

Diplomarbeit

RA₂DIO

DAX_05_0607

Diplomandin: Vicki Schmid

Betreuer: Prof. Dr. Christoph Stamm

Expertin: Dr. Gabriele Neyer, SMA und Partner AG Zürich

Abstract

Bei RA₂DIO handelt es sich um ein Programm, in welchem zukünftig Wellenausbreitungen von Mobilfunkantennen berechnet und in parallelperspektivischer Ansicht zweidimensional dargestellt werden können.

Basierend auf einem Prototypen von RA₂DIO, in welchen Bilddaten geladen und in drei Ansichtsmodi angezeigt werden können, mussten im Rahmen der Diplomarbeit diverse Erweiterungen hinzugefügt werden. Diese umfassten die persistente Datenhaltung, Geländevisualisierung, sowie die entsprechende Erweiterung der GUI.

Sämtliche Neuerungen wurden mit der .NET-Technologie realisiert.

Auftraggeber der Arbeit war die Firma xeraina GmbH in Zürich, welche bereits ein Wellenausbreitungsprogramm in 3D, RA₃DIO, entwickelt hat und vertreibt.

Management Summary Diplomarbeit RA₂DIO

Aufgabenstellung

RA₂DIO soll in Zukunft –wie schon das Programm RA₃DIO in 3D- Wellenausbreitungen von Mobilfunkantennen berechnen und darstellen können, allerdings zweidimensional und perspektivisch. Im Rahmen meiner Projektarbeit im Jahr 05/06 wurde ein Prototyp für die Applikation erstellt, auf welchem diese Diplomarbeit aufbaut. Der Prototyp wurde vom betreuenden Dozenten überarbeitet und musste nun um diverse Funktionalitäten erweitert werden. Dabei sollte ausschliesslich mit der .NET-Technologie gearbeitet werden.

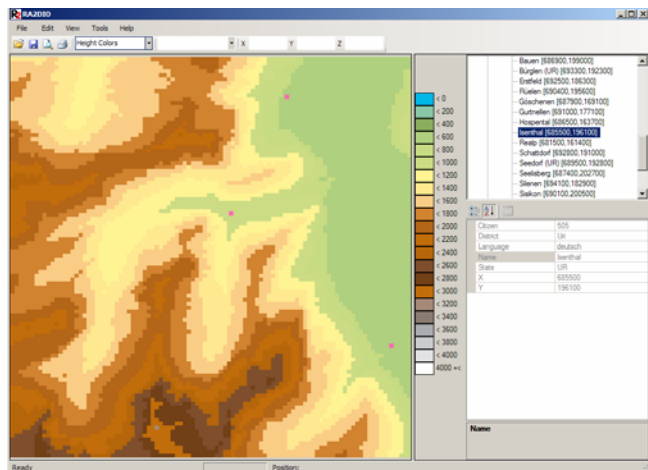
Durchführung

Nach einer Einarbeitungszeit in den bestehenden Code erstellte ich ein Serialisierungskonzept und versuchte dieses umzusetzen. Danach realisierte ich die Datenbankbindung an eine bestehende Access-Datenbank, sowie die Darstellung der Gemeindedaten im TerrainViewer, als auch im TreeViewer und dem PropertyGrid. Ich erstellte ein Konzept für die Darstellung der Höhendaten und setzte die Visualisierung mittels hypsometrischen Farbtönen um. Die GUI wurde durchgehend angepasst.

Grenzen und Ausblick

Während der Diplomarbeit wurden viele Probleme aufgeworfen, mit welchem man in diesem Umfang zu Beginn nicht gerechnet hatte. Die meisten kamen daher, dass bereits im bestehenden Code unmanaged und managed Code gemischt wurden. Viele Konzepte aus .NET, wie die Serialisierung, konnten nicht ohne weiteres für unmanaged Code übernommen werden. Die Anbindung an eine Access-Datenbank mit .NET stellte ebenfalls einige Probleme dar. Aufgrund dieser Schwierigkeiten kam das Projekt zeitlich stark in Verzug und die Aufgabenstellung konnte nicht komplett umgesetzt werden.

Die Applikation wurde dennoch um einige Funktionalitäten erweitert, auf welchen zukünftig aufgebaut werden kann. Zusätzlich entstanden Konzepte, welche zukünftig einbezogen werden können. Bevor man jedoch mit der Entwicklung fortfährt, empfiehlt es sich, eine sorgfältige Evaluation durchzuführen, mit welcher Technologie man weiterarbeiten will. Aufgrund der aufgetretenen Probleme ist es notwendig, das Konzept des mixed Codes zu überarbeiten.



Resultate

Die im Rahmen der Diplomarbeit implementierten Erweiterungen umfassen die Datenbankbindung, die Darstellung der Gemeinden im TerrainViewer und im TreeViewer bzw. dem PropertyGrid, ein Konzept für die Serialisierung, ein Konzept für die Darstellung der Höhendaten als Relief, sowie die Darstellung der Höhendaten mit hypsometrischen Farbtönen im TerrainViewer. Für sämtliche Neuerungen wurde die GUI entsprechend erweitert. Der Benutzer kann mittels GUI Gemeinden anwählen und deren Eigenschaften darstellen, sowie zwischen Höhen- und Texturdaten hin- und herwechseln. Die hypsometrischen Farbtöne können in der GUI editiert und angewendet werden.

Inhaltsverzeichnis

Abstract	2
1 Abkürzungen und Begriffe.....	6
2 Aufgabenstellung	7
2.1 Fachliche Schwerpunkte	7
2.2 Kurzbeschreibung.....	7
3 Grundlagen	8
3.1 RA ₃ DIO	8
3.2 Geländedaten	8
3.2.1 Quelldaten	8
3.2.2 Gemeindedaten (Punktdatei).....	8
3.2.3 Koordinatensysteme	9
3.3 PGF	9
3.4 PGF für Externspeicher (PGFExt)	9
3.5 RA ₂ DIO Prototyp.....	10
3.6 Koordinatensysteme.....	12
4 Systemarchitektur	14
4.1 Aufgabenstellung.....	14
4.2 Grafische Darstellung Systemarchitektur	14
4.3 Benutzeroberfläche	14
4.4 Aufbau GUI.....	15
5 Persistente Datenhaltung.....	16
5.1 Auszug aus der Aufgabenstellung.....	16
5.2 Umsetzung Laden und Speichern der Gemeinde-Daten aus der Datenbank	16
5.2.1 Anbinden der Datenbank mittels DataSource Configuration Wizard, DataSet ...	16
5.2.2 Abfragen der Gemeinden aus der Datenbank	17
5.2.3 Speichern der Gemeindedaten.....	18
5.2.4 Darstellung der Gemeinden.....	20
5.3 Konzept und Umsetzung Serialisierung der Daten.....	27
5.3.1 Serialisierung in .NET	27
5.3.2 Konzept Serialisierung in RA ₂ DIO	27
5.3.3 Umsetzung Serialisierung in RA ₂ DIO	29
6 Geländevisualisierung.....	32
6.1 Auszug aus der Aufgabenstellung.....	32
6.2 Füllen der Höhendaten in die Vertices	32
6.3 Konzept Geländevisualisierung	33
6.3.1 Hypsometrische Farbtöne.....	33
6.3.2 Reliefschattierung.....	33
6.4 Umsetzung Geländevisualisierung mittels hypsometrischen Farbtönen	37
6.4.1 Evaluation: Visualisierung mittels GDI+ oder Bitmap	37
6.4.2 Zeichnen der Höhendaten in die Terrain-Bitmap.....	38
6.4.3 Darstellung der Höhendaten in der Overview-Bitmap	38
6.4.4 Darstellung der Höhenkoordinaten in der Statusbar.....	38
6.4.5 Tiles finden im Terrain	40
6.4.6 Vertices finden im Tile	41
6.5 Ausblick Visualisierung der Höhendaten	42
7 GUI Erweiterungen.....	43
7.1 Auszug aus der Aufgabenstellung.....	43
7.2 Interaktion im TerrainViewer.....	44
7.3 Umsetzung Visualisierung Gemeindedaten.....	44
7.3.1 Notwendige Änderungen an der bestehenden GUI.....	44
7.4 Umsetzung GUI-Erweiterung für Höhendatendarstellung mittels hypsometrischen Farbtönen	44

7.4.1	Colorange-Viewer.....	44
8	Kombination der Datenquellen.....	46
8.1	Kombination der Datenquellen in der Overview-Bitmap	46
8.1.1	Übersicht über die Datenquellen für die Overview-Bitmap	46
8.1.2	Konzept Darstellung in der OverviewBitmap	46
8.1.3	Momentane Darstellung im OverviewModus	47
8.1.4	Ausblick Darstellung im Overview Modus.....	47
8.2	Visualisierte Daten.....	47
9	Automatisches Testen.....	49
9.1	Auszug aus der Aufgabenstellung.....	49
9.2	Testen während der Entwicklung	49
9.3	Probleme beim Verwenden von NUnit.....	49
9.4	Ausblick Testen mit NUnit	50
10	Online- Hilfe.....	51
10.1	Auszug aus der Aufgabenstellung.....	51
10.2	Umsetzung	51
11	Ausblick und weitere Entwicklung	52
11.1	Technologie	52
11.2	Weitere anschliessende Aufgaben	52
12	Persönliche Bemerkungen	54
13	Anhang.....	55
13.1	Komplette Aufgabenstellung (Original).....	55
13.1.1	Aufgabenstellung	56
13.1.2	Einarbeitung.....	56
13.1.3	RA ₂ DIO Weiterentwicklung	56
13.1.4	Geländevisualisierung	57
13.1.5	GUI Erweiterungen	57
13.1.6	Automatisches Testen	58
13.1.7	Online-Hilfe	58
13.1.8	Dokumentation.....	59
13.1.9	Rechtliche Hinweise	59
13.2	Klassendiagramm aktueller Stand.....	61
13.3	Quellenangaben, Literaturverzeichnis	68

1 Abkürzungen und Begriffe

PGF	Progressive Graphics Format
PGFExt	PGF für Externspeicher
DEM	Digital Elevation Model
GIS	Geografisches Informations-System, Geoinformationssystem
STL	Standard Template Library
XML	Extensible Markup Language
SOAP	Simple Object Access Protocol
GUI	Graphical User Interface
ADO	ActiveX Data Objects
GDI	Graphic Device Interface (Microsoft)

2 Aufgabenstellung

2.1 Fachliche Schwerpunkte

- .NET-Entwicklung (VS 2005) eines Gelände-Viewers in C++
- Graphical User Interface (Windows Forms)
- Effiziente, externe Datenspeicherung von Geländedaten (PGF für Externspeicher)
- Visualisierung von geografischen Daten

2.2 Kurzbeschreibung

In geographischen Informationssystemen (GIS) werden sehr grosse Mengen an Geländedaten verarbeitet. Zu den Geländedaten zählen Topographiedaten (Höhendaten), Texturdaten (Luftbilder, Pixelkarten usw.), Landnutzungsdaten und weitere thematische Daten, welche mit geografischen Koordinaten verbunden sind. Diese Geländedaten sollten auf Externspeichern (z.B. Festplatte) verwaltet und nur stückweise bei Bedarf in den Hauptspeicher geladen werden. Dadurch können mit normalgrossem Hauptspeicher riesige Geländeabschnitte prozessiert werden.

RA₂DIO soll in ferner Zukunft wie sein grosser Bruder RA₃DIO die Funkwellenausbreitung in riesigen Geländen simulieren. Nun aber ohne echte 3D-Darstellung, sondern mit normaler, perspektivischer 2D-Ansicht. Dafür aber mit einer ausgeklügelten und effizienten Verwaltung der Geländedaten. Diese Verwaltung soll auf der Technik "PGF für Externspeicher" basieren. Dabei werden Bilddaten in Kacheln aufgeteilt und einzeln mit PGF komprimiert abgelegt. Höhen- und Landnutzungsdaten können als weitere "Bildkanäle" der Texturdaten interpretiert und gemeinsam mit den Bilddaten auf dem Externspeicher abgelegt werden.

Informationen zu PGF und RA₃DIO finden Sie unter www.xeraina.ch/pgf beziehungsweise www.xeraina.ch/ra3dio.

Diese Diplomarbeit ist eine Fortsetzung der gleichnamigen Projektarbeit.[8]

Die komplette Aufgabenstellung ist dem Anhang zu entnehmen.

3 Grundlagen

Die folgenden Kapitel zu den Grundlagen sind Auszüge bzw. Zusammenfassungen aus der Dokumentation der RA₂DIO-Projektarbeit. Für weitere Infos zur Projektarbeit, ist die zugehörige Dokumentation zu konsultieren. [8]

3.1 RA₃DIO

RA₃DIO simuliert Funkwellenausbreitungen im dreidimensionalen Raum. Dabei kann der Benutzer selber Antennen durch Reinklicken erstellen oder die Antennen-Daten via Datenbank verwalten.

Die Geländedaten liegen im Bildformat PGF und als Punktdaten in einer Datenbank vor. Der Benutzer kann aus verschiedenen Perspektiven mittels Maus oder Eingabefelder im Gelände navigieren. Weiter kann er auswählen, welche Geländedaten (Landnutzung, Höhendaten, Texturdaten, etc) er angezeigt haben möchte. [1]

3.2 Geländedaten

3.2.1 Quelldaten

Es liegen folgende Geländedaten vor:

- Texturdaten
- Höhendaten
- Landnutzungsdaten
- Wellenausbreitungsdaten
- Basisstationen

Im Rahmen der Diplomarbeit wurde mit Textur- und Höhendaten, sowie Gemeindedaten (Punktdaten) gearbeitet. Im Folgenden werden nur diese Quelldaten beschrieben. Für Infos über die anderen Datenquellen ist die Dokumentation der Projektarbeit zu konsultieren. [8]

3.2.1.1 Texturdaten

Texturdaten sind Geländedaten wie man sie von Landeskarten kennt. Normalerweise sind dies Pixelkarten, Luft- oder Satellitenaufnahmen. Sie liegen im PGFExt-Format vor.

3.2.1.2 Höhendaten

Höhendaten liegen als Graustufenbild im PGFExt-Format vor und sind auf 1 Dezimeter genau gespeichert. Ein bestimmter Höhenwert hat dabei einen bestimmten Grauwert. Höhendaten werden meist als schattiertes Relief und/oder mit hypsometrischen Farbtönen dargestellt.

3.2.2 Gemeindedaten (Punktdaten)

In einer Access-Datenbank-Tabelle sind alle Gemeinden gespeichert. Für die Diplomarbeit sind dies die Gemeinden der gesamten Schweiz. Pro Gemeinde sind folgende Angaben gespeichert:

- Index (ein Index anhand dessen vordefinierte Abfragen gestartet werden können, wurde für die Diplomarbeit nicht verwendet)
- Name
- District (Bezirk)
- State (Kanton)
- Language
- Citizen (Einwohnerzahl)

- X-Koordinate auf 100 Meter genau
- Y-Koordinate auf 100 Meter genau

3.2.3 Koordinatensysteme

Die Geländedaten können in zwei verschiedenen Koordinatensystemen vorliegen:

- Geografisches Koordinatensystem (λ = Längengrad, ϕ = Breitengrad, z-Achse in metrischer Auflösung)
- Metrisches, kartesisches (Landes-)Koordinatensystem.

Beim metrischen Koordinatensystem befindet sich der Ursprung bei Bordeaux in Frankreich. Dies bedeutet, dass die Ausbreitung in x-Richtung von 480 km - 840 km die Ausbreitung in y-Richtung von 72 km - 300 km geht. Referenzpunkt in der Schweiz ist die Sternwarte in Bern bei 600/200 km.

Man geht für RA₂DIO davon aus, dass alle Daten im selben Koordinatensystem vorliegen.

Weitere Informationen zu den Koordinatensystemen als auch zu den Geländedatentypen können der Web-Site des Bundesamtes für Landestopographie entnommen werden. [4]

3.3 PGF

Das Bildformat PGF (Progressive Graphics Format) wurde ursprünglich für RA₃DIO entwickelt und basiert auf einer schnellen diskreten Wavelettransformation. Es kann als Ersatz für JPEG gesehen werden, bietet aber meist eine schnellere und höhere Kompression als JPEG. Aufgrund der schnellen Kompression und Dekompression sowie seiner Progressivität, eignet sich PGF besonders gut für grosse Datenmengen, wie sie in geografischen Anwendungen wie RA₃DIO verwendet werden. [2]

3.4 PGF für Externspeicher (PGFExt)

PGFExt basiert auf PGF und erlaubt den Zugriff auf einzelne Bildausschnitte. Dazu wird das Bild durch ein fixes Gitter in gleich grosse, rechteckige Teile (Tiles) unterteilt. Diese können unabhängig voneinander mit individueller Tiefe und Qualität komprimiert bzw. dekomprimiert werden. PGFExt-Daten werden in Pixmaps dekomprimiert. [3]

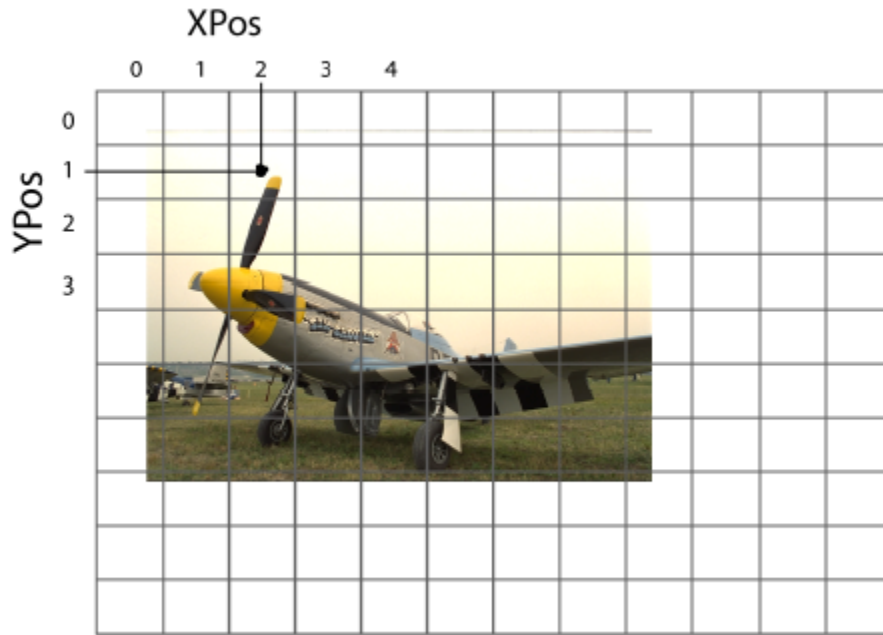


Abb. 1: Aufteilung der PGFExt Datei in Tiles [3]

PGFExt bietet bereits die Schnittstelle um Tiles mit gewünschter Dekomprimierung zu laden.

Weiter stellt PGFExt zwei Caches zur Verfügung:

- Ein Tile-Cache für unkomprimierte Tiles; die Tiles werden verwaltet, der Speicher freigegeben, etc.
- Ein Bildcache für die dekomprimierten Pixmaps. Dieser Bildcache wurde für den RA₂DIO-Prototypen nicht verwendet, sondern durch eine eigene Datenstruktur ersetzt. (siehe 3.5)

3.5 RA₂DIO Prototyp

Für die Diplomarbeit wurde an dem Prototypen von RA₂DIO weitergearbeitet, welcher während meiner Projektarbeit im Jahr 2005/2006 als Partnerarbeit entwickelt, und von Herrn Stamm überarbeitet wurde.

Im Rahmen der Projektarbeit wurde eine Datenstruktur (Terrain) definiert, in welche die PGFExt-Tiles vom Externspeicher (in dem Fall der Festplatte) geladen und zwischengespeichert werden, ehe sie in der GUI dargestellt werden.

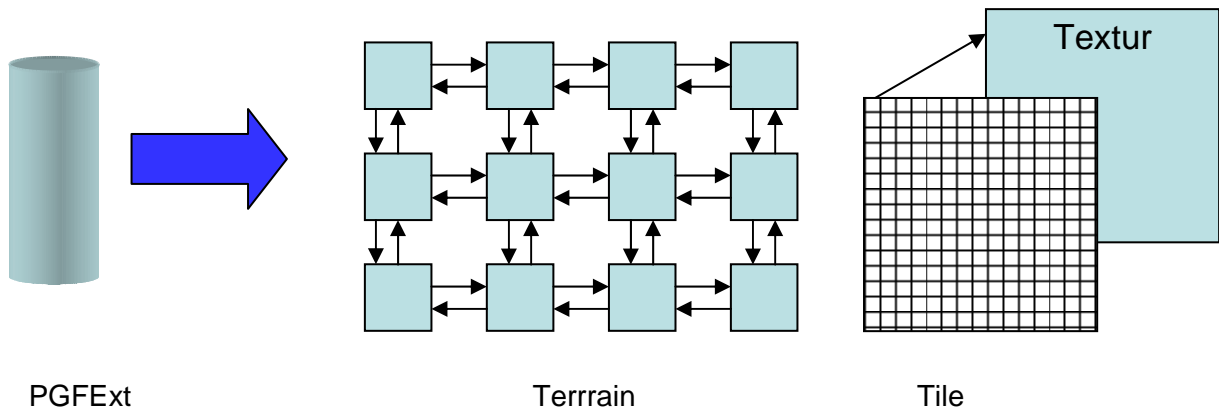


Abb. 2: Laden und Organisieren der geladenen Daten im Terrain

Das Terrain ist eine doppelt verkettete Liste, welche aus Tiles aufgebaut ist. Ein Terrain-Tile entspricht dabei einem PGFExt-Tile. Pro Terrain-Tile ist ein Vertex-Array gespeichert. In einem Vertex werden zum Beispiel die Höhendaten gespeichert, welche für die Visualisierung der Höhendaten gebraucht werden.

Ein Tile kennt seine vier Eckpunkte in geografischen Koordinaten, sowie seine Nachbartiles.

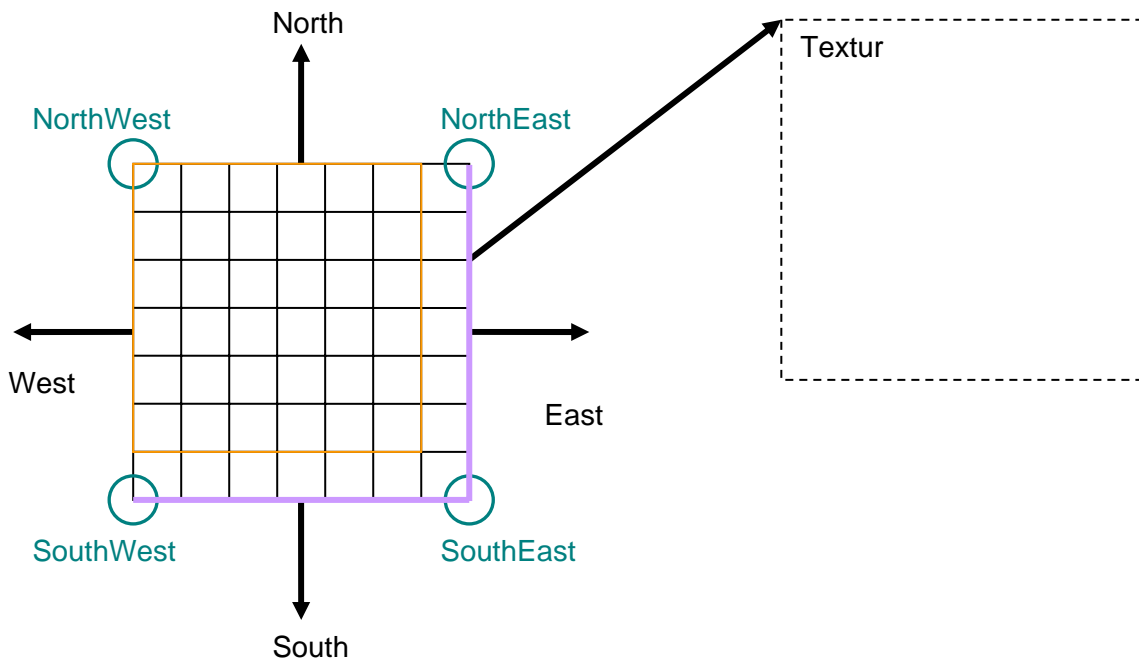


Abb. 3: Aufbau Terrain-Tile

Für den VertexArray (hier dargestellt als Gitter), ist zu beachten, dass am Rand rechts und unten sich bereits die Vertices des Nachbartiles befinden (lila). Die Vertices des Tiles werden nicht bis an diese Grenzen verteilt (orange). Dies ist beim Einfüllen bzw. Auslesen der Vertices zu beachten.

Mittels der Klasse VertexIterator kann über die Vertices iteriert werden. Ansonsten können die Vertices auch via Index im Array angesprochen werden. Die Texturdaten sind direkt an die Terrain-Tiles angehängt mittels Zeiger. Die Terrain-Tiles werden zu einer Bitmap zusammengefügt und im TerrainViewer der GUI dargestellt.

Es gibt allgemein gültige Daten, welche für alle Klassen zugänglich sein müssen, sowie Daten, welche pro PGFExt-Datei abgespeichert sind. Dazu sind die Klassen PGEDataSet und PGEPackage definiert. In PGEDataSet wurden (ursprünglich als Klassenattribute, dies wurde während der Diplomarbeit geändert) sämtliche Infos gespeichert, welche für alle Dateien gültig sind. Pro PGFExt-Datei existiert ein PGEPackage, in welchem die für diese Datei gültigen Infos gespeichert sind.

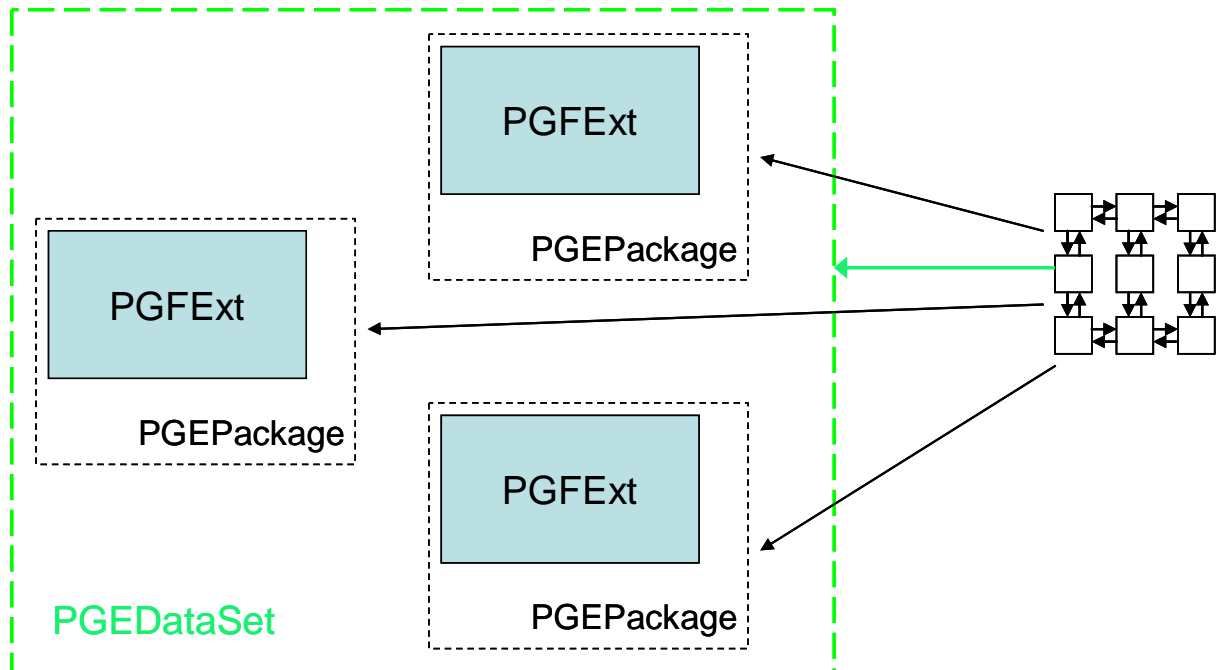


Abb. 4: Organisation mittels PGEPackage und PGEDataSet

Für die Benutzeroberfläche ist die Klasse MainForm zuständig. In ihr werden sämtliche Controls, sowie Methoden und Attribute, welche mit der Anzeige der GUI zu tun haben, verwaltet.

Für die Koordinatensysteme existieren die Basisklassen CCoord, sowie die von ihr abgeleiteten Klassen CSwissCoord und CGeoDecCoord für die metrischen bzw. geografischen Koordinatensysteme. Für Distanzen in Koordinaten gibt es die Klasse CDist mit welcher geografische als auch metrische Distanzen ausgedrückt werden können. Alle genannten Klassen bieten diverse Methoden an, zum Beispiel um Distanzen von Koordinaten zu subtrahieren.

Der übernommene Prototyp kann Texturdaten darstellen und stellt die notwendige Funktionalität bereit, um Höhendaten in die Vertices der Tiles zu laden. Die Darstellung erfolgt in einer einfachen GUI.

3.6 Koordinatensysteme

Es wird in drei verschiedenen Koordinatensystemen operiert:

- Control-Koordinaten (Bildschirmkoordinaten)
- Pixelkoordinaten
- Geografische bzw. metrische Koordinaten

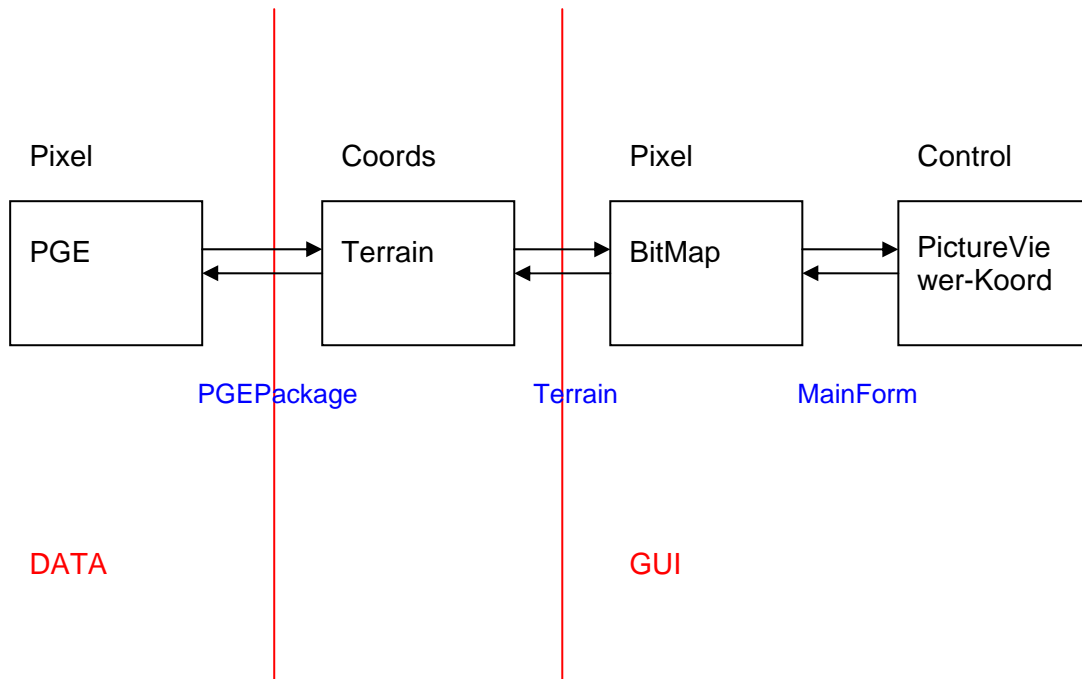


Abb. 5: Koordinatensysteme

Um zwischen den einzelnen Koordinatensystemen umzurechnen, stehen folgende Funktionen zur Verfügung:

```

Point CTerrain::CoordToPixel(bool overview, CCoord* coord) const;
CCoord* CTerrain::PixelToCoord(bool overview, int x, int y) const;
CCoord* CTerrain::PixelToCoord(bool overview, Point point) const ;

Drawing::Point MainForm::ControlToPixels(Drawing::Point pos, bool
overview);
Drawing::PointF MainForm::PixelsToControl(Drawing::Point pos, bool
overview);
  
```

Mittels dem Parameter „Overview“ wird jeweils angegeben, ob man sich im Overview-Modus befindet, oder im Terrain- bzw. Detail-Modus.

4 Systemarchitektur

4.1 Aufgabenstellung

Die aktuelle Version von RA₂DIO basiert auf einer klaren Trennung zwischen grafischer Benutzungsoberfläche (GUI) und interner Datenhaltung/-verarbeitung einerseits und einer Trennung zwischen interner und externer Datenhaltung. Diese dreischichtige Systemarchitektur, welche auf dem Konzept des Model View Control basiert, soll weiter gepflegt und wenn möglich noch deutlicher sichtbar gemacht werden. Eine klare dreistufige Architektur, bei der direkte Zugriffe vom GUI (View-Layer) auf die externen Daten (Model-Layer) verboten sind, ist absolut zentral.

4.2 Grafische Darstellung Systemarchitektur

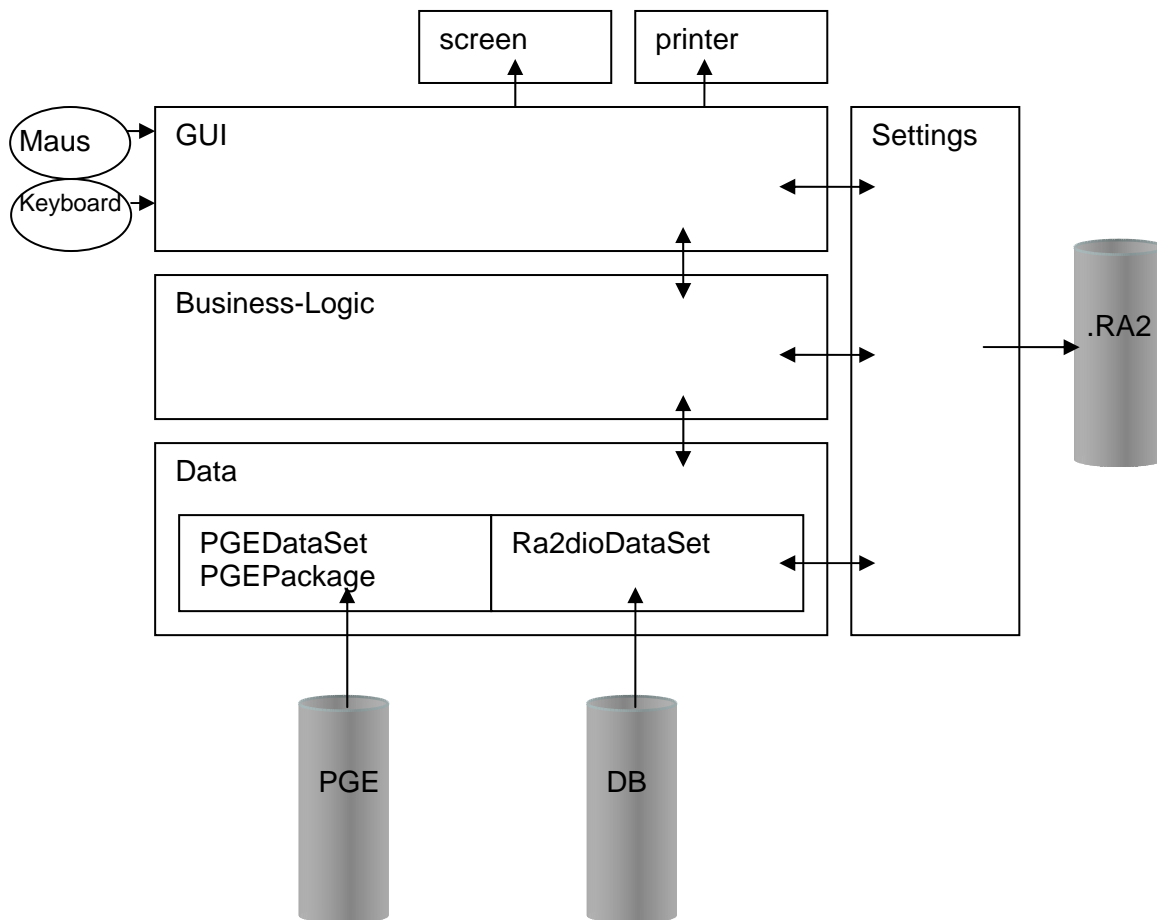


Abb. 6: Dreischichtige Systemarchitektur

Die Daten werden aus den PGFExt-Dateien bzw. der Datenbank in den Data-Layer geladen, und im Business-Logic-Layer für die Anzeige in der GUI aufbereitet. Von allen Layers werden bestimmte Einstellungen (Settings) serialisiert und in einer .RA2-Datei gespeichert.

4.3 Benutzeroberfläche

Da im Folgenden oft die Namen der Elemente aus der GUI verwendet werden, sollen sie hier zu Beginn illustriert werden.

4.4 Aufbau GUI

Die GUI ist folgendermassen aufgebaut:

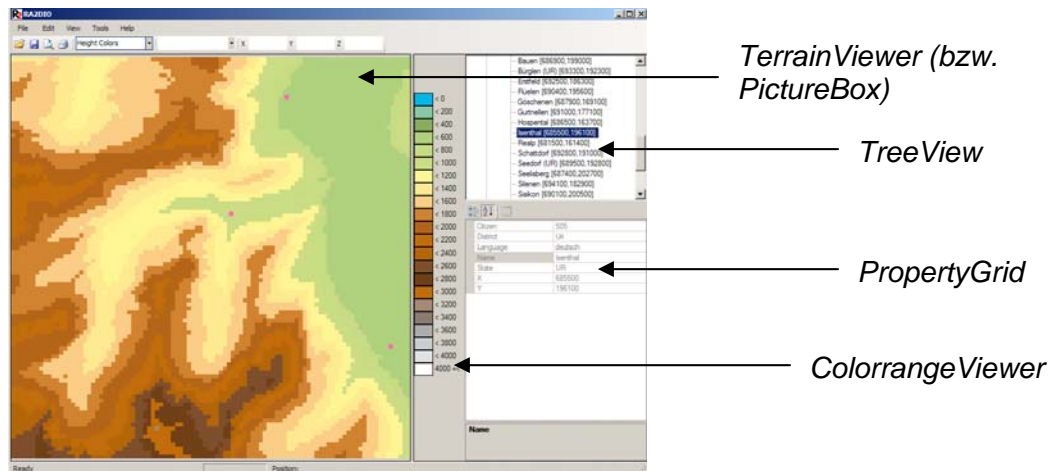


Abb. 7: Aufbau GUI

Im TerrainViewer können drei verschiedene Ansichtsmodi gewählt werden: Overview (eine Übersicht, welche nicht lange laden soll, um sich einen schnellen Überblick zu verschaffen), Terrain (herangezoomt) und Detail (noch näher herangezoomt).

Der Overview-Modus zeigt die ganze PGFExt-Datei an, während im Terrain-Modus das Terrain angezeigt wird, und im Detail-Modus ein Ausschnitt aus dem Terrain.

Im TreeViewer sollen Daten aus der Datenbank angezeigt werden, im PropertyGrid Details zu einzelnen Objekten.

Die PGFExt-Dateien werden stets im TerrainViewer dargestellt (Attribut PictureBox der Klasse MainForm). Dabei werden die Tiles im Terrain zu einer Bitmap, der Terrain-Bitmap, zusammengefügt und dann in die PictureBox geladen. Für den Overview-Modus wird das Bild in eine separate Bitmap, die Overview-Bitmap, geladen.

5 Persistente Datenhaltung

5.1 Auszug aus der Aufgabenstellung

In RA₂DIO müssen unterschiedliche Daten persistent gespeichert werden. Zu diesen Daten gehören unter anderem: Benutzereinstellungen, aktueller Geländezustand, aktuelle Wellenausbreitungsdaten, Basisstationen. Während die Basisstationen in einer Datenbank gehalten werden, sollen der aktuelle Geländezustand und die aktuellen Wellenausbreitungen in einem speziellen Dokument (mit der Endung .RA2) abgelegt werden. Somit kann die Arbeit mit RA₂DIO jederzeit unterbrochen, der aktuelle Stand in einem Dokument gespeichert und die Arbeit zu einem späteren Zeitpunkt durch Öffnen des Dokumentes wieder aufgenommen werden.

Im bestehenden Programm RA₃DIO werden Punktobjekte wie Basisstationen, Messpunkte und Ortschaften in einer relationalen Datenbank gehalten. An diesem Datenbankkonzept soll in RA₂DIO festgehalten werden, weil es sich bewährt hat und die Kunden über umfangreiche Datenbanken verfügen. Dabei soll auf eine Anpassung der Datenbank, wenn möglich, verzichtet werden.

Ziele

Erstellen Sie einfaches und gut umsetzbares Persistenz-Konzept für alle Daten, die in RA₂DIO persistent gehalten werden müssen. Setzen Sie dieses Konzept um.

In der aktuellen Version von RA₂DIO ist Tiny-XML zur Verarbeitung von XML-Dateien eingesetzt worden. Ersetzen Sie Tiny-XML durch die vorhandenen Konzepte und Technologien in .NET.

Studieren Sie die Anbindung von Microsoft Access-Datenbanken in .NET. Der bestehende Quellcode kann hier nur sporadisch übernommen werden, da die vorhandene Datenbankanbindung auf MFC basiert.

Implementieren Sie den Zugriff auf die Tabellen mit den Ortschaftsdaten und visualisieren Sie alle sichtbaren Ortschaften im Gelände-Viewer mittels eines geeigneten Symbols oder Textes.

5.2 Umsetzung Laden und Speichern der Gemeinde-Daten aus der Datenbank

Die .NET-Technologie bietet spezielle Hilfsmittel an, die das Anbinden einer Datenbank an ein Projekt sowie die Manipulation der Daten in der Datenbank ermöglichen.

5.2.1 Anbinden der Datenbank mittels DataSource Configuration Wizard, DataSet

Mittels Data Source Configuration Wizard, können Datenbanken einfach in ein Projekt eingefügt werden. Man wählt dazu im Menu „Data– Add Data Source“. Im Wizard wählt man „Database“ aus und gibt im Feld „New Connection“ die gewünschte Datenbank an.

Der Wizard erstellt dann automatisch vier neue Dateien im Projekt:

- [Datenbankname] DataSet.xsd
- [Datenbankname] DataSet.h
- [Datenbankname] DataSet.xsc
- [Datenbankname] DataSet.xss

Die .xsd-Datei ist eine XML Schema Definition-Datei, welche im CodeView-Modus vom Programmierer normalerweise nicht verändert werden muss. Im DesignView-Modus hingegen können Datenbank-Manipulationen wie Tabellenerstellung, Queries einfügen etc. vorgenommen werden.

Die .h-Datei ist eine C++-Headerdatei, welche Funktionalitäten anbietet, die diverse Zugriffe auf die Tabellen in der Datenbank ermöglichen. Auch diese Datei muss im Normalfall nicht

verändert werden. Sie bietet u.a. für jede Tabelle einen TableAdapter an, mit welchem auf die jeweilige Tabelle zugegriffen werden kann, z.B. um Queries auszuführen.

Die .xss-Datei speichert das Layout des DataSet-Designers und soll nicht modifiziert werden. Die .xsc-Datei soll ebenfalls nicht geändert werden. Sie wurde im Rahmen dieser Diplomarbeit nicht benötigt.

5.2.2 Abfragen der Gemeinden aus der Datenbank

Mittels Ra2dioDataSet und den Ra2dioDataSetTableAdapters kann auf die Tabellen in der Datenbank bzw. auf Tabellen, welche im DataSet-Designer erstellt wurden, zugegriffen werden.

Ursprünglich war geplant, vordefinierte Access-Abfragen der Datenbank zu verwenden. Dies stellte sich unter .NET jedoch als unmöglich heraus, da .NET vorwiegend die Datenbankbindung mit SQLServer unterstützt. Deshalb wurde versucht, die Abfrage „GetCommunities“ im DataSet-Designer nachzubilden. „GetCommunities“ erwartet vier Parameter, wobei folgende Kombinationen und Resultate möglich sind:

Index = Null, Name = Null	Zeigt alle Gemeinden der Datenbank an.
Index != Null, Name = Null	Zeigt die Gemeinde mit dem angegebenen Index an.
Index = Null, Name != Null	Zeigt alle Gemeinden mit angegebenem Namen an.
Index != Null, Name != Null	Zeigt die Gemeinde mit dem angegebenen Index und dem angegebenen Namen an.

Hierzu sind benannte Parameter nötig. Wie sich jedoch herausstellte, unterstützt das DataSet in .NET keine benannten Parameter [6][7], bzw. sie werden wiederum nur für SQL-Server unterstützt. Deshalb wurden die Abfrage „GetCommunities“ ersetzt, indem im DataSet-Designer ein neuer TableAdapter eingefügt wurde. Dazu wurde im Designer aus der Toolbox ein TableAdapter angewählt. Mittels TableAdapter Configuration Wizard werden die betroffenen Tabellen angewählt und eine Query (in diesem Fall ein einfaches SELECT aller Gemeinden) eingefügt. Die Optionen „Fill a DataTable“ sowie „Return a DataTable“ sollen dabei angewählt bleiben. Sie bewirken, dass eine neue Tabelle mit den gewünschten Daten erstellt wird. Im Designer erscheint nun die neue Tabelle (in diesem Fall die Tabelle „CommunitySelection“).

Um die Abfrage „GetCommunities“ zu ersetzen wurden folgende Queries erstellt:

- Fill_AllCommunities()
Lädt alle Gemeinden.
- FillBy_IndexAndName (Index,Name)
Es müssen Index als auch Gemeindennamen übergeben werden.
- FillBy_IndexOrName (Index,Name)
Es muss nur der Index oder der Gemeindennamen übergeben werden.

Für die parametrisierten Queries muss im Designer unter Properties-Parameters bei den Parametern der Datentyp angegeben werden, sowie (falls wie in diesem Fall erwünscht) unter „AllowDBNull“, ob ein Nullwert übergeben werden darf (in diesem Fall „true“).

Das Konzept des Ladens der Daten aus der Datenbank wurde etwas geändert. Zwar stehen die Abfragen mittels Index und Name wieder zur Verfügung, jedoch wurde für die

Diplomarbeit eine neue Query erstellt, mit welcher Gemeinden mittels Koordinaten aus der Datenbank geholt werden können:

- FillBy_Coords (X,Y,X1,Y1)

Ebenfalls dieser Tabelle hinzugefügt wurde eine Query, welche alle Gemeinden zuerst nach State, dann nach District und schliesslich nach Gemeindennamen geordnet liefert:

- FillBy_AllCommOrdered ()

Diese Abfrage wird für das Erstellen des Gemeinde-Trees verwendet.

In der Klasse MainForm sowie in der Klasse CTile werden die Gemeinden aus der Datenbank geladen:

In der Klasse CTile werden in CTile::LoadCommunities(..) alle Gemeinden, welche sich auf einem Tile befinden mittels der Query FillBy_Coords(X, Y, X1, Y1) geladen und in eine ArrayListe an das entsprechende Tile angehängt.

In der Klasse MainForm werden die Gemeinden in der Funktion MainForm::FillCityDict() mittels der Query FillBy_AllCommOrdered() geladen und in einen Dictionary gefüllt.

```
Void MainForm::FillCityDict(){
    //durch alle Communities iterieren und in m_cityDict füllen.
    Ra2dioDataSet dataSet;
    m_cityDict= gcnew Dictionary<Point, CCity^>();
    Ra2dioDataSetTableAdapters::CommunitySelectionTableAdapter^
    tableAdapter= gcnew
    Ra2dioDataSetTableAdapters::CommunitySelectionTableAdapter();
    tableAdapter->FillBy_AllCommOrdered(dataSet.CommunitySelection);
    for (int j=0; j<dataSet.CommunitySelection->Rows->Count; j++){
        CCity^ city= gcnew CCity(dataSet.CommunitySelection[j]->Name,
        dataSet.CommunitySelection[j]->State,
        dataSet.CommunitySelection[j]->District,
        dataSet.CommunitySelection[j]->Language,
        dataSet.CommunitySelection[j]->Citizen,
        dataSet.CommunitySelection[j]->X,
        dataSet.CommunitySelection[j]->Y);
        Point tmpPoint(city->X, city->Y);
        m_cityDict->Add(tmpPoint, city);
    }
}
```

Zuerst wird also ein TableAdapter für die betroffene Tabelle erzeugt. Mittels diesem werden anhand der Query FillBy_AllCommOrdered(..) die sortierten Gemeinden geholt und in die Tabelle „CommunitySelection“ geladen. Indem nun in der for-Schleife durch die Zeilen der CommunitySelection-Tabelle iteriert wird, kann Datensatz um Datensatz ausgelesen und verarbeitet werden.

5.2.3 Speichern der Gemeindedaten

Die Gemeindedaten werden, nachdem Sie aus der Datenbank geladen wurden, in der Klasse CCity gespeichert. Pro Gemeinde wird ein City-Objekt erzeugt. Die City-Objekte werden wiederum in einem Dictionary bzw. in einer ArrayList gespeichert.

5.2.3.1 City-Klasse

Die City-Klasse speichert alle Eigenschaften einer Gemeinde aus der Datenbank. Dazu gehören:

- District
- State
- Language
- Citizen
- X
- Y
- Name

Diese Attribute wurden als Properties deklariert, wodurch sie einfach im der PropertyGrid der GUI angezeigt werden können.

Für die Visualisierung müssen noch zusätzliche Attribute gespeichert werden:

- CityMode
- Node

Damit diese nicht im PropertyGrid erscheinen, sind sie als normale Attribute und nicht als Properties deklariert.

Mittels CityMode wird festgehalten, ob eine Gemeinde markiert (CM_Marked) oder unmarkiert (CM_Unmarked) ist. Dies ist notwendig, um bei der Visualisierung der Gemeinden im TerrainViewer die markierte Gemeinde anders zu colorieren als die unmarkierten Gemeinden.

In Node wird ein Zeiger auf die Gemeinde im TreeView gespeichert. Dieser wird benötigt, um das entsprechende Node im TreeView zu selektieren, wenn im TerrainViewer auf eine Gemeinde geklickt wird. (vergl. 5.2.4 Darstellung der Gemeinden)

5.2.3.2 Dictionary bzw. ArrayList mit City-Objekten

Der Dictionary `m_cityDict` ist Attribut der Klasse MainForm. Darin werden alle Cities gespeichert. Nun können Methoden, welche auf die Gemeinden zugreifen müssen, direkt auf den Dictionary zugreifen, ohne dass eine Datenbankabfrage gemacht werden muss. Der Vorteil des Dictionaries liegt im Gegensatz zu anderen Datenstrukturen, wie z.B. der ArrayList darin, dass mittels Key als auch Value Einträge im Dictionary gesucht werden können.

Für den Dictionary `m_cityDict` wurden als Key ein Point (die Koordinaten X, Y) und ein Zeiger auf ein CityObjekt als Value definiert:

```
Dictionary<Point, CCity^>^ m_cityDict;
```

In der Tile-Klasse werden die Gemeinden, welche auf einem Tile liegen in einer ArrayList gespeichert. Dies ermöglicht ebenfalls einen schnelleren Zugriff für die Darstellung der Gemeinden im TerrainViewer im Terrain- oder Detail-Mode. Da bei der Gemeindevisualisierung auf einem Tile nur durch die Gemeinden iteriert werden muss, aber keine Gemeinden gesucht werden, wurde hier auf die Speicherung in einem Dictionary verzichtet.

5.2.4 Darstellung der Gemeinden

Die Gemeinden sollen im TerrainViewer und im TreeView angezeigt werden. Dabei wird bidirektional interagiert:

- Wird eine Gemeinde im TreeView angeklickt, werden einerseits die Properties (Name, Kanton, Gemeinde, Einwohnerzahl, X- und Y-Koordinate) im PropertyGrid der GUI angezeigt, andererseits wird die betroffene Gemeinde im TerrainViewer selektiert und in einer anderen Farbe eingefärbt. Falls nötig, wird das Terrain in den entsprechenden Bildausschnitt verschoben.
- Wird eine Gemeinde im TerrainViewer angeklickt, wird die entsprechende Gemeinde im TreeView selektiert und die Properties im PropertyGrid angezeigt.

5.2.4.1 Darstellung im TerrainViewer

Für die Darstellung im TerrainViewer wird zwischen den diversen Viewer-Modi unterschieden. Für jeden Modus wurde eine separate Funktion für die Visualisierung geschrieben:

- `Void MainForm::DrawCitiesInOverview(System::Windows::Forms::PaintEventArgs ^ pe);`
- `Void MainForm::DrawCitiesInTerrain(System::Windows::Forms::PaintEventArgs ^ pe);`
- `Void MainForm::DrawCitiesInDetail(System::Windows::Forms::PaintEventArgs ^ pe);`

Wenn in der Methode `MainForm::pictureBox1_Paint(..)`, welche vom `PaintEvent` der `PictureBox` aufgerufen wird, in die `PictureBox` gezeichnet wird, wird abgefragt, in welchem ViewerModus man sich befindet, und dann die entsprechende Funktion aufgerufen.

Die Funktion `DrawCitiesInOverview(..)` zeichnet die Gemeinden für die ganze Overview-Ansicht. Dabei wird mittels „for each“-Schleife durch den `CityDictionary` iteriert und für jede Gemeinde im `Dictionary` die entsprechende Gemeinde im `TerrainViewer` gezeichnet.

Die Funktionen `DrawCitiesInTerrain(..)` und `DrawCitiesInDetail(..)` verwenden die `Broadcast`-Methode von `Terrain` sowie die folgende Methoden:

```
Void MainForm::DrawCitiesInTile(Drawing::Graphics^ g, CTile* tile);

void CTile::DrawCities(CMessage& m){
    Graphics^ g= dynamic_cast<Drawing::Graphics^> (m.GetObject());
    assert(g);
    theApp->MainAppForm->DrawCitiesInTile(g, this);
}
```

Mittels `CTerrain::Broadcast(..)` kann über alle `Tiles` iteriert und pro `Tile` eine Methode der Klasse `Tile` ausgeführt werden. Dabei sieht der Aufruf (z.B. in `MainForm::DrawCitiesInDetail`) wie folgt aus:

```
System::Drawing::Graphics ^g= pe->Graphics;
CMessage msg(Every, g);
theApp->Terrain->Broadcast(B_All, &CTile::DrawCities, msg);
```

Der Broadcast-Methode wird als Parameter übergeben, über welche Tiles iteriert werden soll (B_All= alle Tiles), welche Funktion von CTile aufgerufen werden soll (CTile:: DrawCities) und ein Message-Parameter, welcher die Parameter für letztere Funktion enthält (Every würde verwendet, um über alle Antennen zu iterieren und ist hier nicht relevant, g ist das Graphics-Objekt, auf welches gezeichnet wird).

Die Broadcast-Methode existierte bereits im Prototypen von RA₂DIO. Die Klasse Message war jedoch nur für unmanaged Code vorhanden. Da es sich beim Graphics-Objekt um einen Managed Pointer handelt, musste die Message-Klasse um den Pointer

```
gcroot<System::Object^> m_managedPtr;
```

sowie um die entsprechenden Konstruktoren und Setter- und Getter-Methoden erweitert werden.

In `CTile::DrawCities(CMessage& m)` wird aus dem Message-Parameter das Graphics-Objekt extrahiert und der Methode

```
MainForm::DrawCitiesInTile(Drawing::Graphics^ g, CTile* tile)
```

übergeben.

Man mag sich nun fragen, warum ein Tile nicht direkt die Gemeinden ins Tile zeichnet, sondern dies über die obige Funktion in MainForm tut.

Die Gemeinden werden mittels GDI+ in die PictureBox gezeichnet. Es erschien mir unsauber, wenn die Tiles als „logische“ Klasse diese Zeichenoperation in der GUI direkt vornehmen würden, weshalb ich sie in die MainForm-Klasse ausgelagert habe und vom Tile aus aufrufe.

Das Zeichnen der Gemeinden geschieht, wie oben erwähnt, mittels GDI+. Dabei wird um die tatsächliche Koordinate (dunkel) im TerrainViewer ein gefülltes Rechteck gezeichnet. Je nachdem, ob die Gemeinde angewählt ist und in welchem Viewer-Modus man sich befindet, variieren die Grösse als auch die Farbe des Rechtecks.

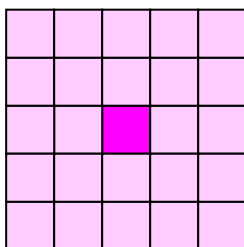


Abb. 8: Zeichnen des Rechtecks um tatsächliches Gemeindepixel

5.2.4.2 Anschreiben der Gemeindennamen

Werden im Terrain- oder Detail-Modus die Gemeinden mit `MainForm::DrawCitiesInTerrain(...)` gezeichnet, so wird in dieser Methode mittels Broadcast und der Funktion `MainForm::DrawCityNamesInTile(Drawing::Graphics^ g, CTile* tile)` die Gemeindennamen angeschrieben. Letztere Funktion iteriert durch alle Gemeinden in der zum Tile zugehörigen ArrayList und schreibt für Gemeinden mit mindestens so vielen Einwohnern wie in der Konstante `MinCitizen` (in „Constants.h“) definiert sind, den Gemeindennamen an.

Im Overview-Modus wird in der Methode `MainForm::DrawCitiesInOverview(..)` nach dem Zeichnen der Gemeinden nochmals durch den City-Dictionary iteriert und nach dem selben Kriterium der Mindest-Einwohnerzahl die Beschriftung an den entsprechenden Gemeinden angebracht.

5.2.4.3 Ausblick Anschreiben Gemeindennamen

In der momentanen Lösung werden die Gemeinden im Dictionary durchlaufen. Ist ein CityObjekt im Dictionary auf „marked“ gesetzt, wird es entsprechend markiert gezeichnet. Danach wird das CityObjekt sofort auf „unmarked“ gesetzt. Dies hat zur Folge, dass, wenn ein neuer Paint-Event in der PictureBox ausgelöst wird, die Gemeinde beim Neuzeichnen nicht mehr hervorgehoben ist.

In der weiteren Entwicklung ist hier ev. ein neues Konzept zu überlegen, welches die Gemeinden allenfalls zu einem anderen Zeitpunkt auf unmarked zurücksetzt.

5.2.4.4 Darstellung im TreeView

Beim Aufstarten der Applikation werden sämtliche Daten in den Dictionary geladen und von dort in den TreeView geschrieben. Im TreeView werden immer alle Gemeinden der Schweiz angezeigt. Dabei ist allerdings zu beachten, dass die vorhandene PGFExt-Datei nicht zwingend die ganze Schweiz abdeckt und somit im TreeView Gemeinden angewählt werden können, welche sich ausserhalb des von den Bilddaten abgedeckten Bereichs befinden. Diese Option wird abgefangen, indem das Terrain nicht verschoben, die City-Properties jedoch angezeigt werden.

Für die Darstellung im TreeView sind die Gemeinden nach Kanton und innerhalb eines Kantons nach Bezirk sortiert. Sämtliche Namen (Kanton, Bezirk, Gemeinde) sind alphabetisch sortiert. Dazu werden die Gemeinden mittels der Query `FillBy_AllCommOrdered(..)` bereits sortiert in die Tabelle „CommunitySelection“ geladen und von dort sortiert in den CityDictionary geschrieben.

Mittels der Funktion `MainForm::ShowCommunityTree()` werden die Gemeinden aus dem CityDictionary geholt und in den TreeView geschrieben. Dabei werden nur die Gemeinde-Attribute Kanton, Bezirk, Gemeindennamen und die Koordinaten geladen und als City-Attribut gespeichert.

Zu jedem Gemeindennamen sind die Koordinaten festgehalten. Anhand dieser x/y-Koordinaten wird die Gemeinde im Dictionary erkannt und kann für die Darstellung im TerrainViewer und im PropertyGrid ausgelesen werden. Die Koordinaten werden wie folgt in den Gemeindennamen geschrieben:

```
treeView1->Nodes[0]->Nodes[stateNodeIndex]->Nodes[districtNodeIndex]-  
>Nodes->Add(tmpCity->Name+ " [" + tmpCity->X + "," + tmpCity->Y + "]);
```

Beim Füllen der City-Objekte in den TreeView muss ihnen neu die Referenz auf das TreeNode ins Node-Attribut geschrieben werden. Dies dient der späteren Erkennung der Cities anhand der Nodes, beim Anwählen im TreeView oder TerrainViewer.

5.2.4.5 Darstellung der Gemeinde-Eigenschaften im PropertyViewer

Wenn im TerrainViewer oder TreeView eine Gemeinde selektiert wird, werden die Properties, die Gemeinde-Informationen, im PropertyViewer angezeigt. Dazu wird mittels Koordinaten das CityObjekt aus dem Dictionary geholt und die Properties des CityObjektes in das PropertyGrid der MainForm geschrieben. Die Methode dazu sieht wie folgt aus:

```
Void MainForm::ShowCityProperties(Point point){  
    CCity^ city= m_cityDict->default[point];
```

```
propertyGrid1->PropertySort= PropertySort::Alphabetical;
propertyGrid1->SelectedObject=city;
}
```

Für die eigentliche Darstellung der City-Properties im PropertyGrid ist eine einzige Zeile zuständig:

```
propertyGrid1->SelectedObject=city;
```

Hierbei ist zu beachten, dass die Art der Implementation der Properties in der City-Klasse Einfluss auf deren Darstellung im PropertyGrid hat: Werden in den Properties Setter-Methoden implementiert, so sind die Properties im PropertyGrid editierbar. Da dies in diesem Fall nicht erwünscht ist, wurden nur Getter-Methoden implementiert.

Bsp:

```
property int Citizen {
    int get() { return m_citizen; }
}
```

Members der Klasse City, welche nicht im PropertyGrid erscheinen sollen (Node und CityMode), dürfen nicht als Properties implementiert sein, sondern als normale Getter- und Setter-Methoden.

Bsp:

```
CityMode GetCityMode() { return m_cityMode; }
```

5.2.4.6 Selektieren der Gemeinden im TerrainViewer

Wird im TerrainViewer eine Gemeinde selektiert, so ist dies prinzipiell ein Maus-Klick in die GUI bzw. die PictureBox. Somit muss anhand der Control-Koordinaten der PictureBox zuerst evaluiert werden, ob sich an der Stelle überhaupt eine Gemeinde befindet. Wurde eine Gemeinde gefunden, so wird sie im TreeView selektiert.

Die Methode

```
Void MainForm::pictureBox1_MouseClick(System::Object^ sender,
System::Windows::Forms::MouseEventArgs^ e);
```

implementiert die Suche, ob sich bei den Mauskoordinaten eine Gemeinde befindet, und wählt diese allenfalls im TreeView an.

In der PictureBox wurden die Gemeinden mittels GDI+ wie folgt gezeichnet (links), dementsprechend muss derselbe Bereich wieder abgefragt werden (rechts):

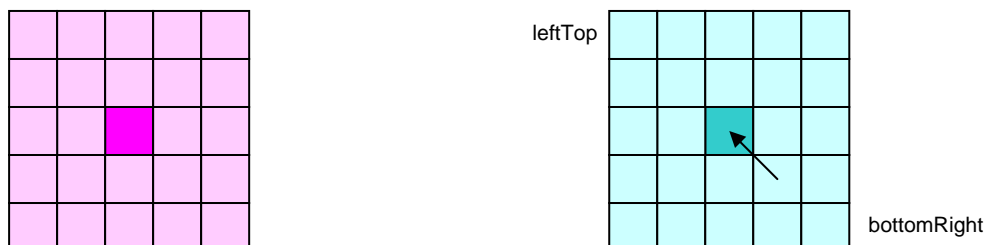


Abb. 9: Grösse des City-Rechtecks in der Picture Box und Abfragebereich beim Mausclick

Klickt nun ein Benutzer in die PictureBox, muss der entsprechende Bereich um die Mauskoordinaten für die Abfrage miteinbezogen werden. Dazu werden die Eckpunkte oben

links und unten rechts (leftTop bzw. bottomRight) in Landeskoordinaten umgerechnet. Anschliessend werden die Koordinaten der beiden Punkte auf 100 Meter Genauigkeit gerundet. Nun wird durch den ganzen CityDictionary iteriert und für jede Gemeinde geschaut, ob sich die Koordinaten zwischen diesen beiden Eckpunkten befinden. Die erste Gemeinde, welche im Dictionary gefunden wird, wird selektiert. Hier wird nun der Node-Member des City-Objekts verwendet, um im TreeView das entsprechende Node zu selektieren:

```
treeView1->SelectedNode= tmpCity->GetTreeNode();
```

5.2.4.7 Ausblick Selektieren der Gemeinden im TerrainViewer

Der Umstand, dass die erste Gemeinde, welche im CityDictionary gefunden wird, im TreeView selektiert und im PropertyGrid dargestellt wird, kann etwas ungenau sein, falls zwei Gemeinden sehr nahe beieinander liegen oder sich beim Zeichnen überschneiden. In der weiteren Entwicklung sollte hier eine Erweiterung implementiert werden, wo zu Beispiel alle Gemeinden, welche sich auf diesem Punkt befinden in einer Liste erscheinen und der Benutzer dann die gewünschte Gemeinde anwählen kann.

5.2.4.8 Selektieren der Gemeinden im TreeView

Wenn eine Gemeinde im TreeView selektiert wurde, wird anhand des Koordinaten-Strings, welcher dem Namen angehängt ist, die Gemeinde anhand ihrer x-/y-Koordinaten im Dictionary adressiert.

Die x-/y-Koordinaten werden in der Methode `MainForm::treeView1_AfterSelect(..)` wie folgt ausgelesen:

```
array<Char>^ chars= {'[', ',', '.', ' '};
array<String>^ splitString= nodeText->Split(chars);
Collections::IEnumerator^ myEnum= splitString->GetEnumerator();
String^ name= nullptr;
String^ x= nullptr;
String^ y= nullptr;
Point point;
CCity^ city= nullptr;
if (e->Node->Level>2){ //nur reagieren, wenn es sich um ein Namens-Node
handelt
    myEnum->MoveNext();
    name= safe_cast<String>(myEnum->Current);
    myEnum->MoveNext();
    x= safe_cast<String>(myEnum->Current);
    myEnum->MoveNext();
    y= safe_cast<String>(myEnum->Current);
    point.X=Convert::ToInt32(x);
    point.Y=Convert::ToInt32(y);
    ...
}
```

In `point` sind nun die Koordinaten der Gemeinde gespeichert, wie sie im CityDictionary bzw. der Datenbank gespeichert sind.

Beim Selektieren einer Gemeinde im TreeNode wird ein TreeView-AfterSelect Event aufgerufen, welcher wiederum die Methode `MainForm::ShowCityProperties(..)` aufruft und die City-Properties der Gemeinde im PropertyGrid anzeigt.

5.2.4.9 Verschieben des Terrains zur Gemeinde

Falls die selektierte Gemeinde ausserhalb des im TerrainViewer dargestellten Gebietes liegt, muss das Terrain zur Gemeinde hin verschoben werden. Dies wird mittels der Methode

MainForm::MoveToCity(Point point), welche die vorimplementierte Methode CTerrain::Move(..) verwendet, erreicht. Dabei wird das Terrain Tile-Reihe um Tile-Reihe (bzw. Spalte um Spalte) an die gewünschte Position verschoben, was nicht sehr schnell ist. Effizienter wäre die bei der Überarbeitung der Projektarbeit neu eingefügte Funktion CTerrain::Reposition(..) gewesen, bei welcher zuerst getestet wird, ob sich Tiles im „alten“ und im „neuen“ Terrain überdecken, und dann falls dies der Fall ist, die zusätzlichen, neuen Tiles geladen werden, oder, falls keine Überdeckung vorhanden ist, das Terrain vollständig abgebaut und an der neuen Position aufgebaut wird. Leider funktionierte diese (bestehende) Funktion nicht. Ihre Fehler konnten nur teilweise behoben werden, da es sich beim einen Problem um einen konzeptionellen Fehler handelte. (vergl 5.2.4.11) Aus diesem Grund blieb ich bei der Verwendung von CTerrain::Move(..).

In der Methode MainForm::MoveToCity(Point point) werden Distanzen in Landeskoordinaten berechnet, wobei zwischen metrischen und geografischen Koordinaten unterschieden wird. Diese Unterscheidung ist zwar aufwendig, aber notwendig, damit getestet werden kann, ob die Koordinaten der angewählten Gemeinde innerhalb des dargestellten Terrains liegen.

Die Methode CTerrain::Move(..) musste noch um die folgende Zeile korrigiert werden, damit der Ursprung des Terrains umplatziert wird:

```
theApp->Origin= this->GetNW()->GetNorthWest();
```

5.2.4.10 Probleme bei der Datenbankanbindung

Die Datenbankanbindung verlief leider nicht so reibungslos, wie ursprünglich erwartet. Die aufgetretenen Probleme sind auch in der weiteren Entwicklung zu beachten.

Es ging viel Zeit verloren, beim Versuch, die in Access vordefinierten Abfragen zu verwenden. Zuerst wollte man sie nicht ersetzen, da zukünftige Kunden mittels dieser Abfragen einfach neue Abfragen in Access hätten erstellen können. Wie sich aber herausstellte, wird diese Funktion in .NET mit Access-Datenbanken nicht mehr unterstützt, sondern nur für SQL Server.

Um die Abfrage für die Gemeinden nachzubauen, wäre eine parametrisierte Query mit benannten Parametern nötig gewesen, dies wird jedoch wie schon in 5.2.2 erwähnt, mit Access nicht unterstützt. [6][7]

5.2.4.11 Probleme beim Verschieben des Terrains mittels Reposition()

Wie schon in 5.2.4.9 erwähnt, verursachte das Verschieben des Terrains mittels CTerrain::Reposition(..) einige Probleme:

Im Falle, dass sich das „alte“ und das „neue“ Terrain nicht überschneiden wird das Terrain komplett abgebaut. Hierzu bedient sich die Funktion der Methoden CTerrain::DeleteRows(..) und CTerrain::DeleteColumns(..) welche ursprünglich im Rahmen der Projektarbeit für die Methode CTerrain::Move(..) implementiert wurden. Damals ging ich davon aus, dass das Terrain immer aus mindestens einem Tile besteht. (Verg. Kap. 5.1.1.2 der Projektarbeit-Dokumentation [8])

Für den Abbau des kompletten Terrains in CTerrain::Reposition(..) waren diese Methoden jedoch nicht geeignet, da es, beim Versuch, das allerletzte Tile abzubauen eine Exception warf.

Dies wird nun abgefangen, indem gleich zu Beginn der Funktionen getestet wird, ob alle Reihen bzw. alle Spalten in Terrain gelöscht werden sollen. Falls ja, wird die neue Funktion CTerrain::DeleteTerrain() aufgerufen. Diese baut das gesamte Terrain ab.

Das Terrain wird nun korrekt weggelöscht, jedoch funktioniert der Aufbau des Terrains ebenfalls nicht. Die folgenden Zeilen zeigen weshalb:

```
// compute intersection
Drawing::Rectangle inter = Drawing::Rectangle::Intersect(curArea, newArea);
if (inter.IsEmpty) {
// compute new origin
CCoord *origin = m_tileNW->GetNorthWest()->Translate(m_tileWidth*inter.X,
m_tileHeight*inter.Y);
```

Hier wird versucht, mittels den x/y-Koordinaten vom Intersektions-Rechteck `inter` den neuen Terrain-Origin in Landeskoordinaten zu berechnen. Doch in der if-Bedingung zuvor wird getestet, ob die Breite und Höhe von `inter` auf 0/0 gesetzt sind. Somit wird in `Translate` mit 0/0 multipliziert. Der Ursprung `m_tileNW` wird folglich um 0/0 verschoben, und bleibt am selben Ort. Das Terrain wird gar nicht verschoben. Leider reichte die Zeit nicht mehr, diese Funktion zu korrigieren. Dies sollte jedoch für die weitere Entwicklung beachtet werden.

5.2.4.12 Probleme beim Verschieben, wenn Textur- und Höhendaten vorhanden sind

Wie sich gegen Ende herausgestellt hat, gibt es Probleme beim Verschieben des Terrains, wenn Höhen- und Texturdaten vorhanden sind. Sind nur Höhendaten vorhanden, kann das Terrain problemlos zur entsprechenden Stelle verschoben werden. Sind jedoch Texturdaten vorhanden, gibt es beim Aufruf von `CTile::LoadDEM(..)`, wo die Höhendaten ins neue Terrain geladen werden, Probleme. In gewisse Bereiche des Terrains bzw. der Tiles werden ungültige Werte geladen. Ich habe dies testweise abgefangen, indem ich diese Werte auf 0 gesetzt und für die Darstellung im entsprechenden Wert eingefärbt habe, worauf folgende Ansicht generiert wurde:

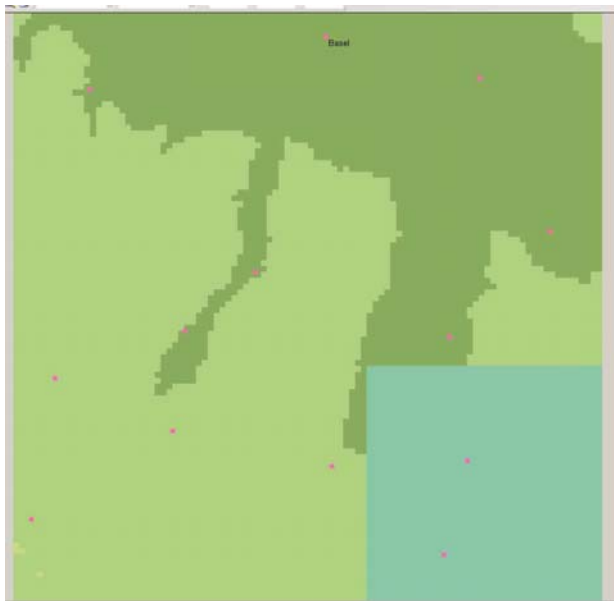


Abb. 10: Screenshot ungültige Höhendaten

Der türkisfarbene Bereich, zeigt wo die Höhendaten ungültig sind. Es scheint, dass entweder die Daten nicht richtig ausgelesen werden können, oder aus einem ungültigen Bereich ausgelesen werden. Da das Problem nur auftritt, wenn Höhen- und Texturdaten gleichzeitig vorhanden sind, vermute ich, dass etwas mit dem Bereich nicht stimmt, in den die Höhendaten dekomprimiert werden. Sicher ist, dass die Werte nicht richtig in den `VertexArray` eingefüllt werden, wenn Höhen- und Texturdaten vorhanden sind.

Das Problem tritt nur auf, wenn (in der hypsometrischen Darstellung) zu einem Bereich hin verschoben wird, für welchen keine Texturdaten vorhanden sind.

5.2.4.13 Ausblick Verschieben des Terrains mit Textur- und Höhendaten

Es ist notwendig, das in 5.2.4.12 genannte Problem zu lösen. Momentan wird abgefragt, ob sich die Gemeinde ausserhalb der in den Texturdaten vorhandenen Ansicht befindet. Nur wenn dies nicht der Fall ist, wird das Terrain zur Gemeinde hin verschoben.

5.3 Konzept und Umsetzung Serialisierung der Daten

5.3.1 Serialisierung in .NET

In .NET können managed Klassen einfach serialisiert werden, indem die zu serialisierende Klasse mit dem Attribut [Serializable] markiert wird. Members der Klasse, welche nicht mitserialisiert werden sollen, werden mit dem Attribut [Nonserializable] gekennzeichnet. Alle anderen Members werden mitserialisiert. Dies gilt auch für Pointers von unmanaged Klassen. Standardmässig, werden die Objekte mittels XML in eine SOAP-Datei geschrieben. Es wäre auch eine Serialisierung in eine reine XML-Datei möglich. SOAP unterstützt zusätzlich den Transport der Daten im Internet. Dies ist bei RA₂DIO eigentlich nicht nötig, weshalb eine Serialisierung in eine XML-Datei reichen würde. Da jedoch die Serialisierung mit SOAP von .NET besser unterstützt wird und die notwendigen Anforderungen erfüllt, habe ich mich für diese Art entschieden.

Die generierte Datei hat die Endung „.xml“. Da für RA₂DIO die Endung „.RA2“ gewünscht ist, kann die Datei problemlos mit dieser Endung abgespeichert werden und wird bei der Deserialisierung automatisch wieder als xml-/Soap-Datei erkannt.

```
Stream^ stream= File::Open("Gelaendedaten.RA2", FileMode::Create);
```

Für die Serialisierung und Deserialisierung müssen zuerst ein Stream, sowie ein SoapFormatter erzeugt werden. Der entsprechende Code sieht folgendermassen aus:

```
Stream^ stream= File::Open("Dateiname.xml", FileMode::Create);  
SoapFormatter^ formatter= gcnew SoapFormatter;  
formatter->Serialize(stream, this);  
stream->Close();
```

Analog verläuft die Deserialisierung. Die serialisierte Klasse wird zuerst in ein Objekt vom Typ „Object^“ deserialisiert und muss dann in die eigentliche Klasse gecastet werden.

```
Stream^ stream = File::Open( "Dateiname.xml", FileMode::Open);  
MeineKlasse^ myObject= nullptr;  
myObject= dynamic_cast<MeineKlasse^>(formatter->Deserialize(stream));  
SoapFormatter^ formatter = gcnew SoapFormatter;  
stream->Close();
```

5.3.2 Konzept Serialisierung in RA₂DIO

5.3.2.1 Zu serialisierende Daten

Bei der Serialisierung der Daten müssen einerseits Geländedaten und andererseits Benutzereinstellungen serialisiert werden.

Die Informationen zu den Geländedaten befinden sich vor allem in den Klassen:

- PGEDataSet
- PGEPackage
- Ra2dioApp

Folgende Daten sollen serialisiert werden:

PGEDataSet:

- Array mit PGEPackages
- Index, welche Packages aktiv sind
- Anzahl PGEs pro Package
- Koordinatensystem
- Vertex-Auflösung in X- und Y-Richtung

PGEPackage:

- Pointer auf die PGE-Datei
- Auflösung der PGE-Datei in Landeskoordinaten
- Ursprung der PGE-Datei in Landeskoordinaten

Ra2dioApp:

- Grösse des Viewers
- Grösse des Terrains
- Anzahl Vertices pro Tile
- NordWest-Koordinaten (Ursprung) von Terrain
- Boolesche Variablen, ob Höhendaten, Texturdaten und Landnutzungsdaten angezeigt werden sollen bzw. nach Abänderung: Variable dispMode

Bei den Benutzerdaten handelt es sich um Benutzereinstellungen, die in der GUI vorgenommen werden, z.B. Farbverläufe für die Darstellung der Höhendaten. Diese Informationen sind in der Klasse MainForm gespeichert.

Die zu serialisierenden Daten aus MainForm sind:

- Position von Detail- und TerrainTracker
- Grösse von Detail- und TerrainTracker
- Farbverläufe für Höhendaten-Darstellung
- Aktueller ViewerMode (Overview, Terrain oder Detail)

5.3.2.2 Anforderungen

Die Benutzereinstellungen müssen für jeden Benutzer separat gespeichert werden können. Je nachdem, welcher Benutzer angemeldet ist und die Applikation startet, sollen die entsprechenden Benutzereinstellungen geladen werden.

Die Benutzereinstellungen sollen gleich wie die anderen Daten (mit Ausnahme der Punktobjekte in der Datenbank) möglichst mobil gehalten werden. Wenn ein Benutzer an einem anderen Computer arbeitet, soll er die Dateien mit diesen Daten öffnen können, und somit seine Benutzereinstellungen ebenfalls anwenden können. Die Daten sollen damit klar vom Programm und der Maschine getrennt werden und portabel bleiben.

Wenn der Benutzer die Applikation startet, sollen automatisch die Benutzereinstellungen angewendet werden. Der Benutzer soll dann eine .RA2-Applikation öffnen können, welche die Geländedaten und die aktuellen Wellenausbreitungsdaten (falls vorhanden) beinhaltet.

Hat der Benutzer noch keine .RA2-Datei erstellt bzw. gespeichert, werden keine Geländedaten und Wellenausbreitungsdaten angezeigt.

Wenn der Benutzer die Applikation schliesst, soll er angefragt werden, ob er die aktuellen Daten speichern möchte. Verneint er dies werden die Änderungen verworfen. Ansonsten werden die Daten, wie wenn der „speichern“-Befehl angewendet wird, in eine neue .RA2-Datei geschrieben.

Es wäre möglich, gewisse Benutzereinstellungen mit der .RA2-Datei für die Geländedaten mitzuspeichern. Wird die Applikation geöffnet, werden zuerst die Benutzereinstellungen aus der serialisierten Benutzereinstellungs-Datei geladen. Wird dann ein .RA2-Gelände-Dokument geöffnet, welches andere Benutzereinstellungen beinhaltet, werden die zuvor geladenen überschrieben.

5.3.2.3 Art der Speicherung

Eine .RA2-Datei mit den Geländeeinstellungen ist gemäss 5.3.1 in Wirklichkeit eine mit dem SOAP-Framework erstellte XML-Datei und wird auch so verarbeitet. Die Benutzerdaten können direkt in einer .xml-Datei gespeichert werden.

5.3.2.4 Ort der Speicherung, Benennung

Prinzipiell können die Daten als XML-Files in einem beliebigen Ordner oder in der Registry abgelegt werden. Damit die unter „weitere Anforderungen“ genannte Portabilität der Daten gewährleistet ist, sollen sie nicht in die Registry geschrieben werden. Dies gilt auch für die Benutzerdaten

Als Ort für die Dateien eignet sich der Ordner „eigene Dateien“ gut, da der Benutzer so einfach darauf zugreifen, die Daten holen, oder falls nötig auch editieren, kann. In der Registry kann der Namen des aktuellen Benutzers abgefragt werden, aufgrund dessen, sollen im Ordner eigene Dateien die Benutzereinstellungen abgefragt werden. Im Ordner „Eigene Dateien“ soll ein Unterordner „Ra2dio“ erstellt werden, wo die Dateien abgelegt werden.

Die Dateien sollen beim Abspeichern vom Benutzer nach gewohnter Art benannt werden. Auf diese Weise lassen sich diverse Dateien mit Geländedaten abspeichern bzw. überschreiben.

5.3.2.5 Abhängigkeit der Daten

Es ist zu prüfen, inwieweit die Daten voneinander abhängig sind.

Die Geländedaten können auch ohne dass die Benutzereinstellungen bekannt sind angezeigt werden. Sind jedoch Benutzereinstellungen vorhanden, sollen diese angewendet werden. Zur Anzeige der Geländedaten kommen die Punktobjekte aus der Datenbank hinzu, falls diese benötigt werden (z.B. Gemeindenamen). Der Datenbankname soll in den Benutzereinstellungen gespeichert werden. Möglicherweise ist eine Kombination der Wellenausbreitungs- und der Geländedaten in einer Datei möglich. Die Benutzereinstellungen sollten dagegen auf jeden Fall getrennt davon in einer separaten Datei gespeichert werden.

5.3.3 Umsetzung Serialisierung in RA₂DIO

Die Umsetzung des Serialisierungskonzeptes stellte sich als komplizierter als angenommen heraus und konnte deshalb nur teilweise vorgenommen werden.

5.3.3.1 Probleme mit Serialisierung von bestehendem Code

Wie schon Herr Jäggi bei der Projektarbeit festgestellt hatte, ist die Serialisierung von mixed Code problematisch. [9]

Geplant war, die Serialisierung mit der in .NET unterstützten Serialisierung mittels [Serializable]- Attribut vorzunehmen. Dies erwies sich in der Art allerdings als unmöglich, da es sich beim übernommenen Code teilweise um unmanaged Code handelt.

In C++ wird unmanaged Code serialisiert, indem die entsprechenden Klassen von der Klasse CObject abgeleitet werden, und die entsprechenden Attribute dann in ein CArchive-Objekt serialisiert. (So wird es in der RA₃DIO-Applikation gemacht, welche ausschliesslich in unmanaged Code programmiert ist.) Sämtlich Recherchen im Internet und in der MSDN zeigen: Für mixedCode gibt es kein konkretes Serialisierungskonzept unter .NET.

Zuerst plante ich die Serialisierung ähnlich umzusetzen, wie dies bereits in der Projektarbeit getan wurde, indem ich vorschlug, eine bis zwei neue managed Klassen zu programmieren, in welche die Daten der unmanaged Klassen vor dem Serialisieren rausgeschrieben werden, und nach dem Deserialisieren wieder zurück in die unmanaged Klassen. In Absprache mit dem betreuenden Dozenten wurde diese Idee jedoch verworfen und versucht, die Serialisierung des mixed Codes anders zu lösen und die Klassen direkt zu serialisieren.

Als erstes stellte sich dabei heraus, dass die als „static“ deklarierten Klassen-Attribute der Klasse DataSet nicht serialisiert wurden. Deshalb musste zuerst die Klasse PGEDataSet in eine Instanz-Klasse umgeschrieben werden und ein Zeiger auf eine PGEDataSet-Instanz der Ra2dioApp-Klasse hinzugefügt werden. Ebenfalls entschied man sich, PGEDataSet in eine managed Klasse umzuschreiben. Dies würde ermöglichen, die Klasse unabhängig von der Ra2dioApp-Klasse zu serialisieren. Die Member von PGEDataSet, welche vom Typ CDist waren, mussten für das managed DataSet umgeschrieben werden in Pointers, da sie sonst nicht als Member einer managed Klasse akzeptiert wurden. Es stellte sich heraus, dass zwar die Pointer von CDist serialisiert wurden, jedoch nicht die Attribute der entsprechenden Klassen.

Auch die Klasse PGEPackage wurde für die Serialisierung mit .NET in eine managed Klasse umgeschrieben. Hier trat jedoch dasselbe Problem mit den Koordinaten-Klassen auf wie bereits in PGEDataSet. Das Umschreiben der Koordinaten-Klassen in managed Klassen erwies sich schliesslich als zu aufwendig im Rahmen dieser Diplomarbeit, da diverse Konzepte eingesetzt werden, welche im managed Code nicht übernommen werden können (u.a. eine Union innerhalb der Klasse CDist, Standard-Strings, sowie die Verwendung des Files Earth.h mit versch. Structs und der Klasse CGeoCoord, welche allesamt ebenfalls unmanaged sind).

Nach und nach bestätigte sich, was sich schon zu Beginn erahnen liess und das Fazit der Serialisierung in der Projektarbeit war: Die Serialisierung in .NET von mixed Code wird nicht unterstützt und lässt sich nur sehr umständlich umsetzen. Eine mögliche Lösung, wäre für alle unmanaged Klassen eine managed Wrapper-Klasse zu programmieren, was jedoch wieder ähnlich der oben genannten Idee mit den „Serialisierungsklassen“ ist.

Alternativ könnte man alle Klassen in managed Klassen umschreiben. Ob sich dieser Ansatz jedoch lohnt ist fragwürdig, wenn man die Probleme mit mixed Code über das ganze Projekt betrachtet.

5.3.3.2 Unternommene Schritte zur Serialisierung

Aufgrund der in 5.3.3.1 genannten Probleme kam das Projekt zeitlich stark in Verzug und die Aufgabe mit der Serialisierung konnte leider nicht gelöst werden.

Folgende Schritte wurden bisher unternommen:

- Umschreiben von PGEDataSet in eine Instanzklasse
- Umschreiben von PGEDataSet in eine managed Klasse
- Umschreiben von PGEPackage in eine managed Klasse
- Vorbereiten der Serialisierungs-Methoden :

```
bool LoadDocument();  
bool SaveDocument();
```

Da die Serialisierung bzw. Deserialisierung nicht realisiert werden konnte, konnte die Initialisierung mittels TinyXML[5] vorerst nicht ersetzt werden.

5.3.3.3 Ausblick Serialisierung

Falls man in der weiteren Entwicklung bei der Serialisierung von mixed Code bleibt, ist zu prüfen, ob während der Serialisierung bzw. Deserialisierung auf den Stream zugegriffen werden kann, und so unmanaged Daten von Hand in den Stream geschrieben werden könnten. Die Stream-Klasse bietet Methoden an, um in den Stream zu schreiben. [13] Es ist auch möglich die SOAP-XML-Serialisierung zu überschreiben. [10]

6 Geländevisualisierung

6.1 Auszug aus der Aufgabenstellung

Die aktuelle Version von RA₂DIO erlaubt lediglich die Visualisierung von Texturen. Da neben Textur- auch Gelände- und Landnutzungsdaten vorhanden sind, müssen diese auch visualisiert werden können. Grundsätzlich sollen in RA₂DIO folgende Darstellungsmodi vorhanden sein:

- Geländedarstellung mit hypsometrischen Farbtönen
- Geländedarstellung mit schattiertem Relief
- Landnutzungsdaten (separat ein- und ausschaltbar)
- Texturdaten (mit und ohne Transparenz und separat ein- und ausschaltbar)

Diese Darstellungsmodi sind alle auch in RA₃DIO vorhanden und können dort studiert werden.

Für hypsometrische Farbtöne und Falschfarbendarstellung bei der Landnutzung sind Erweiterungen der Benutzungsoberfläche notwendig, um die gewünschten Farben und Höhenbereiche individuell festzulegen. Die entsprechenden Einstellungen sind oft Abhängig von den zu visualisierenden Geländedaten.

Zur Geländedarstellung mittels schattierten Reliefs soll ein bekannter, einfacher und effizienter Schattierungsalgorithmus (z.B. Gouraud Shading) eingesetzt werden. Gouraud-Shading braucht neben den Höhendaten auch Normalen. Hierbei gilt es zu überlegen, ob diese Geländenormalen immer wieder neu berechnet oder in einem Preprocessing-Schritt berechnet und dann persistent und effizient gespeichert werden.

Ziele

Organisieren Sie Gelände- und Landnutzungsdaten.

Studieren Sie geeignete Shading-Algorithmen und setzen Sie einen nach Absprache mit Ihrem Betreuer um.

Realisieren Sie die geforderten Geländedarstellungsmodi und passen Sie dazu das GUI entsprechend an.

6.2 Füllen der Höhendaten in die Vertices

Im bestehenden Code wurden mit der Funktion `CTile::LoadDEM(..)` bereits Höhendaten aus der DEM-PGExt-Datei, wo sie als Grauwert codiert sind, in die Vertices der Terrain-Tiles geladen. Allerdings sind die Höhenangaben in der PGExt-Datei in Dezimeter gespeichert und wurden in der übernommenen Version auch so in die Vertices gefüllt. Dort werden die Höhendaten jedoch in Metern gespeichert. Deshalb musste diese Methode angepasst werden, indem man die ausgelesenen Höhendaten vor der Speicherung in den Vertices durch 10 teilt. Die angepasste Codezeile sieht wie folgt aus:

```
it.GetNext().SetHeight(((float)*column) / 10);
```

Weiter stellte sich heraus, dass das Attribut `m_showDEM` der Klasse „Ra2dioApp“ innerhalb des Ra2dioApp-Konstruktors auf true gesetzt werden musste, damit die Höhendaten geladen werden. In der Funktion `Ra2dioApp::Init()` wird `m_showDEM` auf false gesetzt, falls bei der Initialisierung kein PGEPackage für die Höhendaten erstellt wurde:

```
if (!theApp->PGEDataset->DEMisAvailable()) m_showDEM = false;
```

Dieser Ansatz erfordert jedoch, dass das Attribut zuerst auf true gesetzt wurde, denn in der Funktion `CTerrain::ShowDEM(..)`, welche die Höhendaten in die Vertices im Terrain

einfüllt, wird abgefragt, ob `theApp->ShowDEM` true ist. Wenn nicht, werden die Höhendaten nicht in die Vertices gefüllt. Wird `m_showDEM` mit false initialisiert und mittels oben genannter Zeile nicht verändert, kommt die Variable gar nie auf true.

Ich habe das Problem behoben, indem ich `m_showDEM` im Konstruktor auf true gesetzt habe, wie dies im übernommenen Prototypen mit den Texturdaten bereits gemacht worden war. Alternativ könnte man den Konstruktor abändern auf:

```
Ra2dioApp::Ra2dioApp()  
    :   m_showDEM(true),   m_showTexture(true),   m_showLandUsage(true),  
  m_status(D_New){ ... }
```

Das Konzept mittels der Attribute `m_show[Datenquelle]` wurde überarbeitet (vergl 8.2).

6.3 Konzept Geländevisualisierung

Für die Geländevisualisierung habe ich mich im Rahmen dieser DA besonders mit der Visualisierung der Höhendaten auseinandergesetzt. Die Höhendaten können auf zwei verschiedene Arten visualisiert werden: als schattiertes Relief und mit hypsometrischen Farbtönen.

6.3.1 Hypsometrische Farbtöne

Für die hypsometrischen Farbtöne reicht es, die Höhendaten aus den Vertices auszulesen. Es wird ein entsprechendes Höhenraster definiert, innerhalb welchem einem bestimmtem Höhenbereich eine Farbe zugeordnet wird. Gemäss diesen Farben werden die Pixel im Höhenbild eingefärbt. Da sich oftmals zwischen den einzelnen Vertices mehrere Pixel befinden, muss bei diesen die Höhe interpoliert werden.

6.3.2 Reliefschattierung

6.3.2.1 Shading-Algorithmen

Für die Reliefschattierung müssen die Flächen zwischen den Höhenpunkten mit einem entsprechenden Shading-Algorithmus eingefärbt werden.

Der Flat-Shading Algorithmus ist sehr schnell, da die Schattierung nur an einem Vertex pro Polygon angewendet wird und keine Normalen berechnet werden müssen. Dafür entsteht nachteilig ein so genannter „Machband-Effekt“, welcher die Kanten der einzelnen Polygone deutlich erkennen lässt.

Beim Gouraud-Shading Algorithmus, einem weichen Schattierungsalgorithmus, wird die Krümmung der Fläche berücksichtigt. Dazu müssen für alle Ecken des Polygons die Normalen berechnet werden. Anhand diesen Ecknormalen werden die einzelnen Pixel innerhalb des Polygons interpoliert und entsprechend eingefärbt.

Beim Phong-Shading Algorithmus werden zusätzlich noch die einzelnen Normalen für jedes Pixel berechnet. Damit wird die Oberflächenwölbung noch besser simuliert als beim Gouraud-Shading.

Aufgrund der kürzeren Rechenzeit sowie der weichen Schattierung soll für die Reliefschattierung der Gouraud-Shading Algorithmus verwendet werden.

Weitere Details über die Shading-Algorithmen können dem Skript von Marcus Hudritsch entnommen werden. [11]

6.3.2.2 Reliefschattierung mittels Gouraud

Bei der Gouraudschattierung werden die Bereiche zwischen den Vertices (logisch) trianguliert. Anhand dieser Dreiecke werden die Normalen an den Vertices berechnet. Aus den Normalen an den Vertices wird mittels Gouraud-Shading die Farbe eines Dreiecks berechnet. Die folgenden Erläuterungen stützen sich auf das Skript von Marcus Hudritsch [11] und wurden auf die Terrain- bzw. Vertex-Struktur angepasst.

Berechnung der Ecknormalen und der Flächenfarbe:

Die Fläche zwischen den Vertices innerhalb eines Tiles wird (logisch) trianguliert. Diese Dreiecke werden dann in den berechneten Farben eingefärbt, wobei für jedes Pixel die Intensität anhand der Eckpunkte interpoliert wird.

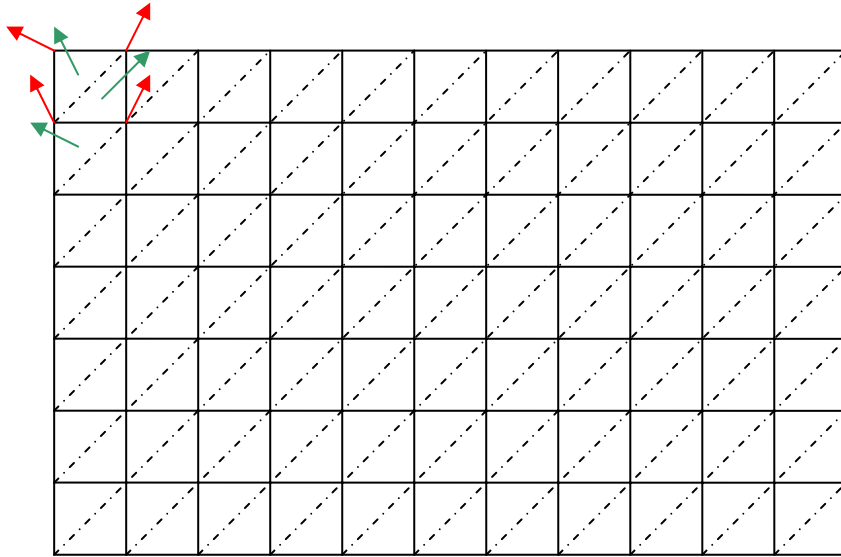


Abb. 11: Logisches Triangulieren zwischen den Vertices.

Zuerst muss für jedes Dreieck die Flächennormale berechnet werden:

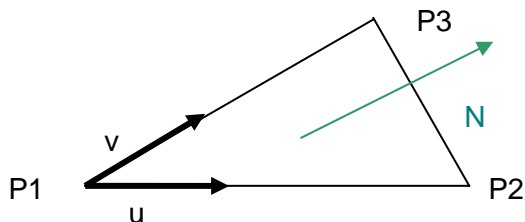


Abb. 12: Berechnung der Flächennormalen

Berechnungsformeln für die Flächennormale N aus den Eckpunkten P1, P2 und P3:

$$u = P2 - P1 \quad v = P3 - P1 \quad N = u \times v$$

Für die Berechnung der Normalen an den Vertices (bzw. den Eckpunkten der Dreiecke), werden die Flächennormalen der sechs umliegenden Dreiecke gebraucht:

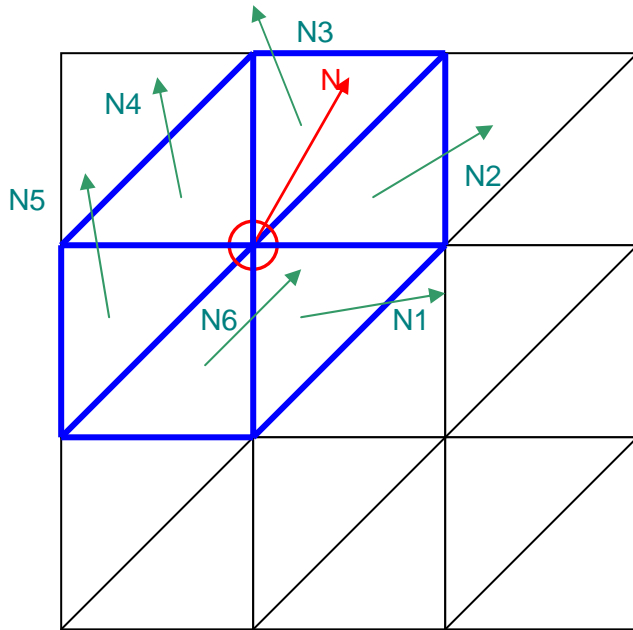


Abb. 13: Berechnung der Ecknormalen

Berechnungsformel für die Ecknormale N aus den Flächennormalen N1, N2...,N6:

$$N = (N1 + N2 + N3 + N4 + N5 + N6) / (| N1 + N2 + N3 + N4 + N5 + N6 |)$$

Die Eckpunkte am Rand sind dabei abzufangen. Die Eckpunkte für die Flächen ausserhalb des bekannten Bereichs sind als Null anzunehmen.

Die Farbe der Dreiecksfläche bzw. deren Pixel wird nun mittels Interpolation anhand der Dreiecks-Eckpunkte ermittelt.

Um den Farbwert an einem Punkt zu ermitteln wird das Skalarprodukt des Normalenvektors des Eckpunktes mit dem Lichtvektor L berechnet.

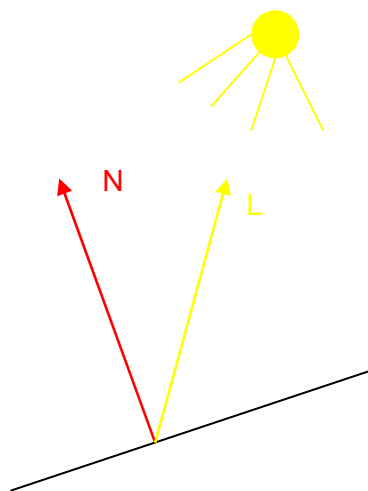


Abb. 14: Berechnung eines Farbwertes anhand des Lichtvektors und der Normalen

$N \cdot L = \text{Farbwert}$

Wobei die Vektoren N und L normalisiert sein müssen.

Für RA₂DIO wird die Lichtquelle in Nordwesten angenommen.

Anhand dieser Farbwerte an den Eckpunkten der Dreiecke (bzw. Vertices), können nun die Pixel zwischen den Eckpunkten mit dem Scanlinealgorithmus interpoliert werden:

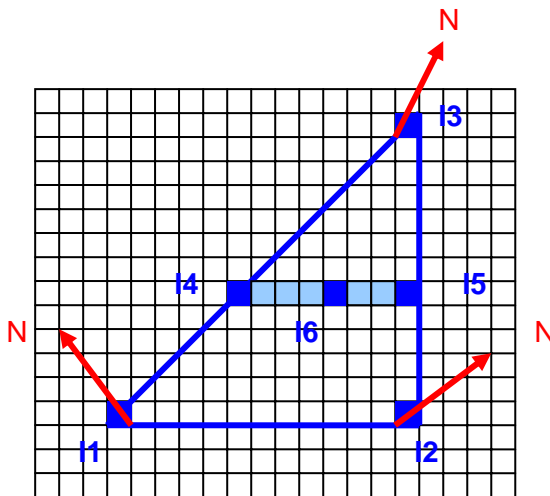


Abb. 15: Interpolation der Pixel mittels Scanline-Algorithmus

Dabei wird Pixelreihe um Pixelreihe interpoliert. Zuerst werden die Punkte am Dreiecksrand (14, 15) berechnet, danach werden für die dazwischenliegenden Pixel in der gleichen Reihe (hellblau) die Farbwerte linear interpoliert.

6.3.2.3 Konzept für die Realisierung der Reliefschattierung in RA₂DIO

Für jeden Vertex muss also die Normale berechnet werden. Da dies eine rechenintensive Operation ist, wäre es sinnvoll, die Normalen der Eckpunkte einmalig in einem separaten Projekt zu berechnen und in einer PGFExt-Datei zu speichern. Somit würde für jeden Vertex bzw. für jeden Grauwert in der PGFExt-Datei mit den Höhendaten eine Normale in der neuen PGFExt-Datei existieren. Für die Visualisierung könnten dann die Normalen aus der PGFExt-Datei ausgelesen und für die Berechnung der Flächenfarbe verwendet werden.

Alternativ könnten die Farbintensitäten an den Vertices bzw. für die Pixel in der Höhendaten-PGFExt-Datei direkt berechnet und dann in einer neuen PGFExt-Datei gespeichert werden.

Für die Berechnung der Normalen an den Eckpunkten werden die Flächennormalen der Dreiecke benötigt. Es stellt sich die Frage, wie man diese Flächennormalen am besten zwischenspeichert, damit bei der Berechnung der Eckpunkt-Normalen gleich klar ist, welches die umliegenden Flächen sind.

Ein möglicher Lösungsansatz für dieses Problem wäre, die Flächennormalen in den Vertices zwischen zu speichern. Somit würde Vertex V1 die Normale N1 speichern, der Vertex V4 die Normalen N2 und N3. Für die Berechnung der Normalen bei den Vertices sind somit rasch die angrenzenden Flächennormalen gefunden.

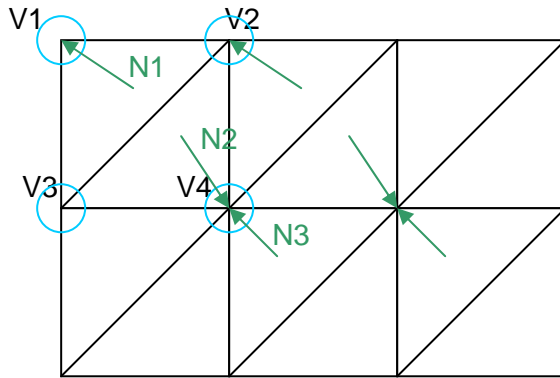


Abb. 16: Zwischenspeichern der Flächennormalen

Für die genannten Berechnungen werden Vektoren gebraucht. Es gibt die Möglichkeit selber eine Vektor-Klasse inklusive der diversen Vektoroperationen zu implementieren, oder vordefinierte, dreidimensionale Vektoren zu nehmen. Bisher wurde dazu die Bibliothek LEDA verwendet, welche aber für RA₂DIO nicht mehr zur Verfügung steht. GDI+ bietet nur zweidimensionale Vektoren an. Im Microsoft.WindowsMobile.DirectX.Direct3D Namespace vom Microsoft Compact Framework stehen dreidimensionale Vektoren zur Verfügung. Allerdings kann das Compact Framework nur in Zusammenhang mit mobilen Applikationen verwendet werden und steht somit für RA₂DIO nicht zur Verfügung.

Aus diesem Grund wurde entschieden, das DirectX 9.0 Framework zu verwenden. Es würde sich anbieten, das Managed DirectX Framework zu verwenden. Hier stösst man allerdings auf das Problem, dass dies nicht mehr unterstützt wird in den Versionen vom Oktober 2006 und Dezember 2006. [12] Die letzte Version, welche Managed DirectX unterstützt ist die Version vom August 2006. Microsoft empfiehlt diese Version zu verwenden (oder für die Gamenentwicklung XNA). Allerdings stellte ich fest, dass die Dokumentation von Managed DirectX in der MSDN nicht mehr vollständig bzw. veraltet ist. Z.B. funktionieren teilweise die Links nicht mehr. Aus diesem Grund ist es fraglich, ob man wirklich Managed DirectX verwenden will, oder nicht die neuste Version vom Dezember 2006. Wobei man wieder auf das Problem mit dem mixed Code stösst.

Das DirectX-Framework bietet auch zum Zeichnen diverse Funktionen an. Es ist in der weiteren Entwicklung zu prüfen, ob Dreiecke anhand von Farbwerten an den drei Eckpunkten direkt gezeichnet werden können. Dann könnte auf das pixelweise einfärben mittels Scanlinealgorithmus verzichtet werden.

6.4 Umsetzung Geländevisualisierung mittels hypsometrischen Farbtönen

Im Rahmen dieser Arbeit wurde beschlossen, die Visualisierung der Höhendaten mittels hypsometrischen Farbtönen zu implementieren. Wie schon unter 6.3.1 beschrieben, können dazu direkt die Höhenangaben in den Vertices verwendet werden.

6.4.1 Evaluation: Visualisierung mittels GDI+ oder Bitmap

Die hypsometrischen Höhenangaben können mittels GDI+ direkt in die PictureBox gezeichnet werden oder in die TerrainBitmap, in welche auch die Texturdaten geladen werden. Zuerst entschied ich mich für den ersten Lösungsansatz mit GDI+. Dies ist jedoch sehr rechenintensiv und hat den Nachteil, dass beim Verändern des Hauptfensters Zwischenräume zwischen den einzelnen Tiles entstehen.

Aus diesem Grund wurde entschieden, in die existierende Terrain-Bitmap zu zeichnen. Die Terrain-Bitmap ist in der MainForm-Klasse gespeichert, die Informationen dazu wie z.B. Breite und Höhe in Landeskoordinaten befinden sich in der Terrain-Klasse.

6.4.2 Zeichnen der Höhendaten in die Terrain-Bitmap

In die bestehende Methode `CTerrain::CreateBitmap()` sowie in die Methode `CTerrain::CreateDEMBitmap()` wurde der Aufruf der neuen Methode `CreateDEMBitmap(bitmap)` eingebaut. Der Funktion wird dabei die `TerrainBitmap`, welche vor dem Aufruf in der `CreateBitmap()` Methode erstellt wurde, als Parameter übergeben.

In `CTerrain::CreateDEMBitmap(Bitmap^ bitmap)` wird durch die ganze `TerrainBitmap` iteriert und für jedes Pixel aus dem entsprechenden Vertex die Höhenmeter ausgelesen, die entsprechende Farbe ermittelt und aus dem `m_colors` Array der `MainForm` die zugehörige Farbe geholt, um das Pixel mit `SetPixel()` einzufärben.

Hier wäre eine Interpolation für die Pixel zwischen den Vertices nötig. Da jedoch im übernommenen Code die `TerrainBitmap` so gross erzeugt wurde, dass für jedes Pixel der `TerrainBitmap` ein Vertex vorhanden ist, kann auf eine Interpolation verzichtet werden.

```
CTerrain::CTerrain(...) {
    (...)
    if (theApp->PGEDataset->TextureIsAvailable()) {
        // use texture PGE to determine bitmap size
        (...)
    } else {
        m_tileBitmapSize.Width = m_tileSize.Width;
        m_tileBitmapSize.Height = m_tileSize.Height;
        (...)
    }
    (...)
    // compute bitmap size
    m_bitmapSize.Width = m_tileBitmapSize.Width*m_size.Width;
    m_bitmapSize.Height = m_tileBitmapSize.Height*m_size.Height;
    (...)
}
```

6.4.3 Darstellung der Höhendaten in der Overview-Bitmap

Im OverviewModus sollen die Höhendaten ohne langen Rechenaufwand in einer Übersicht dargestellt werden. Dazu wurde beschlossen, die `PGFExt-Datei` mit den Höhendaten direkt in die Overview-Bitmap zu kopieren, analog wie dies bereits mit den Texturdaten vorimplementiert war.

6.4.4 Darstellung der Höhenkoordinaten in der Statusbar

Bisher wurden bei Mausbewegungen über den `TerrainViewer` bzw. die `PictureBox` der GUI, die Landeskoordinaten berechnet und in der Statusbar angezeigt. Neu musste die gleiche Anzeige für die Höhenkoordinaten implementiert werden.

Dazu wird zuerst die Mausposition in Control-Koordinaten (Bildschirmkoordinaten innerhalb der `PictureBox`) ausgelesen. Die Mausposition muss in Landeskoordinaten bzw. geografische Koordinaten umgerechnet werden. Mit den berechneten Koordinaten wird abgefragt, ob sich die Maus überhaupt im Terrain befindet. Ist dies nicht der Fall, so wird die z-Koordinate auf 0 gesetzt.

Ansonsten wird die Funktion `MainForm::GetZCoord(Point mousePoint)` aufgerufen. In einer ersten Version berechnete diese Funktion die z-Koordinate in dem mit Landeskoordinaten interpoliert wurde (vergl. Funktion `MainForm::GetZCoordByCoord(CCoord* mousePosCoord)`). Dies stellte sich jedoch als zu rechenintensiv heraus, da zu oft in Landeskoordinaten umgerechnet wurde, weshalb die

Methode nochmals umgeschrieben werden musste und neu mit Bildschirmkoordinaten interpoliert wird.

Da sich die Mausposition nicht zwingend bei einem Vertex befindet, muss die Höheninformation aus den umliegenden vier Vertices interpoliert werden:

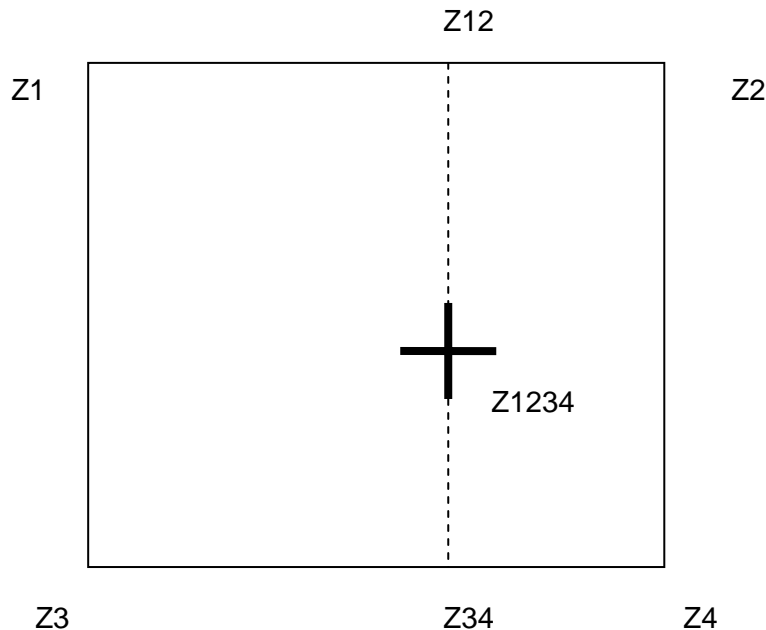


Abb. 17: Bilineare Interpolation des Höhenwertes zwischen 4 Vertices

Dabei wird bilinear interpoliert.

In x-Richtung werden dazu zuerst mittels linearer Interpolation die Punkte Z12 und Z34 aus Z1 und Z2 ermittelt. Danach werden Z12 und Z34 in y-Richtung linear interpoliert, woraus die Höhe am Punkt Z1234 berechnet wird.

Die x- und y-Koordinaten von Z1 und Z2 bzw. Z12 und Z23 sind dabei in Control-Koordinaten ausgedrückt. Die z-Koordinaten von Z1, Z2, Z3 und Z4 werden aus den entsprechenden Vertices ausgelesen. Das Tile wird mittels der Funktion `CTerrain::FindTileByCoord(CCoord* coord)` gefunden (vergl. 5.4.4). Als Parameter wird die Mausposition in Landeskoordinaten umgerechnet mitgegeben. Der Index des VertexArray wird direkt aus den Controlkoordinaten berechnet. Dies geschieht analog der Methode `CTerrain::FindTileByCoord(CCoord* coord)`. (vergl. 5.4.5)

Für die Interpolation wird das Verhältnis der x- bzw. y- Koordinaten benötigt.

Bsp. für die x-Koordinaten:

Z3: $x = 4$

Z4: $x = 5$

Z34: $x = 4.6345$

Die Differenz von 0.6345 wird auf verwendet um in x-Richtung zu interpolieren. Analog wird in y-Richtung interpoliert.

Wenn sich ein Vertex am linken oder unteren Rand des Tiles befindet, so müssten für die Interpolation mit den restlichen Vertices die Nachbartiles geladen werden. Im Moment wurde dies jedoch temporär so gelöst, dass in dem Fall nicht interpoliert, sondern der Vertex oben links verwendet wird.

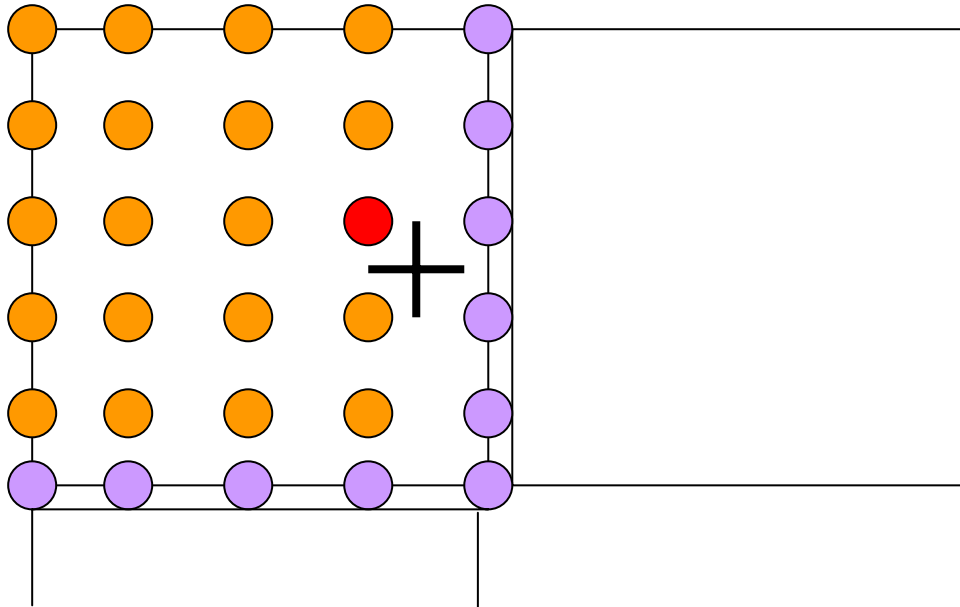


Abb. 18: Vertex-Anordnung

Vergleiche Bsp. Abb. 18: die orangen Vertices gehören zum Tile links. Die violetten Vertices zum Tile rechts bzw. unterhalb.

Für die dargestellte Mausposition wird hier nicht interpoliert, sondern die Höhenangabe des roten Vertex übernommen.

6.4.5 Tiles finden im Terrain

Immer wieder stösst man auf das Problem, dass man sich irgendwo im Terrain befindet und das Tile bzw. den nächstliegenden Vertex auslesen muss. Aus diesem Grund wurde die Methode `CTerrain::FindTileByCoord(CCoord* coord)` implementiert, welche einen Zeiger auf das betroffene Tile liefert. Dabei wird die x- und y-Position innerhalb des Tiles in Vertices berechnet. Anhand dieser wird das entsprechende Tile mittels der vorimplementierten Funktion `CTerrain::FindTile(TileT x, TileT y)` gesucht und zurückgeliefert.

Um die x- bzw. y-Koordinate in Tiles zu finden, wird die Distanz zwischen der Input-Koordinate (x,y) und dem TerrainOrigin (beide in Landeskoordinaten berechnet). Diese Distanz wird dann durch die Breite (width) bzw. durch die Höhe (height) in Landeskoordinaten geteilt und liefert die Position in Tiles (tX, tY).

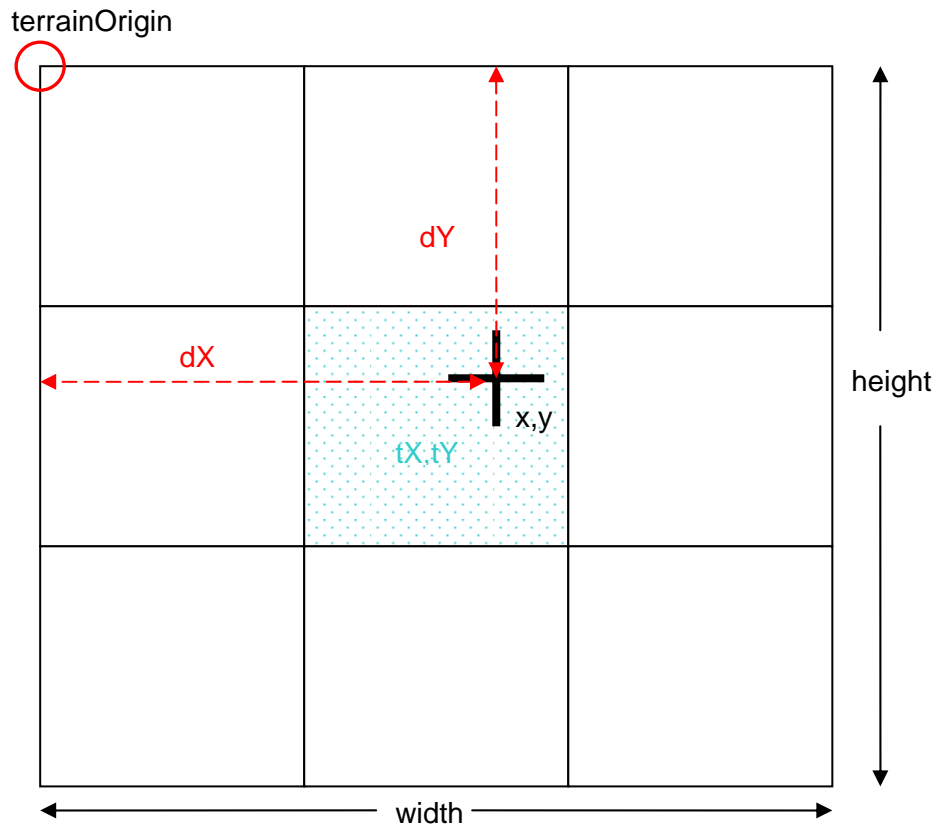


Abb. 19: Tilesuche im Terrain mittels Landeskoordinaten

6.4.6 Vertices finden im Tile

Analog verhält es sich mit dem Ermitteln des nächstliegenden Vertex oben links.

Die Methode um einen Vertex innerhalb eines Tiles anhand der Landeskoordinaten zu ermitteln, wurde neu in der Klasse CTile implementiert: `CTile::FindVertexInfoByCoord(CCoord* coord)`. Sie liefert einen Zeiger auf den betroffenen Vertex.

Auch hier werden die Vertex-Koordinaten (vX , vY) berechnet, indem die Distanzen zwischen dem Tile-Ursprung SE in Landeskoordinaten und den Inputkoordinaten (x,y) in x - und y -Richtung durch Tile-Höhe und -Breite ($resY$, $resX$) in Landeskoordinaten geteilt werden. Wieder werden durch die Integerdivision ganze Zahlen geliefert. Mittels x - und y -Koordinate in Vertices (vX,vY) kann der Index im eindimensionalen VertexArray berechnet werden: $Index = vY * AnzahlVerticesInBreite + vX$. Mit diesem Index wird der Vertex aus dem VertexArray des Tiles geholt und ein Zeiger darauf zurückgeliefert.

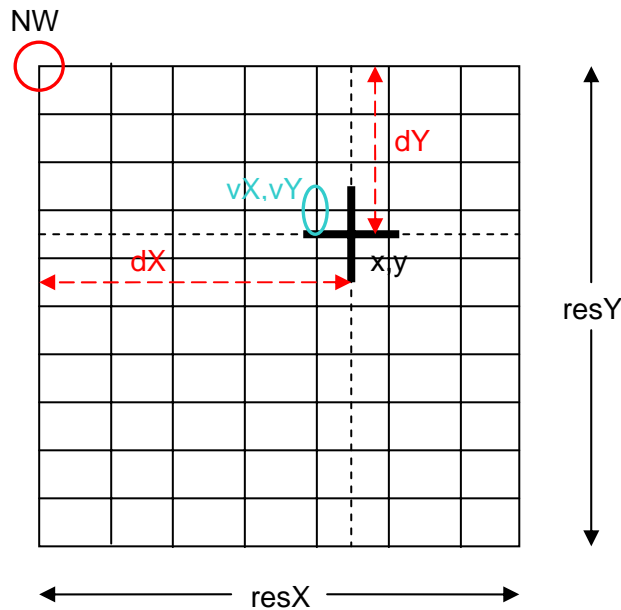


Abb. 20: Vertexsuche mittels Landes-kordinaten

6.5 Ausblick Visualisierung der Höhendaten

Im Moment wird die Terrain-Bitmap mit der Grösse erstellt, dass pro Pixel ein Vertex vorhanden ist. Diese Auflösung muss verbessert werden, indem z.B. 10 Pixel zwischen zwei Vertices liegen. Dementsprechend ist die Farbe für die hypsometrischen Farbtöne zwischen zwei Vertices zu interpolieren. Die Interpolation mittels Nachbarfiles wird im Moment gemäss 6.4.4 weggelassen und muss noch implementiert werden.

Die Visualisierung der Reliefschattierung muss noch gänzlich gemäss obigem Konzept und mittels DirectX (oder äquivalenter Technologie) implementiert werden.

7 GUI Erweiterungen

7.1 Auszug aus der Aufgabenstellung

Die aktuelle grafische Benutzeroberfläche (GUI) in RA₂DIO ist noch sehr rudimentär. Die wichtigsten Grundstrukturen sind jedoch vorhanden: Menubar, Toolbar, Statusbar, Terrain-Viewer, Colorange-Viewer, Tree-Viewer, Tab-Viewer.

Wenig häufige Interaktionsschritte werden über das Menü vorgenommen. Das Menü verfügt über Tastaturkürzel für die wichtigsten Aufgaben, wie z.B. das Öffnen und Speichern von RA2-Dokumenten.

Im Toolbar werden wiederum die wichtigsten Menüpunkte mittels aussagekräftigen Symbolen angeboten. Zudem soll der Toolbar die wichtigsten Auswahl- und Eingabeboxen enthalten.

Der Statusbar gibt ständig Auskunft über den aktuellen Status der Software. So werden zum Beispiel die Koordinaten des Cursors im Gelände-Viewer dargestellt. Der Statusbar ist besonders dann wichtig, wenn Aufgaben im Hintergrund erledigt werden, welche länger dauern und an die Geduld des Benutzers appellieren. Der Verlauf dieser Hintergrundarbeiten wird in einem Progressbar innerhalb des Statusbar dargestellt.

Der Colorange-Viewer befindet sich immer am rechten Rand des Terrain-Viewers. Im Colorange-Viewer sollen die Falschfarbskalen für die Landnutzung, die Hypsometrie und die Wellenausbreitung dargestellt werden.

Im Tree-Viewer sollen Datenbank-Datensätze hierarchisch dargestellt werden. Hier soll die Benutzerin alle Ortschaftsdaten, alle Basisstationen, alle Antennen und weitere Daten direkt anwählen können. Die Eigenschaften des ausgewählten Objektes werden im Property-Grid-Viewer dargestellt.

Im Property-Grid-Viewer sollen die Eigenschaften eines selektierten Objektes (z.B. Basisstation) in Textform dargestellt werden. Hier sollen sich auch die Eigenschaften eines selektierten Objektes verändern lassen. Dies entspricht der heute üblichen Darstellung eines Property-Grids. Informationen zu Property-Grids finden Sie auch unter (www.codeproject.com).

Ziele

Füllen Sie den Tree-Viewer mit den Daten aus der Ortschaftsdatenbank auf und realisieren Sie eine bidirektionale Verbindung zwischen Tree-Viewer und Terrain-Viewer. Ortschaften, welche momentan im Terrain-Viewer sichtbar sind, sollen im Tree-Viewer farblich markiert werden. Selektiert die Benutzerin eine Ortschaft im Tree-Viewer, so soll sie die Möglichkeit erhalten, den sichtbaren Geländeausschnitt so zu verändern, dass die gewählte Ortschaft im Terrain-Viewer sichtbar wird. Umgekehrt, selektiert die Benutzerin eine Ortschaft im Terrain-Viewer, so soll die gleiche Ortschaft im Tree-Viewer auch selektiert werden.

Im Property-Grid-Viewer sollen immer die Eigenschaften des aktuell selektierten Objektes dargestellt werden. Beachten Sie dabei, dass viele Objekte sowohl im Terrain-Viewer als auch im Tree-Viewer selektiert werden können. Werden die Eigenschaften eines selektierten Objektes verändert, so müssen die neuen Daten auf Wunsch in die Datenbank zurück geschrieben werden können.

Implementieren Sie den Colorange-Viewer zumindest für die hypsometrischen Farbtöne. Die angezeigten Farben und Höhenbereiche müssen individuell gewählt und gespeichert werden können.

Die Verschiebung des aktuell sichtbaren Geländeausschnittes im Terrain-Viewer ist momentan sehr rudimentär. Erstellen Sie hier lediglich ein Interaktionskonzept, wie eine sinnvolle Benutzerführung aussehen könnte, um den aktuellen Ausschnitt sowohl in kleinen Schritten als auch in grossen Schritten zu verändern.

7.2 Interaktion im TerrainViewer

Im TerrainViewer erlauben der Detail-Tracker, sowie der Terrain-Tracker die Navigation im Terrain. Über die Menu-Leiste kann ein Modus angewählt und so in die Terrain-, die Detail- oder den OverviewModus gezoomt werden. Mit Drücken der mittleren Maustaste, können die Tracker in der Overview-Ansicht verschoben und in ihrer Grösse verändert werden.

7.3 Umsetzung Visualisierung Gemeindedaten

Vergl. Kap. 5.2.4 Darstellung der Gemeindedaten

7.3.1 Notwendige Änderungen an der bestehenden GUI

Wurde bisher im OverviewModus mit der Maus in den Bereich der Terrain- bzw. Detail-Tracker geklickt oder die Maus über die Tracker bewegt, so erschien der Mauszeiger als Kreuz. Dies erschwerte das Anwählen der Gemeinden im OverviewModus, weshalb hier neu zusätzlich der mittlere Mauszeiger gedrückt werden muss, damit der Mauszeiger als Kreuz erscheint. Ansonsten bleibt der Mauszeiger ein Pfeil, was das Anwählen der Gemeinden erleichtert.

7.4 Umsetzung GUI-Erweiterung für Höhendarstellung mittels hypsometrischen Farbtönen

Für die Darstellung der Höhendaten im TerrainViewer vergl. Kap. 6.4.2

Die Höhendaten werden in einem bestimmten Raster in einer entsprechenden Farbe dargestellt. Dazu wird in der GUI der Colorange-Viewer angezeigt. Der Benutzer kann in ein Farbfeld klicken & die entsprechende Farbe mittels Color-Dialog editieren. Die neue Farbe wird unmittelbar im Terrain-Anzeigebild übernommen.

7.4.1 Colorange-Viewer

Der Colorange-Viewer ist aus Labels aufgebaut, welche dynamisch erzeugt werden. Im File „Constants.h“ habe ich die Variablen MaxHeight (für die maximale Höhe) sowie HeightRange definiert, in welcher abgespeichert ist, wieviele Höhenmeter die selbe Farbe erhalten. Anhand dieser Variable wird ausgerechnet, wieviele Labels erstellt werden müssen und in der Variablen `m_labelNo` in der Klasse MainForm gespeichert. In den Variablen `m_labels` und `m_heightLabels` in MainForm werden die Labels für die Farbwahl bzw. für die Labels für deren Beschriftung in Höhenmeter in Arrays gespeichert. In der Methode `MainForm::InitLabels()` werden beide Arrays initialisiert.

In der Variablen `m_colors` in MainForm werden die Farben für die Farbwahl-Label in einem Array gespeichert. Dieser Array enthält gleich viele Elemente wie `m_labels` und wird in der Methode `MainForm::InitColors()` initialisiert. Im Moment geschieht dies statisch. Zukünftig sollten die Farben dort ebenfalls dynamisch erzeugt werden. Dies ist am besten mit dem HSB-Farbmodell umsetzbar.

Wird nun eine Höhenangabe aus den Vertices ausgelesen, wird berechnet, in welchem Höhenbereich sie sich befindet, und die dementsprechende Farbe aus dem `m_colors` Array ausgelesen um das Pixel in dieser Farbe einzufärben.

Farbänderung mit dem Colorange-Viewer:

Mittels Klicken in ein Farblabel des Colorange-Viewers kann der Benutzer die Farbe eines Höhenbereichs ändern. Dazu wurde dem ganzen Array `m_labels` ein einziger EventHandler zugewiesen. In der Methode `MainForm::m_labels_DoubleClick(System::Object^ sender, System::EventArgs^ e)` wird auf den Event reagiert. Mittels dem Parameter `sender` wird ermittelt, in welches Label geklickt wurde. Falls die Farbe geändert wurde, wird

diese an der entsprechenden Stelle im `m_colors` Array geändert und die Bitmap neu gezeichnet anhand des geänderten `m_colors` Arrays.

8 Kombination der Datenquellen

8.1 Kombination der Datenquellen in der Overview-Bitmap

Wenn verschiedene Datenquellen vorhanden sind, stellt sich die Frage, wie diese im OverviewModus dargestellt werden sollen.

8.1.1 Übersicht über die Datenquellen für die Overview-Bitmap

	Overview	Terrain/ Detail
Texture	PGE-Preview	PGE-Tiles
DEM: - Reliefschattierung - Hypsometrische Darstellung - Hypsometrische Darstellung, transparent über Reliefschattierung	PGE-Preview	Pro Tile ein VertexArray: pro Vertex: - Höhenwert - Normalen (für Schattierung) - oder: Graustufenwert
LandUsage	PGE-Preview	

Die Erstellung der PGE-Preview ist vorimplementiert in PGFExt.

8.1.2 Konzept Darstellung in der OverviewBitmap

Es wurden diverse Prioritäten für die Darstellung der Daten definiert (1. = höchste Priorität):

1. Texturdaten
2. Höhendaten (DEM)
3. Landnutzungsdaten

Gemäss diesen Prioritäten sollen in der Overview-Bitmap die Datenquellen eingefüllt werden. Überdecken sich die (geografischen) Bereiche der einzelnen Datenquellen, so sollen sie in der Overview-Bitmap gemäss den definierten Prioritäten kombiniert werden.
z.B.

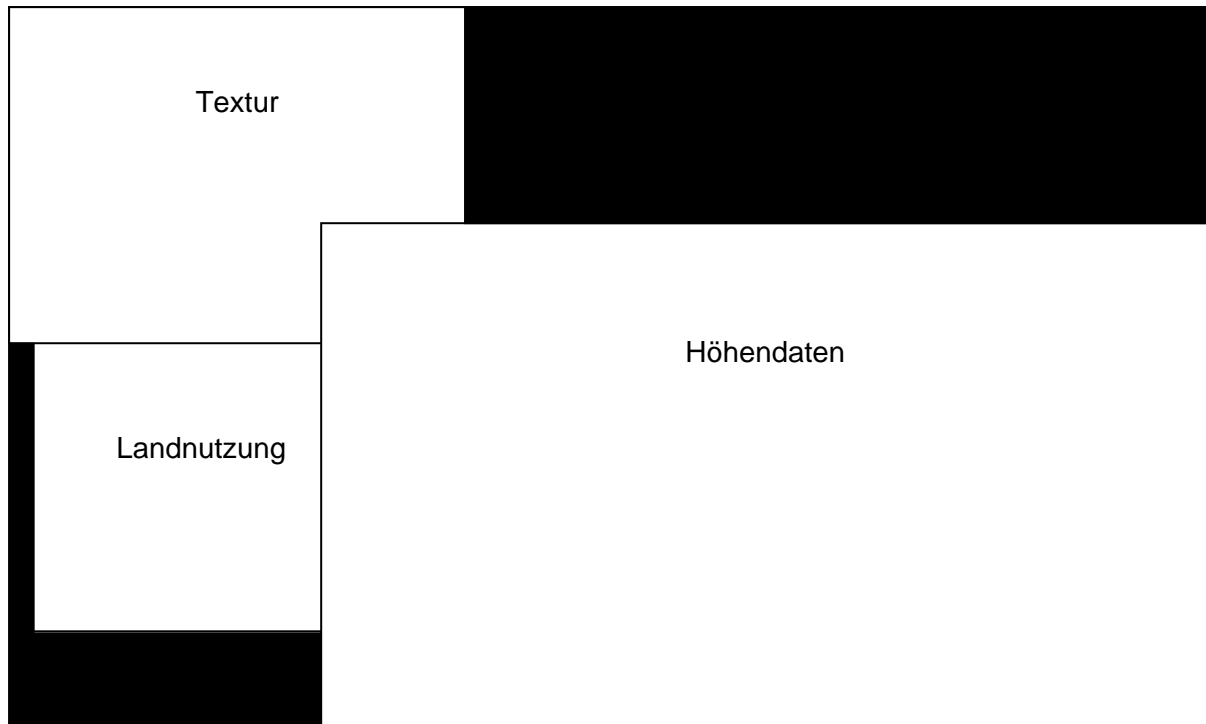


Abb. 21: Kombination der Datenquellen für die Overview-Bitmap

Dabei sollen Bereiche der Overview-Bitmap, welche übrigbleiben und nicht durch eine Datenquelle abgedeckt werden, schwarz eingefärbt werden.

8.1.3 Momentane Darstellung im OverviewModus

In der Methode `CTerrain::CreateOverviewBitmap()` wird abgefragt, welche Daten visualisiert werden, und dementsprechend die `OverviewBitmap` gefüllt.

Es werden somit in der `OverviewBitmap` entweder die PGE-Preview der Texturdaten oder die PGE-Preview der Höhendaten dargestellt.

Der übernommene Code in der Funktion wurde angepasst:

Bisher wurde das Pixelformat für die Bitmap aus dem `PreviewMode` ausgelesen und übernommen:

```
Imaging::PixelFormat format = ImageModeToPixelFormat(preview->Mode());
```

neu wird das Format in jedem Fall auf 24BppRGB gesetzt:

```
Imaging::PixelFormat format = Imaging::PixelFormat::Format24bppRgb;
```

8.1.4 Ausblick Darstellung im Overview Modus

Die Overview-Bitmap muss noch nach dem im 8.1.2 vorgeschlagenen Konzept erstellt werden.

8.2 Visualisierte Daten

Bisher wurde mit drei Attributen in der `Ra2dioApp`-Klasse festgehalten, welche Daten visualisiert sind im `TerrainViewer`:

```
bool          m_showDEM;
bool          m_showTexture;
```

```
bool          m_showLandUsage;
```

Dies wurde geändert und übersichtlicher gestaltet, indem nun eine einzige Variable anzeigt, welche Daten visualisiert werden:

```
DisplayModeT m_dispMode;
```

Wobei:

```
enum DisplayModeT { DM_HeightColor, DM_Gouraud, DM_Texture,  
DM_TextureBlended };
```

(Gemäss Constants.h.)

9 Automatisches Testen

9.1 Auszug aus der Aufgabenstellung

In der Schlussphase Ihrer Projektarbeit haben Sie erlebt, wie wichtig es wäre zu wissen, dass die entwickelten Module/Klassen genau das tun, was von ihnen erwartet wird. Um dieses Mal nicht wieder in die gleichen Probleme hinein zu laufen, sollten Sie zumindest für einen Teil der neu zu entwickelnden Klassen automatisches Testen mittels NUnit (www.nunit.org) einführen.

9.2 Testen während der Entwicklung

Aufgrund der Tatsache, dass die meisten Teilaufgaben direkt in der GUI dargestellt wurden, war hier kein automatisiertes Testen notwendig. Um zu debuggen wurde mit dem Debugger von Visual Studio und mit Konsolenausgaben mittels Trace gearbeitet.

9.3 Probleme beim Verwenden von NUnit

Ich habe NUnit installiert, konnte allerdings das RA₂DIO-Projekt nicht einbinden. In den bestehenden Projekteinstellungen, welche ich von der überarbeiteten PA-Version übernommen habe, wird das Output-File in einen anderen Ordner, als dies in VisualStudio standardmässig der Fall ist, geschrieben. Somit wird es nicht im Unterordner „debug“ auf der gleichen Ebene wie das Projektfile (.vcproj), sondern in einen „debug“-Ordner eine Ebene höher (auf der Eben des Solution Files).

Bei NUnit muss jedoch das NUnit-Projektfile (.nunit) auf der Ebene des VS-Projektfiles (.vcproj) gespeichert werden. Mittels „Add Assembly“ kann dann das .exe-File aus dem debug-Ordner geladen werden. Da dies aber gemäss oben erwähnten Einstellungen in RA₂DIO nicht der standardmässige Debug-Ordner ist, kann das .exe-File nicht eingebunden werden, und es erscheint folgende Fehlermeldung:

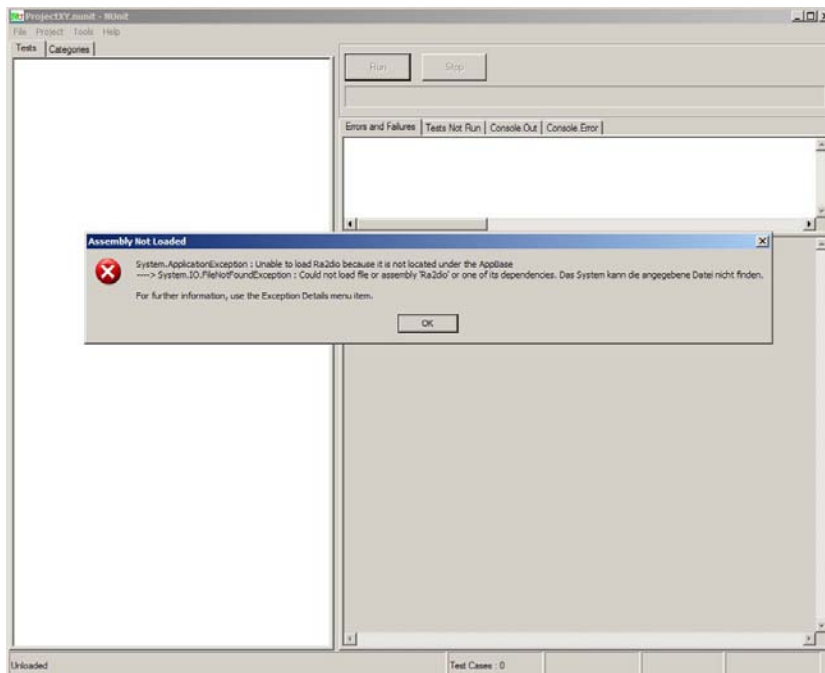


Abb. 22: Screenshot Fehler beim Einbinden des RA₂DIO-Projektes in NUnit (Text siehe Abb. 23)



Abb. 23: Screenshot Fehlermeldung beim Einbinden des RA₂DIO-Projektes in NUnit

Der Versuch, NUnit in einem separaten, eigens dafür erstellten WindowsForms-Testprojekt (siehe Ordner „NUnitTest“) zu verwenden, funktionierte. In diesem Projekt wird das .exe-File in den standard „debug“-Ordner abgelegt, welcher sich auf der selben Ebene wie das .vcproj-File befindet.

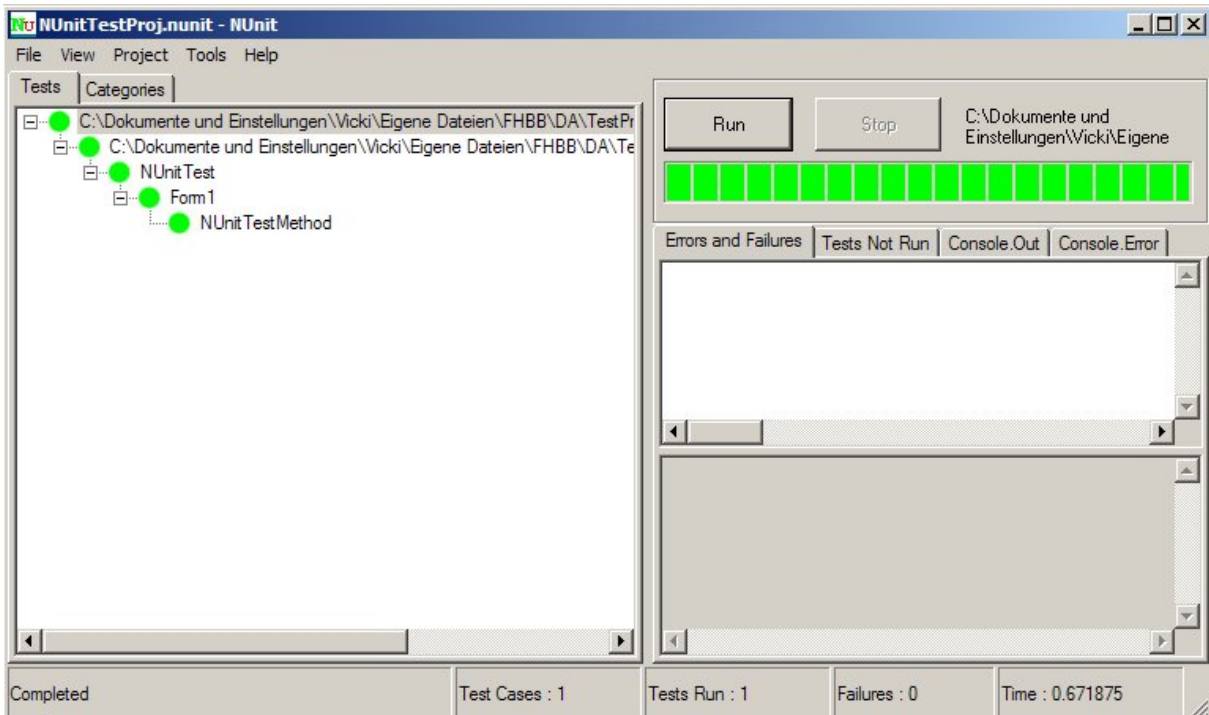


Abb. 24: Screenshot, NUnit-Test mit separatem WindowsForms-Projekt

9.4 Ausblick Testen mit NUnit

Wenn NUnit in der weiteren Entwicklung zum Testen verwendet werden soll, so sind die Projekteinstellungen zu überarbeiten und gemäss VS-Standard einzurichten. Ansonsten kann NUnit die notwendigen Files nicht finden.

10 Online- Hilfe

10.1 Auszug aus der Aufgabenstellung

Eine professionelle Software ohne Online-Hilfe ist nur wenig wert. Die Entwicklung eines integrierten Hilfesystems wird von VS 2005 gut unterstützt. Studieren Sie in MSDN die vorhandenen Möglichkeiten und die notwendigen Entwicklungsschritte.

Ziele

Konzipieren Sie ein Online-Hilfesystem und realisieren ein paar exemplarische Einträge in die Hilfe. Hier geht es darum, dass alle Teilaspekte für die Realisierung des Hilfesystems abgedeckt sind, ohne dass die ganze Fleissarbeit der Textaufbereitung vorgenommen werden muss.

10.2 Umsetzung

Aufgrund der Unklarheit, ob managed Code bzw. .NET in der weiteren Entwicklung von RA₂DIO noch verwendet werden soll, wurde in Absprache mit dem Betreuer diese Teilaufgabe gestrichen.

11 Ausblick und weitere Entwicklung

11.1 Technologie

Während der Bearbeitung der verschiedenen Aufgabenstellungen stiess ich immer wieder auf Probleme mit mixed Code:

- Serialisierung
- Visualisierung der Höhendaten mit DirectX
- Verwendung vordefinierter Access-Abfragen

Spätestens als klar wurde, dass Managed DirectX nicht mehr länger unterstützt wird, stellten Herr Stamm und ich fest, dass es fraglich ist, wieviel Sinn die Mischung von managed und unmanaged Code in RA₂DIO macht.

Für die weitere Entwicklung von RA₂DIO empfiehlt es sich deshalb meiner Meinung nach, die Verwendung von managed und unmanaged Code nochmals genau zu hinterfragen. Ich bekam in den 10 Wochen der Diplomarbeit immer mehr den Eindruck, dass sich die .NET Technologie für diese Applikation nicht eignet.

Zuerst sah es so aus, als wäre es sinnvoll, die alten, von Ra3dio übernommenen unmanaged Klassen in managed Klassen umzuschreiben, um die Probleme mit der Serialisierung zu lösen. Doch bleibt damit die (im Vergleich zu SQL Server) relativ schlechte Unterstützung der Access-Datenbanken mit ADO.NET bestehen. Spätestens seit ich entdeckte, dass Managed DirectX nicht mehr unterstützt wird, stelle ich in Frage, ob es für RA₂DIO sinnvoll ist, die .NET Technologie einzusetzen, oder ob man nicht besser wieder zurück zum unmanaged Code bzw. MFC geht. Entscheidet man sich für .NET ist allenfalls zu prüfen, ob C# als Programmiersprache sinnvoll wäre, da es mir scheint, dass .NET vor allem die Programmierung mit C# unterstützt..

Es wäre sich allenfalls ein Szenario zu überlegen, in welchem für die GUI WindowsForms von .NET verwendet wird und für die restlichen Programmteile unmanaged Code. Dabei gibt es allerdings das Problem, dass die GUI oftmals eng mit den anderen Programmteilen verknüpft ist, z.B. für die Visualisierung der Gemeindedaten. Es müsste hier also eine noch strengere Trennung der Benutzeroberfläche vom restlichen Programm vorgenommen werden, wobei fraglich ist, wie weit dies möglich ist.

Bevor an RA₂DIO weiterentwickelt wird, empfiehlt es sich deshalb, vorsichtig zu evaluieren, mit welchen Technologien gearbeitet werden soll. Allenfalls kann es nötig sein, bestehenden Code umzuschreiben.

11.2 Weitere anschliessende Aufgaben

Wie schon im Kapitel über Serialisierung erwähnt, muss die Serialisierung implementiert werden. Dies wird jedoch abhängig sein vom Entscheid, ob weiterhin mit mixed Code gearbeitet wird oder eine andere Lösung gesucht wird.

Aufgrund der Unsicherheit, ob mit .NET weitergearbeitet wird, wurde die Online-Hilfe gestrichen und muss entsprechend dem weiteren Vorgehen nun noch implementiert werden.

Die Overview-Bitmap in welcher alle vorhandenen Quelldateien in einer Bitmap kombiniert werden ist noch zu realisieren.

In der weiteren Entwicklung muss zudem die Visualisierung der Geländedaten komplett implementiert werden: Dies umfasst die Reliefschattierung sowie die Visualisierung der Landnutzungsdaten. Dabei ist zu beachten, dass zum Beispiel die Reliefschattierung transparent über die hypsometrische Darstellung der Höhendaten gelegt werden kann, oder dass andere Daten gleichzeitig dargestellt werden können, wobei der Benutzer wählen können soll, welche Daten (falls vorhanden) dargestellt werden und welche nicht.

Weiter müssen sämtliche Funktionalitäten im Zusammenhang mit Antennen und Wellenausbreitungsberechnungen implementiert werden, wobei teilweise auf vorhandenen Code aus Ra3dio gegriffen werden kann.

Sollte mit managed Code weitergearbeitet werden, so kann die Datenbank-Anbindung, welche für die Diplomarbeit implementiert wurde, übernommen und damit weitergearbeitet werden.

12 Persönliche Bemerkungen

Bevor ich die Diplomarbeit antrat, erhoffte ich mir, ein laufendes, in sich geschlossenes Projekt erarbeiten zu können. Jedoch war dies leider nicht der Fall.

Grundsätzlich würde ich den Zeitplan wieder gleich erstellen. Wären die Aufgaben problemlos lösbar gewesen, hätte er wahrscheinlich eingehalten werden können.

Leider verunmöglichten die in der Dokumentation genannten Probleme eine effiziente Umsetzung der diversen Aufgabenstellungen. Vieles lief nicht so problemlos ab, wie man sich dies am Anfang vorgestellt hatte, und vor allem die Probleme bei der Serialisierung des mixed Codes, als auch beim Versuch, vordefinierte Access-Abfragen zu verwenden bzw. in einer einzigen Query nachzubauen, brachten dem Projekt einen grossen zeitlichen Verlust ein. Diesen konnte ich, obwohl ich intensiv am Projekt arbeitete, nicht mehr aufholen.

Ein weiteres Problem war, dass der übernommene Code zum Teil nicht ganz fehlerfrei war (zum Beispiel bei der Funktion `Reposition()`), und die Fehlerlokalisierung wiederum oftmals Zeit kostete.

Hinzu kam zusätzlich erschwerend, dass keine Dokumentation nach der Überarbeitung der PA vorhanden war, und der neue Code teilweise nur spärlich oder gar nicht kommentiert war.

Dennoch konnte ich während der 10wöchigen Arbeit profitieren. Ich lernte die .NET-Technologie kennen und setzte mich damit intensiv auseinander, insbesondere für die Datenbankbindung mit .NET und die GUI-Entwicklung mit Windows Forms.

Zudem lernte ich die Probleme der Vermischung von managed und unmanaged Code kennen, was mir zukünftig, sollte ich jemals wieder auf ein solches Projekt stossen, helfen wird, Probleme rechtzeitig zu erkennen und darauf zu reagieren.

13 Anhang

13.1 Komplette Aufgabenstellung (Original)

Fachliche Schwerpunkte

.NET-Entwicklung (VS 2005) eines Gelände-Viewers in C++
Graphical User Interface (Windows Forms)
Effiziente, externe Datenspeicherung von Geländedaten (PGF für Externspeicher)
Visualisierung von geografischen Daten

Kurzbeschreibung

In geographischen Informationssystemen (GIS) werden sehr grosse Mengen an Geländedaten verarbeitet. Zu den Geländedaten zählen Topographiedaten (Höhendaten), Texturdaten (Luftbilder, Pixelkarten usw.), Landnutzungsdaten und weitere thematische Daten, welche mit geografischen Koordinaten verbunden sind. Diese Geländedaten sollten auf Externspeichern (z.B. Festplatte) verwaltet und nur stückweise bei Bedarf in den Hauptspeicher geladen werden. Dadurch können mit normalgrossem Hauptspeicher riesige Geländeabschnitte prozessiert werden.

RA₂DIO soll in ferner Zukunft wie sein grosser Bruder RA₃DIO die Funkwellenausbreitung in riesigen Geländen simulieren. Nun aber ohne echte 3D-Darstellung, sondern mit normaler, parallelperspektivischer 2D-Ansicht. Dafür aber mit einer ausgeklügelten und effizienten Verwaltung der Geländedaten. Diese Verwaltung soll auf der Technik "PGF für Externspeicher" basieren. Dabei werden Bilddaten in Kacheln aufgeteilt und einzeln mit PGF komprimiert abgelegt. Höhen- und Landnutzungsdaten können als weitere "Bildkanäle" der Texturdaten interpretiert und gemeinsam mit den Bilddaten auf dem Externspeicher abgelegt werden.

Informationen zu PGF und RA₃DIO finden Sie unter www.xeraina.ch/pgf beziehungsweise www.xeraina.ch/ra3dio.

Diese Diplomarbeit ist eine Fortsetzung der gleichnamigen Projektarbeit.

Verantwortlicher Dozent und Betreuer

Prof. Dr. Christoph Stamm, FHNW, christoph.stamm@fhnw.ch

Expertin

Frau Dr. Gabriele Neyer, sma Zürich.

Studentin

Victoria Schmid, Chrebsbachstrasse 56, 8162 Steinmaur,
Telefon P: 044 853 36 44, M: 061 711 36 88, vicki.schmid@bluewin.ch

Projektdauer

23. Oktober 2006 bis 10. Januar 2007

13.1.1 Aufgabenstellung

13.1.2 Einarbeitung

13.1.2.1 Vorbereitung

Fixieren Sie einen wöchentlichen Besprechungstermin mit Ihrem Betreuer. Nutzen Sie diese wöchentlichen Besprechungen, um Ideen, Vorschläge und Arbeiten zu präsentieren und zu diskutieren.

Bevor Sie mit der Arbeit richtig loslegen, erstellen Sie einen möglichst detaillierten Zeitplan für die Dauer dieser Diplomarbeit. Definieren Sie die wichtigsten Meilensteine.

13.1.2.2 Infrastruktur

Richten Sie eine Arbeitsstation zur Software-Entwicklung mit Visual Studio 2005 und Quellcode-Management (Subversion) ein. Achten Sie darauf, dass Sie lokalen Zugriff auf MSDN haben, denn MSDN wird Ihr wichtigster Begleiter während dieser Diplomarbeit sein.

13.1.2.3 Sprach- und Softwarestudium

Die Software-Entwicklung für die .NET-Plattform machte Spracherweiterungen in C++ notwendig. Verschiedene dieser Spracherweiterungen sind im bestehenden Quellcode von RA₂DIO bereits eingesetzt worden. Studieren Sie detailliert den vorhandenen Quellcode und die neuen Spracherweiterungen in C++.

13.1.2.4 Ziele

Der bestehende RA₂DIO-Quellcode muss bis ins letzte Detail verstanden sein.

13.1.3 RA₂DIO Weiterentwicklung

In den folgenden Abschnitten sind die geforderten Weiterentwicklungen von RA₂DIO spezifiziert. Die zu erreichenden Ziele sind einzeln aufgeführt.

13.1.3.1 Systemarchitektur

Die aktuelle Version von RA₂DIO basiert auf einer klaren Trennung zwischen grafischer Benutzungsoberfläche (GUI) und interner Datenhaltung/-verarbeitung einerseits und einer Trennung zwischen interner und externer Datenhaltung. Diese dreischichtige Systemarchitektur, welche auf dem Konzept des Model View Control basiert, soll weiter gepflegt und wenn möglich noch deutlicher sichtbar gemacht werden. Eine klare dreistufige Architektur, bei der direkte Zugriffe vom GUI (View-Layer) auf die externen Daten (Model-Layer) verboten sind, ist absolut zentral.

13.1.3.2 Persistente Datenhaltung

In RA₂DIO müssen unterschiedliche Daten persistent gespeichert werden. Zu diesen Daten gehören unter anderem: Benutzereinstellungen, aktueller Geländezustand, aktuelle Wellenausbreitungsdaten, Basisstationen. Während die Basisstationen in einer Datenbank gehalten werden, sollen der aktuelle Geländezustand und die aktuellen Wellenausbreitungen in einem speziellen Dokument (mit der Endung .RA2) abgelegt werden. Somit kann die Arbeit mit RA₂DIO jederzeit unterbrochen, der aktuelle Stand in einem Dokument gespeichert und die Arbeit zu einem späteren Zeitpunkt durch Öffnen des Dokumentes wieder aufgenommen werden.

Im bestehenden Programm RA₃DIO werden Punktobjekte wie Basisstationen, Messpunkte und Ortschaften in einer relationalen Datenbank gehalten. An diesem Datenbankkonzept soll in RA₂DIO festgehalten werden, weil es sich bewährt hat und die Kunden über umfangreiche Datenbanken verfügen. Dabei soll auf eine Anpassung der Datenbank, wenn möglich, verzichtet werden.

13.1.3.3 Ziele

Erstellen Sie einfaches und gut umsetzbares Persistenz-Konzept für alle Daten, die in RA₂DIO persistent gehalten werden müssen. Setzen Sie dieses Konzept um.

In der aktuellen Version von RA₂DIO ist Tiny-XML zur Verarbeitung von XML-Dateien eingesetzt worden. Ersetzen Sie Tiny-XML durch die vorhandenen Konzepte und Technologien in .NET.

Studieren Sie die Anbindung von Microsoft Access-Datenbanken in .NET. Der bestehende Quellcode kann hier nur sporadisch übernommen werden, da die vorhandene Datenbankanbindung auf MFC basiert.

Implementieren Sie den Zugriff auf die Tabellen mit den Ortschaftsdaten und visualisieren Sie alle sichtbaren Ortschaften im Gelände-Viewer mittels eines geeigneten Symbols oder Textes.

13.1.4 Geländevisualisierung

Die aktuelle Version von RA₂DIO erlaubt lediglich die Visualisierung von Texturen. Da neben Textur- auch Gelände- und Landnutzungsdaten vorhanden sind, müssen diese auch visualisiert werden können. Grundsätzlich sollen in RA₂DIO folgende Darstellungsmodi vorhanden sein:

- Geländedarstellung mit hypsometrischen Farbtönen
- Geländedarstellung mit schattiertem Relief
- Landnutzungsdaten (separat ein- und ausschaltbar)
- Texturdaten (mit und ohne Transparenz und separat ein- und ausschaltbar)

Diese Darstellungsmodi sind alle auch in RA₃DIO vorhanden und können dort studiert werden.

Für hypsometrische Farbtöne und Falschfarbendarstellung bei der Landnutzung sind Erweiterungen der Benutzungsoberfläche notwendig, um die gewünschten Farben und Höhenbereiche individuell festzulegen. Die entsprechenden Einstellungen sind oft Abhängig von den zu visualisierenden Geländedaten.

Zur Geländedarstellung mittels schattierten Reliefs soll ein bekannter, einfacher und effizienter Schattierungsalgorithmus (z.B. Gouraud Shading) eingesetzt werden. Gouraud-Shading braucht neben den Höhendaten auch Normalen. Hierbei gilt es zu überlegen, ob diese Geländenormalen immer wieder neu berechnet oder in einem Preprocessing-Schritt berechnet und dann persistent und effizient gespeichert werden.

13.1.4.1 Ziele

Organisieren Sie Gelände- und Landnutzungsdaten.

Studieren Sie geeignete Shading-Algorithmen und setzen Sie einen nach Absprache mit Ihrem Betreuer um.

Realisieren Sie die geforderten Geländedarstellungsmodi und passen Sie dazu das GUI entsprechend an.

13.1.5 GUI Erweiterungen

Die aktuelle grafische Benutzungsoberfläche (GUI) in RA₂DIO ist noch sehr rudimentär. Die wichtigsten Grundstrukturen sind jedoch vorhanden: Menubar, Toolbar, Statusbar, Terrain-Viewer, Colorange-Viewer, Tree-Viewer, Tab-Viewer.

Wenig häufige Interaktionsschritte werden über das Menü vorgenommen. Das Menü verfügt über Tastaturkürzel für die wichtigsten Aufgaben, wie z.B. das Öffnen und Speichern von RA₂-Dokumenten.

Im Toolbar werden wiederum die wichtigsten Menüpunkte mittels aussagekräftigen Symbolen angeboten. Zudem soll der Toolbar die wichtigsten Auswahl- und Eingabeboxen enthalten.

Der Statusbar gibt ständig Auskunft über den aktuellen Status der Software. So werden zum Beispiel die Koordinaten des Cursors im Gelände-Viewer dargestellt. Der Statusbar ist besonders dann wichtig, wenn Aufgaben im Hintergrund erledigt werden, welche länger dauern und an die Geduld des Benutzers appellieren. Der Verlauf dieser Hintergrundarbeiten wird in einem Progressbar innerhalb des Statusbar dargestellt.

Der Colorange-Viewer befindet sich immer am rechten Rand des Terrain-Viewers. Im Colorange-Viewer sollen die Falschfarbskalen für die Landnutzung, die Hypsometrie und die Wellenausbreitung dargestellt werden.

Im Tree-Viewer sollen Datenbank-Datensätze hierarchisch dargestellt werden. Hier soll die Benutzerin alle Ortschaftsdaten, alle Basisstationen, alle Antennen und weitere Daten direkt anwählen können. Die Eigenschaften des ausgewählten Objektes werden im Property-Grid-Viewer dargestellt.

Im Property-Grid-Viewer sollen die Eigenschaften eines selektierten Objektes (z.B. Basisstation) in Textform dargestellt werden. Hier sollen sich auch die Eigenschaften eines selektierten Objektes verändern lassen. Dies entspricht der heute üblichen Darstellung eines Properties-Grids. Informationen zu Property-Grids finden Sie auch unter (www.codeproject.com).

13.1.5.1 Ziele

Füllen Sie den Tree-Viewer mit den Daten aus der Ortschaftsdatenbank auf und realisieren Sie eine bidirektionale Verbindung zwischen Tree-Viewer und Terrain-Viewer. Ortschaften, welche momentan im Terrain-Viewer sichtbar sind, sollen im Tree-Viewer farblich markiert werden. Selektiert die Benutzerin eine Ortschaft im Tree-Viewer, so soll sie die Möglichkeit erhalten, den sichtbaren Geländeausschnitt so zu verändern, dass die gewählte Ortschaft im Terrain-Viewer sichtbar wird. Umgekehrt, selektiert die Benutzerin eine Ortschaft im Terrain-Viewer, so soll die gleiche Ortschaft im Tree-Viewer auch selektiert werden.

Im Property-Grid-Viewer sollen immer die Eigenschaften des aktuell selektierten Objektes dargestellt werden. Beachten Sie dabei, dass viele Objekte sowohl im Terrain-Viewer als auch im Tree-Viewer selektiert werden können. Werden die Eigenschaften eines selektierten Objektes verändert, so müssen die neuen Daten auf Wunsch in die Datenbank zurück geschrieben werden können.

Implementieren Sie den Colorange-Viewer zumindest für die hypsometrischen Farbtöne. Die angezeigten Farben und Höhenbereiche müssen individuell gewählt und gespeichert werden können.

Die Verschiebung des aktuell sichtbaren Geländeausschnittes im Terrain-Viewer ist momentan sehr rudimentär. Erstellen Sie hier lediglich ein Interaktionskonzept, wie eine sinnvolle Benutzerführung aussehen könnte, um den aktuellen Ausschnitt sowohl in kleinen Schritten als auch in grossen Schritten zu verändern.

13.1.6 Automatisches Testen

In der Schlussphase Ihrer Projektarbeit haben Sie erlebt, wie wichtig es wäre zu wissen, dass die entwickelten Module/Klassen genau das tun, was von ihnen erwartet wird. Um dieses Mal nicht wieder in die gleichen Probleme hinein zu laufen, sollten Sie zumindest für einen Teil der neu zu entwickelnden Klassen automatisches Testen mittels NUnit (www.nunit.org) einführen.

13.1.7 Online-Hilfe

Eine professionelle Software ohne Online-Hilfe ist nur wenig wert. Die Entwicklung eines integrierten Hilfesystems wird von VS 2005 gut unterstützt. Studieren Sie in MSDN die vorhandenen Möglichkeiten und die notwendigen Entwicklungsschritte.

13.1.7.1 Ziele

Konzipieren Sie ein Online-Hilfesystem und realisieren ein paar exemplarische Einträge in die Hilfe. Hier geht es darum, dass alle Teilaspekte für die Realisierung des Hilfesystems abgedeckt sind, ohne dass die ganze Fleissarbeit der Textaufbereitung vorgenommen werden muss.

13.1.8 Dokumentation

13.1.8.1 Schriftliche Dokumentation

Dokumentieren Sie schriftlich und elektronisch Ihre Vorgehensweise, Ihre Konzepte, die wichtigsten Punkte der Implementierungen und Ihre Testresultate. Überprüfen Sie auch den geplanten mit dem tatsächlichen Zeitplan und reflektieren Sie Ihre persönlichen Erlebnisse und Erfahrungen während dieser Diplomarbeit.

Beachten Sie, dass eine gute Dokumentation auch noch nach vielen Jahren gelesen werden können muss und dass sie dem Leser ein gut abgerundetes Bild vermittelt, auch dann wenn er nicht direkt an der Arbeit beteiligt war. Bitte legen Sie auch grossen Wert auf sprachliche Qualität.

13.1.8.2 Präsentation

In Absprache mit Ihrem Betreuer und dem Auftragsteller der Arbeit wird von Ihnen eine Präsentation Ihrer Arbeit erwartet. Die Präsentation sollte ca. 20 Minuten dauern und einerseits einen groben Überblick verschaffen und andererseits ein oder zwei wichtige und interessante Details hervorheben. Bei den Zuhörern dürfen Sie von einem technisch versierten Fachpublikum ausgehen.

Zusätzlich zur Präsentation wird eine prägnante Demonstration der Benutzung Ihrer Software erwartet. Dazu stehen Ihnen weitere 20 Minuten zur Verfügung.

13.1.9 Rechtliche Hinweise

13.1.9.1 Vereinbarung über Urheberrechte an Computerprogrammen

13.1.9.1.1 Gegenstand dieser Vereinbarung

Regelung der Urheberrechte an Computerprogrammen, die im Rahmen der Ausbildung aufgrund einer Aufgabenstellung seitens der FHNW oder eines externen Aufgabenstellers durch Studierende ohne Bezahlung geschrieben werden, auf der Basis des Urheberrechtsgesetzes vom 9. Oktober 1992.

13.1.9.1.2 Verwendungs- und Änderungsbefugnisse

Neu und von den Studierenden erstellte Computerprogramme aus der oben genannten Arbeit dürfen nach deren Abschluss sowohl seitens der Aufgabensteller (FHNW und xeraina GmbH) wie auch von den beteiligten und persönlich genannten Studierenden beliebig und vergütungsfrei verwendet werden; sie dürfen auch verändert werden.

13.1.9.1.3 Autorenhinweise

Werden grössere Programmmodule dieser Diplomarbeit weitgehend unverändert verwendet, so soll auf die Studierenden sowie auf die Aufgabensteller hingewiesen werden.

13.1.9.1.4 Urheberrechte an Texten

Die Urheberrechte an den zugehörigen und von den Studierenden verfassten Berichten und Dokumentationen stehen den Studierenden zu; die Aufgabensteller können diese für eigene Zwecke und in Bibliotheken vergütungsfrei verwenden.

13.1.9.2 Vereinbarung zur Benutzung von Quellcode

13.1.9.2.1 Gegenstand dieser Vereinbarung

Regelung der Benützung von urheberrechtlich geschützten Programmbibliotheken, Computerprogrammen und Testdaten der Firma xeraina GmbH, die im Rahmen der Ausbildung aufgrund einer Aufgabenstellung seitens der FHNW oder der Firma xeraina GmbH durch Studierende verwendet werden.

13.1.9.2.2 Verwendungs- und Änderungsbefugnisse

Die von der Firma xeraina GmbH zur Verfügung gestellten Programmbibliotheken, Computerprogramme und Testdaten dürfen nur während der festgelegten Dauer dieser Diplomarbeit und ausschliesslich für den in dieser Aufgabenstellung spezifizierten Zweck verwendet werden.

Änderungen an den Programmbibliotheken und Computerprogrammen sind nur nach Rücksprache mit der zuständigen Kontaktperson der Firma xeraina GmbH zulässig.

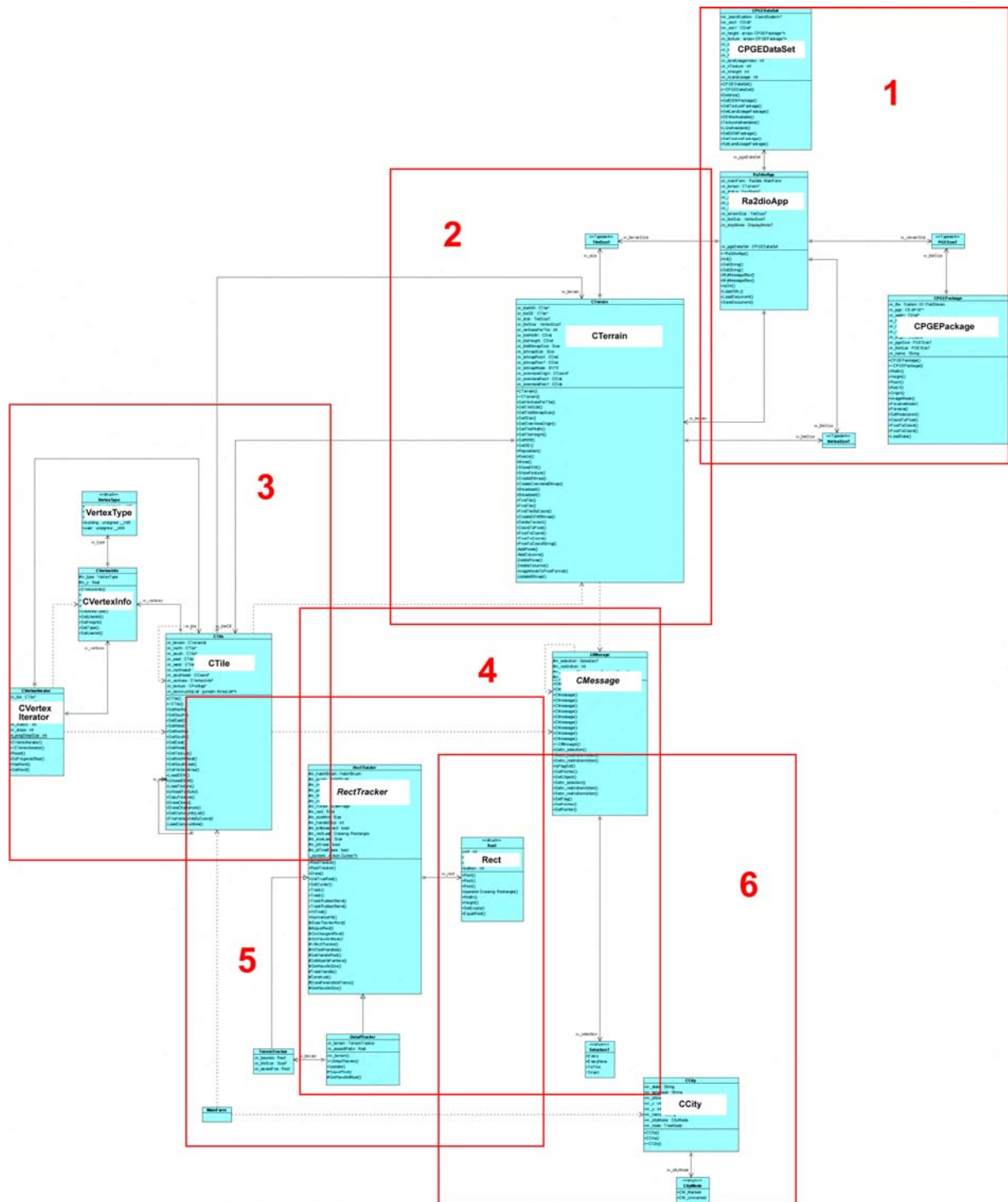
13.1.9.2.3 Reproduktion und Weitergabe

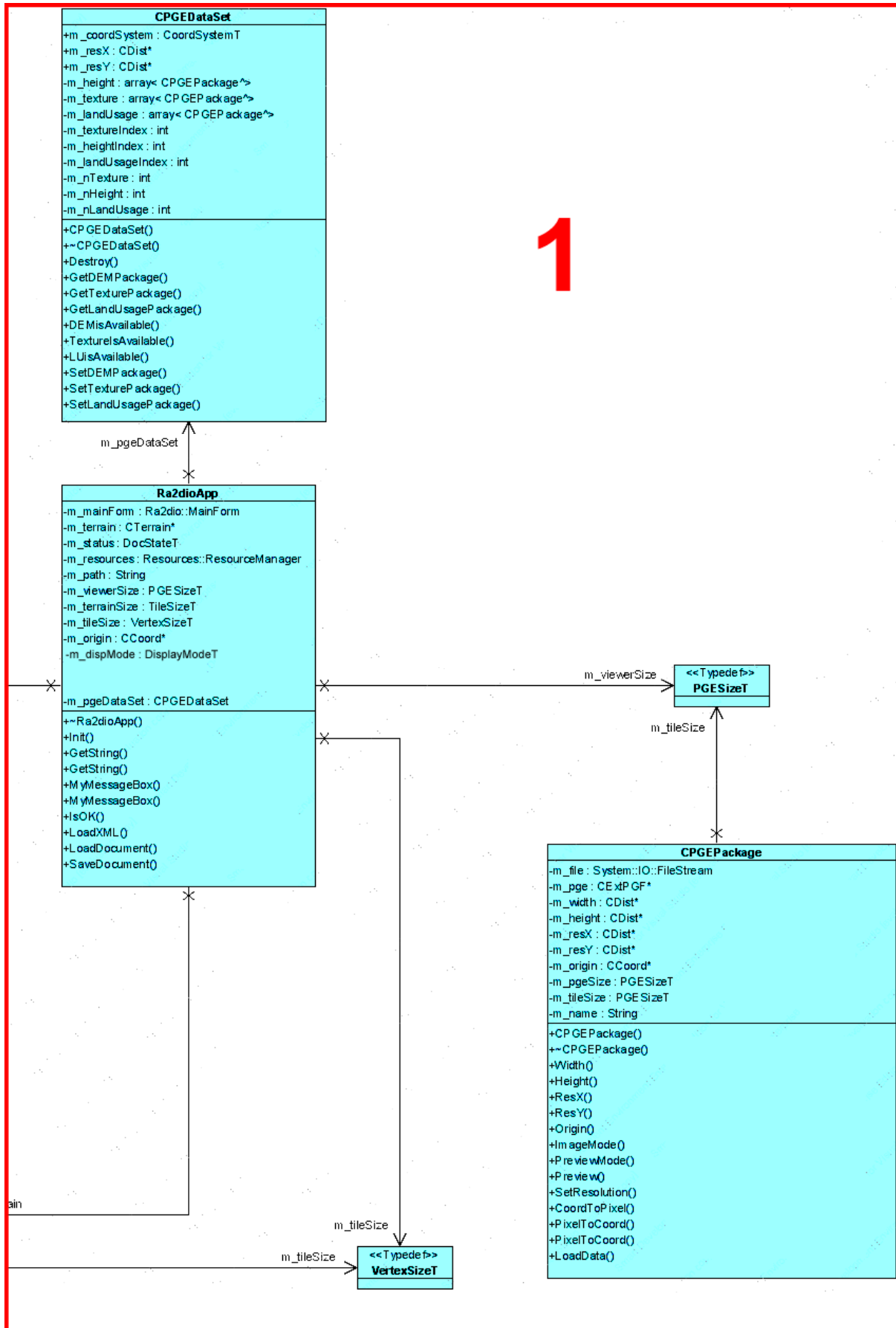
Es ist den Studierenden und auch der FHNW untersagt die Programmbibliotheken, Computerprogramme und Testdaten ohne weitergehende Vereinbarung mit der Firma xeraina GmbH in Teilen wie als Ganzes zu reproduzieren, weiterzugeben und zu vertreiben. Diese Regelung erstreckt sich auch auf neu erstellte Computerprogramme, die Funktionen aus den zur Verfügung gestellten Programmbibliotheken aufrufen.

13.1.9.3 Schlussbestimmung

Die Unterzeichneten anerkennen, den Text gelesen und verstanden zu haben und verpflichten sich mit Ihrer Unterschrift die aufgeführten Punkte und die allgemeine Sorgfaltspflicht einzuhalten.

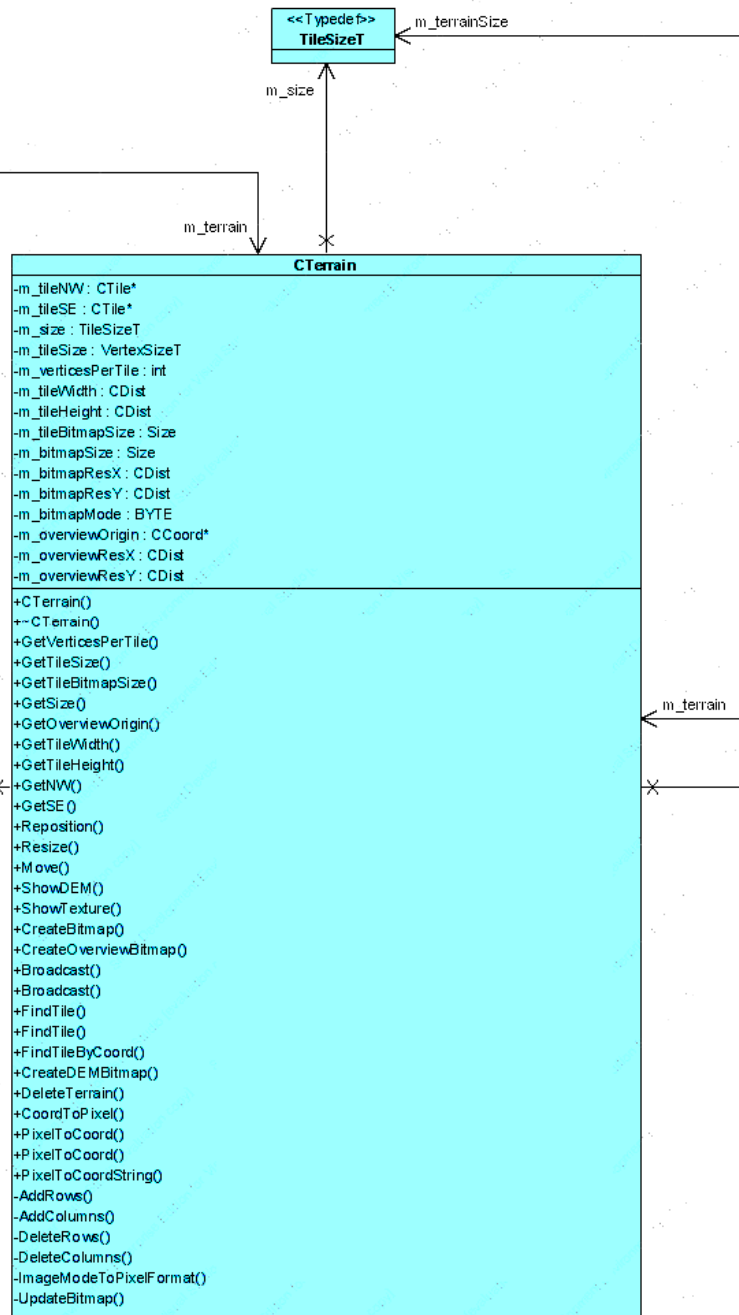
13.2 Klassendiagramm aktueller Stand

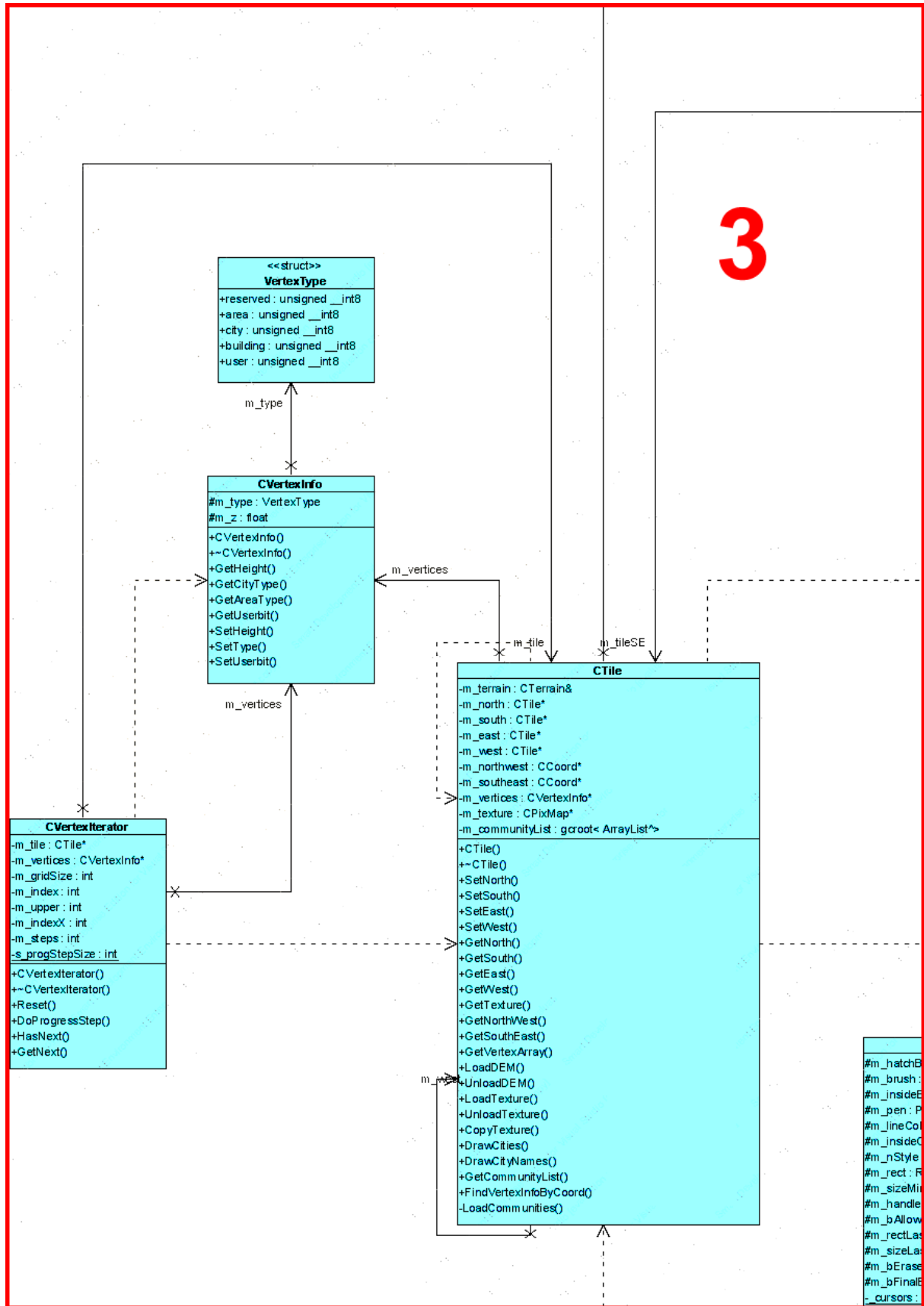




1

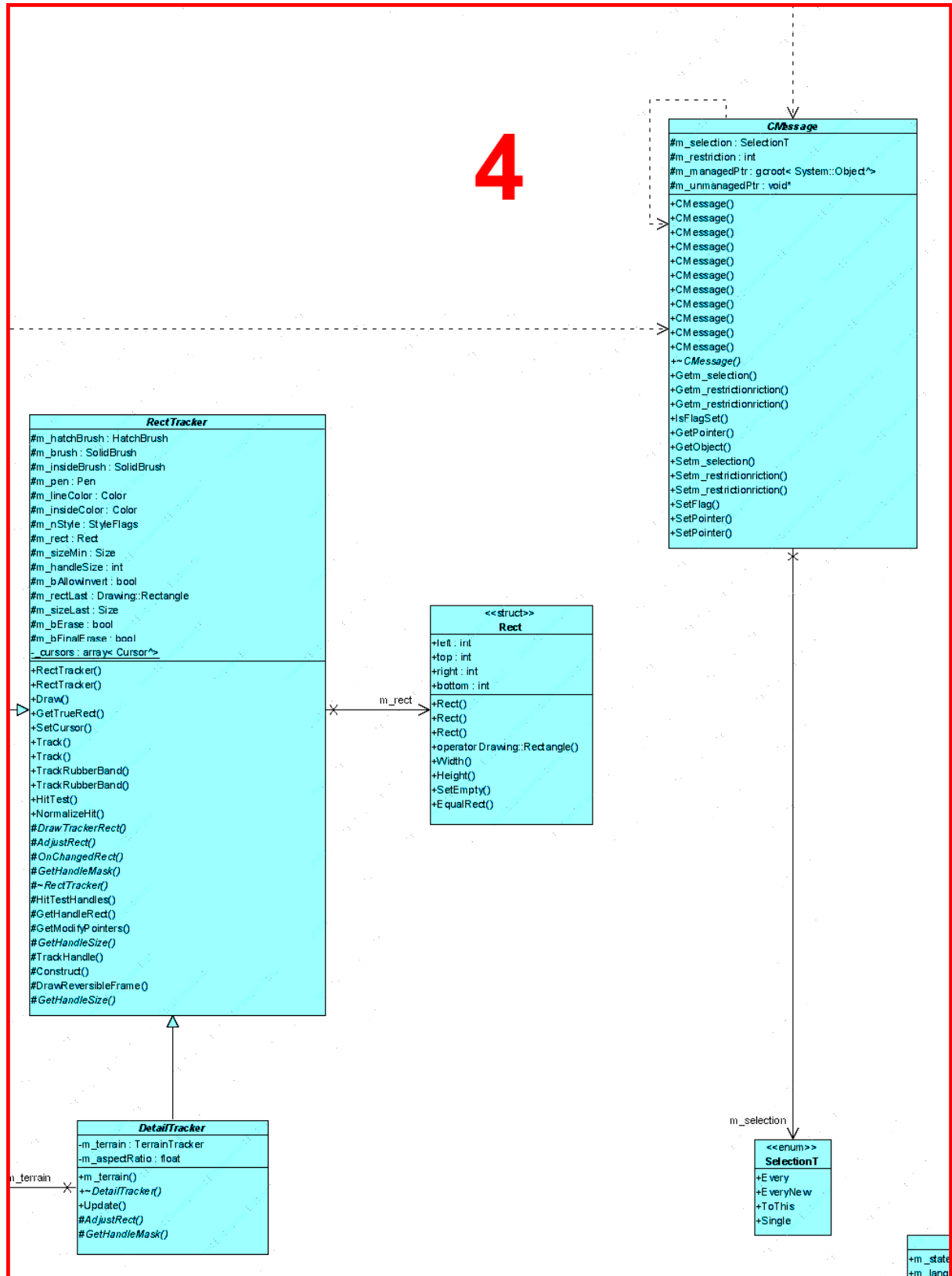
2

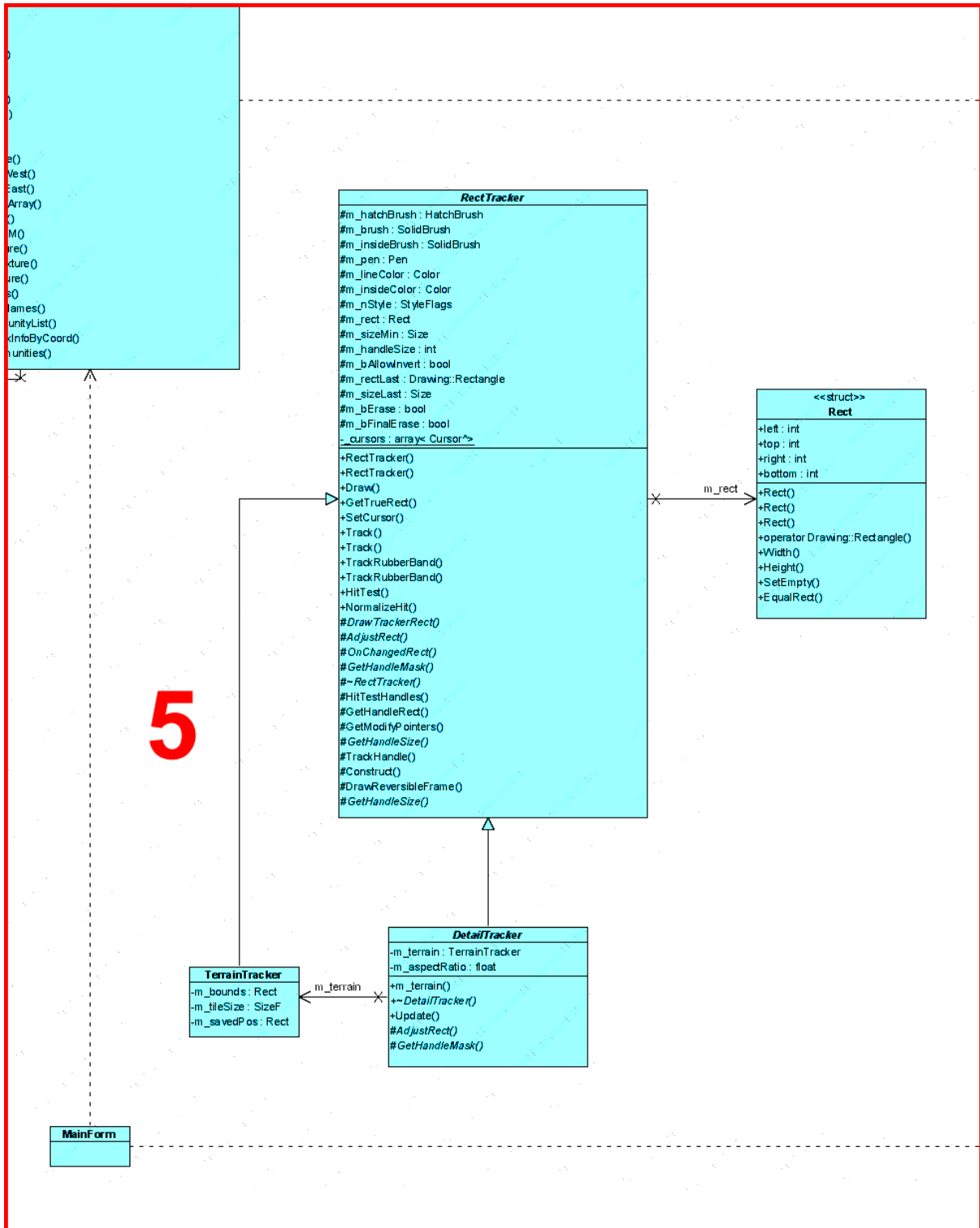


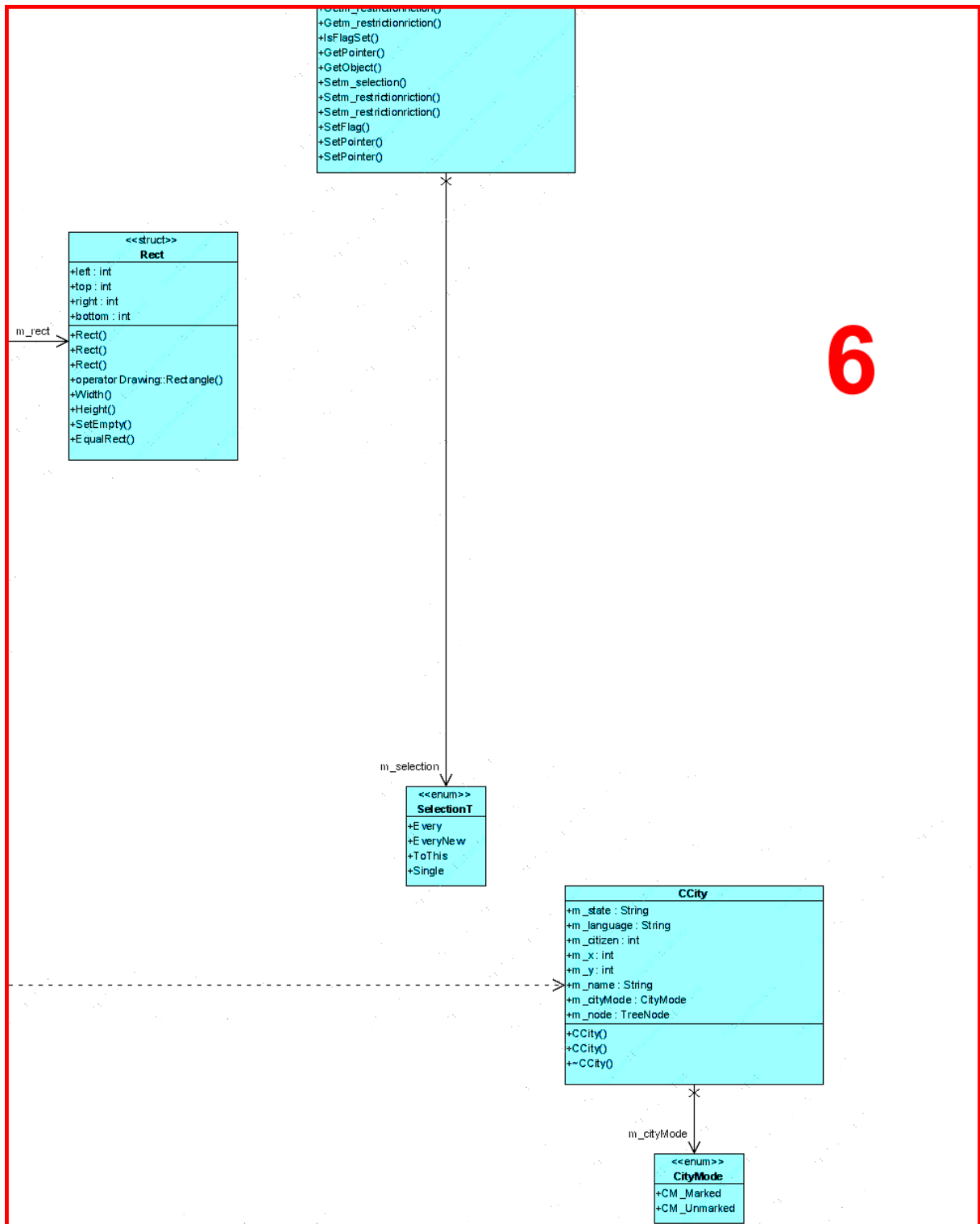


3

4







6

13.3 Quellenangaben, Literaturverzeichnis

- [1] Dr. Christoph Stamm: „Algorithms and Software for Radio Signal Coverage Prediction in Terrains“, 2001, DISS. ETH No.14283
- [2] Dr. Christoph Stamm: “PGF A new progressive file format for lossy and lossless image compression“, 2002
- [3] Basil Achermann: „PGF für Externspeicher“, 2003
- [4] Bundesamt für Landestopographie
<http://www.swisstopo.ch>
- [5] TinyXml: <http://www.grinninglizard.com/tinyxml>
Download auf: <http://sourceforge.net/projects/tinyxml>
- [6] [http://msdn2.microsoft.com/de-de/library/8e5545e1\(VS.80\).aspx](http://msdn2.microsoft.com/de-de/library/8e5545e1(VS.80).aspx)
- [7] [http://msdn2.microsoft.com/de-de/library/system.data.oledb.oledbcommand.parameters\(vs.80\).aspx](http://msdn2.microsoft.com/de-de/library/system.data.oledb.oledbcommand.parameters(vs.80).aspx)
- [8] PA_01_0506, Niklaus Jäggi, Vicki Schmid, 5.Juli 2006
- [9] PA_01_0506, Kap. 5.4.1 und 5.4.2
- [10] [http://msdn2.microsoft.com/de-de/library/bd04skah\(VS.80\).aspx](http://msdn2.microsoft.com/de-de/library/bd04skah(VS.80).aspx)
- [11] Skript Computergrafik II – 3D Echtzeit-Computergrafik, Marcus Hudritsch, 23.Dezember 2003, _CG5.Realtime.pdf
- [12] <http://www.microsoft.com/downloads/details.aspx?FamilyId=D625324C-59B4-4951-849E-640B508DC442&displaylang=en>
- [13] [http://msdn2.microsoft.com/de-de/library/system.io.stream_methods\(VS.80\).aspx](http://msdn2.microsoft.com/de-de/library/system.io.stream_methods(VS.80).aspx)

Ehrlichkeitserklärung

Hiermit bestätigt die unterzeichnende Autorin dieses Berichts, dass alle nicht klar gekennzeichneten Stellen von Ihr selbst erarbeitet und verfasst wurden.

Muttenz, 09. Januar 2007

Vicki Schmid