

Ingenieurfakultät Bau Geo Umwelt
Lehrstuhl für Computergestützte Modellierung und Simulation
Prof. Dr.-Ing. André Borrmann

Entwicklung eines Tools für interaktiven Achsenentwurf

Jonas Schlenger

Bachelorthesis
für den Bachelor of Science im Studiengang Umweltingenieurwesen

Autor:	Jonas Schlenger
Matrikelnummer:	[redacted]
Betreuer:	Prof. Dr.-Ing. André Borrmann
	Štefan Markič, Ivan Bratoev
Ausgabedatum:	21. Mai 2018
Abgabedatum:	15. Oktober 2018

Zusammenfassung

Durch die fortwährende infrastrukturelle Erschließung neuer Gebiete durch den Menschen ist die Planung neuer Verkehrswege, ganz gleich ob Straße, Schiene oder Ähnliches, eine allgegenwärtige Herausforderung. Dabei ist der Entwurf des sogenannten Alignments immer der erste Schritt. Das Alignment beschreibt hierbei die Linienführung in einer sehr vereinfachten Art und Weise und zwar mittels einer dreidimensionalen Raumkurve. Um diese besser entwerfen zu können wird das Alignment für gewöhnlich in einen horizontalen und einen vertikalen Anteil zerlegt.
Im Rahmen dieser Arbeit wird die Entwicklung einer Software beschrieben, welche durch die Verwendung eines interaktiven Tisches, schnelle und intuitive Erstellung eines solchen Alignments ermöglichen soll. Hierzu wird das zugrunde liegende Gelände als Input benötigt. Mit Hilfe von Touch-Gesten kann dann die Linienführung im Horizontalen und Vertikalen gestaltet und letztendlich in einer Industry Foundation Classes (IFC) Datei abgespeichert werden. Dies ermöglicht detailliertere Bearbeitung mit CAD-Programmen in späteren Planungsschritten.

Abstract

Through the continuous infrastructural development of new areas by humans, the planning of new traffic routes, whether road, rail or similar, is an ubiquitous challenge. The design of the so-called alignment is always the first step. The alignment describes the course of the line in a very simplified manner, namely by means of a threedimensional space curve. In order to design it more easily, the alignment is usually decomposed into a horizontal and a vertical portion.
As a part of this thesis the development of a new software is described, which allows designing a new alignment in fast and intuitive way, by the use of an interactive table. Therefore the terrain is needed as an input. With touch gestures the course of the line in horizontal and vertical dimension can be designed and saved in an Industry Foundation Classes (IFC) file. This allows more detailed editing with CAD programs in later planning steps.

Inhaltsverzeichnis

Abkürzungsverzeichnis

3D	Dreidimensionale
BIM	Building Information Modeling
CAD	Computer-aided Design
CDP	Collaborative Design Platform
DGM	Digitale Gelände Modell
EKA	Entwurfsklasse Autobahn
EKL	Entwurfsklasse Landstraße
HMD	head-mounted Display
IFC	Industry Foundation Classes
OpenGL	Open Graphics Library
TIN	Triangulated Irregular Network
TUM	Technische Universität München
UML	Unified Modeling Language
VR	Virtual Reality

Kapitel 1

Einleitung

1.1 Planungsprozess

1.1.1 Kreativer Entwurfsprozess

Obwohl der Computer in Ingenieur- und Architekturbüros nicht mehr wegzudenken ist und für immer mehr Aufgabenbereiche eingesetzt wird, ist der Einsatz des Rechners in frühen Entwurfsphasen von infrastrukturellen Projekten verschwindend gering. Beim Betrachten der Definition des Begriffes entwerfen als *„skizzieren, projektieren, umrißhaft festlegen“* (Pfeifer, 2003) [S. 289] wird bereits das Problem des Entwurfsprozesses deutlich. Die Tatsache, dass nur ungenaue Informationen festgelegt werden. Dabei kann keine eindeutige Lösung gefunden werden. Viel mehr gibt es stets mehrere gleichwertige Wege, die zum Ziel führen. Heutzutage ist es jedoch der Fall, dass unterstützende Software oft sehr detaillierte Eingabedaten benötigt, um aussagekräftige Ergebnisse zu liefern. Somit wird diese erst gegen Ende der Entwurfsphase herangezogen. Dann ist die Möglichkeit der Einflussnahme auf die Projektparameter, z.B. auf die Projektkosten jedoch nur noch geringer. Umso wichtiger ist es deshalb, den kreativen Entwurfsprozess bereits von Beginn an zu unterstützen.

Wenn man nun einen Blick auf analoge Entwurfswerkzeuge wie Skizzen, haptische Modelle oder schriftliche Notizen wirft, kann man erkennen, dass vor allem die visuelle Rückmeldung einen großen Vorteil hat. Sie erleichtert es, komplexe Fragestellungen zu erfassen und mögliche Lösungsmöglichkeiten besser zu erkennen. Möchte man also eine Entwurfssoftware entwickeln, um die anfangs beschriebene Lücke zu füllen, darf dieser Aspekt keinesfalls fehlen. Zudem sollte die Bedienung einer solchen Software sehr intuitiv gestaltet sein, damit die Entwurfsmöglichkeiten durch die Kreativität des Entwerfenden beschränkt sind und nicht durch die Software selbst.

Insgesamt geht es also darum den Entwurfsprozess durch Tendenzen aus Kalkulationen, Analysen, Informationen und Simulationen zu unterstützen, ohne dass dabei der Übergang von realer zu digitaler Welt eine Barriere darstellt (Schubert, 2012, S. 1-3).

1.1.2 Interaktiver Tisch

Das interdisziplinäre Projekt Collaborative Design Platform (CDP) an der Technische Universität München (TUM) hat sich den Herausforderungen des frühen Entwurfs mit der Entwicklung eines interaktiven Tisches genähert.

> *„Im Kern des Projektes steht die Fragestellung nach Möglichkeiten der digitalen Unterstützung in frühen kreativen Entwurfsphasen. Auf Basis der Analyse des Entwurfsprozesses und den damit verbundenen Anforderungen an Entwurfswerkzeuge wurde eine digitale Design-Plattform konzipiert und prototypisch umgesetzt.“* (Schubert, 2012)

Das CDP Projekt wurde hardwaretechnisch wie in Abbildung 1.1 ersichtlich umgesetzt. Der

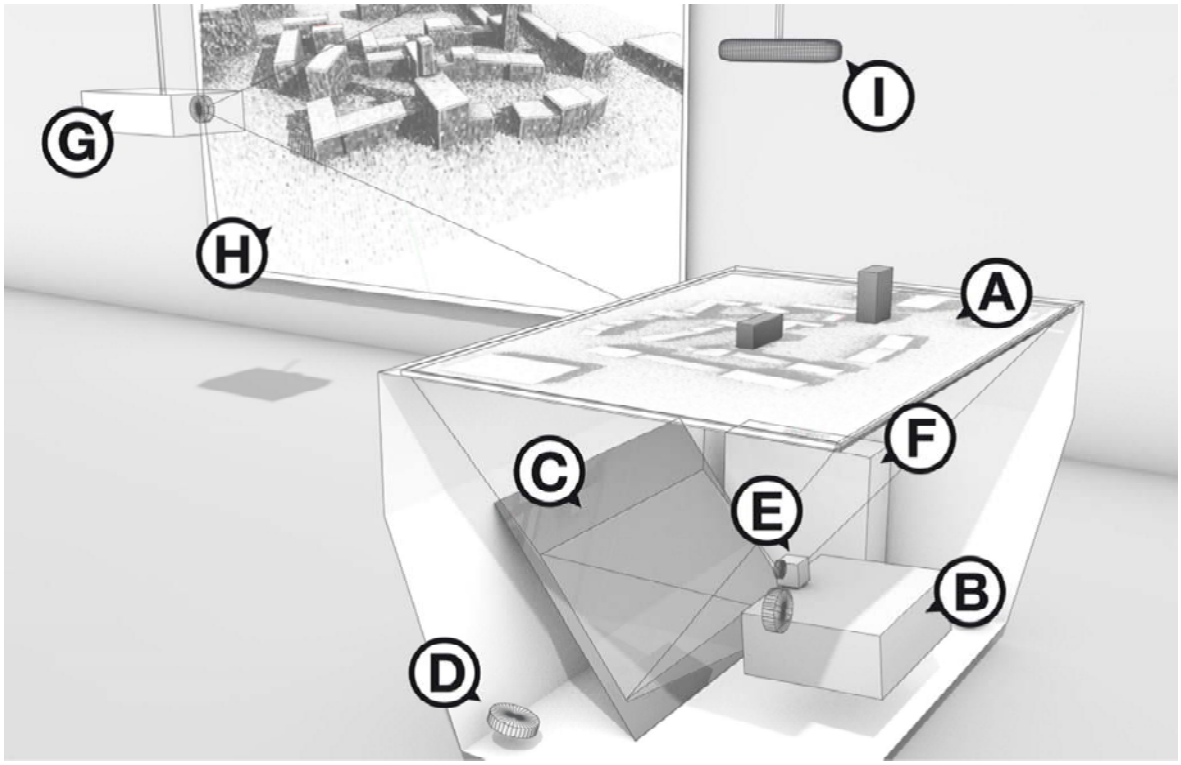


Abbildung 1.1: Schematische Darstellung des interaktiven Tisches der CDP: der interaktive Tisch (A), der erste Beamer (B), der Spiegel zwischen A und B (C), die beiden Infrarotsensoren (D und E), die Recheneinheit (F), der zweite Beamer (G) mit der Projektionsfläche (H) und die Tiefenkamera (I) (Schubert, 2012).

Tisch besteht im Wesentlichen aus zwei Projektionsflächen. Zum einen aus der Oberfläche

des Tisches, welche den Geländeplan in der Perspektive von oben darstellt. Zum anderen aus einer Leinwand mit der perspektivischen Ansicht des Geländes, um das räumliche Vorstellungsvermögen zu unterstützen. Mittels Infrarotsensoren innerhalb des Tisches werden Berührungen und Objekte auf der Tischoberfläche wahrgenommen. Zusätzlich wird eine Tiefenkamera verwendet, um eine Dreidimensionale (3D) Punktwolke aus den sich auf dem Tisch befindlichen Gegenständen, wie zum Beispiel Holzklötzen zu erzeugen.

Durch die Kombination dieser verschiedenen Technologien wird eine Eingabe über Fingerberührung, Stift und 3D Objekte ermöglicht. Dies kommt den herkömmlichen Entwurfsmethoden wie Stift und Papier oder Modellen aus Holz schon äußerst nahe und vereinfacht so die Handhabung.

Softwaretechnisch wurde eine Pluginarchitektur umgesetzt. Diese besteht aus der Middleware (in der Programmiersprache C++ geschrieben), die Input und Output verwaltet und zusätzlich die Anbindung einzelner Plugins (in der Programmiersprache C# geschrieben) ermöglicht. Innerhalb der Plugins können dann je nach Anwendungsgebiet neue Tools erstellt werden, wodurch eine ständige Erweiterung der Funktionsmöglichkeiten stattfinden kann. Die visuelle Darstellung der Berechnungen erfolgt über die Open Graphics Library (OpenGL). Anschließend können die erstellten Geometrien im Industry Foundation Classes (IFC) Format abgespeichert werden (Schubert, 2012, S. 4-7).

1.2 Trassierungsaufgabe

Um nun zum eigentlichen Thema dieser Arbeit zu kommen, der Trassierungsplanung, werden die typischen Planungsphasen eines Infrastrukturprojekts exemplarisch anhand einer Straße erläutert. Die folgenden Abläufe können aber auch sinngemäß auf andere Verkehrswegeprojekte, wie zum Beispiel Eisenbahntrassierungen übertragen werden können.

Der Bau einer Straße von der Aufnahme in den Bedarfsplan bis hin zum befahrbaren Bauwerk ist ein langwieriger Prozess, der viel Planung erfordert und vorgeschriebene rechtliche Verfahren durchlaufen muss. Dieser kann in fünf verschieden Planungsschritte unterteilt werden.

Der erste Schritt ist die Bedarfsplanung. Dabei handelt es sich um Beschlüsse des Bundestages, welche Erweiterungen und Neubauten von Straßen als notwendig erachtet werden und damit in den Bundesverkehrswegeplan aufgenommen werden. Für gewöhnlich werden die Projekte in Kategorien unterschiedlicher Dringlichkeit eingeteilt, um Maßnahmen zum passenden Zeitpunkt in die Wege zu leiten.

Fortgefahren wird mit der Vorplanung. Hierbei legt man den Planungsraum fest, welcher sämtliche Gebiete beinhalten muss, die von möglichen Planungsvarianten betroffen sein können. Hauptaugenmerkt liegt auf dem Entwurf verschiedener Linienführungen, die anhand

vorläufiger Kostenrechnungen und Auswirkungen auf ihr Umfeld miteinander verglichen werden. Abschluss dieser Phase bildet das Raumordnungsverfahren. Bereits hier kommt es zur Beteiligung der Öffentlichkeit, da Bedenken und negative Auswirkungen des Bauvorhabens vorgebracht werden können. Ziel des Raumordnungsverfahrens ist es, Auswirkungen auf das Umfeld möglichst vollständig zu erfassen und notwenige Ausgleichsmaßnahmen festzulegen. Zu beachten ist, dass der Raumordnungsbeschluss noch keine rechtliche Genehmigung für den Bau darstellt.

Bei der Entwurfsplanung, dem dritten Schritt, wird der Straßenverlauf dann weiter konkretisiert. Dies geschieht nicht nur getrennt im Lage- und Höhenplan, sondern wird auch gesamtheitlich in allen drei Dimensionen betrachtet. Kleinere Veränderungen an der Linienführung sind hier nach wie vor möglich. In dieser Phase steht die Optimierung des Projekts im Vordergrund. Dabei werden Wirtschaftlichkeit, Leistungsfähigkeit und Auswirkungen auf Luft, Wasser und Umwelt in Betracht gezogen.

Weiter geht es mit der Genehmigungsplanung, welche die weitere Ausarbeitung des Entwurfs und die letztendliche rechtliche Genehmigung beinhaltet. Zuerst werden wichtige Details in für rechtliche Zwecke notwendiger Genauigkeit festgelegt. Hierzu gehört beispielsweise ein Plan mit allen zu erwerbenden Grundstücken. Daraufhin folgt das Planfeststellungsverfahren. Auch hier werden nochmal Einwände der Öffentlichkeit angehört und gegebenenfalls durch Veränderungen am Projekt berücksichtigt. Der Planfeststellungsbeschluss ist das Ende dieser Phase. Dieser hat umfassende rechtliche Wirksamkeit, was bedeutet, dass keine weiteren Genehmigungen von Nöten sind und auch keine Klage gegen das Vorhaben mehr gestellt werden kann.

Letzter Schritt der Planung ist dann die Ausführungsplanung gefolgt von der Vergabe und der letztendlichen Baudurchführung. Als erstes erstellt man ausführungsreife Pläne, mit denen man sich streng an die Unterlagen des Planfeststellungsbeschlusses halten muss. Nach der Durchführung des Grunderwerbs wird das Vorhaben dann öffentlich ausgeschrieben und für gewöhnlich an die Firma mit dem kostengünstigsten Angebot vergeben. Mit Ende der Baudurchführung und Verkehrsfreigabe der Straße ist der Planungsprozess abgeschlossen (Niedersächsische Landesbehörde, 2018).

1.3 Building Information Modeling

Der eben beschriebene Planungsprozess ist zweifellos äußerst umfangreich und erfordert sehr langfristige Planung, doch mit dem noch relativ neuen Konzept von Building Information Modeling (BIM) ist eine Möglichkeit geboten mit Hilfe eines hochwertigen digitalen Datenmodells einen besseren Überblick über das Gesamtprojekt zu behalten. Bei BIM handelt es sich weder um eine spezielle Software, noch geht es alleine um eine digitale dreidimensionale

Darstellung eines Gebäudes, sondern um ein wesentlich umfangreicheres Modell, das vom National BIM Standard der United States wie folgt definiert wird:

> *„Building Information Modeling (BIM) is a digital representation of physical and functional characteristics of a facility. A BIM is a shared knowledge resource for information about a facility forming a reliable basis for decisions during its life-cycle; defined as existing from earliest conception to demolition“* (National BIM Standard, 2018)

Dabei ist ein großer Vorteil, dass das Modell von der Planung bis hin zum Abbruch eines Bauwerks in jeder Phase genutzt werden kann (siehe Abbildung 1.2), um Prozesse zu optimieren und mögliche Schwierigkeit schon vorzeitig erkennen zu können. Speziell die Beteiligung mehrerer Firmen und Disziplinen an einem Projekt macht ein gemeinsames Gebäudemodell besonders wertvoll. So können beispielsweise Kollisionsprüfungen, Kostenermittlungen und weitere Analysen durchgeführt werden, die die Risiken des Gesamtprojektes verringern. Wichtigste Voraussetzung dafür ist das regelmäßige Zusammentragen der Einzelmodelle der unterschiedlichen Gewerke zu einem Gesamtmodell und somit eine enge Zusammenarbeit aller Beteiligten.

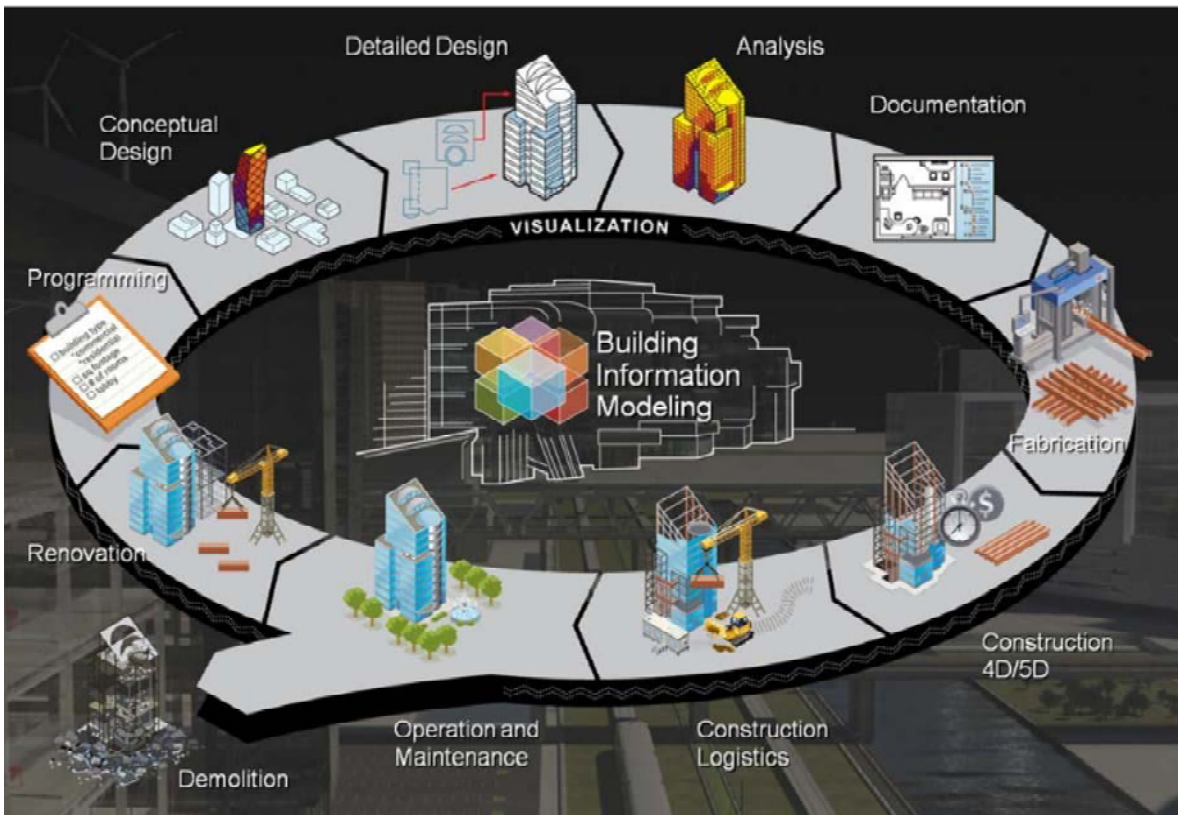


Abbildung 1.2: Lebenszyklusphasen eines Gebäudes bei welchen das Building Information Model zur Anwendung kommen kann (Apogea, 2018).

Ein weiterer Vorteil ist, dass trotz des höheren Planungsaufwandes vor Errichtung des Bauwerks im späteren Verlauf Geld eingespart werden kann. Beispielsweise können durch ein umfassendes digitales Modell bereits vorher Aussagen über die Unterhaltungskosten gemacht werden, welche über viele Jahre hinweg anfallen und somit einen nicht zu vernachlässigenden Anteil der Gesamtgebäudekosten ausmachen (Egger *et al.*, 2013, S. 18-26, S. 32-33). Allerdings bringt die Intensivierung der Planung in den früheren Phasen nicht nur Vorteile mit sich, denn um aussagekräftige Analysen durchführen zu können braucht es auch eine Vielzahl an Informationen. Die Herausforderung ist es dabei, den optimalen Grad an Detaillierung zu verschiedenen Zeitpunkten zu finden. Speziell in Deutschland ist der Umgang mit BIM erst in den Anfangszügen. So ist es häufig der Fall, dass Gebäude schon in Phasen in denen noch große Veränderungen auftreten können zu genau modelliert werden. Folglich ist es also wichtig vorab konkrete Ziele festzulegen, um zum einen die notwenigen Daten vollständig einpflegen zu können und zum anderen nicht durch überflüssige Daten Zeit und Überblick zu verlieren (Steinmann, 2018).

1.4 Ziel der Arbeit

Aus den bisher beschriebenen Herausforderungen und Chancen zum Thema frühe Entwurfsgestaltung von verkehrswegebaulichen Projekten ergibt sich die Motivation hinter dieser Arbeit. Ziel ist die Entwicklung einer Software für den interaktiven Tisch des CDP Projekts, die die Gestaltung eines ersten Entwurfs für ein Alignment ermöglichen soll. Dabei ist das Alignment als vereinfachte Darstellung einer Linienführung mit Hilfe einer dreidimensionalen Raumkurve zu verstehen. Der Fokus des Programms liegt auf einer einfachen und intuitiven Bedienung, damit verschiedene Linienführungen erstellt und miteinander verglichen werden können. Durch die Ausgabe des Entwurfes als IFC Datei soll die entworfene Linienführung auch in den späteren Phasen Verwendung finden. Beispielsweise kann die Datei in ein CAD-Programm importiert werden, um weitere Details auszuarbeiten.

Kapitel 2

Theoretische Grundlagen des Linienführungsentwurfs

Um den dreidimensionalen Verlauf einer neuen Raumkurve, beispielsweise zur Beschreibung eines Straßenverlaufs, ohne größere Schwierigkeiten entwerfen zu können wird der Entwurfsprozess zweigeteilt. Zum einen in die Achse und zum anderen in die Gradiente. Es ist jedoch von großer Bedeutung die beiden Bereiche während der Planung immer wieder aufeinander abzustimmen, um auch im Dreidimensionalen eine verkehrstaugliche Linienführung zu erreichen. Der Vollständigkeit halber ist auch die Querschnittsgestaltung als Teil des Planungsprozesses zu erwähnen, da diese die Entscheidung für die Linienführung allerdings nur sehr geringfügig beeinflusst, ist sie hier nicht weiter von Bedeutung (RAL, 2012, S. 35-37).

2.1 Achse

Die Achse (Abbildung 2.1), die man durch die Projektion der 3D Raumkurve auf die XY-Ebene erhält, sozusagen der Linienverlauf im Lageplan, kann dabei durch unterschiedliche Abfolgen von drei Grundelemente gestaltet werden. Diese sind die Gerade, der Kreisbogen und der Übergangsbogen, als Verbindungsstück zwischen den beiden Erstgenannten. Sowohl bei Verkehrswegen für Auto, als auch für Zug oder Magnetschwebebahn kommen alle drei Elemente zum Einsatz (S. 11-12 Amann *et al.*, 2015; Matthews, 2010, S. 66-67).

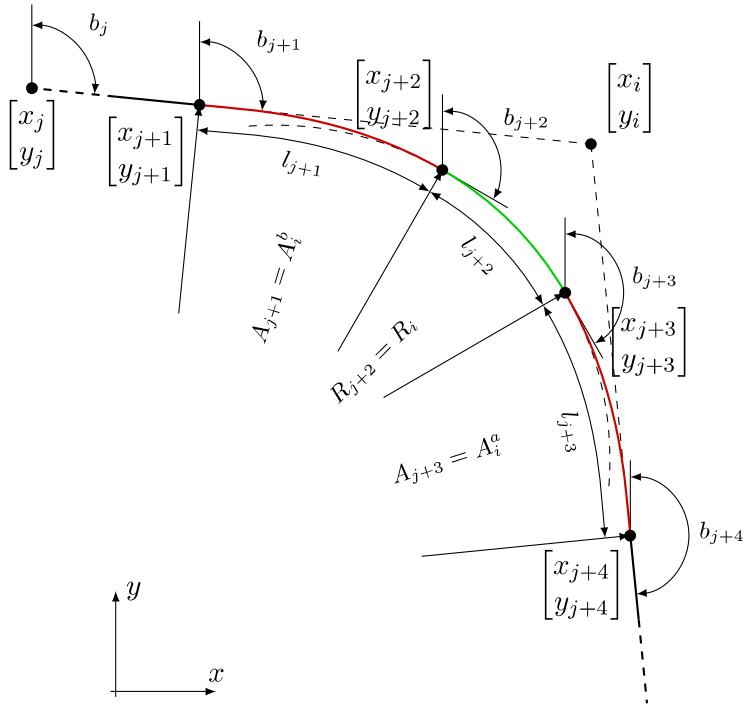


Abbildung 2.1: Typische Darstellung einer Achse mit einem Vektor einzelner Segmente (Gerade (schwarz), Übergangsbogen (rot), Kreisbogen (grün)) und einem Tangentenschnittpunkt (Markič *et al.*, 2018).

2.1.1 Gerade

Die Gerade stellt die kürzeste Verbindung zwischen zwei Punkten dar. Zunächst wird diese im Hinblick auf Straßen betrachtet, bei denen sich folgende Vorteile ergeben:

- Gute Anpassung an Zwangsbedingungen wie Städte oder Täler
- Hohe Übersichtlichkeit bei Knotenpunkten, Ein- und Ausfahrten

Allerdings können Geraden auch erhebliche Nachteile mit sich bringen, speziell wenn sie über größere Straßenabschnitte verlaufen:

- Die Abstände und Geschwindigkeiten von umliegenden Fahrzeugen können nur schwer eingeschätzt werden
- Der Fahrer wird zur Überschreitung des Tempolimits verleitet
- Ein Mangel an Abwechslung fördert die Ermüdung und hat somit einen negativen Einfluss auf die Verkehrssicherheit

Aus diesem Grund sind die Längen von Geraden im Fall von Autobahnen auf maximal 2000 m und im Fall von Landstraßen auf 1500 m zu beschränken. Ebenso wird von Geraden als

Zwischenstück zweier gleichsinnig gekrümmter Kurven abgeraten, da diese oftmals nicht als eigenständiges Straßenelement erkannt werden. Falls nicht vermeidbar sollen diese jedoch eine Mindestlänge von 400 m betragen (S. 35 RAA, 2008; RAL, 2012, S. 27).

Bei der Trassierung von Bahnstrecken stellt die Gerade hingegen das optimale Trassierungselement dar, da auf gerader Strecke die Last gleichmäßig auf beide Schienen verteilt wird. Folglich ergibt sich eine minimale Belastung, sowohl für die Räder, als auch für die Schienen selbst. Aus diesem Grund sind durch die Richtlinien 800.0110 der DB Netz AG (2009) lediglich Mindestlängen für gerade Abschnitte vorgegeben. Diese sind von der Geschwindigkeit v abhängig und können folgendermaßen berechnet werden.

$$\begin{aligned} &bei && v \leq 70\ km/h && min\ l_b\ bzw.\ l_g \geq 0,10 \cdot v[m] \\ &bei\ 70 < && v \leq 100\ km/h && min\ l_b\ bzw.\ l_g \geq 0,15 \cdot v[m] \\ &bei && v > 100\ km/h && min\ l_b\ bzw.\ l_g \geq 0,20 \cdot v[m] \end{aligned} \tag{2.1}$$

mit l_b = Mindestlänge Gleisbogen [m]
l_g = Mindestlänge Gerade [m]

Darüber hinaus wird für durchgehende Hauptgleise jedoch ein Regelwert nach der Gleichung (2.2) empfohlen.

$$reg\ l_b\ bzw.\ l_g \geq 040 \cdot v[m] \tag{2.2}$$

2.1.2 Kreisbogen

Bei der Wahl der Radiusgröße sind vor allem die Höchstgeschwindigkeit der zugrundeliegenden Straße und eine möglichst gute Anpassung an den Geländeverlauf zu beachten. Die Mindestradien sind dabei durch die Reibungskraft zwischen Reifen und Fahrbahn festgelegt und in tabellarischer Form für die unterschiedlichen Entwurfsklassen gegeben. Ebenso sind Mindestlängen vorgeschrieben, um ruckartiges Ein- und Auslenken zu vermeiden.

Darüber hinaus sind auch vorausgehende und nachfolgende Straßenelemente zu berücksichtigen. Bei aufeinanderfolgenden Radien ist zu beachten, dass diese nicht zu sehr voneinander abweichen. Die empfohlenen Radiusabfolgen sind mittels der Radientulpe (siehe Abbildung 2.2) festgelegt. Abhängig von der vorliegenden Entwurfsklasse muss der gute bis brauchbare Bereich eingehalten werden. Falls Radien im zu vermeidenden Bereich dennoch nicht verhindert werden können, ist dies mit entsprechenden Straßenschildern oder gar Geschwindigkeitsbegrenzungen zu kennzeichnen (RAL, 2012, S. 35-36).

Im Hinblick auf Bahnschienen ist gleichermaßen die Höchstgeschwindigkeit bestimmender Faktor für die Wahl eines geeigneten Radius, welcher sich mit der Formel (2.3) bestim-

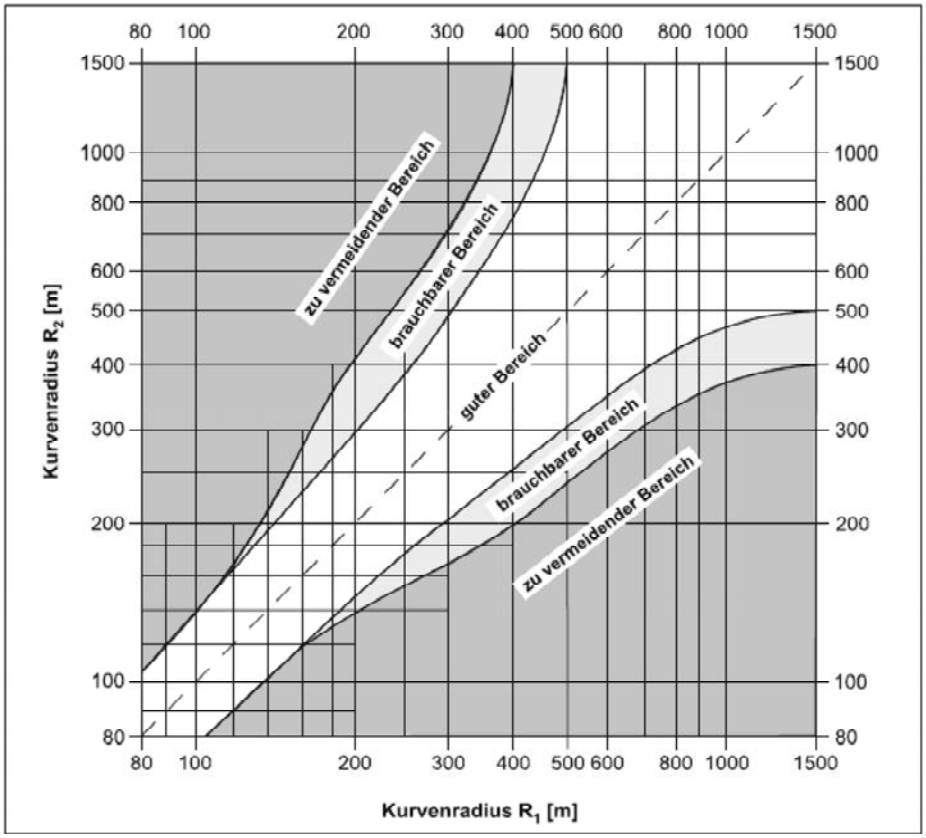


Abbildung 2.2: Verhältnis empfohlener Radiusfolgen nach RAL (2012).

men lässt. Zudem ist auf die Überhöhung und den Überhöhungsfehlbetrag zu achten. Die Überhöhung stellt die leichte Erhöhung der äußeren Schiene innerhalb eines Kreisbogens dar, welche die Auswirkung der Fliehkraft verringert. Der Überhöhungsfehlbetrag ist hingegen der Anteil der Überhöhung, der fehlt um Fliehbeschleunigung komplett auszugleichen.

$$R = 11,8 \cdot \frac{v^2}{u + u_F} \; [m] \tag{2.3}$$

mit R = Radius [m]
v = Geschwindigkeit [m/s]
u = Überhöhung [mm]
u_F = Überhöhungsfehlbetrag [mm]

Unabhängig von der eben genannten Formel sollte $150m \leq R < 25000m$ stets eingehalten werden. Die Bogenlänge kann analog zur Geraden mit den Gleichungen (2.1) berechnet werden (DB Netz AG, 2009).

2.1.3 Übergangsbogen

Um Krümmungssprünge zwischen einer Gerade und einem Kreisbogen, bzw. zweier Kreisbögen zu vermeiden, werden im Verkehrswegebau Übergangsbögen verwendet. Diese führen zu einem kontinuierlichen Krümmungswechsel und verbessern so den Fahrkomfort. Hierbei gibt es drei gängige Varianten von Übergangsbogen. Die lineare Änderung der Krümmung, die sogenannte Klothoide, die mehr geschwungene Krümmungslinie nach Bloss und den Übergang mit s-förmiger Krümmungslinie, auch Sinusoide genannt (Nottbeck, 2016, S. 44-50).

Eisenbahntrassierungen machen von allen drei Varianten des Übergangsbogens Gebrauch, wohingegen im Straßenbau quasi ausschließlich Klothoiden zum Einsatz kommen. Da die Klothoide von beiden Verkehrsarten benutzt wird, ist Weiteres am Beispiel der Klothoide verdeutlicht.

Unter einer Klothoide kann man sich eine Art Spirale vorstellen. Dabei wird stets nur ein Bruchteil einer Klothoide als Übergangsbogen verwendet. Damit eine bestmögliche Anpassung an anschließende Trassierungselemente gegeben ist, wird die Klothoide über den Klothoidenparameter in der Länge verändert. Die Form bleibt hierbei unverändert. Der Zusammenhang von Länge, Radius und Klothoidenparameter ist nach folgender Formel gegeben.

$$A^2 = R \cdot L \tag{2.4}$$

mit A = Klothoidenparameter [m]
R = Radius des nachfolgenden Kreisbogens [m]
L = Länge der Klothoide [m]

Bei der Wahl des exakten Werts für den Parameter A, werden möglichst hohe Werte empfohlen. Im Straßenbau ist jedoch darauf zu achten, dass der Klothoidenparameter $A_{min} = \frac{R}{3}$ nicht unterschreitet und $A_{max} = R$ nicht überschreitet (Freudenstein, 2017b, S. II17-II21).

2.2 Gradiente

Neben der Planung der Linienführung im Lageplan muss auch die vertikale Linienführung ausgearbeitet werden. Dabei stellt eine Koordinate die Länge entlang der Achse dar und die zweite die Höhe Z (Amann *et al.*, 2015, S. 11-12). Ziel ist es eine möglichst gute Geländeanpassung zu erreichen, um die Erdarbeiten möglichst gering zu halten und so die Errichtungskosten abzumindern. Dabei darf allerdings die Verkehrssicherheit nicht vernachlässigt werden. Da der Höhenverlauf der Raumkurve maßgeblich durch die Lage der Achse im Gelände bestimmt wird, sind horizontale und vertikale Planungsschritte immer gemeinsam zu betrachten. Die

Gradiente (Abbildung 2.3), welche den vertikalen Straßenverlauf darstellt, besteht zum einen aus Abschnitten mit konstanter Steigung und zum anderen aus Wannen- und Kuppenausrundungen (Queensland Government & Roads, 2018, S. 12.1). Im Falle von Magnetschwebebahnen kommen, analog zur Achse, zusätzlich auch Übergangsbögen zur Verwendung (Matthews, 2010, S. 66-68).

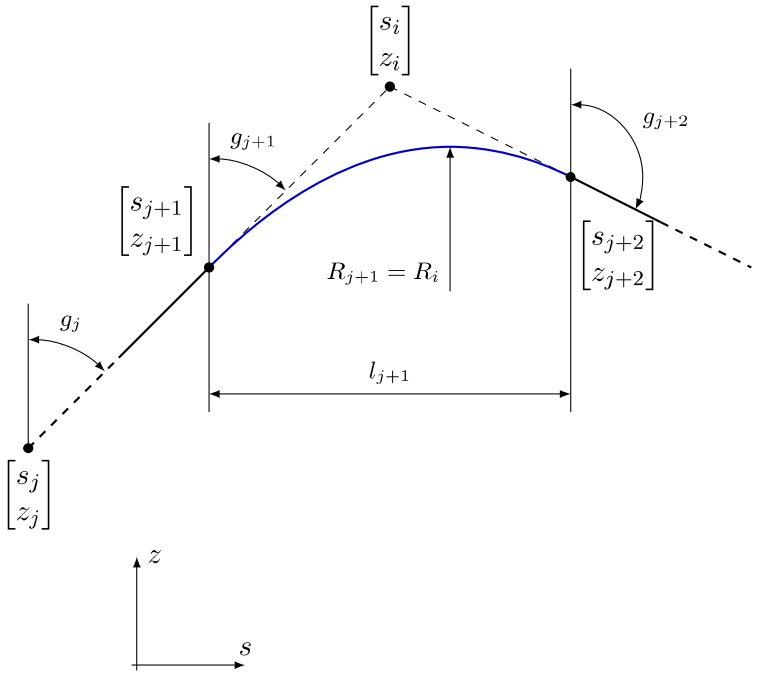


Abbildung 2.3: Darstellung einer Gradiente zum einen durch einzelne Segmente, wie Längsneigungen (schwarz) und Kuppenausrundungen (blau), und zum anderen durch Tangentenschnittpunkte (Markič *et al.*, 2018).

2.2.1 Längsneigung

Längsneigungen sind Elemente mit konstanter Steigung. Wobei abzuwägen ist, wie groß die Steigung gewählt wird, da sowohl flachere als auch steilere Abschnitte ihre Vorteile haben. Im Straßenverkehr erhöhen geringere Längsneigungen den Verkehrsfluss und bieten so höhere Kapazität, denn die geringe Steigung erlaubt höhere Geschwindigkeiten und sorgt außerdem für gleichmäßigere Fahrtgeschwindigkeiten in Betracht auf verschiedene Fahrzeugklassen. Dadurch wird auch die Straßenabnutzung begrenzt und somit die Instandhaltungskosten abgesenkt. Auf der anderen Seite haben höhere Längsneigungen den Vorteil, dass die Straße besser dem Gelände angepasst werden kann. Dies verringert die notwendigen Erdarbeiten und damit die Baukosten erheblich.

Damit die Verkehrssicherheit auf jeden Fall gewährleistet werden kann, gibt es in Deutschland maximale Längsneigungswerte s_{max}, die nicht überschritten werden dürfen. Diese sind in Tabelle 2.1 aufgelistet.

Autobahnen		Landstraßen	
Entwurfsklasse	**max. s (%)**	**Entwurfsklasse**	**max. s (%)**
EKA 1 A	4,0	EKL 1	4,5
EKA 1 B	4,5	EKL 2	5,5
EKA 2	4,5	EKL 3	6,5
EKA 3	6,0	EKL 4	8,0

Tabelle 2.1: Maximale Längsneigungen je nach Entwurfsklasse Landstraße (EKL) und Entwurfsklasse Autobahn (EKA) (Freudenstein, 2017b).

Der Unterwert ist gegeben durch die zwingende Fähigkeit der Straße Wasser auch an Stellen mit fehlender Querneigung ableiten zu können. Er ist auf $s_{min} \geq 1\%$ festgelegt und darf nur im äußersten Ausnahmefall unterschritten werden (S. 29 RAA, 2008; RAL, 2012, S. 39).

Gleise haben, aufgrund der geringen Reibung zwischen Schiene und Rad, im Bezug auf Längsneigungen einen wesentlich geringeren Spielraum. Die maximale Längsneigung darf bei Hauptbahnen 12,5 ‰ und bei Nebenbahnen und S-Bahnen 40 ‰ nicht überschreiten. Unterwert zur Gewährleistung der Entwässerung ist 1 ‰ (DB Netz AG, 2009).

2.2.2 Wannen- und Kuppenausrundung

Sämtliche Änderungen der Steigung im Höhenplan werden entweder mit quadratischen Parabeln oder mit Kreissegmenten ausgerundet. Bei einer Abnahme der Steigung von einer Längsneigung zur nächsten spricht man von einer Kuppenausrundung. Nimmt die Steigung zu handelt es sich hingegen um eine Wannenausrundungen. Die Ausrundungen können über den Ausrundungshalbmesser gesteuert werden, welcher den Halbmesser des Scheitelkrümmungskreises der Parabel angibt. Ähnlich wie bei den Längsneigungen gibt es für diesen Parameter ebenso Tabellen abhängig von Verkehrsmittel und Geschwindigkeit, welche einzuhaltende Wertebereiche angeben. In der Regel sind zwei Ausrundungen mit einem Geradenstück verbunden, sie können aber genauso unmittelbar aneinander anschließen (Freudenstein, 2017a, S. III8-III11). In Abbildung 2.3 ist beispielhaft der Übergang zwischen zwei Längsneigungen (schwarz) mit einer Kuppenausrundung (blau) aufgezeigt.

2.3 Tangentenschnittpunkte

Für die eindeutige Beschreibung von Achse und Gradiente gibt es mehrere unterschiedliche Herangehensweisen. Eine davon ist die Angabe einer Liste von Tangentenschnittpunkten. Die einzelnen Schnittpunkte können durch die Verlängerung aller Elemente ohne Krümmung gewonnen werden, wie auch in den Abbildungen 2.3 mit Hilfe der gestrichelten Linie verbildlicht

wird (Amann & Borrmann, 2015, S. 6-8). Im Fall von vertikalen Tangentenschnittpunkten wird jeder Punkt durch den Ort s (engl. station) und die Höhe e (engl. elevation) festgelegt. Über die Angabe des Ausrundungshalbmessers R kann anschließend die Länge der Ausrundungen l_{j+1} und der dazwischen liegenden Längsneigungen konkretisiert werden (Markič, 2017, S. 7-10).

Kapitel 3

Industry Foundation Classes

Durch die zunehmende Verwendung von BIM Modellen kam es in den vergangenen Jahren zur Herausforderung Gebäude Daten unterschiedlicher Herkunft zu einem Gesamtmodel zusammen zu führen. Das Problem ist vor allem die große Bandbreite an Software Programmen, die in der Baubranche eingesetzt werden. Meist wird für jedes Gewerk ein auf die jeweiligen Aufgabenstellungen angepasstes Programm verwendet. Dadurch können die einzelnen Teilmodelle eines Projekts allerdings nicht ohne weiteres zusammen geführt werden (Allplan, 2015, S. 10).

Mit der Einführung der IFC durch den e.V. buildingSmart International [1], einem Zusammenschluss mehrerer internationaler Firmen, wurde dieses Defizit beseitigt. In diesem Datenformat wurden für verschiedene Objekte aus der Baubranche Klassen möglichst allumfassend, aber dennoch sehr allgemein definiert, sodass ein Datenaustausch unabhängig von der verwendeten Software stattfinden kann. Anfangs beschränkte sich das IFC Format auf die Klassifizierung von herkömmlichen Gebäuden, doch mittlerweile wird an der Implementierung von Klassen für Straßen, Eisenbahnstrecken und den zugehörigen Brücken- und Tunnelbauwerken gearbeitet (Amann & Borrmann, 2015, S. 1-4).

Im Folgenden wird die IFC Struktur für das Digitale Gelände Modell (DGM) und das Alignment, die Input und Output darstellen kurz erläutert.

3.1 IfcTriangulatedIrregularNetwork

Ausgangspunkt jedes Infrastrukturprojekts ist das sogenannte Digitale Gelände Modell (DGM). Dabei handelt es sich um einen abgegrenzten Bereich einer Geländeoberfläche. Dieser wird durch einzelne Punkte im dreidimensionalen Raum mit X-, Y- und Z-Koordinaten dargestellt. Durch eine flächendeckende Vernetzung der Punkte mit Dreieckselementen können

[1] www.buildingsmart.de

die Höhenwerte an sämtlichen Stellen interpoliert werden (Martin-Luther-Universität, 2018). Diese Vernetzung wird Triangulated Irregular Network (TIN) genannt.

Gemäß den IFC wird das Gelände nach diesem Schema in der Klasse IfcTriangulatedIrregularNetwork abgespeichert. Damit die einzelnen Dreiecke jedes Mal auf die selbe Art und Weise gebildet werden, besitzt die Klasse neben einer Liste mit Punktkoordinaten auch eine Liste, in der über Indizes jedem Dreieck seine drei Eckpunkte aus der Koordinatenliste zugeordnet werden. Darüber hinaus verfügt sie auch über weitere Attribute, die aber im Rahmen dieser Arbeit nur von untergeordneter Bedeutung sind.

Damit das zu generierende Netz fehlerfrei ist, gibt es für das IfcTriangulatedIrregularNetwork Bedingungen, die eingehalten werden müssen. Zum einen darf für jeden Punkt in der XY Ebene nur ein Z-Wert existieren. Des Weiteren wird vorgeschrieben, dass für sämtliche Dreiecke die Eckpunkte mit gleichem Orientierungssinn angegeben werden müssen. Dritte und wichtigste Bedingung ist, dass es keine zwei Dreiecke gibt, die einander überlappen (Borrmann *et al.*, 2017, S. 9-11).

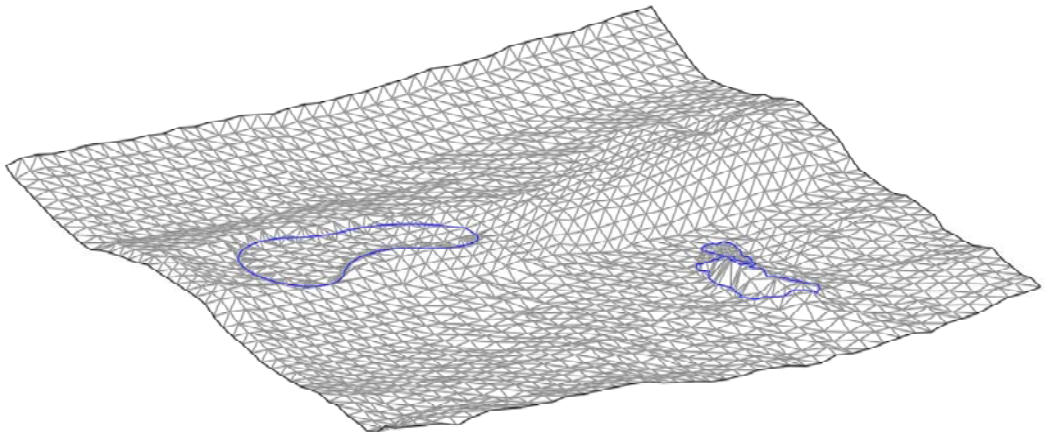


Abbildung 3.1: DGM dargestellt durch IfcTriangulatedIrregularNetwork in AutoCAD. Bruchkanten sind in blau dargestellt

3.2 IfcAlignment

Das Alignment besteht aus den drei Unterklassen: Horizontal Alignment, Vertical Alignment und 3D Alignment, unabhängig davon ob es den Verlauf einer Straße, einer Eisenbahntrasse oder Ähnlichem beschreibt (siehe Bild 3.2). Dadurch gibt es mehrere Möglichkeiten ein Alignment im IFC Format abzuspeichern:

- Definition von horizontalem, vertikalen und 3D Alignment
- Definition über horizontales und vertikales Alignment
- Festlegung alleine über 3D Alignment
- Festlegung nur über horizontales Alignment (für sehr frühe Planungsphasen gedacht)

Darüber hinaus besteht die Möglichkeit horizontale und vertikale Alignments einander zuzuordnen. Jedem vertikalen kann genau ein horizontales Alignment zugewiesen werden, anders herum können mit einem horizontalen aber durchaus mehrere vertikale Alignments verknüpft werden (Amann *et al.*, 2015, S. 1-2, S. 9).

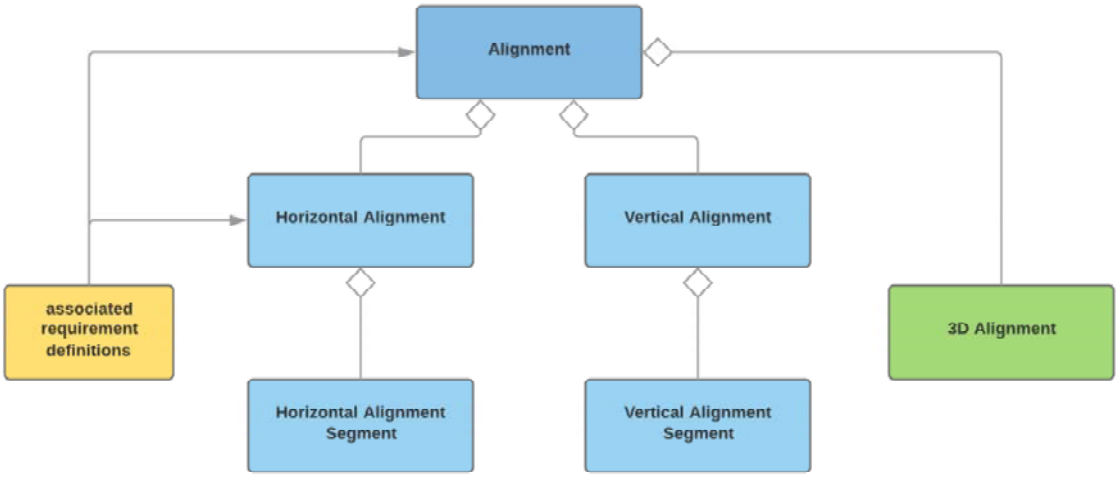


Abbildung 3.2: Klassenstruktur des IfcAlignments (Amann *et al.*, 2015).

3.2.1 IfcAlignment2DHorizontal

Der horizontale Teilbereich kann weiter untergliedert werden. Jedes horizontale Alignment besteht aus einer Liste von einzelnen Segmenten, die in einer festgelegten Reihenfolge angeordnet sind. Wie auch in Kapitel 2.1.1 beschrieben kommen unterschiedliche Elementtypen zum Einsatz und werden dem entsprechend in nachfolgende Klassen eingeteilt:

- IfcLineSegment2D (Gerade)
- IfcCircularArcSegment2D (Kreisbogen)
- IfcTransitionCurveSegment2D (Übergangsbogen)

Aufgrund der Tatsache, dass für alle Segmentarten sehr ähnliche Eingangsparameter notwendig sind, wurden sie der Basisklasse IfcCurveSegment2D hinzugefügt. Dort werden die Attribute

- Startposition
- Startrichtung
- Segmentlänge

definiert und an die jeweiligen Segmentklassen weitervererbt. Für Segmentarten, die weitere Paramater gebrauchen, wie zum Beispiel der Radius im Falle eines Kreisbogens, wurden diese als Attribute in der entsprechenden Klasse hinzugefügt. Die genau Klassenanordnung und die im IFC Format verwendeten Bezeichnungen sind in der Abbildung 3.3 zu finden.

Um ein regelkonformes Alignment zu erstellen, ist noch darauf zu achten, dass der Endpunkt eines Trassierungselements mit dem Startpunkt des darauf folgenden Elements übereinstimmt. Ein tangentialer Anschluss an ein vorhergehendes Segment ist nicht zwingend erforderlich, kann aber ohne weiteres mit angegeben werden (Amann & Borrmann, 2015; buildingSmart International, 2018, S. 3-5).

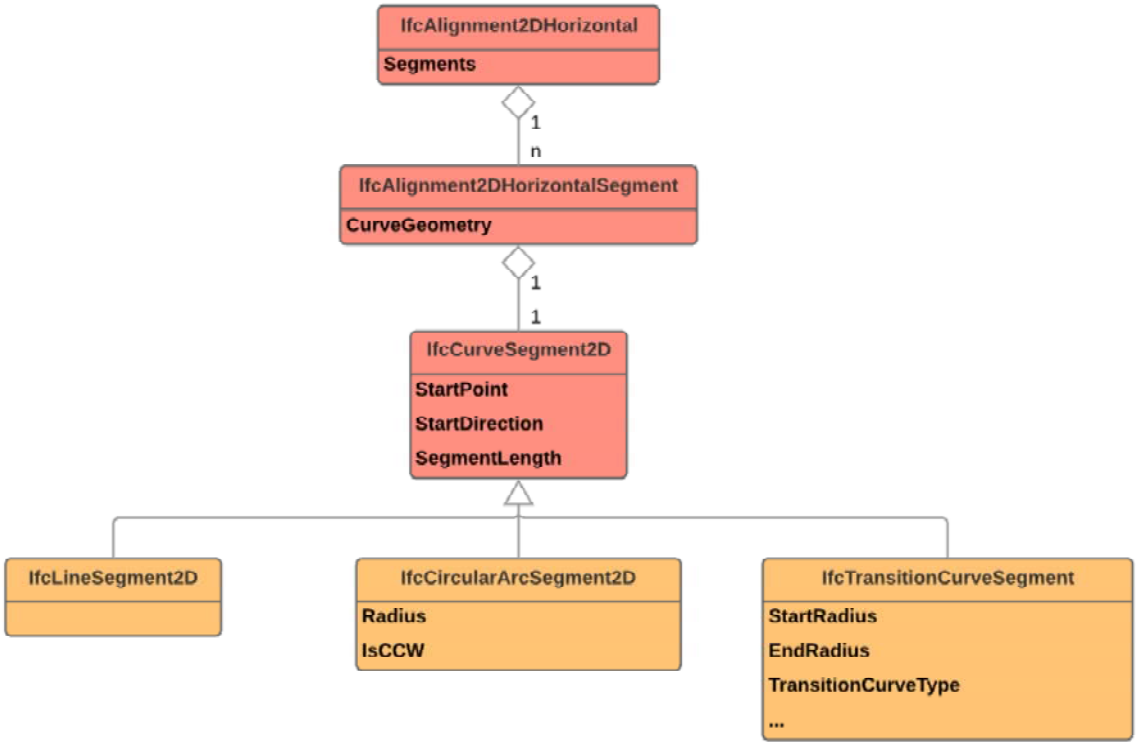


Abbildung 3.3: Klassenstruktur des horizontalen Alignments in IFC (buildingSmart International, 2018).

3.2.2 IfcAlignment2DVertical

Das vertikale Alignment (IfcAlignment2DVertical) hat einen sehr ähnlichen Aufbau wie das horizontale Alignment. Ebenso wie das horizontale Alignment besteht es aus einer Liste einzelner Segmente, die den Klassen

- IfcAlignment2DVerSegLine (Längsneigung)
- IfcAlignment2DVerSegCircularArc (kreisförmige Ausrundung)
- IfcAlignment2DVerSegParabolicArc (parabolische Ausrundung)

zugeordnet werden. Attribute, die von allen Segmenten verwendet werden, sind in der Klasse IfcAlignment2DVerticalSegment festgelegt. Diese sind:

- Starthöhe
- Neigung am Startpunkt
- Horizontale Länge des Segments
- Anfangslänge bezogen auf das zugehörige horizontale Alignment

Alle weiteren Attribute werden nach wie vor in den einzelnen Segmentklassen definiert. Zusätzlich besitzt das IfcAlignment2DVertical die Eigenschaft *ToAlignment*, welche dem vertikalen Alignment das Horizontale zuordnet. Sämtliche Zusammenhänge sind in der Abbildung 3.4 verdeutlicht (Liebich, 2015, S. 9-15).

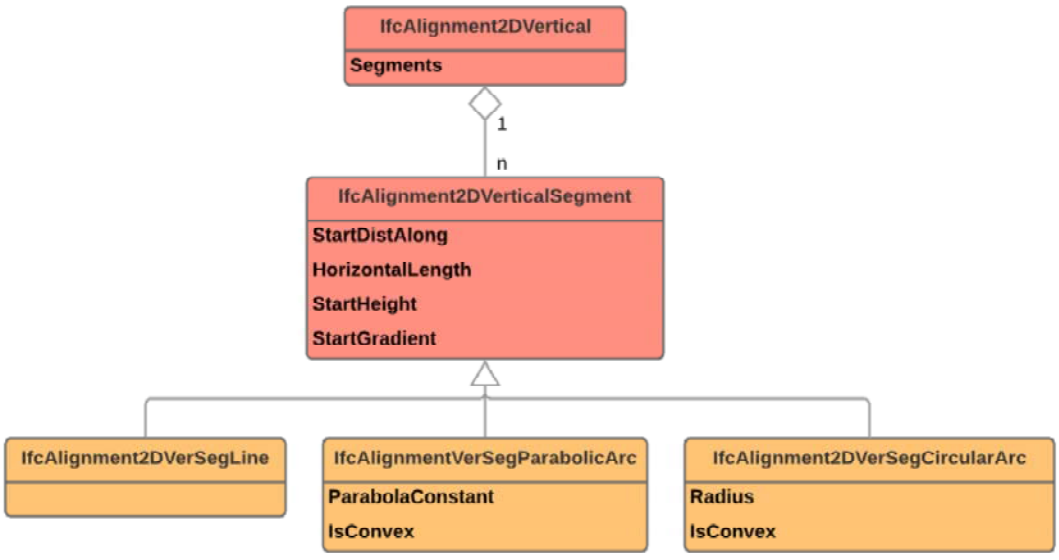


Abbildung 3.4: Klassenstruktur des vertikalen Alignments in IFC (Liebich, 2015).

Kapitel 4

Umsetzung

In diesem Kapitel wird die Umsetzung des Entwurfstools als Plugin für den interaktiven Tisch in Visual C# thematisiert. Zu Beginn wird ein Gesamtüberblick des ablaufenden Prozesses gegeben und in den anschließenden Unterkapiteln dann mit Codeausschnitten genauer auf die einzelnen Teilbereiche eingegangen.

4.1 Gesamtüberblick

4.1.1 Prozess

Ausgangspunkt des Gesamtablaufs innerhalb des Entwurfstools stellt das DGM dar. Dieses ist essentiell, da ohne Kenntnis der Topologie des Geländes kein Anhaltspunkt für eine geeignete Linienführung vorhanden ist. Mit Höhenlinien wird der Geländeverlauf für den Benutzer anschaulich auf dem Tisch dargestellt. Damit kann dann die eigentliche Bearbeitung beginnen. Zuerst kann die Achse der Raumkurve entworfen werden und zwar durch Hinzufügen, Verschieben und Löschen von einzelnen Tangentenschnittpunkten. Außerdem können Ausrundungsradien nach Belieben verändert werden. Als nächster Schritt wird dann der Höhenverlauf entlang der Achse selbstständig berechnet und der eben beschriebene Entwurfsprozess kann gleichermaßen für die Gradiente durchgeführt werden. Sobald Änderungen an der Achse vorgenommen werden, wird das Höhenprofil neu bestimmt und die Gradiente entsprechend der Veränderung gestaucht, gestreckt oder verschoben, damit gleichbleibende Teile der Gradiente nach wie vor mit dem Höhenverlauf entlang der Achse übereinstimmen.

Der Ablauf der Bearbeitung von Achse und Gradiente zusammen mit der automatischen Berechnung des Höhenprofils, kann so lange wiederholt werden, bis das gewünscht Endergebnis vorliegt. Zum Schluss wird die erstellte Raumkurve dann in Form einer IFC Datei exportiert. Dazu muss die Modellierung der Linienführung von der Darstellung mit Tangentenschnitt-

punkten in die Segmentdarstellung überführt werden. Der eben beschriebene Gesamtprozess ist in vereinfachter Form auch in Abbildung 4.1 verbildlicht.

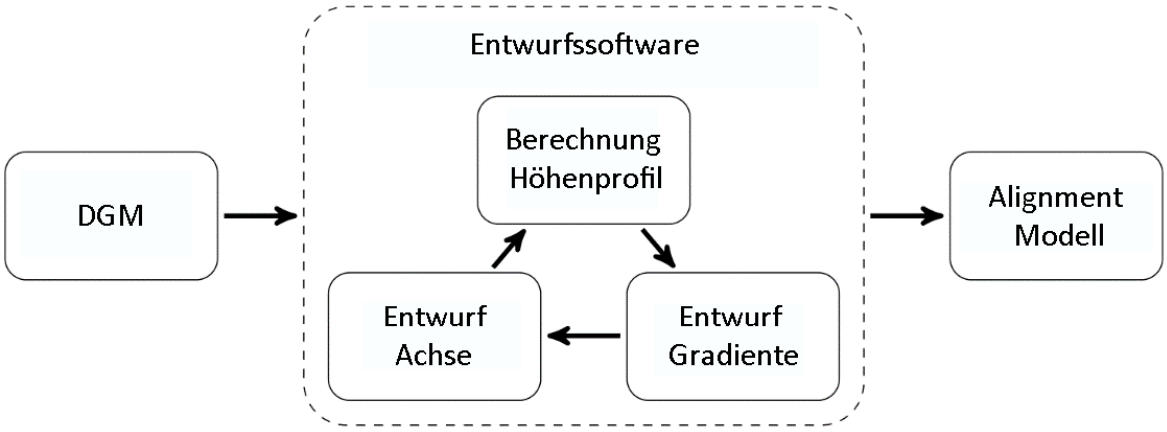


Abbildung 4.1: Gesamtprozess des Entwurfstools (Markič *et al.*, 2018).

4.1.2 Klassenstruktur

Bevor genauer auf die Einzelbereiche eingegangen wird, wird noch kurz die Klassenstruktur des Plugins aufgezeigt, um bestehende Zusammenhänge besser ersichtlich zu machen. Die grundlegendste aller Klassen ist die Klasse *Window*, mit welcher man neue Fenster erstellen kann, innerhalb dieser alle weiteren Funktionen ausgeführt werden. *Window* besitzt die beiden Unterklassen *ControlWindow*, für ein Fenster mit unterschiedlichen Steuerungselementen, und *AlignmentWindow*, für ein Fenster zur Bearbeitung einer Linienführung, das auch ein DGM enthalten kann. Mit den Klassen *Button* und *Slider* kann man einem Steuerungsfenster dann anschließend unterschiedliche Knöpfe und Schieberegler als Kontrollelemente hinzufügen. Alle anderen Klassen werden fast ausschließlich innerhalb des *AlignmentWindow*s benötigt. Hierbei kann über die Variable *eAlign* festgelegt werden ob es sich um eine Fenster für eine Achse oder eine Gradiente handelt. Dem entsprechend werden auch einige Voreinstellungen getroffen. Unabhängig davon besitzt aber jedes *AlignmentWindow* ein Objekt der Klasse *Alignment* in dem die horizontale oder vertikale Linienführung abgespeichert wird. Objekte weiterer Klassen können über zugehörige Funktionen im *AlignmentWindow* nach Belieben erstellt werden. Dazu gehören beispielsweise die Klassen *Profile* für ein Längsprofil, *LevelCurve* für ein DGM mit Höhenlinien, *NorthPointer* für einen Nordpfeil oder *Marker*, um entlang der Linienführung Markierungen in gleichem Abstand zu erstellen (siehe Unified Modeling Language (UML) Diagramm 4.2).

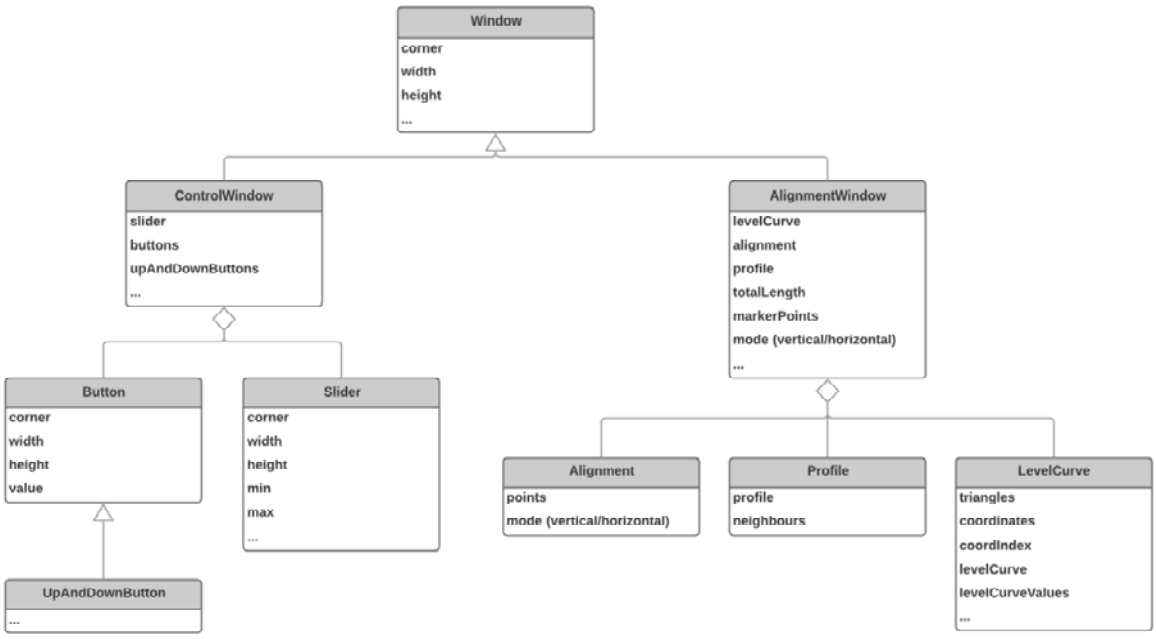


Abbildung 4.2: Klassenstruktur des Plugins in Form eines UML Diagramms.

4.2 DGM

4.2.1 Einlesen aus der IFC Datei

Der allererste Schritt besteht darin, die Datei mit dem vorliegenden Gelände in Form eines TINs einzulesen, um es dann in späteren Schritten auf dem interaktiven Tisch in geeigneter Form anzeigen zu lassen, damit es als Planungsgrundlage dienen kann.

Bei dem Input handelt es sich um eine Datei in IFC Format, weshalb zum Einlesen das xBIM Essentials Nuget Paket verwendet wird. Dieses Paket ist extra für das Einlesen, Erstellen und visuelle Darstellen von BIM Modellen im IFC Format ausgelegt und stellt demensprechenden die dafür notwendigen Befehle zur Verfügung (xBIM, 2018). Hierbei wird Early Binding verwendet, was bedeutet, dass sobald ein neues Objekt eines bestimmten Datentyps erstellt wird, auch eine Verbindung zur jeweiligen Bibliothek erstellt wird. Dadurch ist beispielsweise direkt ersichtlich, welche Methoden und Attribute für dieses Objekt zur Verfügung stehen. Dies erleichtert die Programmierung während der Entwurfszeit, da fehlerhaft angegebene Methoden und Variablen mit für den verwendeten Zweck nicht kompatiblen Datenformaten sofort ersichtlich sind. Damit werden Fehler während der Laufzeit verringert. Zudem macht das Early Binding die Anwendung effizienter und schneller (Microsoft, 2018; Borrmann *et al.*, 2015).

Aufgrund der Tatsache, dass xBIM Essentials das Lesen des aktuellsten IFC4x1 Formats noch nicht unterstützt, wird eine IFC2x3 Datei als Input verwendet. Um alle wesentlichen

Informationen über das Dreiecksnetz zu erhalten, benötigt man lediglich die definierten 3D Punkte (IfcCartesianPoint) und die Information aus welchen Punkten die einzelnen Dreiecke bestehen, was in der Datei über die Definition von IfcPolyLoops festgelegt wird. Im Folgenden wird an Hand eines kurzen Code-Ausschnittes erklärt, wie diese Informationen gewonnen und abgespeichert werden.

Zuerst werden mit Hilfe des IfcStore.Open() Befehls die Informationen aus der Input-Datei in die Variable *model* geladen. Dann wird überprüft wie viele Elemente des Typs IfcCartesianPoint die Datei beinhaltet, um die Matrizen *coordinates* und *hashCodeList* mit selber Größe zu erstellen. In *coordinates* sollen im späteren Verlauf die einzelnen Punkte abgespeichert werden. Die *hashCodeList* hingegen dient dazu, zu vermerken in welcher Zeile der Ursprungsdatei der jeweilige Punkt definiert wurde, denn beim Definieren der Dreiecke wird auf die Punkte an Hand ihrer Zeilennummer, dem Hashcode, verwiesen (Listing 4.1 Z.3-6). Als nächstes werden sämtliche IfcCartesianPoints abgegangen. X-, Y- und Z-Werte werden in *coordinates* eingetragen und die Zeile in der der Punkt festgelegt wird in *hashCodeList* (Listing 4.1 Z.8-19) abgespeichert. Danach werden die IfcPolyLoops abgelaufen, in denen je drei Punkte über den Hashcode angegeben sind. Über die *hashCodeList* findet man heraus, wo sich in der *coordinates* Matrix der dem gesuchten Punkt entsprechende Eintrag befindet (Listing 4.1 Z.21-40). Mit dieser Information wird anschließend eine Matrix (*coordIndex*) erstellt, die ähnlich zu dem IfcPolyLoop auf die drei Punkte eines jeden Dreiecks verweist nur dieses Mal an die Stelle innerhalb von *coordinates*. Abschließend werden die Inhalte der bisherigen Matrizen in einer Matrix (*triangles*) zusammengeführt (Listing 4.1 Z.42-50). Dort werden von allen Eckpunkten jedes Dreiecks X-, Y- und Z-Wert unmittelbar nebeneinander gespeichert, was spätere Prozesse wesentlich erleichtert.

Listing 4.1: Einleseprozess

```
1  using (var model = IfcStore.Open(filename))
2  {
3      var cartesianPoints = model.Instances.OfType<IIfcCartesianPoint>();
4      int count = 0;
5      int[,] hashCodeList = new int[cartesianPoints.Count(), 2];
6      coordinates = new double[cartesianPoints.Count(), 3];

8      foreach (IIfcCartesianPoint point in cartesianPoints)
9      {
10         var x = (double)(point.Coordinates.ElementAt(0).Value);
11         var y = (double)(point.Coordinates.ElementAt(1).Value);
12         var z = (double)(point.Coordinates.ElementAt(2).Value);
13         coordinates[count, 0] = x;
14         coordinates[count, 1] = y;
15         coordinates[count, 2] = z;
16         hashCodeList[count, 0] = count;
17         hashCodeList[count, 1] = point.GetHashCode();
```

```
18          count++;
19      }

21      var tri = model.Instances.OfType<IIfcPolyLoop>();
22      count = 0;
23      int a = 0, b = 0, c = 0;
24      coordIndex = new int[tri.Count(), 3];
25      foreach (IIfcPolyLoop loop in tri)
26      {
27          var firstpoint = loop.Polygon.ElementAt(0).GetHashCode();
28          for (int i = 0; i < hashCodeList.GetLength(0); i++)
29              if (firstpoint == hashCodeList[i, 1])
30              {
31                  a = hashCodeList[i, 0];
32                  break;
33              }
34          ...
35          //selbiges fuer zweiten und dritten Punkt -> b,c
36          coordIndex[count, 0] = hashCodeList[a, 0];
37          coordIndex[count, 1] = hashCodeList[b, 0];
38          coordIndex[count, 2] = hashCodeList[c, 0];
39          count++;
40      }

42      triangles = new double[tri.Count(), 3, 3];
43      for (int i = 0; i < (coordIndex.GetLength(0)); i++)
44      {
45          triangles[i, 0, 0] = coordinates[coordIndex[i, 0], 0];
46          triangles[i, 0, 1] = coordinates[coordIndex[i, 0], 1];
47          triangles[i, 0, 2] = coordinates[coordIndex[i, 0], 2];
48          ...
49          //selbiges fuer die anderen beiden Eckpunkte des Dreiecks
50      }

52      model.Close();
53  }
```

4.2.2 Ableitung einer topologischen Karte

Als nächstes gilt es, das dreidimensionale Dreiecksnetz anschaulich auf der zweidimensionalen Oberfläche des Tisches zu visualisieren. Die Darstellung mittels Höhenlinien ist die gängigste Methode und wird deshalb auch hier angewendet. Da diese nicht unmittelbar aus dem DGM

erkenntlich sind und auch für veränderbare Höhenschritte bestimmt werden sollen, bedarf es einige Rechenschritte.

Zu Beginn sucht man die beiden Stellen mit der minimalen und maximalen Erhebung ($totalMinZ$ und $totalMaxZ$), da diese benötigt werden, um das Höhenintervall festzulegen. Abhängig von der Schrittgröße wird das Höhenintervall des Geländes nach dem Vorgehen in Listing 4.2 gleichmäßig aufgeteilt.

Listing 4.2: Bestimmung der Höhenwerte

```
1 int count = 0;
2 levelCurveValues = new int[(int)((totalMaxZ - totalMinZ) /
      levelCurveDistance) + 1];

4 for (int i = (int)totalMinZ; i <= totalMaxZ; i = i + (int)
      levelCurveDistance)
5 {
6     levelCurveValues[count] = i;
7     count++;
8 }
```

Im Anschluss werden sämtliche Dreiecke daraufhin überprüft, ob sie von einer Höhenlinie geschnitten oder tangiert werden. Wenn die Höhe außerhalb des Dreiecks liegt sind keine weiteren Berechnungen erforderlich, andernfalls muss man den Verlauf der Höhenlinie im Bereich des Dreiecks herausfinden. Dabei gibt es zwei Fälle getrennt zu betrachten.

Beim ersten und einfacheren der beiden Fällen verläuft die Höhenlinie exakt entlang einer der drei Kanten des Dreiecks. Dies ist dann der Fall, wenn die Z-Koordinate von zwei der drei Eckpunkte den gesuchten Höhenlinienwert besitzt. Folglich können die Koordinaten der beiden Eckpunkte als Anfangs- und Endpunkt des Höhenlinienabschnitts übernommen werden, wie in Listing 4.3 verdeutlicht. Die Laufvariable k steht hierbei für den Höhenwert, $levelCurveCount$ zählt mit beim wievielten Höhenliniensegment sich der Prozess gerade befindet und in $levelCurve$ werden Punkte gespeichert entlang derer die Höhenlinien verlaufen. Der eben beschriebene Fall tritt vor allem bei Gewässern innerhalb des DGMs auf, da der Wasserspiegel an allen Stellen annähernd die gleiche Höhe aufweist.

Listing 4.3: Fall 1

```
1 int[] inline = new int[3];
2 for (int m = 0; m < 3; m++)
3 {
4     if (k == triangles[i, m, 2])
5     {
6         inline[ccount] = m;
7         ccount++;
8     }
```

```
9  }
10 if (ccount == 2)
11 {
12     levelCurve[levelCurveCount, 0] = k;
13     levelCurve[levelCurveCount, 1] = triangles[i, inline[0], 0];
14     levelCurve[levelCurveCount, 2] = triangles[i, inline[0], 1];
15     levelCurve[levelCurveCount, 3] = triangles[i, inline[1], 0];
16     levelCurve[levelCurveCount, 4] = triangles[i, inline[1], 1];
17     levelCurveCount++;
18 }
```

Der zweite Fall beinhaltet alle anderen Möglichkeiten, also wenn die Höhenlinie zwei Kanten des Dreiecks schneidet oder genau durch einen der Eckpunkte verläuft. Hierfür muss zuerst erfasst werden, welche der Eckpunkte sich oberhalb der gegebenen Höhe befinden und welche darunter, bzw. genau darauf. In jedem Fall befinden sich zwei Punkte (A und B genannt) in der einen Gruppe und ein einzelner (C genannt) in der anderen. Anschließend wird für die Kanten AC und BC, mit Hilfe linearer Interpolation, der exakte Punkt berechnet an dem die Höhenlinie die Kante schneidet, wie auch in Listing 4.4 ersichtlich. *oneedge* beinhaltet hier die beiden Punkte A und B, wohingegen C in *twoedges* gespeichert ist. Wie auch im ersten Fall werden X- und Y- Koordinaten von Anfangs- und Endpunkt des Höhenliniensegments abgespeichert. Zusätzlich, wird auch die Höhe mit notiert, um die Höhenlinien zur Übersichtlichkeit in verschiedenen Farben zu zeichnen.

Listing 4.4: Fall 2

```
1  ...
2  levelCurve[levelCurveCount, 0] = k;
3  levelCurve[levelCurveCount, 1] = triangles[i, twoedges[0], 0] +
4      ((triangles[i, oneedge[0], 0] - triangles[i, twoedges[0], 0]) /
5      (triangles[i, oneedge[0], 2] - triangles[i, twoedges[0], 2])) *
6      (k - triangles[i, twoedges[0], 2]);
7  levelCurve[levelCurveCount, 2] = triangles[i, twoedges[0], 1] +
8      ((triangles[i, oneedge[0], 1] - triangles[i, twoedges[0], 1]) /
9      (triangles[i, oneedge[0], 2] - triangles[i, twoedges[0], 2])) *
10     (k - triangles[i, twoedges[0], 2]);
11 ...
12 //gleiche Vorgehensweise fuer die andere Kante
13 levelCurveCount++;
```

Mit der kompletten Liste an Schnittgeraden zwischen den Dreiecken des DGMs und den gedachten Ebenen mit bestimmter Höhe, kann das Gelände gezeichnet werden. Wenn hierbei jeder Höhe eine Farbe zugeordnet wird, erhält man das in Abbildung 4.3 ersichtliche Endergebnis.

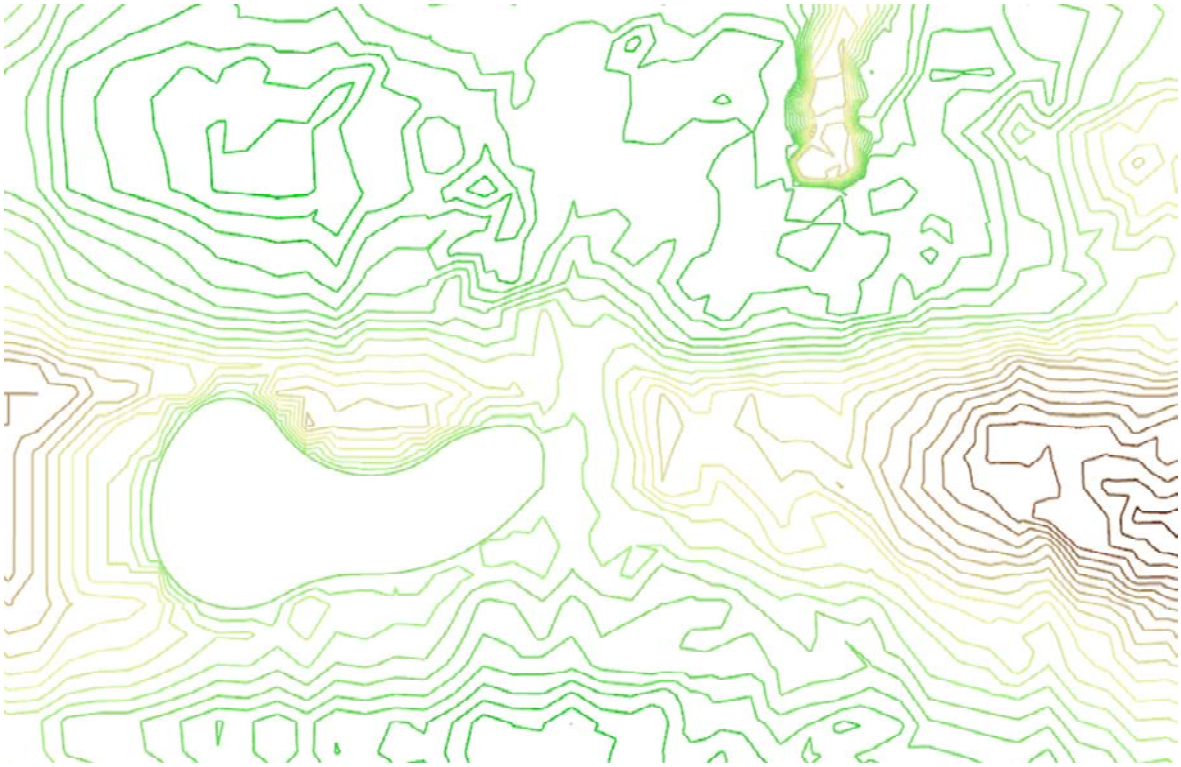


Abbildung 4.3: Darstellung der Höhenlinien auf dem interaktiven Tisch (mit dem in dieser Arbeit entworfenen Tool erstellt).

4.3 Längsprofilberechnung

4.3.1 Gesamtprozess

Um die zur Achse gehörende Gradiente entwerfen zu können, ist die Kenntnis über den Geländeverlauf entlang der Achse notwendig. Hierfür sind diverse Schritte auszuführen, damit sämtliche Informationen vorhanden sind, um das Höhenprofil korrekt wiederzugeben. Dabei wird wie folgt vorgegangen.

Zu allererst müssen für jedes Dreieck des DGMs alle vorhandenen Nachbardreiecke bestimmt werden. Durch Überprüfen auf gemeinsame Eckpunkte, kann dies ohne weiteres durchgeführt werden. Danach beginnt der eigentliche Berechnungsprozess. Für alle horizontalen Tangentenschnittpunkte wird geprüft, in welchem Dreieck sie sich befinden. Die Überprüfung findet mit der dafür erstellten Funktion *InsideTriangle* statt, die später genauer erläutert wird. Sobald das zugehörige Dreieck für den Tangentenschnittpunkt gefunden ist, kann die Z-Koordinate des Punktes, die Höhe, berechnet werden. Auch hierfür gibt es eine separate Funktion *CalculateHeight*, die später thematisiert wird. Der Code für den eben beschriebenen Vorgang kann dem Listing 4.5 entnommen werden.

Listing 4.5: Bestimmung der Dreiecke, in denen sich die Tangentenschnittpunkte befinden

```
1  double [,] height = new double[100, 2];
2  for (int i = 0; i < (numberOfPoints + 1); i++)
3  {
4      if (completeAlignmentHorizontal[i, 0] != 0 ||
           completeAlignmentHorizontal[i, 1] != 0)
5      {
6          for (int j = 0; j < triangles.GetLength(0); j++)
7          {
8              bool inside = InsideTriangle(completeAlignmentHorizontal[i,
                   0] * scaleX, completeAlignmentHorizontal[i, 1] * scaleY,
                   triangles, j);
9              if (inside)
10             {
11                 height[i, 0] = CalculateHeight(
                       completeAlignmentHorizontal[i, 0] * scaleX,
                       completeAlignmentHorizontal[i, 1] * scaleY, triangles,
                        j);
12                 height[i, 1] = j;
13             }
14         }
15     }
16 }
```

Im Anschluss daran werden Punkte zwischen den einzelnen Tangentenschnittpunkten berechnet, für die man ebenfalls die Höhe bestimmt. Ziel ist es, dass sich aufeinanderfolgende Punkte in benachbarten Dreiecken befinden. Somit müssen nämlich immer nur die Nachbardreiecke des vorherigen Punktes abgesucht werden, anstatt sämtliche Dreiecke zu durchsuchen. Innerhalb einer While-Schleife werden solange neue Punkte berechnet, bis sich entweder der neue Punkt und der folgende Tangentenschnittpunkt im gleichen Dreieck befinden oder sich der neue Punkt innerhalb einer bestimmten Schrittweite *step* befindet. Innerhalb der Schleife wird zuerst die Schrittweite auf einen konkreten Wert festgelegt und damit dann ein potentieller nächster Punkt entlang der Achse berechnet. Anschließend wird überprüft, ob sich der berechnete Punkt entweder noch im selben Dreieck befindet, wie der vorherige Punkt, in einem dem Nachbardreiecke oder in keinem von beiden. Befindet der Punkt sich immer noch im selben Dreieck wird die Schrittweite erhöht, befindet er sich in einem Nachbardreieck kann er übernommen und die Höhe berechnet werden. Trifft keiner der beiden Fälle zu wird die Schrittweite verringert (siehe Listing 4.6). Die Schrittweite wird dann solange weiter verändert bis der zweite Fall Eintritt und die Höhe berechnet werden kann. Für den Fall, dass man mit der Schrittweite immer wieder über ein Nachbardreieck hüpft, aber nie genau in das Dreieck trifft, wurde die Variable *breakWhile* eingeführt, die mitzählt, wie oft die Schrittweite verändert wurde, ohne dass ein Punkt in einem der Nachbardreiecke gefunden wurde. Bei 30

Schritten ohne Erfolg wird der beschriebene Vorgang abgebrochen. Für den nächsten Punkt wird das zugehörige Dreieck dann bestimmt, indem alle Dreiecke abgelaufen werden. Dann kann man den Punkt mit zugehöriger Höhe abspeichern und mit der gewohnten Vorgehensweise weiter machen. Sobald ein neu abgespeicherter Punkt nach oben genannten Kriterien zu nahe am folgenden Tangentenschnittpunkt liegt, wird der Tangentenschnittpunkt selbst in die *profile* Matrix eingespeichert und man beginnt schrittweise die Strecke zwischen diesem und dem darauffolgenden Tangentenschnittpunkt abzuarbeiten. All das wird solange fortgeführt bis der letzte Tangentenschnittpunkt der Achse erreicht ist. Neben den X-, Y- und Z-Koordinaten eines neuen Punktes werden auch immer die zugehörige Dreiecksnummer und der Abstand zum letzten Punkt abgespeichert, damit die Lage auf der Achse bekannt ist.

Listing 4.6: Schrittweitenanpassung

```
1  if (InsideTriangle(pointX, pointY, triangles, (int) profile[count - 1,
       4]))
2  {
3      step = step * 1.03;
4      breakWhile++;
5  }
6  else if (!insideNeightbours)
7  {
8      step = step * 0.97;
9      breakWhile++;
10 }
11 else if (insideNeightbours)
12 {
13     profile[count, 2] = pointX;
14     profile[count, 3] = pointY;
15     profile[count, 4] = triangleNumber;
16     profile[count, 0] = CalculateHeight(pointX, pointY, triangles, (int)
           profile[count, 4]);
17     inside = true;
18     count++;
19     breakWhile = 0;
20 }
```

Werden die so berechneten Punkte in ein Diagramm mit dem Streckenkilometer als X-Achse und der Höhe als Y-Achse eingetragen, erhält man beispielsweise eine Darstellung wie in Abbildung 4.4.

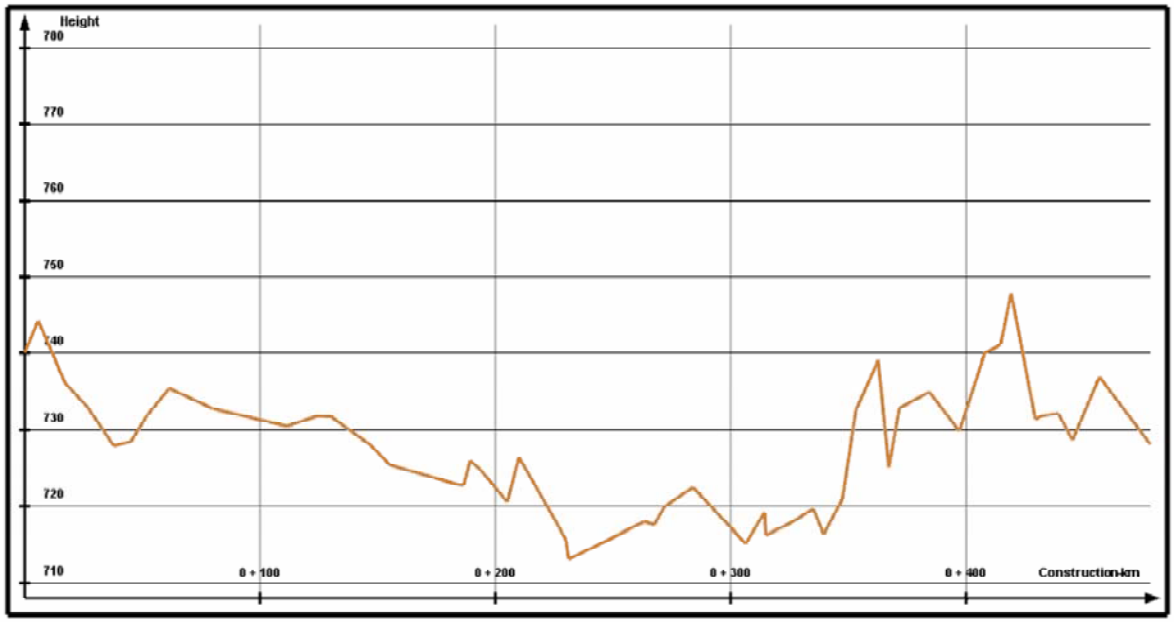


Abbildung 4.4: Beispielhafter Höhenverlauf entlang einer gegebenen Achse (mit dem in dieser Arbeit entworfenen Tool erstellt).

4.3.2 Punkt innerhalb oder außerhalb eines Dreiecks

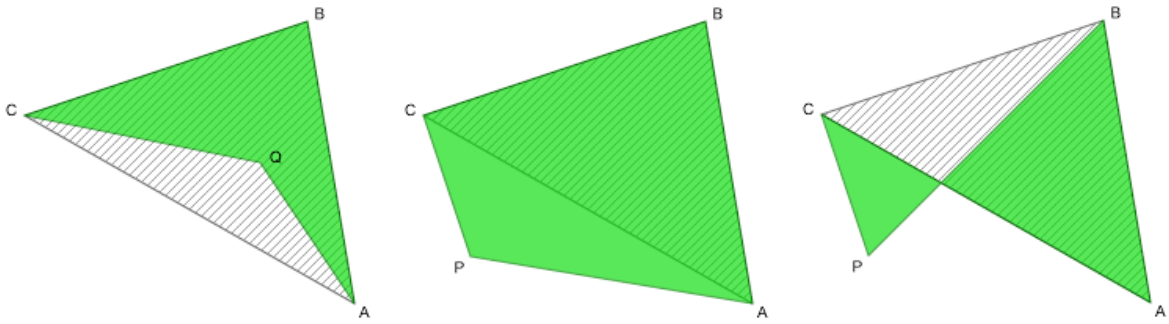
Wie bereits angekündigt wird hier die Funktion beschrieben, die feststellt, ob sich ein Punkt innerhalb eines Dreiecks befindet. Dies ist notwendig, um über Interpolation die Höhe bestimmen zu können. Diese liefert nur richtige Ergebnisse, wenn der Punkt auch innerhalb des jeweiligen Dreieckes liegt. Für eine derartige Fragestellung gibt es unterschiedliche Lösungsansätze. Der Ansatz der hier gewählt wurde, vergleicht den Flächeninhalt des Dreiecks mit dem des Vierecks, das durch die drei Eckpunkte des Dreiecks und dem vierten vorgegeben Punkt gegeben ist.

Zu Beginn wird der Flächeninhalt des vorgegebenen Dreiecks mit Hilfe der Gauß´schen Flächenformel ausgerechnet (4.1).

$$2 \cdot A = \sum_{i=1}^{n} x_i(y_{i+1} - y_{i-1}) \tag{4.1}$$

Anschließenden bildet man mit den Eckpunkten des Dreiecks A, B, C und dem Punkt P, von dem man wissen möchte, ob er sich in dem Dreieck befindet, ein Viereck. Die Fläche des Vierecks kann ebenso mit der Gauß´schen Flächenformel bestimmt werden. Nun können unterschiedliche Fälle eintreten. Wenn der Flächeninhalt des Vierecks größer ist, als der Flächeninhalt des Dreiecks, liegt P auf jeden Fall außerhalb des Dreiecks. Ist dies nicht der Fall kann es entweder sein, dass der Punkt innerhalb des Dreiecks liegt oder es sich um ein überschlagenes Dreieck handelt und P trotzdem außerhalb liegt (siehe Abbildungen 4.5). Aus

diesem Grund müssen mit den Punkten A, B, C und P alle drei möglichen Vierecke gebildet werden, indem P einmal zwischen A und B, einmal zwischen B und C und einmal zwischen C und A liegt. Sofern eines dieser Vierecke eine größere Fläche als das Dreieck hat, befindet sich P außerhalb des Dreiecks, anderenfalls darin (Janecke, 2014).



(a) Punkt innerhalb. **(b)** Punkt außerhalb. **(c)** Überschlagenes Viereck.

Abbildung 4.5: Mögliche Fälle bei der Flächenberechnung des Vierecks (Janecke, 2014).

Anhand von Listing 4.7 wird eine Teil dieses Vorgangs noch einmal kurz verdeutlicht. Hier wird zu Beginn die Reihenfolge der Punkte im Viereck *square* festgelegt. Dann folgt die Berechnung des Flächeninhalts mit der Gauß´schen Flächenformel (4.1). Für den Fall, dass die Viereckfläche *areanew* größer ist wie die Dreiecksfläche *area* wird die Variable *bigger* auf 1 gesetzt, ansonsten bleibt sie auf 0. Selbiges wird für die anderen beiden möglichen Vierecke durchgeführt. Hat *bigger* zum Schluss den Wert 1 liegt P außerhalb von ABC, hat *bigger* jedoch zum Schluss den Wert 0 liegt P innerhalb von ABC.

Listing 4.7: Flächenberechnung

```
1  var square = new double[4, 2];
2  square[0, 0] = x;
3  square[0, 1] = y;
4  square[1, 0] = triangles[number, 0, 0];
5  square[1, 1] = triangles[number, 0, 1];
6  square[2, 0] = triangles[number, 1, 0];
7  square[2, 1] = triangles[number, 1, 1];
8  square[3, 0] = triangles[number, 2, 0];
9  square[3, 1] = triangles[number, 2, 1];

11 for(int i = 0; i < 4; i++)
12 {
13     if (i != 3 && i != 0)
14         areanew += square[i, 0] * (square[i + 1, 1] - square[i - 1, 1]);
15     else if (i == 0)
16         areanew += square[i, 0] * (square[i + 1, 1] - square[3, 1]);
17     else if (i == 3)
18         areanew += square[i, 0] * (square[0, 1] - square[i - 1, 1]);
```

```
19 }
20 areanew = areanew * 0.5;
21 if (Math.Abs(areanew) > area)
22     bigger = 1;
```

4.3.3 Interpolation der Höhe

Um die Längsprofilberechnung abzuschließen, wird noch kurz auf die Höhenberechnung eingegangen. Diese wird mittels linearer Interpolation innerhalb des Dreiecks, in dem sich der gegebene Punkt P befindet, ausgerechnet. Zuerst stellt man für die Gerade der beiden Eckpunkt A und B eine Geradengleichung der Form $y = mx + b$ auf. Selbiges macht man auch für die Gerade, die durch C und P führt (mit bp und mp). Wie in Listing 4.8 zu erkennen wird dann der Schnittpunkt *intersection* der beiden Geraden berechnet. Über lineare Interpolation kann man dann auf die Höhe von P schließen.

Listing 4.8: Interpolation

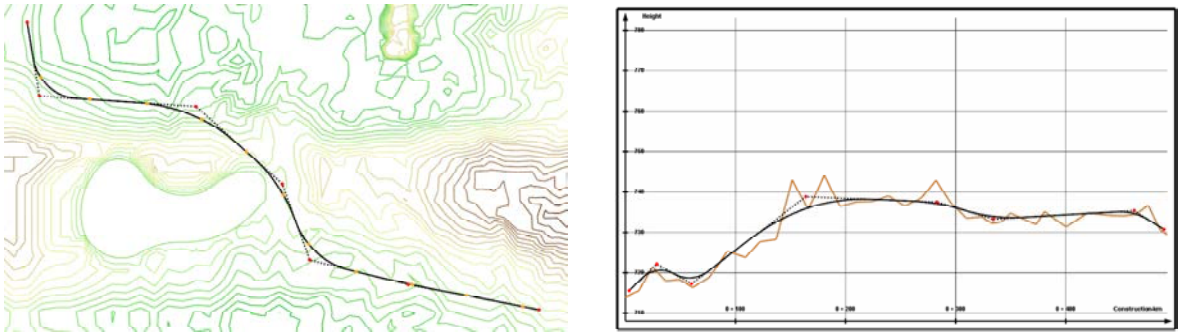
```
1  double intersectionX = (bp - b) / (m - mp);
2  double intersectionY = m * intersectionX + b;
3  double intersectionZ = triangles[number, 0, 2] + (triangles[number, 0, 2]
       - triangles[number, 1, 2]) / (triangles[number, 0, 1] - triangles[
     number, 1, 1]) * (intersectionY - triangles[number, 1, 1]);
4  double height;
5  if ((triangles[number, 2, 1] - intersectionY) == 0 || (y - intersectionY)
       == 0)
6      height = intersectionZ;
7  else
8  {
9      height = intersectionZ + (triangles[number, 2, 2] - intersectionZ) /
           (triangles[number, 2, 1] - intersectionY) * (y - intersectionY);
10 }
11 return height;
```

4.4 Achsen- und Gradientenentwurf

Die Bearbeitung der Achse und Gradiente läuft im Wesentlichen gleich ab. Beide verwenden die Darstellung mit Tangentenschnittpunkten zur geometrischen Repräsentation, da die Bearbeitung sehr einfach ist. Im Vergleich zur Segmentendarstellung müssen nur die Tangentenschnittpunkte angepasst werden und nicht jedes Segment mit seinen einzelnen Attributen. Dabei wird die Abfolge von Tangentenschnittpunkten in der Klasse *Alignment* in der Matrix *Points* abgespeichert. Möchte man also eine Linienführung ändern, stehen über

das Steuerungsfenster unterschiedliche Optionen zur Auswahl. Dazu gehören neue Punkte hinzuzufügen, bestehende Punkte zu löschen oder verschieben oder die Reihenfolge bestehender Punkte zu tauschen. Zu jeder dieser Änderungen, gibt es eine entsprechende Funktion in der Klasse *Alignment*, die die jeweilige Transformation an der *Point* Matrix vornimmt. Um beispielsweise einen neuen Tangentenschnittpunkt zu erstellen, muss man folglich den Schieberegler des Steuerungsfensters auf *AddPoint* stellen und dann an die gewünschte Stelle im Alignment Fenster drücken. Der einzige Unterschied zwischen der Erstellung der Achse und der Gradiente liegt darin, dass bei der Achse neue Punkte in der zeitlichen Reihenfolge der Klicks abgespeichert werden, wohingegen die Gradiente neue Punkte nach ihrem X-Wert sortiert und dem entsprechend in die Matrix mit den Schnittpunkten einordnet.

Aus den Koordinaten der Tangentenschnittpunkten muss dann auf die Geraden und Kreisbögen gefolgert folgert werden, damit die Linienführung auf dem interaktiven Tisch gezeichnet werden kann. Hierfür werden die Tangentenschnittpunkte zuerst mit Geraden verbunden und anschließend mit kreisförmigen Ausrundungen versehen. Für die Kreisbögen wurde dafür eine neue Klasse definiert. Diese vergleicht die Länge der beiden angrenzenden Geraden und erstellt dann ein Bogensegment, das die Hälfte der kürzen der beiden Geraden beansprucht. Damit der Kreisbogen auf der Oberfläche des Tisches gezeichnet werden kann, müssen anschließend noch einzelne Punkte in kleinem Abstand entlang des Bogens ausgerechnet werden. Die Punkte werden dann mit Geraden verbunden und ergeben bei passend gewähltem Punktabstand den Eindruck eines Kreisbogens. Über die Option *ChangeRadius* im Steuerungsfenster können im Nachhinein Kreissegmente ausgewählt und deren Radius verändert werden.



(a) Verlauf einer Achse. **(b)** Verlauf einer Gradiente.

Abbildung 4.6: Horizontaler und vertikaler Entwurf einer Linienführung auf dem interaktiven Tisch.

4.5 Zeichenprozess

Für die Darstellung verschiedener Objekte auf der Oberfläche des interaktiven Tisches wird die OpenGL verwendet. Zu Beginn werden die wichtigsten Grundfunktionen erklärt und dann anhand von Code Beispielen der allgemeine Zeichenprozess vorgestellt.

Die zwei wichtigsten und hier am häufigsten benutzten Funktionen sind *GL.Begin*() und *GL.End*(). Mit diesen kann ein neuer Zeichenprozess gestartet bzw. beendet werden. Über den Inhalt der Klammer von *Gl.Begin*() kann festgelegt werden, welche Art von Objekt man zeichnen möchte. Mit beispielsweise *PrimitiveType.LineStrip* für eine einzelnes Linienelement oder *PrimitiveType.Quads* für eine Viereck. Mit Hilfe von *Gl.Vertex*3(, ,) können dann die einzelnen Punkte angegeben werden, die mit Linien auf eine bestimmte Art und Weise verbunden werden sollen. Darüber hinaus gibt es noch mehrere Befehle mit denen verschiedene Eigenschaften verändert werden können, z.B. *GL.Color*3() um die Linienfarbe zu verändern oder *Gl.Linewidth*() um die Linienbreite einzustellen (OpenGL, 2018). Alle eben genannten Befehle können auch in Listing 4.9 wiedergefunden und beispielhaft nachvollzogen werden. In diesem Beispiel werden Tangentenschnittpunkte über gestrichelte Linien miteinander verbunden. Zur Darstellung von gekrümmten Linien werden ebenfalls gerade Strecken verwendet, allerdings mit sehr kleiner Länge, sodass die Segmente nicht sichtbar werden.

Listing 4.9: Zeichnen des Alignments (gestrichelter Teil)

```
1  GL.Color3(Color.Black);
2  GL.LineStipple(3, 0xAAAA);
3  GL.Enable(EnableCap.LineStipple);
4  if (alignment.Points[1, 0] != 0)
5  {
6      GL.Begin(PrimitiveType.LineStrip);
7      for (int i = 0; i < alignment.Points.GetLength(0); i++)
8          if (alignment.Points[i, 0] != 0)
9              GL.Vertex3(alignment.Points[i, 0], alignment.Points[i, 1], 1);
10     GL.End();
11 }
12 GL.Disable(EnableCap.LineStipple);
```

Die Benutzeroberfläche ist in verschiedene Fenster aufgeteilt. Sämtliche Objekte, die auf der Benutzeroberfläche gezeichnet werden, werden innerhalb einzelner Fenster gezeichnet. Dafür wurde die Klasse *Window* definiert, mit der man Fenster, entweder für die Darstellung einer Linienführung oder von Bedienelementen, erstellen kann. Für das hier entworfene Tool wurden drei Fenster erstellt, eines für die Achse, eines für die Gradiente und eines für Bedienelemente, wie Knöpfe und Schieberegler. Immer wenn man in ein Fenster Objekte zeichnen möchte, sollte die Funktion *StartDrawing* (siehe Listing 4.10) aufgerufen werden. Diese verschiebt den Nullpunkt des Koordinatensystems in die linke obere Ecke des jeweiligen Fensters. Indem man die Variablen *rotation* und *scale* anpasst kann das Koordinatensystem auch gedreht oder skaliert werden.

Listing 4.10: StartDrawing Funktion

```
1  GL.LineWidth(Linewidth);
2  GL.Color3(Color.Black);
```

```
3 world = Matrix4.CreateTranslation(cornerUL.X, cornerUL.Y, 0);
4 GL.MatrixMode(MatrixMode.Modelview);
5 GL.LoadMatrix(ref world);
6 GL.Rotate(rotation, 0, 0, 1);
7 GL.Scale(1, scale, 1);
```

In Listing 4.11 ist der komplette Zeichenvorgang des Fensters für die Achse mit Höhenlinien, Nordpfeil und weiteren Elementen dargestellt.

Listing 4.11: Fenster für die Achse

```
1 horizontalWindow.StartDrawing(true, world);
2 horizontalWindow.DrawTIN();
3 horizontalWindow.DrawLevelCurve();
4 horizontalWindow.DrawWindowFeatures();
5 horizontalWindow.DrawNorthPointer();
6 horizontalWindow.DrawAlignment(emode);
7 horizontalWindow.EndDrawing(world);
```

Folglich müssen Punkte, die über Fingerberührungen auf dem Tisch eingegeben werden, immer darauf überprüft werden, in welchem Fenster sie sich befinden. Anschließend werden sie auf das Fensterkoordinatensystem umgerechnet. In Abbildung 4.7 ist die komplette Benutzeroberfläche mit den drei vorher beschriebenen Fenstern ersichtlich. Dabei sind die beiden Fenster mit Linienführung mit dickeren Linien umrandet, das Kontrollfenster hingegen nicht.

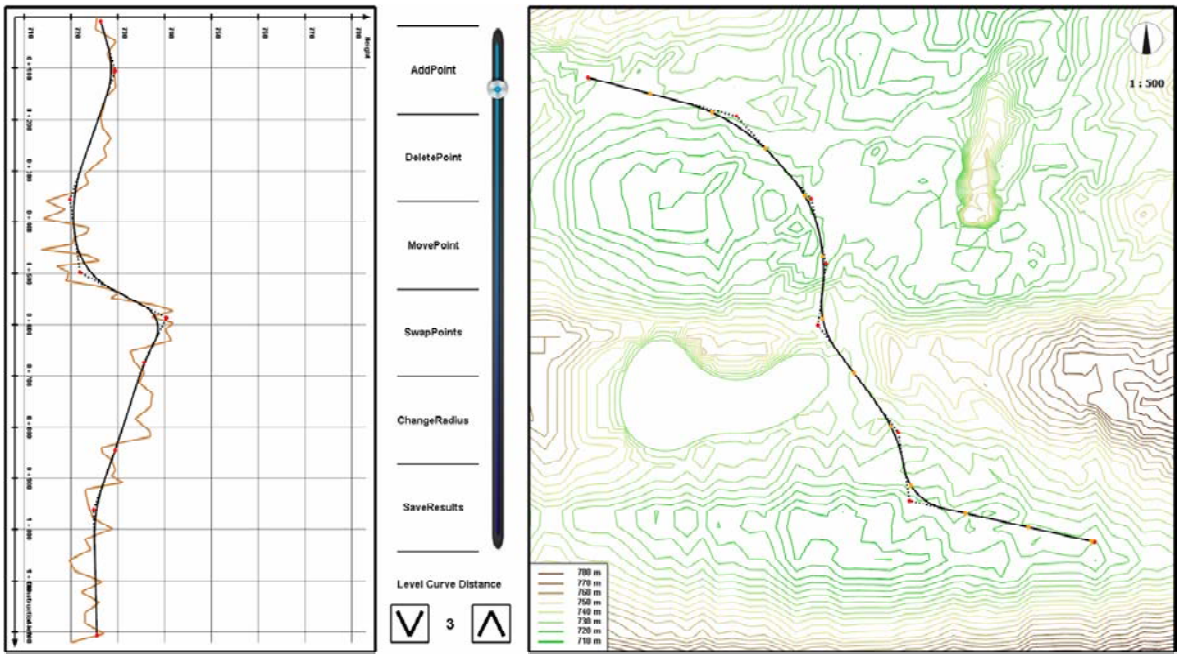


Abbildung 4.7: Benutzeroberfläche.

4.6 Ausgabe der IFC Datei

Abschließend muss die erstellte Linienführung abgespeichert werden. Da dies über das IFC Format durchgeführt wird, muss die Darstellung des Linienverlaufs mit Tangentenschnittpunkten in eine Darstellung mit einzelnen Segmenten überführt werden. Hierfür wird die IFCEngine verwendet. Im Gegensatz zum Einlesen der Geländedatei handelt es sich hier um Late Binding. Das bedeutet, dass während des Entwurfsprozesses noch nicht ersichtlich ist, ob beispielsweise verwendete Funktionen für ein bestimmtes Objekt überhaupt existieren. Derartige Fehler werden nämlich erst während der Laufzeit erkannt. Über das $IDispatchInterface$ wird während der Laufzeit zuerst die ID mit der Funktion $GetIDsOfNames$ gesucht, mit welcher dann die *Invoke* Funktion die bestimmte Funktion aufrufen kann (Microsoft, 2018; Borrmann *et al.*, 2015).

Der genaue Speicherungsprozess wird im Folgenden noch kurz anhand von Code Beispielen erläutert. Wie bereits erwähnt wird für den IFC Export die IFCEngine verwendet. Diese stellt mehrere Grundfunktionen zur Verfügung, mit welchen die Datei nach dem IFC Schema befüllt werden kann (siehe Listing 4.12). Zu Beginn wird ein neues Model erstellt, in welchem alle weiteren Schritte durchgeführt werden. Mit der Funktion $CreateInstance$ kann man eine neue Instanz erstellen, also beispielsweise ein Objekt der Klasse $IfcAlignmentCurve$. Die Attribute dieser Klasse können dann mit $PutAttr$ einzeln befüllt werden. Falls eine Attribute eine ganze Liste an Objekten benötigt muss zuerst mit $CreateAggr$ eine Aggregation für das Attribut erstellt werden. Anschließend können mit *Append* einzelne Objekte ans Ende der Liste angeheftet werden (IfcEngine, 2018).

Listing 4.12: Beim IfcExport verwendete Grundfunktionen

```
1  public Int64 CreateInstance(string instance, Int64 model)
2  {
3      Int64 newInstance;
4      newInstance = sdaiCreateInstanceBN(model, instance);
5      return newInstance;
6  }
7  public void PutAttr(Int64 instance, string attr, double value)
8  {
9      sdaiPutAttrBN(instance, System.Text.Encoding.UTF8.GetBytes(attr),
10                   (Int64)9, ref value);
11 }
12 public Int64 CreateAggr(Int64 instance, string attributeName)
13 {
14     Int64 newAggr;
15     newAggr = sdaiCreateAggrBN(instance, attributeName);
16     return newAggr;
17 }
```

```
18 public void Append(Int64 list, double value)
19 {
20     sdaiAppend(list, (Int64)9, ref value);
21 }
```

Nach dem in Kapitel 3.2.1 erläuterten Schema kann dann ein IfcAlignment2DHorizontal folgendermaßen erzeugt werden. Zuerst erstellt man eine neue Instanz IfcAlignment2DHorizontal. Da diese das Attribut *Segments* besitzt, welches eine Liste an einzelnen Linienführungselementen darstellt, muss hierfür eine neue Aggregation erstellt werden. Dann kann man zum Beispiel eine neues Geradenstück IfcLineSegment2D erzeugen, dessen Attribute wiederum mit den jeweiligen Werten befüllt werden und abschließend wird dieses Element dann mit *Append* der Segmentliste vom IfcAlignment2DHorizontal hinzugefügt (siehe Listing 4.13).

Listing 4.13: Beispiel zum IfcExport

```
1 Int64 ifcAlignment2DHor = CreateInstance("IfcAlignment2DHorizontal",
      model);
2 PutAttr(ifcAlignment2DHor, "StartDistAlong", 0.0);
3 Int64 horSegments = CreateAggr(ifcAlignment2DHor, "Segments");

5 ifcCurveSegment2D = CreateInstance("IfcLineSegment2D", model);
6 PutPoint(ifcCurveSegment2D, "StartPoint", PointX, PointY, model);
7 PutAttr(ifcCurveSegment2D, "StartDirection", startDirection);
8 PutAttr(ifcCurveSegment2D, "SegmentLength", segmentLength);
9 Append(horizontalSegments, ifcCurveSegment2D);
```

Sobald sämtliche Informationen in das Model eingetragen wurden, kann die IFC Datei abgespeichert werden und der Entwurfsprozess mit dem interaktiven Tisch ist abgeschlossen. Die erzeugte Datei kann in anschließenden Planungsschritten weiter verwendet werden, indem man zum Beispiel weitere Detaillierungen in CAD-Programmen vornimmt.

Kapitel 5

Schlussgedanke

5.1 Fazit

Zusammenfassend ist das Ergebnis dieser Arbeit eine Entwurfssoftware für den interaktiven Tisch des CDP Projekts, die im Speziellen die frühen Planungsschritte unterstützt und durch die einfache und schnelle Bedienung das Aufstellen mehrerer verschiedener Linienführungsvarianten ermöglicht.

Die Software liest das zugrunde liegende Gelände in Form einer IFC Datei ein und stellt dieses mit Höhenlinien auf dem Tisch dar. Über das Hinzufügen, Löschen und Bewegen von einzelnen Tangentenschnittpunkten kann zuerst die Achse gestaltet werden. Das Programm berechnet dann automatisch den Höhenverlauf entlang der konzipierten Achse. Mit dieser Information kann dann mit der Erstellung der Gradiente fortgefahren werden. Dies funktioniert auf gleiche Weise wie die Erstellung des horizontalen Linienverlaufs und zwar mit der Bearbeitung von Tangentenschnittpunkten. Sowohl bei der Achse, wie auch bei der Gradiente wird der Linienführungsverlauf aus Graden und Kreisbögen zusammengesetzt. Übergangsbögen sind in dem Entwurfswerkzeug hingegen noch nicht implementiert. Darüber hinaus ist aber die Funktion gegeben die Radiusgröße einzelner Kreissegmente zu verändern und so mehr Einfluss auf den genauen Linienführungsverlauf zu nehmen. Iterativ kann der horizontale oder vertikale Anteil des Alignments weiter verändert werden, bis das Endergebnis den gewünschten Anforderungen entspricht. Dabei wird bei jeder Änderung an der Achse die Gradiente selbstständig mit angepasst, also zum Beispiel gestaucht, gestreckt oder verschoben. Wenn der Bearbeitende mit der Linienführung zufrieden ist, ermöglicht die Software das Abspeichern des Alignments in einer IFC Datei. Dadurch kann das erzeugte Alignment auch noch in späteren Planungsprozessen genutzt und beispielsweise mit CAD-Programmen verfeinert werden.

5.2 Fortführungsvorschläge

Das im Rahmen dieser Bachelorarbeit erstellte Werkzeug zum Entwurf von neuen Straßen und Bahnstrecken ist noch keineswegs perfekt, sondern stellt lediglich eine erste Version mit Grundfunktionen dar. Da alles andere den Umfang dieser Arbeit sprengen würde. Deshalb gäbe es noch diverse Verbesserungs- und Erweiterungsmöglichkeiten, um den Entwurfsprozess weiter zu erleichtern und die Entscheidungsfindung zu unterstützen. Ein paar dieser Möglichkeiten werden hier abschließend erläutert.

An erster Stelle ist hier die Option von Übergangsbögen in der Linienführung zu nennen. Diese fehlen bei der momentanen Version des Tools, sind aber von grundlegender Bedeutung, unabhängig davon ob es sich um die Planung von Verkehrswegen für Autos, Zügen oder Magnetschwebebahnen handelt. Zu empfehlen wäre vor allem die Implementierung von Klothoiden, da diese nicht nur bei Straßen, sondern häufig auch bei Schienen zur Verwendung kommen. Andere Übergangsformen sind, wie in Kapitel 2.1.3 beschrieben vor allem für schienengebundene Fahrzeuge relevant.

Eine weitere sehr wichtige Ergänzung, um das Entwurfstool tatsächlich anwendbar zu machen, wäre die Integrationen weiterer Geodaten. Bisher wird alleine die Topologie des Geländes auf dem interaktiven Tisch dargestellt. Da beispielsweise eine neue Straße aber immer in irgendeiner Form an das bestehende Straßennetz anschließt, ist die Darstellung von bestehender Infrastruktur auf dem Tisch unabdingbar. Dazu gehören Schienen, Straßen, aber auch Gebäude diverser Art. Im Zuge dessen wäre genauso die Darstellung von Gewässern, Wasser- und Naturschutzgebieten im Lageplan denkbar.

Zu den Funktionen, die für das Tool nicht unbedingt notwendig wären, dem Entwerfenden aber definitiv die Entscheidungsfindung erleichtern würden, gehört die Normüberprüfung. Da es im Verkehrswegebau die unterschiedlichsten Vorschriften gibt, wie eine Straße, bzw. ein Schienenverlauf auszusehen hat, wäre es sehr hilfreich wenn unmittelbar angezeigt würde, ob bestimmte Normen eingehalten sind oder eben nicht. Beispielsweise sind aufeinanderfolgende Kurvenradien aufeinander abzustimmen. Sind diese Vorgaben nicht eingehalten könnte man einen Kurvenradius in roter Farbe zeichnen lassen oder andernfalls in grün.

Des Weiteren wären unterschiedliche Hintergrundsimulationen denkbar, zum Beispiel zur Berechnung von anfallenden Erarbeiten oder erzeugter Lärmbelastung. Bei der Lärmberechnung könnte man auch sehr gut die Funktion des interaktiven Tisches 3D Objekte auf der Tischoberfläche zu erkennen mit einbauen, da diese bisher noch nicht zur Anwendung kam. So könnte man die Lärmausbreitung ausgehend von der Linienführung berechnen und anzeigen lassen und dann mit der Platzierung von kleinen 3D Blöcken, die Lärmschutzwände repräsentieren, entstehende Veränderungen sichtbar machen.

Außerdem gäbe es weitere Möglichkeiten Gelände und Entwurfsprozess weiter zu veranschaulichen. Eine Möglichkeit wäre über den zweiten Beamer des interaktiven Tisches eine perspektivische Ansicht von Linienführung und Gelände an die Seitenwand zu projizieren. Eine zweite Möglichkeit wäre, den Entwurf mit Virtual Reality (VR) zu unterstützen, mittels eines head-mounted Display (HMD), so wie es Makanae & Matsuda (2018) vorschlagen.

Literaturverzeichnis

Allplan (2015). *BIM-Kompedium, Theorie und Praxis* (2. Aufl.). München: Allplan GmbH.

Amann, J. & Borrmann, A. (2015). Open BIM for Infrastructure - mit OKSTRA und IFC Alignment zur internationalen Standardisierung des Datenaustauschs. Forschungsbericht, Technische Universität München, Leonard Obermeyer Center.

Amann, J., Borrmann, A., Chipmann, T., Lebegue, E., Liebich, T. & Scarponcini, P. (2015). IFC Alignment Project, Conceptual Model. Forschungsbericht, buildingSMART.

Apogea, V. B. S. (2018). Qué es el BIM? http://www.apogeavirtualbuilding.com/que-es-el-bim/. Stand: 2018-09-13.

Borrmann, A., Amann, J., Chipmann, T., Hyvärinen, J., Liebich, T., Muhič, S., Mol, L., Plume, J. & Scarponcini, P. (2017). IFC Infra Overall Architecture Project, Documentation and Guidlines. Forschungsbericht, buildingSMART.

Borrmann, A., König, M., Koch, C. & Beetz, J. (2015). *Building Information Modeling, Technologische Grundlagen und industrielle Praxis*. Springer Vieweg.

buildingSmart International (2018). IfcCurveSegment2D. https://www.buildingsmart-tech.org/ifc/IFC4x1/final/html/schema/ifcgeometryresource/lexical/ifccurvesegment2d.htm. Stand: 2018-08-09.

DB Netz AG (2009). *Richtlinie 800.0110, Teil Linienführung*. Ruschestraße 104, 10365 Berlin: Deutsche Bahn Netz AG.

Egger, M., Hausknecht, K., Liebich, T. & Przybylo, J. (2013). BIM Leitfaden für Deutschland, Information und Ratgeber. Forschungsbericht, Forschungsinitiative ZukunftBAU.

Freudenstein, S. (2017a). Verkehrswegebau Grundmodul Übung. Technische Universität München, Lehrstuhl Verkehrswegebau.

Freudenstein, S. (2017b). Verkehrswegebau Grundmodul Vorlesung. Technische Universität München, Lehrstuhl Verkehrswegebau.

IfcEngine (2018). IfcEngine Documentation. http://rdf.bg/ifcdoc/. Stand: 2018-10-27.

Janecke, M. (2014, Dezember). Liegt der Punkt im Dreieck? https://prlbr.de/2014/liegt-der-punkt-im-dreieck/. Stand: 2018-09-13.

Liebich, T. (2015). IFC Alignment Project, IFC Extension Development. Forschungsbericht, buildingSMART.

Makanae, K. & Matsuda, H. (2018). Application of the Simplified HMD-Based VR System of Road Design Process. In: *Proceeding of the 17th International Conference on Computing in Civil and Building Engineering*, Tampere, Finland.

Markič, Š. (2017, Januar). Vertical Alignment Optimization using Genetic Algorithm and simulated Annealing. Master thesis, Technische Universität München.

Markič, Š., Schlenger, J. & Bratoev, I. (2018). Tangible Alignment Design. In: *Forum Bauinformatik 2018*, Weimar.

Martin-Luther-Universität (2018). Glossar, Digitales Höhen- und Geländemodell (DHM/DGM). https://http://mars.geographie.uni-halle.de/mlucampus/geoglossar/terme_datenblatt.php?terme=Digitales%20H%F6hen-%20und%20Gel%E4ndemodell%20(DHM/DGM). Stand: 2018-08-14.

Matthews, V. (2010). *Bahnbau*. Vieweg+Teubner Verlag.

Microsoft, S. (2018). Using early binding and late binding in Automation. https://support.microsoft.com/en-us/help/245115/using-early-binding-and-late-binding-in-automation. Stand: 2018-09-10.

National BIM Standard (2018). Frequently asked Questions - What is BIM? http://nationalbimstandard.org/faqs#faq1. Stand: 2018-07-31.

Niedersächsische Landesbehörde, S. u. V. (2018). Von der Planung zum Bau: Verfahrensablauf bei Bundesverkehrsstraßen. http://strassenbau.niedersachsen.de/projekte/verfahrensablauf/. Stand: 2018-07-31.

Nottbeck, A. B. (2016, Juli). *Untersuchungen zu Auswirkungen von Geschwindigkeitserhöhungen auf Bahnstrecken im Bestand*. Dissertation, Technische Universität München, München.

OpenGL (2018). OpenGL API Documentation. http://docs.gl/. Stand: 2018-10-29.

Pfeifer, W. (2003). *Etymologisches Wörterbuch des Deutschen* (6. Aufl.). München: DTV Verlag.

Queensland Government, D. o. T. & Roads, M. (2018). Road planning and design manual - 1st edition, Chapter 12: Vertical Alignment. https://tmr.qld.gov.au/. Stand: 2018-07-31.

RAA (2008). *Richtlinien für die Anlage von Autobahnen.* Wesselinger Str. 17, 50999 Köln: Forschungsgesellschaft für Straßen- und Verkehrswesen, FGSV Ver1ag.

RAL (2012). *Richtlinien für die Anlage von Landstraßen.* Wesselinger Str. 17, 50999 Köln: Forschungsgesellschaft für Straßen- und Verkehrswesen, FGSV Ver1ag.

Schubert, G. (2012, Januar). Early Design Support: Interaktive Simulationen in frühen Entwurfsphasen. Forschungsbericht, Technische Universität München, Lehrstuhl Architekturinformatik.

Steinmann, R. (2018, April). Die Zuverlässigkeit steigt - Ist BIM der Baustandard der Zukunft? *Technik in Bayern.*

xBIM (2018). xBIM Toolkit. http://docs.xbim.net/. Stand: 2018-09-10.

Abbildungsverzeichnis

Tabellenverzeichnis

Anhang A

C# Programm Code

Der vollständige Code, der im Rahmen dieser Arbeit erstellt wurde kann auf Gitlab im CDP Projekt der Fakultät Architektur, Lehrstuhl Bauinformatik unter dem Plugin Interactive Alignment Design gefunden werden (https://gitlab.lrz.de/cdp/cdp/tree/feature/plugin_interactiveAlignmentDesign/Plugins/Plugin_InteractiveAlignmentDesign).

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Bachelor-Thesis selbstständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.

Ich versichere außerdem, dass die vorliegende Arbeit noch nicht einem anderen Prüfungsverfahren zugrunde gelegen hat.

Gilching, 15. Oktober 2018

Jonas Schlenger

Jonas Schlenger
[redacted]