die einzelnen ArcGIS Softwareprodukte aufgebaut sind. Mit ArcObjects lässt sich somit jede ArcGIS-Komponente nach eigenen Anforderungen anpassen und beliebig erweitern. Die Klassenbibliothek ArcObjects steht dem Entwickler als Windows COM- Komponente zur Verfügung. Als erstes folgt in diesem Kapitel eine grundsätzliche Beschreibung der eingesetzten Entwicklungskomponenten und deren Funktionsweise. Anschliessend soll der Aufbau und die Zusammenhänge der von uns implementierten Funktionserweiterung Limes aufgezeigt werden. Zuletzt werden die beiden in der Funktionserweiterung Limes eingesetzten Algorithmen, automatische Liniendetektion und Bestimmung einer Ausgleichsgerade, detailliert beschrieben.

4.1 Eingesetzte Entwicklungskomponenten und Tools

4.1.1 Programmbibliotheken

Eine Programmbibliothek ist eine Sammlung von Sourcecode für zusammengehörende Aufgaben. Programmbibliotheken sind im Unterschied zu ausführbaren Programmen keine eigenständige Programme, sondern Pakete welche Programmen als fertige Softwarekomponenten zur Verfügung stehen. Zusätzlich wird zwischen statischen und dynamischen Programmbibliotheken unterschieden. Bei den statischen Programmbibliotheken wird die kompilierte Softwarekomponente vor dem Ausführen des Anwendungsprogramms mit einem Link verbunden. Hingegen werden dynamische Programmbibliotheken erst bei Bedarf in den Arbeitsspeicher geladen und mit dem ausführbaren Anwendungsprogramm verbunden. Damit müssen Programmbibliotheken, welche von mehreren Anwendungen benutzt werden nur einmal im Speicher gehalten werden. Es wird zwischen folgenden Typen unterschieden:

Dynamic Link Library (DLL)

Im Betriebsystem Windows versteht man unter einer DLL eine dynamische Programmbibliothek, welche von Anwendungen zur Laufzeit genutzt werden kann.

Application Programming Interface (API)

Eine API ist eine Anwendungsprogrammierschnittstelle, mit welcher anderen Anwendungen standardisierte Softwarewerkzeuge zur Verfügung gestellt werden. Zweck einer API ist, durch das Angebot von vordefinierten Programmbausteinen die Programmierung neuer Anwendungen zu vereinfachen.

ActiveX

ActiveX ist die Bezeichnung für ein Softwarekomponentenmodell von Microsoft. Diese Komponenten entsprechen dem proprietären COM- Standard von Microsoft. So können DLL' s oder auch ausführbare EXE Anwendungen als ActiveX- Komponenten zur Verfügung gestellt werden.

4.1.2 Windows COM- Komponentenkonzept

Das "Component Object Model" (Komponenten-Objekt-Modell) ist ein Verfahren für den Datenaustausch zwischen verteilten Objekten in einem Objekt-orientierten Computer-Netz. Die proprietäre Spezifikation von der Firma Microsoft beschreibt Softwarekomponenten für neu zu schreibende oder bereits bestehende Programme. COM- Komponenten sind für den Einsatz unter den Windows- Betriebssystemen konzipiert. Sie können in mehreren Programmiersprachen wie z.B. Visual C++, Visual Basic, .Net oder Delphi geschrieben werden. Das "Component Object Model" definiert eine Software-Schnittstelle, durch die Software- Komponenten (auch als 'Objekte' bezeichnet) Daten miteinander austauschen. Sie ermöglicht es beliebigen Komponenten, miteinander zu kommunizieren, ungeachtet des Computer- Typs, auf dem sie ausgeführt werden, welches Windows- Betriebssystem auf den Rechnern installiert ist und in welcher Programmiersprache die Komponenten geschrieben wurden. Dank der als Durchschaubarkeit des Speicherortes bezeichneten Eigenschaft von COM spielt es für den Programmierer, der eine Komponenten schreibt, keine Rolle, ob sich die übrigen Komponenten zum Beispiel in einer "Dynamic Link Library" (DLL), in einer lokalen EXE- Datei oder auf einem anderen Netzwerk-Rechner befinden. So braucht der Programmierer auch keine Veränderung an seiner Komponente vorzunehmen, wenn andere Komponenten an einen anderen Ort im Computer-Netz verschoben werden. COM- Komponenten können während der Laufzeit entfernt werden, ohne dass dazu das Programm erneut kompiliert werden muss.

4.1.3 ESRI ArcObjects Klassenbibliothek

ArcObjects bildet die Basis der neuen ArcGIS 8 Generation und dient als Entwicklungsplattform für diese Produkte. Sämtliche Produkte wie ArcMap, ArcScene, ArcCataloge oder ArcView usw. bauen auf dieser technologischen Grundlage auf. Mit ArcObjects können die bestehenden Applikation verändert oder erweitert werden. ArcObjects basiert auf der Windows COM- Architektur, und stellt dem Entwickler dynamische Programmbibliotheken zur Verfügung. Das ArcObjects Objektmodell besteht aus mehreren Teilmodellen, wobei die einzelnen Teilmodelle bestimmte grundlegende Funktionen zur Verfügung stellen. Die vollständige Objektmodell Dokumentation ist auf der ESRI Homepage verfügbar.

4.1.4 Visual Basic 6.0

Für die Entwicklung der zusätzlichen Funktionen benutzen wir die Programmiersprache Visual Basic 6.0. Wir wählten Visual Basic, weil diese Programmiersprache das COM- Komponentenkonzept unterstützt und die Sprache sehr schnell zu erlernen ist. Diese Sprache sollte es ermöglichen, möglichst viele Anforderungen während der kurzen Projektdauer des Vertiefungsblocks umzusetzen.
Visual Basic ist eine Programmiersprache von Microsoft. Sie basiert auf der Programmiersprache BASIC und erleichtert die Programmierung durch eine visuelle Programmierumgebung. Visual Basic 6.0 ist nur eingeschränkt objektorientiert. So kann man Objekte aus Klassen erstellen und benutzen und Polymorphie über das Schnittstellenkonzept umsetzen. Jedoch wird die Vererbung und das Konzept der Erweiterung nicht unterstützt.

4.2 Architektur Funktionserweiterung "Limes"

4.2.1 Komponentenkonzept

Die gewählte Softwarearchitektur der zusätzlichen Funktionen basiert auf einem Komponentenkonzept. Die einzelnen Komponenten sind so gewählt, dass diese eine klar abgegrenzte Funktionalität zur Verfügung stellen. Die einzelnen Komponenten sind als dynamische Programmbibliotheken (DLL) ausgestaltet. Die folgende Abbildung zeigt die einzelnen Komponenten und deren grundlegende Funktionalität.

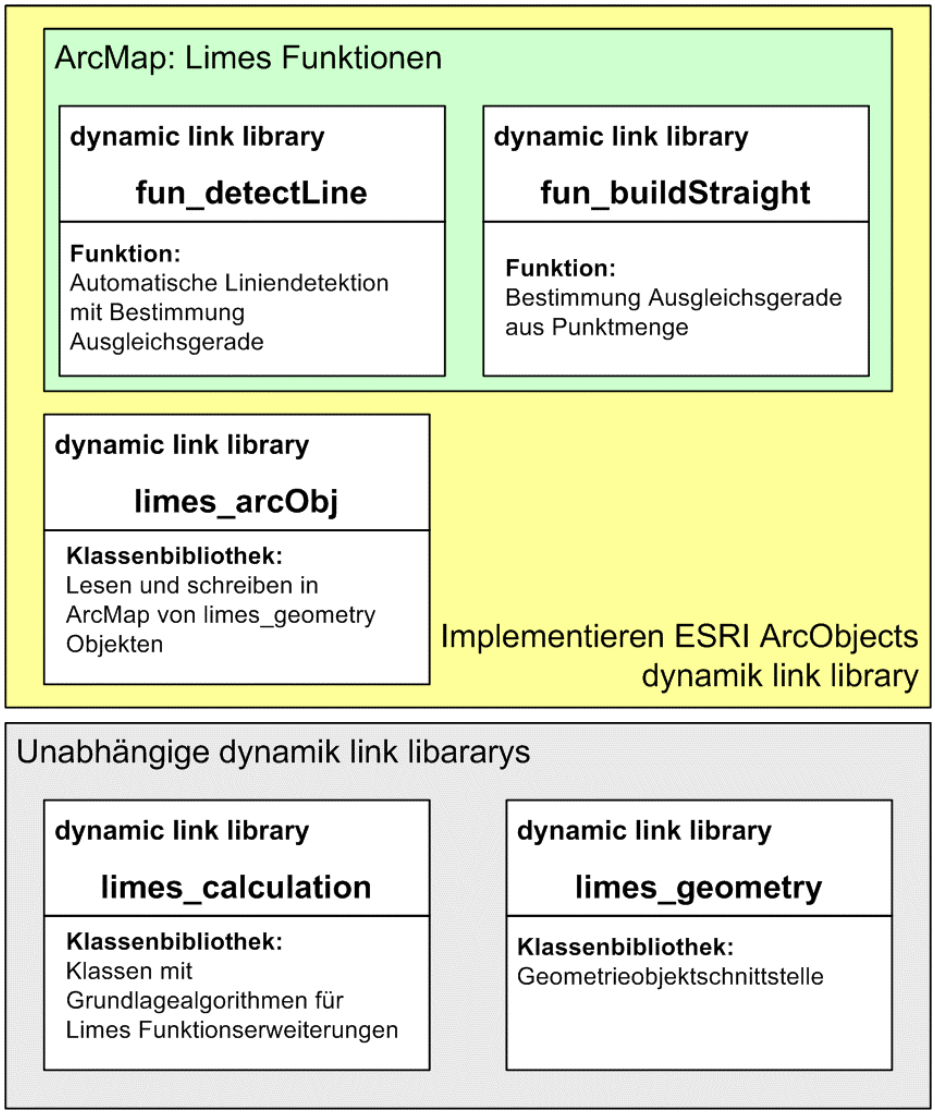


Abbildung 4-1: Dynamische Programmbibliotheken Limes

Die einzelnen Komponenten enthalten nun verschiedene Klassen, welche die Funktionalität über deren Methoden zur Verfügung stellen. Die einzelnen Komponenten sind anhand von UML- Klassendiagrammen in der Beilage A detailliert dokumentiert.

Die beiden DLL' s ***fun_detectLine*** und ***fun_buildStraight*** sind Programmbibliotheken, welche in ArcMap eingebunden werden können. Dabei definiert jede Bibliothek eine zusätzliche ArcMap Funktion. Damit eine Funktion in ArcMap eingebunden werden kann, muss eine Klasse in dieser Bibliothek zwei Schnittstellen aus der ArcObjects Bibliothek implementieren. Zusätzlich enthalten diese Komponenten die Graphische Benutzeroberfläche der Funktion und die Steuerung des Funktionsablaufs.

Die als unabhängig bezeichneten DLL' s ***limes_calculation*** und ***limes_geometry*** beinhalten nun die eigentlichen Algorithmen der Limes Funktionserweiterung. Die

Komponente *limes_calculation* stellt nun folgende zwei grundlegenden Methoden zur Verfügung:

- Automatische Detektion von Linien aus einem beliebigen Punkthaufen. Dabei muss die Linie aus mindestens drei Punkten gebildet werden. Der implementierte Algorithmus wird in Kapitel 4.3 detailliert beschrieben.
- Bestimmung der Ausgleichsgerade aus einer Menge Punkte. Die Details dieser Methode sind in Kapitel 4.4 beschrieben.

Die Bibliothek *limes_geometry* enthält Klassen, welche Geometrietypen wie Punkt und Linie repräsentieren. Die Methoden von *limes_calculation* verwenden solche Objekte als Übergabe- und Rückgabewerte. Diese Architektur wurde gewählt, weil dadurch die Programmbibliothek *limes_calculation* nicht durch Datenstrukturen von ArcObjects beeinflusst wird. Wenn nun ein GIS- System das COM- Komponentenmodell unterstützt, können diese Programmbibliotheken auch von diesem genutzt werden.

Die DLL ***limes_arcObj*** ist das Bindeglied zwischen der ArcObjects und *limes_calcualation* Bibliothek. Sie beinhaltet die Funktionalität, mit welcher selektierte Geometrieobjekte (Punkte) in ArcMap gelesen werden und in Objekte der Komponente *limes_geometry* konvertiert werden. Weiter können Geometrieobjekte dieser Komponente in die Applikationsabhängige Datenquelle gespeichert werden. Dabei werden die Daten auf das in Kapitel 3.3 beschriebene Datenschema abgebildet.

4.2.2 Funktion automatische Liniendetektion

Diese Funktion detektiert aus einer beliebigen Punktmenge sämtliche Linien, auf welcher mindestens 3 Punkt in einer bestimmten Toleranz liegen. Anschliessend wird aus den detektierten Punkten einer Linie die ausgleichende Gerade bestimmt. Die Ausgleichsgerade und die dazugehörenden Punkte werden als Resultat dieser Funktion im applikationsabhängigen Datenschema gespeichert. Der detaillierte Funktionsablauf ist in einem UML- Sequenzdiagramm in Beilage B aufgezeigt.

4.2.3 Funktion Bestimmung Ausgleichsgerade aus Punktmenge

Mit dieser Funktion kann aus einer selektierten Punktmenge die Ausgleichsgerade bestimmt werden. Wiederum wird das Resultat im applikationsabhängigen Datenschema gespeichert. Der Funktionsablauf ist in der Beilage B mit Hilfe eines UML- Diagramms detailliert dokumentiert.

4.3 Algorithmus Liniendetektion

4.3.1 Problem

In einem Punkthaufen Punkte herauszufiltern, welche sich mit einer bestimmten Ungenauigkeit auf einer gemeinsamen Linie befinden.

4.3.2 Variantenstudium Liniendetektion

Eine Suche nach bestehenden Algorithmen ist erfolglos geblieben. Daraufhin wurden einige Ideen gesammelt, um Linien detektieren zu können. Drei verschiedene Ansätze wurden in Betracht gezogen, um das Problem lösen zu können.

Gerade finden durch Berechnung des Abstandes

Ein Standardproblem in der Informatik ist die Berechnung des Abstandes eines Punktes von einer Einzellinie. Diese Einzellinie wird durch zwei Punkte bestimmt. Der Abstand eines dritten Punktes ist dann ein direktes Mass, ob sich die drei Punkte auf einer Linie befinden.
Bei dieser Variante müssen alle Punkte mit allen anderen kombiniert werden. Mit einer grösseren Anzahl Punkte ergeben sich somit eine zu grosse Anzahl Berechnungen, weshalb diese Variante nicht weiter entwickelt wurde.

Drehendes Rechteck

Der hier als drehendes Rechteck bezeichnete Ansatz basiert auf der Standardabfrage, ob sich ein Element in einem bestimmten Rechteck befindet. Diese Abfrage ist einfach mit If- Bedingungen umzusetzen. Das Rechteck wird um einen Pol rotiert. Bei jeder Position werden alle Punkte innerhalb des Rechtecks zu einer Linie gezählt.

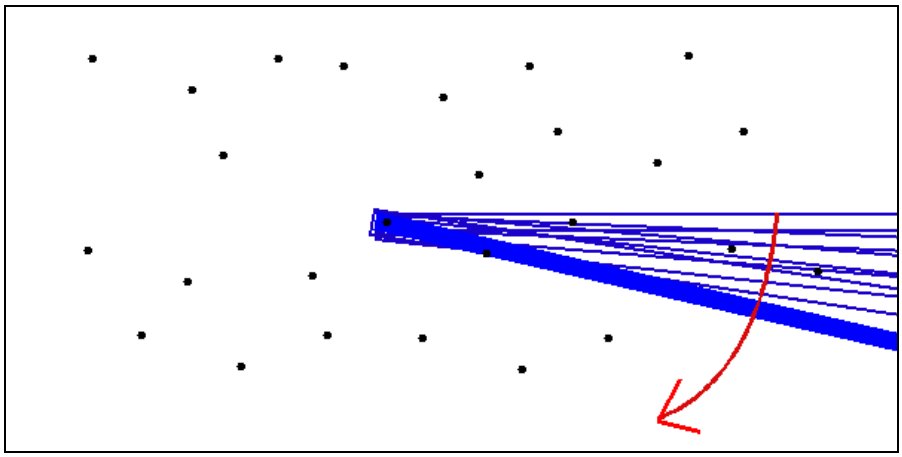


Abbildung 4-2: Variante drehendes Rechteck

Diese Variante wird als eine mögliche Lösung angesehen. Offen bleiben jedoch die folgenden Fragen: Wo soll mit der Rotation des Suchfensters begonnen werden und

in welchem Winkel soll sich das nächst folgende Suchfenster anschliessen? Was passiert, wenn Punkte in mehreren Suchfenstern gefunden werden? Diese Fragen konnte nicht schlüssig beantwortet werden.

Gerade finden durch Azimut

Verbindet man immer jeweils zwei Punkte durch eine Einzellinie und vergleicht deren Azimut, so sind Punkte dann in einer Linie, wenn die beiden Einzellinien das gleich Azimut aufweisen und jeweils der eine Endpunkt bei beiden Einzellinien vorkommt.

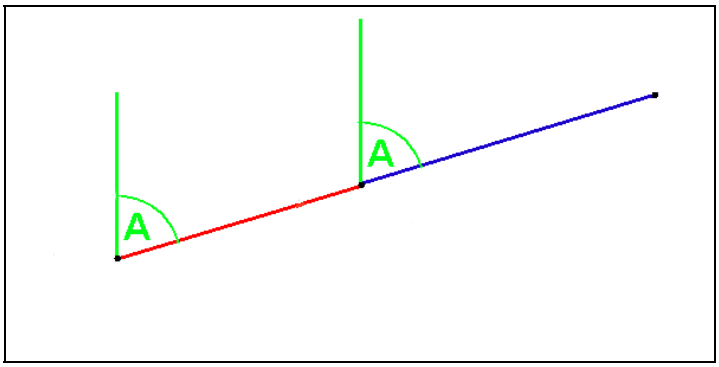


Abbildung 4-3: Variante Azimut

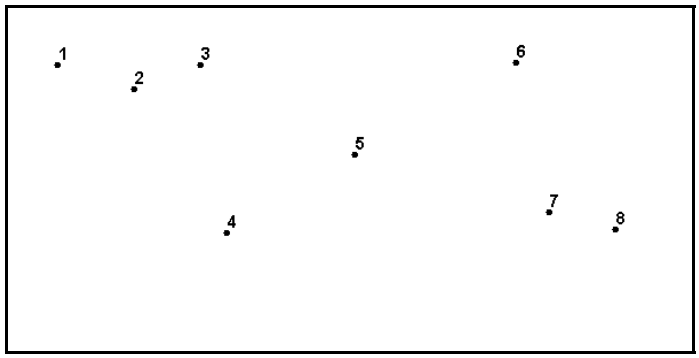
Diese Variante wurde gewählt, weil es dabei um einen einfachen Vergleich von zwei Azimuten handelt.

4.3.3 Funktionsweise des Algorithmus Liniendetektion

Als Einführung wird nachfolgend die Funktionsweise des entwickelten Algorithmus zur Liniendetektion anhand eines Beispiel aufgezeigt.

Sortieren

Die vom Benutzer ausgewählten Punkte und Eingabeparameter sind die Ausgangsdaten der Funktion Liniendetektion. In einem ersten Schritt werden alle Punkte aufsteigend nach ihrer Y- Koordinate sortiert und nummeriert. Im weiteren wird eine Tabelle erstellt um die Einzellinien speichern zu können. Die Tabelle enthält: eine Identifikationsnummer der Einzellinie (ID), Anfangspunkt der Einzellinie (A), Endpunkt der Einzellinie (E), Azimut, Distanz, Vorgänger (V) und Nachfolger (N) der Einzellinie.

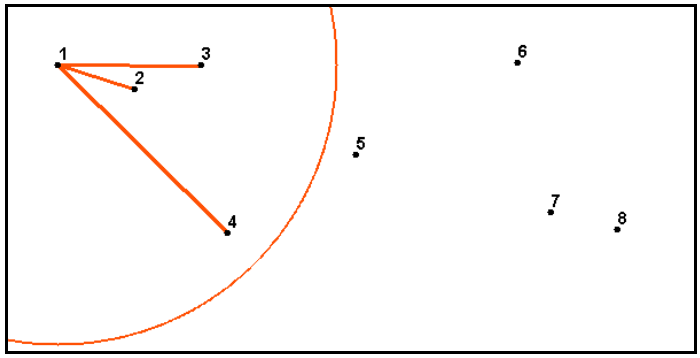


ID	A	E	Azi	Dist	V	N

Abbildung 4-4: Funktionsweise - Schritt 0

Einzellinien generieren

Es wird nun ausgehend vom Punkt 1 begonnen, alle Punkte innerhalb eines bestimmten Abstandes mit einer Einzellinie zu verbinden. Die Einzellinien werden in der Tabelle mit Anfangspunkt, Azimut, Distanz und Endpunkt gespeichert.

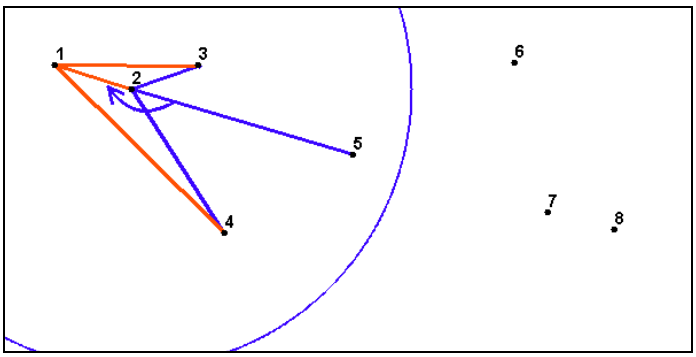


ID	A	E	Azi	Dist	V	N
1	1	2	110	50		
2	1	3	98	100		
3	1	4	130	150		

Abbildung 4-5: Funktionsweise - Schritt 1

Azimutvergleich

Sind auf dem ersten Punkt alle Verbindungen abgespeichert worden, wird der Ausgangspunkt auf Punkt 2 gewechselt. Im nachfolgenden wird dieser Ausgangspunkt als Standpunkt bezeichnet. Vom neuen Standpunkt aus werden wiederum alle Punkte innerhalb eines bestimmten Abstandes mit Einzellinien verbunden und in die Tabelle abgespeichert. Es werden dabei nur diejenigen Punkte verwendet, die östlicher vom Standpunkt liegen. Aufgrund der vorgängigen Sortierung aller Punkte, sind dies all jene Punkte mit einer höheren Identifikationsnummer. Bei jeder neu gespeicherten Einzellinie wird zudem geprüft, ob in der Tabelle schon eine Einzellinie mit gleichem Azimut vorhanden ist und bei welcher der Endpunkt dem aktuellen Standpunkt entspricht. Ist dies der Fall (wie in diesem Beispiel bei der Einzellinie 6), so werden die beiden Einzellinien durch die Vergabe der Attribute Vorgänger und Nachfolger miteinander verbunden.

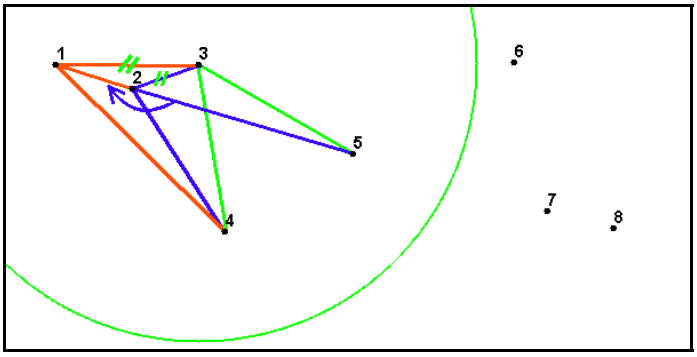


ID	A	E	Azi	Dist	V	N
1	1	2	110	50		6
2	1	3	98	100		
3	1	4	130	150		
4	2	3	30	50		
5	2	4	160	180		
6	2	5	110	130	1	

Abbildung 4-6: Funktionsweise - Schritt 2

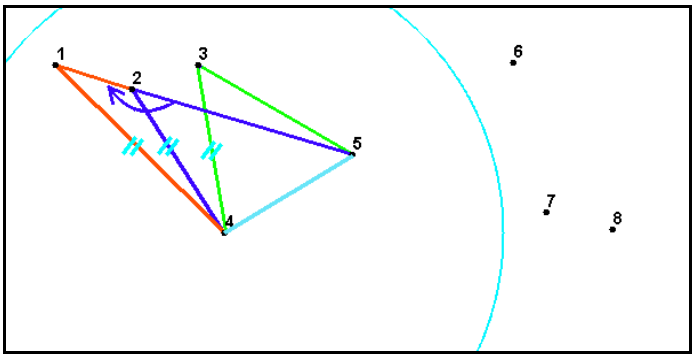
Einzellinien löschen

Dieser Vorgang der Einzellinien Generierung und Azimutvergleich wird wiederum bei allen weiteren Punkten wiederholt. Jeder Punkt wird der Reihe nach zum Standpunkt. Am Ende eines Standpunktes können sämtliche Einzellinien mit dem Standpunkt als Endpunkt gelöscht werden, welche keinen Nachfolger und Vorgänger besitzen.



ID	A	E	Azi	Dist	V	N
1	1	2	110	50		6
~~2~~	~~1~~	~~3~~	~~98~~	~~100~~		
3	1	4	130	150		
~~4~~	~~2~~	~~3~~	~~30~~	~~50~~		
5	2	4	160	180		
6	2	5	110	130	1	
7	3	4	170	200		
8	3	5	120	200		

Abbildung 4-7: Funktionsweise - Schritt 3



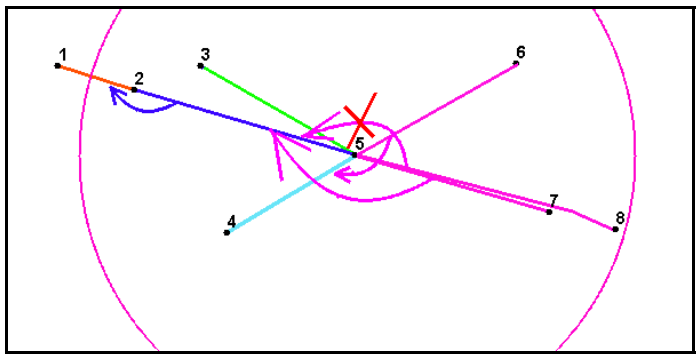
ID	A	E	Azi	Dist	V	N
1	1	2	110	50		6
~~3~~	~~1~~	~~4~~	~~130~~	~~150~~		
~~5~~	~~2~~	~~4~~	~~160~~	~~180~~		
6	2	5	110	130	1	
~~7~~	~~3~~	~~4~~	~~170~~	~~200~~		
8	3	5	120	200		
9	4	5	45	170		

Abbildung 4-8: Funktionsweise - Schritt 4

Mehrfache Nachfolger

Beim nächsten Standpunkt Nummer 5 werden die Beziehungen der Einzellinien 10 zur Einzellinie 9 vergeben. Bei der Einzellinie 6 wird nur die erste kürzere Einzellinie als Nachfolger zugelassen. Die Einzellinie 12 wird ignoriert, obschon das Azimut

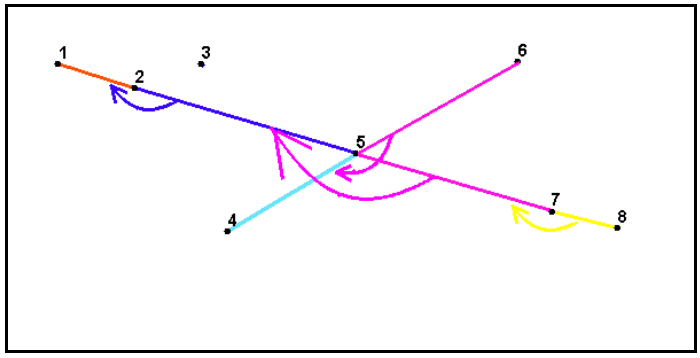
demjenigen der Einzellinie 6 entspricht. Dadurch sollen Doppellinien über die gleichen Punkte vermieden werden.



ID	A	E	Azi	Dist	V	N
1	1	2	110	50		6
6	2	5	110	130	1	
8	3	5	120	200		
9	4	5	45	170		10
10	5	6	45	190	9	
11	5	7	110	200	6	
~~12~~	~~5~~	~~8~~	~~110~~	~~240~~	~~6~~	

Abbildung 4-9: Funktionsweise - Schritt 5

Die im vorhergehenden Schritt ignorierte Einzellinie wird auf dem Standpunkt 7 auf den Punkt 8 mit dem Standardvorgehen verlängert.



ID	A	E	Azi	Dist	V	N
1	1	2	110	50		6
6	2	5	110	130	1	
8	3	5	120	200		
9	4	5	45	170		10
10	5	6	45	190	9	
11	5	7	110	200	6	13
13	7	8	110	50	11	

Abbildung 4-10: Funktionsweise - Schritt 6

Durch die gespeicherten Beziehungen der Vorgänger oder Nachfolger können die Einzellinien zu einer Linie über mehr Punkte zusammengehängt werden. Dieses Zusammenhängen wird weiter unten in einem weiteren Kapitel beschrieben. Das bis hier erläuterte Beispiel zeigt die Funktionsweise des Algorithmus fast vollständig auf. Zum vollen Verständnis der Funktionsweise des ganzen Algorithmus sind jedoch folgende zwei Erläuterungen noch notwendig.

Suchtoleranz

Der Umstand das nur Einzellinien mit Vorgänger und Nachfolger miteinander in Beziehung gebracht werden, welche das gleiche Azimut haben, wurde im Beispiel vereinfacht dargestellt. Natürlich werden Einzellinien miteinander verhängt, wenn ihre Azimute eine gewisse Toleranz nicht überschreiten. Die Toleranz wir von den beiden Eingabeparameter Querdistanz und Suchwinkel vorgeben. Für den Azimutvergleich

muss Querdistanz in einen Winkel umgerechnet werden. Da dieser Winkel distanzabhängig ist, erfolgt die Berechung für jede neue Einzellinie. Der Winkel der Querdistanz summiert mit dem Eingabeparameter Suchwinkel ergibt die Azimuttoleranz, innerhalb derer die Einzellinien zusammengehängt werden.

Diese Suchtoleranz lässt sich einfach darstellen als Annahmebereich (siehe nachfolgende Abbildung). Siehe nachfolgende Grafik. Befindet sich im violetten Bereich ein Punkt, so bilden die drei Punkte eine gemeinsame Line. Wobei die verwendete Ausgangsrichtung im nächsten Abschnitt näher erläutert wird.

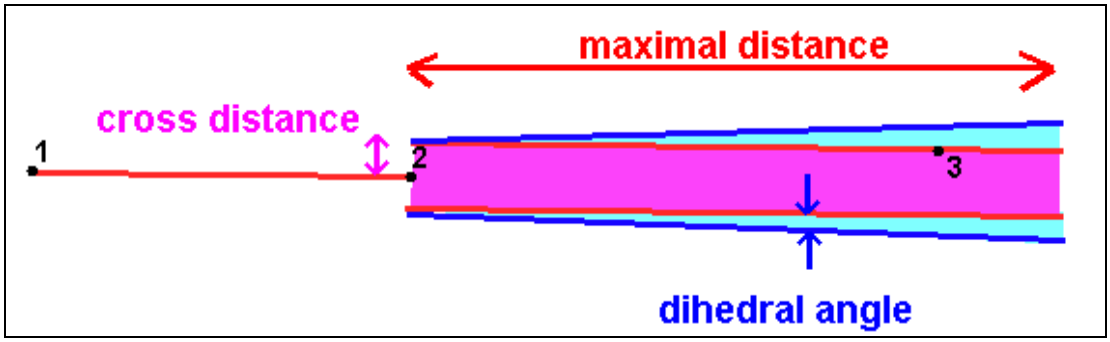


Abbildung 4-11: Suchbereich

Verwendete Ausgangsrichtung

Die Wahl der Ausgangsrichtung kann nur in dem Fall, wo bereits mehrere Einzellinien eine Linie bilden, verschieden gewählt werden. Eine Möglichkeit wäre, nur das Azimut der letzten Linie zu verwenden. Es wurde jedoch eine andere Variante gewählt, die alle vorangehenden Einzellinien mit einbezieht.

Die Ausgangsrichtung wird aus dem Standpunkt und dem ersten Punkt der bis zu diesem Zeitpunkt detektierten Linie berechnet. Für die weitere Suche dieser Linie wird somit nicht das Azimut der letzten Einzellinie verwendet, sondern das Azimut zwischen dem ersten Anfangspunkt der ersten Einzellinie und dem Endpunkt der letzten Einzellinie. Diese einfache Berechnung ist zu vergleichen mit einer gewichteten Mittelbildung der Azimute von allen Einzellinien einer Linie.

Mehrere Vorgänger

Wie die Möglichkeit besteht, dass eine Einzellinie mehrere Nachfolger hat, so existiert auch der Fall von mehreren Vorgängern. Wie bei der Subfunktion "Mehrere Nachfolger", so wird auch hier die Strategie verfolgt, möglichst nur eine Einzellinie mit einer Einzellinie zusammenzuhängen. Mehrere Vorgänger werden nur in jenem Fall zugelassen, bei welchem zwei Vorgänger der Einzellinien beide wiederum Vorgänger aufweisen. Weist nur eine der beiden Einzellinien der Vorgänger eine

Einzellinie auf, so wird sie mit dieser verbunden. Die andere Verbindung wird gelöscht. Im Fall bei welchen keine der Vorgänger einer gemeinsamen Einzellinie einen Vorgänger haben, wird jener Vorgänger gewählt, welcher die kleinere Azimutdifferenz aufweist.

1. Fall: Zwei Einzellinien (hellblau und dunkelblau) haben die gleiche Einzellinie (rot) als Nachfolger. Da die blauen Einzellinien beide keine Vorgänger besitzen wird hier die Beziehung der hellblauen Einzellinie mit der Roten vervollständigt. Die dunkelblaue Einzellinie wird gelöscht.

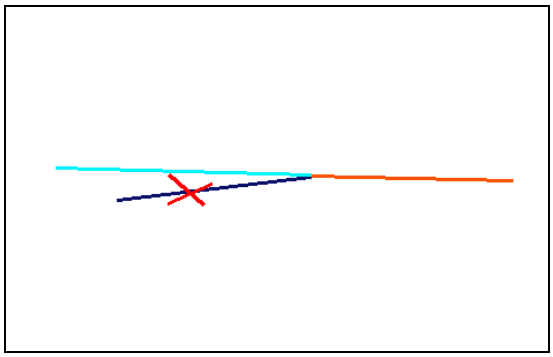


Abbildung 4-12: Mehrere Vorgänger 1

2. Fall: Eine der beiden Einzellinien (hellblau und dunkelblau) besitzt einen Vorgänger(grün). Die Einzellinie (dunkelblau) mit dem Vorgänger wird definitiv mit dem gemeinsamen Nachfolger verknüpft. Die hellblaue Einzellinie wird gelöscht, obwohl das Azimut besser übereinstimmt.

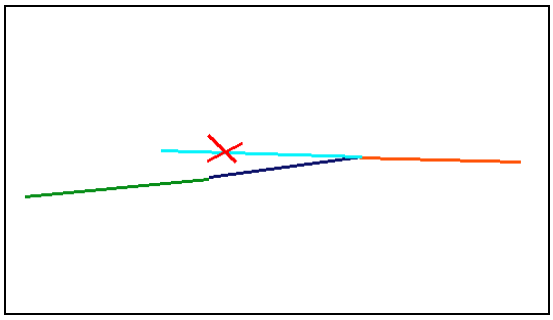


Abbildung 4-13: Mehrere Vorgänger 2

3. Fall: Beide Einzellinien (hellblau und dunkelblau) besitzen einen Vorgänger. Hier behalten beide Einzellinien die rote Einzellinie als Nachfolger.

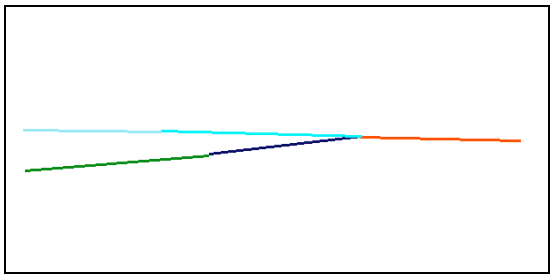


Abbildung 4-14: Mehrere Vorgänger 3

Einzellinien zusammenhängen

All die bis hier beschriebenen Prozesse bilden und verändern lediglich die Tabelle der Einzellinien. Der Aufbau und die Änderungen dieser Tabelle konnten im oben aufgeführten Beispiel verfolgt werden. Das Endresultat des Beispiels ist hier nochmals abgebildet.

ID	A	E	Azi	Dist	V	N
1	1	2	110	50		6
6	2	5	110	130	1	
8	3	5	120	200		
9	4	5	45	170		10
10	5	6	45	190	9	
11	5	7	110	200	6	13
13	7	8	110	50	11	

Abbildung 4-15: Tabelle Einzellinien

Für den nächsten Programmteil der Ausgleich müssen diese Einzellinien nun noch zusammengehängt werden. Dies erfolgt mit den gespeicherten Beziehungen der Nachfolger. Alle Punkte einer Linie werden in einer Punkttabelle gespeichert. Für jede weitere Linie wird dann eine neue weitere Tabelle mit Punkten erstellt. Jede Tabelle entspricht somit einer Linie. All diese Tabellen werden dann dem nächsten Programmteil der Ausgleichung übergeben.

Log File

All die Abläufe und Berechnungen des Algorithmus Liniendetektion werden in einem Log File dokumentiert. Es ist wird im gleichen Ordner des Projektfiles mit dem Namen Limes.log abgelegt. Dieses Log File wurde grundsätzlich für die Entwicklung benutzt. Im fertigen Programm kann es aber bei Problemen als wichtige Information herangezogen werden.

Funktion Liniendetektion als Ganzes

Im hier dargelegten Beispiel wurde die Funktionalität des Algorithmus aufgezeigt. Es wurde jedoch nicht darauf eingegangen, wie die einzelnen Abläufe zusammenhängen. Der gesamte Algorithmus ist in der Beilage C mit einem Nassi – Schneidermann Diagramm detailliert dokumentiert.

4.3.4 Idee der verwendeten Strategie

Nachdem die Funktionsweise des Liniendetektionsalgorithmus im vorangehenden Kapitel aufgezeigt wurde, wird in diesem Kapitel die Begründung der gewählten Implementierung diskutiert. Im weiteren wird ebenfalls auf die Entstehung des Algorithmus und auf einige Erfahrungen eingegangen.

Das grundlegende Konzept für den Algorithmus basiert auf folgender Überlegung. Das Ziel des Algorithmus ist es sich von West nach Ost über jeden Punkt im Punkthaufen zu bewegen und dabei den aktuellen Punkt an die bereits generierten Linien anzuhängen, wenn dieser sich im Annahmebereich befindet (Siehe Abb. 5-11).

Auf diese Weise stehen nach der Abarbeitung des westlichsten Punktes alle detektierten Linien zur Verfügung. Mit anderen Worten ausgedrückt, wird jeder Punkt nacheinander geprüft, in welchen möglichen Kultlinien dieser Punkt enthalten ist. Dieses Konzept hat den Vorteil, dass das Problem mit annähernd linearem Aufwand zur Anzahl der Inputpunkte gelöst werden kann. Dieser grundlegende Ansatz ist wie folgt umgesetzt.

Im Variantenstudium wurde bereits der erste Grundsatz, dass die Suche nach Linien über ein Azimutvergleich erfolgt, festgelegt. Für diesen Vergleich werden die Einzellinien nach vorangehendem System berechnet und abgespeichert. Da die Generierung der Linien immer von West nach Ost erfolgt, stimmen die Azimute von parallelen Einzellinien immer überein (keine Differenz in die Gegenrichtung →+/- 180°). Die Sortierung der Punkte und Abarbeitung in einer Richtung bietet noch einen weiteren Vorteil. Die Punkte sind räumlich indexiert (in einer Richtung), dass heisst nahe beieinander liegende Punkte haben ähnliche Nummern. Für die Generierung von Einzellinien müssen somit nicht alle Punkte des ganzen Datensatzes abgearbeitet werden, nach dem ersten Punkt welcher östlicher liegt als die Y-Koordinate des Standpunktes plus die Maximaldistanz kann die Schlaufe abgebrochen werden.

Im Verlauf der Programmierung zeigte sich zudem, dass alle Probleme auf dem Standpunkt zu lösen sind. Das heisst die Mehrdeutigkeit, mehrfache Vorgänger oder Nachfolger, muss sofort auf dem jeweiligen Standpunkt eindeutig gelöst werden. Eine zuvor eingeschlagene Philosophie, diese Verzweigungen erst am Schluss zu bereinigen, musste fallen gelassen werden. Es entstanden zu viele Möglichkeiten die jeweils einzeln hätten behandelt werden müssen.

4.4 Algorithmus Bestimmung Ausgleichsgerade

Mit diesem Algorithmus soll eine nach den kleinsten Quadraten ausgeglichene Gerade durch eine beliebige Punktmenge bestimmt werden. Dabei kann jeder Punkt mit einer unterschiedlichen Punktgenauigkeit in das System eingeführt werden. Weiter soll die Möglichkeit bestehen, die einzelnen Punkte als Festpunkte oder als Massenpunkte in der Ausgleichung zu berücksichtigen. Im Unterschied zu den Festpunkten, werden Massenpunkte nicht für die Bestimmung der Gerade verwendet. Zusätzlich zur Bestimmung der Ausgleichsgerade werden folgende Grössen des Datenschemas berechnet und in Objekten der Programmbibliothek *limes_geometry* gespeichert:

- Rechtwinkliger Abstand von der Gerade zu sämtlichen Punkten der Linie und mittlerer Punktabstand zur Ausgleichsgerade.

- Bestimmung der Reihenfolge nach aufsteigendem Rechtswert (Y Koordinate) der Punkte. Die sortierte Punktreihenfolge wird anschliessend benötigt, damit die Attributwerte *NextPunkt* und *NextPunktAbstand* in der Relation *Polar_Punkte* bestimmt werden können.
- Bestimmung des Azimuts der Gerade

Die folgenden Unterkapitel sollen die implementierte Lösungen für die vorgängig aufgezählten Problemstellungen aufzeigen und begründen. In Beilage D wird der sequentielle Ablauf des implementierten Algorithmus mit Hilfe eines Diagramms übersichtlich dargestellt.

4.4.1 Bestimmung Ausgleichsgerade

Da eine Punktkoordinate in X- und in Y- Richtung nicht fehlerfrei ist, kann nicht das einfache Regressionsmodell für die Gerade verwendet werden, bei welchem lediglich die Verbesserungen in Y- Richtung minimiert werden. Für unsere Anwendung soll somit das Quadrat des rechtwinkligen Abstands d_i von der ausgeglichenen Gerade zu den Festpunkten minimiert werden (siehe folgende Abbildung).

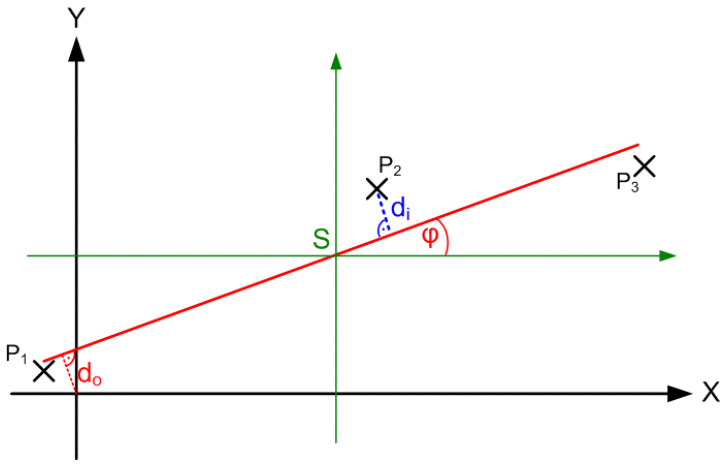


Abbildung 4-16: Skizze Modell Ausgleichsgerade

Der Abstand d_i normal zur Ausgleichsgeraden (rot) berechnet sich nach folgender Formel:

$$d_i = -x_i \sin\varphi + y_i \cos\varphi - d_o$$

Die Summe der gewichteten Abstandsquadrate ist zu minimieren,

$$\Omega = \underline{d}^T P \underline{d} \rightarrow \min$$

was auf die Auflösung des folgenden Gleichungssystems führt.

$$\frac{\partial\Omega}{\partial d_o} = 0 \quad (1) \text{ und } \frac{\partial\Omega}{\partial\varphi} = 0 \quad (2)$$

Die quadratische Gewichtsmatrix P wird aus den Punktgenauigkeiten der Festpunkte nach folgendem Schema gebildet:

$$P = \begin{bmatrix} 1/\sigma_1^2 & 0 & 0 & 0 \\ 0 & 1/\sigma_2^2 & 0 & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & 1/\sigma_n^2 \end{bmatrix}$$

Damit sich das Problem vereinfacht, werden die Punktkoordinaten auf die Schwerpunktkoordinaten X_S und Y_S des Punkthaufens bezogen (grünes Koordinatensystem).

Damit wird erreicht, dass sich die Gleichung (1) auf folgende einfache Form reduziert.

$\frac{\partial \Omega}{\partial d_o} = 2 d_o \underline{e}^T P \underline{e} = 0$ womit für den Abstand zum Nullpunkt folgt $d_o = 0$

Für die Gleichung (2) folgt:

$$\frac{\partial \Omega}{\partial \varphi} = \sin 2\varphi (\underline{x}^T P \underline{x} - \underline{y}^T P \underline{y}) - 2\cos 2\varphi (\underline{x}^T P \underline{y}) = 0$$

Wird diese Gleichung nach φ aufgelöst, erhält man

$$\varphi = \arctan\left(\frac{2\underline{x}^T P \underline{y}}{\underline{x}^T P x - \underline{y}^T P \underline{y}} \right) \Big/ 2$$

Mit dem Steigungswinkel φ kann die ausgeglichene Gerade in der Hessischen Normalform der Geradengleichung geschrieben werden.

$0 = ax + by + c$ mit $a = \tan\varphi$, $b = -1$ und $c = 0$

Damit eine eindeutige Lösung für φ erzielt werden kann, wird der Punkthaufen vor der Ausgleichung mittels Translation und Drehung auf diese Weise transformiert, dass dieser in Richtung positive X- Achse zu liegen kommt.

4.4.2 Berechnung Punktabstand und mittlere Abweichung

Der Abstand eines Punktes P_i von der ausgeglichenen Geraden wird mit folgender Formel berechnet:

$$d_i = \frac{\left| ax_p + by_p + c \right|}{\sqrt{a^2 + b^2}}$$

Für die Berechnung der mittleren Abweichung der Punkte, werden lediglich die Festpunkt verwendet und nach folgender Formel bestimmt:

$$mA = \frac{|d_i| * P_i}{spur(P)}$$

4.4.3 Generierung Punktreihenfolge

Die Reihenfolge der sich auf der Geraden befindenden Punkte, wird nach folgendem Ansatz bestimmt. Zuerst wird jeder Punkt normal auf die Gerade projiziert. Ausgehend von den projizierten Punktkoordinaten wird mit einem QuickSort Sortieralgorithmus nach aufsteigenden X- Werten sortiert. Anschliessend können die Abstände zum nächsten Punkt aus den projizierten Koordinaten berechnet werden. Das Linienobjekt, einer Gerade ist definiert durch den ersten und letzten Punkt der sortierten Punktreihenfolge. Aus diesen beiden Punkten wird das Azimut der Gerade bestimmt, wobei für die Berechnung des Azimuts der letzte Punkt als Zielpunkt verwendet wird. Damit die letzten beiden Berechnungen durchführbar sind, müssen die Punktkoordinaten vorgängig ins globale Koordinatensystem invers transformiert werden.

5. Umgesetzte Anforderungen und mögliche Erweiterungen

5.1 Implementierte Funktionalität

In der dem Technischen Bericht beigelegten Bedienungsanleitung sind die grundsätzlichen Bestandteile der GIS- Applikation "Frühe Vermessungen" kurz beschrieben. Die zusätzlichen Funktionen werden detailliert beschrieben und es wird aufgezeigt, wie diese sinnvoll eingesetzt werden. Zudem finden sich in der Anleitung einige Tipps und Tricks wie die bestehende Grundfunktionalität von ArcGIS für die gegebene Problemstellung genutzt werden kann. Hier sollen einige kurze Kommentare folgen, wie die definierten Projektanforderungen (siehe Kapitel 2) in der GIS- Applikation umgesetzt sind.

5.1.1 Umsetzung Datenschema

Die Daten werden in unserer Applikation mit einer Microsoft Access Datenbank verwaltet. Die Anforderungen an das Datenschema konnten vollständig im implementierten Datenschema umgesetzt werden. Zusätzliche wurden einige Erweiterungen eingebaut. Das implementierte Datenschema ist in Kapitel 3.3 beschrieben.

5.1.2 Umsetzung Funktionalität

Die meisten an die Applikation gestellten Anforderungen an die Funktionalität sind mit der zur Verfügung stehen Version der GIS- Applikation in direkter oder indirekter Form umgesetzt. Die ArcGIS Komponenten ArcView, ArcCatalog und ArcMap decken die Anforderungen (1), (3) und (4) aus Kapitel 2.2 mit der Grundfunktionalität vollständig ab. Die Anforderung automatische Detektion der Kultlinie (5) steht dem Benutzer als zusätzliche Funktion in ArcMap zur Verfügung. Diese zusätzliche Funktion implementiert sämtliche an sie gestellte Anforderungen. Wird die zusätzliche Funktion Bestimmung der Ausgleichsgerade aus einem beliebigen Punkthaufen eingesetzt in Kombination mit der bestehenden Grundfunktionalität, können die Anforderungen (7), (8) und (10) erfüllt werden. Hierzu muss erwähnt sein, dass mit der zusätzlichen Funktion lediglich eine neue Kultlinie aus einer beliebigen Punktkombination neu bestimmen werden kann. Wenn allenfalls die Kultlinie aus welcher die neue Kultlinie bestimmt wird, gelöscht werden soll, muss dies anschliessend mit der Grundfunktionalität von ArcGIS durchgeführt werden. Die Berechnung und Darstellung eines Höhenprofils einer Kultlinie (9) kann mit der

Funktionalität von ArcGIS durchgeführt werden. Jedoch wird dazu die 3D Analysis Extension von ArcGIS benötigt. Einzig die Anforderung (6) konnte aus Zeitgründen in der derzeitigen GIS- Applikation nicht implementiert werden. Diese Anforderung lässt sich auch nicht mit einer bestehende Funktion der ArcGIS Produktefamilie umsetzten.

5.2 Mögliche Erweiterungen

Die Applikation "Frühe Vermessung" ermöglicht die Detektion von möglichen Kultlinien und berechnet deren Lage mit einer Ausgleichung. Der nächste wichtige Schritt in der Suche nach Kultlinien ist die Deutung der Richtungen der Kultlinien, welche schon als Anforderung dieser Arbeit aufgestellt wurde. Siehe Anforderung (6). Mit dieser Funktion wäre der Teil Polarsysteme abgeschlossen und einsetzbar.

Eine weitere mögliche Erweiterung der Applikation ist die Suche nach römischen Limitations- Raster. Eine umfangreiche Kartenstudie wurde zu diesem Thema von Meyer [3] durchgeführt. Im Gegensatz zu den Polarsystemen steht hier die Suche nach orthogonalen Strukturen im Vordergrund. Diese Orthogonalsysteme können als weitere Topic in die Datenbank "Frühe Vermessung" eingefügt werden. Das bestehende Datenschema erlaubt eine Erweiterung. Die bestehenden Limes Programmbibliotheken können genutzt werden und erleichtern damit den Einstieg für die Programmierung weiterer Funktionen.

6. Schlusswort

Es ist kein Wunder, dass schon einige Personen in ihrer Freizeit Kultlinien in Karten gesucht haben. Die Suche nach christlichen Bauwerken, die häufig heidnische Kultplätze bedecken, welche sich in einer Geraden befinden, ist einfach. Es benötigt lediglich ein Massstab, Landkarten und Zeit. Die Suche und Analyse kann mit der entwickelten GIS- Applikation "Frühe Vermessungen" nun automatisch durchgeführt werden. Die ersten Auswertungen mit unserer Applikation haben gezeigt, dass unzählige Linien aus einem beliebigen Punkthaufen relativ rasch gefunden werden können. Die Anzahl der detektierten Linien hängt dabei stark von der zugelassenen Toleranz bei der Detektion ab. Eine gefundene Linie lässt dann auch Spekulationen zu, ob dies jetzt wirklich eine Kultlinie ist. Gewisse Spekulationen sind da erlaubt und beleben die Fantasie. Dies macht die Suche interessant und faszinierend.

Uns ging es genau so, als wir die ersten Tests mit dem Programm durchführten. Das Finden der Linien mit dem Programm geht nun sehr schnell. Ob es sich bei den Linien wirklich um richtige Kultlinien handelt, erfordert nun noch weitere detaillierte Abklärungen und Hintergrundinformationen. Wir hoffen, dass die GIS- Applikation allen interessierten Forscher für die Auswertungen von Kultlinien dienlich ist und sind gespannt auf diese Resultate.

Literaturverzeichnis

[1] U. und G.Büchi: Die Megalithe der Surselva, mehrere Bände, Desertina Verlag

[2] K. Ammann: Spuren früherer Vermessungen und Raumordnungen in der Region Basel und im Alpengebiet, VPK 7/02

[3] P.Meyer-Maurer: Römische Landesvermessung in der Schweiz, mit Kartenauszügen, Helvetica archäologica 115/116, 1998

W. Schlosser, J. Cierny: Sterne und Steine - Eine praktische Astronomie der Vorzeit, Stuttgart: Theiss, 1997

M. Höck, J. Manegold: ArcObject/ArcMap – Programmierung mit VBA, im Eigenverlag 2001

B. Oestereich: Objektorientierte Softwareentwicklung mit der Unified Modeling Language, Oldenbourg Verlag 1997

R. Sedgewick: Algorithmen in C++, Addison-Wesley 1992

Abbildungsverzeichnis

Klassendiagramme: Programmbibliotheken Limes

Erklärung UML- Klassendiagramme:

Jede dynamische Programmbibliothek ist als UML- Klassendiagramm dargestellt. Zu jeder Klasse werden die Membervariabeln und Methoden dargestellt. Die mit einem Plus definierten Einträge stehen für einen öffentlichen Zugriff. Die Einträge mit einem Minus für einen privaten Zugriff. Zusätzlich sind die Assoziationen zwischen den Objekten abgebildet. Die kursiv geschriebenen Klassennamen beschreiben eine abstrakte Klasse.

Sämtliche Klassen dieser Programmbibliotheken müssen mit einer sogenannten Konstruktormethode instanziert werden. Diese sogenannten Konstruktorenmethoden ersetzen die von Visual Basic nicht unterstützte Konstruktorfunktionalität.
Die Instanzierung erfolgt nun auf folgende weise (Konstruktorenmethode fett):

```
Dim bspObjekt As LimesBspKlasse
Set bspObjekt = New LimesBspKlasse
bspObjekt.LimesBspKlasse1 mögliche parameter
```

Die Konstruktormethoden werden in den Klassendiagrammen mit fetter Schrift dargestellt.

Die in den Klassendiagrammen mit rot dargestellten Linien stellen eine Vererbung dar. Dies ist speziell zu erwähnen, da die Vererbung grundsätzlich von Visual Basic nicht unterstützt wird. Mit der Verwendung des Schnittstellenkonzepts und einigem Aufwand kann trotzdem eine Vererbung implementiert werden.

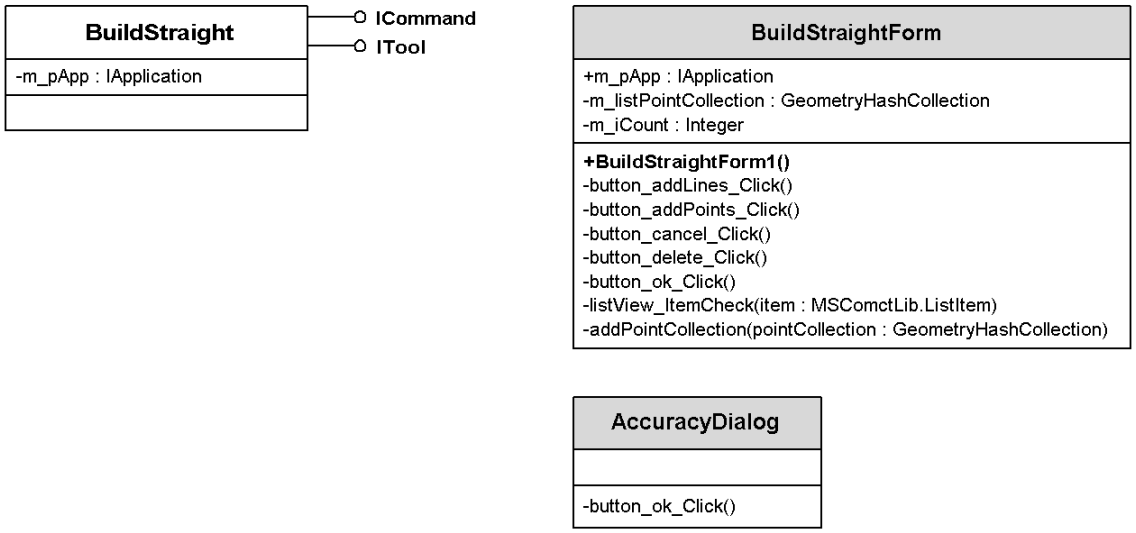
Formulare und Dialoge (Klassen welche Benutzeroberflächen repräsentieren) werden in den Diagrammen durch einen grauen Hintergrund dargestellt. Diese Klasen enthalten keine Konstruktormethoden.

Klassendiagramm: fun_detectLine

Eingebundene Programmbibliotheken:
- Visual Basic For Application
- Visual Basic runtime objects and procedures
- OLE Automation
- **ESRI Object Library (esriCore.olb)**
- **limes_arcObj (limes_arcObj.dll)**
- **limes_geometry(limes_geometry.dll)**
- **limes_calculation(limes_calculation.dll)**

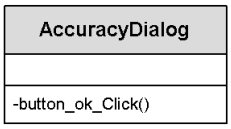
Klassendiagramm: fun_buildStraight

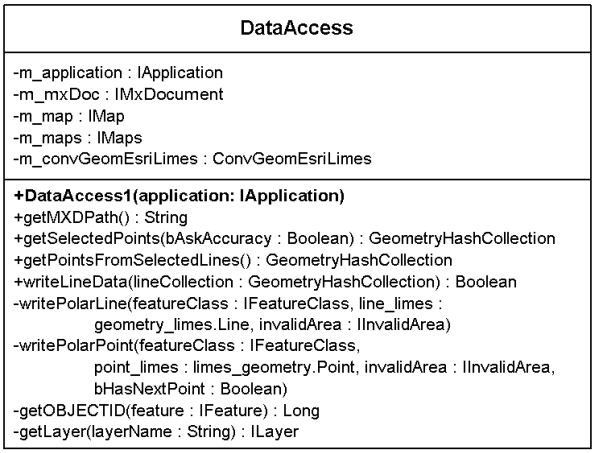
Eingebundene Programmbibliotheken:
- Visual Basic For Application
- Visual Basic runtime objects and procedures
- OLE Automation
- **ESRI Object Library (esriCore.olb)**
- **limes_arcObj (limes_arcObj.dll)**
- **limes_geometry(limes_geometry.dll)**
- **limes_calculation(limes_calculation.dll)**
- Microsoft Windows Common Controls 6.0

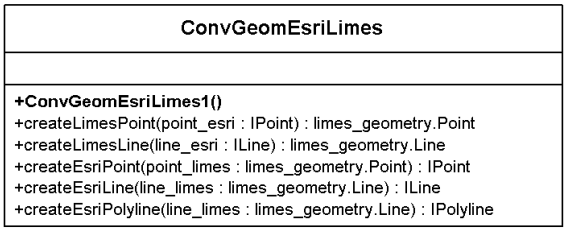


Klassendiagramm: limes_arcObj

Eingebundene Programmbibliotheken:
- Visual Basic For Application
- Visual Basic runtime objects and procedures
- Visual Basic objects and procedures
- OLE Automation
- **ESRI Object Library (esriCore.olb)**
- **ESRI ArcCatalog Object Library (esriGx.olb)**
- **limes_geometry(limes_geometry.dll)**
- **limes_calculation(limes_calculation.dll)**

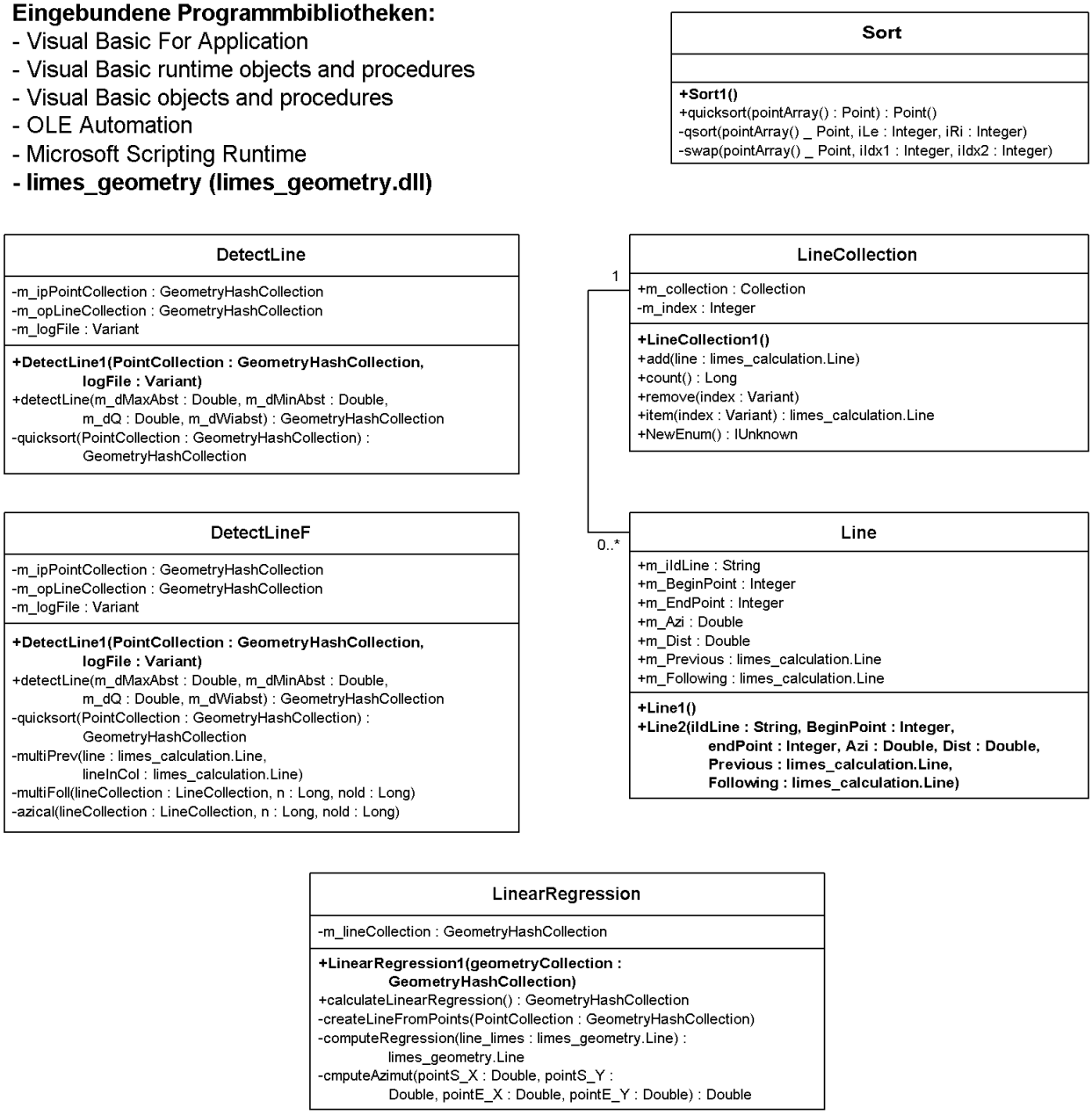






Klassendiagramm: limes_calculation

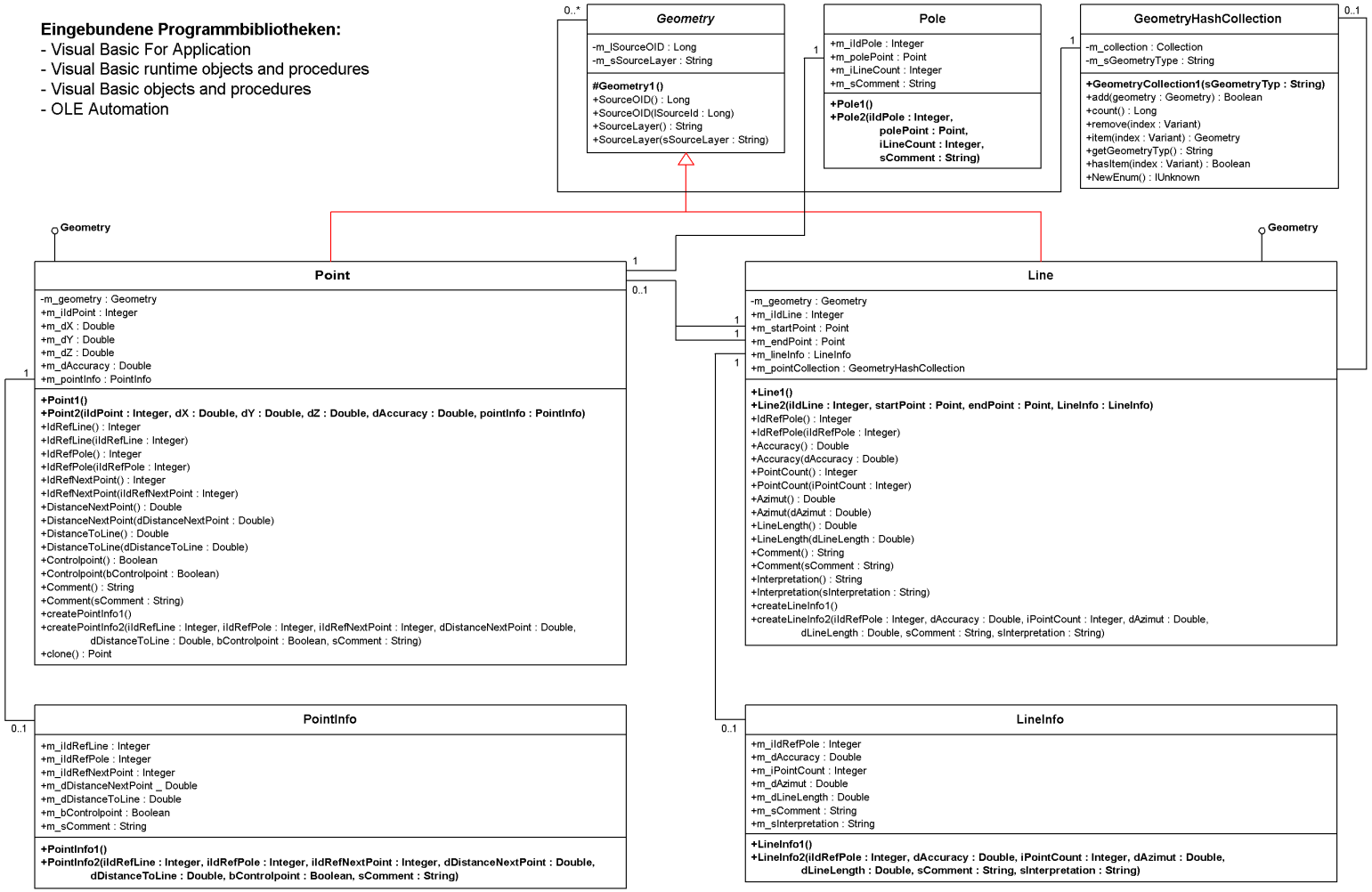
Eingebundene Programmbibliotheken:
- Visual Basic For Application
- Visual Basic runtime objects and procedures
- Visual Basic objects and procedures
- OLE Automation
- Microsoft Scripting Runtime
- **limes_geometry (limes_geometry.dll)**



Die Klassen *DetectLine* und *DetectLineF* stellen beide Methoden für die automatische Liniendetektion aus einem beliebigen Punkthaufen zur Verfügung. Jedoch unterscheiden sich die beiden Klassen durch eine unterschiedliche Implementierung. Die Klasse *DetectLineF* entstand aus den Erkenntnissen der Implementierung der Klasse *DetectLine*. In der derzeitigen GIS- Applikation wird die Klasse *DetectLineF* verwendet.

Klassendiagramm: limes_geometry

Eingebundene Programmbibliotheken:
- Visual Basic For Application
- Visual Basic runtime objects and procedures
- Visual Basic objects and procedures
- OLE Automation



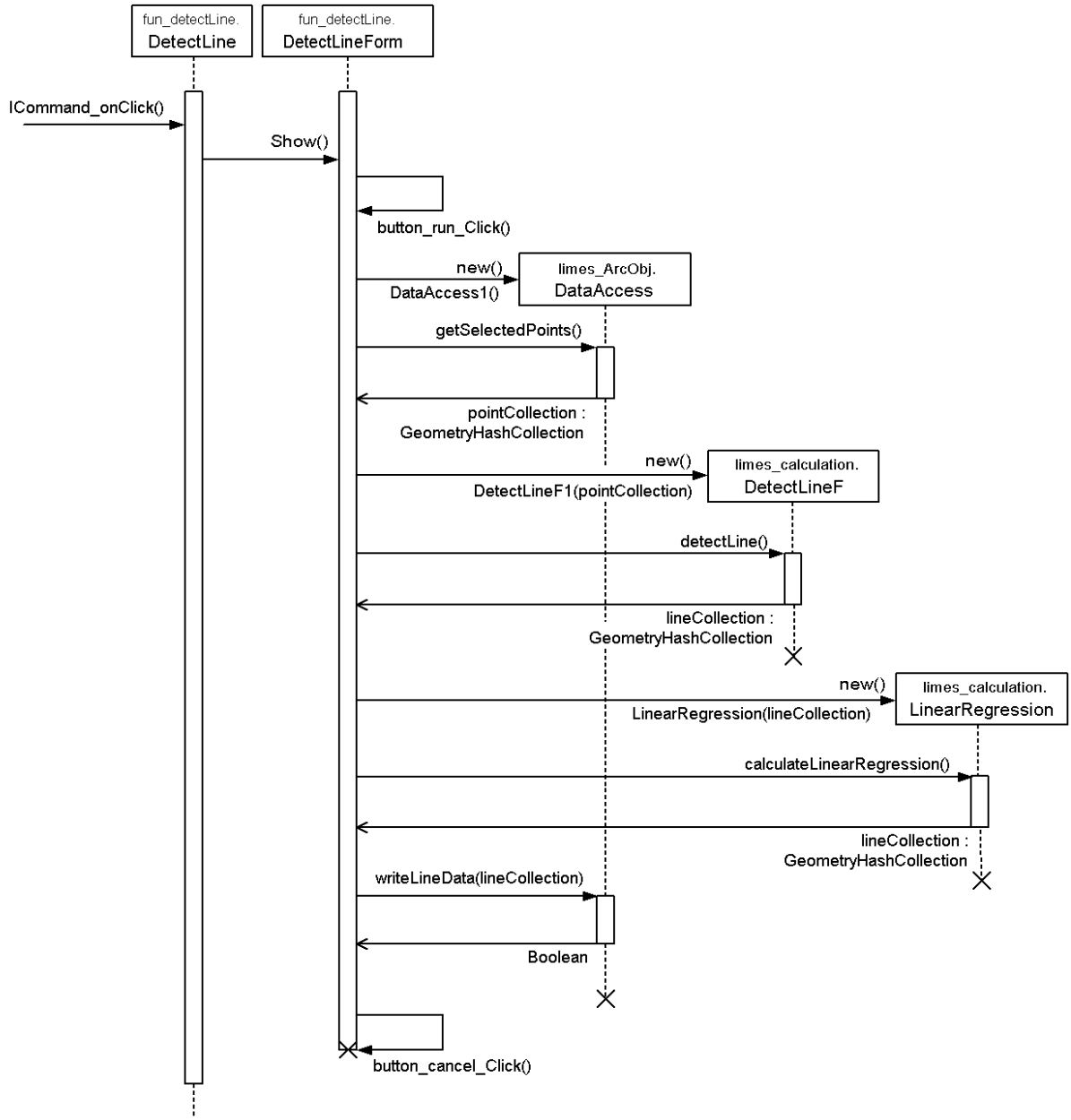
Im Unterschied zu den anderen Klassen dieser Klassenbibliothek bildet die Klasse *GeometryHashCollection* kein Geometrieobjekt ab sondern dient als Speicherstruktur, in welcher Instanzen der anderen Klassen abgelegt werden können. Sie basiert auf der Visual Basic Klasse *Collection*, welche einen dynamischen Array implementiert. Die *GeometryHashCollection* ist nun so angepasst, dass entweder *limes_geometry.Point* Objekte oder *limes_geometry.Line* Objekte im Array gespeichert werden können. Das Hash im Klassennamen steht für die Möglichkeit, mittels einem eindeutigen Identifikator direkt auf eine Objekt im Array zugreifen zu können. Die Klasse bezieht den eindeutigen Identifikator aus den Membervariabeln *m_iIdPoint* bzw. *m_iIdLine*.

Sequenzdiagramme: Funktionen Limes

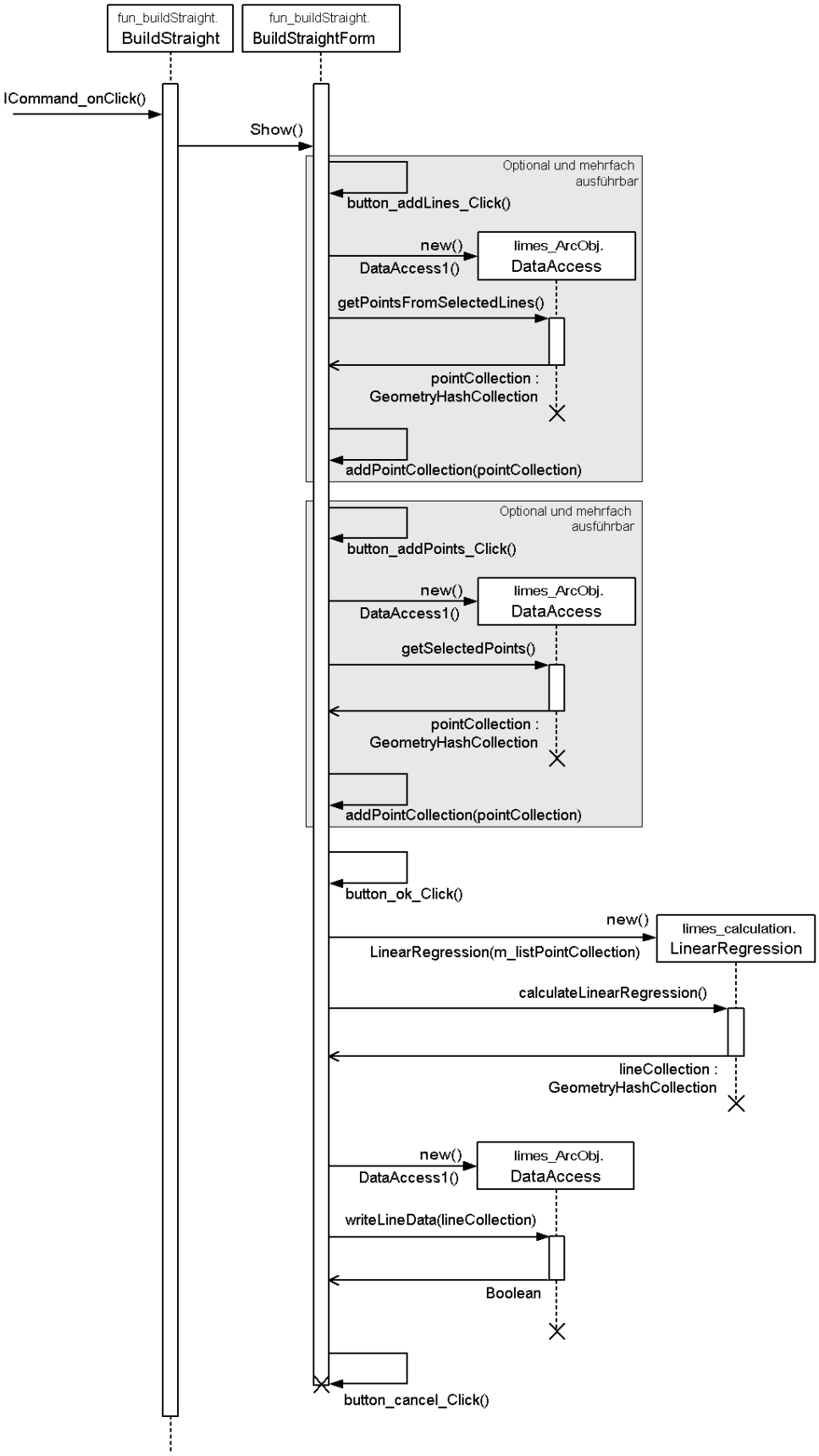
Erklärung UML- Sequenzdiagramme:

Ein Sequenzdiagramm zeigt den zeitlichen Ablauf der Nachrichten welche zwischen den einzelnen Objekten ausgetauscht werden.
Anhand dieser Diagramme soll der Aufbau und Ablauf der beiden implementierten Funktionen besser verständlich dargestellt werden. Es werden die verwendeten Objekte und die eingesetzten Methoden im zeitlichen Ablauf bei der Funktionsausfürung aufgezeigt. Es sind lediglich die für das Verständnis nötigen Objekte und Methoden aufgeführt. Zusätzlich sind wichtige Parameter (Objekte oder atomare Datentypen) dargestellt, welche zwischen den Instanzen ausgetauscht werden.

Sequenzdiagramm: Funktion automatische Liniendetektion



Sequenzdiagramm: Funktion Ausgleichsgerade aus Punktmenge

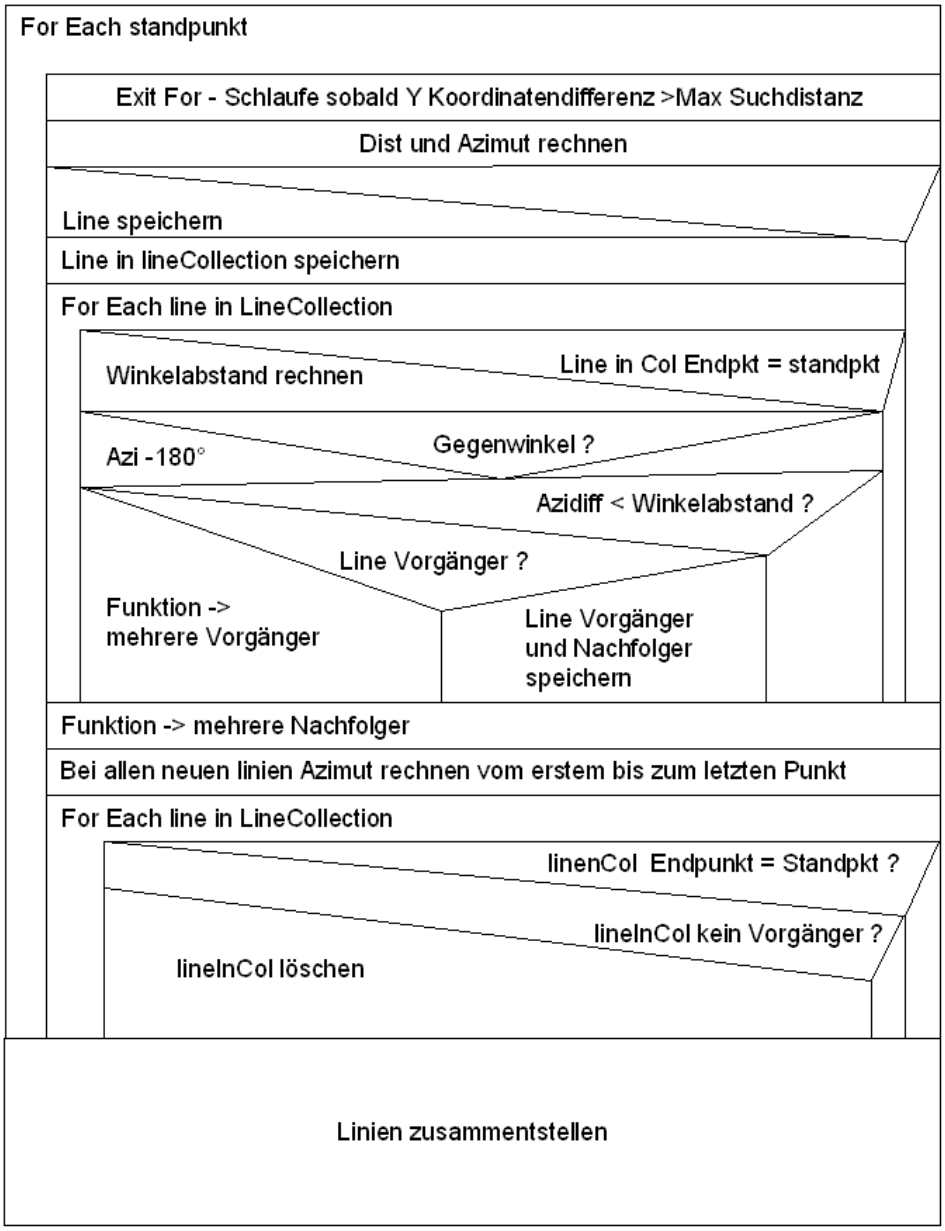


Nassi- Schneidermann Diagramm: Algorithmus Liniendetektion

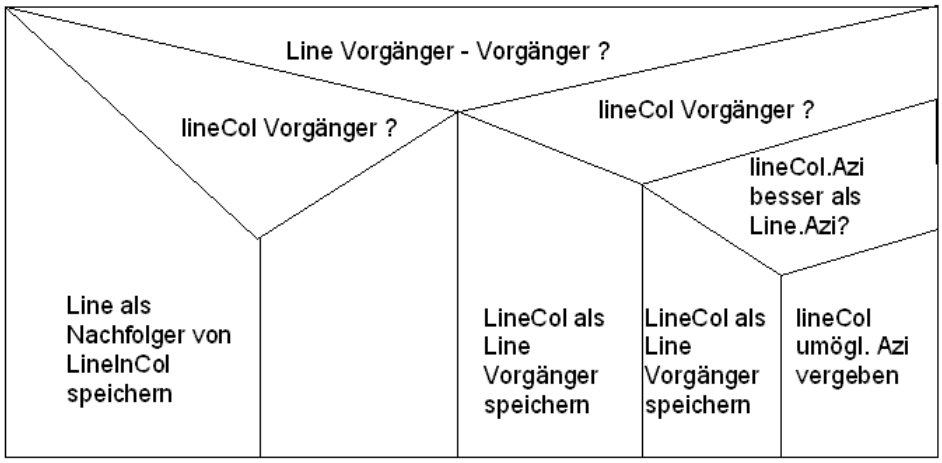
Information zum Diagramm:

Links einer Entscheidung ist die Bedingung "Wahr", rechts einer Entscheidung ist die Bedingung "Falsch".

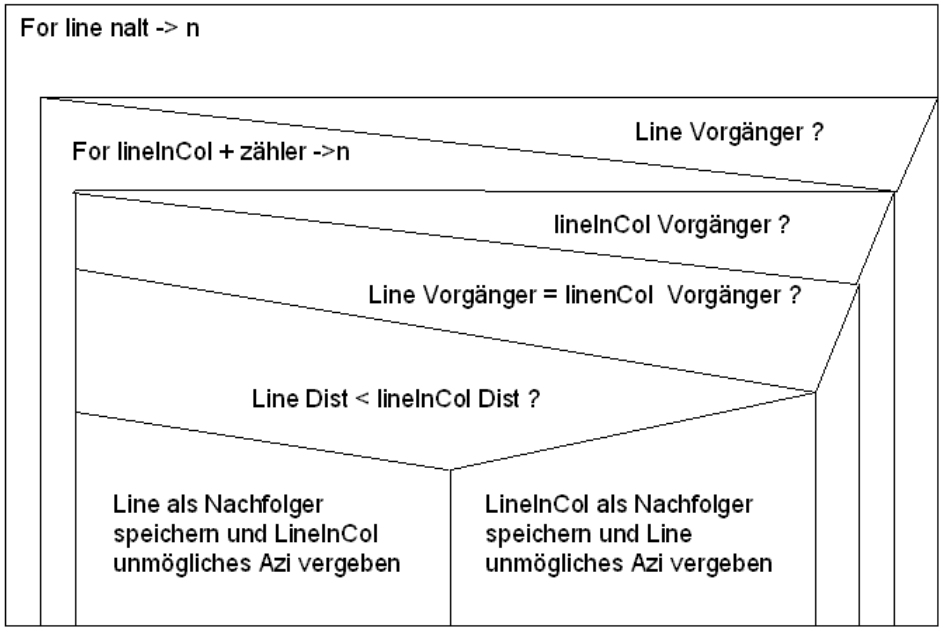
Algorithmus Liniendetektion:



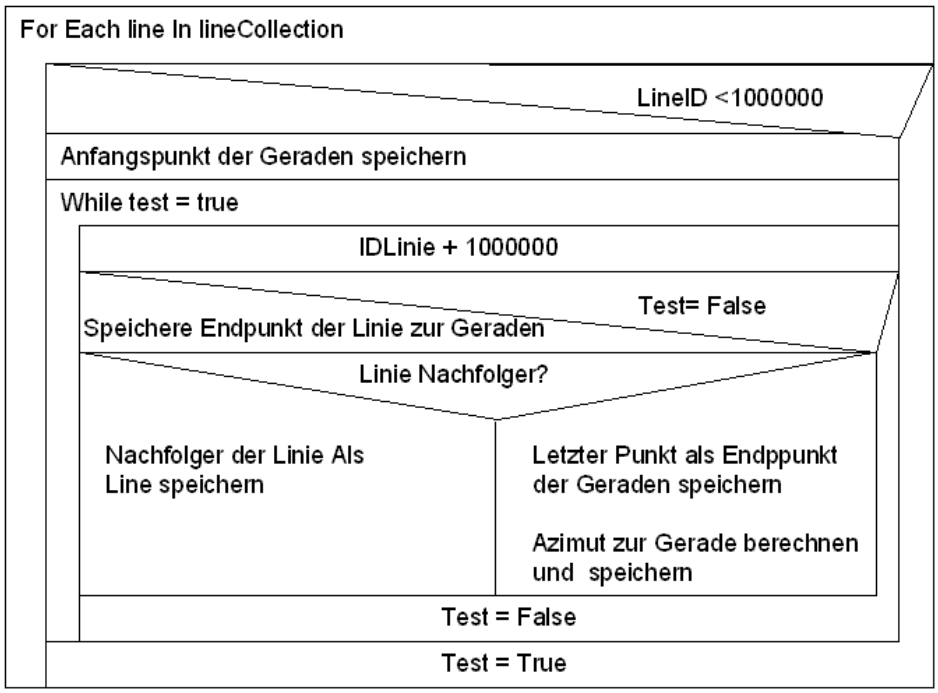
Funktion "mehrere Vorgänger":



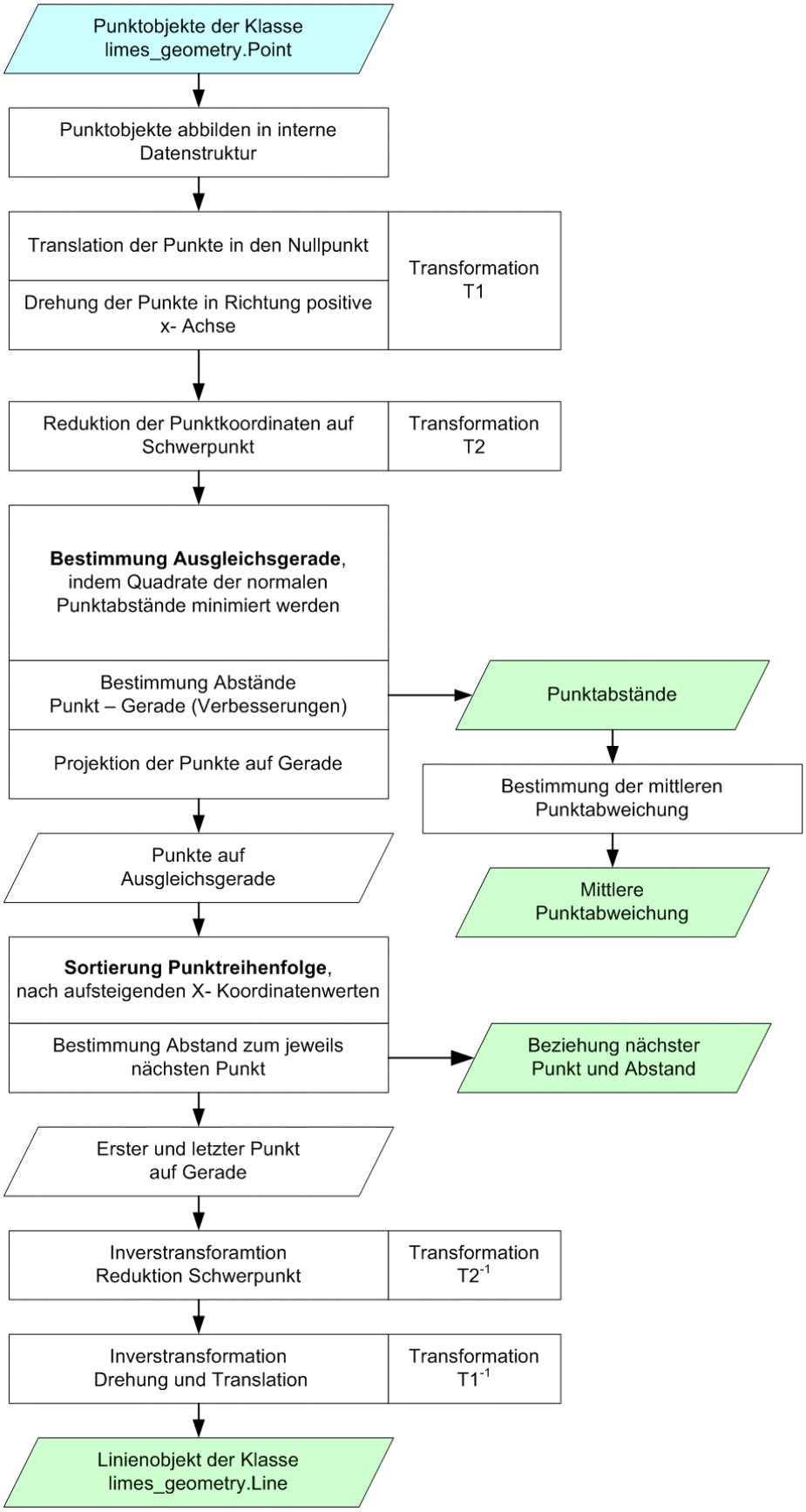
Funktion "mehrere Nachfolger":



Zusammenstellen der Einzellinien zu Kultlinien:



Ablaufdiagramm: Algorithmus Bestimmung Ausgleichsgerade



Sourcecode Programmbibliotheken "Limes"

Dynamic Link Library: limes_geometry

Klasse: Geometry

```
'Abstract Class: limes_geometry.Geometry
'Extends:

'*********************************************************************
'Definition membervariables

Private m_lSourceOID As Long
Private m_sSourceLayer As String

'Definition constructors
Friend Sub Geometry1()

End Sub

'Definition get set methodes

Public Property Get SourceOID() As Long
   SourceOID = m_lSourceOID
End Property

Public Property Let SourceOID(ByVal lSourceId As Long)
   m_lSourceOID = lSourceId
End Property

Public Property Get SourceLayer() As String
   SourceLayer = m_sSourceLayer
End Property

Public Property Let SourceLayer(ByVal sSourceLayer As String)
   m_sSourceLayer = sSourceLayer
End Property
```

Klasse: Point

```
'Class limes_geometry.Point
'Extends limes_geometry.Geometry

Option Explicit
Implements limes_geometry.geometry

'*********************************************************************
'Definition membervariables

Private m_geometry As geometry

Public m_iIdPoint As Integer
Public m_dX As Double
Public m_dY As Double
Public m_dZ As Double
Public m_dAccuracy As Double
Public m_pointInfo As limes_geometry.pointInfo

'Definition constructor
Public Sub point1()
    Set m_geometry = New geometry
    m_geometry.Geometry1
End Sub

Public Sub point2(ByVal iIdPoint As Integer, _
                  ByVal dX As Double, _
                  ByVal dY As Double, _
                  ByVal dZ As Double, _
                  ByVal dAccuracy As Double, _
                  ByRef pointInfo As limes_geometry.pointInfo)
    Set m_geometry = New geometry
    m_geometry.Geometry1

    m_iIdPoint = iIdPoint
    m_dX = dX
    m_dY = dY
    m_dZ = dZ
    m_dAccuracy = dAccuracy
    Set m_pointInfo = pointInfo
End Sub

'Definition get set methodes

Public Property Get IdRefLine() As Integer
    If Not m_pointInfo Is Nothing Then
        IdRefLine = m_pointInfo.m_iIdRefLine
    Else
        IdRefLine = Empty
    End If
End Property

Public Property Let IdRefLine(ByVal iIdRefLine As Integer)
    If Not m_pointInfo Is Nothing Then
        m_pointInfo.m_iIdRefLine = iIdRefLine
    End If
End Property

Public Property Get IdRefPole() As Integer
    If Not m_pointInfo Is Nothing Then
        IdRefPole = m_pointInfo.m_iIdRefPole
    Else
        IdRefPole = Empty
    End If
End Property

Public Property Let IdRefPole(ByVal iIdRefPole As Integer)
    If Not m_pointInfo Is Nothing Then
        m_pointInfo.m_iIdRefPole = iIdRefPole
    End If
End Property
```

```
Public Property Get IdRefNextPoint() As Integer
    If Not m_pointInfo Is Nothing Then
        IdRefNextPoint = m_pointInfo.m_iIdRefNextPoint
    Else
        IdRefNextPoint = Empty
    End If
End Property

Public Property Let IdRefNextPoint(ByVal iIdRefNextPoint As Integer)
    If Not m_pointInfo Is Nothing Then
        m_pointInfo.m_iIdRefNextPoint = iIdRefNextPoint
    End If
End Property

Public Property Get DistanceNextPoint() As Double
    If Not m_pointInfo Is Nothing Then
        DistanceNextPoint = m_pointInfo.m_dDistanceNextPoint
    Else
        DistanceNextPoint = Empty
    End If
End Property

Public Property Let DistanceNextPoint(ByVal dDistanceNextPoint As Double)
    If Not m_pointInfo Is Nothing Then
        m_pointInfo.m_dDistanceNextPoint = dDistanceNextPoint
    End If
End Property

Public Property Get DistanceToLine() As Double
    If Not m_pointInfo Is Nothing Then
        DistanceToLine = m_pointInfo.m_dDistanceToLine
    Else
        DistanceToLine = Empty
    End If
End Property

Public Property Let DistanceToLine(ByVal dDistanceToLine As Double)
    If Not m_pointInfo Is Nothing Then
        m_pointInfo.m_dDistanceToLine = dDistanceToLine
    End If
End Property

Public Property Get Controlpoint() As Boolean
    If Not m_pointInfo Is Nothing Then
        Controlpoint = m_pointInfo.m_bControlpoint
    Else
        Controlpoint = Empty
    End If
End Property

Public Property Let Controlpoint(ByVal bControlpoint As Boolean)
    If Not m_pointInfo Is Nothing Then
        m_pointInfo.m_bControlpoint = bControlpoint
    End If
End Property

Public Property Get Comment() As String
    If Not m_pointInfo Is Nothing Then
        Comment = m_pointInfo.m_sComment
    Else
        Comment = Empty
    End If
End Property

Public Property Let Comment(ByVal sComment As String)
    If Not m_pointInfo Is Nothing Then
        m_pointInfo.m_sComment = sComment
    End If
End Property

'Definition methodes
Public Sub createPointInfo1()
    If m_pointInfo Is Nothing Then
        Set m_pointInfo = New pointInfo
        m_pointInfo.PointInfo1
    End If
End Sub
```

```
Public Sub createPointInfo2(ByVal iIdRefLine As Integer, _
                            ByVal iIdRefPole As Integer, _
                            ByVal iIdRefNextPoint As Integer, _
                            ByVal dDistanceNextPoint As Double, _
                            ByVal dDistanceToLine As Double, _
                            ByVal bControlpoint As Boolean, _
                            ByVal sComment As String)
    Set m_pointInfo = New pointInfo
    m_pointInfo.PointInfo2 _
            iIdRefLine, _
            iIdRefPole, _
            iIdRefNextPoint, _
            dDistanceNextPoint, _
            dDistanceToLine, _
            bControlpoint, _
            sComment
End Sub

Public Function clone() As limes_geometry.point

    Dim clonePoint As limes_geometry.point
    Set clonePoint = New limes_geometry.point
    clonePoint.point1

    clonePoint.m_dX = m_dX
    clonePoint.m_dY = m_dY
    clonePoint.m_dZ = m_dZ
    clonePoint.m_dAccuracy = m_dAccuracy
    clonePoint.m_iIdPoint = m_iIdPoint

    clonePoint.Geometry_SourceLayer = Geometry_SourceLayer
    clonePoint.Geometry_SourceOID = Geometry_SourceOID

    If Not m_pointInfo Is Nothing Then
        clonePoint.createPointInfo1

        clonePoint.Comment = Comment
        clonePoint.Controlpoint = Controlpoint
        clonePoint.DistanceNextPoint = DistanceNextPoint
        clonePoint.DistanceToLine = DistanceToLine
        clonePoint.IdRefLine = IdRefLine
        clonePoint.IdRefNextPoint = IdRefNextPoint
        clonePoint.IdRefPole = IdRefPole
    End If

    Set clone = clonePoint

End Function


'*********************************************************************
'Implemnts: limes_geometry.Geometry

Friend Sub Geometry_Geometry1()

End Sub

Public Property Get Geometry_SourceOID() As Long
   Geometry_SourceOID = m_geometry.SourceOID
End Property

Public Property Let Geometry_SourceOID(ByVal lSourceId As Long)
   m_geometry.SourceOID = lSourceId
End Property

Public Property Get Geometry_SourceLayer() As String
   Geometry_SourceLayer = m_geometry.SourceLayer
End Property

Public Property Let Geometry_SourceLayer(ByVal sSourceLayer As String)
   m_geometry.SourceLayer = sSourceLayer
End Property
```

Klasse: PointInfo

```
'Class: limes_geometry.PointInfo
'Extends:

'**********************************************************************
'Definition membervariables

Public m_iIdRefLine As Integer
Public m_iIdRefPole As Integer
Public m_iIdRefNextPoint As Integer
Public m_dDistanceNextPoint As Double
Public m_dDistanceToLine As Double
Public m_bControlpoint As Boolean
Public m_sComment As String


'Definition constructors

Public Sub PointInfo1()

End Sub

Public Sub PointInfo2(ByVal iIdRefLine As Integer, _
                      ByVal iIdRefPole As Integer, _
                      ByVal iIdRefNextPoint As Integer, _
                      ByVal dDistanceNextPoint As Double, _
                      ByVal dDistanceToLine As Double, _
                      ByVal bControlpoint As Boolean, _
                      ByVal sComment As String)
    m_iIdRefLine = iIdRefLine
    m_iIdRefPole = iIdRefPole
    m_iIdRefNextPoint = iIdRefNextPoint
    m_dDistanceNextPoint = dDistanceNextPoint
    m_dDistanceToLine = dDistanceToLine
    m_bControlpoint = bControlpoint
    m_sComment = sComment
End Sub
```

Klasse: Line

```
'Class: limes_geometry.Line
'Extends: limes_geometry.Geometry

Option Explicit
Implements limes_geometry.geometry

'*********************************************************************
'Definition membervariables

Private m_geometry As geometry

Public m_iIdLine As Integer
Public m_startPoint As limes_geometry.point
Public m_endPoint As limes_geometry.point
Public m_lineInfo As limes_geometry.LineInfo
Public m_pointCollection As limes_geometry.GeometryHashCollection

'Definition constructors

Public Sub Line1()
    Set m_geometry = New geometry
    m_geometry.Geometry1
End Sub

Public Sub Line2(ByVal iIdLine As Integer, _
                 ByRef startPoint As limes_geometry.point, _
                 ByRef endPoint As limes_geometry.point, _
                 ByRef LineInfo As limes_geometry.LineInfo)
    Set m_geometry = New geometry
    m_geometry.Geometry1

    m_iIdLine = iIdLine
    Set m_startPoint = startPoint
    Set m_endPoint = endPoint
    Set m_lineInfo = LineInfo
End Sub

'Definition get set methodes

Public Property Get IdRefPole() As Integer
    If Not m_lineInfo Is Nothing Then
        IdRefPole = m_lineInfo.m_iIdRefPole
    Else
        IdRefPole = Empty
    End If
End Property

Public Property Let IdRefPole(ByVal iIdRefPole As Integer)
    If Not m_lineInfo Is Nothing Then
        m_lineInfo.m_iIdRefPole = iIdRefPole
    End If
End Property

Public Property Get Accuracy() As Double
    If Not m_lineInfo Is Nothing Then
        Accuracy = m_lineInfo.m_dAccuracy
    Else
        Accuracy = Empty
    End If
End Property

Public Property Let Accuracy(ByVal dAccuracy As Double)
    If Not m_lineInfo Is Nothing Then
        m_lineInfo.m_dAccuracy = dAccuracy
    End If
End Property

Public Property Get PointCount() As Integer
    If Not m_lineInfo Is Nothing Then
        PointCount = m_lineInfo.m_iPointCount
    Else
        PointCount = Empty
```

```
    End If
End Property

Public Property Let PointCount(ByVal iPointCount As Integer)
    If Not m_lineInfo Is Nothing Then
        m_lineInfo.m_iPointCount = iPointCount
    End If
End Property

Public Property Get Azimut() As Double
    If Not m_lineInfo Is Nothing Then
        Azimut = m_lineInfo.m_dAzimut
    Else
        Azimut = Empty
    End If
End Property

Public Property Let Azimut(ByVal dAzimut As Double)
    If Not m_lineInfo Is Nothing Then
        m_lineInfo.m_dAzimut = dAzimut
    End If
End Property

Public Property Get LineLength() As Double
    If Not m_lineInfo Is Nothing Then
        LineLength = m_lineInfo.m_dLineLength
    Else
        LineLength = Empty
    End If
End Property

Public Property Let LineLength(ByVal dLineLength As Double)
    If Not m_lineInfo Is Nothing Then
        m_lineInfo.m_dLineLength = dLineLength
    End If
End Property

Public Property Get Comment() As String
    If Not m_lineInfo Is Nothing Then
        Comment = m_lineInfo.m_sComment
    Else
        Comment = Empty
    End If
End Property

Public Property Let Comment(ByVal sComment As String)
    If Not m_lineInfo Is Nothing Then
        m_lineInfo.m_sComment = sComment
    End If
End Property

Public Property Get Interpretation() As String
    If Not m_lineInfo Is Nothing Then
        Interpretation = m_lineInfo.m_sInterpretation
    Else
        Interpretation = Empty
    End If
End Property

Public Property Let Interpretation(ByVal sInterpretation As String)
    If Not m_lineInfo Is Nothing Then
        m_lineInfo.m_sInterpretation = sInterpretation
    End If
End Property

'Definition methodes

Public Sub createLineInfo1()
    If m_lineInfo Is Nothing Then
        Set m_lineInfo = New LineInfo
        m_lineInfo.LineInfo1
    End If
End Sub

Public Sub createLineInfo2(ByVal iIdRefPole As Integer, _
                           ByVal dAccuracy As Double, _
                           ByVal iPointCount As Integer, _
```

```
                          ByVal dAzimut As Double, _
                          ByVal dLineLength As Double, _
                          ByVal sComment As String, _
                          ByVal sInterpretation As String)
    Set m_lineInfo = New LineInfo
    m_lineInfo.LineInfo2 _
            iIdRefPole, _
            dAccuracy, _
            iPointCount, _
            dAzimut, _
            dLineLength, _
            sComment, _
            sInterpretation
End Sub

'*********************************************************************
'Implemnts: limes_geometry.Geometry

Friend Sub Geometry_Geometry1()

End Sub

Public Property Get Geometry_SourceOID() As Long
   Geometry_SourceOID = m_geometry.SourceOID
End Property

Public Property Let Geometry_SourceOID(ByVal lSourceId As Long)
   m_geometry.SourceOID = lSourceId
End Property

Public Property Get Geometry_SourceLayer() As String
   Geometry_SourceLayer = m_geometry.SourceLayer
End Property

Public Property Let Geometry_SourceLayer(ByVal sSourceLayer As String)
   m_geometry.SourceLayer = sSourceLayer
End Property
```

Klasse: LineInfo

```
'Class: limes_geometry.LineInfo
'Extends:

'*********************************************************************
'Definition membervariables

Public m_iIdRefPole As Integer
Public m_dAccuracy As Double
Public m_iPointCount As Integer
Public m_dAzimut As Double
Public m_dLineLength As Double
Public m_sComment As String
Public m_sInterpretation As String

'Definition constructors

Public Sub LineInfo1()

End Sub

Public Sub LineInfo2(ByVal iIdRefPole As Integer, _
                     ByVal dAccuracy As Double, _
                     ByVal iPointCount As Integer, _
                     ByVal dAzimut As Double, _
                     ByVal dLineLength As Double, _
                     ByVal sComment As String, _
                     ByVal sInterpretation As String)
    m_iIdRefPole = iIdRefPole
    m_dAccuracy = dAccuracy
    m_iPointCount = iPointCount
    m_dAzimut = dAzimut
    m_dLineLength = dLineLength
    m_sComment = sComment
    m_sInterpretation = sInterpretation
End Sub
```

Klasse: Pole

```
'Class: limes_geometry.Pole
'Extends:

'**********************************************************************
'Definition membervariables

Public m_iIdPole As Integer
Public m_polePoint As limes_geometry.point
Public m_iLineCount As Integer
Public m_sComment As String

'Definition constructors

Public Sub Pole1()

End Sub

Public Sub Pole2(ByVal iIdPole As Integer, _
                 ByRef polePoint As limes_geometry.point, _
                 ByVal iLineCount As Integer, _
                 ByVal sComment As String)
    m_iIdPole = iIdPole
    Set m_polePoint = polePoint
    m_iLineCount = iLineCount
    m_sComment = sComment
End Sub
```

Klasse: GeometryHashCollection

```
'Class limes_geometry.GeometryHashCollection
'Extends

'*********************************************************************
'Definition membervariables

Private m_collection As Collection
Private m_sGeometryType As String

'Definition constructors

'Constructor GeometryCollection1:
'alternative values for sGeometryTyp are:
'    "Point" - For a collection with limes_geometry.Point instances
'    "Line" - For a collection with limes_geometry.Line instances
Public Sub GeometryCollection1(ByVal sGeometryTyp As String)
    Set m_collection = New Collection
    m_sGeometryType = sGeometryTyp
End Sub

'Definition methodes

Public Function add(ByRef geometry As limes_geometry.geometry) As Boolean
    If TypeOf geometry Is limes_geometry.point Then
        If m_sGeometryType = "Point" Then
            Dim point As limes_geometry.point
            Set point = geometry
            Dim sIndex1 As String
            sIndex1 = "id_" & point.m_iIdPoint
            m_collection.add point, sIndex1
            add = True
        Else
            add = False
        End If
        Exit Function
    End If
    If TypeOf geometry Is limes_geometry.line Then
        If m_sGeometryType = "Line" Then
            Dim line As limes_geometry.line
            Set line = geometry
            Dim sIndex2 As String
            sIndex2 = "id_" & line.m_iIdLine
            m_collection.add line, sIndex2
            add = True
        Else
            add = False
        End If
        Exit Function
    End If
    add = False
End Function

Public Function count() As Long
   count = m_collection.count
End Function

Public Sub remove(ByVal index As Variant)
    Dim sIndex As String
    sIndex = "id_" & index
    m_collection.remove sIndex
End Sub

Public Function item(ByVal index As Variant) As limes_geometry.geometry
    On Error GoTo ErrorHandler
    Dim sIndex As String
    sIndex = "id_" & index
    Set item = m_collection.item(sIndex)
    Exit Function

ErrorHandler:
    Set item = Nothing
End Function
```

```
Public Function getGeometryTyp() As String
    getGeometryTyp = m_sGeometryType
End Function

Public Function hasItem(ByVal index As Variant) As Boolean
    If Me.item(index) Is Nothing Then
        hasItem = False
    Else
        hasItem = True
    End If
End Function

Public Function NewEnum() As IUnknown
   Set NewEnum = m_collection.[_NewEnum]
End Function
```

Dynamic Link Library: limes_calcualation

Klasse: DetectLineF

```
'Class: limes_calculation.DetectLineF
'Extends:

Option Explicit

'*********************************************************************
'Definition membervariables

Private m_ipPointCollection As limes_geometry.GeometryHashCollection
Private m_opLineCollection As limes_geometry.GeometryHashCollection
Private m_logFile As Variant



'Definition constructors

Public Sub DetectLine1(PointCollection As limes_geometry.GeometryHashCollection, _
                       logFile As Variant)
    Set m_ipPointCollection = PointCollection
    Set m_logFile = logFile

    Set m_opLineCollection = New GeometryHashCollection
    m_opLineCollection.GeometryCollection1 "Line"

End Sub

'Definition methodes

Public Function DetectLine(m_dMaxAbst As Double, m_dMinAbst As Double, _
        m_dQ As Double, m_dWiabst As Double) _
        As limes_geometry.GeometryHashCollection


    Dim lineCollection As limes_calculation.lineCollection
    Set lineCollection = New limes_calculation.lineCollection
    lineCollection.LineCollection1

    Dim stpt As limes_geometry.point
    Dim targetpt As limes_geometry.point
    Dim lineInCol As limes_calculation.line

    Dim pi As Double
    Dim k As Integer
    Dim n As Long
    Dim nold As Long
    nold = 1
    Dim m As Integer
    Dim i As Integer

    m = 1

    Dim totPt As Integer
    totPt = m_ipPointCollection.count

    Dim Dist As Double
    Dim Azi As Double
    Dim dX As Double
    Dim dY As Double

    Dim QinAngel As Double
    Dim totalSearchAngle As Double
    Dim Azdiff As Double

    'Write head into limes.log - File
    m_logFile.WriteLine ("Limes Linedetection protocol")
    m_logFile.WriteLine ("***************************")
    m_logFile.WriteBlankLines (1)
    m_logFile.WriteLine ("Selected Points : " & totPt)
```

```
    m_logFile.WriteBlankLines (1)
    m_logFile.WriteLine ("search parameter")
    m_logFile.WriteLine ("maximal dist: " & m_dMaxAbst)
    m_logFile.WriteLine ("minimal dist: " & m_dMinAbst)
    m_logFile.WriteLine ("search Q: " & m_dQ)
    m_logFile.WriteLine ("search angle: " & m_dWiabst)
    m_logFile.WriteBlankLines (1)

    pi = 4 * Atn(1)
    m_dWiabst = m_dWiabst / 180 * pi


    'sort Point

    Set m_ipPointCollection = quicksort(m_ipPointCollection)

    m_logFile.WriteLine ("End sort points")

    'generate Line

    For Each stpt In m_ipPointCollection       'Iteration over standpoint
        m_logFile.Write (",S" & m)

        For i = (m + 1) To totPt               'Iteration over neighbourpoint
            Set targetpt = m_ipPointCollection.item(i)

            If targetpt.m_dX - stpt.m_dX > m_dMaxAbst Then  'change standpoint
                Exit For                                    'there is no more
            End If                                          'Dist < MaxDist

            dX = targetpt.m_dX - stpt.m_dX
            dY = targetpt.m_dY - stpt.m_dY
            Dist = Sqr((dX * dX) + (dY * dY))

            If Not dY = 0 Then                  'catch Division zero
                Azi = (Atn(dX / dY))
            Else
                Azi = 90
            End If

            If Dist < m_dMaxAbst And Dist > 10 Then
                    Dim line As limes_calculation.line
                    Set line = New limes_calculation.line
                    line.line2 n, stpt.m_iIdPoint, targetpt.m_iIdPoint, _
                    Azi, Dist, Nothing, Nothing
                    lineCollection.add line

                    n = n + 1

                    m_logFile.Write (" Z" & i)

                    For Each lineInCol In lineCollection

                        If lineInCol.m_EndPoint = stpt.m_iIdPoint Then

                            QinAngel = Tan(m_dQ / Dist)
                            totalSearchAngle = QinAngel + m_dWiabst

                            Azdiff = Abs(Azi - lineInCol.m_Azi)

                                If Azdiff > (pi - totalSearchAngle) Then
                                    Azdiff = Abs(Azdiff - pi)
                                End If

                                If Azdiff < totalSearchAngle Then

                                    If line.m_Previous Is Nothing Then
                                        Set line.m_Previous = lineInCol
                                        Set lineInCol.m_Following = line
                                        line.m_Azi = Azi
                                        m_logFile.Write ("*")
                                    Else
                                        Call multiPrev(line, lineInCol)
                                        m_logFile.Write ("MP")
                                    End If

                                End If
```

```
                    End If
                Next
            End If
        Next

        'solve multiple following lines
        If nold > 0 Then
            Call multiFoll(lineCollection, n, nold)
        End If

        'calculate azimut
        If nold > 0 Then
            Call azical(lineCollection, n, nold)
        End If

        m_logFile.Write ("ES")

        'delete Line
        k = 1
        For Each lineInCol In lineCollection
            If lineInCol.m_EndPoint = stpt.m_iIdPoint Then
                If lineInCol.m_Following Is Nothing _
                And lineInCol.m_Previous Is Nothing Then
                    lineCollection.remove (k)
                    k = k - 1
                    n = n - 1
                End If
            End If
            k = k + 1
        Next
        nold = n
        m = m + 1
    Next
    m_logFile.WriteBlankLines (1)
    m_logFile.WriteLine ("End generating single lines")
    m_logFile.WriteLine ("Count of single lines: " & lineCollection.count)

    m_logFile.Write ("Assembling lines : 0")

    'assembling line
    Dim test As Boolean
    Dim pointtest As Boolean
    Dim defline As limes_geometry.line
    Dim point As limes_geometry.point
    Dim m_pointCollection As limes_geometry.GeometryHashCollection

    n = 1
    test = True
    k = 0
    For Each line In lineCollection
     If line.m_iIdLine < 1000000 Then

        Set point = m_ipPointCollection.item(line.m_BeginPoint)
        Set point = point.Clone
        If point.m_pointInfo Is Nothing Then
            point.createPointInfo1
            point.Controlpoint = True
        End If

        m_logFile.Write (",L" & n)                          'write linenumber and startpoint
        m_logFile.Write (" " & point.m_iIdPoint)            'in protocol-file

        Set defline = New limes_geometry.line
        defline.line2 n, point, Nothing, Nothing
        defline.createLineInfo1
        Set m_pointCollection = New limes_geometry.GeometryHashCollection
        m_pointCollection.GeometryCollection1 "Point"
        Set defline.m_pointCollection = m_pointCollection
        defline.m_pointCollection.add point

        i = 1
        While test = True
            line.m_iIdLine = line.m_iIdLine + 1000000

            pointtest = m_pointCollection.hasItem(line.m_EndPoint)

            If pointtest = False Then
```

```
                Set point = m_ipPointCollection.item(line.m_EndPoint)
                Set point = point.Clone
                If point.m_pointInfo Is Nothing Then
                    point.createPointInfo1
                    point.Controlpoint = True
                End If

                m_logFile.Write (" " & point.m_iIdPoint)      'write linepoints in protocol

                defline.m_pointCollection.add point
            End If

            If Not line.m_Following Is Nothing And i < 10 Then
                Set line = line.m_Following
            Else
                Set defline.m_EndPoint = point
                If line.m_Azi < 0 Then
                    line.m_Azi = line.m_Azi + 2 * pi
                End If
                defline.Azimut = line.m_Azi
                m_opLineCollection.add defline

                n = n + 1
                test = False
            End If
            i = i + 1
        Wend
        test = True
     End If

    Next
    m_logFile.WriteBlankLines (1)

    m_logFile.WriteLine ("Number of lines detected: " & m_opLineCollection.count)
    m_logFile.WriteLine ("End Limes Linedetection ")
    m_logFile.Close

    Set DetectLine = m_opLineCollection
End Function


Function quicksort(PointCollection As limes_geometry.GeometryHashCollection) _
        As limes_geometry.GeometryHashCollection

    Dim totPt As Integer
    totPt = PointCollection.count
    ReDim pointArray(0 To totPt - 1) As limes_geometry.point
    Dim point As limes_geometry.point

    Dim a As Integer
    a = 0
    For Each point In PointCollection
        Set pointArray(a) = point
        a = a + 1
    Next

    Dim sort As limes_calculation.sort
    Set sort = New limes_calculation.sort
    sort.Sort1

    pointArray = sort.quicksort(pointArray)
    Set PointCollection = New limes_geometry.GeometryHashCollection
    PointCollection.GeometryCollection1 "Point"

    a = 0
    For a = 0 To totPt - 1
        pointArray(a).m_iIdPoint = a + 1
        PointCollection.add pointArray(a)
    Next a

    Set quicksort = PointCollection

End Function

Function multiPrev(line As limes_calculation.line, _
                   lineInCol As limes_calculation.line)
```

```
    If line.m_Previous.m_Previous Is Nothing Then
        If lineInCol.m_Previous Is Nothing Then
            Dim Azdiff1 As Double
            Dim Azdiff2 As Double
            Azdiff1 = line.m_Azi - line.m_Previous.m_Azi
            Azdiff2 = line.m_Azi - lineInCol.m_Azi
            If Abs(Azdiff2) < Abs(Azdiff1) Then
                Set line.m_Previous.m_Following = Nothing
                line.m_Previous.m_Azi = 1000000
                Set line.m_Previous = lineInCol
                Set lineInCol.m_Following = line
            Else
                lineInCol.m_Azi = 1000000
            End If
        Else
                Set line.m_Previous.m_Following = Nothing
                Set line.m_Previous = lineInCol
                Set lineInCol.m_Following = line
        End If
    Else
        If lineInCol.m_Previous Is Nothing Then

        Else
        Set lineInCol.m_Following = line
        End If

    End If
End Function

Function multiFoll(lineCollection As limes_calculation.lineCollection, _
                   n As Long, nold As Long)
    Dim i As Long
    Dim k As Long
    Dim line As limes_calculation.line
    Dim lineInCol As limes_calculation.line

    For i = nold To n
        Set line = lineCollection.item(i)
        If Not line.m_Previous Is Nothing Then
            For k = (i + 1) To n
                Set lineInCol = lineCollection.item(k)
                If Not lineInCol.m_Previous Is Nothing Then
                    If line.m_Previous Is lineInCol.m_Previous Then
                        m_logFile.Write ("MF ")
                        If line.m_Dist > lineInCol.m_Dist Then
                            Set line.m_Previous.m_Following = lineInCol
                            Set line.m_Previous = Nothing
                            line.m_Azi = 1000000
                            m_logFile.Write ("F2-" & line.m_EndPoint)
                        Else
                            Set line.m_Previous.m_Following = line
                            Set lineInCol.m_Previous = Nothing
                            lineInCol.m_Azi = 1000000
                            m_logFile.Write ("F-" & lineInCol.m_EndPoint)
                        End If
                    End If
                End If
            Next
        End If
    Next
    multiFoll = n
End Function
Function azical(lineCollection As limes_calculation.lineCollection, _
                   n As Long, nold As Long)
    Dim i As Long
    Dim endPoint As limes_geometry.point
    Dim startPoint As limes_geometry.point
    Dim line As limes_calculation.line
    Dim lineInCol As limes_calculation.line
    Dim first As Boolean

    Dim dX As Double
    Dim dY As Double
    Dim Azi As Double

    For i = nold + 1 To n
        first = True
```

```
        Set line = lineCollection.item(i)
        Set endPoint = m_ipPointCollection.item(line.m_BeginPoint)
        Set lineInCol = line
        While Not lineInCol.m_Previous Is Nothing
            If first = True Then
                lineInCol.m_iIdLine = lineInCol.m_iIdLine + 1000000
            End If
            first = False
            Set lineInCol = lineInCol.m_Previous
            Set startPoint = m_ipPointCollection.item(lineInCol.m_BeginPoint)
            dX = endPoint.m_dX - startPoint.m_dX
            dY = endPoint.m_dY - startPoint.m_dY
            If dY = 0 Then
                Azi = 90
            Else
                Azi = (Atn(dX / dY))
            End If
            line.m_Azi = Azi
        Wend
    Next
End Function
```

Klasse: Line

```
'Class limes_calculation.Line
'Extends limes_calculation.Line

Option Explicit

'*********************************************************************
'Definition membervariables

Public m_iIdLine As String
Public m_BeginPoint As Integer
Public m_EndPoint As Integer
Public m_Azi As Double
Public m_Dist As Double
Public m_Previous As limes_calculation.line
Public m_Following As limes_calculation.line



'Definition constructor

Public Sub line1()

End Sub

Public Sub line2(ByVal iIdLine As String, _
                 ByVal BeginPoint As Integer, _
                 ByVal endPoint As Integer, _
                 ByVal Azi As Double, _
                 ByVal Dist As Double, _
                 ByVal Previous As limes_calculation.line, _
                 ByVal Following As limes_calculation.line)

    m_iIdLine = iIdLine
    m_BeginPoint = BeginPoint
    m_EndPoint = endPoint
    m_Azi = Azi
    m_Dist = Dist
    Set m_Previous = Previous
    Set m_Following = Following
End Sub
```

Klasse: LineCollection

```
'Class limes_calculation.LineCollection
'Extends

Option Explicit

'*********************************************************************
'Definition membervariables

Private m_collection As Collection
Public m_index As Integer


'Definition constructors

Public Sub LineCollection1()

    Set m_collection = New Collection

End Sub

'Definition methodes

Public Sub add(ByRef line As limes_calculation.line)

            m_collection.add line

End Sub

Public Function count() As Long
   count = m_collection.count
End Function

Public Sub remove(ByVal index As Variant)
   m_collection.remove index
End Sub

Public Function item(ByVal index As Variant) As limes_calculation.line
    Set item = m_collection.item(index)
End Function

Public Function NewEnum() As IUnknown
   Set NewEnum = m_collection.[_NewEnum]
End Function
```

Klasse: LinearRegression

```
'Class: limes_calculation.LinearRegression
'Extends:

Option Explicit

'*********************************************************************
'Definition membervariables

Private m_lineCollection As limes_geometry.GeometryHashCollection
Private pi As Double

'Definition constructors

Public Sub LinearRegression1(geometryCollection As limes_geometry.GeometryHashCollection)

    pi = 4 * Math.Atn(1)

    If geometryCollection.getGeometryTyp = "Line" Then
        Set m_lineCollection = geometryCollection
    End If

    If geometryCollection.getGeometryTyp = "Point" Then
        createLineFromPoints geometryCollection
    End If


End Sub

'Definition methodes

Public Function calculateLinearRegression() As limes_geometry.GeometryHashCollection

    If m_lineCollection.count = 0 Then
        Set calculateLinearRegression = m_lineCollection
        Exit Function
    End If

    Dim line_limes As limes_geometry.line

    For Each line_limes In m_lineCollection
        computeRegression line_limes
    Next

    Set calculateLinearRegression = m_lineCollection

End Function

Private Sub createLineFromPoints(PointCollection As limes_geometry.GeometryHashCollection)

    Set m_lineCollection = New limes_geometry.GeometryHashCollection
    m_lineCollection.GeometryCollection1 "Line"

    Dim line As limes_geometry.line
    Set line = New limes_geometry.line
    line.line1
    line.createLineInfo1
    line.m_iIdLine = 1

    If PointCollection.count >= 2 Then
        'Compute azimuth which is used for transformation the points to X-axis
        Dim point As limes_geometry.point
        Dim pointS As limes_geometry.point
        Dim pointE As limes_geometry.point
        Dim a As Integer
        a = 0
        For Each point In PointCollection
            If a = 0 Then
                Set pointS = point
            End If

            If a = 1 Then
                Set pointE = point
```

```
            End If

            a = a + 1
        Next

        line.Azimut = computeAzimut(pointS.m_dX, pointS.m_dY, pointE.m_dX, pointE.m_dY)
    End If

    Set line.m_pointCollection = PointCollection

    m_lineCollection.add line

End Sub

Private Function computeRegression(line_limes As limes_geometry.line) As limes_geometry.line

    '*************************************************************************
    'Set points of line (line_limes) to intern datastructure

    Dim dShiftX As Double
    Dim dShiftY As Double
    Dim dRotationAngle As Double
    Dim iPointCount As Integer
    iPointCount = line_limes.m_pointCollection.count

    ReDim iVecId(0 To iPointCount - 1) As Integer
    ReDim dVecX(0 To iPointCount - 1) As Double
    ReDim dVecY(0 To iPointCount - 1) As Double
    ReDim dVecP(0 To iPointCount - 1) As Double
    ReDim bVecCP(0 To iPointCount - 1) As Boolean
    ReDim dVecXg(0 To iPointCount - 1) As Double
    ReDim dVecYg(0 To iPointCount - 1) As Double
    ReDim dVecD(0 To iPointCount - 1) As Double

    Dim dTotalP As Double
    dTotalP = 0
    Dim dV_n As Integer
    dV_n = 0

    Dim point As limes_geometry.point

    Dim a As Integer
    a = 0
    For Each point In line_limes.m_pointCollection
        iVecId(a) = point.m_iIdPoint
        dVecX(a) = point.m_dX
        dVecY(a) = point.m_dY
        bVecCP(a) = point.Controlpoint
        If point.m_dAccuracy = 0 Then
            point.m_dAccuracy = 1
        End If
        dVecP(a) = 1 / (point.m_dAccuracy * point.m_dAccuracy)
        If point.Controlpoint = True Then
            dTotalP = dTotalP + dVecP(a)
            dV_n = dV_n + 1
        End If
        a = a + 1
    Next


    '*************************************************************************
    'Translation Points to Origin and rotate the Points to X- axis

    'Translation
    dShiftX = dVecX(0)
    dShiftY = dVecY(0)

    Dim d As Integer
    d = 0
    For d = 0 To iPointCount - 1
        dVecX(d) = dVecX(d) - dShiftX
        dVecY(d) = dVecY(d) - dShiftY
    Next d

    'Rotation
    dRotationAngle = pi / 2 - line_limes.Azimut
```

```
d = 0
For d = 0 To iPointCount - 1
    Dim dX As Double
    Dim dY As Double
    dX = dVecX(d)
    dY = dVecY(d)
    dVecX(d) = dX * Math.Cos(dRotationAngle) + dY * Math.Sin(dRotationAngle)
    dVecY(d) = dY * Math.Cos(dRotationAngle) - dX * Math.Sin(dRotationAngle)
Next d


'**************************************************************************
'Compute linear regression

'Reduction to center of gravity
Dim dSPointX As Double
Dim dSPointY As Double

Dim b As Integer
b = 0
For b = 0 To iPointCount - 1
    If bVecCP(b) = True Then
        dSPointX = dSPointX + (dVecX(b) * dVecP(b))
        dSPointY = dSPointY + (dVecY(b) * dVecP(b))
    End If
Next b

dSPointX = dSPointX / dTotalP
dSPointY = dSPointY / dTotalP

b = 0
For b = 0 To iPointCount - 1
        dVecX(b) = dVecX(b) - dSPointX
        dVecY(b) = dVecY(b) - dSPointY
Next b

'Compute gradient of the straight and segment of Y- axis

Dim dFunValuePhi As Double
Dim dPhi As Double
Dim dV_xTPy As Double
Dim dV_xTPx As Double
Dim dV_yTPy As Double

Dim c As Integer
c = 0
For c = 0 To iPointCount - 1
    If bVecCP(c) = True Then
        dV_xTPy = dV_xTPy + (dVecX(c) * dVecP(c) * dVecY(c))
        dV_xTPx = dV_xTPx + (dVecX(c) * dVecP(c) * dVecX(c))
        dV_yTPy = dV_yTPy + (dVecY(c) * dVecP(c) * dVecY(c))
    End If
Next c

dFunValuePhi = (2 * dV_xTPy) / (dV_xTPx - dV_yTPy)

dPhi = Math.Atn(dFunValuePhi) / 2

'y = ax +  b -> dA*x + dB*y + dC = 0 with dB = -1, dA = a and dC = b
Dim dA As Double
Dim dB As Double
Dim dC As Double

dA = Math.Tan(dPhi)
dB = -1
dC = 0

'Compute residuals d and coordinates (Xg, Yg)
'                  +P
'                  |d
'           -------+-------- "Straight"
'                   (Xg,Yg)

Dim f As Integer
f = 0
For f = 0 To iPointCount - 1
    Dim dNabs As Double
```

```
        dNabs = Math.Sqr(dA * dA + dB * dB)
        'Residual d of "Hessischer Normalform"
        dVecD(f) = (dA * dVecX(f) + dB * dVecY(f) + dC) / dNabs
        'Compute Point (Xg,Yg) up straight
        dVecXg(f) = dVecX(f) - dVecD(f) * (dA / dNabs)
        dVecYg(f) = dVecY(f) - dVecD(f) * (dB / dNabs)
    Next f

    'Precision analysis
    Dim dV_vTPv As Double
    dV_vTPv = 0
    Dim dMAbweichung As Double
    dMAbweichung = 0
    Dim dMf_messung As Double
    dMf_messung = 0
    Dim dMf_mittel As Double
    dMf_mittel = 0

    Dim g As Integer
    g = 0
    For g = 0 To iPointCount - 1
        If bVecCP(g) = True Then
            dV_vTPv = dV_vTPv + (dVecD(g) * dVecP(g) * dVecD(g))
            dMAbweichung = dMAbweichung + (Math.Abs(dVecD(g)) * dVecP(g))
        End If
    Next g

    dMAbweichung = dMAbweichung / dTotalP
    dMf_messung = Math.Sqr(dV_vTPv / (dV_n - 1))
    dMf_mittel = dMf_messung / Math.Sqr(dTotalP)

    'Sort points
    ReDim pointArray(0 To iPointCount - 1) As limes_geometry.point
    Dim h As Integer
    h = 0
    For h = 0 To iPointCount - 1
        Set pointArray(h) = New limes_geometry.point
        pointArray(h).point1
        pointArray(h).m_iIdPoint = iVecId(h)
        pointArray(h).m_dX = dVecXg(h)
        pointArray(h).m_dY = dVecYg(h)
        pointArray(h).createPointInfo1
        pointArray(h).DistanceToLine = dVecD(h)
    Next h

    Dim sort As limes_calculation.sort
    Set sort = New limes_calculation.sort
    sort.Sort1
    pointArray = sort.quicksort(pointArray)

    Dim newPointCollection As limes_geometry.GeometryHashCollection
    Set newPointCollection = New limes_geometry.GeometryHashCollection
    newPointCollection.GeometryCollection1 "Point"

    Dim dStartPointX As Double
    Dim dStartPointY As Double
    Dim dEndPointX As Double
    Dim dEndPointY As Double

    h = 0
    For h = 0 To iPointCount - 1
        Dim index As Integer
        Set point = line_limes.m_pointCollection.item(pointArray(h).m_iIdPoint)
        Set point = point.Clone
        point.m_iIdPoint = h
        point.DistanceToLine = pointArray(h).DistanceToLine
        If h < iPointCount - 1 Then
            'Distance to next point in the line
            Dim dDiffX As Double, dDiffY As Double
            dDiffX = pointArray(h).m_dX - pointArray(h + 1).m_dX
            dDiffY = pointArray(h).m_dY - pointArray(h + 1).m_dY
            point.DistanceNextPoint = Math.Sqr(dDiffX * dDiffX + dDiffY * dDiffY)
        End If
        newPointCollection.add point

        If h = 0 Then
            dStartPointX = pointArray(h).m_dX
```

```
            dStartPointY = pointArray(h).m_dY
        End If
        If h = iPointCount - 1 Then
            dEndPointX = pointArray(h).m_dX
            dEndPointY = pointArray(h).m_dY
        End If
    Next h

    'Inverse reduction of center of gravity of start- and endpoint of line

    dStartPointX = dStartPointX + dSPointX
    dStartPointY = dStartPointY + dSPointY
    dEndPointX = dEndPointX + dSPointX
    dEndPointY = dEndPointY + dSPointY



    '*************************************************************************
    'Translate return to Global coordinates

    'Rotation inverse
    dX = dStartPointX
    dY = dStartPointY
    dStartPointX = dX * Math.Cos(dRotationAngle) - dY * Math.Sin(dRotationAngle)
    dStartPointY = dY * Math.Cos(dRotationAngle) + dX * Math.Sin(dRotationAngle)
    dX = dEndPointX
    dY = dEndPointY
    dEndPointX = dX * Math.Cos(dRotationAngle) - dY * Math.Sin(dRotationAngle)
    dEndPointY = dY * Math.Cos(dRotationAngle) + dX * Math.Sin(dRotationAngle)


    'Translation inverse
    dStartPointX = dStartPointX + dShiftX
    dStartPointY = dStartPointY + dShiftY
    dEndPointX = dEndPointX + dShiftX
    dEndPointY = dEndPointY + dShiftY


    '*************************************************************************
    'Set parameters of new regression line

    'PointCollection
    Set line_limes.m_pointCollection = newPointCollection

    'Set start and end point
    Dim startPoint As limes_geometry.point
    Dim endPoint As limes_geometry.point
    Set startPoint = New limes_geometry.point
    Set endPoint = New limes_geometry.point
    startPoint.point1
    endPoint.point1

    startPoint.m_dX = dStartPointX
    startPoint.m_dY = dStartPointY
    endPoint.m_dX = dEndPointX
    endPoint.m_dY = dEndPointY

    Set line_limes.m_startPoint = startPoint
    Set line_limes.m_EndPoint = endPoint

    'Set point count up this line
    line_limes.PointCount = iPointCount
    'Set accuracy
    line_limes.Accuracy = dMAbweichung
    'Set azimut
    Dim dAzimut As Double
    dAzimut = computeAzimut(dStartPointX, dStartPointY, dEndPointX, dEndPointY)
    line_limes.Azimut = (dAzimut / pi) * 180
    'Set line length
    line_limes.LineLength=Math.Sqr((dStartPointX - dEndPointX) * (dStartPointX - dEndPointX) _
            + (dStartPointY - dEndPointY) * (dStartPointY - dEndPointY))

    Set computeRegression = line_limes

End Function
```

```
Private Function computeAzimut _
        (pointS_X As Double,pointS_Y As Double,pointE_X As Double,pointE_Y As Double) As _
        Double

    Dim dDeltaX As Double
    Dim dDeltaY As Double
    dDeltaX = pointE_X - pointS_X
    dDeltaY = pointE_Y - pointS_Y

    Dim dFunValue As Double
    Dim dAzimut As Double

    If Not dDeltaY = 0 Then
        dFunValue = dDeltaX / dDeltaY

        If dFunValue >= 0 Then
            If dDeltaY > 0 Then
                dAzimut = Math.Atn(dFunValue)
            ElseIf dDeltaY < 0 Then
                dAzimut = pi + Math.Atn(dFunValue)
            End If
        ElseIf dFunValue <= 0 Then
            If dDeltaY > 0 Then
                dAzimut = 2 * pi + Math.Atn(dFunValue)
            ElseIf dDeltaY < 0 Then
                dAzimut = pi + Math.Atn(dFunValue)
            End If
        Else
            If dDeltaY > 0 Then
                dAzimut = 0
            ElseIf dDeltaY < 0 Then
                dAzimut = pi
            End If
        End If
    Else
        If dDeltaX > 0 Then
            dAzimut = pi / 2
        ElseIf dDeltaX < 0 Then
            dAzimut = (3 * pi) / 2
        End If
    End If

    computeAzimut = dAzimut

End Function
```

Klasse: Sort

```
'Class: limes_calculation.Sort
'Extends:

Option Explicit

'**********************************************************************
'Definition membervariables

'Definition constructors
Public Sub Sort1()

End Sub

'Definition methodes

Public Function quicksort(pointArray() As limes_geometry.point) _
        As limes_geometry.point()
    Call qsort(pointArray, 0, UBound(pointArray))
    quicksort = pointArray
End Function

Private Sub qsort(pointArray() As limes_geometry.point, iLe As Integer, iRi As Integer)

    Dim iLo As Integer, iHi As Integer
    iLo = iLe
    iHi = iRi

    If iHi > iLo Then

        Dim pMid As limes_geometry.point
        Set pMid = pointArray((iLo + iHi) / 2)
        While iLo <= iHi
            While iLo < iRi And pointArray(iLo).m_dX < pMid.m_dX
                iLo = iLo + 1
            Wend
            While iHi > iLe And pointArray(iHi).m_dX > pMid.m_dX
                iHi = iHi - 1
            Wend
            If iLo <= iHi Then
                Call swap(pointArray, iLo, iHi)
                iLo = iLo + 1
                iHi = iHi - 1
            End If
        Wend

        If iLe < iHi Then
            Call qsort(pointArray, iLe, iHi)
        End If
        If iLo < iRi Then
            Call qsort(pointArray, iLo, iRi)
        End If

    End If

End Sub

Private Sub swap(pointArray() As limes_geometry.point, iIdx1 As Integer, iIdx2 As Integer)

    Dim pointTmp As limes_geometry.point
    Set pointTmp = pointArray(iIdx1)
    Set pointArray(iIdx1) = pointArray(iIdx2)
    Set pointArray(iIdx2) = pointTmp

End Sub
```

Dynamic Link Library: limes_arcObj

Klasse: DataAccess

```
'Class: limes_arcObj.DataAccess
'Extends:

Option Explicit

'*********************************************************************
'Definition membervariables

Private m_application As esriCore.IApplication
Private m_mxDoc As esriCore.IMxDocument
Private m_map As esriCore.IMap
Private m_maps As esriCore.IMaps
Private m_convGeomEsriLimes As limes_arcObj.ConvGeomEsriLimes

'Definition constructors

Public Sub DataAccess1(application As esriCore.IApplication)
    Set m_application = application
    Set m_mxDoc = m_application.Document
    Set m_maps = m_mxDoc.Maps
    Set m_map = m_maps.Item(0)

    Set m_convGeomEsriLimes = New limes_arcObj.ConvGeomEsriLimes
    m_convGeomEsriLimes.ConvGeomEsriLimes1

End Sub

'Definition methodes

Public Function getMXDPath() As String

    Dim templates As ITemplates
    Dim lCount As Long
    Dim path As String

    Set templates = m_application.templates
    lCount = templates.Count
    path = templates.Item(lCount - 1)

    Dim lPosition As Long
    lPosition = Strings.InStrRev(path, "\")
    path = Strings.Left(path, lPosition)

    getMXDPath = path

End Function

Public Function getSelectedPoints(bAskAccuracy As Boolean) As _
       limes_geometry.GeometryHashCollection

    Dim enumLayer As esriCore.IEnumLayer
    Dim layer As esriCore.ILayer
    Dim uID As esriCore.IUID
    Dim featureSelection As esriCore.IFeatureSelection
    Dim selectionSet As esriCore.ISelectionSet
    Dim featureCursor As esriCore.IFeatureCursor

    Dim pointCollection As limes_geometry.GeometryHashCollection
    Set pointCollection = New limes_geometry.GeometryHashCollection
    pointCollection.GeometryCollection1 "Point"

    Set uID = New esriCore.uID
    uID.Value = "{E156D7E5-22AF-11D3-9F99-00C04F6BC78E}"
    Set enumLayer = m_map.Layers(uID, True)

    Dim i As Integer
    i = 1
```

```
    enumLayer.Reset
    Set layer = enumLayer.Next
    Do While Not layer Is Nothing
        Set featureSelection = layer
        Set selectionSet = featureSelection.selectionSet
        If selectionSet.Count > 0 Then
            Dim dAccuracy As Double
            dAccuracy = 0
            If bAskAccuracy = True Then
                AccuracyDialog.label_2.Caption = layer.Name
                AccuracyDialog.Show vbModal
                dAccuracy = AccuracyDialog.text_1.Text
                Unload AccuracyDialog
            End If

            selectionSet.Search Nothing, False, featureCursor

            'Read feature geometry
            Dim feature As IFeature
            Set feature = featureCursor.NextFeature

            Do While Not feature Is Nothing
                Dim geometry As IGeometry
                Set geometry = feature.Shape

                If geometry.GeometryType = esriGeometryPoint Then

                    Dim point_esri As esriCore.IPoint
                    Set point_esri = geometry

                    Dim point_limes As limes_geometry.point

                    Set point_limes = m_convGeomEsriLimes.createLimesPoint(point_esri)

                    point_limes.m_iIdPoint = i
                    point_limes.m_dAccuracy = dAccuracy
                    point_limes.Geometry_SourceOID = getOBJECTID(feature)
                    point_limes.Geometry_SourceLayer = layer.Name

                    pointCollection.Add point_limes
                    i = i + 1

                End If

                Set feature = featureCursor.NextFeature
            Loop
        End If

        Set layer = enumLayer.Next
    Loop

    Set getSelectedPoints = pointCollection

End Function

Public Function getPointsFromSelectedLines() As limes_geometry.GeometryHashCollection

    Dim pointFeatureSelection As esriCore.IFeatureSelection
    Dim pointSelectionSet As esriCore.ISelectionSet
    Dim pointFeatureCursor As esriCore.IFeatureCursor
    Dim featureSelection As esriCore.IFeatureSelection
    Dim selectionSet As esriCore.ISelectionSet
    Dim featureCursor As esriCore.IFeatureCursor
    Dim i As Integer
    i = 1

    Dim pointCollection As limes_geometry.GeometryHashCollection
    Set pointCollection = New limes_geometry.GeometryHashCollection
    pointCollection.GeometryCollection1 "Point"

    Set pointFeatureSelection = getLayer("Polar_Punkte")
    If pointFeatureSelection Is Nothing Then
        MsgBox ("Layer Polar_Punkte is not available !")
        Set getPointsFromSelectedLines = pointCollection
        Exit Function
    End If
```

```
    Set featureSelection = getLayer("Polar_Linien")
    If featureSelection Is Nothing Then
        MsgBox ("Layer Polar_Linien is not available !")
        Set getPointsFromSelectedLines = pointCollection
        Exit Function
    End If

    Set selectionSet = featureSelection.selectionSet
    If selectionSet.Count > 0 Then

        selectionSet.Search Nothing, False, featureCursor

        'Arribute name of IdRef_linie col of relation Polar_Punkte
        Dim sIdRef_linie As String
        sIdRef_linie = getAttrNameOfIdRefLinie

        'Read feature geometry
        Dim feature As IFeature
        Set feature = featureCursor.NextFeature

        Do While Not feature Is Nothing

            'Select points of actual line
            Dim lIdLine As Long
            lIdLine = getOBJECTID(feature)

            Dim queryFilter As esriCore.IQueryFilter
            Set queryFilter = New esriCore.queryFilter

            'queryFilter.WhereClause = "IdRef_linie = " & lIdLine
            queryFilter.WhereClause = sIdRef_linie & " = " & lIdLine

            Dim orgSelectionSet As esriCore.ISelectionSet
            Set orgSelectionSet = pointFeatureSelection.selectionSet
            pointFeatureSelection.SelectFeatures queryFilter, esriSelectionResultNew, False
            Set pointSelectionSet = pointFeatureSelection.selectionSet
            If pointSelectionSet.Count > 0 Then
                pointSelectionSet.Search Nothing, False, pointFeatureCursor

                'Read feature geometry
                Dim pointFeature As IFeature
                Set pointFeature = pointFeatureCursor.NextFeature

                    Do While Not pointFeature Is Nothing
                        Dim geometry As IGeometry
                        Set geometry = pointFeature.Shape

                        If geometry.GeometryType = esriGeometryPoint Then

                            Dim point_esri As esriCore.IPoint
                            Set point_esri = geometry

                            Dim point_limes As limes_geometry.point

                            Set point_limes = m_convGeomEsriLimes.createLimesPoint(point_esri)
                            point_limes.createPointInfo1
                            point_limes.m_iIdPoint = i

                            Dim fields As esriCore.IFields
                            Dim field As esriCore.IField
                            Set fields = pointFeature.fields

                            Dim a As Integer
                            a = 0
                            For a = 0 To (fields.FieldCount - 1)
                                Set field = fields.field(a)

                                If field.Name = "Bemerkung" Then
                                    point_limes.Comment = pointFeature.Value(a)
                                End If

                                If field.Name = "Festpunkt" Then
                                    If pointFeature.Value(a) = "Ja" Then
                                        point_limes.Controlpoint = True
                                    Else
                                        point_limes.Controlpoint = False
                                    End If
```

```
                                End If

                                If field.Name = "Punktgenauigkeit" Then
                                    point_limes.m_dAccuracy = pointFeature.Value(a)
                                End If

                                If field.Name = "Source_OID" Then
                                    point_limes.Geometry_SourceOID = pointFeature.Value(a)
                                End If

                                If field.Name = "Source_FeatureClass" Then
                                    point_limes.Geometry_SourceLayer = pointFeature.Value(a)
                                End If

                            Next a

                            pointCollection.Add point_limes
                            i = i + 1

                        End If

                        Set pointFeature = pointFeatureCursor.NextFeature
                    Loop

                Set pointFeatureSelection.selectionSet = orgSelectionSet

            End If

            Set feature = featureCursor.NextFeature
        Loop
    End If

    Set getPointsFromSelectedLines = pointCollection

End Function


Public Function writeLineData(lineCollection As limes_geometry.GeometryHashCollection) As
Boolean

    Dim featureLayer As esriCore.ILayer
    Dim editLayers As esriCore.IEditLayers
    Dim editor As esriCore.IEditor
    Dim editorID As New esriCore.uID
    Dim invalidArea As esriCore.IInvalidArea
    Dim sMDBPath As String
    Dim workspaceFactory As esriCore.IWorkspaceFactory
    Dim workspace As esriCore.IWorkspace
    Dim featureClass As esriCore.IFeatureClass

    If lineCollection Is Nothing Then
        writeLineData = False
        Exit Function
    End If

    If Not lineCollection.getGeometryTyp = "Line" Then
        writeLineData = False
        Exit Function
    End If

    On Error GoTo ErrorHandler

    'Get a handle to the editor extension
    editorID = "esriCore.Editor"
    Set editor = m_application.FindExtensionByCLSID(editorID)
    Set editLayers = editor 'QI

    'Start editing

    sMDBPath = Me.getMXDPath & "limes.mdb"
    Set workspaceFactory = New esriCore.AccessWorkspaceFactory
    Set workspace = workspaceFactory.OpenFromFile(sMDBPath, 0)

    editor.StartEditing workspace

    Set invalidArea = New invalidArea
    Set invalidArea.Display = editor.Display
```

```
'Start operation
editor.StartOperation

'Write to Polar_Linien featureClass
'Set Polar_Linien to active Layer
Dim i As Integer
For i = 0 To m_map.LayerCount - 1
    If m_map.layer(i).Name = "Polar_Linien" Then
        Set featureLayer = m_map.layer(i)
    End If
Next i

editLayers.SetCurrentLayer featureLayer, 0


Set featureClass = editLayers.CurrentLayer.featureClass

If Not featureClass.ShapeType = esriGeometryPolyline Then
    writeLineData = False
    Exit Function
End If

Dim line_limes As limes_geometry.Line

For Each line_limes In lineCollection

    writePolarLine featureClass, line_limes, invalidArea

Next



'Write to Polar_Punkte featureClass
'Set Polar_Linien to active Layer
i = 0
For i = 0 To m_map.LayerCount - 1
    If m_map.layer(i).Name = "Polar_Punkte" Then
        Set featureLayer = m_map.layer(i)
    End If
Next i

editLayers.SetCurrentLayer featureLayer, 0


Set featureClass = editLayers.CurrentLayer.featureClass

If Not featureClass.ShapeType = esriGeometryPoint Then
    writeLineData = False
    Exit Function
End If

For Each line_limes In lineCollection

    If Not line_limes.m_pointCollection Is Nothing Then

        Dim point_limes As limes_geometry.point
        Dim iCount As Integer, iCounter As Integer
        iCount = line_limes.m_pointCollection.Count
        iCounter = 1
        For Each point_limes In line_limes.m_pointCollection
            If iCounter < iCount Then
                writePolarPoint featureClass, point_limes, invalidArea, True
            End If
            If iCounter = iCount Then
                writePolarPoint featureClass, point_limes, invalidArea, False
            End If
            iCounter = iCounter + 1
        Next

    End If

Next


editor.StopOperation ("Write Polar Data")
```

```
    editor.StopEditing True


    'Refresh the display
    invalidArea.Invalidate esriAllScreenCaches

    writeLineData = True
    Exit Function

ErrorHandler:
    writeLineData = False

End Function

Private Sub writePolarLine( _
        featureClass As esriCore.IFeatureClass, _
        line_limes As limes_geometry.Line, _
        invalidArea As esriCore.IInvalidArea)

    'Create instance
    Dim newFeature As esriCore.IFeature
    Set newFeature = featureClass.CreateFeature

    'Create geometry
    Dim polyline_esri As esriCore.IPolyline
    Set polyline_esri = m_convGeomEsriLimes.createEsriPolyline(line_limes)
    Set newFeature.Shape = polyline_esri

    'Create thematic data
    Dim fields As esriCore.IFields
    Dim field As esriCore.IField
    Set fields = newFeature.fields

    Dim i As Long
    i = 0
    For i = 0 To (fields.FieldCount - 1)
        Set field = fields.field(i)

        If field.Name = "OBJECTID" Then
            line_limes.m_iIdLine = newFeature.Value(i)

            Dim point As limes_geometry.point
            For Each point In line_limes.m_pointCollection
                If point.m_pointInfo Is Nothing Then
                    point.createPointInfo1
                End If
                point.IdRefLine = line_limes.m_iIdLine
            Next

            Exit For
        End If
    Next i

    If Not line_limes.m_lineInfo Is Nothing Then

        i = 0
        For i = 0 To (fields.FieldCount - 1)
            Set field = fields.field(i)

            If field.Name = "IdRef_pol" Then
                newFeature.Value(i) = line_limes.IdRefPole
            End If

            If field.Name = "AnzPunkte" Then
                newFeature.Value(i) = line_limes.PointCount
            End If

            If field.Name = "Azimut" Then
                newFeature.Value(i) = line_limes.Azimut
            End If

            If field.Name = "Linienlänge" Then
                newFeature.Value(i) = line_limes.LineLength
            End If

            If field.Name = "MittlereAbweichung" Then
                newFeature.Value(i) = line_limes.Accuracy
```

```
            End If

            If field.Name = "Deutung" Then
                newFeature.Value(i) = line_limes.Interpretation
            End If

            If field.Name = "Bemerkung" Then
                newFeature.Value(i) = line_limes.Comment
            End If

        Next i

    End If


    newFeature.Store
    invalidArea.Add newFeature

End Sub

Private Sub writePolarPoint( _
        featureClass As esriCore.IFeatureClass, _
        point_limes As limes_geometry.point, _
        invalidArea As esriCore.IInvalidArea, _
        bHasNextPoint As Boolean)

    'Create instance
    Dim newFeature As esriCore.IFeature
    Set newFeature = featureClass.CreateFeature

    'Create geometry
    Dim point_esri As esriCore.IPoint
    Set point_esri = m_convGeomEsriLimes.createEsriPoint(point_limes)
    Set newFeature.Shape = point_esri
    Dim lOBJECTID As Long

    'Create thematic data
    Dim fields As esriCore.IFields
    Dim field As esriCore.IField
    Set fields = newFeature.fields

    Dim i As Long
    i = 0
    For i = 0 To (fields.FieldCount - 1)
        Set field = fields.field(i)

        If field.Name = "OBJECTID" Then
            lOBJECTID = newFeature.Value(i)
        End If

        If field.Name = "Punktgenauigkeit" Then
            newFeature.Value(i) = point_limes.m_dAccuracy
        End If
    Next i


    If Not point_limes.m_pointInfo Is Nothing Then

        i = 0
        For i = 0 To (fields.FieldCount - 1)
            Set field = fields.field(i)

            If field.Name = "IdRef_pol" Then
                newFeature.Value(i) = point_limes.IdRefPole
            End If

            If field.Name = "IdRef_linie" Then
                newFeature.Value(i) = point_limes.IdRefLine
            End If

            If field.Name = "NextPunkt" Then
                If bHasNextPoint = True Then
                    newFeature.Value(i) = lOBJECTID + 1
                Else
                    newFeature.Value(i) = lOBJECTID
                End If
            End If
```

```
            If field.Name = "AbstandNextPunkt" Then
                newFeature.Value(i) = point_limes.DistanceNextPoint
            End If

            If field.Name = "Abstand" Then
                newFeature.Value(i) = point_limes.DistanceToLine
            End If

            If field.Name = "Bemerkung" Then
                newFeature.Value(i) = point_limes.Comment
            End If

            If field.Name = "Festpunkt" Then
                If point_limes.Controlpoint = True Then
                    newFeature.Value(i) = "Ja"
                Else
                    newFeature.Value(i) = "Nein"
                End If
            End If

            If field.Name = "Source_OID" Then
                newFeature.Value(i) = point_limes.Geometry_SourceOID
            End If

            If field.Name = "Source_FeatureClass" Then
                newFeature.Value(i) = point_limes.Geometry_SourceLayer
            End If

        Next i

    End If

    newFeature.Store
    invalidArea.Add newFeature

End Sub

Private Function getOBJECTID(feature As esriCore.IFeature) As Long

    Dim lOBJECTID As Long
    Dim fields As esriCore.IFields
    Dim field As esriCore.IField
    Set fields = feature.fields

    Dim i As Long
    i = 0
    For i = 0 To (fields.FieldCount - 1)
        Set field = fields.field(i)

        If field.Name = "OBJECTID" Then
            lOBJECTID = feature.Value(i)
            Exit For
        End If
    Next i

    getOBJECTID = lOBJECTID
    Exit Function

End Function

Private Function getLayer(layerName As String) As esriCore.ILayer

        Dim enumLayer As esriCore.IEnumLayer
        Dim layer As esriCore.ILayer
        Dim uID As esriCore.IUID

        Set uID = New esriCore.uID
        uID.Value = "{E156D7E5-22AF-11D3-9F99-00C04F6BC78E}"
        Set enumLayer = m_map.Layers(uID, True)

        enumLayer.Reset
        Set layer = enumLayer.Next
        Do While Not layer Is Nothing
            If layer.Name = layerName Then

                Set getLayer = layer
```

```
                Exit Function

            End If
            Set layer = enumLayer.Next
        Loop

        Set getLayer = Nothing

End Function

Private Function getAttrNameOfIdRefLinie() As String

    Dim pDispTable As IDisplayTable
    Dim pTable As ITable
    Set pDispTable = getLayer("Polar_Punkte")
    Set pTable = pDispTable.DisplayTable

    'Test is relation Polar_Punkte joined
    If Not TypeOf pTable Is IRelQueryTable Then
        getAttrNameOfIdRefLinie = "IdRef_linie"
        Exit Function
    End If

    If TypeOf pTable Is IRelQueryTable Then
        getAttrNameOfIdRefLinie = "Polar_Punkte.IdRef_linie"
        Exit Function
    End If


End Function
```

Klasse: ConvGeomEsriLimes

```
'Class: limes_arcObj.ConvGeomEsriLimes
'Extends:

Option Explicit

'*********************************************************************
'Definition membervariables

'Definition constructors

Public Sub ConvGeomEsriLimes1()

End Sub

'Definition methodes

Public Function createLimesPoint(point_esri As esriCore.IPoint) As limes_geometry.point

    Dim point_limes As limes_geometry.point
    Set point_limes = New limes_geometry.point
    point_limes.point1

    Dim dX As Double, dY As Double
    point_esri.QueryCoords dX, dY

    point_limes.m_dX = dX
    point_limes.m_dY = dY
    point_limes.m_dZ = 0

    Set createLimesPoint = point_limes

End Function

Public Function createLimesLine(line_esri As esriCore.ILine) As limes_geometry.Line

    Dim line_limes As limes_geometry.Line
    Set line_limes = New limes_geometry.Line
    line_limes.Line1

    Dim pStart_esri As esriCore.IPoint
    Dim pEnd_esri As esriCore.IPoint
    Set pStart_esri = New esriCore.point
    Set pEnd_esri = New esriCore.point

    line_esri.QueryCoords pStart_esri, pEnd_esri

    Dim pStart_limes As limes_geometry.point
    Set pStart_limes = createLimesPoint(pStart_esri)

    Dim pEnd_limes As limes_geometry.point
    Set pEnd_limes = createLimesPoint(pStart_esri)

    line_limes.m_startPoint = pStart_limes
    line_limes.m_endPoint = pEnd_limes

    Set createLimesLine = line_limes

End Function

Public Function createEsriPoint(point_limes As limes_geometry.point) As esriCore.IPoint

    Dim point_esri As esriCore.IPoint
    Set point_esri = New esriCore.point
    point_esri.PutCoords point_limes.m_dX, point_limes.m_dY

    Set createEsriPoint = point_esri

End Function

Public Function createEsriLine(line_limes As limes_geometry.Line) As esriCore.ILine

    Dim line_esri As esriCore.ILine
    Set line_esri = New esriCore.Line
```

```
    Dim pStart_esri As esriCore.IPoint
    Set pStart_esri = createEsriPoint(line_limes.m_startPoint)

    Dim pEnd_esri As esriCore.IPoint
    Set pEnd_esri = createEsriPoint(line_limes.m_endPoint)

    line_esri.PutCoords pStart_esri, pEnd_esri

    Set createEsriLine = line_esri

End Function

Public Function createEsriPolyline(line_limes As limes_geometry.Line) As esriCore.IPolyline

    Dim line_esri As esriCore.ILine
    Set line_esri = createEsriLine(line_limes)

    Dim polyline_esri As esriCore.ISegmentCollection
    Set polyline_esri = New esriCore.Polyline
    polyline_esri.AddSegment line_esri

    Set createEsriPolyline = polyline_esri

End Function
```

Dynamic Link Library: fun_detectLine

Klasse: DetectLine

```
Option Explicit

Implements ICommand
Implements ITool

Private m_pApp As IApplication

Private Property Get ICommand_Bitmap() As esriCore.OLE_HANDLE
  'ICommand_Bitmap = frmResources.imlBitmaps.ListImages(3).Picture
End Property

Private Property Get ICommand_Caption() As String
  ICommand_Caption = "Detect Lines automaticaly"
End Property

Private Property Get ICommand_Category() As String
  ICommand_Category = "Limes"
End Property

Private Property Get ICommand_Checked() As Boolean
  ICommand_Checked = False
End Property

Private Property Get ICommand_Enabled() As Boolean
  ICommand_Enabled = True
End Property

Private Property Get ICommand_HelpContextID() As Long
  ' No help implemented for this tool
End Property

Private Property Get ICommand_HelpFile() As String
  ' No help implemented for this tool
End Property

Private Property Get ICommand_Message() As String
  ICommand_Message = "Detect Lines automaticaly"
End Property

Private Property Get ICommand_Name() As String
  ICommand_Name = "Detect Lines automaticaly"
End Property

Private Sub ICommand_OnClick()

    If DetectLineForm.Visible = False Then
        DetectLineForm.Show
    End If

End Sub

Private Sub ICommand_OnCreate(ByVal hook As Object)
  Set m_pApp = hook
  Set DetectLineForm.m_pApp = m_pApp
End Sub

Private Property Get ICommand_Tooltip() As String
  ICommand_Tooltip = "DetectLine"
End Property

Private Property Get ITool_Cursor() As esriCore.OLE_HANDLE

End Property

Private Function ITool_Deactivate() As Boolean

    ITool_Deactivate = True
End Function
```

```
Private Function ITool_OnContextMenu(ByVal X As Long, ByVal Y As Long) As Boolean

End Function

Private Sub ITool_OnDblClick()

End Sub

Private Sub ITool_OnKeyDown(ByVal keyCode As Long, ByVal Shift As Long)

End Sub

Private Sub ITool_OnKeyUp(ByVal keyCode As Long, ByVal Shift As Long)

End Sub

Private Sub ITool_OnMouseDown(ByVal Button As Long, ByVal Shift As Long, ByVal X As Long,
ByVal Y As Long)

End Sub

Private Sub ITool_OnMouseMove(ByVal Button As Long, ByVal Shift As Long, ByVal X As Long,
ByVal Y As Long)

End Sub

Private Sub ITool_OnMouseUp(ByVal Button As Long, ByVal Shift As Long, ByVal X As Long, ByVal
Y As Long)

End Sub

Private Sub ITool_Refresh(ByVal hDC As esriCore.OLE_HANDLE)

End Sub
```

Formular: DetectLineForm

```
Option Explicit

Public m_pApp As IApplication

Private m_dMaxAbstand As Double
Private m_dMinAbstand As Double
Private m_dQuerTol As Double
Private m_dAngle As Double

Private Sub button_cancel_Click(Index As Integer)
    Unload Me
End Sub

Private Sub button_run_Click(Index As Integer)

    'Create esri-access object
    Dim dataAccess As limes_arcObj.dataAccess
    Set dataAccess = New dataAccess
    dataAccess.DataAccess1 m_pApp

    'Create Log- File
    Dim fso
    Dim logFile
    Dim logFilePath As String
    logFilePath = dataAccess.getMXDPath
    Set fso = CreateObject("Scripting.FileSystemObject")
    Set logFile = fso.CreateTextFile(logFilePath & "limes.log", True)

    'Read values in formular
    m_dMaxAbstand = text_field(0).Text
    m_dQuerTol = text_field(2).Text
    m_dAngle = text_field(3).Text

    'Read actual selectet points
    Dim pointCollection As limes_geometry.GeometryHashCollection
    Set pointCollection = dataAccess.getSelectedPoints(True)
    If pointCollection.Count = 0 Then
        MsgBox "No points selected !"
        Exit Sub
    End If

    'Detect lines
    Dim lineCollection As limes_geometry.GeometryHashCollection

    Dim detectLine As limes_calculation.DetectLineF
    Set detectLine = New limes_calculation.DetectLineF
    detectLine.DetectLine1 pointCollection, logFile
    Set lineCollection = detectLine.detectLine(m_dMaxAbstand, m_dMinAbstand, m_dQuerTol,
m_dAngle)

    'Compute regression lines
    Dim linRegression As limes_calculation.LinearRegression
    Set linRegression = New limes_calculation.LinearRegression
    linRegression.LinearRegression1 lineCollection

    Set lineCollection = linRegression.calculateLinearRegression

    'Write data to database
    Dim bTest As Boolean
    bTest = dataAccess.writeLineData(lineCollection)

    MsgBox "DetectLine successfully"

End Sub
```

Dynamic Link Library: fun_buildStraight

Klasse: BuildStraight

```
Option Explicit

Implements ICommand
Implements ITool

Private m_pApp As IApplication

Private Property Get ICommand_Bitmap() As esriCore.OLE_HANDLE
  'ICommand_Bitmap = frmResources.imlBitmaps.ListImages(3).Picture
End Property

Private Property Get ICommand_Caption() As String
  ICommand_Caption = "Build regression straight"
End Property

Private Property Get ICommand_Category() As String
  ICommand_Category = "Limes"
End Property

Private Property Get ICommand_Checked() As Boolean
  ICommand_Checked = False
End Property

Private Property Get ICommand_Enabled() As Boolean
  ICommand_Enabled = True
End Property

Private Property Get ICommand_HelpContextID() As Long
  ' No help implemented for this tool
End Property

Private Property Get ICommand_HelpFile() As String
  ' No help implemented for this tool
End Property

Private Property Get ICommand_Message() As String
  ICommand_Message = "Build regression straight"
End Property

Private Property Get ICommand_Name() As String
  ICommand_Name = "Build regression straight"
End Property

Private Sub ICommand_OnClick()

    If BuildStraightForm.Visible = False Then
        BuildStraightForm.BuildStraightForm1
        BuildStraightForm.Show
    End If

End Sub

Private Sub ICommand_OnCreate(ByVal hook As Object)
  Set m_pApp = hook
  Set BuildStraightForm.m_pApp = m_pApp
End Sub

Private Property Get ICommand_Tooltip() As String
  ICommand_Tooltip = "BuildStraight"
End Property

Private Property Get ITool_Cursor() As esriCore.OLE_HANDLE

End Property

Private Function ITool_Deactivate() As Boolean

    ITool_Deactivate = True
```

```
End Function

Private Function ITool_OnContextMenu(ByVal X As Long, ByVal Y As Long) As Boolean

End Function

Private Sub ITool_OnDblClick()

End Sub

Private Sub ITool_OnKeyDown(ByVal keyCode As Long, ByVal Shift As Long)

End Sub

Private Sub ITool_OnKeyUp(ByVal keyCode As Long, ByVal Shift As Long)

End Sub

Private Sub ITool_OnMouseDown(ByVal Button As Long, ByVal Shift As Long, ByVal X As Long,
ByVal Y As Long)

End Sub

Private Sub ITool_OnMouseMove(ByVal Button As Long, ByVal Shift As Long, ByVal X As Long,
ByVal Y As Long)

End Sub

Private Sub ITool_OnMouseUp(ByVal Button As Long, ByVal Shift As Long, ByVal X As Long, ByVal
Y As Long)

End Sub

Private Sub ITool_Refresh(ByVal hDC As esriCore.OLE_HANDLE)

End Sub
```

Formular: BuildStraightForm

```
Option Explicit

Public m_pApp As IApplication
Private m_listPointCollection As limes_geometry.GeometryHashCollection
Private m_iCount As Integer

Public Sub BuildStraightForm1()

    Set m_listPointCollection = New limes_geometry.GeometryHashCollection
    m_listPointCollection.GeometryCollection1 "Point"
    m_iCount = 1

    listView.View = lvwReport

    'Create column names
    Dim colHeader1 As ColumnHeader
    Dim colHeader2 As ColumnHeader
    Dim colHeader3 As ColumnHeader
    Dim colHeader4 As ColumnHeader

    Set colHeader1 = listView.ColumnHeaders.Add()
    Set colHeader2 = listView.ColumnHeaders.Add()
    Set colHeader3 = listView.ColumnHeaders.Add()
    Set colHeader4 = listView.ColumnHeaders.Add()
    colHeader1.Text = "Control Point"
    colHeader1.Width = listView.Width / 4
    colHeader2.Text = "Point Id"
    colHeader2.Width = listView.Width / 4
    colHeader3.Text = "Source Layer"
    colHeader3.Width = listView.Width / 4
    colHeader4.Text = "Accuracy [m]"
    colHeader4.Width = listView.Width / 4
End Sub


Private Sub button_addLines_Click()

    Dim esriAccess As limes_arcObj.DataAccess
    Set esriAccess = New limes_arcObj.DataAccess
    esriAccess.dataAccess1 m_pApp

    Dim pointCollection As limes_geometry.GeometryHashCollection
    Set pointCollection = esriAccess.getPointsFromSelectedLines

    addPointCollection pointCollection

End Sub

Private Sub button_addPoints_Click()

    Dim esriAccess As limes_arcObj.DataAccess
    Set esriAccess = New limes_arcObj.DataAccess
    esriAccess.dataAccess1 m_pApp

    Dim pointCollection As limes_geometry.GeometryHashCollection
    Set pointCollection = esriAccess.getSelectedPoints(False)

    addPointCollection pointCollection
    listView.SelectedItem = Nothing

End Sub

Private Sub button_cancel_Click()

    Unload Me

End Sub

Private Sub button_delete_Click()

    Dim item As ListItem
    Set item = listView.SelectedItem
```

```
    If Not item Is Nothing Then
        If item.Selected Then
            Dim iKey As Integer
            iKey = Strings.Left(item.key, Strings.Len(item.key) - 2)
            m_listPointCollection.Remove iKey
            listView.ListItems.Remove item.Index
        End If
    End If

End Sub

Private Sub button_ok_Click()

    'Compute regression lines
    Dim lineRegression As limes_calculation.LinearRegression
    Set lineRegression = New limes_calculation.LinearRegression

    lineRegression.LinearRegression1 m_listPointCollection
    Dim lineCollection As limes_geometry.GeometryHashCollection
    Set lineCollection = lineRegression.calculateLinearRegression

    'Write data to database
    Dim dataAccess1 As limes_arcObj.DataAccess
    Set dataAccess1 = New limes_arcObj.DataAccess
    dataAccess1.dataAccess1 m_pApp

    Dim bTest As Boolean
    bTest = dataAccess1.writeLineData(lineCollection)

    MsgBox "BuildLine successfully"

End Sub

Private Sub button_set_Click()
    Dim point_limes As limes_geometry.Point
    Dim item As ListItem
    Set item = listView.SelectedItem
    If Not item Is Nothing Then
        If item.Selected Then
            Dim iKey As Integer
            iKey = Strings.Left(item.key, Strings.Len(item.key) - 2)
            Set point_limes = m_listPointCollection.item(iKey)

            AccuracyDialog.label_2.Caption = point_limes.Geometry_SourceOID
            AccuracyDialog.Show vbModal
            point_limes.m_dAccuracy = AccuracyDialog.text_1.Text
            Unload AccuracyDialog

            item.ListSubItems.Add 3, , point_limes.m_dAccuracy
        End If
    End If

    listView.SelectedItem = Nothing

End Sub

Private Sub listView_ItemCheck(ByVal item As MSComctlLib.ListItem)

    Dim point_limes As limes_geometry.Point

    Dim iKey As Integer
    iKey = Strings.Left(item.key, Strings.Len(item.key) - 2)

    If item.Checked = False Then
        item.Text = "False"
        Set point_limes = m_listPointCollection.item(iKey)
        point_limes.controlPoint = False
    Else
        item.Text = "True"
        Set point_limes = m_listPointCollection.item(iKey)
        point_limes.controlPoint = True
    End If

End Sub
```

```
Private Sub addPointCollection(pointCollection As limes_geometry.GeometryHashCollection)

     'Set values to list
    Dim point_limes As limes_geometry.Point
    For Each point_limes In pointCollection
        Dim item As ListItem
        Dim key As String
        key = m_iCount & "id"
        Set item = listView.ListItems.Add(, key)

        If point_limes.m_pointInfo Is Nothing Then
            point_limes.createPointInfo1
            point_limes.controlPoint = True
        End If

        If point_limes.controlPoint = True Then
            item.Text = "True"
            item.Checked = True
        Else
            item.Text = "False"
            item.Checked = False
        End If

        item.ListSubItems.Add 1, , point_limes.Geometry_SourceOID
        item.ListSubItems.Add 2, , point_limes.Geometry_SourceLayer

        'Set default point accuracy
        If point_limes.m_dAccuracy = 0 Then
            point_limes.m_dAccuracy = 10
        End If
        item.ListSubItems.Add 3, , point_limes.m_dAccuracy

        point_limes.m_iIdPoint = m_iCount
        m_listPointCollection.Add point_limes

        m_iCount = m_iCount + 1
    Next

End Sub
```