DIPLOMARBEIT

Modellierung von Straßen für Echtzeitvisualisierung

ausgeführt am Institut für

COMPUTERGRAPHIK UND ALGORITHMEN 186

der Technischen Universität Wien

unter Anleitung von

Prof. Dipl.-Ing. Dr.techn. Werner Purgathofer

und

Univ.-Ass. Dipl.-Ing. Dr.techn. Dieter Schmalstieg

als verantwortlich mitwirkendem Universitätsassistenten

durch

HUMMEL Gerald

PaulHeyseg. 15/14
A-1110 Wien

Zusammenfassung

Diese Diplomarbeit ist die theoretische Abhandlung von *VRMG*, einem Softwareprojekt, das im Rahmen des praktischen Teiles der Arbeit realisiert wurde und die Modellierung von Terrain und Straßen virtueller Städte zum Zwecke der Echzeit-Visualisierung bewerkstelligt. Ausgehend von den Vermessungsdaten in Form von geografischen Höhenwerten und dem Straßennetz, das als Straßengraph vorliegt, werden alle Schritte zur Umsetzung dieser Rohdaten in Geometriedaten für die separate Modellerstellung des Terrains und der Straßenzüge behandelt. Der Schwerpunkt der Arbeit liegt auf dem Anwendungsfall – Stadtstraßen. Somit umfaßt *VRMG* zahlreiche Elemente für den Aufbau von Straßen, wie sie typischerweise im Stadtbereich zu finden sind. Dadurch ist es möglich, zahlreiche Straßen-Konfigurationen aus der Realität nachzubilden. Das Terrain wird als konventioneller, Regulärer-Raster erfaßt. Nach der Berechnung der Straßen- bzw. Höhenfeldgeometrie, die auf Grundlage von Beschreibungsfiles zur Eingabe der spezifischen Parameter erfolgt, liegt das Ergebnis dieses Schrittes in polygonaler Repräsentationsform vor. Dadurch, daß die Modell-Geometrie vorzugsweise für Echtzeitanwendungen eingesetzt wird, ist die Aufbereitung der polygonalen Repräsentation durch Nachfolgeoperationen zur Optimierung der Geometrie bedeutsam. Durch die Verfahren der Delaunay-Triangulierung und des Triangle-Stripping werden die Polygone mittels Dreieckszerlegung tesseliert und dadurch in eine Form gebracht, die die derzeitigen 3D-Hardwarebeschleunigerchips unterstützen. Das endgültige Resultat wird als Szenegraph, der als Baumstruktur die Geometrie- und Texturierungdaten in hierarchischer Form aufnimmt, gespeichert und für die Verwendung, etwa als Simulationsumgebung, zur Verfügung gestellt.

Abstract

This thesis is a theoretical description of *VRMG*, a software-project, which was developed as the practical part of this thesis, enabling the modeling of the terrain and the roads of virtual cities for Real-Time visualization. Based on surveying-data of altitude and elevation of the terrain's grid together with roadnetwork-data, all steps toward the final model are covered. Emphasizing city-roads, *VRMG* contains most elements for generating road-variations found in this area. The terrain is modeled on the basis of a conventional, regular grid. As a result of computation of the terrain's and road's geometry, in dependence of parametric description files for the data-input, a polygonal representation of these two units is generated. Since the geometry-data should be able to be used in a Real-Time simulation or visualization framework, further postprocessing operations for optimizations of the polygonal geometry are necessary. These opeartions include Delaunay-triangulation and Triangle-stripping for dividing polygons through tesselation to achieve compatibility with nowaday's hardware chips for 3D-graphics acceleration. The final result is stored as scene-graph, which stores all vertex- and texture-data in a hierarchical tree structure, and is ready to be used for further usage, as environment for simulations for instance.

Inhaltsverzeichnis

1.) Einleitung:

Gegenstand der vorliegenden Arbeit ist die Modellierung der *Grundfläche(Terrain)* und des *Straßennetzes* großer, virtueller Stadteile für *Echtzeit-Visualisierung.* In den Bereich der Echtzeit-Visualisierung fallen alle Systeme, die durch Methoden der Computergrafik Daten in anschaulicher, dreidimensionaler Form auf dem Bildschirm bringen und durch hohe Bildwiederholraten eine interaktive Navigation bzw. Manipulation in der virtuellen Szene erlauben.

Grundsätzlich existieren verschiedene Möglichkeiten, um solche komplexe Modellierungs-aufgaben zu bewältigen, angefangen von der direkten Ausmodellierung der gesamten Stadt mit einem 3D-Modeler bis hin zur Berechnung der Stadtgeometrie durch Auswertung von Luftbildaufnahmen. Welche Methode gewählt wird hängt in erster Linie von der Aufgabenstellung ab. Unabhängig von der Art der Modellierung besteht bei allen Lösungen, bei denen eine interaktive Komponente integriert wird, ein grundsätzlicher Trade-Off zwischen Detailreichtum und akzeptabler Darstellungs-Geschwindigkeit.

Zu den *Anwendungsbereichen* eines Modellierungssystems für virtuelle Städte zählen:

- **Architektur-Walkthroughs:** Durch authentische Nachbildung von Gebäuden aller Art also auch jener, die es nicht mehr oder noch nicht gibt, lassen sich diese von allen Perspektiven betrachten und durchstreifen.

- **Fahrsimulatoren:** Durch Steuerung eines virtuellen Fahrzeuges können im Rahmen einer Ausbildung für Kraftfahrer verschiedene Verkehrsszenarien gefahrlos geübt werden.

- **Verkehrssimulation**: In der Verkehrsplanung wird das Zusammenwirken von vielen Verkehrsteilnehmern wie Pkw‘s, Radfahrern, Fußgängern usw. wird durch eine Computersimulation evaluiert und angewandt.

- **Computerspiele:** Um den Wünschen nach immer ansprechenderen (3D-)Kulissen für Videospiele gerecht zu werden, ist die Verwendung eines 3D-Modelers für Gebäude, Straßen usw. für die Entwickler zur gängigen Praxis geworden. Der flüssigen Spielverlauf ist hier wichtiger Realismus.

- **Virtual-Reality:** Für die Ausgestaltung der virtuellen Welt, die vom Benutzer in Form von HMD’s(Head-Mounted-Display, „Datenbrille“) gesehen und durch Steuerungs-mechanismen (Joystick, „Datenhandschuh“ etc.) beeinflußt wird, werden in der Praxis 3D-Modeler mit Geometrie-Optimierung zum Erzielen echtzeitfähiger Darstellungs-geschwindigkeit benötigt.

- **Stadtplanung:** Systeme für diesen Anwendungsbereich erlauben neben der 3D-Ansicht der Stadt eine interaktive Manipulation von Objekten(entfernen, hinzufügen, verändern). Dadurch können verschiedene Gestaltungsvarianten noch vor der realen Umsetzung beurteilt werden.

VRMG wurde im Rahmen des Forschungsprojektes *URBANVIZ*, welches die Modellierung und Visualisierung von Städten umfaßt, entwickelt. Allerdings kann *VRMG* unter Berücksichtigung gewisser Einschränkungen, auf die in Folgekapiteln noch näher eingegangen wird, für alle diese Anwendungsbereiche verwendet werden, um ein Höhenfeld- und Straßenmodell zu erzeugen.

Der Ansatz, welcher im Rahmen von *URBANVIZ* für die Modellierung der Stadt im allgemeinen gewählt wurde, basiert auf der Zerlegung der Stadt in Teilkomponenten, die zunächst separat behandelt und nach der Erstellung als fertige Teile zusammengefaßt werden. Die Unterteilung der Stadt erfolgt in folgende strukturellen Einheiten:

a) Terrain (Höhenfeld)
b) Gebäude (Häuserfronten)
c) Straßennetz (Straßen, Kreuzungen, Gehwege)
d) Freiflächen (Parks, unbebautes Gebiet)
e) Pflanzen (Bäume, Sträucher)
f) Objekte (Ampeln, Verkehrszeichen)

Für alle diese Teilkomponenten wurde eine oberflächenbasierte (polygonale-) Repräsentation gewählt, die durch ein *parametrisches Modell*, das die jeweilige Ausprägung eines Objektes durch eine Menge von Parametern beschreibt, realisiert werden kann. Die Anforderungen an ein Modell in diesem Zusammenhang sind einerseits eine optimale Anpassung an unterschiedliche Gegebenheiten und andererseits ein hierarchischer Aufbau durch Substrukturen.

Die *Informationsgrundlage* für die Modellierung einer konkreten Stadt bilden in erster Linie GIS-Daten(GIS steht für Geographische Informations-Systeme, diese Daten beihalten Geometrie-(Vermessungs-) daten, die durch eine Datenbank verwaltet werden. Es besteht die zusätzliche Möglichkeit, für alle erfaßten Objekte bestimmte Attribute tabellarisch zu erfassen). Die Akquisition dieser Daten erfolgt in der Praxis durch Landschaftsvermessung und photogrammetrische Methoden. Die zur Modellerstellung relevante (minimal-) Information besteht aus einem Höhenwerte-Raster und der Lage von Häuserkanten. Um authentischen Detailreichtum zu erreichen muß diese Grundinformation noch erheblich erweitert werden, etwa durch Anfertigung von entsprechenden Detailfotos. Das Resultat des Modellierungsvorganges ist die polygonale Geometrie der Teilkomponenten.

Zusammenfassend betrachtet besteht das Ziel des Projektes in einem Software-Tool für die Umsetzung von GIS-Daten in Geometrie-Daten, die den Anforderungen für Echtzeit-Visualisierung entsprechen. Gegenstand der vorliegenden Arbeit sind die Stadt-Komponenten „Terrain“ und „Straßennetz“, welche von *VRMG* geometrisch modelliert werden.

Das Problem ist durch große Datenmengen gekennzeichnet, daher ist die Beachtung folgender Faktoren bedeutsam:

- Automatisierung von Verarbeitungsschritten
- universelle Anwendbarkeit auf verschiedene Städtedaten(Flexibilität)
- Kapselung der Verarbeitungseinheiten(Modularer Aufbau)
- Optimierung der erstellten Geometriedaten

Das Erreichen dieser Konzepte kann durch zeitliche Gliederung in
folgende Verarbeitungseinheiten realisiert werden:

1.) Aufbereitung der Eingangsdaten:

Die digitalen Daten für das Höhenfeld, die Gebäudegrundrisse und das Straßennetz werden aus der GIS-Datenbank entnommen.

2.) Erstellung der Straßeninformation:

Ausgehend von dem Straßennetz soll eine Straße automatisch generiert werden. Grundlage für die Straßengenerierung ist ein Straßengraph, der den Verlauf der Straßen festlegt. Falls dieser nicht direkt von GIS zur Verfügung gestellt wird, soll eine Möglichkeit bestehen, einen hypothetischen Straßengraph aus der Lage der Gebäudegrundrisse zu errechnen. Die Kreuzungen entstehen nun durch sich schneidende Straßensegmente. Damit ist allerdings nur der Verlauf und die räumliche Lage bekannt, weitere Information ist nötig, um das Aussehen der Straße zu charakterisieren: Anordnung und Breite der Fahrstreifen, Aussehen der Bodenmarkierung, Verengungen und Verbreiterungen usw. Diese Eigenschaften (Parameter) werden durch das parametergesteuerte Modell erfaßt, aus dem *VRMG* das genaue geometrische Modell erzeugt.

3.) Erstellen des Höhenfeldes:

Das Höhenfeld beinhaltet einerseits die lokale Höheninformation an den Rasterschnitt-punkten für das Terrain, andererseits ist es Grundlage für die Triangulierung von Restflächen(alle Flächen, die durch keine Modellierungs-komponente erfaßt werden, wie Grünflächen, Berge, Zwischenräume zwischen Straßen und Häusern usw.). Dadurch, daß das Straßenmodell flächenbasiert(2.5D) ist, bildet das Höhenfeld die Berechnungsgrundlage für die geo-graphische Höhe der Bestandteile des Straßennetzes.

4.) Erstellen und Optimieren der Geometrie:

Außer der optischen Qualität des erstellten geometrischen Modells ist in der Optimierungsaspekt wesentlich, um der optimalen Unterstützung der Grafikhardware gerecht zu werden, sodaß hohe Bildwiederholraten erzielt werden können. Die polygonalen Ausgabedaten sollen an die Fähigkeiten der 3D-Beschleuniger Hardware angepaßt werden und von der Struktur her den Nachfolgeoperationen, die das Rendern der Szene bewerkstelligen, entgegenkommen. Verschiedene Optionen zur Nachoptimierung der Geometriedaten sollen vorhanden sein.

2.) Der Stand der Technik:

Grundsätzlich befassen sich zwei Forschungsrichtungen mit der Modellierungsproblematik von Straßen und Terrains:

- Simulatorentwickler:

Forschungsgruppen dieser Kategorie befassen sich
primär mit der logischen Seite des Straßennetzes, um Verkehrsszenarien zu simulieren. Für die Modellierung der Stadtkomponenten wird entweder ein kommerzielles Entwicklungswerkzeug verwendet, oder ein eigener 3D-Modeler entwickelt, der den spezifischen Anforderungen des Gesamtprojektes angepaßt ist. An einem Simulatorprojekt sind häufig Experten aus verschiedenen Sparten(Computergrafik, Robotik, Artificial Intelligence) beteiligt. Wesentliche Anforderungen an ein solches Modell sind die Bewältigung erheblicher Datenmengen in Echtzeit für Interaktion zur Laufzeit des Systems. Möglichkeiten des konzeptuellen Aufbaus werden im Abschnitt 2.1. anhand von Beispielprojekten aufgezeigt.

- **Verkehrsplaner, Bauingenieure:**
  Im Gegensatz zur Simulatorentwicklung steht hier die Planung von Bauprojekten im Vordergrund. Das bedeutet, daß sich das Modell stark an den realen Gegebenheiten und Konventionen orientiert und in der Lage sein muß, Planungselemente wie Terrainlängen- und Terrainquerprofile, Geschwindigkeitsdiagramme, Sichtweitendiagramme, Querprofile der Straße, Massenverteilungspläne etc. durch Software zu automatisieren [5]. Dadurch hat sich auch die Arbeitsmethode im Laufe der letzten Zeit gewandelt. Während in der Vergangenheit graphische Planung und Berechnungen mit Hilfe von Tafelwerken den Arbeitsablauf bestimmten, sind durch den Einsatz von Computerprogrammen neue Möglichkeiten erschlossen worden, etwa die Berechnung der Klothoidenwerte durch Reihenentwicklung oder die rasche Bestimmung von Schleppkurven für einen beliebigen Fahrbahnverlauf[6]. Ergänzt werden diese Verbesserungen im Gesamtkontext des Planungswesens, wozu das Bereitstellen von digitalen Terraindaten (DTM), Profilaufnahmen durch photogrammetrische und tachymetrische Methoden auf der Eingabe-Seite einerseits und ansprechende graphische Ausgabe andererseits zählen. Für diese Zwecke werden kommerzielle Produkte verwendet, wie in Abschnitt 2.2. beschrieben.

2.1.) Wissenschaftliche Forschung:

Der Hauptteil der Forschungsarbeit dieses Bereiches konzentriert sich auf die Erstellung von *Fahrzeugsimulatoren*, sodaß die Modellerstellung von Straßen, Gebäuden und so weiter nur ein Teilbereich ist und die Funktionalität im Gesamtkontext zu betrachten ist, das heißt der Schwerpunkt liegt auf *Interaktion* zwischen bewegten Objekten(Fahrzeuge, Fußgänger) und festen Objekten(Häuser, Straßen). Es besteht also die grundsätzliche Notwendigkeit, die Daten so zu strukturieren, daß zur Laufzeit der Simulation rasch auf Information aus der Datenbank zugegriffen werden kann. In der Praxis wird deshalb typischerweise ein *Mehrschichtenmodell* eingesetzt, um geometrische, topologische und semantische Aspekte zu kombinieren.

Bereits im Anfangsstadium dieses Forschungsgebietes vor etwa 20 Jahren findet man bei älteren Simulatoren, wie *TRAF-NETSIM*[26], folgende Grundkonzepte:

- *Modularisierung:* Es handelt sich hier um ein zweigeteiltes System mit *Eingabemodul*, bestehend aus einem eigenen Editor und Preprozessor für die Datenerfassung in Form eines eigenen Eingabefiles, sowie einem *Verarbeitungs- oder Simulationsmodul* für die Evaluierung der Simulation und dem grafischen Feedback.

- *Straßengraph:* ein Straßennetz besteht aus *Kanten*(gerichtete Strecken oder Segmente) und *Knoten*(Verbindungen, an denen sich Segmente teilen oder verbinden).

Die Möglichkeiten der damaligen Simulatoren bestanden in der Auswertung von realen(kritischen) Verkehrssituationen, die durch *semantische Regeln*(Ampeln, Verkehrszeichen) und *stochastische Methoden*(induziertes Verkehrsaufkommen) nachgebildet wurden. Die Grenzen waren durch limitierte Hardwarefähigkeiten sehr begrenzt, sodaß für komplexere Simulationen auf die Grafikausgabe verzichtet werden mußte und das Ergebnis nur in Form einer quantitativen, statistischen Auswertung vorlag.

Untersucht man die Parallelen von gegenwärtigen Simulatoren[7,8,12,13], so läßt sich ein Zusammenwirken von 3 Modellen, die auch als Schichten bezeichnet werden, erkennen:

- **Netzwerkmodell(geometrische Schicht)**:
  Straßen bestehen aus Kanten, Knoten und Einmündungen. Kanten entsprechen Segmenten, die aus einem oder mehreren Fahrstreifen bestehen und über Knoten oder Einmündungen mit anderen Fahrstreifen verbunden sind. Mögliche Attribute für Segmente sind die Anordnung der Fahrstreifen, die Richtung für den Verkehr, die zugelassene Höchstgeschwindigkeit, die Abmessungen von Länge und evtl. Breite sowie die Bodenmarkierungen.

- **Verkehrsmodell(topologische Schicht):**
  Dieses legt den möglichen Bewegungsraum für die Verkehrsteilnehmer fest und hat das Netzwerkmodell als räumliche Grundlage. Die Datenstruktur der Wahl ist in diesem Fall ein gerichteter Graph, auf dem sich die Fahrzeuge quasi wie auf einer Schiene fortbewegen. Analytisch anspruchsvollere Modelle sehen für die Bewegung mehr Freiheitsgrade vor, sodaß ein Fahrzeug nicht unbedingt auf der Fahrbahn bleiben, sondern von dieser auch abkommen kann.

- **Verkehrskontrollmodell(semantische Schicht):**
  Zur Regulierung des Verkehrs durch Kontrollmechanismen wie Ampeln oder Verkehrszeichen wird eine Semantik definiert. Im Zusammenwirken mit den beiden anderen Modellen, besteht die Hauptfunktion des Kontrollmodells in der Laufzeitauswertung von Entscheidungsanfragen der Verkehrsteilnehmer.

Diese mehrschichtige Struktur läßt sich u.a. bei folgenden Simulatorprojekten feststellen:

VUEMS – Virtual Urban Environment Modeling System (Donikian)[12,13],
EDF – Environment Description Framework (Willemsen)[7],
SIRCA – SImulador Reactivo de Conduccion de Automóviles(Bayarri)[8],
AIMSUN2 - Advanced Interactive Microscopic Simulator for Urban and Non-Urban Networks (J. Barceló)[28,29,30]

3D-Modeler:

In vielen Fällen wurde im Rahmen des Simulatorprojektes ein eigener *3D-Modeler*[12,13] entwickelt, in dem die Daten für die 3 Schichtenmodelle editiert werden. Der typische Aufbau eines solchen Entwicklungswerkzeuges ist folgender:

Die *Eingabestufe* baut auf drei Informationsquellen auf:

1) Kartographische Datenbank, in der Vermessungsdaten der Stadtplaner mit Objekt-klassifizierender Baumstruktur, in der Leitungsnetze von Gas, Wasser und Strom, aber auch Information über Bäume, Lampen etc. enthalten sind.

2) Eingescannter Plan, welcher die fehlende Information ergänzen soll, dabei handelt es sich etwa um Straßendetails wie Verkehrszeichen oder Bodenmarkierungen.

3) Beschreibung der Ampelregelung mit Position der Ampeln und der lokalen- und globalen Synchronisation dieser untereinander.

Die *Modelingstufe* ist als interaktive Graphic-Design Oberfläche gestaltet, es wird die Generierung von folgenden drei Arten von Objekten unterstützt:

1) Das *Straßennetz* ist von Fall zu Fall unterschiedlich repräsentiert. Einige Varianten sind:

SHIVA - Simulated Highways for Intelligent Vehicle Algorithms(Rahul Sukthankar)[24,25]:
Straßen werden durch ein verbundenes Netz von Segmenten definiert, wobei ein Segment eine beliebige Krümmung und eine feste Anzahl von Fahrstreifen aufweist. Zusätzliche Attribute beziehen sich auf die für dieses Segment vorgeschriebene Höchstgeschwindigkeit und die Bodenmarkierung. Die Verbindung der aufeinanderfolgenden Segmente erfolgt für jeden Fahrstreifen getrennt, um Straßenteilung und Straßenzusammenführung zu ermöglichen. Die Segmentgeometrie wird durch Querschnitte beschrieben, diese liegen normal auf die Tangente im Stützpunkt, welcher auf der linken Straßenkante liegt. Ausgehend vom Stützpunkt wird die Position der Fahrstreifen durch die Entfernung vom linken Straßenrand spezifiziert.

VUEMS – Virtual Urban Environment Modeling System (Donikian)[12,13]:
Die Straße wird durch die Achse(Kubischer Spline) und Symbole(Parameter für die Spezifikation der Fahrstreifen) modelliert.
Die Berechnung von Kreuzungen erfolgt automatisch bei Überschneidung zweier Straßen. Das Straßennetz wird aus folgenden Objektklassen zusammengesetzt:

Normale Straßenstücke(Gerade, Erweiterung, Verengung)
Verbindungsstücke(Gabelung, Teilung, Verbindung, Zusammenführung)
Kreuzungsstücke(T-, Y-, X-Form: Tunnel, Brücken, Überkreuzung).

Zusätzlich ist die Information der Fahrstreifen durch eine eigene Grammatik codiert:

```
C = [LI]*L‘!‘[IL]* | ‘!‘[LI]*L
C = [LI]*L‘!‘[IL]* | ‘!‘[LI]*L     (Querschnitt, ‘!‘ ist Lage der Achse)
I = [‘:‘ | ‘|‘ | ‘-‘]              (unterbrochene/durchgezogene/keine
                                   Mittellinie)
L = [‘N‘ | ‘R‘]*T width*           (neue/wegfallende Spur, Streifenbreite)
T = [`D`|`O`|`DO`|`DL`| usw.]     (Fahrtrichtung für Streifen)
```

Abbildung-1 zeigt im folgenden die Form und den Aufbau der elementaren Straßenteile mit den entsprechenden Konfigurationen für die Fahrtrichtung:

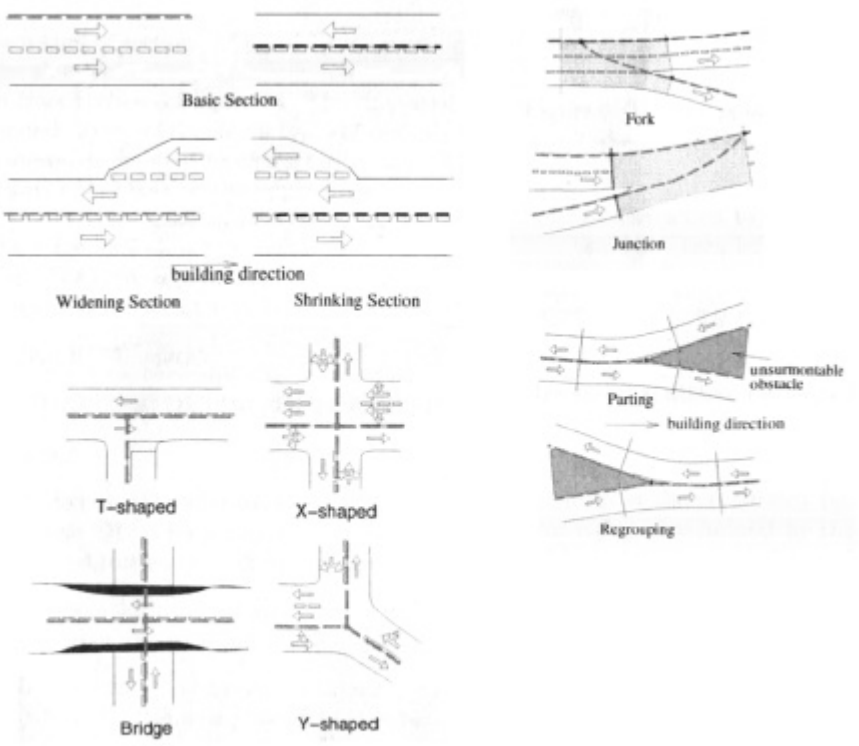


Abbildung 1: Komponenten des Straßenmodells bei VUEMS: Segmentteile(links oben) mit möglicher Änderung der Fahrspuranzahl, Teilungsstücke(rechts) mit variierender Anordnung der Fahrtrichtung, Kreuzungsstücke(links unten) mit Einmündug, X- und Y-Formen und einer Brücke.

EDF – Environment Description Framework (Willemsen)[7]:
Spezifikation der Achse durch eine 3D-Splinekurve. Die Position eines Punktes auf der Straßenkurve wird durch ein 2D-Referenzkoordinatensystem beschrieben. Und zwar mit der Raumkurve als Hauptachse und der Nebenachse (= Normalvektor auf die Hauptachse). Die loaklen Koordinaten sind ein Parameterpaar: (d, s), wobei d der Kurvenlänge auf der Hauptachse und s dem Normalabstand von der Kurve entspricht. Das Vorzeichen von s ist für die Seite links der Achse positiv, rechts negativ. Die Oberfläche der Straße wird durch sogenannte *Segmente* beschrieben, deren Parameter die Breite links und rechts von der Achse, die Krümmung der Achse, die Tangenten, die Länge und die Oberflächennormale sind. Aufgrund dieser Eigenschaften wären alle denkbaren Straßenverläufe möglich, wie Loopings oder schraubenförmige Verwindungen. Die Verbindung einzelner Segmente zu einem Straßenzug erfolgt durch eine Baumstruktur. Die 3 Grundformen der Segmente sind *Gerade, Kurve* und *Spirale*. Jedes Segment kann durch 2 (Anfangs- und Endpunkt) oder 3 Punkte definiert sein. Im ersteren Fall ändern sich die Koordinaten linear, im zweiten Fall parabolisch, zusätzlich besteht die Option, eine Querneigung anzugeben. Spiralförmige Segmente können als Übergangsbogen verwendet werden, wobei die Anpassung von Gerade an Kreis oder umgekehrt durch Interpolation erfolgt. Zusätzlich zu den 3 Grundformen können beliebige Segmentverläufe auch durch Spline-

Interpolation von Stützstellen realisiert werden. Die Spezifikation der Fahrstreifen sieht vor, daß diese erstens nebeneinander liegen mit Anordnung parallel zur Mittelachse und außerdem für die Länge eines Segments homogen in den Querschnittsbreiten sind. Die Fahrstreifen sind in Klassen, je nach Verwendungsart(Pkw, Fußgänger usw.), unterteilt. Das Labeling der Fahrstreifen umfaßt neben dem Typ auch die Bewegungsrichtung für den Verkehr. EDF-Kreuzungen sind eigenständige Objekte und werden geometrisch durch ein konvexes Polygon beschrieben, das aus den Querschnitten der zuführenden Straßen gebildet wird.

2) *Verkehrszeichen und Gegenstände* können bei diesen fensterorientierten Benutzer-oberflächen aus einem Menü ausgewählt und durch Angabe der 2D-Position placiert werden, die Verkehrszeichen werden einer Position auf der Straßenachse zugeordnet und deren Parameter ebenfalls durch Symbole codiert.

3) *Gebäude, Parks, Plätze* sind neben den Straßen als zusätzliche Elemente der Stadt durch 2D-Polygone repräsentiert und werden durch eine Liste von Attributen beschrieben.

Schließlich wird die Information in der *Ausgabestufe* im allgemeinen in zwei Einheiten zur Verfügung gestellt:

1) Ein 3D-Szenemodell in Grafikfile-Format
2) eine objektorientierte Datenbank, auf die bei der Animation zur Simulation des Verkehrs und der virtuellen Verkehrsteilnehmer interaktiv zurückgegriffen wird.

Eingabe durch ein File:

Als Alternative zum 3D-Modeler kann die Eingabe der Netzwerkdaten für die Simulation in Form eines *Files* mittels *Übersetzer*(Parser, Preprozessor) erfolgen:

Grundidee ist die Beschreibung des Straßennetzes durch eine Grammatik, um beim Generierungsprozeß der Szene aus einem Beschreibungsfile sowohl logische Straßeninformation für die Simulation des Verkehrs(Ergebnis ist ein *gerichteter Straßengraph*), als auch geometrische Information über den *polygonalen Objektaufbau* zu erhalten, wodurch die Synchronisation beider Informationseinheiten sichergestellt werden soll. Als Beispielprojekt wird hier NSL angeführt, da es diese Eingabeform als einer der wenigen gegenwärtigen Simulatoren verwendet:

NSL – Network Specification Language(van Wolffelaar)[22,23]):
Die *Syntax* von orientiert sich typischerweise an High-level-Programmiersprachen und ermöglicht eine Definition der Komponenten (Straßensegmente, Kreuzungen, Verkehrszeichen) mittels strukturierten Blöcken, die konstant oder variabel sein können, sowie unterschiedliche Parameter und Referenzen enthalten können. Die Auswahl an Parametern ist relativ komplex und ermöglicht eine detaillierte Beschreibung von horizontalen- und vertikalen Abmessungen des Straßenprofils, sowie Positionen von Verbreiterungen, Verengungen und Einmündungen mit Angabe der relativen Position zum Startpunkt der Straße. Die wichtigste Information für die Generierung des Straßengraphen ist dabei die Richtung des Verkehrs pro Segment und Fahrspur. Zur Flexibilisierung der Struktur können Segmente auch geteilt und verbunden werden, sodaß entgegengesetzte Fahrspuren nicht notwendigerweise parallel laufen müssen. Die Auswertung des Beschreibungsfiles erfolgt durch einen *Übersetzer*, welcher Ausdrücke auswertet, fehlende Information ergänzt und Referenzen auflöst und den Verlauf der Fahrspuren durch einen sogenannten *Pfad* beschreibt, der die einzelnen Straßenteile verbindet und Grundlage der logischen Straßenbeschreibung ist.

Zusammenfassung:

Es wurden einige Möglichkeiten der Repräsentation genannt, das in dieser Arbeit vorgestellte Modell - *VRMG* entspricht der ersten (geometrischen-) Schicht und beinhaltet folgende Konzepte:

- *parametergesteuert:* Zum Erfassen der Variationsmöglichkeiten von Straßen werden Parameter(Attribute) verwendet, diese beziehen sich im wesentlichen auf die Form, die Anordnung der Fahrstreifen, deren Verwendungstyp und der Bodenmarkierung.
- *hierarchisch:* Ein Baukastenprinzip ermöglicht durch Segmentzerlegung von Straßenstücken die Umsetzung in eine Repräsentation durch Objektklassen. Dadurch lassen sich auch komplexere Strukturen wie Kreuzungen aus einfacheren Teilen zusammensetzen.
- *2.5D:* die Straßen werden im Grundriß(x,y) erfaßt und die Höhenwerte aus einem Höhenfeldmodell errechnet. Auf die Verwendung von Raumkurven wird verzichtet, da die Straßenoberflächen im Stadtbereich kaum Oberflächensprünge aufweisen.
- *Eingabe durch Netzwerk-Beschreibungsfile*: um der Verwendung verschiedener Informationsquellen für die Straßennetzdaten gerecht zu werden, wurde als universelle Form der Dateneingabe das Beschreibungsfile gewählt.

2.2.) Kommerzielle Modellierungs- und Planungssoftware:

Seit ungefähr zwei Jahrzehnten existieren *kommerzielle Softwareprodukte* für Planung, Berechnung und Visualisierung, wobei sich der Visualisierungsaspekt erst in den letzten Jahren auf der Grundlage einer leistungsfähigen Grafikhardware entfalten konnte. Die derzeit bekanntesten Produkte sind *Multigen* und *RoadViz*. Die Übergänge vom reinen Planungszweck für Bauingenieure und Stadtplaner zur ausschließlichen Ausgestaltung einer Virtuellen Welt durch Künstler und Multimedia-Designer sind fließend. Auch *AutoCAD* ist durch die Fähigkeit des computerunterstützten Designs in 2D- oder 3D-Grafik seit einiger Zeit zur Projektierung von Straßennetzen und der Neugestaltung von Stadtteilen in Verwendung.

Multigen[21] besitzt eine *interaktive Entwicklungsumgebung*(„WYSIWYG – What you see is what you get“), es stehen zahlreiche Grafikwerkzeuge zur Erstellung von 3D-Objekten zur Verfügung(„Copy and Paste“, Texturierung mit OpenGL-Unterstützung, 3D-Soundunterstützung, „DOF's – Spezifikation der möglichen Freiheitsgrade für ein Objekt). Für die Aufnahme der Geometrieinformation wird eine hierarchische Datenbank erstellt, diese läßt sich in Hinblick auf Echtzeit-Visualisierung optimieren, so können von jedem erzeugten Objekt mehrere Levels-Of-Detail(LOD's) generiert werden. Weitere Funktionen sind die Berechnung von *Bounding-Volumes* für *Occlusion Culling* und Umstrukturierungsmöglichkeiten der Datenstrukturen. Das Softwarepaket umfaßt ein Basismodul und mehrere optionale Zusatzmodule für unterschiedliche Zwecke. Die in diesem Zusammenhang wichtigen Module heißen *RoadPro*(Modellierung von Straßen) und *TerrainPro*(Höhenfeldmodell). Die Entwicklung von RoadPro erfolgte in Zusammenarbeit mit der Universität Iowa, die theoretischen Konzepte wurden weiter oben im Abschnitt „Wissenschaftliche Forschung“ unter „Environment Description Framework (EDF)“ beschrieben. Obwohl Multigen ursprünglich für militärische Anwendungen entwickelt wurde, ist es geeignet, als Baustein in ein komplexeres System eingefügt zu werden, wie dies z.B. beim *Urban Simulator*-Projekt der Fall war[1]. bei dem eine Benutzer-Schnittstelle zur interaktiven Stadtplanung Gegenstand der Arbeit war: Die Möglichkeiten erstrecken sich vom Durchwandern des Stadtmodells bis hin zum Manipulieren, Ersetzen und Informationsabfrage von Objekten. Der

geometrische Modellierungsprozeß erfolgt hierbei durch Multigen, das aufgrund von Luftbildern einerseits 3D-Modelle für Straßen, Häuser und Bäume erstellt und andererseits diese Information in einer Datenbank speichert. Zur genauen Erfassung, wie etwa von Häuserfronten, werden digitale Videoaufnahmen perspektivisch korrigiert und in der Texturdatenbank gespeichert, welche nach topologischen Gesichtspunkten strukturiert ist. Seit der Fertigstellung im Jahr 1995 ist dieser Simulator für Stadtplanungszwecke in Verwendung.

RoadViz[32] wurde von *Bashir Research* entwickelt, wobei eine Einteilung der Funktionalität in die Bereiche: Schnappschüsse aus der Szene, Animationen und 3D-Modelle vorgenommen wurde. Es stehen verschiedene voreingestellte Optionen für die Position der Kamera zur Verfügung, dazu zählen die Vogelperspektive und die Betrachtung der Szene aus einem Fahrzeug oder aus der Sicht eines Fußgängers. Der Verwendungszweck ist im Planungswesen angesiedelt, wobei der Schwerpunkt auf Visualisierung und letztendlich der Publikation einer projektierten Veränderung im Stadtbild liegt. Die Softwaremäßige Anbindung erfolgte an *AutoCAD* auf der Eingabeseite und an *VRML* oder *Performer* auf der Ausgabeseite, was einer Anpassung an derzeitige Standards entspricht. Spezieller Wert wurde auf folgende Aspekte gelegt:

- Einhaltung von planungstechnischen Genauigkeitsanforderungen.
- Straßengenerierung aufgrund von vermessungstechnischen Rohdaten oder AutoCad-Daten. Dafür stehen verschiedene Kurvenformen für Straßenstücke zur Auswahl, außerdem können Querschnittsprofile umgesetzt werden, um z.B. Kurven eine bestimmte Steilheit zu geben.
- Erweiterung oder Verbesserung der Möglichkeiten von übergeordneten Anwendungsprogrammen.
- Textur-Optionen zur Erhöhung des Realismus.
- Einbeziehung dynamischer Elemente wie Ampeln oder Fahrzeuge.

Folgende Anwendungsbereiche werden vom Hersteller angegeben:

- Planungsanalyse von Autobahnprojekten
- Auswertung von Bauprojekten
- Publikationen für Öffentlichkeitsinformation
- Fahrsimulationen
- Verkehrsanalyse

Insgesamt betrachtet lassen sich mit *RoadViz* sehr ansprechende und komplexe Straßenkonstrukte mit allen erdenklichen Möglichkeiten wie Brücken, Mittelinseln, Leitschienen usw. erstellen. Zusätzlich wurde großer Wert auf die Einhaltung derzeitiger Standards gelegt, so etwa durch die Verwendung von VRML für WWW-Publikationen oder von *bildbasierten Elementen*, auf die weiter unten im Kapitel 3.1. – „Modellierung von 3D-Objekten“, näher eingegangen wird.

AutoCAD 2000[31]ist die derzeit aktuelle Version von AutoDesk’s AutoCAD. Die Hauptanwendungsbereiche sind das Entwerfen von Fertigungsteilen in der Industrie durch den technischen Designer und die Bauprojektierung durch Architekten. Die Benutzeroberfläche ist im Vergleich zu Vorgängerversionen an konventionelle Windows-Standards angepaßt worden, wodurch die Arbeitergonomie verbessert wurde. Ein Großteil der Kommunikation zwischen Anwender und System erfolgt dialogorientiert. Zu den AutoCAD Funktionen zählen u.a.:

- *3D-Orbit*:
  Visualisierungswerkzeug(engl.: *Viewer*) für 3D-Objekte, damit können diese in Echtzeit rotiert und gezoomt werden. Weiters kann die Art der Projektion(orthographisch oder perspektivisch)

eingestellt werden und der Bereich des Sichtbereiches(engl: *Viewing frustum*) durch eine vordere und hintere Schnittebene(engl.: *clipping-plane*) begrenzt werden. Es stehen verschiedene Schattierungsmodelle zur Auswahl, u.a.: *Gouraud-* und *Hidden Line-*Schattierung. Für jeden Augpunkt läßt sich zur Erhöhung der Flexibilität ein *UCS*(User Coordinate System) definieren, wodurch der Wechsel zwischen verschiedenen Ansichten erleichtert wird.

- *Object Properties Manager*:
  dieser dient zur tabellarischen Erfassung und Verwaltung von Attributen der Konstruktionsobjekte.
- *Layer Properties Manager:*
  ein wesentliches Merkmal von AutoCAD ist die Möglichkeit der Zuweisung von Entwurfsobjekten zu bestimmten Schichten(engl.: *Layers*), damit kann eine Selektion einer bestimmten Objektkategorie in Bezug auf die Ausgabe erreicht werden. Der *Layer Properties Manager* übernimmt alle Aufgaben, die in diesem Zusammenhang notwendig sind.

Außer den genannten Funktionen, die allesamt dem mehr oder weniger interaktiven Design zugeordnet sind, ist die Form der Ausgabe an neuere Standards angepaßt worden. Die häufigste Art der Ausgabe von AutoCAD-Entwürfen ist ein technischer Plan in Form eines *Plots*. Durch die Verwendung von *HDI*(Heidi Device Interface mit gepipelineter Grafikdaten-Verarbeitung) wurde durch die bessere Ausnutzung der Grafikhardware die Leistung um etwa 20% erhöht. Allerdings bezieht sich diese Leistung nur auf die Ausgabegeschwindigkeit während der Entwicklung, nicht auf die Optimierung der erzeugten Geometriedaten, die im DXF AutoCAD-Format gespeichert werden. Somit sind speziell für Simulatorumgebungen 3D-Modeler, die diese Optimierungsoptionen unterstützen, vorzuziehen.

3.) Theoretische Grundlagen:

3.1.) Modellierung von 3D-Objekten:

Grundsätzlich unterscheidet man zwischen *oberflächenbasierten-* und *volumenbasierten-* 3D-Objekten. Oberflächenbasierte Objekte bestehen quasi nur aus der „äußeren Hülle“, die sich aus mehreren Flächen zusammensetzt. Der Vorteil dieser Repräsentationsform ist, daß Operationen wie Berechnung der Sichtbarkeit, Schattierung, Texturierung usw. relativ gut durchführbar sind und auch von gegenwärtiger Grafikhardware unterstützt werden. Volumenbasierte Objekte repräsentieren Objekte in ihrer kompakten Form durch Zusammensetzung von Primitiven(Grundobjekte wie Quader, Zylinder, Kugel usw.). Diese werden durch Mengenoperationen(Vereinigung, Differenz usw.) verknüpft.

Für die Modellierung von Echtzeit-Simulationen wird generell die oberflächenorienterte Repräsentation verwendet. Die Form eines Objektes wird durch eine polygonale Oberfläche approximiert, diese setzt sich aus stückweise linearen Flächen zusammen. Diese Vereinfachung(Approximation) der Oberfläche wird durch Dreieckszerlegung erreicht. Diese kann unter der Voraussetzung, daß alle Oberflächenpunkte etwa koplanar sind, in 2 Dimensionen erfolgen, wie dies bei der *Delaunay-Triangulierung* der Fall ist[19]. Die aktuellen Chips der

Hardwarebeschleuniger sind auf Dreiecksoperationen spezialisiert, weshalb alle Grafikdaten in die Form der Dreiecksvermaschung gebracht wird.

Für die *Modellierung* von dreidimensionalen Szenen für interaktive Anwendungen kommen noch weitere Überlegungen in Betracht:

Das Ziel jeder *interaktiven Grafik-Anwendung* ist erstens eine flüssige Darstellung, was bedeutet, daß die Berechnung und Auffrischung des Monitorbildes mit akzeptabler Geschwindigkeit erfolgt. Für das menschliche Auge ist eine Bildgenerierungsrate von etwa 70 Hz. wünschenswert. Zweitens sollte die Bildneuberechnung etwa konstant sein, das bedeutet, daß die zwischen dem Setzen eines Signales und der Darstellung des entsprechenden Resulates benötigte *Latenzzeit* auch für schnelle Änderungen nicht wesentlich größer werden sollte.

Demzufolge sind außer der Erstellung von reinen Geometriedaten(Koordinaten, Transformationen, Beleuchtung etc.) folgende Aspekte für die Performance von Bedeutung[20]:

- **Optimierung der Datenbank**[14,15,16]**:** für komplexe Szenen werden die Geometriedaten in einer Datenbank gespeichert. Das zyklische Auswerten erfolgt durch die CPU und ist ein zeitkritischer Prozeß. Speziell für große bis sehr große Szenen entstehen zusätzliche Probleme, einerseits im Speicher, wenn die Daten nachgeladen werden müssen, weil die Datenbank die Speicherkapazität des Hauptspeichers übersteigt, andererseits im Fall der Netzwerkübertragung, bei der die Begrenzung durch die Bandbreite den Datenfluß verzögert. Mögliche Lösungen wären eine Strukturierung der Datenbank nach topologischen Aspekten im Sinne der Vorhersagewahrscheinlichkeit für aufeinanderfolgende Zugriffe. Um Verzögerungen in der Bildgenerierung bei Netzwerkübertragungen von 3D-Szenen zu vermeiden, werden verschiedene Techniken angewandt, z.B. Datenkompression oder sukzessive Bildverfeinerung durch zwei oder mehrere Auflösungsstufen der Geometriedaten.

- **Geometrische Komplexität**[19,20]**:** da ein grundsätzlicher „Trade-Off“ zwischen Detailreichtum und Darstellungsgeschwindigkeit besteht, ergibt sich als logische Konsequenz die Notwendigkeit der Reduktion von nicht sinnvollen Berechnungsschritten (d.h. nicht sichtbare Details werden nicht gerendert). In diesem Zusammenhang entstand die Idee der Generierung und Auswahl von mehreren Auflösungsstufen, diese werden als *LOD*(Levels-of-Detail) bezeichnet und bezeichnen die Repräsentation eines 3D-Objektes in mehreren Auflösungsstufen. Dadurch wird erreicht, daß Objekte im Vordergrund detaillierter gezeigt werden als jene, die bedingt durch die Perspektive, sehr klein erscheinen. Die (aufwendigen) Berechnungen für die Objektrepräsentanten erfolgen also schon vor dem Start der Simulation und beschränken sich für dessen Dauer auf (geringfügige) Berechnungen für die Auswahl der geeigneten Auflösungsstufe.

- **Sichtbarkeitsberechnung**[18]**:** Die Tatsache, daß beim Betrachten der virtuellen Stadt oder Szene im allgemeinen nur ein (geringer) Teilauszug derselben sichtbar ist, macht plausibel, daß sich das Rendering auf die Berechnung des sichtbaren Bereiches beschränken sollte. Der grundlegende Ansatz für dieses Problem sieht vor, daß einige repräsentative Objektflächen (meist Polygone im Sichtfeld der Kamera mit räumlicher Nähe oder Größe in Bezug zum Augpunkt, engl.: *Occluders*) gegen andere Objekte (im „Sichtkegel„) getestet werden. Sind die Objekte verdeckt, so werden sie nicht gerendert. Im Rahmen des Projektes *URBANVIZ*, wurden diese Aspekte von Wonka et al. kombiniert und eine als *Occluder Shadows* bezeichnete Methode entwickelt, die besonders für schnelles Durchfahren der Szene (etwa bei einer Fahrsimulation) geeignet ist. Geaueres dazu findet man in[18].

- **Bildbasierte Darstellungsverfahren:** Die Grundidee bei dieser Methode ist, daß Teile von 3D-Objekten nicht bis ins Detail ausmodelliert werden, sondern feinere Details durch Aufbringen eines entsprechenden (2D-)Bildes ersetzt werden. Da ein Bild - unabhängig vom Inhalt – eine konstante Anzahl von Bildpunkten hat, ist der Grundsatz, daß mehr Details mehr Rechenaufwand implizieren, in diesem Fall nicht gültig.

Die Summe dieser Überlegungen hat direkten Einfluß auf die Konzeption von Optionen eines 3D-Modells für gute Laufzeit-Performance.

3.2.) Grundlagen der Höhenfeldmodellierung:

Unter einem Höhenfeld versteht man die approximative Modellierung eines begrenzten Teiles der Erdoberfläche. Ausgehend von den Höhenwerten an bestimmten Meßpunkten, die regelmäßig(regular) oder unregelmäßig(irregular) verteilt sein können, spricht man von einem *regulären* oder einem *irregulären Raster.*

Die *digitale Repräsentationsform* eines *Höhenfeldmodells* ist in der Computergrafik im einfachen Fall ein *regulärer (rechteckiger-) Raster*, dessen Stützpunkte durch eine polygonale Oberfläche verbunden sind. Ausgehend von diesem (Grund-)Raster können verschiedene Auflösungs-stufen(Levels-of-Detail) generiert werden, sodaß je nach gewünschter Darstellungsgenauigkeit, eine dieser Stufe für die visuelle Ausgabe verwendet wird [20].

Eine weitere Repräsentationsform wäre die *Hierarchical Subdivision* Methode basierend auf einer Baumstruktur(Quadtree, k-d Baum und hierarchische Triangulierung mittels „Divide and Conquer“), bei diesen erfolgt eine Anpassung an lokale Änderungen im Höhenfeld: für die (rekursive) Unterteilung wird ein Toleranzparameter gesetzt, sodaß nur jene Regionen weiter zerlegt werden, deren Höhenwertdifferenzen noch nicht im Toleranzbereich sind. Grundsätzlich läßt sich auch das einfache, auf einem Regulären Raster(Grid) basierende Höhenfeld, im „bottom-up“ Verfahren (im Gegensatz zum hierarchischen („top-down“-) Verfahren) nachträglich durch *Dezimierung* der Punkte für die Triangulation durch Verwendung derselben Kriterien(ähnliche Höhenwerte in einer Region) verbessern. Das Ergebnis dieser Verfahren ist ein *Triangulated irregular Network(TIN)*, also eine Dreiecksvermaschung, bei der die Höhe eine Funktion von x und y ist: H(x,y) [19]. Daniel Michel und David L. Brock verwenden ein *TIN* für die Repräsentation des Höhenfeldes in verschiedenen Auflösungsstufen für interaktive Simulation[3]: Bei diesem Ansatz wird das Terrain aus unterschiedlichen Auflösungen zusammengesetzt, wofür eine hierarchische Baumstruktur verwendet wird. Zur Laufzeit wird in Abhängigkeit vom Beobachtungspunkt die für einen Bereich geeignete Auflösungsstufe ausgewählt, was letztendlich der Levels-of-Detail Idee entspricht.

3.3.) Grundlagen - Straßenbau:

Der erste Schritt zur Planung einer Straße in der Grundrißdarstellung heißt in der Straßenbaupraxis *Absteckungsberechnung.* Die folgende Abbildung zeigt einen Polygonzug, der den Verlauf der Straße festlegt:

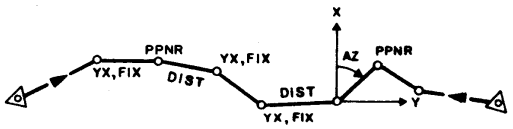


Abbildung 2: Mittelachse einer projektierten Straße als Polygonzug

Ausgehend von den *Hauptpunkten*(„PPNR“) werden durch Angabe von absoluten Winkel(Azimut, „AZ“), relativen Winkel(Richtungsänderung) und Distanz(„DIST‘), Fixpunkte(„FIX“) berechnet[5]. Diese Polygonpunkte wurden früher häufig nur nach der graphischen Methode bestimmt, allerdings nunmehr nach der rechnerischen Methode ermittelt, deren Voraussetzung ist allerdings das Vorliegen aller nötigen Koordinatenwerte für Anschluß- oder Hauptpunkte. Der Anschluß des Polygons an bereits bestehende Straßen erfolgt durch den Anfangs- bzw. Endpunkt und die dazugehörige Richtung[6]. Die iterative Berechnung aller Punkte P0..Pm des Polygonzugs in Rechenrichtung ist durch folgende Formeln gegeben:

$$vn = v(n-1) + bn \pm 200\,gon$$

mit v(0) als Startwinkel(Azimut) im Anschlußpunkt oder Hauptpunkt, bn ist der Winkel der relativen Richtungsänderung. Somit ist der Streckenvektor von je zwei aufeinander-folgenden Punkten des Polygonzuges:

$$\Delta yn = sn \cdot \sin(vn),$$
$$\Delta xn = sn \cdot \cos(vn)$$

mit sn ist gleich der Streckenlänge von P(n-1) nach P(n).

Der nächste Schritt ist die *Trassierung*(Linienführung), dabei werden die Polygonpunkte durch *Trassierungselemente* verbunden. Aufgabe der Trassierung ist es, aus dem Polygonzug eine Raumkurve zu erzeugen, die der Bezugslinie(Mittellinie, Achse, Gradiente) der Straße entspricht und verkehrsergonomischen Anforderungen gerecht wird.

Folgende Trassierungselemente(Entwurfselemente) werden verwendet:

1. Gerade
2. Kreisbogen
3. Klothoide
4. Übergangsbögen als Kombination von 1-3

Die Gerade:

Diese bildet die kürzeste Verbindung zwischen zwei Punkten. Obwohl diese für die Absteckungsberechnungen die einfachste und naheliegendste Form ist, wird sie aufgrund zahlreicher Nachteile(Blendwirkung durch Gegenverkehr, schwere Abschätzung der Geschwindigkeit des Gegenverkehrs usw.) primär nur im innerörtlichen Bereich verwendet[4,9].

Der Kreisbogen:

Als natürliche und einfache Form der Kurve kommt als erstes der Kreisbogen in Betracht. Fahrtechnisch ist er durch konstanten Lenkeinschlag charakterisiert. Bei aufeinander-folgenden Kreisstücken achtet man in der Straßenbaupraxis auf die *Relationstrassierung*, bei dieser sind die Abweichungen in den Radien durch Grenzen festgesetzt, um einen harmonischen Übergang zu erzielen[4,9].

Der Länge(L) und der Radius(R) wird in Abhängigkeit von der Entwurfs-geschwindigkeit(Vc) durch folgende einfache Formel ausgedrückt[4]:

minL = Vc/1.8 als Mindestmaß für die Bogenlänge.

Ausgehend von der Länge des kreisbogenförmigen Straßenabschnitts wird der Radius bemessen:

R=400m für L>=300m und R>L für L<300m.

Die für Absteckungsrechnungen wichtigen Formeln werden im folgenden angeführt und sind relativ zum Kreismittelpunkt zu verstehen:

Tangentenlänge:

$$t = r \cdot \tan(\alpha / 2)$$

Scheiteltangente:

$$ts = r \cdot \tan(\alpha / 4)$$

Scheitelabstand:

$$f = \frac{r}{\cos(\alpha / 2)} - r$$

Pfeilhöhe:

$$p = r - r \cdot \cos(\alpha / 2)$$

Sehnenlänge:

$$s = 2 \cdot r \cdot \sin(\alpha / 2)$$

Bogenlänge:

$$b = \frac{p \cdot r \cdot \alpha}{200}$$

Die Klothoide:

Das dritte wichtige Trassierungselement, welches im Straßenbau verwendet wird ist die *Klothoide(Cornu-Spirale)*. Diese spielt als Übergangsbogen von Kreis zu Gerade oder umgekehrt eine wichtige Rolle, und zwar aus Gründen der Fahrergonomie, um abrupte Richtungsänderungen

zu vermeiden. Als Übergangsbogen wird deswegen eine mathematische Kurve benötigt, deren Krümmung mit der durchfahrenen Länge stetig zunimmt[4][10].

Diese Bedingung erfüllt die Klothoide, ihr Bildungsgesetz lautet:

$$A^2 = r \cdot l$$

Der Klothoidenparameter A ist konstant und bestimmt die Form, während die Klothoidenkrümmung linear zunimmt, r bezeichnet den stetig abnehmenden Klothoidenradius und l die Bogenlänge. Aus praktischen Gründen wählte man für den Parameter A die quadratische Form, da bei Berechnungen sowohl r als auch l in m verwendet werden und damit auch A in m angegeben werden kann. Dieser Parameter hat die gleiche Wirkung der Vergrößerung bzw. Verkleinerung, wie sie vom Einheitskreis her bekannt ist. Somit sind alle Klothoiden in ihrer Form ähnlich. Sie haben für das Verhältnis r/a immer die gleiche Form, da an dieser Stelle der Tangentenwinkel T stets gleich ist[9]:

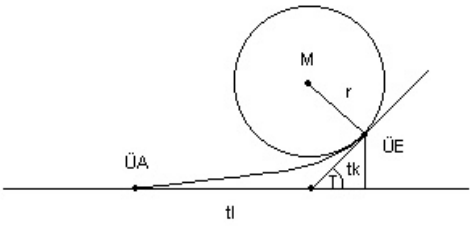


Abbildung 3: Klothoide als Übergangsbogen von Gerade zu Kreis

Die auf der Abbildung gezeigten Konstruktionselemente sind:

- ÜA ist der Übergangsbogenanfang, bei dem die Gerade mit der Tangente der Klothoide am Punkt r=8 zusammenfällt.
- ÜE ist das Übergangsbogenende, an diesem Punkt geht der Bogen tangential in den Kreisbogen über. An dieser Stelle ist der Radius auf der Klothoide gleich groß wie der des Hauptbogens.
- Die Anfangstangente(tl) in ÜA bildet mit der Endtangente(tk) in ÜE den Winkel T. Die Längen der Klothoidentangenten sind wegen der sich stetig ändernden Krümmung der Klothoide verschieden.

Es gelten folgende Formeln:

Abschätzung des Klothoidenparameters A:

$$\sqrt{2a} \cdot r < A < p \cdot r$$

Winkel zwischen Tangente im Übergangsbogenanfang und in beliebigem Punkt:

$$T = \frac{l}{2 \cdot r}$$

Klothoidenpunkt relativ zum Übergangsbogenanfang:

$$X = \frac{A}{\sqrt{2}} \cdot \int_0^T T^{-\frac{1}{2}} \cdot \cos T \partial T \qquad Y = \frac{A}{\sqrt{2}} \cdot \int_0^T T^{-\frac{1}{2}} \cdot \sin T \partial T$$

Bei diesen Gleichungen handelt es sich um Fresnelsche Integrale, die in der Praxis mittels der Reihenentwicklung für Trigonometrische Funktionen gelöst werden:

$$X = \frac{A}{\sqrt{2}} \cdot \int_0^T (T^{-\frac{1}{2}} - \frac{T^{\frac{3}{2}}}{2!} + \frac{T^{\frac{7}{2}}}{4!} \pm ...) \partial T$$

ergibt für die Abszisse:

$$X = A \cdot \sqrt{2 \cdot T} \cdot (1 - \frac{T^2}{5 \cdot 2!} + \frac{T^4}{9 \cdot 4!} - \frac{T^6}{13 \cdot 6!} \pm ...)$$

und

$$Y = \frac{A}{\sqrt{2}} \cdot \int_0^T (T^{\frac{1}{2}} - \frac{T^{\frac{5}{2}}}{3!} + \frac{T^{\frac{9}{2}}}{5!} \pm ...) \partial T$$

ergibt für die Ordinate:

$$Y = A \cdot \sqrt{2 \cdot T} \cdot (\frac{T}{3} - \frac{T^3}{7 \cdot 3!} + \frac{T^5}{11 \cdot 5!} - \frac{T^7}{15 \cdot 7!} \pm ...)$$

Innerhalb gewisser Grenzen läßt sich die Klothoide durch eine *kubische Parabel* ersetzen:

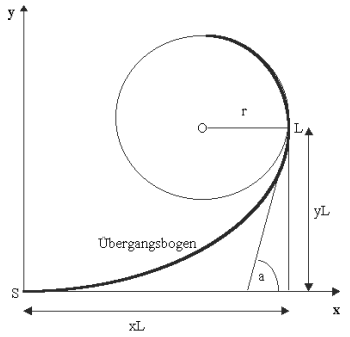


Abbildung 4: Kubische Parabel als Übergangsbogen von Gerade zu Kreis

Obwohl diese in der Straßenbaupraxis nicht verwendet wird, ist sie wegen der leichteren Berechenbarkeit für Modellierungszwecke durchaus geeignet.

$$y = m \cdot x^3$$

Der Skalierungsparameter m ist für eine bestimmte Parabel konstant und spezifiziert dabei die Krümmunganpassung.

Absteckungsberechnung durch Kleinpunkte:

Sind die Hauptpunkte durch geeignete Trassierungselemente verbunden, erfolgt die Verfeinerung des Verlaufs durch *Kleinpunkte*(Zwischenpunkte), deren Position durch die Art des *Trassierungselements* bestimmt wird und auf dem entsprechenden Bogen liegt. Die folgende Abbildung zeigt die Absteckungsberechnung am Beispiel eines Übergangsbogens nach der *Pfeilmethode:*

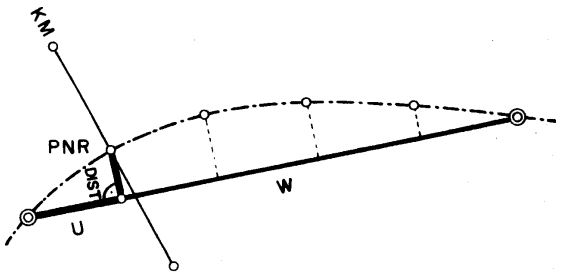


Abbildung 5: Übertragung der Stützpunkte nach der Pfeilmethode

Hier sind zwei aufeinanderfolgende Fixpunkte durch eine Sehne verbunden, der (Bogen-) Kleinpunkt(„PNR“) wird durch die Position(durch „U“ und „W“) und den Normalabstand („DIST“) relativ zur Sehne charakterisiert[5]. Für jeden dieser Punkte wird noch die Breite („DL“ und „DR“) angegeben, wie auf folgender Abbildung gezeigt wird:

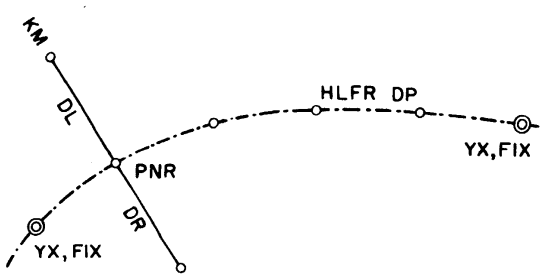


Abbildung 6: Absteckung des Straßenrandes

Diese Prinzipien zur Approximation des Straßenverlaufs durch einen Polygonzug sind außer für die Straßenbaupraxis, wo es darum geht, diese Absteckungspunkte ins Gelände zu übertragen, auch für die Geometrische Modellierung von Straßen in einem 2-oder 3-dimensionalen Koordinatensystem anwendbar.

Straßentypen:

Die Einteilung der Straßen erfolgt in Kategoriengruppen, wobei die Einteilung nach dem Verwendungszweck, der Lage und der Anbaufreiheit erfolgt[9]:

- *Kategoriengruppe A:*

Anbaufreie, also ohne Zufahrt zu Grundstücken, flächenerschließende Straßen außerhalb bebauter Gebiete mit einem oder zwei Fahrstreifen.
- *Kategoriengruppe B:*
  Anbaufreie Hauptverkehrs- oder Hauptsammelstraßen im städtischen Bereich oder im Übergangsbereich am Stadtrand mit einem oder zwei Fahrstreifen.
- *Kategoriengruppe C:*
  Angebaute Straßen mit Verbindungsfunktion innerhalb bebauter Gebiete mit höchstens zwei Fahrstreifen.
- *Kategoriengruppe D:*
  Straßen mit Erschließungsfunktion in bebauten Gebieten mit einem Fahrstreifen, Gehsteig und eventuell Radweg.
- *Kategoriengruppe E:*
  Straßen in Wohngebieten mit Aufenthaltsfunktion.

Querschnittsgestaltung von Straßen:

Je nach Verwendungszweck des jeweiligen Fahrstreifens wird eine *Fahrstreifengrundbreite* eines Bemessungsfahrzeuges zur Beurteilung herangezogen. Die endgültige Breite errechnet sich wie folgt:

Fahrstreifengrundbreite = Fahrzeugbreite + Bewegungsspielraum,
Fahrstreifenbreite = Fahrstreifengrundbreite + Gegenverkehrszuschlag.

Typische Werte für die einzelnen Querschnittsbreiten, die als Mindestmaß anzusehen sind, wurden wie folgt festgelegt:

Fahrzeugbreite:
Fahrbahnstreifen für Kfz – 2.5m
Fahrradstreifen – 0.6m
Gehsteig – 0.75m

Bewegungsspielraum: 0.25m
Gegenverkehrszuschlag: 0.25m pro Fahrspur mit Gegenverkehr.

Die einzelnen Bestandteile des Straßenquerschnitts, die sich baukastenmäßig zusammensetzen lassen sind[9]:

- *Fahrbahn:* Diese setzt sich aus den einzelnen Fahrstreifen inklusive Randstreifen zusammen.
- *Fahrstreifen:* Dieser ist die Fläche, die für ein Fahrzeug für die Fahrt in einer Richtung zur Verfügung steht.
- *Randstreifen:* Die Aufgabe des Randstreifens besteht im seitlichen Abschluß des Straßenrandes bei Fehlen eines Randsteins und zur Aufnahme der seitlichen Begrenzungsmarkierung.
- *Trennstreifen:* Dazu zählen alle Trennungselemente zwischen Fahrstreifen mit unterschiedlichen Verwendungszweck, also Mittelstreifen, Seitentrennstreifen und Grünstreifen.
- *Seitenstreifen:* Die Aufgabe dieser Streifen ist die temporäre Aufnahme von Fahrzeugen. Zu den Seitenstreifen zählen Randstreifen, Mehrzweckstreifen und Parkstreifen.
- *Bankett:* Diese sind unbefestigte Seitenstreifen zum Böschungsrand hin, auf denen Leiteinrichtungen wie Leitschienen, Reflektoren usw. untergebracht sein können.

- *Radweg:* Dieser wird häufig durch Randsteine von der Fahrbahn abgesetzt und beiderseits der Fahrbahn geführt.
- *Gehweg:* Die durch Randsteine von der Fahrbahn abgesetzten Gehwege sollten mindestens 2m breit sein.
- *Gemeinsame Rad- und Gehwege:* Diese etwa 2m breiten Wege trennen den Fußgänger- und Radverkehr durch Bodenmarkierungen.

Knotenpunkte(Kreuzungen):

Die Knotenpunkte verbinden die einzelnen Straßen zu einem *Straßennetz*, endet eine Straße an einer durchgehenden Straße, so entsteht eine *Einmündung* oder ein dreiarmiger Knotenpunkt, kreuzen sich zwei Straßen, so spricht man von einer *Kreuzung* oder einem vierarmigen Knotenpunkt. Komplexere Knotenpunkte mit mehr als vier Armen werden auf die beiden einfachen Fälle zurückgeführt. Die Einteilung von Einmündungen bzw. Kreuzungen wird nach den Kriterien Anzahl der Fahrstreifen, Straßenkategorie und Erfordernis einer Signalregelung der einmündenden Straßen vorgenommen[4]:

- *Grundform I:* Diese verbindet zwei Straßen mit je zwei Fahrstreifen, wobei optional Linksabbiegestreifen vorhanden sein können.
- *Grundform II:* Hier erfolgt eine Verbindung einer zweibahnigen(übergeordneten) Straße mit einer untergeordneten(zweistreifigen) Straße. Dieser Knotenpunktstyp ist im allgemeinen mit einer Lichtsignalanlage ausgestattet.
- *Grundform III:* Eine Verbindung dieser Form hat zwei zweibahnige Straßen, welche zumeist der Straßenkategorie C angehören, als Knotenpunktsarme. Die Links-abbiegespur und eine Lichtsignalregelung sind hier vorhandene Einrichtungen.

Alle Knotenpunkte der Grundformen I-III können entweder als Einmündung oder als Kreuzung ausgeführt sein. Außer diesen gibt es noch weitere Grundformen:

- *Grundform IV:* Diese Art verbindet eine zweistreifige Straße mit einer Brücke mit einer zweistreifigen Straßen, welche überkreuzt wird, durch eine Schleifenrampe.
- *Grundform V:* Werden zwei zweistreifige Straßen nicht durch eine Kreuzung des Typs I, sondern durch zwei versetzte Einmündungen desselben Typs verbunden, so spricht man von einem Versatz.
- *Grundform VI:* Diese Form entsteht durch die Aufweitung einer Einmündung des Knotenpunkttyps III, die aufgeweitete Straße erhält also zwischen den Fahrbahnen eine Mittelinsel.
- *Grundform VII:* In diese Kategorie fallen alle Arten vom Kreisverkehr.

Für die *Abrundungsbögen* kommen die Planungselemente Kreis und Klothoide in Frage oder eine Kombination. In jedem Fall hat die Abrundung in Übereinstimmung mit *Schleppkurven* zu erfolgen. Die Schleppkurve entsteht durch die von einem Bemessungsfahrzeug überstrichene Fläche, wobei Fahrzeugbreite und Achsenabstand die maßgeblichen Parameter sind. Dadurch wird zumeist eine Verbreiterung am Innen- und Außenrand der Fahrbahn notwendig. Im einfachsten Abrundungsfall wird ein einzelner Kreisbogen verwendet, wobei der Radius in Abhängigkeit vom Winkel, in dem die Fahrbahnränder aufeinandertreffen, gemäß folgender Tabelle ermittelt werden[4]:

a	R
80 gon	20 m
100 gon	25 m
120 gon	25 m

Ist ein einfacher Abrundungsbogen für den Rechtsabbiegeverkehr aus verkehrstechnischen Gründen unzureichend, wird mit mehreren aufeinanderfolgenden Kreisen(Korbbögen) abgerundet, wobei das optimale Verhältnis der Radien in Fahrtrichtung 2 : 1 : 3 beträgt[4,9]. Die folgende Abbildung zeigt Kombinationen von Kreisbögen und Klothoiden:

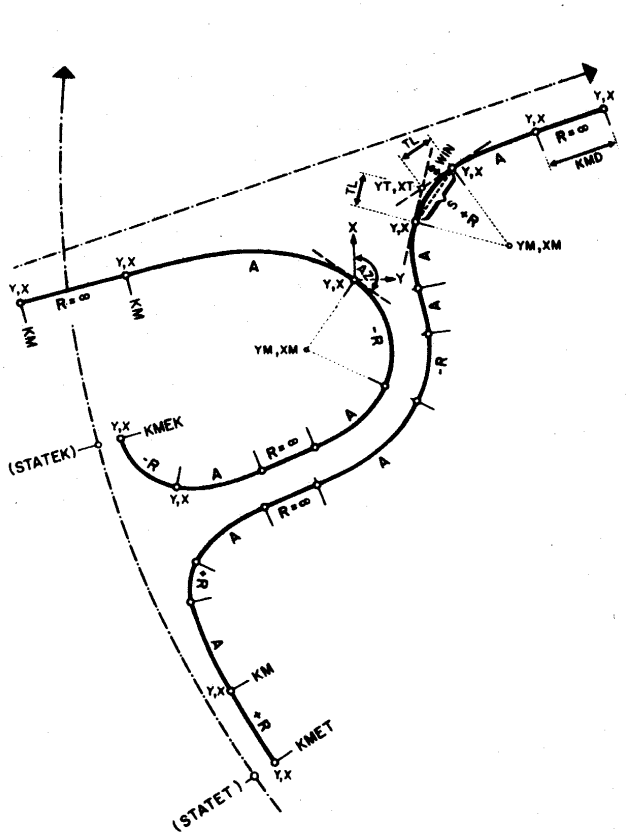


Abbildung 7: Komplexer Knotenpunkt mit Übergangsbögen

Die Übergangsbögen werden durch die Parameter „R“ (Radius) und „A“ (Klothoidenparameter) spezifiziert. Gerade sind hier durch einen Kreis mit R = 8 angegeben. Das Vorzeichen des Radius ist für eine Rechtskrümmung positiv, sonst negativ mit Bezug auf die Fahrtrichtung. Die mit „Y,X“ bezeichneten Punkte sind Punkte der seitlichen Fahrbahnränder, die von den Detailpunkten aus, welche auf der Mittelachse liegen, vertikal auf die Achse im Abstand „DL“ bzw. „DR“ ermittelt wurden. Mittelpunkt der Kreisbögen sind mit „YM, XM“ bezeichnet worden. Ebenfalls vorhanden ist die Kombination Kreis – Klothoide – Klothoide – Kreis, die einer Wendelinie entspricht[5].

4.) VRMG: Modellierung des Straßennetzes und des Höhenfeldes:

4.1.) Konzeptueller Aufbau:

In Übereinstimmung mit vergleichbaren Ansätzen ist der grundlegende erste Schritt die Gliederung des Problems in modulare Einheiten. Dadurch erreicht man einen Überblick über die Verarbeitungseinheiten(Module) und den Datenfluß. Abbildung 8 zeigt den Datenfluß von oben nach unten und die Relationen zwischen den Modulen, das System kann grob durch 3 Blöcke beschrieben werden:

- **Eingabe:** erfolgt bei *VRMG* durch 3 Beschreibungsfiles für die Grundfläche, die Objektgrundrisse und das Straßennetz. Für die Auswertung des Beschreibungsfiles für das Straßennetz wird ein Übersetzungsmodul(*Parser*) verwendet, der nicht nur die Straßendaten einliest, sondern gleichzeitig Plausibilitätstests durchführt. Für alle 3 Beschreibungsfiles wurde eine eigene Syntax definiert. Ergebnis dieses Schrittes ist der Aufbau von Datenstrukturen mit der Terrain- und Straßeninformation im Hauptspeicher.

- **Geometrieerzeugung:** ausgehend von den aufbereiteten(strukturierten) Eingangsdaten wird das Grundflächenmodell durch das Modul(*Cityplane*), das Straßenmodell durch das Modul(*Cityareas*) erstellt. Die Operationen in diesem Verarbeitungsschritt umfassen Berechnungen für die Geometrie. Ausgehend von einer Punktmenge, die quasi als Stützpunkte die Form eines 3D-Objekts definieren, werden durch Verbindungen dieser in Form von Kanten Polygone erzeugt, welche durch Triangulierung weiter in Dreiecke zerlegt werden. Außer der reinen Geometrie wird auf dieser Stufe auch die Texturierung durchgeführt. Diese umfaßt außer der Bestimmung einer geeigneten Textur in Form einer Bilddatei für eine bestimmte Fläche auch das *Texture-Mapping*. Dadurch wird mittels 2D-Texturkoordinaten die Textur auf der Fläche ausgerichtet und dimensioniert. Als Optimierung kann auf die Dreiecksmenge optional Triangle-Stripping(Zusammenfassen von benachbarten Dreiecken an gemeinsamen Kanten) angewandt werden, wodurch redundante Eckpunkte eliminiert werden. Daraus resultiert eine deutliche Reduktion der Eckpunkte, weil statt 3*n nur noch n+2 Punkte zu speichern sind.

- **Ausgabe:** diese erfolgt separat für die Grundfläche und die Straßen und umfaßt das Erstellen einer *hierarchischen Baumstruktur(Szenegraph)*. Wie schon erwähnt ist für Echzeitanwendungen die Optimierung des Szenegraphen bedeutsam. Durch Sortierung, Umordnung und Aufteilung in der Gesamtszene nach gruppierten Zellen erreicht man zur Systemlaufzeit einen schnelleren Zugriff für das erforderliche zyklische Durchlaufen. Hier liegt der Schwerpunkt auf der Berücksichtigung von den in Abschnitt 3.1. – Modellierung von 3D-Objekten, beschriebenen Echtzeitkriterien. Das Ergebnis wird in einem standardisierten Grafikfileformat als Datei gespeichert.

Abbildung-8 zeigt den strukturellen Aufbau mit den einzelnen Modulen und deren Zusammenwirken:

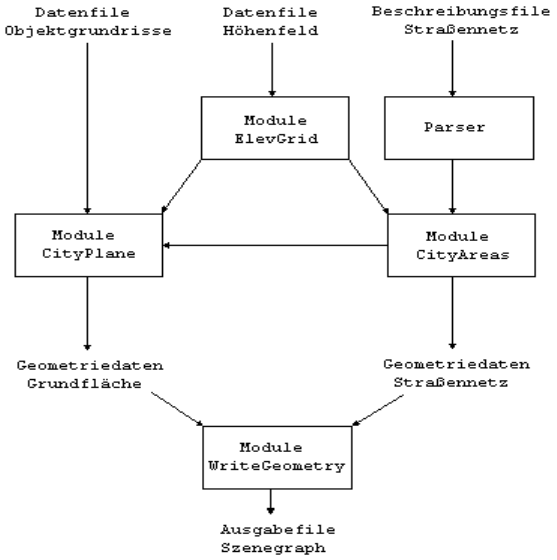


Abbildung 8: Struktogramm des modularen Systemaufbaues von VRMG.

4.2.) VRMG – Höhenfeld (Modul *CityPlane*):

Aus dem Höhenfeldmodell wird direkt die Geometrie für die Grundfläche („Terrain“) der virtuellen Stadt abgeleitet, außerdem wird die Höheninformation eines beliebigen Punktes auf der Grundfläche aus dem Höhenfeldmodell errechnet.

Die Spezifikation des Höhenfeldmodells von *VRMG* orientiert sich an dem in der Computergrafik konventionellen Höhenfeldmodell(vgl. VRML 97):

Rechteckiger Raster(Regulärer Grid), dessen Gitterpunkte
in 2 (Koordinaten-) Achsrichtungen (x,z) liegen.
height: Abstand des Höhenfeldpunktes zur x,z-Ebene.
x-dimension: Anzahl der Gitterpunkte in x-Richtung.
z-dimension: Anzahl der Gitterpunkte in z-Richtung.
x-spacing: Seitenlänge einer Rasterzelle in x-Richtung.
z-spacing: Seitenlänge einer Rasterzelle in z-Richtung.
center: Lage des Ursprungs der (lokalen) Rasterkoordinaten.
Zur Abgleichung von Unstetigkeiten in den Höhenfeldpunkten
können BSplines oder Catmull-Rom Splines verwendet werden,
dabei werden aus jeweils 4 benachbarte Polygonen in x und z
mittels einer Kontrollpunktmatrix bikubische(3D-Spline)
Oberflächen-Patches errechnet.

Die folgende Abbildung zeigt als Beispiel einen 4 x 5 – Raster:

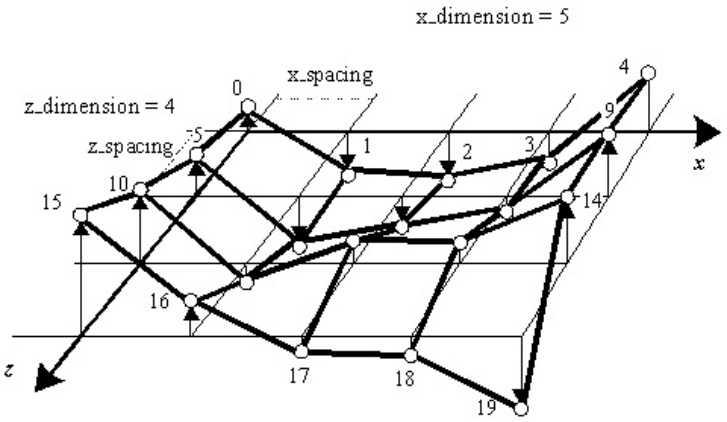


Abbildung 9: Einfaches Höhenfeldmodell mit 4 x 5 Zellen.

Zusätzlich kann das Modell – außer den Rasterpunkten - noch weitere Punkte aufnehmen, wodurch das Höhenfeld einerseits lokal verfeinert und andererseits Objektgrundrisse festgelegt werden können. Zu den für die geometrische Optimierung bedeutsamen Operationen zählen die Delaunay-Triangulierung aller Höhenfeldpunkte sowie das Entfernen von definitiv nicht sichtbaren Polygonen (Grundflächen von Objekten wie z.B. von Häusern). Das Endergebnis enthält daher alle triangulierten Restflächen und wird mit einer eigenen Textur versehen.

4.3.) VRMG – Straßenmodell (Modul *CityAreas*):

Das *VRMG* Straßenmodell wurde für den Anwendungsbereich „Städtevisualisierung“ konzipiert. Die wesentlichen Eigenschaften von Stadtstraßen lassen sich folgendermaßen charakterisieren:

- Die Straßenoberfläche ist nahezu homogen in Bezug auf die Grundfläche und weist keine großen Unstetigkeiten in den Oberflächen auf.
- Wesentliche Elemente sind Geradenstücke und 3- oder vierarmige Kreuzungen. Im Straßenquerschnitt finden sich eine ein- oder zweibahnige Fahrspur mit Grüninseln und Parkstreifen sowie Gehwege und evtl. Radwege.
- Der Straßenrand ist auf weite Strecken von Häuserfronten gesäumt.
- Der Ursprung eines Großteils der Straßen liegt weit in der Vergangenheit, sodaß sich die aktuellen verkehrstechnischen Richtlinien nur bedingt anwenden lassen.
- Die Mehrzahl der Straßen ist relativ kurz und weist mitunter starke Richtungsänderungen auf.

Diese Eigenschaften wurden bei der Entwicklung des Straßenmodells berücksichtigt, um zunächst ein Standardstraßennetz erzeugen zu können, das durch optionale Verwendung von Zusatzparametern an lokale Gegebenheiten angepaßt werden kann.

4.3.1.) Ausgangsbasis:

In Übereinstimmung sowohl mit straßentechnischen Planungskonventionen als auch mit vergleichbaren Simulationsprojekten ist es sinnvoll, das Skelett des Straßennetzes als Graph zu beschreiben. Dieser Graph bildet die Basis für die verschieden Schichten des Systems, wie oben unter Abschnitt 2.1. angeführt. Der im Projekt *URBANVIZ* verwendete Straßengraph entsteht durch Auswertung der Grundrißdaten von Häusern durch eine Heuristik mittels Voronoi-Diagrammen. Für die Visualisierung des Ergebnisses wird *ARCVIEW* verwendet. Auf folgender Abbildung erkennt man, wie Knoten und Kanten als die Grundelemente des Straßennetzes Kreuzungen und Verbindungsstücke repräsentieren:

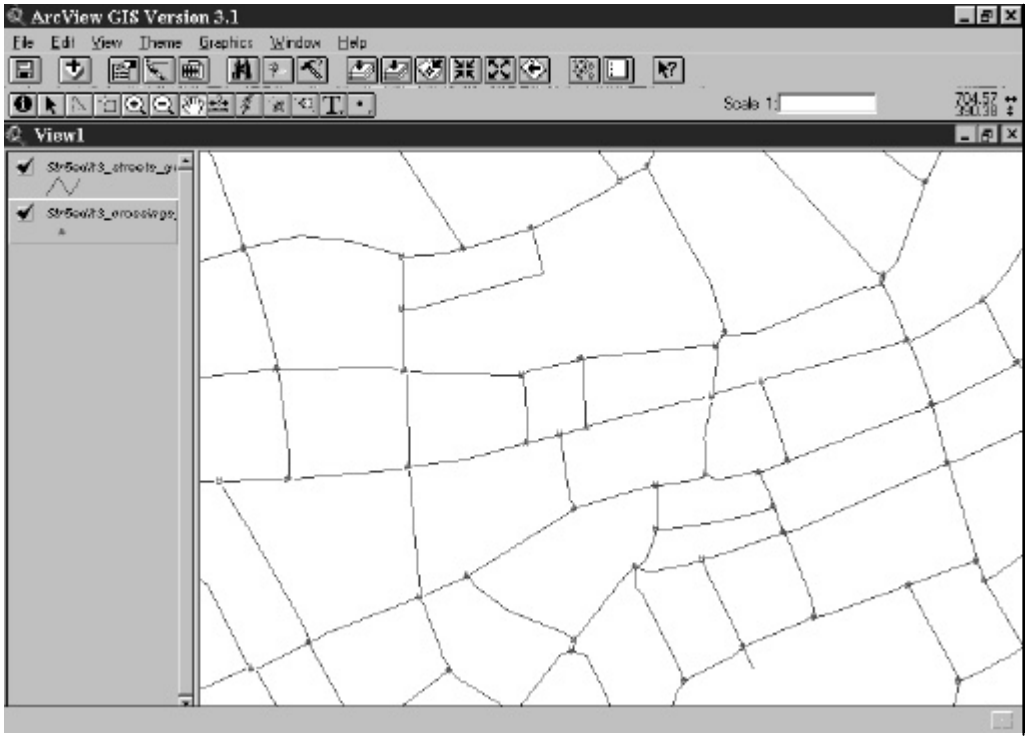
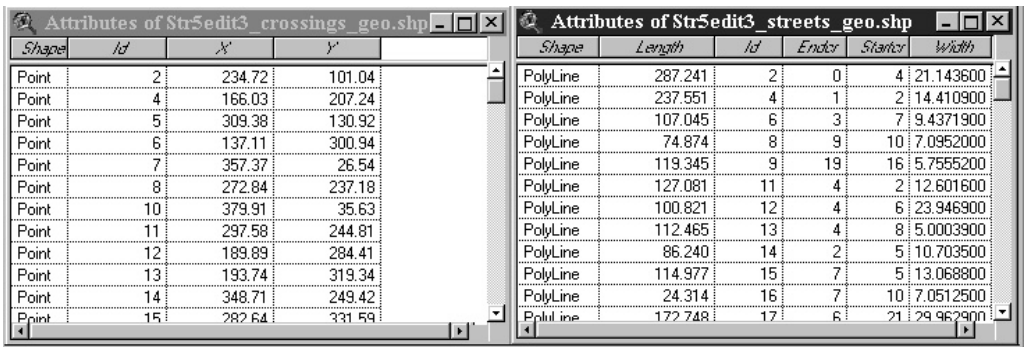


Abbildung 10: Grundriß des Straßengraphen in *ARCVIEW*.

Die in *ARCVIEW* erfaßten Straßen und Kreuzungen, i.E. Kanten und Knoten des Straßengraphs, werden mit folgenden Attributen tabellarisch beschrieben:

Shape: Punkt bzw. Polygonzug
Id: eindeutige Identifikationszahl eines Objektes vom Typ *Shape*
X,Y: Koordinaten eines Punktes im Grundriß
Startcr: Startpunkt einer Linie eines Polygonzuges
Endcr: Endpunkt einer Linie eines Polygonzuges
Width: Breite der Linie

Abbildung-11 zeigt eine Beispieltabelle mit den Daten eines Straßengraphen:



Attributes of Str5edit3_crossings_geo.shp

Shape	Id	X	Y
Point	2	234.72	101.04
Point	4	166.03	207.24
Point	5	309.38	130.92
Point	6	137.11	300.94
Point	7	357.37	26.54
Point	8	272.84	237.18
Point	10	379.91	35.63
Point	11	297.58	244.81
Point	12	189.89	284.41
Point	13	193.74	319.34
Point	14	348.71	249.42
Point	15	282.64	331.59

Attributes of Str5edit3_streets_geo.shp

Shape	Length	Id	Endcr	Startcr	Width
PolyLine	287.241	2	0	4	21.143600
PolyLine	237.551	4	1	2	14.410900
PolyLine	107.045	6	3	7	9.4371900
PolyLine	74.874	8	9	10	7.0952000
PolyLine	119.345	9	19	16	5.7555200
PolyLine	127.081	11	4	2	12.601600
PolyLine	100.821	12	4	6	23.946900
PolyLine	112.465	13	4	8	5.0003900
PolyLine	86.240	14	2	5	10.703500
PolyLine	114.977	15	7	5	13.068800
PolyLine	24.314	16	7	10	7.0512500
PolyLine	172.748	17	6	21	29.962900

Abbildung 11: Attribute von Knoten (links) und Kanten (rechts)

4.3.2.) Aufbau von Straßen:

Das VRMG Modell des Straßennetzes wurde dem Anwendungsfall „Stadtstraßen“ angepaßt. Die Beschreibung des prinzipiellen Aufbaus eines *VRMG*-Straßennetzes durch Zusammensetzung von elementaren Segmenten ist Gegenstand dieses Abschnitts. Die geometrische Modellierung erfolgt – unter der Annahme, daß keine Überlappungen wie Brücken oder Verwindungen wie Steilkurven auftreten – im Grundriß aus den (x,y)-Werten. Die z-Koordinate wird vom Höhenfeldmodul abgefragt, wodurch die Straße quasi auf dem Terrain liegt. Trotzdem besteht aber in einigen Fällen die Möglichkeit, explizit Höhenwerte angeben zu können, um etwa den Straßenrand z.B. an Häusergrundrisse anpassen zu können.

Straßensegmente:

Als Grundformen werden die Elemente *Gerade* und *Kreis* verwendet, aus diesen werden durch Kombination komplexere Objekte wie Verbreiterungen- und Verengungen der Straße und, wie weiter unten beschrieben, auch Kreuzungen aufgebaut. Der grundlegende Aufbau wird anhand eines einfachen Beispiels für ein gerades Straßenstück illustriert: die Mittelachse legt die lineare Verlaufsrichtung durch den Startpunkt „Ms“ und den Endpunkt „Me“ fest. Anfangs- und Endquerschnitt stehen normal auf die Mittellinie und spezifizieren durch Abstand zum linken „bl“ und rechten „br“ Fahrstreifenrand die Fahrstreifenfläche. Dadurch entsteht ein Rechteck durch die Punkte P0, P1, P2, P3:

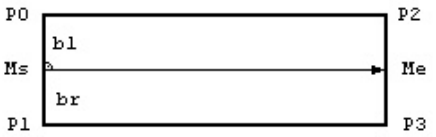


Abbildung 12: Gerades Straßensegment

Eine Verbreiterung oder Verengung wird durch unterschiedliche „bl“- und „br“- Werte definiert. Der Fahrbahnrand kann auf 3 Arten gestaltet sein:

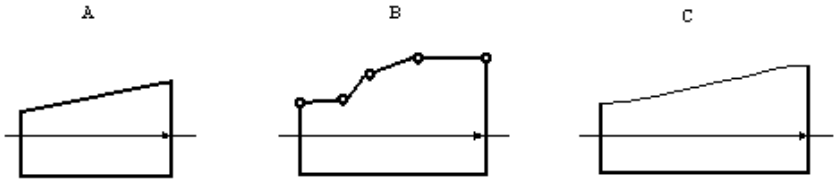


Abbildung 13: Randgestaltung bei Verbreiterung durch (A) Gerade, (B) Polyline, (C) Spline.

Durch die Verwendung der Option „Polyline“ für den Rand wird die Approximation oder Anpassung an beliebige Bedingungen für den Fahrstreifenrand möglich. Die Option „Spline“ errechnet einen kubischen Spline, der die Differenz zwischen Anfangs- und Endbreite ausrundet. Weitere Splineoptionen beinhalten die Spezifikation einer Stützpunktmenge und die Vorgabe eines Spannungsparameters, der die Stärke der Ausrundung beeinflußt.

Der Kreisbogen als zweite Grundform eines Straßenstückes wird analytisch durch Festlegung von Mittelpunkt und Radius erfaßt. Für die Geometrieerzeugung, die auf polygonalen Flächen beruht, wird der Kreisbogen diskretisiert und somit auf die Approximation durch Geradenstücke zurückgeführt. Zu beachten ist, daß die Berechnung von Mittelpunkt und Radius implizit aus dem Kontext erfolgt, sodaß sich vor und nach dem Bogenstück ein Geradenstück befinden muß. Die folgende Abbildung zweigt ein im Uhrzeigersinn definiertes Straßenstück mit Kreisbogenform. Die Oberfläche des Fahrstreifens wird wie bei der Geraden durch 2 Normalvektoren links und rechts auf die Mittelachse mit Länge „bl“ bzw. „br“ jeweils im Anfangs- und Endquerschnitt aufgespannt. Abbildung-14 illustriert die Approximation der Gesamtfläche des Kreisbogens mit 5 viereckigen, gleich großen Polygonen:

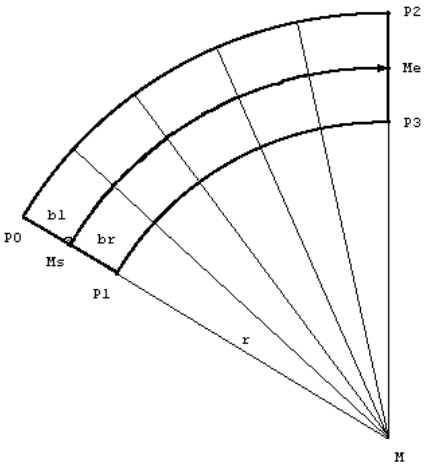


Abbildung 14: Kreisbogensegment in *VRMG.*

Die Verbreiterung oder Verengung von kreisbogenförmigen Straßensegmenten erfolgt in der gleichen Weise wie bei Geradenstücken:

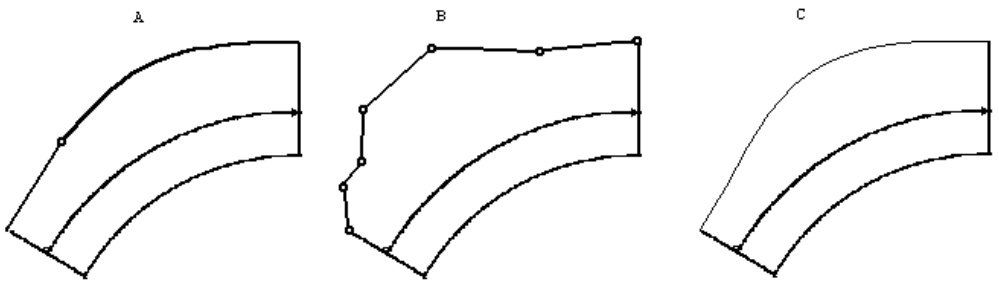


Abbildung 15: Randgestaltung bei Verbreiterung mit (A) Linie+Kreisbogen, (B) Polyline, (C) Spline.

Die einstellbaren Parameter-Optionen für die Polyline und den Spline sind mit jenen der Geradenstücke identisch.

Mehrere Fahrstreifen:

Die Realisierung von mehreren Fahrstreifen erfolgt durch axiales Nebeneinanderreihen der Basissegmente. Die standardmäßige Bezugslinie für den Rand zwischen 2 Fahrstreifen ist die Mittelachse. Die Aneinanderreihung erfolgt im Anfangs- und Endquerschnitt des Straßensegments durch Angeben der Position eines Fahrstreifenrandes. Die Fahrstreifen liegen also von links nach rechts sortiert, wobei die Bezugsrichtung vom Anfangs- zum Endquerschnitt definiert ist. Ein weiterer wichtiger Punkt ist, daß die einzelnen Fahrstreifen ohne Zwischenräume Stoß ans Stoß liegen, deshalb ist es ausreichend, nur die Querschnittspositionen für die Übergänge zu spezifizieren. Das Prinzip wird anhand von Abbildung-16 illustriert:

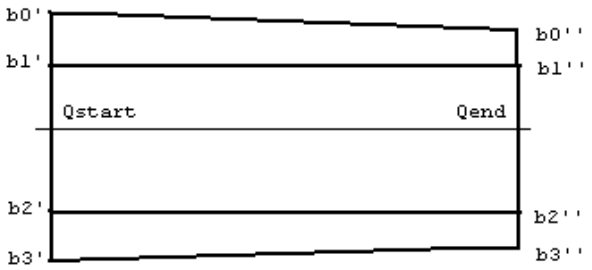


Abbildung 16: Straßensegment mit 3 Fahrstreifen.

Mathematisch läßt sich dies durch ein n-Tupel der Breiten *bi* anschreiben, wobei die *bi* absteigend sortiert sind. Ein *positives bi* bezieht sich auf eine Position links der Mittelachse. Umgekehrt liegt eine Übergangsposition mit *negativem bi* rechts der Achse.

Die *Grundbedingung* für die Tupel der *bi* eines Straßensegments ist, daß diese an Anfangs- und Endquerschnitt in ihrer Anzahl übereinstimmen.

Änderungen im Straßenquerschnitt:

Die Einbeziehung von Abbiegespuren, Parkstreifen, Grüninseln usw. erfordert eine Umstrukturierung des Straßenquerschnitts zwischen zwei aufeinanderfolgenden Straßensegmenten. Zur Modellierung von Änderungen in der Anordnung der Fahrstreifen auf die Länge der Straße ist gemäß der Grundbedingung für Segmente eine Segmentteilung an der Stelle der Änderung vorzunehmen. Abbildung-17 zeigt einen typischen Fall mit einer Abbiegespur:

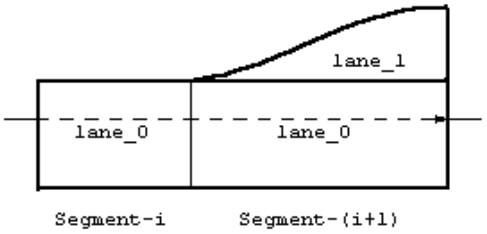


Abbildung 17: Segmentteilung durch Abbiegespur

Man erkennt, daß die durchgehende Fahrspur mit „lane_0“ und die Abbiegespur mit „lane_1“ bezeichnet wurde. Der Grund für die Identifikation der Fahrstreifen mittels einer „lane-Id“ ist die Notwendigkeit der (rechnerischen) Nachvollziehbarkeit der Verlaufs-zuordnung bei Segmentübergängen.

Querschnittsprofil:

Abstufungen im Straßenquerschnitt, wie er etwa bei Randsteinen entsteht, werden bei diesem Modell durch eine Höhendifferenz beschrieben. Die *Grundbedingung* ist, daß die Summe aller relativen Höhenwerte, bezogen auf den gesamten Querschnitt, gleich Null ist und stellt dadurch sicher, daß der rechte- und linke Straßenrand mit dem Höhenfeld übereinstimmt. Dabei sind die hi die relativen Höhendifferenzen zwischen zwei (im Querschnitt von links nach rechts) aufeinanderfolgenden Fahrstreifen und n die Anzahl der Fahrstreifen. Abbildung-18 zeigt das Querprofil einer 2-bahnigen Straße mit Mittelinsel und beidseitigen Gehsteigen und einem Radweg auf der rechten Seite zwischen Fahrbahn und Gehsteig:

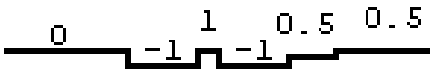


Abbildung 18: Querschnittsprofil mit Abstufungen

Es gilt, daß die korrespondierenden Höhenwerte von zwei aufeinanderfolgenden Segmenten in den Querschnittsprofilen nicht übereinstimmen müssen. Im Falle einer solchen Höhenwertdifferenz wird diese zwischen den Querschnitten interpoliert. Dadurch werden u.a. lokale Absenkungen des Randsteins möglich. Allerdings gilt hier durch die Verwendung von relativen Höhenwerten als grundsätzliche Einschränkung, daß die Fläche eines Fahrstreifens stets in Abhängigkeit vom Terrain liegt und z.B. Optionen für die Wölbungen der Fahrbahn oder Steilkurven nicht mit einbezogen wurden.

Kreuzungen und Einmündungen:

Aus der Sichtweise des Straßengraphen findet man bei Überschneidung zweier Kanten eine *Kreuzung* und bei Berührung eines Endpunktes einer Kante mit einer anderen Kante eine *Einmündung*. Als Voraussetzung gilt, daß jede Kreuzung oder Einmündung die Straße in zwei Hälften teilt. Ausgangsbasis für die Konstruktion der Kreuzung sind die Querschnitte der an die Kreuzung angrenzenden Straßen. Eine *VRMG*-Kreuzung ist somit durch folgende Aspekte definiert:

- Anzahl der beteiligten Straßen(Knotenpunktsarme) größer gleich 2.
- Durch die im Gegenuhrzeigersinn geordneten, zulaufenden Straßen.
- Für jede dieser Straßen wird angegeben, ob die Straße an der Kreuzung beginnt oder endet.

Abbildung-19 zeigt als Beispiel einen Straßengraphauszug mit gerichteten Straßenkanten, einer Einmündung und einer Kreuzung:

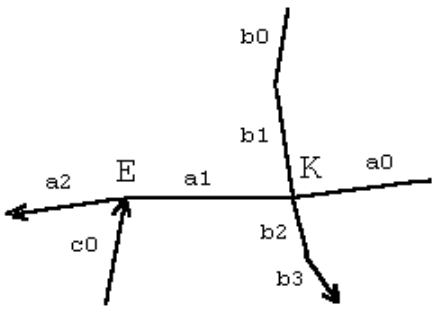


Abbildung 19: Straßengraphauszug mit (E)inmündung und (K)reuzung

Wie aus der Abbildung ersichtlich, besteht die Straße „a“ aus 3 Teilen. Straße „b“ besteht a priori aus einem 3-teiligen Polygonzug, wobei der Mittelteil durch Überkreuzung(„K“) mit Straße „a“ in 2 Teile zerlegt wird. Straße „c“ berührt Straße „a“ am dritten Polygonpunkt und impliziert eine Einmündung(„E“). Die Beschreibung von „E“ wäre per Definitionem durch (a1-end, a2-start, c0-end) gegeben.

Der Grundaufbau für die Repräsentation einer *VRMG*-Kreuzung durch Polygone hat folgende Struktur:

- *Fahrbahnteil*: dieses Polygon wird direkt aus den Querschnittsflächen der zulaufenden Fahrbahnen gebildet.

- *Abrundungsteil*: alle (Querschnitts-)Komponenten der Straßen, die außerhalb der Fahrbahn liegen, wie Randsteine, Radwege und Gehwege, zählen zum Abrundungsteil, da diese von je zwei paarweise benachbarten Knotenpunktsarmen kombiniert und im Winkel des Aufeinandertreffens gebogen werden.
- *Inselteil*: im Fahrbahnteil können ein oder mehrere Verkehrsteiler(„Inseln“) angebracht werden, die sich typischerweise durch eine Erhöhung vom Fahrbahnteil abheben.

Abbildung-20 illustriert die räumliche Anordnung dieser Grundkomponenten:

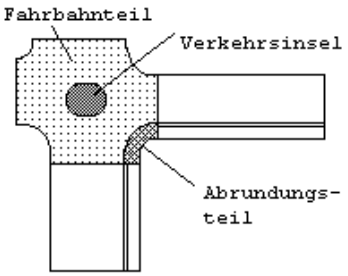


Abbildung 20: Die 3 Grundbausteine einer Kreuzung.

Für die genaue Beschreibung der Kreuzung ist es notwendig, die an die Kreuzung angrenzenden Straßenquerschnitte zu untersuchen. Grundlage der bogenförmigen Abrundung sind die äußeren Begrenzungspunkte zwischen Fahrbahnrand und Randstein. Der *Fahrbahnteil* ist daher jene Fläche, die sich (im Gegenuhrzeigersinn) aus der Sequenz Straßenquerschnitt-i, Bogenkurve-i mit i=0..n-1 (n..Anzahl der Knotenpunktsarme) ergibt. Falls vorhanden, werden die Flächen der Verkehrsinseln aus dieser Fläche ausgeschnitten. Der *Abrundungsteil* folgt der Bogenkurve und variiert nach dem Aufbau der Querschnitte von den (im Gegenuhrzeigersinn) angeordneten Straßen. In diesem Zusammenhang lassen sich folgende Fälle für den Abrundungsteil herausheben:

- *T-Typ*(„Einmündung“): für jene Straße, die durch die Einmündung nicht abgebrochen wird, werden alle Teile(Randstein, Gehweg, Radweg usw.) an der Gegenüberliegenden Seite der Einmündung logisch weitergeführt.

- *N-Typ*(„Normalfall“): der Abrundungsteil besteht nur aus Randstein plus einer Fläche zwischen Randstein und Außenrand des Gehweges.

- *P-Typ*(„Polyline“): zur Anpassung an Häuserfronten oder an beliebige Verlaufsformen wird der Abrundungsteil durch eine Polyline beschrieben. Die Polyline kann optional für jede Übergangskante zwischen zwei Abrundungsteilen(z.B. Straße – Randstein) definiert werden.

Abbildung-21 zeigt die möglichen Fälle bei einer Einmündung und einer Kreuzung auf:

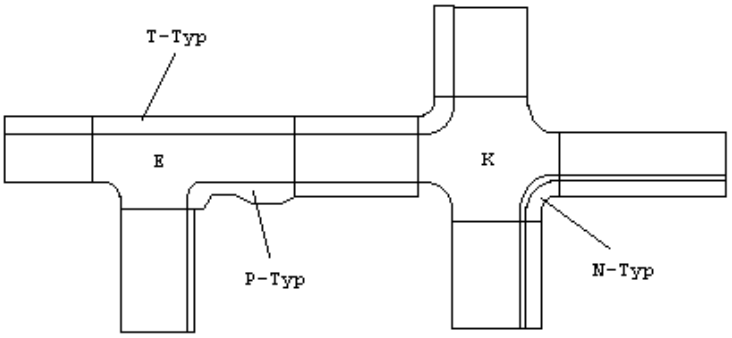


Abbildung 210: Abrundungsfälle für (E)inmündung und (K)reuzung.

Der *Abrundungsteil* entfällt, wenn mindestens eine der beiden benachbarten Straßen auf der entsprechenden Seite nur durch den Fahrbahnrand begrenzt wird, auf Abbildung-21 tritt dieser Fall z.B. bei der rechten oberen Kreuzungsecke auf. Untersucht man den Abrundungstyp „T-Typ“, so läßt sich dieser auf dieselbe Weise wie ein Straßensegment modellieren, wobei dieses Segment alle Komponenten außerhalb der Fahrbahn enthält und die Mittelachse am äußeren Fahrbahnrand angenommen wird.

Für den *Abrundungsbogen* stellt *VRMG* 3 Varianten zur Auswahl:

- *Kreisbogen:* Standardfall der Abrundung, der auch in der Straßenbaupraxis verwendet wird.
- *S-Kurve:* diese Form verbindet zwei annähern parallele, aufeinander zulaufende Straßen.
- *Gerade:* die Bezugspunkte der äußeren Straßenränder werden durch eine einfache Gerade verbunden.

Obwohl alle 3 Varianten unterstützt werden, hat der Kreisbogen die höchste Priorität, wobei der Radius spezifiziert werden kann. Die S-Kurve ist eher ein Spezialfall, der statt 2 entgegengesetzt gekrümmten Kreisbögen angewandt wird. Schließlich wird die Gerade verwendet, wenn die vorherigen beiden Varianten rechnerisch nicht anwendbar sind. Abbildung-32 gibt einen Überblick über die 3 Möglichkeiten:

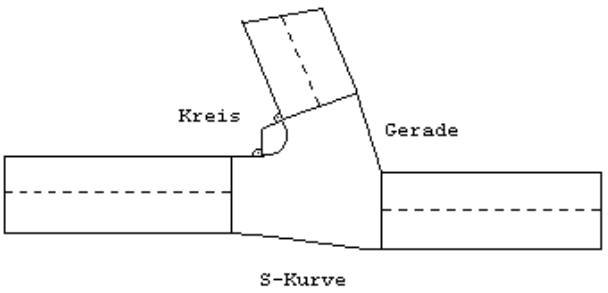


Abbildung 22: die 3 Abrundungsvarianten Kreis, S-Kurve und Gerade.

4.4.) VRMG – Ausgabe (Modul *WriteGeometry*):

Die Ausgabestufe umfaßt den Aufbau eines Szenegraphen und Speicherung dieser Struktur in einem *OpenInventor*-File. Dabei werden das Terrain und die Straßen in getrennten Szenegraphen gespeichert. Für die Straßen kann optional eine Aufteilung der Ausgabe nach Teilbäumen aus dem gesamten Szenegraphen der Straßen erfolgen. Auf diese Details wird weiter unten im Kapitel 6. näher eingegangen.

Im Prinzip ist das *VRMG*-Straßenmodell durch alle in diesem Kapitel vorgestellten Grundkonzepte in abstrakter Form realisierbar. Durch Kombination der Grundformen lassen sich zahlreiche, komplexe Straßennetze modellieren. Die detaillierte Beschreibung der geometrischen Berechnungsschritte wird im nun folgenden Kapitel ausgeführt.

5.) Geometrische Algorithmen:

Die detaillierte Beschreibung der Geometrieberechnungen ist Inhalt dieses Kapitels, das in die 3 thematischen Abschnitte Höhenfeld, Straßen und Optimierung der Geometriedaten unterteilt ist.

5.1.) Höhenfeld:

Ausgehend von den im Kapitel 4.2 vorgestellten Höhenfeldmodell sei ein *(m,n)*-Höhenfeld *H* im folgenden durch eine geordnete Sequenz von Punkten definiert, wobei *m* die Anzahl der Zeilen und *n* die Anzahl der Spalten ist und durch die Funktion *f* das Indexpaar*(i,j)* von einem 2D-Raster auf ein 1D-Feld abgebildet wird:

$$f(i,\ j) = i \cdot m + j \qquad i = 0..m-1,\ j = 0..n-1.$$

Die Dimension einer Rasterzelle wird mit *u,v* bezeichnet. Jeder (Raster-) Punkt *P* aus *H* wird durch die Position *(Px,Py,Pz)* und den Normalvektor *np* charakterisiert. Jedem *P* ist umkehrbar eindeutig ein Indexpaar*(i,j)* zugeordnet:

$$i = y / v,\ j = x / u$$

unter der Voraussetzung, daß der Ursprung des Rasters in *(0,0)* liegt, anderenfalls ist für obige Formel zur *i,j*-Berechnung für *P* eine Translation um den (Rasterursprungs-)Vektor *vc* nötig:

$$i = (y - y_c) / v,\ j = (x - x_c) / u$$

Die Funktion *P(i,j)?R²* bildet jeden indizierten Rasterpunkt auf seinen entsprechenden *(x,y)*-Vektor ab.

Die Höhenwertfunktion *h(x,y): R²->R* berechnet zu jedem *Q* aus *H* den entsprechenden Höhenwert durch *bilineare Interpolation*:

$$h(Q_x, Q_y) = (1-\Delta y)\cdot\left[(1-\Delta x)\cdot h(P(i,\ j)) + \Delta x \cdot h(P(i,\ j+1))\right] + \Delta y \cdot\left[(1-\Delta x)\cdot h(P(i+1,\ j)) + \Delta x \cdot h(P(i+1,\ j+1))\right]$$

mit $\Delta x = Q_x - P(i,\ j)_x,\ \ \Delta y = Q_y - P(i,\ j)_y.$

für jeden Rasterpunkt gilt:

$$h(P(i, j)) = P_z$$

Die Berechnung des Normalvektors *np* eines Rasterpunktes *P* wird mit der *Zentraldifferenz-Methode* durchgeführt und von den resultierenden Vektoren in *x*- und *y*- Richtung das Kreuzprodukt gebildet:

$$np(P) = \begin{pmatrix} -v \cdot \partial z_x \\ -u \cdot \partial z_y \\ +u \cdot v \end{pmatrix} \qquad \text{mit} \qquad \begin{pmatrix} \partial z_x \\ \partial z_y \end{pmatrix} = \begin{pmatrix} h(P(i, j+1)) - h(P(i, j-1)) \\ h(P(i+1, j)) - h(P(i-1, j)) \end{pmatrix}$$

Die Berechnung des Normalvektors *nq* von *Q* kann mittels *bilinearer Interpolation* über alle 3 Koordinaten von den Normalvektoren der 4 umliegenden Rasterpunkte erfolgen:

$$nq(Q) = (1-\Delta y) \cdot [(1-\Delta x) \cdot n(P(i, j)) + \Delta x \cdot n(P(i+1, j))] + \\ \Delta y \cdot [(1-\Delta x) \cdot n(P(i+1, j)) + \Delta x \cdot n(P(i+1, j+1))]$$

mit $\Delta x = Q_x - P(i, j)_x, \quad \Delta y = Q_y - P(i, j)_y.$

Um die Auflösung von *H* zu verändern, wird *H* mit den Werten *u'*,*v'* der veränderten Zellgröße abgetastet:

$$P'(i', j') = \begin{pmatrix} x_c + j' \cdot u' \\ y_c + i' \cdot v' \end{pmatrix} \qquad i' = 0..m'-1,\ j' = 0..n'-1 \qquad P'_z = h(P'(i', j'))$$

mit den Normalvektoren:

$$np' = np(P'(i', j')) \qquad i' = 0..m'-1,\ j' = 0..n'-1$$

Die Ausdehnung von *H* in *x*- und *y*-Richtung bleibt konstant, es gilt:

$$v \cdot n = v' \cdot n',\ u \cdot m = u' \cdot m'$$

<u>Texturkoordinaten der Höhenfeldpunkte</u>:

Per definitionem stellt das Höhenfeld im Rahmen von *VRMG* die polygonalen Restflächen dar. Aus diesem Grund kann es praktisch auch nur als Ganzes mit einer *globalen Textur* versehen

werden. Für die Texturkoordinaten werden die *(x,y)*-Werte als Ausgangsbasis verwendet, wobei die Funktion $gt: R^2 -> R^2$ die Transformation durchführt:

$$(u, v) = gt(x, y)$$
$$u = sz \cdot x, \quad v = sz \cdot y$$

Der Parameter *sz* heißt *Minifikationsfaktor* und paßt die Texturgröße an das Modell an. Abbildung-23 zeigt das Mapping bei einem Minifikationsfaktor von *sz=0.5*:

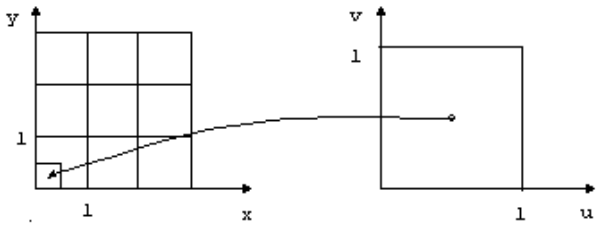


Abbildung 23: Raster-Grundriß (links) und Texturkoordinatensystem (rechts) mit sz=0.5.

5.2.) Straßen:

Die mathematische Definition des Straßengraphen ist folgende :

Der *Graph G* besteht aus den Mengen *V(G)*, den Knoten von *G*, und *E(G)*, den Kanten von *G*, mit einer Funktion *f*, die jedem Element aus *E(G)* ein geordnetes Paar von Elementen aus *V(G)* zuordnet. Ist *f(e)=<x,y>*(geordnetes Paar), so heißt *x Anfangspunkt* und *y Endpunkt.* Anfangs- und Endpunkt sind über die Kante *e* benachbart oder adjazent.

Überträgt man die abstrakte mathematische Definition auf den konkreten Fall des Straßennetzes, so ergeben sich folgende Analogien:

- *G = SN* (Straßennetz)
- *V(G) = K(SN) U M(SN)* (Knotenpunkte und Mittelpunkte einer Straße *S*)
- *E(G) = T(SN)* (Menge aller Straßensegmente)
- *f = S(SN) (*Menge der Straßentupel *S={<M0,..,Mk> | Mi¹Mj, mit 0<=i,j<=k,i¹j}* impliziert *T(SN)={<Mi,Mi+1> | i=0..k-2,* für alle Straßen aus *S*, wobei *k* der Anzahl der Straßensegmente entspricht*}*)

Für die Menge der *Straßensegmente T* muß die Definition um die Menge *ST* aller *Segmenttypen st* erweitert werden, die *ti* beschreiben dabei den Kurventyp des *i*-ten Segments:

$$ST = \{st_0,...,st_{k-1}\}, \qquad T(SN) = \{< M_0,...,M_n >\} \cup \{t = (t_0,...,t_n) \mid t_i \in ST, 0 \le i \le n\}$$

Die Menge *STK* gibt für eine (geordnete) Menge von Segmenttypen *ST* zu jedem *stj* aus *ST* die Menge kompatibler Segmenttypen für den Kontext von *stj* an:

$$STK = \{STK_0,...,STK_{k-1}\},\ STK_i = \bigcup_{j=0}^{l} st_j \qquad i = 0..k,\ 0 \le l < |ST|$$

Zu jedem Element aus *ST* kann eine *Randbedingung* angegeben werden, die den Kontext für den Typ des *i*-ten Straßensegmentes *ti* auswertet:

$$t_i \;\; gültig \Leftrightarrow (t_{i-1} \in STK_j) \wedge (t_{i+1} \in STK_j) \qquad 0 < i < n.$$

dabei entspricht *ti* dem Segmenttyp *stj* und daher *STKj* den zu *stj* kompatiblen Segmenttypen.

Beispiel:

Es gibt 3 Segmenttypen: Gerade, Kreis und Parabel. Ein gerades Segment kann beliebige Nachbarsegmente haben, jeder Kreis muß von Geraden- oder Parabel-stücken eingeschlossen sein, um die Richtung der Tangenten festzulegen, jede Parabel muß sich zwischen Geraden oder Kreisen oder einer Kombination von beiden befinden:

ST={„gerade“,“kreis“,“parabel“},
STK={{true,true},{„gerade“,“parabel“},{„gerade“,“kreis“}}

Für Menge der *Knotenpunkte K* ist *f* die Zuordnungsvorschrift für die Menge aller Paare *<Mi,Kj>* mit *i=0..nj-1,j=0..|K|-1*, wobei *nj* gleich der Anzahl der Knotenpunktsarme der j-ten Kreuzung sind. Für die *Mi* gilt als Zusatzbedingung, daß diese um das Zentrum *Kj* im Gegenuhrzeigersinn geordnet sind:

$$j_i < j_{i+1}, i = 0..n_j - 1, \; j = 0..|K| - 1$$

mit der relative Lage der *Mi* zu *Kj* in Polarkoordinaten durch *(ri,fi)*.

Zur Beschreibung der Fläche eines Straßensegmentes wird die Definition von *T* um folgende Parameter erweitert, es gilt die in (4.3.2) angegebene „von links nach rechts“ Ordnung für die *n*-Tupel an den Segmentquerschnitten, *st* entspricht dem Segmenttyp:

$$T(SN) = \{st, < M_i, M_{i+1} >, B, H, MD, t, g, PL, TY\}$$

Startpunkt und Endpunkt der Mittelachse:

$$< M_i, M_{i+1} >, \quad M_i, M_{i+1} \in R^3$$

Querschnittsbreiten der Fahrstreifen:

$$B = (b_0, .., b_{n-1}), \quad b \in R$$

relative Höhenwerte der Fahrstreifen:

$$H = (h_0, .., h_{n-1}), \quad h \in R$$

Status des Querschnittsaufbaus:

$$MD = (md_0, .., md_{n-1}), \quad md \in \{c, e, s\}$$

Tangente im Start- und Endpunkt:

$$t = (t_0, t_1), \quad t_0, t_1 \in R^2$$

Winkel zwischen *T* und *Ti-1* bzw. *T* und *Ti+1*:

$$g = (g_0, g_1), \quad g_0, g_1 \in R$$

Polygonzüge der Fahrstreifenkanten:

$$PL = (PL_0,...,PL_{n-1}) \cup \{\ \}$$
$$PL_i = (P_0,...,P_{n-1}),\quad P_i \in R^3$$

Verwendungszweck des Fahrstreifens:

$$ty = ty_0,..,ty_{n-2},\quad ty \in Typ$$
$$Typ = \{´pkw´,´bike´,´parking´,´island´,´sidewalk´\}$$

Polygone der Fahrstreifen eines Straßensegments:

Um aus den angegebenen Breiten *B* und Höhen *H* des *l*-ten Querschnitts die Tupel der Breiten *Qs* bzw. *Qe* sowie die Tupel der Höhen *Hs* bzw. *He* für den Start- und Endquerschnitt abzuleiten, muß der Parameter *MD* herangezogen werden: für eine neue(wegfallende) Fahrspur tritt der Breitenwert des Punktes im Querschnitt, von dem die Änderung ausgeht, zweimal hintereinander auf. Für die Gültigkeit des *k*-ten Breitenwertes bezüglich der beiden Segmente links und rechts vom Querschnitt gilt folgendes:

$$md_{l,k} = \begin{cases} ´c´ & b_{l,k} : Segment(l-1),\ Segment(l) \\ ´e´ & b_{l,k} : Segment(l) \\ ´s´ & b_{l,k} : Segment(l-1) \end{cases}$$

Da die Höhenwerte stehts mit den Breitenwerten korrelieren, ist an Stellen mit doppelt definierten Breiten auch der Höhenwert doppelt definiert und muß daher in Äbhängigkeit vom Kontext jeweils einmal auf 0 gesetzt werden. Die Gültigkeit des *k*-ten Höhenwertes im Querschnitt für die angrenzenden Segmente beschränk sich auf folgende Fälle:

$$md_{l,k} = \begin{cases} ´c´ & Segment(l-1), Segment(l) \\ ´e´ & Segment(l),\ falls(b_{l,k} = b_{l,k+1})\ oder\ (b_{l,k-1} = b_{l,k}) \\ ´s´ & Segment(l-1),\ falls(b_{l,k} = b_{l,k+1})\ oder\ (b_{l,k-1} = b_{l,k}) \end{cases}$$

Daraus resultieren die Tupel der Breiten *Qs* bzw. *Qe* und Höhen *Hs* bzw. *He* für den Anfangs- und Endquerschnitt eines Straßensegments:

$$Qs = (bs_0,..,bs_s),\ Hs = (hs_0,..,hs_s)$$
$$Qe = (be_0,..,be_e),\ He = (he_0,..,he_e)$$

Seien *Qs,Qe* die Start-, und Endquerschnitte, El und Er die linke und rechte Kante, *Pe* die Eckpunkte des i-ten Fahrstreifens, dann gelten folgenden Eigenschaften:

$$Pe_{i,0} \in Qs, Pe_{i,0} \in El_i \qquad Pe_{i,1} \in Qs, Pe_{i,1} \in Er_i$$

$$Pe_{i,2} \in Qe, Pe_{i,2} \in El_i \qquad Pe_{i,3} \in Qe, Pe_{i,3} \in Er_i$$

Für die *x,y*-Werte der *Pe* gilt außerdem:

$$Pe_{i,1} = Pe_{i+1,0}, i = 0..n-2 \quad \text{bzw.} \quad Pe_{i,3} = Pe_{i+1,2}, i = 0..n-2$$

die entsprechenden Höhenwerte sind:

$$Pe_{i+1,0}z = Pe_{i,1}z + hs_{i+1}, i = 0..n-2 \quad \text{bzw.} \quad Pe_{i+1,0}z = Pe_{i,1}z + he_{i+1}, i = 0..n-2$$

wobei *hs* und *he* die relative Höhendifferenz zwischen i- und i+1.ten Fahrstreifen ist am Anfang bzw Ende des Segments ist.

Die absolute Lage der Eckpunkte eines Straßensegments kann nun durch folgende Formeln errechnet werden:

$$Pe_{i,k} = M_s + (-1)^k \cdot b_{i,k} \cdot n_s + (-1)^s \cdot b_{i,k} \cdot t_s \cdot \tan\left(\frac{g_s}{2}\right)$$
$$i = 0..n-2, k \in \{0,1,2,3\}, s \in \{0,1\}, b_{i,k} \in R, g \in \left[0..\pm\frac{p}{2}\right]$$

dabei ist *s=0* für den Startquerschnitt und *s=1* für den Endquerschnitt des Segments. Der Parameter *t* indiziert in Analogie zu obiger Definition den Eckpunkt. Der Vektor *n* ist der zu den Tangentenvektoren *t0* bzw. *t1* gehörende Normalvektor und gibt die Richtung der Mittelachse in Start- oder Endpunkt an. Schließlich wird noch der Winkel *y* für die Richtungsänderung zwischen aufeinanderfolgenden Segmenten benötigt(positiver Wert für Rechtsknick, sonst negativ). Die *bi* sind die den Eckpunkten zugeordneten Breitenwerte.
Abbildung-24 zeigt ein Beispiel mit 3 aufeinanderfolgenden Geradenstücken, diese treffen im Winkel *y1* bzw. *y2* aufeinander, man erkennt, daß die Tangenten *t0* und *t1* in *M0* und *M1* identisch sind, da es sich um ein Geradenstück handelt:

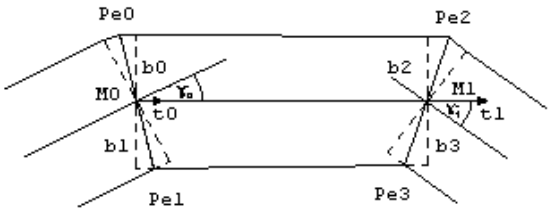


Abbildung 24: Parameter eines beidseitig geknickten Geradenstückes.

Für *Geradenstücke* mit einem großen Länge : Breite – Verhältnis ist es sinnvoll, die Fläche durch folgende Formel feiner zu unterteilen:

$$(1.1) \qquad P_{j,k} = Pe_{i,j} + i \cdot \frac{p}{n}; k = 0..n, j \in \{0,1\}, p = \vec{P}e_{i,j}, \vec{P}e_{i,j+2}$$

dabei gibt *n* die Anzahl der gewünschten Teile an.
Außer der Verwendung von *absoluten Koordinaten* ist es nützlich, zusätzlich ein *lokales Koordinatensystem* festzulegen, sodaß jeder Punkt relativ zu einem bestimmten Segement durch

ein Parameterpaar *(a,b)* lokalisierbar ist. Für diesen Zweck wird eine Funktion *gl: R² x R²->R²* definiert, die die Abbildung durchführt:

$$gl(x1, y1, x, y) \rightarrow (a, b)$$

$$P = \begin{pmatrix} x \\ y \\ z \end{pmatrix}, PB = \begin{pmatrix} x1 \\ y1 \end{pmatrix}$$

$$p = \vec{PB}, \vec{S}; q = \vec{S}, \vec{P}; S = g_1(PB, t_0) \cap g_2(P, nv_0).$$

$$a = \|p\| \cdot (-1)^m; m = 0 \Leftrightarrow p \cdot t_0 > 0, sonst : m = 1.$$

$$b = \|q\| \cdot (-1)^n; n = 0 \Leftrightarrow q \cdot nv_0 > 0, sonst : n = 1.$$

damit wird für einen beliebigen Punkt *P* die relative Lage zu einem festgelegten Punkt *PB* des Straßensegments berechnet. Der Punkt *S* bezeichnet den Fußpunkt des Normalvektors *nv0* auf die Tangente *t0* im Startpunkt, auf dem der Punkt *P* liegt. Man erkennt aus der Formel, daß für ein *P* links der Mittellinie *b* positiv ist und ein positives *a* bedeutet, daß *P* (in Rechenrichtung der Straße) hinter *PB* liegt.
An dieser Stelle ist es notwendig, die Begriffe *vor/hinter* und *links/rechts* bezüglich eines Straßensegmentes zu definieren:

P liegt *vor Q* ⇔ *b(P)<b(Q)*
P liegt *links* ⇔ *a(P)>0*

Die Umkehrabbildung von *gl:R²xR²->R²* sieht folgendermaßen aus:

$$gl^{-1}(x1, y1, a, b) \rightarrow (x, y)$$

$$P = PB + t_0 \cdot a + b \cdot nv_0$$

Abbildung-25 zeigt die Zusammenhänge anhand einer Grafik:

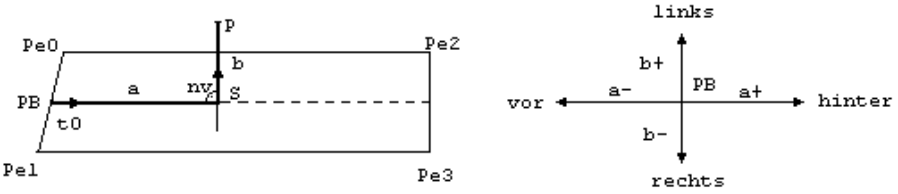


Abbildung 25: Lokale und globale Koordinaten (links) und Orientierung (rechts).

Kreisbogenstücke können durch die Erweiterung um die Parametermenge *{M,r}* definiert werden. Dabei bezeichnet *M* den Mittelpunkt des Bogens für die Straßenachse, *r* ist der dazugehörige Radius, die Berechnung aus den Ausgangsparametern für ein Straßensegment erfolgt durch Schnitt der Geraden g1 und g2, die normal auf die Achsen der den Kreisbogen umschließenden Segmente stehen und durch die Mittelpunkte *M0* bzw. *M1* verlaufen:

$$g1: x = M_0 + l \cdot nv_0, g2: x = M_1 + l \cdot nv_1$$

$$M = g1 \cap g2, \quad r = \|\vec{M}, \vec{M}_0\|.$$

mit den Normalvektoren *nv0* und *nv1* auf die Tangenten *t0* und *t1*. Folgende weitere Kenngrößen sind ableitbar:

- die Winkel a in *M0* und ß in *M1*, bezogen auf den Einheitskreis.
- die Richtung *cw* e {0,1} für den Umlaufsinn: *cw=1*⇔ a>ß.

Für die Verengung und Verbreiterung von Fahrstreifen werden pro Übergangskante zwischen benachbarten Fahrstreifen die Zusatzparameter $M´,P´$ und $r´$ benötigt, es gelten die Bedingungen:

(Bedingung-1: Anfangs- und Endbreite des Fahrstreifens ist gleich)

$$\left\|\vec{P}e_{i,j},\vec{M}\right\| = \left\|\vec{P}e_{i,j+2},\vec{M}\right\|; M´= M; r´= r; i = 0..n-2,\ j \in \{0,1\}$$

$$r´= \min\left(\left\|\vec{M},\vec{P}e_{i,j}\right\|,\left\|\vec{M},\vec{P}e_{i,j+2}\right\|\right)$$

(Bedingung-2: Endbreite des Fahrstreifens ist größer als die Anfangsbreite)

$$\left\|\vec{P}e_{i,j},\vec{M}\right\| < \left\|\vec{P}e_{i,j+2},\vec{M}\right\|; i = 0..n-2,\ j \in \{0,1\};$$

$$M´= \vec{P}e_{i,j} - \frac{\left(\vec{M},\vec{P}e_{i,j+2}\right)\cdot r´}{\left\|\vec{M},\vec{P}e_{i,j+2}\right\|},$$

$$P´= \vec{M}´+\frac{\left(\vec{M},\vec{P}e_{i,j+2}\right)\cdot r´}{\left\|\vec{M},\vec{P}e_{i,j+2}\right\|},$$

(Bedingung-3: Anfangsbreite des Fahrstreifens ist größer als die Endbreite)

$$\left\|\vec{P}e_{i,j},\vec{M}\right\| > \left\|\vec{P}e_{i,j+2},\vec{M}\right\|; i = 0..n-2,\ j \in \{0,1\};$$

$$M´= \vec{P}e_{i,j+2} - \frac{\left(\vec{M},\vec{P}e_{i,j+2}\right)\cdot r´}{\left\|\vec{M},\vec{P}e_{i,j+2}\right\|},$$

$$P´= \vec{M}´+\frac{\left(\vec{M},\vec{P}e_{i,j}\right)\cdot r´}{\left\|\vec{M},\vec{P}e_{i,j}\right\|},$$

Abbildung-26 zeigt die 3 möglichen Fälle:

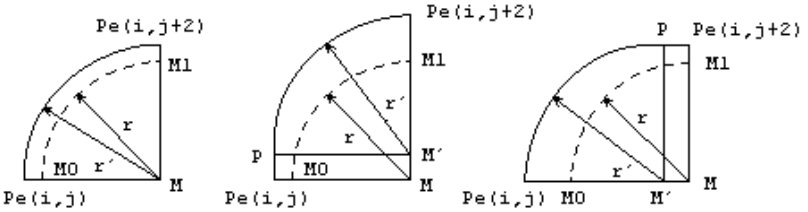


Abbildung 26: Mögliche Formen der Übergangskanten für Bed.1 (links), Bed.2 (mitte), Bed.3 (rechts).

Da die kreisbogenförmige Segmentfläche in eine polygonale Form gebracht werden muß, erfolgt eine Segmentzerlegung in mehrere, aufeinanderfolgende Geradenstücke durch Approximation. Grundlage für die Verfeinerung ist die Unterteilung der Mittelachse des Kreisbogens in *t* gleiche Abschnitte. Für Verbreiterungen und Verengungen, für die Bedingung I oder II gilt und *P* definiert ist, erstreckt sich die Approximation sowohl über den Kreisbogen der Übergangskante, als auch das gerade Ausgleichsstück. Die Berechnung der Ausgangswinkel a, ß und dem Differenzwinkel d erfolgt durch die Formeln:

$$a = \tan^{-1}\begin{pmatrix} M_0 y - M_y \\ M_0 x - M_x \end{pmatrix}$$

$$b = \tan^{-1}\begin{pmatrix} M_1 y - M_y \\ M_1 x - M_x \end{pmatrix}$$

$$d = b - a; g = d / n,$$

Ausgehend von Start- und Endquerschnitt und den Winkeln wird die Iteration durchgeführt:

(1.2)

$$for(k = 0; k \le n; k++)$$

$$M_k = M + r \cdot \begin{pmatrix} \cos a + k \cdot g \\ \sin a + k \cdot g \end{pmatrix}$$

$$if\,(Bed.1) P_{j,k} = \vec{M}_k + \left\| \vec{M}, \vec{P}e_{i,j} \right\| \cdot \frac{\left(\vec{M}, \vec{M}_k\right)}{\left\| \vec{M}, \vec{M}_k \right\|}; j \in \{0,1\};$$

$$if\,(Bed.2)$$

$$if\,(P_{j,k} \in \overline{Pe_{i,j}, P})$$

$$P_{j,k} = \vec{P}e_{i,j} + \frac{\left(\vec{P}e_{i,j}, \vec{P}\right) \cdot r' \tan g}{\left\| \vec{P}e_{i,j}, \vec{P} \right\|}$$

else

$$P_{j,k} = \vec{M} + \left(\vec{M}, \vec{M}_k\right) \cdot \frac{\max\left(l \cdot \cos r, l \cdot \cos r - \sqrt{(l \cdot \cos r)^2 + r'^2 - l^2} \right)}{l}; l = \left\| \vec{M}, \vec{M}_k \right\|; r = d - k \cdot g;$$

$$if\,(Bed.3)$$

$$if\,(P_{j,k} \in \overline{P, Pe_{i,j+2}})$$

$$P_{j,k} = \vec{P}e_{i,j+2} + \frac{\left(\vec{P}e_{i,j+2}, \vec{P}\right) \cdot r' \cdot \tan(d - k \cdot g)}{\left\| \vec{P}e_{i,j+2}, \vec{P} \right\|}$$

else

$$P_{j,k} = \vec{M} + \left(\vec{M}, \vec{M}_k\right) \cdot \frac{\max\left(l \cdot \cos r, l \cdot \cos r - \sqrt{(l \cdot \cos r)^2 + r'^2 - l^2} \right)}{l}; l = \left\| \vec{M}, \vec{M}_k \right\|; r = k \cdot g$$

Die Funktion zur Transformation von globalen in lokale Koordinaten $gl: R^2 x\ R^2 -> R^2$ ist für Bogenstücke sinngemäß durch Verwendung der Bogenlänge für die Parameter *(a,b)* definiert, wobei die Berechnung in Abhängigkeit der Bedingungen 1-3 erfolgt:

$$if\,(Bed.1)$$
$$a = r \cdot b; b = \left(\left\|\vec{M}, \vec{M}_0\right\| - r\right) \cdot (-1)^{cw+1}$$

Im Fall 2-3 ist zu bestimmen, ob der Bezugspunkt(Schnittpunkt der Strecke M,P mit der referenzierten Übergangskante) auf dem geraden Ausgleichsstück oder dem Kreisbogen liegt, dazu wird der Öffnungswinkel zu P bestimmt und bei der Bestimmung der lokalen Koordinaten mit F verglichen:

$$r = \cos^{-1} \frac{\left(\vec{M}, \vec{Pe}_{i,j}\right) \cdot \left(\vec{M}, \vec{P}\right)}{\left\|\vec{M}, \vec{Pe}_{i,j}\right\| \cdot \left\|\vec{M}, \vec{P}\right\|};$$
$$f = b - a.$$
$$g = \cos^{-1} \frac{\left(\vec{M}', \vec{P}'\right) \cdot \left(\vec{M}', \vec{P}\right)}{\left\|\vec{M}', \vec{P}'\right\| \cdot \left\|\vec{M}', \vec{P}\right\|}.$$

Damit sind die beiden Öffnungswinkel zwischen den Segmentstartpunkten an der i-ten Übergangskante mit dem dem Kreisbogenmittelpunkt *M* sowie der für die Berechnung nötige Winkel *y*, dem Öffnungswinkel zwischen Übergangspunkt vom Kreis zur Ausgleichsgeraden oder umgekehrt mit dem i-ten Bogenmittelpunkt *M´*
bestimmt:

$$if\,(Bed.2)$$
$$if\,(r \langle f)$$
$$S = g1\left(\vec{Pe}_{i,j}, \left(\vec{Pe}_{i,j}, \vec{P}'\right)\right) \cap g2\left(\vec{P}, \left(\vec{M}, \vec{P}'\right)\right); i = 0..n-2;\ j \in \{0,1\};$$
$$a = \left\|\vec{Pe}_{i,j}, \vec{S}\right\| \cdot (-1)^m; m = 0 \Leftrightarrow \left(\vec{Pe}_{i,j}, \vec{S}\right) \cdot \left(\vec{Pe}_{i,j}, \vec{P}'\right); sonst : m = 1;$$
$$b = \left\|\vec{S}, \vec{P}\right\| \cdot (-1)^k; k = 0 \Leftrightarrow \left(\left(\vec{S}, \vec{P}\right) \cdot \left(\vec{M}, \vec{Pe}_{i,j}\right) \cdot (-1)^{cw+1}\right) > 0; sonst : k = 1;$$
$$else$$
$$a = r' \cdot |g| + \left\|\vec{Pe}_{i,j}, \vec{P}'\right\|;$$
$$b = \left(\left\|\vec{M}', \vec{P}\right\| - r'\right) \cdot (-1)^{cw+1}$$
$$if\,(Bed.3)$$
$$if\,(r > f)$$
$$S = g1\left(\vec{P}', \left(\vec{P}', \vec{Pe}_{i,j+2}\right)\right) \cap g2\left(\vec{P}, \left(\vec{M}, \vec{Pe}_{i,j+2}\right)\right); i = 0..n-2;\ j \in \{0,1\};$$
$$a = \left\|\vec{P}', \vec{S}\right\| \cdot (-1)^m + r' \cdot \cos^{-1} \frac{\left(\vec{M}', \vec{Pe}_{i,j}\right) \cdot \left(\vec{M}, \vec{P}'\right)}{\left\|\vec{M}', \vec{Pe}_{i,j}\right\| \cdot \left\|\vec{M}, \vec{P}'\right\|};$$
$$m = 0 \Leftrightarrow \left(\vec{Pe}_{i,j+2}, \vec{S}\right) \cdot \left(\vec{Pe}_{i,j+2}, \vec{P}'\right) > 0; sonst : m = 1;$$
$$b = \left\|\vec{S}, \vec{P}\right\| \cdot (-1)^k; k = 0 \Leftrightarrow \left(\left(\vec{S}, \vec{P}\right) \cdot \left(\vec{M}, \vec{Pe}_{i,j+2}\right) \cdot (-1)^{cw+1}\right) > 0; sonst : k = 1;$$
$$else$$
$$a = r' \cdot |g|;$$
$$b = \left(\left\|\vec{M}', \vec{P}\right\| - r'\right) \cdot (-1)^{cw+1}$$

Die Umkehrabbildung von gl:R^2->R^2 hat folgende Gestalt:

$$m = a + (-1)^{cw} \cdot a; r = \left\| \vec{M}, \vec{P'} \right\|;$$
$$P = \vec{M} + (b + r) \cdot \begin{pmatrix} \cos b \\ \sin b \end{pmatrix}$$

Texturemapping für Segmente:

Grundsätzlich wird für das *Texturemapping* von Polygonen ein 2-Tupel *(u,v)* der 2D-Texturkoordinaten verwendet, wobei *u* und *v* auf dem Ausgangsbild, das als Textur verwendet wird, normiert sind. Für Straßen sind folgende Punkte zu beachten:

- Die Achsen des Texturkoordinatensystems entsprechen der Mittelachse (*v*-Dimension) und dem Normalvektor auf diese (*u*-Dimension).
- (periodische) Fortsetzung einer Textur eines bestimmten Fahrstreifens von einem Straßensegment zum nächsten, sofern keine Änderung in der Bodenmarkierung auftritt. Dieser Aspekt betrifft den Parameter *v*, der inkrementell entlang der Straße berechnet wird.
- Anpassungsmöglichkeit der Textur an Übergangskanten im Querschnitt, wenn dies durch eine Bedingung erfordert wird. Ein Beispiel wäre eine strichlierte Mittellinie zwischen zwei aneinanderliegenden Fahrstreifen. Dies läßt sich dadurch erreichen, daß für die Berechnung von *v* der anzupassenden Fahrstreifen dieselbe Bezugslinie gewählt wird.
- Festlegung des realen Größenwertes der Textur durch einen Parameter *sz*.
- Für den Fall, daß *sz* ungleich der entsprechenden Fahrstreifenbreite ist, gibt es 3 Möglichkeiten für den Parameter *u*: 1.) die linke Texturkante fällt mit dem linken Fahrstreifenrand zusammen, 2.) wie (1) für den rechten Rand, 3.) die Textur wird durch Verzerrung an die Breite des Fahrstreifens angepaßt. Der Referenzierungsparameter wird im folgenden mit *ref* ∈ *{´l´,´r´,´m´}* bezeichnet.

Faßt man diese Aspekte zusammen, so wird erkennbar, daß die weiter oben formulierten *lokalen Koordinaten* zur Berechnung der Texturkoordinaten verwendet werden können. Die Menge *RF={{B0,B1},ref,sz}* enthält die Charakteristika für die Texturkoordinaten des Segments mit der Bezugsachse von B0 nach B1 für den Querschnitt des i-ten Fahrstreifens:

$$B_{i,j} = M_j \Leftrightarrow ref = \acute{}m\acute{}; j = 0,1;$$
$$B_{i,j} = Pe_{i,2*j} \Leftrightarrow ref = \acute{}l\acute{}; j = 0,1;$$
$$B_{i,j} = Pe_{i,2*j+1} \Leftrightarrow ref = \acute{}r\acute{}; j = 0,1;$$

Für die Transformation *lt:RF x R^2->R^2* gelten die Formeln:

(1.2.1)

$$P(a,b) = gl(B_0(x,y), P(x,y)); P \in R^2; x, y, a, b \in R;$$
$$(u,v) = lt(RF_i, a, b); i = 0..n-2;$$
$$u_k = -\frac{b}{sz} \Leftrightarrow ref \in \{l,r\}, sonst: u_k \in \{0,1\} \mid u_k = 0 \Leftrightarrow P \in El_i, sonst: u_k = 1; k = 0..m-1; u_k, b, sz \in R$$
$$v_k = (v_{alt} + a + v_c)/sz; a \in R; v_{alt} = \sum_{l=0}^{k-1} v_l;$$

mit *m* Straßensegmenten und den *(uk,vk)* für den Punkt *P* des *k*-ten Segments, wobei *P* dem *i*-ten Fahrstreifen zugeordnet ist. Der Korrekturwert *vc* wird in Abhängigkeit von der Referenzierung *Ref* und dem oben definierten Winkel *y* berechnet:

$$
\begin{aligned}
&if\,(ref = \text{´}m\text{´})\\
&v_c = \frac{\left\| \vec{M}, \vec{P}e_{i,j} \right\|}{\sin\frac{g}{2}}; i = 0..n-2;\ j = 0 \Leftrightarrow P \in El_i, sonst : j = 1;\\
&else\\
&if\,((ref = \text{´}l\text{´}) \wedge (P \in El_i)) \vee ((ref = \text{´}r\text{´}) \wedge (P \in Er_i))\\
&v_c = 0;\\
&else\\
&v_c = \frac{\left\| \vec{P}e_{i,p}, \vec{P}e_{i,q} \right\|}{\sin\frac{g}{2}}; i = 0..n-2; (p = 0, q = 1) \Leftrightarrow ref = \text{´}l\text{´}, sonst : (p = 1, q = 0);
\end{aligned}
$$

Abbildung-27 zeigt die 3 Möglichkeiten der Referenzierung anhand eines einfachen Geradenstückes, das am Anfang eine Abschrägung aufweist, die Straße verläuft von links nach rechts, sodaß der linke Straßenrand der oberen Kante entspricht. Zu beachten ist die Lage des Ursprungs des lokalen Texturkoordinatensystems in Abhängigkeit des Referenzparameters *ref*:

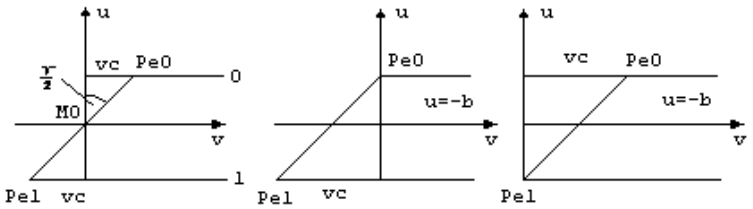


Abbildung27: Textur-Referenzierung mit *ref*=´m´(links), *ref*=´l´(mitte), *ref*=´r´(rechts).

Polyline - Fahrstreifenrand:

Die formale Definition der Polyline für die i-te Übergangskante zwischen zwei Fahrstreifen hat die Gestalt:

$$Pl_i = \{P_0, .., P_l\}; Pl \in R^3; i = 0..n-2; l \in N.$$

Die durch ein solches geordnetes *l*-Tupel angegebene Polyline wird durch die Funktion *tp:S x R³->R³* an die i-te Übergangskante des Straßensegments *S* angepaßt:

$$(1.3)$$

$$Pl' = tp(S, Pl);$$

$$Pl' = \{Pl_0', .., Pl_m'\} |$$

$$\exists p \geq 0, \overline{Pl_p, Pl_{p+1}} \cap \overline{Pe_{i,0}, Pe_{i,1}} = Pl_0';$$

$$sonst : Pl_0' = Pe_{i,0} \Leftrightarrow P \in El_i, Pl_0' = Pe_{i,1} \Leftrightarrow P \in Er_i;$$

$$\exists q \geq p, \overline{Pl_q, Pl_{q+1}} \cap \overline{Pe_{i,2}, Pe_{i,3}} = Pl_m';$$

$$sonst : Pl_m' = Pe_{i,2} \Leftrightarrow P \in El_i, Pl_m' = Pe_{i,3} \Leftrightarrow P \in Er_i;$$

$$Pl_k' \in Pl \cup Pe_{i,j}; k = 0..m; i = 0..n-2; j \in \{0,1,2,3\};$$

Die möglichen Fälle für die Anpassung werden auf Abbildung-28 gezeigt:

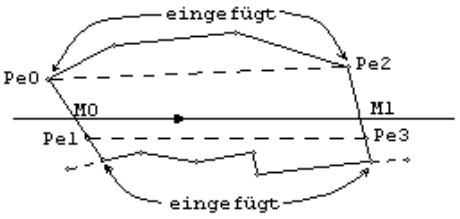


Abbildung 28: Zuschneiden der Polyline am linken Rand (oben) und rechten Rand (unten).

Die Grafik zeigt als Beispiel ein von links nach rechts durch die Achse M0,M1 definiertes Straßensegment, bei dem der linke und rechte Straßenrand durch eine Polyline angegeben ist. Die Polyline *P0={Pl0,Pl1}* des linken Randes besteht aus nur 2 Punkten, sodaß durch Anwendung der Transformation *tp(Pl)* die Polyline *P0´={Pe0,Pl0,Pl1,Pe2}* entsteht. Die Polyline des rechten Randes *P1={Pr0,..,Pr5}* wird auf den durch die Geraden vom Anfangs- bzw. Endquerschnitt gebildeten, gültigen Bereich zugeschnitten, d.h. es werden die Schnittpunkte *S0* und *S1* eingefügt. Die Polyline des rechten Randes nach Anwendung der Funktion ist somit *P1´={S0,Pr1,..,Pr4,S1}.*

Spline – Fahrstreifenrand[33,34,35]:

Für weiche, S-förmige Übergänge an den Fahrbahnränder vom Segmentanfang zum Segmentende, werden die Punkte der Polyline mit einer *Splinefunktion unter Spannung* berechnet. Dadurch läßt sich ein unerwünschtes Oszillieren (Ausschwingen) der Kurve vermeiden. Der Spline wird durch die folgende Parametermenge charakterisiert:

$$SP = \{\{Pe_{i,j}, Pe_{i,j+2}\} \cup P, t, m\}; i = 0..n-2; j \in \{0,1\}; P = \{P_0, .., P_l\} \cup \{\}; P \in R^2; t \in R^+; l, m \in N.$$

dabei ist *j=0* wenn der Spline den linken Rand repräsentiert und j=1 für den rechten Rand. Die Menge *P*, die auch leer sein kann, spezifiziert eine optionale Menge von Stützpunkten. Der Spannungsparameter *t* erstreckt sich global über die ganze Länge der gesuchten Kurve und approximiert für kleine Werte (<=1) die *kubische Splineform* und für große Werte (gegen 8) eine *Polyline*. Der Parameter *m* gibt schließlich die Anzahl der Verfeinerungs-intervalle an.

Die für diesen Zweck verwendete Splinefunktion selbst heißt *Exponentialspline* und ist eine Linearkombination der Basisfunktionen[35]:

$$1, x, \sinh\left(\frac{t \cdot x}{x_{i+1} - x_i}\right), \cosh\left(\frac{t \cdot x}{x_{i+1} - x_i}\right); i = 0..l; l = 2 + |P|; i, l \in N.$$

Die auf dem [0..1]-Intervall durch die Stützstellen in [x(i),x(i+1)] definierte Splinefunktion *u(x)* besitzt folgende Eigenschaft(*Euler-Lagrange Gleichung*):

$$u^{(4)}(x) - \frac{t^2}{(x_{i+1} - x_i)^2} \cdot u^{(2)}(x) = 0.$$

Aus dem Ansatz:

$$(1.4)$$
$$u(x) = A \cdot \left(\frac{x_{i+1} - x}{h_i}\right) + B \cdot \left(\frac{x - x_i}{h_i}\right) + C \cdot \frac{\sinh(t \cdot (x_{i+1} - x_i))}{\sinh(t \cdot h_i)} + D \cdot \frac{\sinh(t \cdot (x - x_i))}{\sinh(t \cdot h_i)};$$
$$h_i = x_{i+1} - x_i;$$

mit den Koeffizienten:

$$A = y_i - \frac{u^{(2)}(x_i)}{t^2}; B = y_{i+1} - \frac{u^{(2)}(x_{i+1})}{t^2}; C = \frac{u^{(2)}(x_i)}{t^2}; D = \frac{u^{(2)}(x_{i+1})}{t^2};$$

wird das tridiagonale, symmetrische Differenzialgleichungssystem mit der Stetigkeitsbedingung in der ersten Ableitung an den Stützpunkten gelöst[33,34]. In Analogie zur Polyline wird durch Anwendung von der Splinefunktion *u(x)* ein Polygonzug *Pli* für die *i*-te Übergangskante zwischen im Straßenquerschnitt benachbarten Fahrstreifen berechnet.

<u>Stufenkanten zwischen Fahrstreifen - Randsteine:</u>

Die Bedingung für die Randsteingenerierung läßt sich folgendermaßen formulieren:

$$(P_j x = Q_j x) \wedge (P_j x = Q_j x) \mid P_j \in Er_i, Q_j \in El_{i+1}, P, Q \in R^3, j = 0..|Er_i| - 1; 0 \le i \le n - 3; i, j \in N.$$
$$(hs_{i+1} \neq 0) \vee (he_{i+1} \neq 0); hs_{i+1} \in Hs, he_{i+1} \in He;$$

d.h. die *(x,y)*-Werte der Randpunkte der benachbarten Fahrstreifen stimmen über die Länge des Segments überein und die Abstufung ist zumindest am Start- oder am Endquerschnitt ungleich Null. Das im Gegenuhrzeigersinn definierte Randsteinpolygon entsteht demnach aus den Randpunkten der benachbarten Fahrstreifen:

$$P = (R_0, .., R_l, P_l, .., P_0); P \in R^3; l = |El_{i+1}| - 1 = |Er_i| - 1;$$

Zur Anpassung der Textur an die beteiligten Fahrstreifen werden die Texturkoordinaten direkt von diesen übernommen, wodurch Übereinstimmung an den Randsteinkanten erreicht wird.

Zusammensetzung einer Straße aus den Segmenten:

Sei *S* eine Straße aus dem Straßennetz *SN* und *T* die (geordnete)Menge von Segmenten aus *S*, dann beschreibt die Parametermenge *PT={id,tex,sz,ref}* die Attribute eines Fahrstreifens mit der (pro Segment) eindeutigen *id* und der Textur *tex* und den dazugehörigen Texturattributen *sz* und *ref*. Der Aufbau der Straße läßt sich als Matrix *LUT* beschreiben:

$$LUT = \begin{pmatrix} pt_{0,0} & . & . & . & pt_{0,n-2} \\ . & & & & . \\ . & & & & . \\ . & & & & . \\ pt_{m-1,0} & . & . & . & pt_{m-1,n-2} \end{pmatrix};$$

$$pt_{i,j} \in PT \cup \{\}; i, j \in N; i = 0..m-1; j = 0..n-2.$$

dabei steht *LUT* für *LookUpTable*, weil die gesamte Information für die Fahrstreifen über die Länge der Straße in dieser gespeichert ist bzw. aus dieser entnommen wird. Der Parameter *n* bezieht sich auf die Anzahl der Mittelpunkte, d.h. die Straße besitzt *n-1* Segmente und der Parameter *m* ist die maximale Anzahl der Fahrstreifen pro Segment auf die Gesamtlänge der Straße. Für die kompakte Speicherung wird folgende Form der *LUT* gewählt:

$$\exists i < m \mid pt_{i,j} \neq \{\}, pt_{i+1,j} = \{\} \forall j = 0..n-2.$$

Die *Fahrstreifenpolygone FP={FP0,..,FPl}* werden aus der *LUT* abgeleitet, die Berechnung erfolgt von links nach rechts und von oben nach unten. Die nach (1.1)-Gerade, (1.2)-Kreis, (1.3)-Polyline, (1.4)-Spline berechneten Polygonpunkte *Pi_0* und *Pi_1* des *j*-ten Segments und *i*-ten Fahrstreifens werden unter folgenden Bedingungen mit jenen des selben (*k*-ten) Fahrstreifens (nach der *id*) im (*j+1*-ten) Nachfolgesegments verbunden:

Bedingung-1: $id(pt_{i,j}) = id(pt_{k,j+1});$

Bedingung-2: $tex(pt_{i,j}) = tex(pt_{k,j+1})$

(1.5)

$$Pi_q = \bigcup Pi_{q,l} \mid ((id(pt_{i,j}) = id(pt_{k,j+1})) \wedge (tex(pt_{i,j}) = tex(pt_{k,j+1}))); 0 \le i,k < m; 0 \le j < n-1;$$
$$p = 0..l; l = |Pi_q|; q \in \{0,1\}$$

Geometrische Daten zur vollständigen Fahrstreifenbeschreibung:

Das *i*-te Fahrstreifenpolygon *FP* besteht aus den im Gegenuhrzeigersinn geordneten Punkten des linken und rechten Randes, die in (1.5) berechnet wurden und den zugeordneten Texturkoordinaten *FPT*, die nach (1.2.1) berechnet wurden:

$$(1.6)$$
$$FP_i = (P_{0,0},...,P_{0,p-1},P_{1,p-1},...,P_{1,0}); P_{j,r} \in R^3, P_{0,r} \in Pi_0; P_{1,r} \in Pi_1; 0 \le r < p; j \in \{0,1\}.$$
$$FPT_i = (PT_{0,0},...,PT_{0,p-1},PT_{1,p-1},...,PT_{1,0}); PT_{j,r} \in R^2; j \in \{0,1\}.$$

Das Polygon *RP* des *Randsteins* zwischen *i*-ten und *i+1*-ten Fahrstreifen wird, wie oben beschrieben, aus den linken Randpunkten *Pii* des *i+1*-ten und den rechten Randpunkten *Pi* des *i*-ten Fahrstreifens gebildet, allerdings erstreckt sich dieses über alle Segmente, für die der Bezugsfahrstreifen dieselbe Textur aufweist: die Konvention der Randsteintexturierung bei *VRNG* sieht vor, die Textur jenes Fahrstreifens(=Bezugsfahrstreifen) an der Oberkante des Randsteins zu verwenden. Die Menge der entsprechenden Texturkoordinaten wird mit *RPT* bezeichnet. Unter der Voraussetzung, daß zwischen *i*- und *i+1*-tem Fahrstreifen ein Randstein existiert, gilt:

$$(1.7)$$
$$if\,(hs_{i+1} < 0)$$
$$RP_i = (Pii_{0,0},...,Pii_{0,l-1},Pi_{1,l-1},...,Pi_{1,0}); Pii_{0,r}, Pi_{1,r} \in R^3; 0 \le r < l; ii = i+1, i < n-3;$$
$$P_{0,r} \in \{Pii_0 \mid Bed.1\}; P_{1,r} \in \{Pi_1 \mid Bed.1 \wedge Bed.2\}; 0 \le r < l.$$
$$tex(RP_i) = tex(tp_{i,j}); i, j \in N; 0 \le i < n-2; 0 \le j < m-1;$$
$$if\,(hs_{i+1} > 0)$$
$$RP_i = (Pii_{0,0},...,Pii_{0,l-1},Pi_{1,l-1},...,Pi_{1,0}); Pii_{0,r}, Pi_{1,r} \in R^3; 0 \le r < l; ii = i+1, i < n-3;$$
$$P_{0,r} \in \{Pii_0 \mid Bed.1 \wedge Bed.2\}; P_{1,r} \in \{Pi_1 \mid Bed.1\}; 0 \le r < l.$$
$$tex(RP_i) = tex(tp_{i+1,j});$$
$$RPT_i = (PTii_{0,0},...,PTii_{0,l-1},PTi_{1,l-1},...,PTi_{1,0}); PTii_{0,r}, PTi_{1,r} \in R^2; 0 \le r < l; ii = i+1, i < n-3;$$

mit den weiter oben definierten relativen Höhendifferenzen *hs*. Diese Formulierung impliziert auch, daß sich die Randsteintextur ändert, wenn sich jene des Bezugsfahrstreifens ändert.

Problem der Überlappung kurzer, aufeinanderfolgender Fahrstreifen:

Per Definitionem kann die Länge eines Straßensegments beliebig kurz werden, sodaß besonders bei kurzen, aufeinanderfolgenden Straßensegmenten mit wechselnder Verlaufsrichtung eine Art Rückfaltung auftreten kann, dadurch ist ein bestimmter Bereich quasi von mehreren Segmenten überlagert. Dieses bringt geometrische Problem mit sich und es ist daher wichtig, bei der (inkrementellen) Generierung der Straße eine solche Situation zu erkennen und zu vermeiden. Abbildung-29 zeigt ein dafür typisches Beispiel:

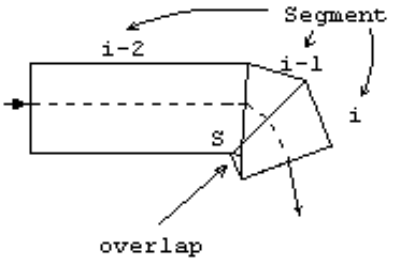


Abbildung 29: Überlappungsbeispiel mit 3 Geradenstücken

Man erkennt, daß die rechten Kanten der Segmente *i* und *i-2* einen Schnittpunkt *S* haben, der *vor* dem Schnittpunkt der rechten Kanten von den Segmenten *i-2* und *i-1* liegt. Die ursprüngliche Vorgangsweise zur Behandlung von Knicken im Straßenverlauf, wie auf Abbildung-24 zu sehen war, ist also nur bedingt anwendbar. Der folgende Algorithmus kann nun dazu verwendet werden, das aktuelle Segment *i* auf Überschneidung mit den Segmenten *i-1,i-2,...* zu testen:

$$
\begin{aligned}
&for(si = i-1; si \geq 0; si--)\\
&for(pl = 0; pl \leq \#lanes(segment\_si); pl++)\\
&for(pi = 0; pi \leq \#lanes(segment\_pi); pi++)\\
&if(S = (\overline{Pe_{pl,j1}(i), Pe_{pl,j1+2}(i)} \cap \overline{Pe_{pi,j2}(si), Pe_{pi,j2+2}(si)}) \neq \{\ \})\\
&Ve_{pl,j1}(i) = Ve_{pi,j2+2}(si) = S;\\
&j1, j2 \in \{0,1\};
\end{aligned}
$$

dabei werden also alle linken/rechten Kanten des aktuellen Segments mit allen linken/rechten Kanten der Vorgängersegment auf Überschneidungen getestet und ggf. die Eckpunkte *Pe0,..,Pe3* durch die entstehenden Schnittpunkte *Ve0,..,Ve3* ersetzt.

Für den formalen Rechengang zur Ermittlung der Kantenpunkte eines Fahrstreifens nach Formel(1.5) gilt es daher, für das *j*-te Segment einen Gültigkeitsbereich in Form eines Intervall lokaler Koordinaten *[a0,..,a1]* auf der linken/rechten Kante festzulegen:

$$
\begin{aligned}
&\exists[a0, a1] \Leftrightarrow ((Pe_{i,j} \neq Pe_{i,j+2}) \wedge (a1 > a0)); i = 0..m-1; j \in \{0,1\};\\
&a0 = gl(Ve_{i,j}); a1 = gl(Ve_{i,j+2}),
\end{aligned}
$$

es werden also die *a*-Werte der lokalen Koordinaten ermittelt und verglichen und festgestellt, ob die Kante überhaupt gültige Polygonpunkte enthält.

Zusammenfassung des Algorithmus zur Straßenmodellierung:

Die gesamte Struktur des Auswertungsprozesses bei der Straßengenerierung wird in Form eines Ablaufdiagramms auf Abbildung-30 gezeigt:

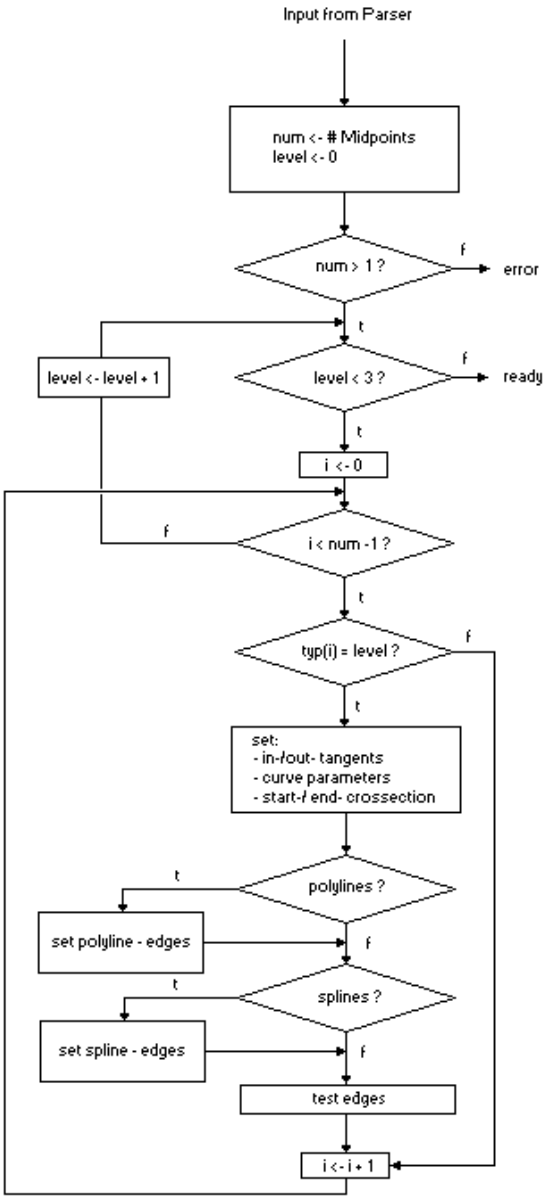


Abbildung 30: Struktogramm der sukzessiven Straßengenerierung mit hierarchischer Levelauswertung nach dem Typ des Segments (z.B. Kreis) und dem Initialisieren der Segmentattribute (Tangenten, Kanten usw.).

Geometrie der Knotenpunkte – Kreuzungsberechnungen:

In Kapitel-4.3.2 wurde bereits angeführt, daß die Kreuzung als eigenständige Einheit angesehen wird und quasi aus den Enden der Knotenpunktsarme, nach den Winkel im Einheitskreis, im Gegenuhrzeigersinn sortiert, erzeugt wird. Wie eingangs bei der Definition des Straßengraphen angeführt, ist jedem Knotenpunkt *K* aus *SN* also eine Menge von zulaufenden Straßen zugeordnet:

$$K(SN) = \{S, C, r, tp, PL, IP, IPtx, IPsz, Rtx, IH, tex, sz\}; tex \in Strings; sz \in R;$$

Menge der zulaufenden Straßen:

$$S = (S_0,..,S_{k-1}); i,k \in N; S_i \in S(SN); 0 \le i < k;$$

0/1..Straße endet/beginnt an *K*:

$$C = (c_0,..,c_{k-1}); c_i \in \{0,1\};$$

Radien der Abrundungsteile:

$$r = (r_0,..,r_{k-1}); r_i \in R^+ \cup \{-1\};$$

Form des Abrundungsteils:

$$tp = (tp_0,..,tp_{k-1}); tp_i \in RTyp; RTyp = \{0,1,2\};$$

Polyline-Kanten:

$$PL = (PL_0,...,PL_{k-1}); PL_i = (PL_{i,0},...,PL_{i,q}); 0 \le i < k; i,q \in N;$$

$$PL_{i,j} = (P_0,...,P_t); P_i \in R^3; 0 \le i \le t;$$

Polygone der Verkehrsinseln:

$$IP = (IP_0,..IP_{p-1});$$

$$IP_i = (P_0,..,P_{k-1}); 0 \le i < p; P_j \in R^3; 0 \le j < k; j,p \in N;$$

Texturen der Verkehrsinseln:

$$IPtx = (Itx_0,..,Itx_{p-1}); ltx \in Strings;$$

Textur-Minifikationswerte:

$$IPsz = (Isz_0,..,Isz_{p-1}); lsz \in R;$$

Randsteintexturen der Verkehrsinseln:

$$Rtx = (Rtx_0,..,Rtx_{p-1}); Rtx \in Strings;$$

Randsteinhöhen-Verkehrsinseln:

$$IH = (Ih_0,..,Ih_{p-1}); Ih \in R^+;$$

Aus den am Knotenpunkt beteiligten Straßen werden jene Straßensegmente *T* ermittelt, die an den Knoten anschließen:

$$T_i = T_0(S_i) \Leftrightarrow c_i = 1; sonst: T_i = T_{r-1}; 0 \le i < k-1; r = |T(S_i)|; i,k \in N;$$

$$T_{i+1} = T_0(S_{i+1}) \Leftrightarrow c_{i+1} = 1; sonst: T_{i+1} = T_{r-1}; 0 \le i < k-1; r = |T(S_{i+1})|; i,k \in N;$$

dadurch sind die Segmente an der Abrundung zwischen *i*- und *i+1*-ter Straße durch *Ti* und *Ti+1* festgelegt. Für weitere Berechnungschritte werden außerdem die an die Kreuzung anschließenden Querschnittsflächen *l* und die der Abrundung zugewandte äußere Straßenkante *e* benötigt:

(1.8)

$$e_i = \overline{Pe_{0,0}(T_i), Pe_{0,2}(T_i)} \Leftrightarrow c_i = 1; sonst: e_i = \overline{Pe_{m-1,1}(T_i), Pe_{m-1,3}(T_i)};$$
$$m = |ty(T_i)|; ty \in Typ; m \in N;$$
$$l_i = (P_{i,0}, \ldots, P_{i,m-1}) =$$
$$= (Pe_{m-1,1}(T_i), \ldots, Pe_{0,0}(T_i)) \Leftrightarrow c_i = 1; sonst: l_i = (Pe_{0,2}(T_i), Pe_{0,3}(T_i), \ldots, Pe_{m-1,3}(T_i));$$
$$e_{i+1} = \overline{Pe_{n-1,1}(T_{i+1}), Pe_{n.1,3}(T_{i+1})} \Leftrightarrow c_{i+1} = 1; sonst: e_{i+1} = \overline{Pe_{0,0}(T_{i+1}), Pe_{0,2}(T_{i+1})};$$
$$n = |ty(T_{i+1})|; ty \in Typ; n \in N;$$
$$l_{i+1} = (P_{i+1,0}, \ldots, P_{i+1,n-1}) =$$
$$= (Pe_{n-1,1}(T_{i+1}), \ldots, Pe_{0,0}(T_{i+1})) \Leftrightarrow c_{i+1} = 1; sonst: l_{i+1} = (Pe_{0,2}(T_{i+1}), Pe_{0,3}(T_{i+1}), \ldots, Pe_{n-1,3}(T_{i+1}));$$

Damit sind die Querschnittsflächen um das Zentrum des Knotens im Gegenuhrzeigersinn ausgerichtet, die 4 möglichen Fälle für die Richtung der Straßensegmente sind auf Abbildung-31 zusammengefaßt:

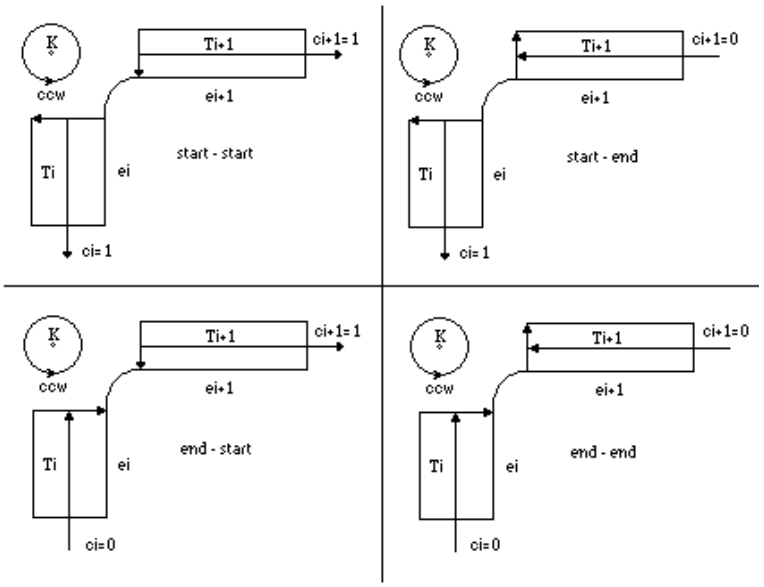


Abbildung 31: die 4 möglichen Fälle für die Richtung der im Gegenuhrzeigersinn (ccw) um das Zentrum (K) benachbarten Straßensegmente Ti und Ti+1 mit der Richtung der Achsen (ci, ci+1) und der Richtung (links->rechts) des Straßenquerschnitts am Knoten sowie der für die Abrundung relevanten Seitenkante (ei, ei+1).

Für die Berechnung der Abrundungsgeometrie ist es zunächst wichtig, einen Referenzpunkt *RFP* an *Ti* bzw. *Ti+1* festzulegen. Wie in (4.3.2) unter dem Abschnitt „Knotenpunkte“ beschreiben, handelt es sich um jenen Punkt, der die Fahrbahn vom Randstein trennt. Sei *ty=(ty0,...,tym-1)* das einem Segment
zugeordnete *n*-Tupel von Fahrstreifentypen (von links nach rechts), dann läßt sich dieses folgendermaßen partitionieren:

$$ty = (ty_0, \ldots, ty_r), (ty_{r+1}, \ldots, ty_{s-1}), (ty_s, \ldots, ty_{m-1}) \mid (ty_{r+1} = ty_{s-1} = \acute{} pkw\acute{}) \wedge (ty_p \neq \acute{\acute{}} pkw\acute{})$$
$$\forall p \mid p \in [0, r] \cup [s, m-1]; 0 \leq r \leq s; 0 \leq s \leq m; p, r, s, m \in N.$$

Mit den Indices *r,s* lassen sich auch *li* und *li+1* partitionieren:

$$l_i = l_i' \cup l_i'' \cup l_i''' = (P_{i,0},...,P_{i,r}),(P_{i,r+1},...,P_{i.s-1}),(P_{i,s},...,P_{i,m-1});$$
$$l_{i+1} = l_{i+1}' \cup l_{i+1}'' \cup l_{i+1}''' = (P_{i+1,0},...,P_{i+1,r}),(P_{i+1,r+1},...,P_{i+1.s-1}),(P_{i+1,s},...,P_{i+1,n-1});$$

die *r,s* für *li* und *li+1* sind dabei nicht notwendigerweise übereinstimmend.
Aus der Partition lassen sich direkt die Referenzpunkte ablesen:

$$RFP_i = P_{i,s-1};$$
$$RFP_{i+1} = P_{i+1,r+1};$$

Im nächsten Schritt ist die Art der Abrundung zu bestimmen. In *VRMG* gibt es die Grundformen *Gerade*, *Kreisbogen* und *S-Kurve*, wobei der Kreisbogen dem Standardfall entspricht(vgl. Kapitel(4.3.2)-Kreuzungen: Abrundungsbögen). Abbildung-32 zeigt die typischen Fälle des Aufeindertreffens zweier an die Kreuzung anschließender Straßensegmente:

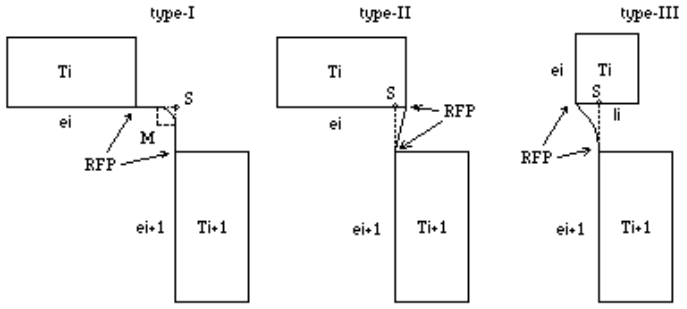


Abbildung 32: Grundtypen Abrundungsbogen: Kreis(links), Gerade(mitte) und S-Kurve(rechts).

Die Abbildung verdeutlicht die Abhängigkeit des Abrundungsbogentyps von der Lage des Schnittpunkts *S*.

Kreisbogen(Typ-I):

Durch Schnitt der Geraden durch die Referenzpunkte mit den entsprechenden Tangentenrichtungen der Segmente erhält man den Bezugspunkt des Abrundungsbogens mit Radius r (undefiniert: r=-1):

$$(1.9a)$$

$$S = g(RFP_i, t_i) \cap g(RFP_{i+1}, t_{i+1}); S \in R^2;$$

$$Bed.-Kreis : (S \neq \{\}) \wedge ((S \notin e_i) \wedge (S \notin e_{i+1}));$$

$$t_i = t_0(S_i), sonst : t_i = t_1(S_i); t_{i+1} = t_0(S_{i+1}), sonst : t_{i+1} = t_1(S_{i+1});$$

$$\|t_i\| = \|t_{i+1}\| = 1;$$

$$if(r_i > 0)$$

$$t_i = \begin{pmatrix} t_i x \\ t_i y \end{pmatrix} \Rightarrow nt_i = \begin{pmatrix} t_i y \\ -t_i x \end{pmatrix}; Pe_i' = \vec{P}e_i' + r_i \cdot nt_i;$$

$$t_{i+1} = \begin{pmatrix} t_{i+1} x \\ t_{i+1} y \end{pmatrix} \Rightarrow nt_{i+1} = \begin{pmatrix} t_{i+1} y \\ -t_{i+1} x \end{pmatrix}; Pe_{i+1}'' = \vec{P}e_{i+1}'' + r_i \cdot nt_{i+1};$$

$$M = g(Pe_i', t_i) \cap g(Pe_{i+1}'', t_{i+1});$$

$$AP_0 = \vec{M} - r_i \cdot nt_i; AP_1 = \vec{M} - r_i \cdot nt_{i+1}; M \in R^2; AP_0, AP_1 \in R^3;$$

$$AP_0 \in \overline{RFP_i, S}; AP_1 \in \overline{S, RFP_{i+1}};$$

else

$$d_i = \left\| \vec{S}, \overrightarrow{RFP_i} \right\|, d_{i+1} = \left\| \vec{S}, \overrightarrow{RFP_{i+1}} \right\|;$$

$$AP_0 = RFP_i, AP_1 = RFP_{i+1} \Leftrightarrow (d_i = d_{i+1});$$

$$AP_0 = RFP_i, AP_1 = \vec{S} - d_i \cdot t_{i+1} \Leftrightarrow (d_i < d_{i+1}); AP_1 \in \overline{S, RFP_{i+1}};$$

$$AP_0 = \vec{S} - d_{i+1} \cdot t_i, AP_1 = RFP_{i+1} \Leftrightarrow (d_i > d_{i+1}); AP_0 \in \overline{RFP_i, S};$$

Der Abrundungsbogen von einem Referenzpunkt zum nächsten besteht also aus den Teilen *Gerade->Kreis->Gerade*, wobei für den Fall, daß der Übergangspunkt *AP* von *Gerade* zu *Kreis* bzw. *Kreis* zu *Gerade* mit dem Referenzpunkt zusammenfällt, der entsprechende Geradenteil entfällt.

<u>Gerade(Typ-II):</u>

Ist nach Formel(1.9) die Bedingung für eine Abrundung durch einen Kreisbogen(*Bed.-Kreis*) nicht erfüllt und gilt:

$$Bed.-Gerade : (S \neq \{\}) \wedge ((S \in e_i) \wedge (S \in e_{i+1}));$$

dann liegt entweder eine *Überlappung* oder eine *Verschiebung* wie auf Abbildung-32(Mitte) vor. *VRMG* verwendet in diesem Fall entweder eine einfache Verbindungsgerade zwischen den Referenzpunkten oder berechnet die theoretisch korrekten Endpunkte der Segmentmittelpunkte:

$$(1.9b)$$

$$M_i' = \vec{M}_i - t_i \cdot \left(\left\| \dot{RFP}_i, \vec{S} \right\| + e \right) \Leftrightarrow S \in e_i;$$

$$M_i = M_0(S_i) \Leftrightarrow c_i = 1; sonst : M_i = M_1(S_i); e \in R^+;$$

$$M_{i+1}' = \vec{M}_{i+1} - t_{i+1} \cdot \left(\left\| \dot{RFP}_{i+1}, \vec{S} \right\| + e \right) \Leftrightarrow S \in e_{i+1};$$

$$M_{i+1} = M_0(S_{i+1}) \Leftrightarrow c_{i+1} = 1; sonst : M_{i+1} = M_1(S_{i+1});$$

Für e=0 reduziert sich der Abrundungsbogen zu einem Punkt *RFPi=RFPi+1,* für e>0 wird die Abrundung auf Typ-II zurückgeführt.

S-Kurve(Typ-III):

Liegen die Kanten *ei* und *ei+1* parallel oder quasi-parallel, d.h.:

$$S = g(RFP_i, t_i) \cap g(RFP_{i+1}, t_{i+1}); (S = \{\}) \vee ((S \in l_i) \vee (S \in l_{i+1}));$$

dann wird die Abrundung mit folgendem Algorithmus durchgeführt, der einem S-förmigen Verlauf entspricht:

$$(1.9c)$$

$$v = \overrightarrow{RFP}_i + \overrightarrow{RFP}_{i+1}; a = \cos^{-1}\left(\frac{t_i \cdot v}{\|t_i\| \cdot \|v\|}\right);$$

$$AP_2 = \dot{RFP}_i + \frac{v}{2}; AP_0 = \dot{RFP}_i + \cos a \cdot \frac{\|v\|}{6} \cdot t_i; AP_1 = \dot{AP}_0 + \frac{v}{6};$$

$$AP_3 = \overrightarrow{AP}_1 + 2 \cdot \left(\overrightarrow{AP}_1, \overrightarrow{AP}_2\right); AP_4 = \overrightarrow{AP}_3 + \frac{v}{6};$$

Insgesamt besteht der Abrundungsbogen *BG* nach den Formeln 1.9a-1.9c aus folgender Menge von Punkten:

$$BG = (RFP_i, AP, RFP_{i+1}); AP = (AP_0, ..., AP_{a-1}) \cup \{\};$$

Die Berechnung des Abrundungsteils *AT* erfolgt analog zum Straßensegment:

$$AT_0(BG) = \{< RFP_i, AP_0 >, B_0, H_0, MD_0, t_0, g_0, P_0, TY_0\},$$

.

.

$$AT_a(BG) = \{< AP_{a-1}, RFP_i >, B_a, H_a, MD_a, t_a, g_a, P_a, TY_a\}$$

dabei ist die Punktmenge *BG* mit der Mittelachse der Abrundungsteil-Segmente identisch. Zur Anpassung der *Breitenwerte* der Querschnittsflächen des Abrundungsteils wird eine Korrektur derselben durchgeführt:

$$B_0 = b_j(T_i) - b_s(T_i); j = s..m-1; B_a = b_j(T_{i+1}) - b_r(T_{i+1}); j = 0..r; j \in N;$$

dabei korrespondieren die *r* und *s* mit jenen von *li* bzw. *li+1,* wie oben eingeführt.

Die Bestimmmung der Parameter für die Querschnitte an den *BG* erfolgt durch Interpolation der Parameter-Tupel zwischen *RFPi* und *RFPi+1*:

$$l = \sum_{k=1}^{a} \left\| \dot{BG}_k, \dot{BG}_{k-1} \right\|;$$

$$\Delta B = \left(b_j(B_0) - b_j(B_a) \right);\ j = 0..\min\left(|B_0|, |B_a| \right);$$

$$B_j = \bigcup_{w=0}^{n} \frac{\left(\sum_{k=1}^{j} \left\| \overrightarrow{BG}_k, \overrightarrow{BG}_{k-1} \right\| \right) \cdot \Delta b_w}{l};\ n = |\Delta B| - 1;\ j = 1..a - 1;$$

$$H_j = \bigcup_{w=0}^{n} \frac{\left(\sum_{k=1}^{j} \left\| \overrightarrow{BG}_k, \overrightarrow{BG}_{k-1} \right\| \right) \cdot \left(h_w(T_i) - h_w(T_{i+1}) \right)}{l};$$

$$MD_j = \text{´}c\text{´};$$

$$t_j = (t_i, t_{i+1}) \Leftrightarrow Bed. - Kreis(nach(1.9a)),$$

$$sonst: t_j = (t_0, t_1) \mid t_0 = t_1 = \frac{\left(\dot{BG}_{j-1}, \dot{BG}_j \right)}{\left\| \overrightarrow{BG}_{j-1}, \overrightarrow{BG}_j \right\|};$$

$$g_j = 0;$$

$$P_j = PL_{i,j};$$

$$TY_j = TY(T_i);$$

nach Anwendung der Berechnungen für Segment *AT* erhält man nach (1.6) für das Polygon des *j*-ten Fahrstreifenteils:

(1.10)

$$FP_j = \left(P_{0,0}, ..., P_{0,p-1}, P_{1,p-1}, ..., P_{1,0} \right);\ P_{k,t} \in R^3; 0 \le t < p; k \in \{0,1\}.$$

$$FPT_j = (PT_{0,0}, ..., PT_{0,p-1}, PT_{1,p-1}, ..., PT_{1,0});\ PT_{k,t} \in R^2; k \in \{0,1\}.$$

sowie nach (1.7) den das entsprechende Randsteinpolygon, sofern es existiert:

(1.11)

$$RP_j = \left(Pjj_{0,0}, ..., Pjj_{0,l-1}, Pj_{1,l-1}, ..., Pj_{1,0} \right);\ Pjj_{0,t}, Pj_{1,t} \in R^3; 0 \le t < l;\ jj = j + 1,\ j < n - 3;$$

$$RPT_j = \left(PTjj_{0,0}, ..., PTjj_{0,l-1}, PTj_{1,l-1}, ..., PTj_{1,0} \right);\ PTjj_{0,t}, PTj_{1,t} \in R^2; 0 \le t < l;\ jj = j + 1,\ j < n - 3;$$

Der im Kapitel-(4.3.2) definierte *Fahrbahnteil FBT* ist daher durch die Unterkante der Randsteinpunkte(oder den äußeren Rand des Abrundungsteils) an der Achse des Abrundungsteils *AT* begrenzt. Falls die Punktfolge dabei dem Randstein entnommen wird, ist die Reihenfolge der *Pt* zur Einhaltung des Gegenuhrzeigersinns von *FBT* umzukehren:

$$FBT = \bigcup_{i=0}^{k-1} \begin{array}{l} (P_{l-1}, ..., P_0) \mid P_t \in RP_0(AT_i) \Leftrightarrow \left(RP_0(AT_i) \neq \{\ \} \right); l = \frac{|RP_o(AT_i)|}{2}; \\ (P_0, ..., P_{l-1}) \mid P_t \in FP_0(AT_i) \Leftrightarrow \left(RP_0(AT_i) = \{\ \} \right); 0 \le t < l; l = \frac{|P(AT_i)|}{2}; \end{array}$$

Die Texturkoordinaten der Textur *tex* des *FBT*-Polygons werden so wie die Höhenfeldpunkte global mit der Funktion *gl* berechnet:

$$(u,v) = gt(x,y);$$
$$u(P_t) = sz \cdot x; v(P_t) = sz \cdot y; \forall P_t \in FBT; sz \in R.$$

Das endgültige Aussehen des *i*-ten Abrundungsteils *ATi* hängt davon ab, ob dieser dem *T-Typ*, *N-Typ* oder *P-Typ* entspricht (vgl. Kapitel 4.3.2):

- *T*-Typ: es werden (von innen nach außen) alle (*a+1*) gemeinsamen Fahrstreifen des Abrundungsteils der Segmente *Ti* und *Ti+1* verbunden, d.h. alle (*a+1*) nach (1.10) erzeugten Polygone plus Texturkoordinaten und nach (1.11) erzeugten vorhandenen Randsteine plus Texturkoordinaten werden verwendet:

$$AT_i = RP_{0,} FP_0, ..., RP_a, FP_a;$$

- *N*-Typ: hierbei besteht der Abrundungsteil nur aus 3 Polygonen: Randsteinpolygon(vertikal), Randsteinpolygon(horizontal=innerster Streifen des Abrundungsteils) und einem Polygon, das an den Randstein anschließt und (nach außen hin) alle restlichen Streifen der Segmente *Ti* und *Ti+1* verbindet:

$$AT_i = RP_0, FP_0, \bigcup_{j=1}^{a} RP_j \cup FP_j,$$

- *P*-Typ: diese Variante verbindet die Außenkante des Abrundungsteils mit einer Polyline *PLY*:

$$AT_i = RP_0, FP_0, (P_0, ..., P_{l-1}) \cup (P_{i,s+1}, P_{i,m-1}, PLY, P_{i+1,n-1}, P_{i+1,r-1});$$
$$P_t \in FP_1(AT_i), P_{i,j} \in l_i'''', j \in \{s+1, m-1\}, P_{i+1,k} \in l_{i+1}', k \in \{r-1, n-1\}, 0 \le t < l;$$
$$PLY = (PLY_0, ...PLY_{p-1}) \cup \{\}; PLY_q \in R^3; 0 \le q < p;$$

die Standardeinstellungen für *PLY* sind:

$$PLY = \{\},$$
$$PLY = S = g(P_{i,m-1}, t_i) \cap g(P_{i+1}, t_{i+1}),$$
$$PLY = PL_{i,q}; q = |PL_i| - 1; PL_{i,q} \in PL;$$

Damit sind der Fahrbahnteil und die Abrundundssegmente definiert, die *Verkehrsinseln* als drittes Element des Knotenpunkts setzen sich aus je 2 Polygonen zusammen: horizontales Inselpolygon plus Randsteinpolygon:

$$IP_i = (P_0, ..., P_{k-1});$$
$$IP_i' = (P_0', ..., P_{k-1}') \mid z(P_t') = z(P_t) - Ih_p; t = 0..k-1; t \in N.$$
$$IRP_i = (P_0', ..., P_{k-1}', P_0', P_0, P_{k-1}, ..., P_0);$$

Die Texturkoordinaten der Textur IP*tx* des *IP*-Polygons und des zugehörigen Randsteins mit Textur Rtx und Texturkoordinaten IRPT werden global mit der Funktion *gl* berechnet:

$$(u,v) = gt(x,y);$$
$$IPT_i = (IPT_0,\ldots,IPT_{k-1}); IPT_t \in R^2 \mid u(P_t) = IPsz_i \cdot x; v(P_t) = IPsz_i \cdot y; \forall P_t \in IP_i;$$
$$IRPT_i = (IRPT_0,\ldots,IRPT_k, IRPT_{k+1},\ldots,IRPT_n); IRPT \in R^2 \mid$$
$$((u(IRPT_t) = 0) \Leftrightarrow 0 \le t \le k, (u(IRPT_t) = 1) \Leftrightarrow k < t \le n),$$
$$\left(\begin{array}{l}(v(IRPT_t) = v(IPT_t) \forall t = 0..k-1), (v(IRPT_k) = v(IRPT_{k+1}) = IPT_0),\\ (v(IRPT_{t+2}) = v(IPT_{n-t}) \forall t = k..n-2)\end{array}\right\}$$

Dadurch sind die 3 Grund-Bausteine der Kreuzung: Fahrbahnteil, Abrundungsteile und optionale Verkehrsinseln geometrisch erfaßt.

Das Resultat von diesem Schritt 5.1. war es also, Straßen, Kreuzungen und dem Terrain in eine *polygonale Form* zu bringen und mit den entsprechenden Texturparametern zu versehen. Im nächsten Schritt, der bereits zu den Geometrischen Optimierungen zählt, erfolgt die Zerlegung der vorliegenden Polygone in Elementareinheiten, d.h. in Dreiecke.

5.3.) Delaunay-Triangulierung[17,19,36]:

Wie bereits mehrfach erwähnt, ist der erste Optimierungsschritt für die Geometriedaten die Zerlegung der im vorigen Abschnitt erzeugten Polygone in Dreiecke. Ziel ist es, die in der Form eines *PSLG(Planar-Straight-Line-Graph)*:

$$P = (P_0,\ldots,P_{n-1});$$
$$E = ((0,1),\ldots,(i,i+1),\ldots,(n-1,0)); i = 0..(n-1) \bmod n, i \in N.$$

vorliegenden Polygone, definiert durch Punkte(*P* - im Gegenuhrzeiger-sinn sortiert) und Kanten(*E* - indiziert in Bezug auf *P*), zu triangulieren, d.h. in eine Menge von Tripel:

$$E' = \{(i,j,k)\} \forall i,j,k \in N \mid i \ne j \ne k, 0 \le i,j,k < n.$$

wobei die Tripel *(i,j,k)* ein indiziertes Dreieck mit den Eckpunkten *Pi,Pj,Pk* repräsentieren und als *Dreiecksvermaschung*(engl.:*triangle-mesh*) bezeichnet werden. Obwohl im allgemeinen mehrere Möglichkeiten für die Dreieckszerlegung existieren, ist es sinnvoll, jene Zerlegung zu wählen, die möglichst günstige Eigenschaften aufweist:

- Maximierung der Dreieckswinkel durch Einstellen eines Minimalwinkels.
- Erhalten von vordefinierten Kanten.
- Möglichkeit zum Ausschneiden von Hohlflächen.

Diese Anforderungen erfüllt die *Delaunay-Triangulierung*, die auf die Bedingung aufbaut, daß für alle Dreiecke gilt, daß sich kein Punkt aus *P* innerhalb des Umkreises *UK* eines Dreiecks befindet[36].

Die 3 wichtigsten Algorithmen für die Delaunay-Triangulierung, die auch von *R.Shewchuck's TRIANGLE*[17] unterstützt werden sind:

- *Divide and Conquer Algorithm* (D.T.Lee, B.J.Schachter (1980)): Grundidee bei diesem *rekursiven* Verfahren ist die schrittweise Partition(„*Divide*") der Punktmenge *P* in Zellen, deren Größe mit jedem Rekursionschritt abnimmt, solange bis die Zelle weniger als 4 Punkte enthält. Dadurch ist das Problem innerhalb der Zelle auf die 3 Trivialfälle reduziert, sodaß der Beitrag einer Zelle zum Ganzen entweder eine Kante oder ein Dreieck ist. Der zweite Schritt ist die schrittweise Vereingung(„*Merge*") von benachbarten Zelleelementen(Kanten, Dreiecke) unter Berücksichtigung der Delaunay-Bedingung[17]. Der Rechenaufwand beträgt *O(n*log(n))*.

- *Incremental Insertion Algorithm* (C.L.Lawson (1977)): Im Gegensatz zum vorigen Verfahren erfolgt hier ein *sequentielles* Einfügen der Punkte. Ausgehend von einem Dreieck, bestehend aus den ersten 3 Punkten, wächst die Dreiecksmenge durch Triangulierung jedes neu eingefügten Punktes. In jedem Schritt ist die (lokale) Erfüllung der Delaunay-Bedingung zu prüfen und im Fall der Verletzung dieser ggf. eine Umordnung der bestehenden Kanten vorzunehmen. Im Prinzip werden dabei 2 Dreiecke an der Kante getestet, diese ergeben zusammen ein Viereck, dessen eine Kante(1.Diagonale) gegen die andere Kante(2.Diagonale) getauscht wird. Für eine detaillierte Beschreibung siehe[36,37].

- *Sweepline Algorithm* (S.Fortune (1987)): Hier werden die Delaunay-Dreiecke quasi direkt aus den Umkreisen erzeugt. Die Vorgangsweise ist die folgende: eine Gerade parallel zur *x*-Achse, die sogenannte *Sweepline*, wird ausgehend vom Punkt mit dem minimalen *y*-Wert, aufwärts verschoben. Das Verschieben wird durch eine *Event-Queue* gesteuert, diese reagiert auf 2 Ereignisse: 1.) die *Sweepline* passiert einen neuen Knoten durch Aufwärtsbewegung oder 2.) es wird unter den bereits besuchten Punkten ein neues Delaunay-Dreieck gefunden. Die *Event-Queue* beihaltet dabei stets die Menge der Punkt-Kandidaten für die Triangulierung. Die *Sweepline* wird in eine Menge von aufeinanderfolgenden Intervallen partitioniert, daraus resultiert eine Ordnung der Punktmenge in *x* und *y*. Für die Konstruktion eines (theoretisch) möglichen Umkreises *K* nimmt man einen (theoretischen) Punkt *Q* auf der Sweeplinie an, für den gilt, daß dieser der höchste Punkt von *K* durch *Q* und einen Punkt *Pi* aus der Punktmenge *P* ist. Tritt nun der Fall ein, daß *K* auch die dem linken- und rechten Nachbarintervall zugeordneten *Pi-1* und *Pi+1* aus *P* berührt, so ist ein neues Delaunay-Dreieck gefunden worden. Der Aufwand beträgt ebenfalls *O(n*log(n))*. Die formale Herleitung und der Beweis des Algorithmus' läßt sich in [37] nachlesen.

Einen Performance-Vergleich der 3 Verfahren findet man z.B. in[17]. Allen 3 Verfahren ist gemeinsam, daß aus dem Bereich der (Triangulierungs-) Möglichkeiten eine Lösung, die der Delaunay-Bedingung genügt, gefunden wird, unter der Voraussetzung, daß der Algorithmus terminiert(für eine Problembeschreibung von Delaunay-Triangulierungsalgorithmen bei ungünstiger Konstellation der Eingbedaten siehe[17]). Ist im konkreten Anwendungsfall die Triangulierung mit den oben aufgelisteten, angestrebten Eigenschaften zu verknüpfen, so wird bei *TRIANGLE* der *Constraint-* oder *Refinement Algorithmus* von J.Ruppert verwendet:

1.) Im ersten Schritt wird mittels eines der 3 Standardverfahren von oben eine Delaunay-Triangulierung der Punktdaten erstellt. Da der *PSLG* vordefinierte Kanten enthält, die erhalten bleiben sollen, müssen jene Kanten, die nicht bereits bei der Triangulierung (zufällig) erstellt wurden, eingefügt werden.

2.) Es gibt 2 Möglichkeiten, die Kanten des *PSLG* einzufügen:

   a) Rekursives Einfügen des (Strecken-)Teilungspunktes der Kantenendpunkte nach der *Incremental Insertion* Methode, solange bis die gesuchte Kante in der Dreiecksvermaschung vorhanden ist.

   b) Die Kante wird ohne Hilfspunkte direkt eingefügt und alle bisherigen Dreiecke mit Kanten, die die eingefügte Kante schneiden, werden entfernt und lokal in der Nachbarschaft der eingefügten Kante retrianguliert.

3.) Entfernung von Hohlflächen innerhalb des Polygons: Hohlflächen werden in derselben Weise definiert wie Polygone und entsprechend mittrianguliert. Entspricht ein Polygon einer Hohlfläche, so ist es bei diesem Verfahren nötig, einen Punkt innerhalb dieses Polygons zu spezifizieren, der als *hole* bezeichnet wird. Das Ausschneiden dieses Polygons erfolgt nun von diesem als „hole“ angegebenen Punkt aus, d.h. es werden alle Dreiecke entfernt, solange bis eine Kante aus dem *PSLG* erreicht wird.

4.) Um einen gewünschten Minimalwinkel der Dreiecke zu erzielen, wird folgendes Verfahren angewandt: Die Kanten jener Dreiecke, die die Bedingung, daß zumindest einer der Winkel α kleiner als das gesuchte Minimum ist, nicht erfüllen, werden rekursiv in 2 Teile geteilt und der Teilungspunkt, in diesem Zusammenhang als *Steinerpunkt* bezeichnet, eingefügt und zwar solange, bis die Winkel-Bedingung und die Delaunay-Bedingung erfüllt sind.

Das Ergebnis der Delaunay-Triangulierung liegt also in folgender Form vor:

$$P = (P_0, \ldots, P_{n-1}),\ P_i \in R^3$$
$$E' = \{(i, j, k)\},\ 0 \le i, j, k < n,\ i \ne j \ne k$$

Falls die *z*-Werte der *Pi* aus *P* nicht bekannt sind (z=8), etwa weil sie erst bei der Triangulierung eingefügt wurden, werden sie aus dem Höhenfeld durch Anwendung der Funktion *h* berechnet:

$$P_i z = h(P_i x, P_i y) \Leftrightarrow P_i z = \infty, P_i \in P.$$

Damit wurde also erreicht, daß die im dreidimensionalen nicht notwendigerweise koplanaren Dreiecke, aus denen ein Polygon besteht, durch Anwendung der Transformation
$R^3 \to R^2$: $(x,y,z) \to (x,y)$ mittels Triangulierung erstellt wurden und anschließend ins dreidimensionale rücktransformiert wurden. Eine Zusammenfassung von weiteren Anwendungsmöglichkeiten und Vorzügen der Delaunay-Triangulierung findet man in[36].

5.4.) Triangle-Stripping[14,15,16]:

Der Prozeß des *Triangle-Stripping* fügt die im vorigen Schritt erzeugten Dreiecke wieder zu größeren Einheiten, den sogenannten *Tristrips* zusammen. Der Hauptgrund im Sinne der Datenreduktion ist, daß sich benachbarte Dreiecke je eine Kante teilen und deren Eckpunkte mehrfach gespeichert werden. Diese Redundanz kann durch die Verkettung von solchen Dreiecken in *„Dreiecksstreifen"* vermieden werden, sodaß sich die Anzahl der Abzuspeichernden Punkte von *3*n* auf *n+2* reduziert. Abbildung-34 zeigt einen solchen Tristrip bestehend aus 6 Dreiecken:

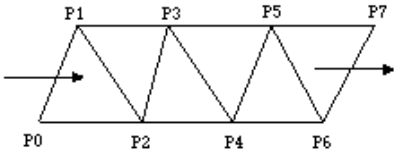


Abbildung 33:Triangle-Strip Sequenz mit 8 Punkten und 6 Delaunay-Dreiecken.

Das Ergebnis der Delaunay-Triangulierung für dieses Beispiel ist:

$$DT = \{(0,2,1),(1,2,3),\ldots,(5,6,7)\}$$

wobei die Indices der Tripel im Gegenuhrzeigersinn geordnet sind. Die Index-Sequenz des zugeordneten Tristips wäre:

$$TS = (0,1,2,3,4,5,6,7);$$

obwohl je 3 aufeinanderfolgende Indices ein Dreieck repräsentieren, erkennt man den alternierenden Ordnungsinn der Punkte. Eine mögliche Lösung, die bei *FTSG*[14] verwendet wird, ist die Verwendung eines „Null-Dreiecks" oder *swap-bits*, wodurch die Indizierung für alle Dreiecke wieder korrekt im Gegenuhrzeigersinn erfolgt:

$$TS = (0,2,1,2,3,2,4,\ldots); resp(0,2,1,swap,3,swap,4,\ldots)$$

Folgende 3 Kriterien charakterisieren die angestrebten Eigenschaften jedes Triangle-Stripping-Algorithmus:

- *Maximierung der Striplänge*: durch das Auffinden möglichst langer Pfade in der Triangulierung.
- *Reduktion von Mehrfachindices*: jedes Swap-Bit verringert die Effizienz der Tristrips.
- *Minimierung der Anzahl isolierter Dreiecke*: durch Auswahl jenes (Kandidaten-) Dreiecks während der Verkettung, das in der lokalen Nachbarschaft am schwächsten verbunden ist.

Es existieren zahlreiche *Triangle-Stripping* Algorithmen, von diesen werden an dieser Stelle die Konzepte von *STRIPE* und *FTSG* behandelt.

1.) STRIPE[15,16]:

Die Ausgangsbasis von *STRIPE* bildet der *SGI-Algorithmus*, dieser zielt vor allem auf die Einhaltung der Kriterien 1 und 3 ab und gliedert sich in folgende Schritte:

1) Lege die Nachbarschaftsbeziehungen zwischen den Dreiecken(gemeinsame Kante vorhanden!) in einer *Hash-Tabelle* ab.

2) Bestimme die Anzahl der Nachbarn jedes Dreiecks(1-3 möglich), speichere diese in einer *Priority-Queue*(Dreiecke mit geringer Anzahl von Nachbarn erhalten höhere Priorität).

3) Wähle Dreieck mit hoher Priorität für Stripanfang.

4) Verlängere den Strip durch Erweiterung der aktuellen Sequenz um das Nachbardreieck mit höchster Priorität. Im Fall, daß mehrere Kandidaten existieren, prüfe die Nachbarschafts-beziehungen aller Kandidaten, falls dies ebenfalls zu keiner Entscheidung führt, wird eines dieser Dreieck per Zufall ausgwählt.

5) Falls die Anzahl der Kandidaten aus der Nachbarschaft des zuletzt integrierten Dreiecks größer Null, gehe zu Schritt-(4). Ist die Anzahl der noch nicht erfaßten Dreiecke größer Null, so wird bei Schritt-(3) ein neuer Tristrip begonnen, ansonsten terminiert der Algorithmus.

Dieser Algorithmus ist allerdings als strikt *lokal* mit *statischer Triangulierung* zu bezeichnen und demzufolge in seiner Flexibilität sehr eingeschränkt, sodaß *STRIPE* folgende Erweiterungen enthält:

- *Dynamische Triangulierung*: für jedes Polygon wird eine Anfangs- und Endkante definiert, durch diese verläuft der Tristrip.

- Für den lokalen SGI-Algorithmus wurden mehrere Optionen zur Auflösung von Mehrdeutigkeiten in der Kandidatenauswahl verwendet: Auswahl den „Nächstbesten“, richtungsalternierend, stochastisch und sequentiell(d.h. jenes Dreieck, das keinen *swap* benötigt).

- *Globales Patching*: die Struktur der Polygone wird untersucht, mit dem Ziel, sogenannte *Patches* zu extrahieren, d.h. Regionen von Rechtecken, die (zufällig) nach Zeilen und Spalten geordnet sind und daher leicht in einen Tristrip umgesetzt werden können. Allerdings ist hier ein Vorverarbeitungsschritt zur vertikalen und horizontalen Sortierung notwendig.

Eine detaillierte Beschreibung zusammen mit Performance-Daten und formalen Beweisen ist in[15] zu finden.

2.) *FTSG*(Fast Triangle Strip Generator)[14]:

Im Gegensatz zu *STRIPE* wird bei *FTSG* eine andere Suchstrategie verwendet. Dieser Algorithmus setzt sich aus folgenden Schritten zusammen:

1) *Triangulierung*: dieser Schritt entfällt, wenn die Datenmenge der Punkte bereits vollständig trianguliert ist.

2) *Adjazenzgraph*: Erstellen eines Graphen, dessen Kanten benachbarte Dreiecke verbinden und Knoten, die den Dreiecken zugeordnet sind. Die Vorgangsweise ist eine kombinierte *Tiefen-* und *Breitensuche.*
   Erzeugen des *Spannender Baumes*: dieser ist der zum *Adjazenzgraph duale Graph.* Zur Optimierung des *Spannenden Baumes* ist es hinsichtlich der im nächsten Schritt durchgeführten Suche nach *Hamilton-Strips* günstig, die Zahl der Kanten pro Knoten gering zu halten, woraus eine geringere Anzahl an Pfaden und somit eine Maximierung der *Tristriplänge* vorbereitet wird. Bei der Suche nach dem nächsten Dreieck auf dem aktuellen Pfad wird mit den in *STRIPE* umgesetzten Methoden zur Auflösung von Mehrdeutigkeiten operiert.

3) Unterteilung des *Spannenden Baumes* in Teilbäume, die einer *Hamilton-Triangulierung* entsprechen: eine Triangulierung ist *hamilton'sch* und *sequentiell,* wenn folgende Bedingungen erfüllt sind:

   I) es existiert ein *Hamilton-Pfad* im *Adjazenzgraph,* sodaß jedes Dreieck bei Durchlauf genau einmal besucht wird.
   II) von den auf diesem Pfad überschrittenen Kanten gilt, daß sich 3 nacheinander überquerte Kanten keinen gemeinsamen Endpunkt teilen.

Diese Partition wird als *Path Peeling Algorithmus* bezeichnet und hat linearen Aufwand.

4) Umsetzung der in (3) generierten *Hamilton-Strips* in *sequentielle Tristrips.* Diese Operation inkludiert die Einhaltung einer einheitlichen Orientierung(Uhrzeigersinn oder Gegenuhrzeigersinn) der Dreiecke eines Tristrips.

5) Verkettung der in (4) erzeugten *Tristrips* in längere Strips durch Nach-bearbeitungsschritte basierend auf einer Heuristik. Besonderes Augenmerk wird hierbei auf Integration von isolierten Dreiecken gelegt: diese sind können prinzipiell an allen 3 Kanten mit benachbarten Tristrips verbunden werden, während Strips mit Länge größer als eins entweder über das erste oder das letzte Dreieck an Nachbardreiecke angeschlossen werden können, d.h. es stehen hier nur 2 Kanten zur Verfügung. Zur Einhaltung der Orientierungsordnung muß ggf. ein Nulldreieck oder ein *swap-bit* eingefügt werden.

Eine vergleichende Beurteilung von *STRIPE* und *FTSG* sowie die theoretischen Grundlagen und Beweise findet man in[14].

Im Rahmen des *VRMG*-Projektes wurde die Entscheidung zugunsten von *FTSG* getroffen. Gründe dafür waren die bessere Performance-Leistung, die Möglichkeit der Spezifikation von Polygongruppen, deren Triangulierung separat in Tristrips umgesetzt werden kann und nicht zuletzt aus Gründen der Programmergonomie, worauf im Kapitel-6 noch näher eingegangen wird.
Ein wesentlicher Schritt im Zusammenhang mit der Generierung der Tristrips ist die *Gruppierung* der Delaunay-Dreiecke nach Zellen, wie in Kapitel-3.1.) - Modellierung von 3D-Objekten / Sichtbarkeitsberechnungen beschrieben wurde. Das Ergebnis soll also eine (im Grundriß des Modells) nach Zeilen und Spalten geordnete Ausgabestruktur sein, wobei jedes Dreieck umkehrbar eindeutig auf eine solche Zelle abgebildet werden kann. Seien *xlow,ylow,xhigh,yhigh* die Koordinaten der Eckpunkte des Höhenfeldes und *m* bzw. *n* die Anzahl der Zellen in *x*- bzw. *y*-Richtung, dann wird jedes Dreieck durch *P0,P1,P2* aufgrund seines Schwerpunktes einer Zelle durch die Funktion $cell: R^3 x\ R^3 x\ R^3 -> N^2$ eindeutig zugeordnet, wobei *dx,dy* gleich der Dimension der resultierenden Zelle ist:

$$(2.0)$$

$$(i, j) = cell(P_0, P_1, P_2);\ P_t \in R^3,\ t = 0..2;$$

$$dx = \frac{|xhigh - xlow|}{m},\ dy = \frac{|yhigh - ylow|}{n}$$

$$i = \frac{\left(\frac{1}{3} \cdot \sum_{t=0}^{2} P_t x\right) - xlow}{dx},\ j = \frac{\left(\frac{1}{3} \cdot \sum_{t=0}^{2} P_t y\right) - ylow}{dy}$$

FTSG unterstützt diese Vorgangsweise, indem eine Verkettung der (nach Zellen sortierten) Dreiecke zu Tristrips separat nach jeder solchen Zelle erfolgt.

5.5.) Operationen auf der Datenbank(*Szenegraph*):

In Kapitel-4.1. wurde bereits darauf hingewiesen, daß die Ausgabestruktur als *Szenegraph* gespeichert wird. Nachdem bei *VRMG* eine Anbindung an *OpenInventor* erfolgt, orientiert sich die Ausgabe an der Struktur von *OpenInventor*. Aus diesem Grund wird an dieser Stelle kurz auf die für die *VRMG*-Modellierung relevanten Konzepte einer *OpenInventor*-Struktur eingegangen:

- *Baumstruktur:* alle Objekte, die das Aussehen und die Aktionen innerhalb einer Szene bestimmen, werden als Knoten in einer Baumstruktur gespeichert. Das *Rendering* einer solchen Struktur besteht im Durchlaufen des Baumes von oben nach unten und von links nach rechts sowie dem Auswerten des aktuellen Knotens.

- *Vererbung:* Eigenschaften wie Texturen oder Material(Farbe), die ebenfalls als Knoten vorliegen, werden defaultmäßig in Abarbeitungsrichtung *vererbt* und zwar solange, bis eine explizite Neudefinition derselben Eigenschaft erfolgt.

- *Knotentypen:* diese sind als Objekte mit spezifischen Parametern anzusehen, die in diesem Zusammenhang verwendeten Knotentypen sind:

- *Complexity-Knoten*: dieser Knoten enthält eine einstellbare Variable im Bereich zwischen 0 und 1, die die optische Qualität der Ausgabe beeinflußt.

- *Texture-Knoten*: Zur Aufnahme von Bilddateien, die als Textur auf alle in der Folge definierten Flächen verwendet wird.

- *Material-Knoten*: verschiedene Attribute, die Farbe und die Reflexionseigenschaften von in der Folge definierten Oberflächen beschreiben, sind in diesem Knoten gespeichert.

- *VertexProperty-Knoten*: hier werden alle Koordinaten der Punkte zusammen mit den dazugehörigen Texturkoordinaten in einem Feld gespeichert.

- *IndexedTriangleStripSet-Knoten*: diese verbinden die Punkte des *VertexProperty-Knoten* über Indices, die dem Kantengerüst der Tristrips entsprechen und letztendlich implizit die Objektoberflächen ausmachen.

- *Separator-Knoten*: diese sind Verzweigungsknoten und haben als Nachfolgeknoten entweder wieder *Separatoren* oder eine Auswahl der obigen Knoten zur Objektbeschreibung.

Das Konzept zum Aufbau des Szenegraphen von *VRMG* beruht auf der Strukturierung in Teilbäume. Dabei ist der Inhalt jeder Zelle *Z* einem Teilbaum mit Separatorknoten *S* zugeordnet:

$$S_k = Z(i, j),\ k = j \cdot (ihigh+1) + i,\ \ 0 \le i < m,\ 0 \le j < n$$

Abbildung-34 zeigt die Ordnung der Zellen auf dem Höhenfeld:

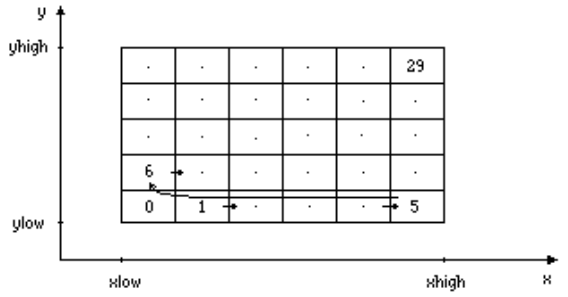


Abbildung 34: Grundriß des Höhenfeldes mit 6 x 5 Zellen.

Der *m x n*-Raster wird also durch *Linearisierung* der Koordinatenindices sequentiell in einer Ebene des Szenegraphen angeordnet.

Die Unterschiede zwischen dem Szenegraph mit und ohne Tristrips werden auf Abbildung-35 gezeigt:

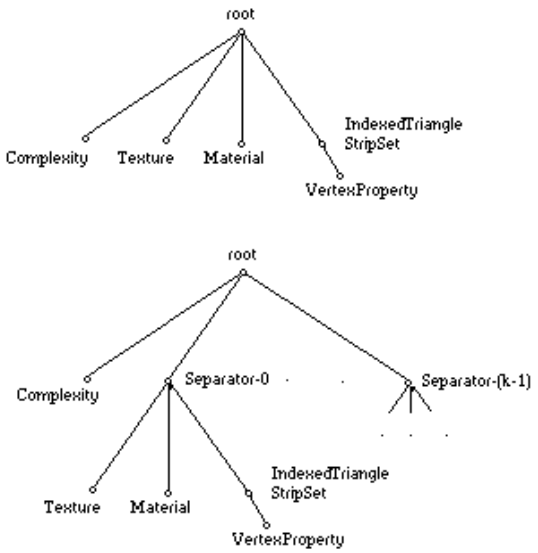


Abbildung 35: *VRMG*-Szenegraph ohne(oben) und mit(unten) Gruppierung nach *k*-Rasterzellen.

Man erkennt, daß für die Gruppierung nach Zellen eine zusätzliche Hierarchieebene notwendig ist, sowie die Anordnung der Zellelemente in links-rechts Ordnung, entsprechend den linearisierten Rasterindices.

Der letzte Optimierungsschritt schließlich verwendet das *OpenInventor* Programm *IVFIX*, dieses dient der Nachoptimierung der Separator-Knoten und führt diese in 2 Phasen aus:

1) Analyse des Szenegraphen hinsichtlich *Kohärenzen*, dadurch werden ggf. Teilgraphen nach verschiedenen gemeinsamen Gesichtspunkten umstrukturiert werden.

2) Flächen werden, falls erforderlich, neu trianguliert und Tristrips gebildet bzw. zusammengefaßt.

Damit ist die Stufe der Geometrieerzeugung abgeschlossen und die fertigen Szenegraphen, die separat für das Höhenfeld und de Straßen erstellt wurden, können in Form eines *OpenInventor*-Files gespeichert werden. Für einen Gesamtüberblick sei auf Abbildung-8 im Kapitel-4 verwiesen.

6.) Implementierung:

Der Inhalt dieses Kapitels ist in 3 Abschnitte gliedert:

- Allgemeines
- Verwendung von *VRMG*
- Implementierung von *VRMG*

Im ersten Teil wird auf die Entwicklungsplattform eingegangen und die Funktion der verwendeten Entwicklungswerkzeuge beschrieben, im zweiten Teil werden alle für den Anwender wichtigen Hinweise für die Benutzung des Programms gegeben und der letzte Teil, der sich an den erfahrenen Programmierer richtet, soll einen Einblick in die Details der Implementierung geben und als Basis für eventuelle Verbesserungen oder Erweiterungen des Programmcodes dienen.

6.1.) Allgemeines:

Grundsätzlich erfolgte die Entwicklung unter Verwendung einer typischen Standardhardware und von kommerzieller Software unter Microsoft Windows. Dadurch ist das Programm für die Mehrzahl möglicher Anwender oder Entwickler, die über die MS-Windows Version von *TGS's OpenInventor* verfügen, verwendbar.

Hardware:

Für die Entwicklung wurde ein Standard PC mit folgender Ausstattung verwendet:

- Intel Celeron Prozessor (433mHz)

- Hauptspeicher mit 128mB SDRAM
- S3 Trio64V+
- Diamond Monster3D-II Hardwarebeschleunigerkarte mit Voodoo2 Chip
- Seagate 20gB Festplatte

Software:

Als Betriebssystem wurde Windows98 verwendet, getestet wurde auch Windows95 und WindowsNT. Als Entwicklungswerkzeuge wurden folgende Softwarepakete eingesetzt:

- Microsoft Visual C++ (Version 6.0, Service Pack-3)
- Open Inventor (Version 2.5.2) von Template Graphic‘s Software
- Arcview GIS (Version 3.1) von Environmental Systems Research Institute Inc.

Zusätzlich wurden folgende Public Domain Programme, die im Rahmen wissenschaftlicher Projekte verschiedener Universitäten entwickelt wurden, in *VRMG* zur Lösung von Teilaufgaben einbezogen:

- *Triangle* (J.R. Shewchuck)
- *FTSG* (Xinyu Xiang)
- *Spline* (J.R. Van Zandt)

Die Funktion und der Verwendungsgrund für die Einbeziehung dieser Programme wurden im Kapitel-5 erklärt, auf Modifikationen, die zur Anpassung an die *VRMG*-Umgebung notwendig waren, wird im Abschnitt „Implementierung“ näher eingegangen.

6.2.) Verwendung von *VRMG*:

Das Hauptprogramm *CityPlane* ist als Konsolenanwendung entwickelt worden. Die *Standardversion*, bei der 3 getrennte Eingabefiles für die Gebäudegrundrisse, den Grundflächenraster und die Straßen verwendet, läßt sich durch folgenden Aufruf von der MSDOS-Kommando-zeile aus starten:

CityPlane *Grundrisse Terrainraster IVFläche Straßenfile IVStraßen*

Beschreibung der Daten-Files in der Kommandozeile(Anm.: Schlüssel-wörter werden fett dargestellt, Variablennamen kursiv) :

1.) Grundrisse:

Die Grundrißdaten umfassen die Grundrisse aller Objekte der virtuellen Stadt, die keine Straßen sind und die separat modelliert werden. Typischerweise handelt es sich hierbei um Gebäudegrundrisse. Dadurch, daß der Grundriß unsichtbar ist, werden alle Grundriß-Polygone aus der Grundfläche ausgeschnitten. Das File hat folgenden Aufbau:

xyz
id, x, y, z
...
...
x, y, z
END
.

.
.
id, x, y, z
...
...
x, y, z
END
END

Jeder Block entspricht einem Polygon, das durch seine eindeutige *id* identifiziert wird und einer (im Gegenuhrzeigersinn) geordneten Sequenz von *x,y,z*-Tripel mit den Koordinaten der Eckpunkte, wobei die *x,y*-Werte auf der Grundfläche liegen und die *z*-Koordinate dem geographischen Höhenwert entspricht.

Beispiel für 2 Polygone mit 4 und 3 Eckpunkten:

xyz
0, 1641.625000, 2472.500000, 179.019000
1664.125000, 2469.094000, 179.019000
1636.750000, 2440.281000, 179.019000
1641.625000, 2472.500000, 179.019000
END
1, 1590.187000, 2483.875000, 179.851000
1634.688000, 2477.594000, 179.851000
1628.750000, 2435.406000, 179.851000
END
END

Kennung: Grundrißfiles erhalten die Endung „.txt“.

2.) Terrainraster:

Dieses File enthält die Beschreibung der Grundfläche in Form eines regulären Rasters, in Übereinstimmung mit der Spezifikation in Kapitel 4 ergibt sich folgende Form für den Aufbau des Fileformates:

ncols *unsigned integer*
nrows *unsigned integer*
xllcorner *float*
yllcorner *float*
cellsize *float*
NODATA_value *val*
z0 z1 ... zn

Bedeutung der Parameter:

ncols.....Anzahl der Rasterzellen in *x*-Richtung.
nrows.....Anzahl der Rasterzellen in *y*-Richtung.
xllcorner.....*x*-Wert des linken unteren Eckpunktes des Rasters.
yllcorner.....*y*-Wert des linken unteren Eckpunktes des Rasters.

cellsize.....Seitenlänge der (quadratischen) Zelle.
NODATA_value....Kennzahl eines ungültigen Höhenwertes.
z.....Höhenwerte in den Mittelpunkten der Rasterzellen, die Reihenfolge ist von links nach rechts und von oben nach unten.

Beispiel für einen 50 x 51-Raster(„vienna_small_grid20.asc“):

```
ncols        50
nrows        51
xllcorner    -50
yllcorner    -50
cellsize     20
NODATA_value  -9999
199.4082 199.5333 199.5407 199.4756 199.3809 199.2845 199.1741 199.1976 199.3443 199.5043 199.6547
199.7639 199.9007 199.9043 199.9537 199.9255 199.8214 199.695 199.4754 199.2238 199.0364 198.624 198.2029
197.6448 197.1225 196.6926 196.0411 195.5348 195.018 194.6501 194.4832 194.2028 194.0424 193.7426 193.4879
193.2673 192.8846 192.6033 192.2883 192.065 191.8838 191.4762 191.108 190.6773 190.4127 190.0339 189.8109
189.6252 189.4981 189.1919 ...
```

Die x Werte gehen also von -50 bis 950 (xllcorner + ncols * cellsize)
Die y Werte gehen also von -50 bis 970 (yllcorner + nrows * cellsize)

Die Daten werden reihenweise abgespeichert: (Es gibt also 51 Zeilen von Daten im file), dabei ist zu beachten, daß sich der Samplepunkt in der Zellenmitte befindet.

Die Zuordnung der Höhenwerten zu den Rasterpunkten ist also:

```
(-40,960)            (940 / 960)
     z0 z1 . . . . . . . . .z49
     .                     .
     .                     .
     .                     .
     z499 . . . . . . . . .z2549
(-40,-40)            (940,-40)
```

Der erste Wert z0 = 199.4082 ist also bei (x = -40, y = 960)

Die allgemeine Berechnungsformel für die Koordinaten Px, Py eines Rasterpunktes P für gegebene Indices i, j für Zeile und Spalte lautet:

$$Px = \left(xllcorner + \frac{cellsize}{2} \right) + i \cdot cellsize$$

$$Py = \left(yllcorner + nrows \cdot cellsize - \frac{cellsize}{2} \right) - j \cdot cellsize$$

$$0 \le i < ncols,\ 0 \le j < nrows.$$

Kennung: Terrainrasterfiles erhalten die Endung „.asc“.

3.) IVFläche:

Dieses ist das Ausgabefile im *OpenInventor*-Fileformat für die fertig modellierte Grundfläche. Der Filename muß die Form „*.iv“ haben, um als für Windows als Inventor-File identifizierbar zu sein. Die Visualisierung erfolgt z.B. mit dem *SceneViewer* von *OpenInventor*.

4.) Straßenfile:

Die Daten dieses Files umfassen alle Attribute zur Erfassung von Straßen und Kreuzungen. Dieses ist das einzige der 3 Eingabe-Files, welches durch einen Übersetzer gefiltert wird und die Daten auf Plausibilität prüft, allerdings ist die Einhaltung der syntaktischen Korrektheit eine Grundvoraussetzung für das erfolgreiche Einlesen der Daten. Das Fileformat orientiert sich an der *Erweiterten Backus-Naur Form(EBNF)*(Anm.: Terminalsymbole und Non-Terminalsymbole sind in Blockbuchstaben geschrieben, Schlüsselwörter fett und Werte kursiv).Zu beachten ist, daß die Definitionsrichtung bei den Straßen vom Anfangs- zum Endpunkt(über die Mittelpunkte der Straßenachse eines Segments) und von links nach rechts(auf den Segmentquerschnitten, Blick in (Definitions-) Richtung der Straße) ist, und bei den Kreuzungen die Ordnung der zulaufenden Straßen im Gegenuhrzeigersinn(um das Kreuzungszentrum) eingehalten werden muß:

ROADNETWORK = { ROAD | CROSSING }

ROAD = **road { road_id** *UINT* { ROADPARTS }+ **}**

ROADPARTS = *CT*
{ start_midpoints POINT
{
[ROADLANES]
}
}

ROADLANES = *LT*
{
lane_id *UINT*
mode *MOD*
width *DOUBLE*
[**texture** *STRING*]
[**size** *DOUBLE*]
[**align** *AT*]
[**height** *DOUBLE*]
[**edge** *ET* [**intervals** *UINT*] [**tension** *DOUBLE*] [PL]]
[**dimensions** *D*]
}

CROSSING = **crossing**
{
crossing_id *UINT*
connections
{
[CONNECTIONS]
}
[**texture** *STRING*]
[**size** *DOUBLE*]
{ **island** POLYGON }
}

CONNECTION = *TT*

{
from*UINT* *CON_STATUS*
to *UINT* *CON_STATUS*
[**radius** *DOUBLE*]
[**texture** *STRING*]
[**size** *DOUBLE*]
{ **polyline***UINT* [PL] [**dimensions** *D*] }
}

POLYGON =
{
[**texture** *STRING*]
[**curbtexture** *STRING*]
[**height** *DOUBLE*]
[**dimensions** *D*]
points PL
}

UINT = unsigned integer
DOUBLE = double
CON_STATUS = **start** | **end**
TT = **normal** | **tcrossing** | **corner**
POINT = *DOUBLE DOUBLE* [*DOUBLE*]
D = **2** | **3**
CT = **straight** | **circle**
MOD = **c** | **w** | **s**
LT = **pkw** | **bike** | **parking** | **island** | **tram** | **sidewalk**
ET = **parallel** | **spline** | **polyline**
AT = **not** | **left** | **right**
PL = **[** POINT { , POINT } **]**

Bedeutung der Symbole und Parameter:

[ ]:	optionales Auftreten des geklammerten Arguments.
{}:	beliebig häufiges Auftreten des geklammerten Arguments.
{}+:	mindestens ein Auftreten des geklammerten Arguments.
|:	Auswahloperator (enweder-oder).
unsigned int:	Ganzzahl größer oder gleich Null.
double:	Realzahl mit doppelter Genauigkeit.
id:	eine einzigartige Identifikationsnummer vom Typ UINT.
size:	Größe einer quadratischen(!) Texture in (Maß-)Einheiten (meter), (default ist 1).
point:	x,y- oder x,y,z-Werte eines Punktes(2D oder 3D, nach Kontext).
D:	Dimension einer Polyline, default ist 2D. Der z-Wert wird nur verwendet, wenn "dimension 3" angegeben wird.
width:	Distanz von der des Fahrstreifenrandes zur Mittelline(„Breitenposition“).
height:	relative Höhendifferenz zwischen aufeinanderfolgenden Fahrstreifen oder für Verkehrsinseln als relative Höhendifferenz zur Fahrbahn.
radius:	Für Abrundungsbögen als Radius, ein Wert von 0 (default), führt zu einer programminternen Berechnung mit Verwendung des Maximalradius‘.
CT:	Segmentverlauf zum nächsten Straßenmittelpunkt.
texture:	Diese muß zumindest für das erste Segment am Straßenanfang in Form einer Bilddatei definiert werden, für undefinerte Texturen wird die Textur des Vorgängersegments verwendet.
LT:	Verwendungszweck des Fahrstreifens z.B. „pkw“ für eine Fahrbahn.

MOD: wird zur Kennzeichnung der Gültigkeit von Fahrstreifenpositionen verwendet:
c: „continuing“ - für das vorige und nächste Segment gültig.
w: „widening“ - nur für nächstes Segment gültig.
s: „shrinking“ – nur für voriges Segment gültig.

ET: Verlauf der Kante eines Fahrstreifens:
parallel: wie Mittellinie(default).

spline: programminterne generierung einer Spline-Übergangskurve.

polyline: Definition einer beliebigen Verlaufsform durch einen Polygonzug.

AT: Ausrichtung der Textur auf einem Fahrstreifen:
left: an linker Kante(Wiederholung oder Schnitt rechts).
right: an rechter Kante(Wiederholung oder Schnitt links).
not: die Textur wird gestreckt oder gezerrt(default.

PL: Spezifikation einer Kante durch einzelne (geordnete) Punktmenge(Polygonzug).
Achtung: bei Kreuzungen folgt auf „polyline“ ein UINT-Wert, der der Fahrstreifennummer des Abrundungsteils entspricht: dadurch wird die Übergangskante „adressiert“: 0 – innerste Übergangskante am Fahrbahnteil, nummeriert nach außen.

con_status: Straßenanschluß an die Kreuzung: start - erster Straßenquerschnitt, end - letzter Straßenquerschnitt.

TT: Form des Übergangsteils der Abrundung von Kreuzungen:
normal: der Abrundungsteil wird aus dem Randstein und einer Fläche zwischen Randstein und Außenrand des Abrundungsteils gebildet.
t-crossing: vorzugsweise für die durchgehende Straße bei einer Einmündung, dabei werden (sofern möglich) alle gemeinsamen Streifen der beiden beteiligten Straßenäste fortgeführt.
corner: wie Typ –normal-, allerdings wird ein Eckpunkt(=Schnittpunkt der Außenkanten der beiden beteiligten Straßenäste) verwendet.

Spline-Parameter:

intervals: Verfeinerungswert für Spline-Kurve, dabei wird das Segment in n-Teile gleichmäßig unterteilt und für jede Intervallgrenze ein Stützpunkt berechnet(default ist 10).

tension: Wert kleiner als 1 ergeben bikubischen Spline, große Werte approximieren eine Polyline(default ist 1).

Beispiel für ein gerades Straßensegment mit 4 Spuren(Randstein, Fahrbahn(2 Fahrspuren), Randstein) und Anfangs und Endquerschnitt:

```
road {    road_id 4
          straight
          {    start_midpoints 60.2565 -6.46531
          {    sidewalk { lane_id 0 mode c width 8 texture „pave.jpg" }
               pkw { lane_id 1 mode c width 7.8 texture "as_mid_dash.jpg" size 10 height-0.2 }
               pkw { lane_id 5 mode c width 0 texture "as_mid_dash.jpg" size 10 height 0 }
               sidewalk { lane_id 2 mode c width -7.8 texture "pave.jpg" height 0.2 }
               sidewalk { lane_id 3 mode c width -8 texture "pave.jpg" }
          }
          }
          straight
          {    start_midpoints 63.6591 -4.36239 {
          {    sidewalk { lane_id 0 mode c width 8 texture „pave.jpg" }
               pkw { lane_id 1 mode c width 7.8 texture "as_mid_dash.jpg"  size 10 height-0.2 }
               pkw { lane_id 5 mode c width 0 texture "as_mid_dash.jpg"   size 10 height 0 }
               sidewalk { lane_id 2 mode c width -7.8 texture "pave.jpg"      height 0.2 }
               sidewalk { lane_id 3 mode c width -8 texture "pave.jpg" }
```

```
        }
        }
    }
```

Kennung: Straßenbeschreibungsfiles werden im Rahmen von *VRMG* mit der Endung „.dat“ versehen.

5.) IVStraßen:

Dieses ist das Ausgabefile im *OpenInventor*-Fileformat für die fertig modellierten Straßen und Kreuzungen. Der Filename muß die Form „*.iv“ haben, um für Windows als Inventor-File identifizierbar zu sein. Die Visualisierung erfolgt z.B. mit dem *SceneViewer* von *OpenInventor*.

6.3.) Implementierung von *VRMG*[38]:

Die Implementierung von *VRMG* wurde in Form des *Visual C++ Projects CityPlane* realisiert. Das Projekt ist von *modularem Aufbau* und hat *objektorientierte Struktur*. Für die Fileerstellung gilt die Konvention, daß jede Klasse ein eigenes gleichnamiges File mit dazugehörigem Headerfile besitzt. Zusätzlich zu den *CityPlane* – Klassen wurden folgende externe Klassen verwendet:

- *STL-Vektoren*: diese entstammen der *Standard C++ Library(STL)* und unterstützen dynamische Arrayoperationen.
- *Inventor-Objekte*[38]: aus dem Sortiment des *objektorientierten 3D-Toolkits* von OpenInventor wurden ausschließlich sogenannte *Datenbank-primitive* für Tristrip-Flächen und deren Eigenschaften verwendet.
- *PrivateErrHandler*: Klasse zur Protokollierung von Fehlern.
- *IfFixer*: *Interface*-Klasse zur Nutzung der *IVFIX*-Funktionalität.

Die Verwendung von *MFC-Klassen* wurde aus Gründen der *Portierbarkeit* vermieden.

6.3.1.) Umsetzen der Bestandteile von *VRMG*-Objekten in C++ Klassen:

Die in Kapitel-4 bzw. Kapitel-5 vorgestellten Komponenten des Modells und die darauf aufbauenden Operationen lassen sich gut als Hierarchie von *Objektklassen* umsetzen. Zusätzliche Klassen wurden für die in Standard-C geschriebenen, externen Optimierungsprogramme *Triangle* und *FTSG* Schnittstellenklassen geschrieben, die dazu dienen, die Ein- und Ausgabedatenstrukturen zu konvertieren. Die gegenseitige Abhängigkeit der *CityPlane*-Klassen zusammen mit der Klassenhierarchie der Straßenklasse ist auf Abbildung-36 zu sehen:

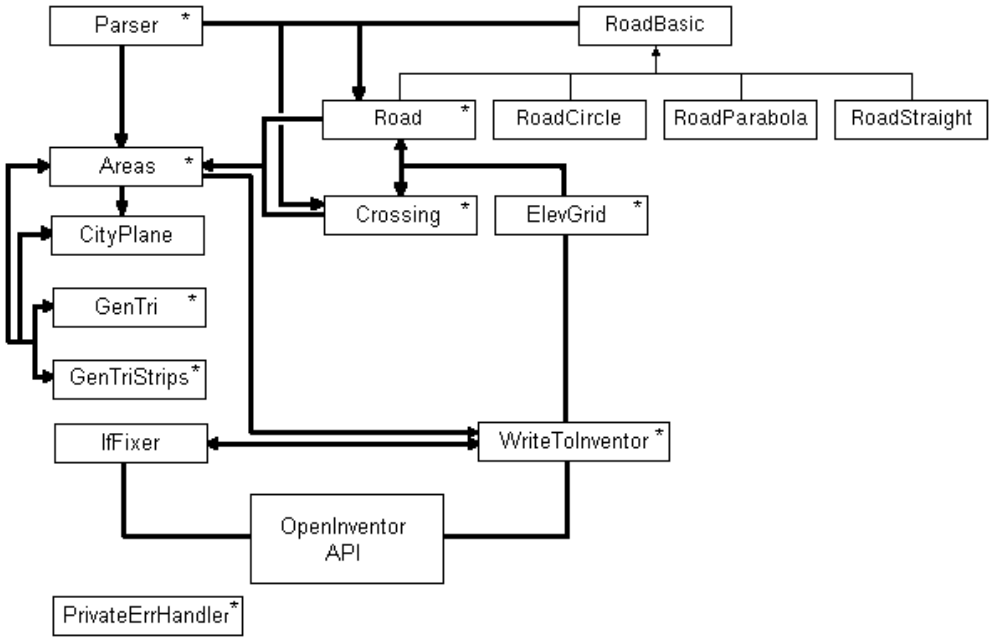


Abbildung 36: Abhängigkeiten der CityPlane-Klassen (Anm.: mit "*" bezeichnete Klassen verwenden die Klasse PrivateErrHandler)

6.3.2.) Headerfiles der Klassen mit Beschreibung:

Aufgabe dieses Abschnitts ist es, die wichtigsten Klassen und deren Methoden vorzustellen. Dadurch eröffnen sich viele Möglichkeiten, den Programmablauf durch klassenspezifische Einstellungen(**Set...)** zu beeinflussen. Das Hauptprogramm *main.cpp*, das im Project *CityPlane* enthalten ist, zeigt beispielhaft die Verwendung verschiedener Settings auf der *Areas*- und *Plane*-Klasse. Nachfolgend sind die Deklarationen der für den Umgang mit *CityPlane* wichtigen Klassen aufgelistet, wobei nur jene Methoden(fett gedruckt) auf der rechten Seite erklärt werden, die zur Steuerung des Ablaufs oder zum Setzen von Attributen der Ausgabe beitragen bzw. bei der Verwendung als externem Modul bedeutsam sind:

Areas Class Reference……Verwaltungsklasse für Straßen

Public Methods
Areas ()
virtual **~Areas** ()
bool **SetInFile** (char *fname).....Eingabefile der Straßen("*.dat")
bool **SetOutFile** (char *fname).....Inventor-Ausgabefile des Straßenmodells
void **SetElevGrid** (**ElevGrid** *newgrid).....Zeiger auf dynamisches Höhenfeldobjekt
void **SetGridMode** (bool on).....Schalter 2D/3D
void **SetGridClipMode** (bool on).....Schalter für Clipping am Höhenfeldrand
void **SetRoadCorrectureAtCrossing** (bool on).....Schalter für automatische Kreuzungskorrektur
void **SetSwapYZ** (bool s).....Schalter zwischen *Skitter*- und *Jack*-Koordinaten
void **SetTriMinAngle** (double TriMinAngle=20).....Minimalwinkel-Bedingung für Triangulierung

void **SetTriMaxArea** (double Area).....Maximalflächen- Bedingung für Triangulierung
void **SetTriMethod** (**trimethod** tm).....Schalter für Triangulationsmethode(-> GenTri)
void **SetTexFactor** (float factor).....Minifikationswert für Texturkoordinaten
void **SetTextureMode** (bool on).....Ein/Aus-Schalter für Texturierung
bool **SetTexture** (char* filename).....Filename für Textur setzen
void **SetTextureQuality** (float value).....Qualitätsmaß zwischen 0,1 für Texturmapping
void **SetMaterialMode** (bool on).....Schalter für Verwendung eine Materialknotens
void **SetMaterial** (float diffuse[3], float specular[3], float ambient[3], float emissive[3], float shininess, float transparency).....Attribute für Materialeingenschaften
void **SetStripMode** (bool mode).....Ein/Aus-Schalter für Tristrip-Generierung
void **SetNormalsMode** (bool mode).....Ein/Aus-Schalter für Normalvektorberechnung
void **SetNumGroups** (UINT numX, UINT numY).....Anzahl der Gruppen in x,y für Tristrips
void **SetNumBoxes** (UINT **NumGrperBoxX**, UINT **NumGrperBoxY**)...Anzahl der Gruppen in x,y für die ein separates File der Straßen erstellt wird.
bool **ReadAreas** ().....Einlesen der Straßendaten
bool GetPoly (**PPTR**& p, UINT **n**, UINT l)
UINT **GetNumRoads** ().....Anzahl der generierten Straßen abfragen
UINT **GetNumCrossings** ().....Anzahl der generierten Kreuzungen abfragen
PPTR GetRoadFootprint (UINT numRoad).....Umriß der Straßen als Polygone anfordern
PPTR GetCrossingFootprint (UINT numCrossing).....Umriß der Kreuzungen als Polygone anfordern
bool **SetTriangulationClass** (**GenTri** *TriClass).....Zeiger auf dynamisches Triangulationsobjekt
UINT **BuildIvGraph** (double *texcoords, **DPTR** pointlist=NULL, UINT numberofpoints=0, **UPTR** trianglelist=NULL, UINT numberoftriangles=0).....Inventor-Szenegraph der Straßen aufbauen
bool **SetTriStripperClass** (**GenTriStrips** *StripClass).....Zeiger auf dynamisches Tristripper-Objekt
bool **TriangulateAreas** ().....Triangulierung der Straßen

CityPlane Class Reference.....Zentrale Managementklasse

Public Methods
CityPlane ()
virtual **~CityPlane** ()
bool **SetInFileBuildings** (char *filename).....Eingabefile für Gebäudegrundrisse(“*.txt”)
bool **SetInFileElevGrid** (char *filename).....Eingabefile für Höhenfeldraster(“*.asc”)
bool **SetOutFile** (char *filename).....Ausgabefile des Höhenfeldmodells
void **SetCutAreas** (bool mode).....Ein/Aus-Schalter für Ausschneiden von Flächen
bool **SetSnapDistance** (double dist).....Toleranzwert für Datenunschärfe(ε-Umgebung)
void **SetSwapYZ** (bool s).....Schalter zwischen *Skitter*- und *Jack*-Koordinaten
void **SetTriMinAngle** (double TriMinAngle=20).....Minimalwinkel-Bedingung für Triangulierung
void **SetTriMaxArea** (double Area).....Maximalflächen- Bedingung für Triangulierung
void **SetTriMethod** (**trimethod** tm)..... Schalter für Triangulationsmethode(->GenTri)
void **SetNumGroups** (UINT numX, UINT numY)..... Anzahl der Gruppen in x,y für Tristrips
void **SetGridOutput** (bool on).....Ein/Aus-Schalter der Rasterpunkte in der Ausgabe
void **SetGridResolution** (double u, double v).....Einstellen der Zellgröße in x,y für Resampling
void **SetGridClipMode** (bool on).....Schalter für Clipping am Höhenfeldrand
void **SetTexFactor** (float factor).....Ein/Aus-Schalter für Texturierung
void **SetTextureMode** (bool on).....Ein/Aus-Schalter für Texturierung
bool **SetTexture** (char* filename).....Filename für Textur setzen
void **SetTextureQuality** (float value).....Qualitätsmaß zwischen 0,1 für Texturmapping
void **SetMaterialMode** (bool on)..... Schalter für Verwendung eine Materialknotens
void **SetMaterial** (float diffuse[3], float specular[3], float ambient[3], float emissive[3], float shininess, float transparency) .....Attribute für Materialeingenschaften
void **SetStripMode** (bool mode).....Ein/Aus-Schalter für Tristrip-Generierung
void **SetNormalsMode** (bool mode) .....Ein/Aus-Schalter für Normalvektorberechnung
void **SetAreas** (**Areas** *areas).....Zeiger auf dynamisches Straßenobjekt
void **SetTriangulationClass** (**GenTri*** TriClass).....Zeiger auf dynamisches Triangulationsobjekt
bool **TriangulatePlane** ().....Triangulierung des Höhenfeldes

Crossing Class Reference.....Klasse zur Erzeugung von Kreuzungen

Public Methods
Crossing ()
virtual **~Crossing** ()
UINT **SetCrossing** (**CrossBuf** *crossing).....Generierung der Kreuzung
bool **AddRoad** (**Road** *road, bool start).....Knotenpunktsarm hinzufügen
void **SetElevGrid** (**ElevGrid** *newgrid).....Zeiger auf dynamisches Höhenfeldobjekt
void **SetRoadCorrecture** (bool on).....Ein/Aus-Schalter für automatische Kreuzungskorrektur
bool GetPolyPoints (**DPTR**& points, UINT &num)
CPTR GetIslandTexture (UINT islandnum)
CPTR GetIslandCurbTexture (UINT islandnum)
CPTR GetTexture (UINT polynum)

ElevGrid Class Reference.....Klasse zur Verwaltung des Höhenfeldes

Public Methods
ElevGrid ()
virtual **~ElevGrid** ()
void **SetNormalsMode** (bool mode).....Ein/Aus-Schalter für Normalvektorberechnung
bool **GetNormalsMode** ().....Status für Normalvektorberechnung abfragen
void **SetClipMode** (bool mode).....Schalter für Clipping am Höhenfeldrand
bool **GetAltitude** (double x, double y, double& z).....Höhenwert für Punkt in x,y anfordern
bool **GetNormal** (float* **n**, double x, double y).....Normalvektor für Punkt x,y anfordern
bool GetPointlist (DPTR& pointlist, UINT& num)
void **GridNormals** ().....Normalvektoren für Rasterpunkte generieren
UINT **EdgeLength** (UINT i)
bool **ReadGroundRect** (FILE *file).....Höhenfeldraster einlesen
double GetValueAt (Coord c, UINT e, UINT i)
UINT **GetNumPoints** (bool with_edges).....Anzahl der Höhenfeldraster-Punkte abfragen
void **GetSize** (double& xlen, double& ylen).....Dimension des Höhenfeldes abfragen
void **Resample** (double mx, double my).....Höhenfeld resampeln

GenTri Class Reference.....Front-End Klasse für *Triangle*

Public Methods
GenTri ()
virtual **~GenTri** ()
void **StartNewTriangulation** (bool reset_switches=false)....Reset-Schalter mit/ohne Einstellungsänderung
bool **AddPoly** (**POLYGON*** poly, bool cut).....Polygon hinzufügen (Ausschneiden: cut=true)
void **AddPoints** (**DPTR** points, UINT num).....Punkt hinzufügen
void **SetTriMinAngle** (double TriMinAngle=20).....Minimalwinkel-Bedingung
void **SetTriMaxArea** (double Area).....Maximalflächen-Bedingung
void **SetTriMethod** (**trimethod** tm).....mögliche Methoden: dc-divide&conquer, sl-sweepline, ii-incremental insertion
bool **SetSnapDistance** (double dist).....Toleranzwert für Datenunschärfe(ε-Umgebung)
void **GetPoints** (**DPTR**& pointlist, UINT& num).....Punkte aus der Triangulierung anfordern
void **GetIndices** (**UPTR**& trianglelist, UINT& num).....Indizierte Kanten aus der Triangulierung anfordern
bool **DoTriangulation** ().....Triangulierung starten

GenTriStrips Class Reference.....Front-End für *FTSG*

Public Methods
GenTriStrips ()
virtual ~GenTriStrips ()
void **StartNewStripping** ()..... Reset Schalter
bool SetBoundingRectangle (float left, float bottom, float right, float top)
bool **SetNumBoxes** (UINT NumX, UINT NumY)..... Anzahl der Gruppen in x,y
bool **SetPoints** (double *Points, UINT num)..... Koordinaten der Eingabe-Punkte
bool **SetIndices** (UINT *Indices, UINT num)..... Indices der Eingabe-Kanten
void **SetSkipIsolatedTri** (bool skip)..... Ein/Aus-Schalter zum Verwerfen isolierter Dreiecke
bool **GenerateStrips** ()..... TriStrips getrennt nach Gruppen bilden
SPTR& **GetStrips** ()..... TriStrips anfordern
void **SetFTSG_QUIET** (bool tf).....
void **SetFTSG_BFS** (bool tf) .
void **SetFTSG_DFS** (bool tf) .
void **SetFTSG_GROUP** (bool tf) .
void **SetFTSG_CONCAT** (bool tf) .
void **SetFTSG_FAN** (bool tf) .
void **SetFTSG_SEQ** (bool tf) . Einstellungen für *FTSG*-
void **SetFTSG_SGI** (bool tf) . Bedeutung der Werte in
void **SetFTSG_ALT** (bool tf) . Referenz-[14]
void **SetFTSG_SYNC** (bool tf) .
void **SetFTSG_GEOM** (bool tf) .
void **SetFTSG_DUP** (bool tf) .
void **SetFTSG_OGL** (bool tf) .
void **SetFTSG_OPT** (bool tf) .
void **SetFTSG_CLEANUP** (bool tf).....

IfFixer Class Reference..... Front-End für *IVFIX*

Public Types
enum ReportLevel { NONE, LOW, HIGH }

Public Methods
IfFixer ()
~IfFixer ()
void **setReportLevel** (ReportLevel level, FILE *fp).....
void **setStripFlag** (SbBool flag) .
void **setVertexPropertyFlag** (SbBool flag) . externes Modul - Bedeutung
void **setNormalFlag** (SbBool flag) . in *OpenInventor*-Dokumentation
void **setTextureCoordFlag** (SbBool flag) .
void **setMatchingFlag** (SbBool flag) .
SoNode* **fix** (SoNode *root).....

Parser Class Reference..... Übersetzer für Straßenfile

Public Methods
Parser ()
virtual **~Parser** ()
bool **ParseFile** (char *filename, vector<**RoadBuf** *> &roadinput, vector<**CrossBuf** *> &crossinput)
..... Übersetze Straßenfile

PrivateErrHandler Class Reference..... Klasse für Fehlerprotokolle

Public Methods
PrivateErrHandler ()
std::ostream& GetCurrentLogStream ()
void **PrintLogStream** (bool overrideconsole).....
void **InitLogStream** (const char *filename, bool _logtoconsole) . externes Modul - Bedeutung
void **SetConsoleStreamHandler** (LogStreamToConsoleFunc handler) . siehe *Yare*-Dokumentation
void **SetLogStream** (bool _logtoconsole).....

Road Class Reference.....Oberklasse für Straßengenerierung

Inherits **RoadBasic**.

Public Methods
Road ()
virtual **~Road** ()
UINT **SetRoad** (**RoadBuf** *part).....Straßengenerierung starten
UINT **ResetRoad** (RoadBuf *part).....Straßenparameter neu setzen
void **SetElevGrid** (ElevGrid *newgrid).....Zeiger auf dynamisches Höhenfeld-Objekt
bool **GetPolyPoints** (UINT polynum, **DPTR**& points, UINT &num, UINT &numrgt, **DPTR** &texpoints)
.....Polygone für Fahrstreifen anfordern
CPTR **GetTexture** (UINT polynum).....Textur für Fahrstreifen anfordern
double GetTextureSize (UINT polynum)
CPTR GetCTexture (UINT polynum)
double GetCTextureSize (UINT polynum)
void **GetCrossingSection** (vector<**_3DPOINT** *> &pts, bool start).....Start-/Endquerschnitt anfordern
bool **GetValidTangents** (UINT l, bool start, bool left, **_2DVECTOR** &**tin**, **_2DVECTOR** &**tout**)
.....Tangenten für erstes/letztes Segement anfordern
int GetOuterLanes (bool start, bool left, **LaneType** lt, double &reflevel, vector<double> &cross, vector<double> &height, vector<char *> &texture, vector<double> &texsize, vector<**Alignment**> &texclip)
int GetLUT (UINT lid, UINT i)
void GetLUTLaneIds (vector<UINT>& lane_ids)
bool BuildDefaultLUT ()
bool BuildAdvancedLUT (RoadBuf *part)
void HandleBends (UINT pindex)

Static Public Methods
void SetTexCoords (UINT l, vector<**POLYGON** *>, double size, **DPTR**& texcoords, **Alignment** clip=not, **Alignment** align=not)
bool GetGamma (**_2DVECTOR** v1, **_2DVECTOR** v2, double &gamma)
void AdjustString (**CPTR** &filename)

RoadBasic Class Reference.....Basisklasse für Straßensegmente

Inherited by **Road**, **RoadCircle**, **RoadParabola**, and **RoadStraight**.

Public Methods
RoadBasic ()
virtual **~RoadBasic** ()
void **Set** (**_2DPOINT** M[2], vector<double> cross0, vector<double> crossn, vector<double> height0, vector<double> heightn, **_2DVECTOR** t1, **_2DVECTOR** t2, double gamma_in, double gamma_out, **_3DPlist** *edges=NULL, UINT *edgestat=NULL)
virtual bool **SetPart** ()
virtual void **GetLocalCoord** (**_3DPOINT PE**, **_3DPOINT P**, double &a, double &b)
virtual void **GetGlobalCoord** (**_3DPOINT PE**, double a, double b, **_3DPOINT** &**P**)
void **TransformToGlobal** (**_3DPOINT PE**, **_3DPlist**& localpolyline, double fa, double fb)
virtual void **RefineWithPolyline** (UINT l, vector<double> &lpoints, vector<double> &rpoints)

virtual void **Refine** (UINT l, vector<double> &lpoints, vector<double> &rpoints, vector<BYTE> &truezl, vector<BYTE> &truezr, vector<double> &ltexpts, vector<double> &rtexpts, double texsize, **Alignment** clip=not, **Alignment** align=not)
bool **AdjustPolylineEdge** (UINT l, bool left, bool &newfirst, bool &newlast)
virtual UINT **CutOff** (UINT l, **_2DPOINT P**, **_2DVECTOR** v, **_2DPOINT** &S1, **_2DPOINT** &S2)
void **SetRefinement** (double value)
UINT GetNumLanes ()
_3DPOINT GetCornerPoint (UINT lanenum, UINT p)
void **GetCrossingSection** (vector<**_3DPOINT** *> &pts, bool start)
void **GetTangents** (_2DVECTOR &in, _2DVECTOR &out)
_2DVECTOR **GetMidpoint** (bool start)
CurbStat CurbStatus (int l1, int l2)
bool **EdgeEnabled** (UINT l, bool leftedge)
virtual void **SetLowerLimit** (UINT l, **_2DPOINT** S, bool leftedge)
virtual void **SetUpperLimit** (UINT l, **_2DPOINT** S, bool leftedge)

RoadCircle Class Reference.....Klasse für kreisbogenförmige Segmente

Inherits **RoadBasic**.

Public Methods
RoadCircle ()
virtual **~RoadCircle** ()
virtual bool **SetPart** ()
virtual void **GetLocalCoord** (**_3DPOINT PE**, **_3DPOINT P**, double &a, double &b)
void **GetLocalRefCoord** (**_3DPOINT** PS, **_3DPOINT** PB, **_3DPOINT** PL, **_3DPOINT P**, **_2DPOINT** Mc, **circstat** stat, double lim, double **alpha**, double delta, double rc, double &a, double &b)
virtual void **GetGlobalCoord** (**_3DPOINT PE**, double a, double b, **_3DPOINT** &**P**)
virtual void **RefineWithPolyline** (UINT l, vector<double> &lpoints, vector<double> &rpoints)
virtual void **Refine** (UINT l, vector<double> &lpoints, vector<double> &rpoints, vector<BYTE> &truezl, vector<BYTE> &truezr, vector<double> &ltexpts, vector<double> &rtexpts, double texsize=1,**Alignment** clip=not, **Alignment** align=not)
virtual void **SetLowerLimit** (UINT l, **_2DPOINT** S, bool leftedge)
virtual void **SetUpperLimit** (UINT l, **_2DPOINT** S, bool leftedge)

RoadParabola Class Reference.....Klasse für parabolische Segmente

(Anmerkung: Implementierung der Methoden nicht vollständig)

Inherits **RoadBasic**.

Public Methods
RoadParabola ()
virtual ~**RoadParabola** ()
virtual bool **SetPart** ()
virtual void **GetLocalCoord** (**_3DPOINT PE**, **_3DPOINT P**, double &a, double &b)
virtual void **GetGlobalCoord** (**_3DPOINT PE**, double a, double b, **_3DPOINT** &**P**)
virtual void **RefineWithPolyline** (UINT l, vector<double> &lpoints, vector<double> &rpoints)
virtual void **Refine** (UINT l, vector<double> &lpoints, vector<double> &rpoints, vector<BYTE> &truezl, vector<BYTE> &truezr, vector<double> &ltexpts, vector<double> &rtexpts, double texsize, **Alignment** clip, **Alignment** align)
virtual void **SetLowerLimit** (UINT l, **_2DPOINT** S, bool leftedge)
virtual void **SetUpperLimit** (UINT l, **_2DPOINT** S, bool leftedge)

RoadStraight Class Reference.....Klasse für gerade Segmente

Inherits **RoadBasic**.

Public Methods
RoadStraight ()
virtual ~**RoadStraight** ()
virtual bool **SetPart** ()
virtual void **GetLocalCoord** (**_3DPOINT PE**, **_3DPOINT P**, double &a, double &b)
void **GetLocalRefCoord** (_3DPOINT PS, _3DPOINT PE, _3DPOINT P, double &a, double &b)
virtual void **GetGlobalCoord** (**_3DPOINT PE**, double a, double b, **_3DPOINT** &**P**)
virtual void **Refine** (UINT l, vector<double> &lpoints, vector<double> &rpoints, vector<BYTE> &truezl, vector<BYTE> &truezr, vector<double> &ltexpts, vector<double> &rtexpts, double texsize=1, **Alignment** clip=not, **Alignment** align=not)
virtual void **RefineWithPolyline** (UINT l, vector<double> &lpoints, vector<double> &rpoints)
virtual UINT **CutOff** (UINT l, **_2DPOINT P**, **_2DVECTOR** v, **_2DPOINT** &S1, **_2DPOINT** &S2)
virtual void **SetLowerLimit** (UINT l, **_2DPOINT** S, bool leftedge)
virtual void **SetUpperLimit** (UINT l, **_2DPOINT** S, bool leftedge)

WriteToInventor Class Reference.....Klasse für Inventor-Ausgabe

Public Methods
WriteToInventor ()
virtual ~WriteToInventor ()
void **SetTexFactor** (float factor).....Minifikationswert für Texturkoordinaten
void **SetTextureMode** (bool on).....Ein/Aus-Schalter für Texturen
bool **SetTexture** (char* filename).....Filename der Textur
void **SetTextureQuality** (float value).....Qualitätsmaß zwischen 0,1 für Texturemapping
void **SetSwapYZ** (bool s).....Schalter zwischen *Skitter* und *Jack* Koordinaten
void **SetNormalsMode** (bool mode).....Schalter für Normalverktorberechnung
void **SetMaterialMode** (bool on).....Schalter für Verwendung eines Materialknotens
void **SetMaterial** (float diffuse[3], float specular[3], float ambient[3], float emissive[3], float shininess, float transparency).....Attribute des Materialknotens setzen
bool **SetElevGrid** (ElevGrid *newgrid).....Zeiger auf dynamisches Höhenfeld-Objekt
bool **WriteGeometry** (**ModeSwitch** msw, char *ivFilename, double *pointlist=NULL, UINT numberofpoints=0, UINT *trianglelist=NULL, UINT numberoftriangles=0, double *texcoords=NULL, **SPTR** StripGroups=NULL, UINT NumGrX=0, UINT NumGrY=0, UINT NumGrperBoxX=0, UINT NumGrperBoxY=0).....ModeSwitch msw: a - Inventorgraph erweitern, ...bw - Inventorgraph aufbauen, in File speichern w - aktuellen Szenegraph in File speichern

6.3.3.) Zusatzprogramme mit kurzer Beschreibung:

Außer diesen Klassen sei noch kurz auf folgende Module eingangen, die Teil des Projects *CityPlane* sind und für die Lösung von Teilaufgaben eingesetzt werden, wobei die entsprechenden Algorthmen bereits im Kapitel-5 vorgestellt wurden und an dieser Stelle nur die Eigenschaften der Software und deren Einbindung erläutert wird:

Triangle:

Dieses Programm ist in Standard-C implementiert und grundsätzlich portabel gehalten, durch verschiedene Preprozessor-Definitionen läßt sich eine Anpassung an zahlreiche Plattformen erreichen. In diesem Abschnitt beschränkt sich die Erklärung auf jene Einstellungen, die für die Anbindung an *CityPlane* vorgenommen wurden:

#define TRILIBRARY

dadurch wird das Programm für die Verwendung als externe Prozedur compiliert.

```
#define NO_TIMER
```

diese Funktion ist unter Windows nicht verfügbar.

Die *Triangle*-Prozdur für die Triangulierung hat folgende Deklaration:

```
void triangulate(char *, struct triangulateio *,
                 struct triangulateio *,
                 struct triangulateio *);
```

Der Default-Algorithmus für die Triangulierung ist *Divide&Conquer*. Für die einfache Triangulierung ohne Winkel- oder Flächen-Constraints wird die Prozedur mit dem String „Qpzo3“ als ersten Parameter aufgerufen. Die Bedeutung der Attribute ist folgende:

- Q: quiet, dadurch wird eine Ausgabestatistik auf die Standardausgabe unterdrückt.
- p: Verwendung eines *PSLG*(vgl. Kapitel-5, Triangulierung)
- z: die Arrays werden von Null weg indiziert.
- o3: dadurch wird die 3.Koordinate der Punkte mitgenommen, obwohl die Triangulierung stets ausschließlich auf den *x*- und *y*-Werten erfolgt.

Neben diesen Defaultattributen sind durch die Settings in manchen Klassen(siehe oben) noch weiter Optionen zugänglich:

- qα: α ist dabei der Wert eines spezifizierten Minimalwinkels.
- a*F*: *F* ist der Wert für die Maximalfläche eines Dreiecks.
- F: Triangulierung erfolgt nach *Sweepline*-Algorithmus.
- i: Triangulierung erfolgt nach *Incremental Insertion*-Algorithmus.

Der Typ der restlichen Parameter ist die Daten-Übergabestruktur von *Triangle*, wobei der 2.Parameter ein Zeiger auf die Eingabedaten und der 3.Parameter ein Zeiger auf die Daten aus der Triangulierung ist. Der 4.Parameter schließlich ist ein Zeiger auf das berechnete Voronoi-Diagramm(dieses wird im Rahmen von *CityPlane* nicht verwendet).

Im Rahmen der Entwicklung von *CityPlane* wurde zur flexibleren Verwendung der Prozedur *triangulate* die Schnittstellenklasse *GenTri* geschaffen, wodurch folgende Möglichkeiten eröffnet wurden:

- objektorientierte, dynamische Datenstrukturen durch Verwendung von STL-Vektoren.
- Speicherung von Punkten und Indices der (optionalen) Kantenverbindungen (Polygone) in der Klasse, dadurch können diese während des Programmdurchlaufs akkumuliert werden und durch die Anweisung *DoTriangulation* in einem Schritt trianguliert werden.
- Polygone können als Hohlflächen übergeben werden die Ermittlung eines Punkt innerhalb dieses Polygons, der zum Ausschneiden der Fläche verwendet wird, erfolgt durch *GenTri*.
- Durch einen Reset-Aufruf können alle bis zu diesem Zeitpunkt gespeicherten Datenwerte gelöscht werden, die Einstellungen für die Triangulierung können dabei entweder erhalten bleiben oder wieder auf den Standardwert gesetzt werden.

Bei der Arbeit mit *Triangle* bzw. *GenTri* sind folgende Probleme aufgetreten:

- Die Methode zur Ermittlung eines (Innen-)Punktes einer Hohlfläche versagt insofern, als ein Punkt auf der Kante des Polygons berechnet wird. In einem solchen Fall trifft *Triangle* eine Zufallentscheidung darüber, auf welcher Seite der Kante der Punkt anzunehmen ist. Im ungünstigen Fall werden also die Dreiecke falsch ausgeschnitten. Der Grund für dieses Fehlverhalten liegt einerseits in numerischen Ungenauigkeiten aufgrund ungünstiger Datenkonstellation(schleifender Schnitt) oder in einer inkorrekten definierten Polygonfläche(Ordnung der Punkte und Kanten nicht im Uhrzeigersinn, Überschneidungen von Kanten, Fläche des Polygons verschwindend klein usw.)
- Der Triangulationsalgorithmus terminiert nicht korrekt. Der Grund dafür war bei den Testläufen von *CityPlane* ebenfalls eine ungünstige Datenkonstellation, die dazu geführt hat, daß die Länge von Kanten, die während der Triangulierung erzeugt wurden, verschwindend klein wurde und somit die (Delaunay-)Tests nicht mehr durchführbar wurden(vgl.Kapitel-5, Triangulierung).

Insgesamt betrachtet waren die Ergebnisse der Triangulierung bei den Testläufen zufriedenstellend, was letztendlich auch der Grund dafür ist, das *Triangle* einer der prominentesten Vertreter unter den über das WWW frei verfügbaren Triangulationsprogrammen ist.

FTSG:

Dieses Programm ist ebenfalls in Standard-C implementiert. Die Timerfunktionen von UNIX-Systemen mußten, so wie bei *Triangle*, ausgeklammert werden, da sie in dieser Form unter Windows nicht nutzbar sind. Ansonsten entspricht das Programm den gestellen Anforderungen und ist als externe Prozedur problemlos in ein C/C++ -Project integrierbar, wobei die Deklaration der Schnittstellenprozedur von folgender Form ist:

```
void ftsg(int *groups, int numGroups, PLIST *pLists, int numVerts,
        int numFaces, ISTREAM **strips, int *numStrips, STAT *stat,
        int atrib);
```

Grundlage für die Verwendung ist die Sortierung der Polygone(Dreiecke) in Gruppen. Im Rahmen von *CityPlane* ist dies die Einteilung des Höhenfeldes in Zeilen und Spalten, die on links nach rechts und von unten nach oben durchlaufen werden(vgl. Abbildung-34). Dabei gibt der erste Parameter Intervallgrenzen nach der Anzahl der Polygone pro Gruppe an und der zweite Parameter die Anzahl der Gruppen, nach denen separat trianguliert werden soll. Der 3., 4. und 5. Parameter sind für die Aufnahme der Information über Punkte und Indices der Polygone, danach folgen ein Zeiger auf das Ergebnis der Tristrip-Generierung und die Zahl der erzeugten Tristrips pro Gruppe. Der letzte Parameter „atrib“ setzt sie Attribute für die Einstellungen(vgl. Settings von *GenTriStrips*).

Die Schnittstellenklasse, die als „Front-End“ für *FTSG* fungiert, vereinfacht den Umgang mit den TriStripper auf folgende Weise:

- Direktes Einlesen der von *Triangle* erzeugten Datenfelder für die Punkte und die Indices der Dreiecke in Tripel-Form.
- Setzen des Umrisses des Höhenfeldes und der Anzahl der Gruppen in *x*- und *y*-Richtung, dadurch erfolgt eine klasseninterne Sortierung der Geometriedaten nach Gruppen in der oben angegebenen Ordnung.
- Schalter für einen Reset und das getrennte Setzen der Attribute.
- Möglichkeit zum Unterdrücken isolierter Dreiecke in der Ausgabe.

- Dynamische, auf STL-Vektoren beruhende Datenstruktur in 3 Dimensionen zur Aufnahme der gruppierten und indizierten Tristrips.

Beurteilung:

FTSG ist auch ohne Modifikationen gut in ein C++ -Project integrierbar und erfüllt praktisch alle Anforderungen, die man an einen Tristripper stellt. Bei den Testläufen im Rahmen von *CityPlane* sind keine nennenswerten Probleme bei der Anwendung aufgetreten. Im Gegensatz dazu weist der alternative Tristripper *STRIPE* deutliche Schwächen sowohl im Algorithmus (siehe Kapitel-5) als auch in der Ergonomie der Integration auf.

Spline:

Dieses ist als „Standalone" G-Programm geschrieben und bietet folgende Funktionalität(soweit für *CityPlane* relevant):

- Berechnung einer 1D-Splinefunktion unter Spannung durch Interpolation der spezifizierten Stützpunkte.
- Spezifikation von Tangenten in Anfangs- und Endpunkt.
- Unterteilung des Wertebereiches auf der *x*-Achse in gleichmäßige Intervalle zur Berechnung der gesuchten (*x*,*y*)-Werte.

Der Hauptgrund für die Verwendung einer Spline-Routine war die Möglichkeit zur automatischen Berechnung von weichen Übergängen der Fahrstreifenränder, wobei nur Anfangs- und Endpunkt(mit Tangenten), die Stärke der Ausbuchtung(Spannung) und die Genauigkeit der Approximation durch Angabe der gewünschten Ergebnispunkte angegeben werden brauchen. Während auf die theoretischen Aspekte im Kapitel-5 eingegangen wurde, soll hier nur noch kurz die Anpassung an *CityPlane* aufgezeigt werden. In erster Linie war die Umwandlung des Programm für die Verwendung als externe Prozedur durch Schaffung einer Schnitstelle notwendig, da *spline* in seiner Originalversion den Datentransfer über Standardein- und Standardausgabe vollzieht:

```
void GetSpline(double *xin, double *yin, unsigned int numpoints,
               double sl1, double sln, double tension,
               char *args, unsigned int numintervals, _3DPlist &points);
```

die Parameter umfassen die gesuchten Eingangs- und Ausgangsgrößen und das Ergebnis wird als Punktliste in Form eines STL-Vektors retouniert. Zu beachten ist allerdings, daß das Ergebnis nur in eindimensionaler Form vorliegt. Bei der Verwendung als „Kantengenerator" im Rahmen der Auswertung einer Straße durch die Klasse *Road* wurde folgende Vorgangsweise gewählt:

- Verwendung lokaler Koordinaten(vgl. Kapitel-5), wobei die Mittelachse des Segments als Abszisse aufgefaßt wird.
- Normierung der *x*-Werte.
- Die *y*-Werte entsprechen den Abständen des gesuchten Fahrstreifenrandes von der Mittelachse in den Stützpunkten.

- Als Tangenten werden die Tangenten des Straßensegments in Anfangs- und Endpunkt verwendet.
- Die gesuchte Anzahl an Splinepunkten wird von *spline* berechnet.
- Die Splinepunkte werden in das globale Koordinatensystem rücktransformiert und bilden den approximativen Rand des Fahrstreifens.

Beurteilung:

Obwohl ein breites Spektrum an frei verfügbaren Splineprogrammen existiert und auch andere Programme getestet wurden, fiel die Entscheidung zugunsten von *spline* aus. Hauptgrund dafür war neben der leichten Möglichkeit der Einbindung durch Plattformunabhängigkeit die Möglichkeit, den Kurvenlauf relativ einfach durch den globalen Spannungsparameter zu beeinflussen ohne den Nachteil des Überschwingens der *kubischen Splines* in Kauf nehmen zu müssen, was bei der Verwendung im Zusammenhang mit Straßen inakzeptabel wäre. Als Nachteil ist allerdings die Eindimensionalität zu nennen, allerdings ist der Berechnungsaufwand der Transformation der Punkte von einem lokalen- in ein globales Koordiantensystem geringer als für 2D-Splines wie etwa *B-Splines*, nicht zuletzt auch deshalb weil die Anzahl der Splinepunkte typischerweise gering ist.

6.3.4.) Weiterführende Möglichkeiten:

In diesem Abschnitt sollen abschließend einige bestehende und erweiterbare Möglichkeiten von *CityPlane* aufgelistet werden:

Aufteilung des IV-Straßenfiles:

Durch Setzen der *Box*-Größe in *x*- und *y*-Richtung kann das Ausgabefile der Straßen auf mehrere aufgeteilt werden. Dabei wird angegeben, wieviele Gruppen von Tristips jeweils in *x*- und *y*-Richtung zu einer sog. „Box“ zusammengefaßt werden. Dadurch wird für jede solche *Box* ein eigenes File erstellt und der Index als Postfix an den Filenamen des *iv*-Straßenfiles gehängt. Die Ordnung der Indices erfolgt gemäß der *CityPlane*-Konvention wieder von links nach rechts und von unten nach oben.

Eingabe in Form eines *OpenInventor*-Files:

Nachdem die erste Testversion auf ein *OpenInventor*-File, das Gebäude und Grundfläche einer Stadt repräsentiert, angewandt wurde, besteht die Möglichkeit durch die Präprozessor-Anweisung:

#define IV_INPUT

im Headerfile der Klasse *ElevGrid* und Rebuild, die Eingabe durch ein solches *iv*-File zu ermöglichen. Der Aufruf von *CityPlane* ist in diesem Fall:

CityPlane *IVEingabefile IVFläche Straßenfile IVStraßen*
Das *IVEingabefile* muß folgende (minimal-)Struktur aufweisen:

Separator {
Label { label "QuadMesh..." **}**

```
Coordinate3 { point [
    x y z,
     .
    .
    .
    x y z   ] }
}
}
.
.
.
Separator {
Label { label "QuadMesh..." }
Coordinate3 { point [
    x y z,
     .
    .
    .
    x y z   ] }
}
}
Separator {
  Label { label "DTM-QuadMesh..." }
  Coordinate3 { point [
   xmin ymin z,
    .
   .
   .
   xmax ymin z,
    xmin .        .
   .
   .
   .
   xmax ymax z
 ]}  }
}
```

Im oberen Teil des Files werden sequentiell die Gebäudegrundrisse durch Spezifikation der Punkte als geschlossener Polygonzug(im Gegenuhrzeiger-sinn) in je einem Separatorknoten angegeben und im letzten Separatoknoten die Rasterpunkte des Höhenfeldes, wobei die Ordnung wieder von links nach rechts(x-Werte) und von unten nach oben(y-Werte) ist.

Erweiterung der Verlaufsformen eines Straßensegments:

Bei der aktuellen Version von *CityPlane* werden nur gerade und kreisförmige Straßensegmente unterstützt, für die kreisförmigen Segmentbögen gilt als zusätzliche Einschränkung, daß der Öffnungswinkel nicht mehr als 90° beträgt. Für Stadtstaßen in bebauten Gebieten ist dies ausreichend. Die Struktur der von *RoadBasic* abgeleiten Klassen(*RoadStraight, RoadCircle*) ist so gehalten, daß eine Erweiterung durch folgende Mechanismen unterstützt wird:

- *Aufteilung der Parameter*: nachdem die Grundinformation für jeden Segmenttyp die gleiche ist(Querschnitte, Eckpunkte usw.), ist diese in der Basisklasse enthalten, die abgeleiteten Klassen enthalten lediglich für die jeweilige Kurve spezifische Werte(z.B.: Mittelpunkt und Radius bei Kreisbogen).
- *virtuelle Methoden*: alle (Kurven-)verlaufsspezifischen Operation (Transformationen, Berechnung von Kantenpunkten usw.) erfolgen über die virtuellen Methoden.

Die möglichen Erweiterungen beziehen sich vornehmlich auf Übergangsbögen wie *Parabel* und *Klothoide*, die theoretischen Grundlagen für die numerischen Berechnungschritte sind im Kapitel-3.3, Grundlagen des Straßenbaus, zu finden.

Anbindung der Ausgabe an andere Grafik-API's:

Obwohl *CityPlane* in der aktuellen Version auf *OpenInventor* aufbaut, kann die Ausgabe prinzipiell auch in einem Format erfolgen. Da alle *OpenInventor*-spezifischen Operationen(Szenegraph aufbauen, Baumstruktur in *iv*-File speichern usw.) in der Klasse *WriteToInventor* gekapselt sind, ist eine Umstellung dieser Klasse auf eine andere Szenegraph-Architektur konzeptuell möglich.

Umstellung des Straßenmodells von 2.5D auf 3D:

Grundlage des *City-Plane* Straßenfiles für die Eingabe ist die Beschreibung der Straße im Grundriß, lediglich für Randsteinkanten wird lokal eine relative Höhendifferenz festgelegt. Um komplexere Straßenkonstruktionen wie Über- und Unterführungen modellieren zu können reicht dieser Ansatz offensichtlich nicht aus. Eine Umstellung auf reines 3D erfordert also eine Erweiterung des Fileformats zur Straßenbeschreibung. Dabei muß also jeder Punkt entweder einen absoluten oder relativen Höhenwert erhalten. Zusätzlich dazu ist die Triangulierung so zu erweitern, daß auch Vertikalflächen trianguliert werden können. Bei allen polygonalen Flächen ist die räumliche Lage nicht mehr als etwa horizontal anzunehmen, sondern muß über die Richtung des Normalvektors identifiziert werden. Ein Musterprojekt wäre *EDF*(siehe Kapitel-2), bei welchem die Straßenachsen als Raumkurve realisiert werden.

7.) Resultate:

Im vorigen Kapitel wurde der modulare Aufbau und die grundlegenden Voraussetzungen für den Umgang mit *CityRoad* erläutert. Gegenstand dieses Kapitel ist es, die Theorie in die Praxis umzusetzen und anhand von repräsentativen Beispielfiles zusammen mit Bildern des Ergebnisses, typische Konfigurationen aus der Straßenentwurfspraxis zu illustrieren. Das Kapitel wird grob in 4 Abschnitte gegliedert:

- *Höhenfeldmodellierung*
- *Modellierung von Straßen*
- *Modellierung von Kreuzungen*
- *Beispielbilder aus dem Projekt URBANVIZ*

7.1.) Höhenfeldmodellierung:

Als Ausgangsbasis für ein ASC-File(vgl.:Kapitel-6.2, Höhenfeldrasterfile) werden im Fall der Verwendung für die Modellierung einer *realen Stadt* GIS-Daten benötigt. Handelt es sich um eine *fiktive Stadt* könnten die Höhenwerte der Rasterdaten z.B. durch Methoden der *fraktalen Geometrie* unter Anwendung von stochatischen Methoden generiert werden. Grundsätzlich gilt

natürlich, daß Städte auf relativ ebenem Gelände mit einer gröberen Rasterung auskommen. In jedem Fall muß der File-Kopf folgende Information enthalten, als Beispiel wird das Rasterfile(vienna_small_grid_20m.asc) für einen Teilauszug von Wien verwendet:

```
ncols         50
nrows         51
xllcorner     -50
yllcorner     -50
cellsize      20
NODATA_value  -9999
```

In diesem Beispiel wäre die Grundfläche also 1000m x 1020m mit dem rechten oberen Eckpunkt bei (950, 970) und einer Zellgröße von 20m.

Abbildung-37 zeigt das Ergebnis für das Beispielfile:

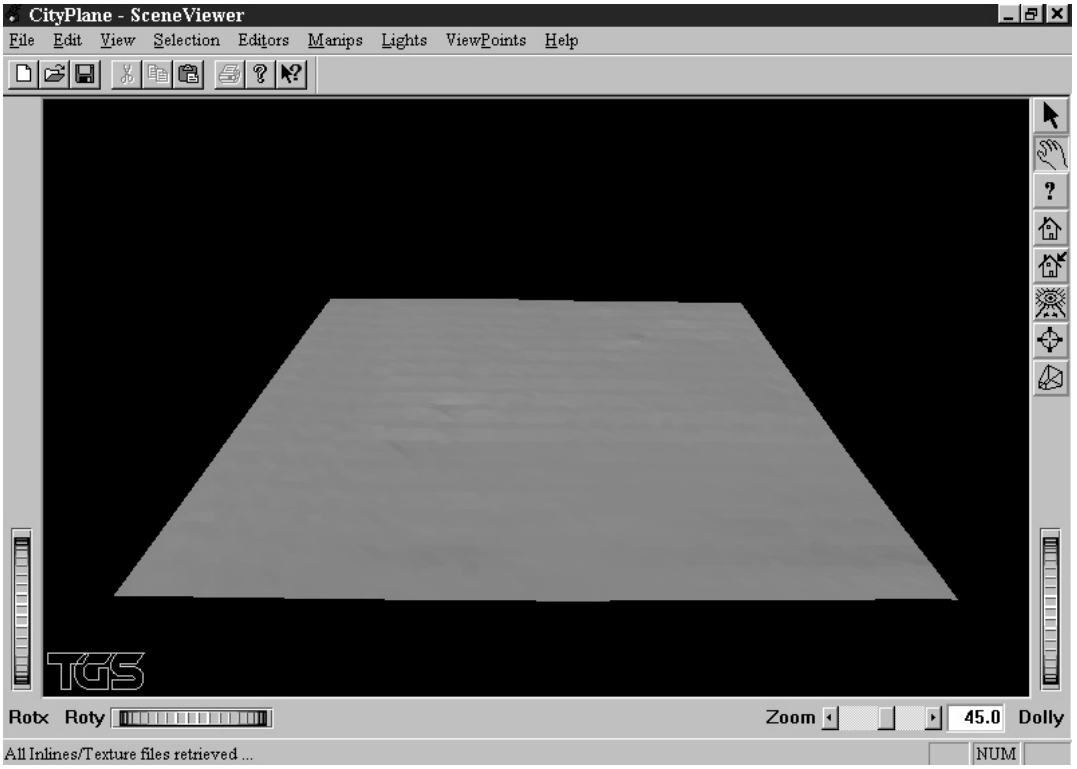


Abbildung 37: OpenInventor-Modell des Höhenfeldes für vienna_small_grid_20m.asc.

Man erkennt, daß das Höhenfeld aus dieser Perspektive recht flach aussieht, die Unterschiede an der Oberfläche werden allerdings bei Walkthroughs sichtbar.

Neben dem Eingabefile für die Höhenfelddaten ist die Information über die Lage von Grundrissen von Häusern, Straßen und anderen Objekten wichtig, da deren Grundfläche aus der Grundfläche ausgeschnitten werden. Entscheidend ist, daß die Polygonpunkte der Grundfläche im Gegenuhrzeigersinn angegeben werden. Ein Häusergrundriss aus dem File *blocks1000.txt*, der aus einem Orthophoto extrahiert wurde, hat 10 Kanten und ist durch 11 Punkte angegeben, d.h. das Polygon ist in sich geschlossen.

xyz
0, 288.339254, 324.259172, 182.647705
313.623186, 326.434134, 182.299728
338.907118, 328.609096, 182.144028
341.082080, 306.678230, 181.436630
343.257041, 284.747365, 181.382187
345.432003, 262.816499, 181.139816
319.876201, 262.000888, 181.730408
294.320399, 261.185278, 182.157822
292.326684, 282.209909, 182.169052
290.332969, 303.234541, 182.144699
288.339254, 324.259172, 182.647705
END

Insgesamt umfasst das Datenfile 85 Grundrisse, die aus der Grundfläche auszuschneiden sind. Die Punktmenge für die Triangulierung setzt sich demzufolge aus den Rasterpunkten plus den Eckpunkten dieser Polygone zusammen, wobei die Kanten der Grundrisspolygone dem in Kapitel 5, Triangulierung, definierten *Planar Straight Line Graph(PSLG),* entsprechen. Für jedes Grundrisspolygon ermittelt *CityPlane* einen Punkt innerhalb dieser Fläche, um von *Triangle* ausgeschnitten zu werden. Das Resultat zeigen die Abbildungen 38a und 38b, einmal als Fläche und einmal als gezoomtes Drahtmodell in die Struktur der Triangulierung:

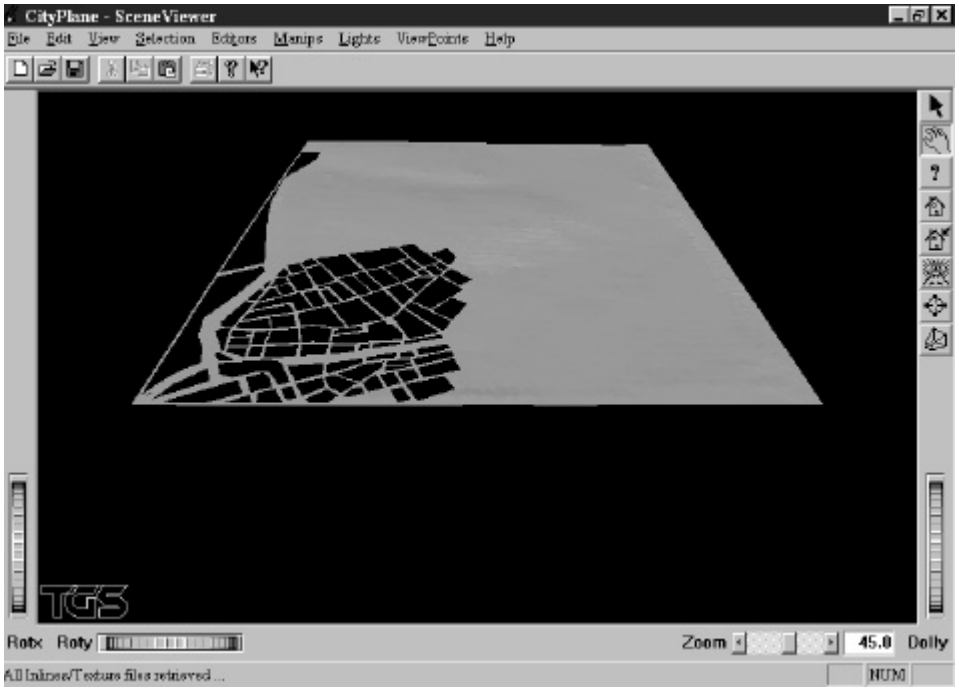


Abbildung 38a: Grundfläche eines Stadtauszugs von Wien mit teilweise ausgeschnittenen Häusergrundrissen.

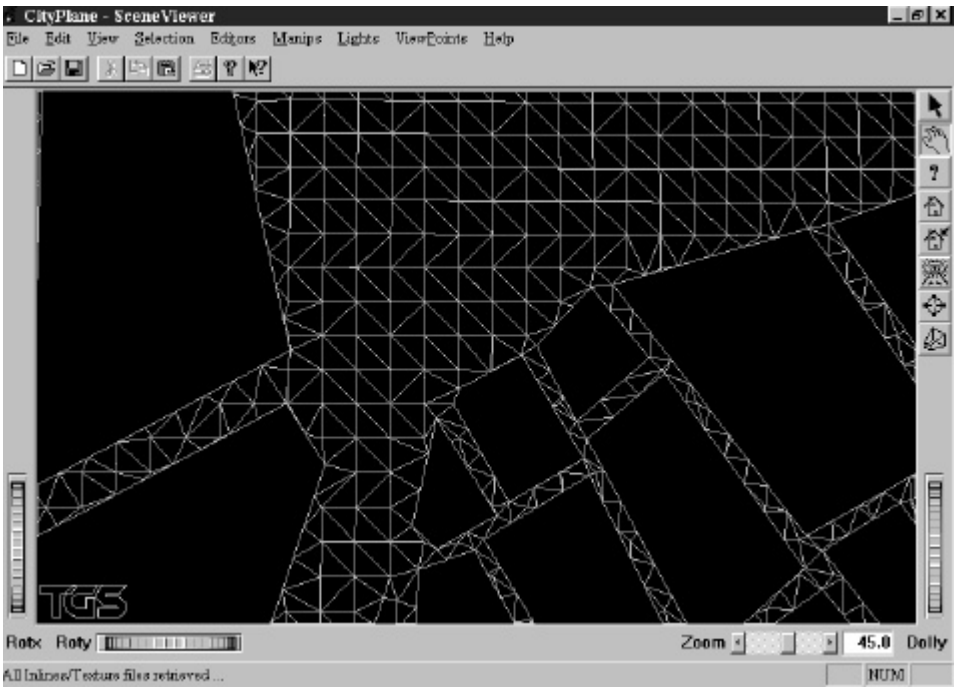


Abbildung 38b: Detailansicht der Abbildung-38a mit triangulierten Rasterpunkten und und den Grundrisse als Drahtmodell.

7.2.) Modellierung von Straßen:

Die beiden Beispiele in diesem Abschnitt sollen die im Straßenbereich am häufigsten vorkommenden Elemente umfassen und zeigen, welche Information das Straßenfile in konkreten Fällen enthalten muß, um z.B. Verkehrsinseln in der Fahrbahnmitte korrekt zu berechnen. Grundlage für diesen Abschnitt ist daher ein gutes Verständnis des in Kapitel 6.2 definierte Fileformat der Straßen.

Das erste Beispiel zeigt eine Variante von *Pattern Based Texturierung*, deren Grundgedanke im Wesentlichen darin besteht, die Textur an den Kanten benachbarter Polygone in Übereinstimmung zu bringen. Das File *multilanecurve.dat* hat folgendes Aussehen:

```
road { road_id 0
        straight
        { start_midpoints 0 0
        { pkw { mode c width 1 texture "blank_dash.jpg" size 2 align no }
          pkw     { mode    c width 0 texture "dash_dash.jpg"  size 2 align no }
          pkw     { mode    c width -1 texture "dash_blank.jpg"   size 2 align no }
          pkw     { mode    c width -2 }
        }
        }
        circle
        { start_midpoints 0 10
        { pkw { mode c width 1 texture "blank_dash.jpg" size 2 align no }
          pkw     { mode    c width 0 texture "dash_dash.jpg"  size 2 align no }
          pkw     { mode    c width -1 texture "dash_blank.jpg"   size 2 align no }
          pkw     { mode    c width -2 }
        }
        }
```

```
        straight
        { start_midpoints 10 20
        { pkw { mode c width 1 texture "blank_dash.jpg" size 2 align no }
          pkw     { mode    c width 0 texture "dash_dash.jpg"  size 2 align no }
          pkw     { mode    c width -1 texture "dash_blank.jpg"   size 2 align no }
          pkw     { mode    c width -2 }
        }
        }
        straight
        { start_midpoints 20 20
        { pkw { mode c width 1 texture "blank_dash.jpg" size 2 align no }
          pkw     { mode    c width 0 texture "dash_dash.jpg"  size 2 align no }
          pkw     { mode    c width -1 texture "dash_blank.jpg"   size 2 align no }
          pkw     { mode    c width -2 }
        }
        }
}
```

Man erkennt, daß der Straßenquerschnitt für alle Segmente gleich bleibt und aus 3 Segmenten(Gerade,Kreis,Gerade,Gerade), definiert durch 4 Querschnitte, besteht. Da es sich offensichtlich um 3 benachbarte Fahrspuren handelt, teilen sich je 2 Fahrspuren eine Mittellinie, dadurch daß der Parameter *align* für die Texturausrichtung auf „no“ gesetzt ist, werden alle Texturkoordinatenwerte auf die Mittellinie referenziert, das bedeutet, daß die Textur an den Kantenübergängen zusammenpaßt, da die Texturkoordinaten übereinstimmen. Abbildung-39a zeigt die Ausgabe von OpenInventor mit Textur und Abbildung-39b dasselbe als Drahtmodell, dadurch erkennt man die Approximation des Kreisbogens durch kurze Geradenstücke sowie die Übereinstimmung der Kantenpunkt an den Fahrstreifenübergängen:

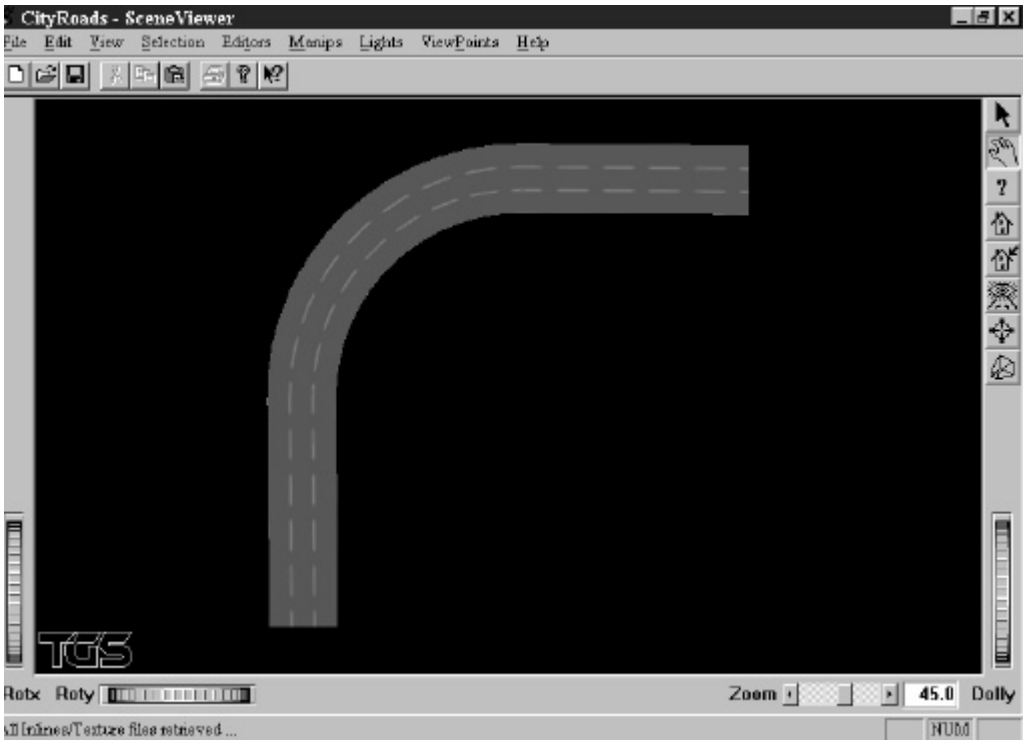


Abbildung 39a: 3-spurige Straße mit Texture-matching an den Fahrstreifenübergängen.

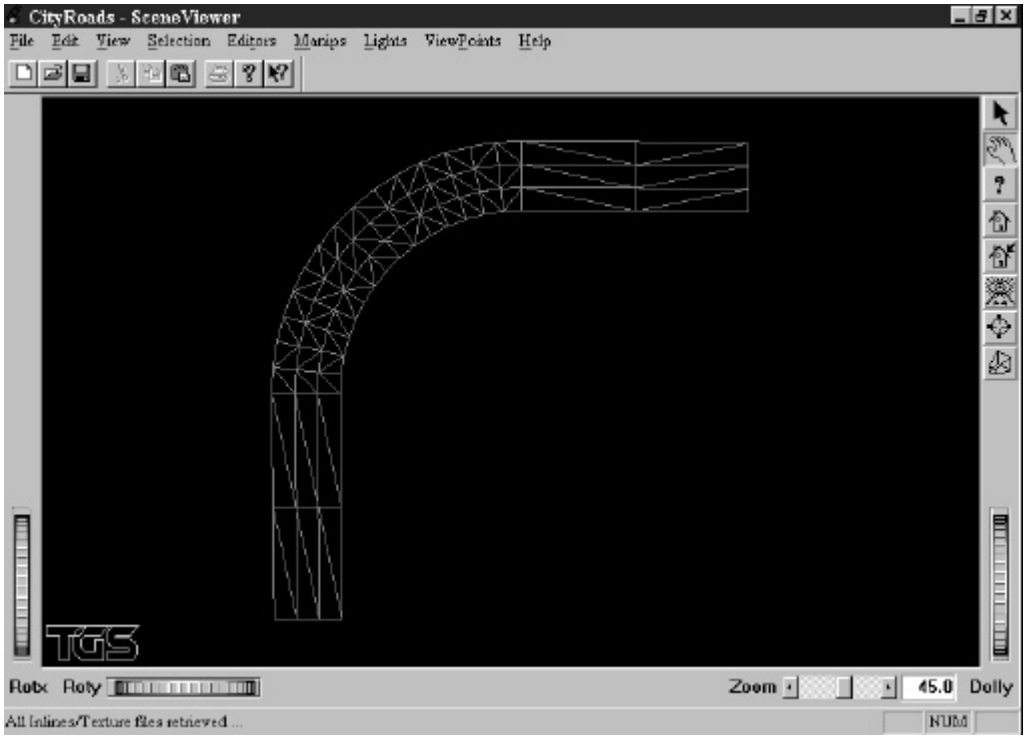


Abbildung 39b: Drahtmodell der 3-spurigen Fahrbahn mit approximativen Kantenpunkten des Kurvenstücks.

Das nächste Beispielfile *lanechangetest.dat* beihaltet 3 zentrale Aspekte:

- Texturwechsel entlang eines Fahrstreifens durch einen Fußgängerübergang mit Zebra-streifen.
- Wechsel der Fahrstreifenanordnung durch beiderseitige Parkstreifen längs der Fahrbahn.
- Fahrbahnteiler in Form einer Grüninsel in der Fahrbahnmitte.

Nachstehend sind auszugsweise einige Teile des Files abgedruckt, um die wesentlichen Aspekte hervorzuheben:

<u>Straßenquerschnitt:</u>

Der Aufbau des Querschnitts von links nach rechts ist: Gehsteig („sidewalk“), Randstein („sidewalk“), erste Fahrspur („pkw“), zweite Fahrspur („pkw“), Randstein („sidewalk“), Gehsteig („sidewalk“), rechter Gehsteigrand („sidewalk“). Jeder dieser Fahrstreifen erhält das Attribut „mode c“, weil alle Fahrstreifen bis zum nächsten Querschnitt fortgesetzt werden sollen. Zu beachten ist, daß jede Textur einen Minifikationswert „size“ erhält, um die Größe anzupassen. Bei den Fahrspuren soll die Textur ohne *Repeat* oder *Clip* an den linken und rechten Rand angepaßt werden und erhält dadurch einen „size“-Wert von 5, der der Breite der Fahrspuren entspricht:

```
road {      road_id 0
            straight
            {     start_midpoints 0 0
            {     sidewalk
            {     lane_id     0
                  mode        c
```

```
        width       9
        texture     "pave.jpg" size 2       align right
    }
        sidewalk
    {   lane_id     2
        mode        c
        width       6
        texture     "curbstone.jpg"
    }
        pkw
    {   lane_id     4
        mode        c
        width   5
        height      -1
        texture "blank_dash.jpg" size    5
    }
        pkw
        {   lane_id     5
            mode        c
            width   0
            texture "dash_blank.jpg" size    5
        }
        sidewalk
        {
            lane_id     1
            mode        c
            width   -5
            height      1
            texture     "curbstone.jpg"
        }
        sidewalk
        {
            lane_id     3
            mode        c
            width   -6
            texture     "pave.jpg" size 2       align left
        }
        sidewalk
        {
            lane_id     7
            mode        c
            width       -9
        }
    }
```

Texturwechsel eines Fahrstreifens:

Oft ist es notwendig, die Textur auf die Länge eines Fahrstreifens zu wechseln, wie bei diesem Beispiel, wo die Fahrbahn des 2. Segments durch einen Fußgängerübergang mit Zebrastreifen unterbrochen wird, nachdem die übrigen Teile(Gehsteige mit Randsteinen) gleich bleiben, werden diese hier nicht angeführt:

```
straight
{   start_midpoints 50 0
{   .
    .
    .
    pkw
    {
        lane_id     4
```

```
            mode        c
            width   5
            height      -1
            texture "zebrastripe.jpg"   size  1.5   align       left
        }
        pkw
        {
            lane_id     5
            mode        s
            width   0
        }
        .
        .
        .
}
}
```

Wie man sieht, wird der Teil für den Fußgängerübergang durch einen einzigen Teil repräsentiert, sodaß quasi der linke Fahrstreifen fortgeführt wird und die Textur des Zebrastreifen erhält, während der rechte Fahrstreifen durch das Attribut „mode s“ beendet wird.

Parkstreifen:

Das Einfügen eines Parkstreifens wird hier exemplarisch für die linke Straßenseite gezeigt, wobei der Parkstreifen mit dem nächsten Querschnitt beginnt und zwischen Randstein und erstem Fahrstreifen eigefügt werden soll, d.h. das Attribut „mode w“ und eine neue „lane_id“ erhält:

```
straight
{       start_midpoints 100 0
{       .
        .
        sidewalk
        {
            lane_id     2
            mode        c
            width       6
            texture     "curbstone.jpg"
        }
        pkw
        {
            lane_id     9
            mode        w
            width   5
            height      -1
            texture "blank_blank.jpg"
            size  5
        }
        pkw
        {
            lane_id     4
            mode        c
            width   5
            height      -1
            texture "blank_dash.jpg"
            size  5
        }
        .
        .
}
```

```
}
```

Zu beachten ist, daß der Punkt, von dem die Aufspaltung ausgeht, praktisch doppelt zu definieren ist, wobei zumindest die „width“ und „height“ Attribute übereinstimmen müssen. Dieses war die Aufspaltung nach links, d.h. für die korrelierenden Punktattribute wird das Attribut „mode w“ in der Querschnittsbeschreibung für den ersten der beiden gesetzt, während bei einer Aufspaltung nach rechts der umgekehrte Fall anzuwenden ist. Im Folgequerschnitt scheinen diese beiden Punkte, von denen die Verbreiterung ausgegangen ist, getrennt auf:

```
straight
{     start_midpoints 110 0
{     .
      .
      pkw
      {
            lane_id     9
            mode        c
            width   6
            height      -1
            texture "blank_blank.jpg"
            size  5
      }
      pkw
      {
            lane_id     4
            mode        c
            width   5
            texture "blank_dash.jpg"
            size  5
      }
      .
      .
}
}
```

Soll die zuletzt eingefügte Fahrspur wieder verschwinden, so werden die Breiten der Kanten wieder zu einen Punkt zusammengeführt, dabei werden wieder die beiden in „width“ und „height“ übereinstimmenden Punkte angegeben, wobei beim Fall, daß der linke Streifen verschwindet, das „mode s“ Attribut zuerst gesetzt wird und im gegenteiligen Fall umgekehrt:

```
straight
{     start_midpoints 160 0
{     .
      .
pkw
      {
            lane_id     9
            mode        s
            width   5
            height      -1
            size  5
      }
      pkw
      {
            lane_id     4
            mode        c
            width   5
            height      -1
            texture "blank_dash.jpg"
            size  5
```

```
        }
        .
        .
        }
        }
```

Verkehrsinseln:

Das Prinzip ist das Gleiche wie bei den Parkstreifen, allerdings erfolgt die Aufspaltung bzw. Zusammenziehung nach beiden Richtungen. Im Straßenbeispiel soll also nun zwischen den beiden Fahrspuren eine Grüninsel eingefügt werden und anschließend wieder aufhören:

```
straight
{     start_midpoints 170 0
{     .
      .
      pkw
      {
            lane_id     4
            mode        c
            width   5
            height      -1
            texture "blank_dash.jpg"
            size  5
      }
      pkw
      {
            lane_id     15
            mode        w
            width   0
            height      1
            texture "grass.jpg"
            size  5
      }
      pkw
      {
            lane_id     5
            mode        s
            width   0
      }
      pkw
      {
            lane_id     16
            mode        w
            width   0
            height      -1
            texture "dash_blank.jpg"
            size  5
      }
      .
.
  }
  }
```

Die Realisierung des Anfangs von diesem Fahrbahnteiler sieht also so aus, daß der erste Fahrstreifen normal weitergeführt wird, danach wird ein neuer Streifen mit „lane_id 15“ als Insel begonnen und der zweite Fahrstreifen beendet („mode s“) und mit einer neuen „lane_id 16“ begonnen.

Im weiteren Straßenverlauf scheint dann die Vekehrsinsel normal im Straßenquerschnitt mit den gewünschten Breitenwerten und dem „mode c“ Attribut auf. Das Ende des Inselteils wird folgendermaßen erreicht:

```
straight
{     start_midpoints 198 0
{     .
      .
      pkw
      {
            lane_id     4
            mode        c
            width   5
            height      -1
            texture "blank_dash.jpg"
            size  5
      }
      pkw
      {
            lane_id     15
            mode        s
            width   0
            height      1
      }
      pkw
      {
            lane_id     5
            mode        w
            width   0
            texture "dash_blank.jpg"
            size  5
      }
      pkw
      {
            lane_id     16
            mode        s
            width   0
            height      -1
      }

}
}
}
```

Man erkennt hier praktisch die „Umkehroperation“ zum Anfang, da der zweite Fahrstreifen die „lane_id 5“ „zurückerhält.

Abbildung-40 zeigt die OpenInventor-Ausgabe des Straßenbeispiels mit einem Verlauf von links nach rechts:

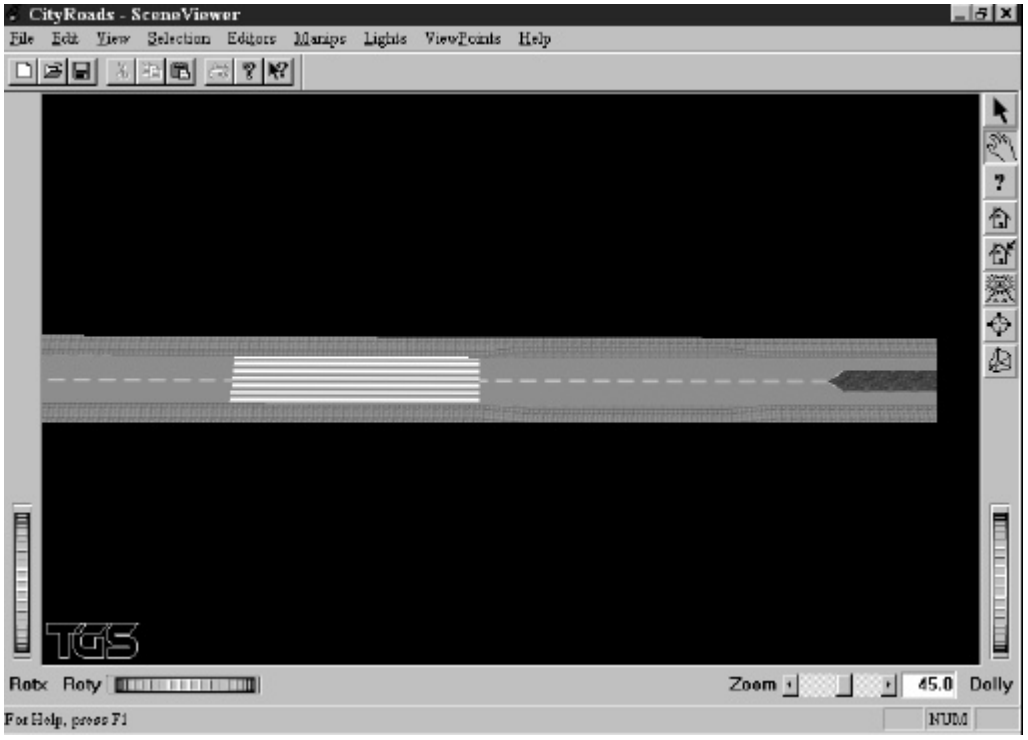


Abbildung 40: Gerade Straße mit Fußgängerübergang, beidseitigen Parkstreifen und einer Mittelinsel.

7.3.) Modellierung von Kreuzungen:

Nachdem alle Straßen berechnet sind, werden diese an den Knotenpunkten zusammengeführt. Als Grundlage dieses Abschnitts wird das Beispielfile *crosstest.dat* für eine Einmündung herangezogen, das folgende Aspekte zeigen soll:

- Einmündung oder *T*-Typ mit Polyline als Gehsteigrand.
- Benutzerdefinierte Abrundung des Randsteins durch Polylines.
- Verkehrsinsel in der Kreuzungsmitte.

Es gilt auch hier, daß nur für die Erklärung wesentliche Auszüge aus dem Beschreibungsfile abgedruckt sind und das Verständnis der Definition des Straßenfileformats vorausgesetzt wird. Die Grobform des Files hat folgenden Aufbau, der die Anbindung(„connections“) des Knotens an die Straßen über die *id* zeigt, wobei die Verbindungsart(z.B.:“tcrossing“) des Parameterblocks für die Abrundung angegeben wird. In diesem Beispiel werden 3 Straßen zusammengeführt, es ergeben sich also genausoviele Abrundungsteile:

```
road
{     road_id 0
      .
      .
}
```

```
road
{       road_id 1
        .
        .
}
road
{       road_id 2
        .
        .
}
crossing
{  crossing_id 1
   connections
        {       tcrossing
                {       from        0 start
                        to          1 end
                        polyline    2 [ 50 -9 ,
                                        80 -12 ,
                                        90 -9
                                      ]
                }
                corner
                {       from        1 end
                        to          2 start
                        radius      10
                        texture     "pave.jpg"
                        size        2
                        polyline    0 [ 85 5 ,
                                        76 9 ,
                                        75 20
                                      ]
                        polyline    1 [ 85 6 ,
                                        78 12 ,
                                        76 20
                                      ]
                        polyline    2 [ 85 15 ]
                }
                normal
                {       from 2 start
                        to     0 start
                        radius      0
                }
        }
        texture "blank_blank.jpg"
        size  2
        island
        {       points [ 65 0 185 , 75 0 185 , 70 4.5 185 ]
                dimensions 3
                height 1
                texture "grass.jpg"
                curbtexture "curbstone.jpg"
        }
        }
```

Einmündung, Polylinekanten:

Der Standardanwendungsfall für die Einmündung tritt auf, wenn eine Straße über die Kreuzung hinweg fortgesetzt wird. Der Parameterblock des Abrundungsteils könnte dabei wie folgt aussehen:

```
tcrossing
```

```
{     from      0 start
      to        1 end
      polyline  2 [ 50 -9 ,
                    80 -12 ,
                    90 -9
                  ]
}
```

Es werden als der Anfang der Straße mit *id 0* und das Ende der Straße mit *id 1* über einen Abrundungsteil an der Kreuzung verbunden. Das Beispiel zeigt auch den in der Praxis häufig auftretenden Fall, den äußeren Rand an andere Objekte(z.B. Häuser) durch eine Polyline anpassen zu müssen. Die Polylinepunkte sind grundsätzlich als von der ersten Straße zur zweiten Straße verlaufend aufzulisten. Der Wert „2“ hinter dem Schlüsselwort „polyline“ bezieht sich auf die (Adressierungs-)Kante des Abrundungsteils. Zum Verständnis dessen werden die Querschnitte der beiden Straßen gezeigt, wobei momentan irrelavante Information unterdrückt wurde:

```
road {      road_id 0
straight
{
      start_midpoints 50 0
{     sidewalk
      {     .
            .
      }
      sidewalk
      {     .
            .
      }
      pkw
      {     .
            .
      }
      sidewalk
      {     .
            .
      }
      sidewalk
      {     .
            .
      }
}
}
```

Da die (Kanten-)Adresse der Polyline 2 ist und 0 die innerste Kante des Abrundungsteils an der Randsteinkante ist, wird also der Außenrand durch die Polyline ersetzt. Der zweite Abrundungsteil ersetzt alle 3 Kanten durch Polylines, ist aber vom Typ „corner“, dadurch besteht dieser nur aus dem Randstein und einem Polygon, das nach außen hin den Abschluß des Gehsteigs bildet:

```
corner
{     from      1 end
      to        2 start
      radius    10
      texture   "pave.jpg"
      size      2
      polyline  0 [ 85 5 ,
                    76 9 ,
                    75 20
                  ]
```

```
        polyline   1 [ 85 6 ,
                     78 12 ,
                     76 20
                   ]
        polyline   2 [ 85 15 ]
    }
```

Verkehrsinseln:

Die in diesem Beispiel in die Kreuzung integrierte Mittelinsel zeigt die grundsätzliche Anwendung:

```
island
{   points [ 65 0 185 , 75 0 185 , 70 4.5 185 ]
    dimensions 3
    height 1
    texture "grass.jpg"
    curbtexture "curbstone.jpg"
}
```

Hier ist die polygonale Inselfläche durch 3 Punkte definiert, von denen alle 3 Dimensionen bekannt sind und durch „dimensions 3“ wird auch der *z*-Wert verwendet. Zusätzlich wird die Höhendifferenz zwischen Insel und Umgebung durch „height“ festgelegt.

Das Ergebnis in OpenInventor-Form für *crosstest.dat* zeigt Abbildung-41:

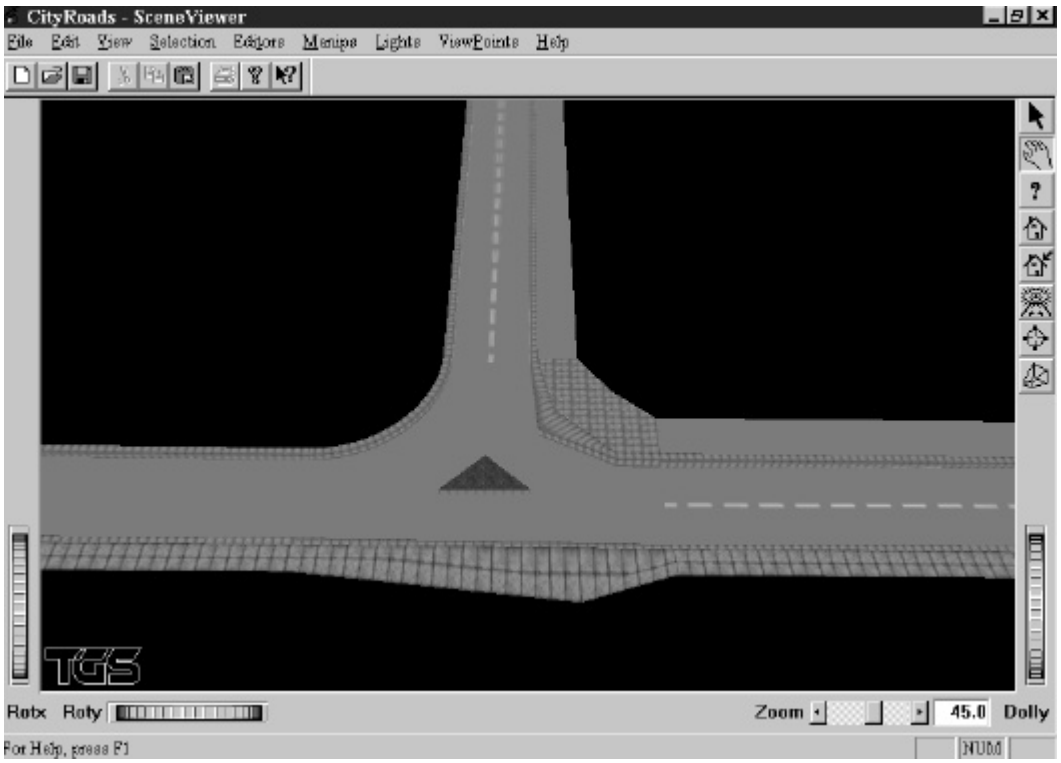


Abbildung 41: Einmündung mit Mittelinsel und verschiedenen Abrundungstypen.

7.4.) Beispielbilder aus dem Projekt *URBANVIZ*:

Der letzte Teil dieses Kapitels ist dem Projekt *URBANVIZ* gewidmet, in dess Rahmen die Entwicklung von *CityPlane* erfolgte. Dabei wurden die Ergebnisfiles aller fertig modellierten Objekte der Stadt(Grundfläche, Häuser, Straßen) zusammengefügt. Für diesen Schritt kann beispielsweise folgendes *OpenInventor-Skript* verwendet werden, das die Objekte von den Files einliest und in einem einzigen Szenegraph zusammenfügt:

```
Separator {
 File {
  name IVTerrainrasterfile
 }
 File {
  name IVStraßenfile
 }
   File {
  name IVHäuserfile
 }
      .
      .
      .
}
```

Die Abbildungen-42 bis 45 zeigen Ansichten eines Modells der Stadt Wien:

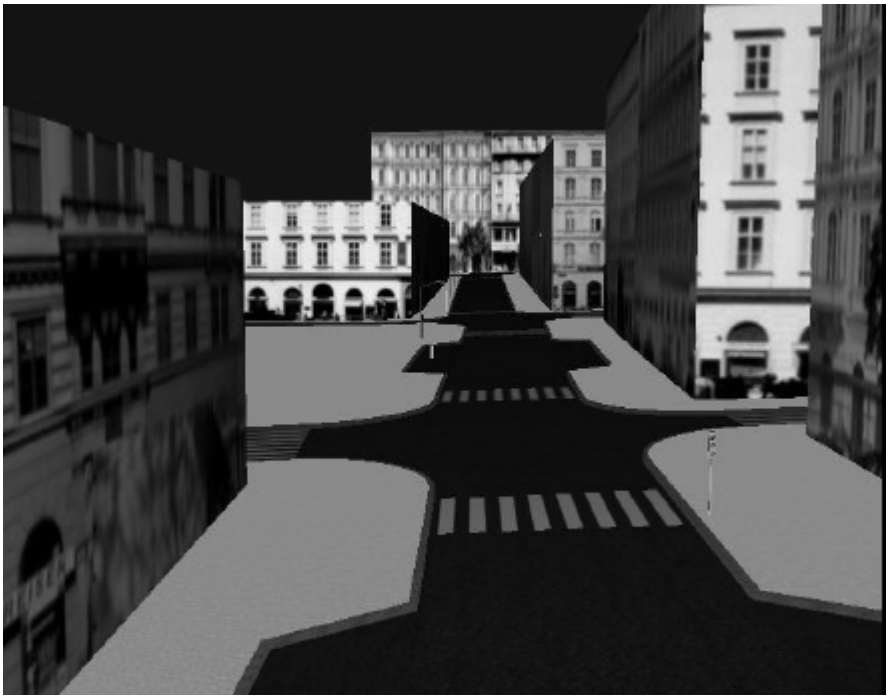


Abbildung 42: Kreuzungen mit und ohne Ampelregelung, Fußgängerübergängen und Parkbuchten.

Abbildung 43: Kreuzungen mit und ohne Ampelregelung, Fußgängerübergängen und Parkbuchten.

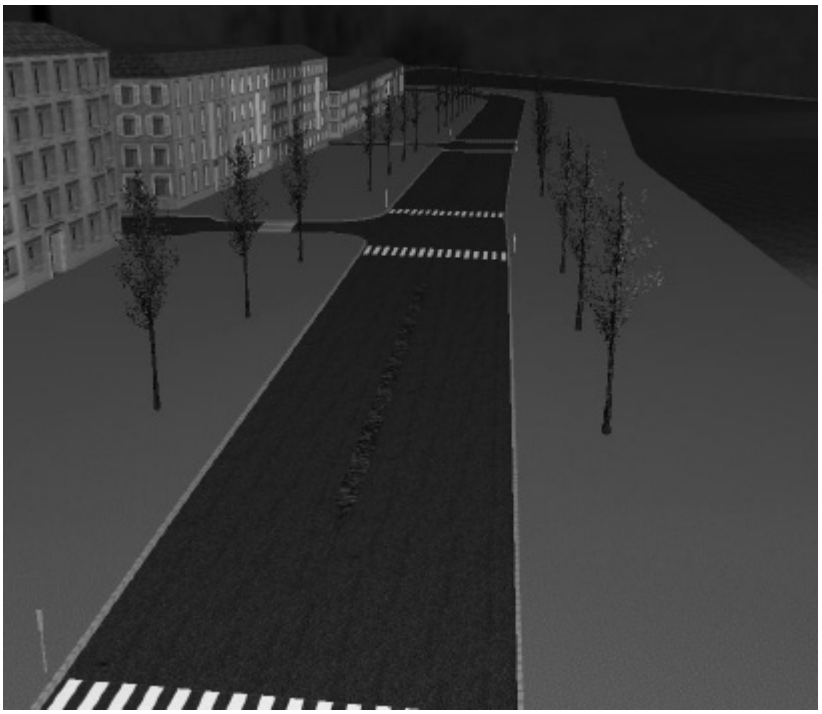


Abbildung 44: Straße am Stadtrand mit Allebäumen und Fußgängerübergängen.

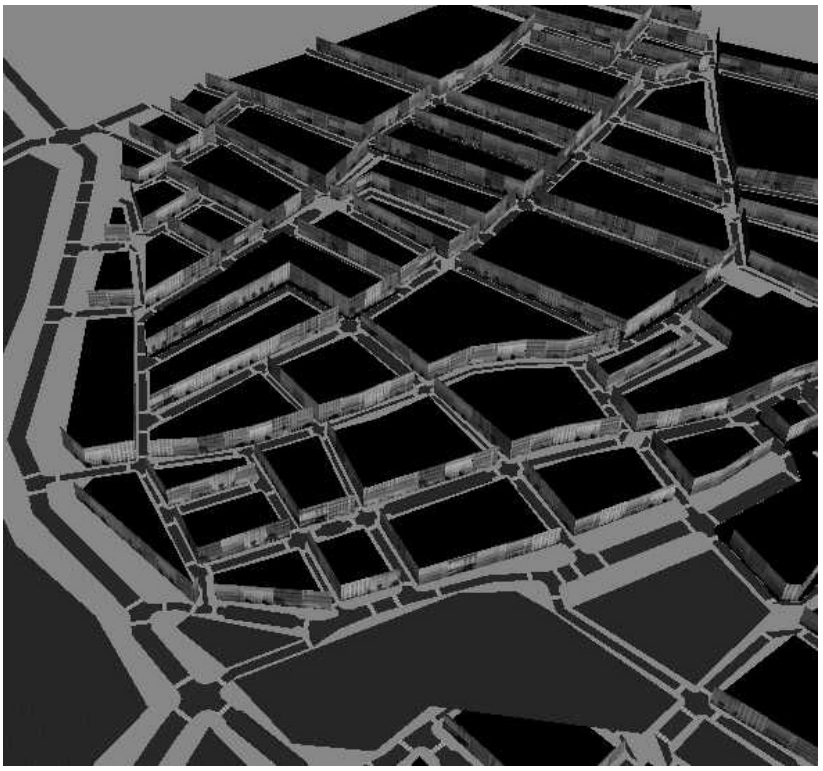


Abbildung 11: Straßennetz, Terrain und Häuserfronten aus der Vogelperspektive.

8.) Zusammenfassung und Schlussfolgerungen:

Die im Rahmen dieser Arbeit vorgestellten Möglichkeiten von *VRMG* decken viele Variationen von Straßen im Stadtbereich ab, dennoch ist noch genügend Raum für Erweiterungen und Verbesserungen geblieben, dessen Ausschöpfung den Zeitrahmen dieses Projektes sprengen würde. Zusammenfassend seinen noch die Möglichkeiten und Grenzen von *VRMG* angeführt:

8.1.) Möglichkeiten von *VRMG*:

Die *flexible Modellerstellung durch spezifische Parameter* ist für die meisten Aufgaben in den Zielbereichen der Anwendung ausreichend. Es war die vorherrschende Absicht, ausgehend von einem primär generierten Standardmodell, vom Groben ins Detail zu gehen, um eine Anpassung an lokale Gegebenheiten durch nachträgliche Verfeinerungen und Ergänzungen der Eingabedaten zu erreichen.

Eine *modulare, portierbare Implementierung* soll die Arbeit für nachfolgende Entwicklungen von *VRMG* erleichtern und durch eine solide Basis mögliche Erweiterungen, auf die im Text mehrfach eingegangen wurde, bilden. Diese Erweiterungen umfassen dabei sowohl die geometrische- und mathematische Seite etwa durch Umsetzung in neue C++ Klassen, als auch durch Ersetzung der Optimierungsverfahren und die Anbindung an zukünfige Grafik-API's.

8.2.) Grenzen von *VRMG*:

Die *Schwächen* sind primär *auf der Eingabeseite* der Daten zu finden und beziehen sich in erster Linie auf der Schwierigkeit, geeignete Eingabefiles zu Erstellen. Im Verleich zu bestehenden Simulatoren, wie sie in Kapitel-2 vorgestellt wurden, ist die Gewinnung von Daten, die zur Modellierung verwendet werden, relativ schwerfällig und nur durch geeignete Hilfsprogramme, die *VRMG* nicht zur Verfügung stellt, zu erreichen.

Die *Höhenfeldklasse* könnte durch die optionale Erweiterung in Richtung eines irregularen Rasters(*TIN-Network*) und die Verwendung einer *Spline*-Fläche mehr Realismus in der Ausgabe und eine bessere Anpassung an (lokale) Höhenfelddaten erzielen.

Es besteht eine gewisse *Desintegration von Verkehrsleit-Elementen*, da alle Zusatzelemente wie Verkehrszeichen und Ampeln nicht in *VRMG* enthalten sind und durch separate Komponenten hinzugefügt werden müssen.

Wie mehrfach in der Arbeit erwähnt wurde, existiert eine *Anzahl an Konstrukten aus dem Straßenbau*, die *nicht erfaßt* wurden. Diese Erweiterung betrifft etwa die Klothoide, deren Integration in das Modell wäre allerdings vor allem nur dann sinnvoll, wenn *VRMG* in der Straßenplanung eingesetzt werden soll.

8.3.) Danksagung:

Abschließend möchte ich meinen beiden Betreuern Dieter Schmalstieg und Peter Wonka für die Motivation, Unterstützung und Inspiration zu einem solch komplexen Unterfangen wie der Erstellung eines Modellierungs-werkzeuges für eine virtuelle Stadt danken. Diese Arbeit wäre auch ohne vorbildende Lehrveranstaltungen, die an der TU-Wien vom Institut für Computergrafik angeboten werden, nicht möglich gewesen. Es war für mich eine lehrreiche Erfahrung mit der Verkehrs- und Straßenplanung, verbunden mit der Erkenntnis, daß hier wesentlich mehr theoretische Konzepte zusammenwirken, als man auf den ersten Blick vermuten würde. Schließlich möchte ich auch allen danken, die ihre Software uneigennützig zu Forschungszwecken über das WWW anbieten und damit erst die Synthese von bewährten Implementierungen zur Lösungen von Teilproblemen zu einem *High-Level* Programmpaket ermöglichen.

Referenzen:

[1] William Jepson, Robin Liggett and Scott Friedman.
An Environment for Real-time Urban Simulation.
1995 Symposium on Interactive 3D-Graphics,
ACM SIGGRAPH, Seite 165--166, 1995.

[2] Robin Liggett, Scott Friedman and William Jepson.
Interactive design/decision making in a virtual urban
world: Visual simulation and GIS.

Fifteenth Annual ESRI User Conference Proceedings,
Palm Springs, California,
Environmental Systems Research Institute, Inc., 1995.

[3] Daniel Michel und David L. Brock.
A 3D Envirmonemt for an Observer Based Multiresolution Architekture.
1997 Spring Simulation Interoperability Workshop Orlando, FL, 1997.

[4] Wolfgang Pietzsch und Günter Wolf.
Straßenplanung.
6.Aufl., Werner Ingenieur Texte 37,
Copyright Werner Verlag Gmbh Düsseldorf, 2000.

[5] P.H.Niederhauser und U. Graf.
Elektronische Datenverarbeitung im Straßenbau.
Institut für Verkehrsplanung und Transporttechnik der Technischen Hochschule Zürich
Forschungsarbeit 15/79, 1982.

[6] Horst Osterloh.
Straßenplanung mit Klothoiden und Schleppkurven.
Bauverlag Gmbh, Wiesbaden und Berlin. 5.Aufl., 1991.

[7] Peter Jason Willemsen.
Behaviour and Scenario Modeling for Real-Time Virtual Environments.
degree in Computer Science in the Graduate College of The University of Iowa.
thesis for the Doctor of Philosophy, 2000.

[8] Salvador Bayarri, Marcos Fernandez und Mariano Perez.
Virtual Reality for Driving Simulation.
Communications of the ACM, Vol. 39, No. 5, 1996.

[9] Henning Natzschka.
Straßenbau - Entwurf und Bautechnik.
B.G. Teubner Stuttgart, 1996.

[10] V.S.Rao Sasipalli, G.S.Sasipalli and Koichi Harada.
Single Spirals in Highway Design And Bounds for their scaling.
IEICE Trans. Inf.&Syst., Vol. E80-D, No. II, 1997.

[11] Fabrice Neyret and Marie-Paule Cani.
Pattern-Based Texturing Revisited.
iMAGIS, CNRS, INRIA, Institute National Polytechnique de Grenoble
and Universite Joseph Fourier, Grenoble cedex 09, France, 1999.

[12] Gwenola Thomas and Stephane Donikian.
Modeling virtual cities dedicated to behavioural animation.
IRISA, Campus de Beaulieu, 35042 Rennes cedex, France, 1997.

[13] Stephane Donikian.
How to introduce live in Virtual Environments: a Urban Environment
Modeling System for Driving Simulation.

IRISA/CNRS, 35042 Rennes cedex, France, 1997.

[14] Xinyu Xiang, Martin Held and Joseph S.B.Mitchell.
Fast and Effective Stripification of Polygonal Surface Models.
Dep. of Applied Mathematics and Statistics,
State University of New York, Stony Brook, NY 11794-3600, 1999.

[15] Francine Evans, Steven Skiena and Amitabh Varshney.
Efficiently Generating Triangle Strips for Fast Rendering.
Department of Computer Science,
State University of New York at Stony Brook
Stony Brook, NY 11794-4400, 1996.

[16] Francine Evans, Steven Skiena and Amitabh Varshney.
Optimizing Triangle Strips for Fast Rendering.
State University of New York at Stony Brook:
IEEE Visualization 96, October 27 - November 1, 1996.

[17] Jonathan Richard Shewchuk.
Triangle: Engineering a 2D Quality Mesh Generator and
Delaunay Triangulator.
School of Computer Science,
Carnegie Mellon University
Pittsburgh, Pennsylvenia 15213,
1996.

[18] Peter Wonka, Michael Wimmer and Dieter Schmalstieg.
Visibility Preprocessing with Occluder Fusion for
Urban Walkthroughs.
Vienna University of Technology, 1999.

[19] Paul S. Heckbert and Michael Garland.
Survey of Polygonal Surface Simplification Algorithms.
Multiresolution Surface Modeling Course SIGGRAPH `97,
School of Computer Science,
Carnegie Mellon University
Pittsburgh, Pennsylvenia 15213,
1997.

[20] Dieter Schmalstieg.
A Survey of Advanced Interactive 3-D Graphics Techniques.
Vienna University of Technology, 1997.

[21] Multigen Creator.
Premier Tools for Realtime 3D-Database Development.
Produktbeschreibung von Multigen Paradigm,
http://www.multigen.com,
1999.

[22] Peter van Wolffelaar.
Functional Aspects of the Driving Simulator at the University of Groningen.

Centre for Environmental and Traffic Psychology,
Rijksuniversiteit Groningen, 1999.

[23] Peter van Wolffelaar, Salvador Bayarri, Inmaculada Coma.
Script-based definition of complex driving simulator scenarios.
Driving Simulator Conference DSC'99, Paris, 1999.

[24] Rahul Sukthankar, Dean Pomerleau, Charles Thorpe.
Simulated Highways for Intelligent Vehicle Algorithms.
Robotics Institute,
Carnegie Mellon University,
Pittsburgh, PA 15213-3891,
1995.

[25] Rahul Sukthankar, John Hancock, Chuck Thorpe.
Tactical-level Simulation for Intelligent Transportation Systems.
Robotics Institute,
Carnegie Mellon University,
Pittsburgh, PA 15213-3891,
1998.

[26] Joan D. Sulzberg, Michael J. Demetsky.
Demonstration of Traf-Netsim for Traffice Operations Management:
Final Report.
Virginia Department of Transportation, University of Virginia, 1990.

[27] Eric Bernauer, Laurent Breheret, Staffan Algers, Marco Boero, Carlo Di Taranto, Mark Dougherty, Ken Fox und Jean-François Gabard.
Review of Micro-Simulation Models.
Appendix D, SMARTEST Project Deliverable D3
Submission Date: 30 June 1997
Circulation Status: P - Public
The SMARTEST Project, Contract No: RO-97-SC.1059
Institute for Transport Studies, University of Leeds, 1997.

[28] L. Montero, E.Codina, J. Barceló, P. Barceló.
Combinig Macroscopic and Microscopic
Approaches for Transportation Planning and
Design of Road Networks.
Universitat Politècnica de Catalunya, Barcelona, 1996.

[29] J.Barceló, J.L.Ferrer, D. García and R. Grau.
Microscopic Traffic Simulation for ATT Systems Analysis
a Parallel Computing Version.
Universitat Politècnica de Catalunya, Barcelona, 1998.

[30] J. Barceló, J. Casas, J.L. Ferrer und D. García.
Modelling Advanced Transport Telematic Applications with Microscopic Simulators: The Case of AIMSUN2.
Universitat Politècnica de Catalunya, Barcelona,
TSS-Transport Simulation Systems, Barcelona, Spain, 1998.

[31] AutoCAD 2000 Detailed Feature Guide.
111 McInnis Parkway
Autodesk Inc.
San Rafael, CA 94903, USA, 1999.

[32] RoadViz 2000 Technical Description.
Bashir Research
http://www.my3d.com/roadviz.html,
2000.

[33] J.Hoschek, D.Lasser.
Grundlagen der Geometrischen Datenverarbeitung.
2.Auflage, Teubner Stuttgart, 1992.

[34] Kenneth W.Brodlie.
Mathematical methods in computer graphics and design.
Academic Press, 1980.

[35] Miljenko Marusic.
A fourth/secon or er accurate collocation method
for singularly perturbe two-point bounary value
problems using tension splines.
Department of Mathematics, University of Zagreb,
Bijenicka 30, 10000 Zagreb, Croatia;
Oktober 16, 2000 – Springer-Verlag, 2000.

[36] Jonathan Richard Shewchuk.
Lecture Notes on Delaunay Mesh Generation.
Department of Electrical Engineering and Computer Science,
University of California at Berkely,
Berkely, CA 94720,
1999.

[37] S. Fortune.
Voronoi diagrams and Delaunay triangulations,
in Euclidean Geometry and Computers.
D.A. Du, F. K. Hwang, ed., pp. 193-233,
World Scientific Publishing Co., 1992.

[38] The Inventor Mentor:
Programming Object Oriented 3D Graphics with Open Inventor.
Release 2, 1997.