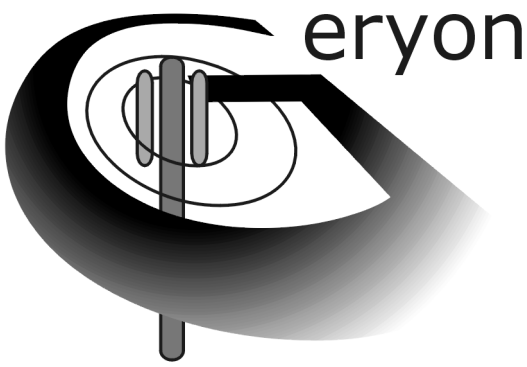

Geryon
Projektdokumentation

Stephan Brumme
Stephan Kirsch
Haik Lorenz

31. März 2003

Inhaltsverzeichnis

1. Zielstellungen

Ziel unseres Bachelorprojekts ist die Planung, das Design und die Implementierung eines größeren, softwareintensiven Systems. Dieses System, von uns Geryon getauft, soll auf einfache und klare Weise einem nicht-technischen Publikum die Auswirkung eines Mobilfunkstandortes bezüglich der Belastung durch elektromagnetischen Wellen verdeutlichen.

Entwickelt wird das System für und in Zusammenarbeit mit der Firma T-Mobile Deutschland. Bedient wird das Programm zwar von technisch geschulten Personen, aber nicht zwangsweise von einem Funknetzplaner.

Es soll eine einfache Möglichkeit bestehen, einen Mobilfunkstandort interaktiv in einem dreidimensionalen Gelände zu platzieren. Dieser Standort soll mit einer Rundstrahlantenne oder mit bis zu sechs Sektorantennen ausgestattet werden können. Neben der Antenne müssen Leistungen und ein Pauschalwert für die Verluste (Kabel, Stecker u.s.w.) angegeben werden.

Ausgehend von dieser Sendeanlage soll mit Hilfe des dreidimensionalen Antennendiagramms die Feldstärke in der näheren Umgebung (0 bis maximal 10 km) nach einem simplen Ausbreitungsmodell berechnet und plakativ dargestellt werden. Eine Navigation im Gelände und in der Stadt soll möglich sein, um an jedem Ort die dort auftretende Feldstärke sichtbar machen zu können.

Treten an einem Ort Feldstärken von mehreren Antennenanlagen auf, so sind diese leistungsmäßig zu addieren und die Summe dann darzustellen.

Die T-Mobile stellt sich vor, das Antennendiagramm als ein dreidimensionales transparentes Objekt, ähnlich einer deformierten Seifenblase, im Gelände oder in der Stadt darzustellen. Diese „Antennenblase“ wird von den anderen Objekten, Gelände, Gebäude, Wald etc. durchdrungen. An den Grenzflächen entstehen Feldstärken, die über eine bestimmte Farbe visualisiert werden sollen. Treten im Laufe der Realisierung andere ggf. bessere Ideen auf, die der Zielsetzung eher entsprechen, können diese nach Absprache zusätzlich oder alternativ umgesetzt werden.

Die Farbzuordnung der berechneten Feldstärke muss interaktiv vom Nutzer geändert werden können. Alle Daten und Einstellungen müssen gespeichert werden können, so dass sie beim nächsten Start des Programms wieder vorhanden sind.

Bestimmte funktechnische Eigenschaften müssen durch den Nutzer festgelegt und vom Programm bei der Berechnung berücksichtigt werden. Eigenschaften einer Sendeanlage (Standort) sind:

- Standort (Länge, Breite, Höhe über dem Gelände)
- 1 bis 6 Antennenanlagen
- Leistung in Watt
- Verluste in dB
- Frequenzbereich (900 MHz, 1800 MHz oder 2000 MHz)

Eigenschaften einer Antennenanlage sind:

- Koordinaten der Antenne (Länge, Breite, Höhe über dem Gelände)
- Bezeichnung der Antenne
- Ausrichtung (Azimuth) in Grad
- Mechanische Absenkung in Grad (positive Werte entsprechen einer Absenkung aus der Horizontalen nach unten)
- Elektrische Absenkung in Grad

Zu jedem Antennendiagramm gehört ein Typ, zu jedem Antennentyp gibt es ein dreidimensionales Diagramm (Abstrahlcharakteristik). Das Diagramm wird durch jeweils 360 Werte, die einen horizontalen und einen vertikalen Schnitt durch das Diagramm bilden, beschrieben. Eine genaue Beschreibung des Formats stellt die T-Mobile zur Verfügung[14].

Dem gesamten Programm zuzuordnende Eigenschaften sind:

- Parameter des Ausbreitungsmodells
- Maximale Entfernung der Berechnung
- Farbauswahl
- Spracheinstellung

Die Justierung von Parametern der Sendeanlage soll interaktiv möglich sein. Denkbar ist es, die Ergebnisse schrittweise mit zunehmender Entfernung zu visualisieren. Die Navigation im dreidimensionalen Gelände bzw. im Stadtmodell muss in „Echtzeit“ passieren, d.h. es müssen ausreichend Bilder pro Sekunde generiert werden, um eine flüssige Darstellung zu ermöglichen.

Der Nutzer soll sich auf die Aufgabe der Präsentation konzentrieren. Aus diesem Grund sind möglichst wenige, klar erkennbare Bedienelemente vorzusehen. Die Benutzerführung soll denkbar einfach und intuitiv sein. Die Sprache der Benutzeroberfläche soll konfigurierbar (z.B. Umschaltung eines Message-Katalogs) sein. Das Produkt sollte jedoch mit einer deutschsprachigen Oberfläche ausgeliefert werden.

2. Theoretische Grundlagen

2.1 Koordinatensysteme in Geryon

Zur Bestimmung eines Punktes auf der Erdoberfläche nutzt man meist ein zweidimensionales Koordinatensystem, das auf einer Einteilung in Längen- und Breitengrade beruht. Dieses findet seit Jahrhunderten Verwendung in der Vermessung, Schifffahrt etc. und ist daher weltweit standardisiert.

Die dreidimensionale Visualisierung eines Stadtmodells und des dazugehörigen Terrains erfordert spezielle Randbedingungen bezüglich des Koordinatensystems. Ein Problem besteht darin, dass der Abstand zwischen zwei Längengraden vom momentanen Breitengrad abhängt. Bei den uns vorliegenden Daten aus dem Frankfurter Stadtmodell ist dieser Abstand nur noch in etwa halb so groß wie am Äquator.

Ebenso stellen Consumer-Level-Grafikkarten ein nicht unerhebliches Problem dar: Sie besitzen eine zu geringe Genauigkeit bei der Verwendung von Gleitkommazahlen. Wir sind für eine exakte Darstellung auf dem Bildschirm darauf angewiesen, auch noch im Meterbereich eine korrekte Visualisierung zu erzielen.

Um diese Schwierigkeiten bewältigen zu können, entschieden wir uns für den Einsatz verschiedener Koordinatensysteme. Sie beruhen auf den vom Kunden bereitgestellten Daten (Stadtmodell von T-Mobile), der verwendeten Geovisualisierungs-Bibliothek (LandExplorer) und den umgesetzten Algorithmen (Wellenausbreitung in Stadtmodellen).

Alle benutzten Koordinatensysteme werden im Folgenden vorgestellt. Der Endanwender kommt nur mit dem zuerst vorgestellten geographischen Koordinatensystem in Berührung. Die anderen beiden sind technisch bedingt, sie werden nur intern verwendet und sind daher lediglich für Entwickler interessant. Alle Umrechnungen zwischen den Koordinatensystemen sind in der Klasse CoodinateTransformer zu finden.

2.1.1 Geographisches Koordinatensystem

Meridiane sind vom Nordpol zum Südpol verlaufende Halbkreise, in deren Mittelpunkt sich der Erdmittelpunkt befindet. Der Nullmeridian geht durch die Sternwarte Greenwich bei

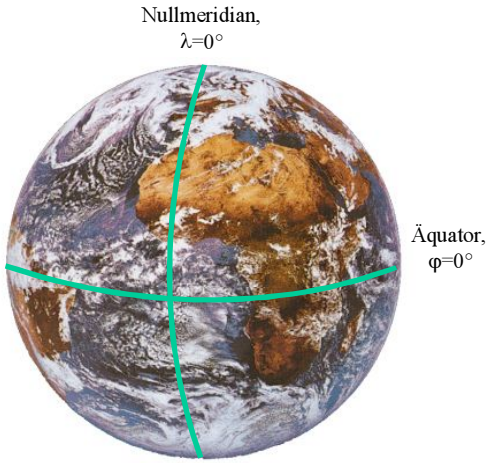

Abb. 1: Geographisches Koordinatensystem

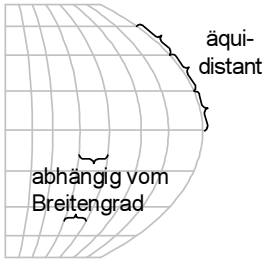

Abb. 2: Ausmaße eines Grades

London. Zu jedem Meridian wird neben dem Winkel, den er mit dem Nullmeridian einschließt, noch die Lage bezüglich Greenwich in östlich und westlich unterteilt.

Als Winkeleinheit benutzt man Grad, wobei sich die Längengrade λ im Bereich von 180° westlicher Länge über den Nullmeridian bis 180° östlicher Länge erstrecken können (allerdings fallen diese beiden Grenzen wieder zu einem gemeinsamen Meridian zusammen).

Sogenannte Breitenparallelen sind zum Äquator parallel verlaufende Kreise, deren Mittelpunkt auf der Erdachse liegt. Nur am Äquator fällt dieser Punkt auch mit dem Erdmittelpunkt zusammen.

Die Breitengrade φ liegen zwischen 90° südlicher Breite (Südpol) und 90° nördlicher Breite (Nordpol). Der Breitengrad 0° ist der Äquator. Abb. 1 veranschaulicht den Sachverhalt.

Zur Angabe verwendet man Grad, wobei 360° einem Vollkreis entsprechen. Ein Grad besteht aus 60 Winkelminuten. Eine Winkelminute kann in 60 Winkelsekunden unterteilt werden. Eine Winkelsekunde wird, je nach Anwendungsgebiet, manchmal noch in 1000 Winkelmillisekunden verfeinert. Die Stadt Frankfurt/Main befindet sich beispielsweise bei 50°07' nördlicher Breite und 8° 40' östlicher Länge.

Das uns vorliegende Stadtmodell umgeht die technisch aufwändige Unterteilung in Grad, Winkelminuten, Winkelsekunden durch eine Abbildung auf Winkelmillisekunden:

$$geo = (((Grad \cdot 60) + Minuten) \cdot 60 + Sekunden) \cdot 1000 + Millisekunden \qquad (1)$$

Ferner stellt man westliche Länge durch östliche Länge dar:

$$Ost = -West \qquad (2)$$

Die Anzahl unterschiedlicher darzustellender Werte liegt somit bei $6{,}48 \cdot 10^8$, was problemlos mit einer 32-Bit-Ganzzahl umsetzbar ist. Eventuelle Höhenangaben über Normalnull erfolgen in Zentimetern ebenfalls als Ganzzahl. Eine typische Koordinate in Winkelmillisekunden plus Höhe, entnommen aus dem Frankfurter Stadtmodell, hat die Gestalt: (31.041.698; 180.387.715; 9.560) und entspricht 8° 37' 21,698" östlicher Länge bei 50° 6' 27,715" nördlicher Breite in einer Höhe von 95,6 m über Normalnull.

2.1.2 Geo-Koordinatensystem zur Visualisierung

Ein nichttriviales Problem stellen die uns zur Verfügung stehenden Grafikkarten dar. Sie nutzen nur die Genauigkeit eines 32-bit-Gleitkomma-Datentyps aus. Dieser kann zwar sogar einen größeren Zahlenbereich als 32-bit-Ganzzahlen überdecken, bietet aber nicht deren Präzision. Als Faustregel gilt, dass nur etwa 7 Stellen zuverlässig sind, in unserer Anwendung sind aber bis zu 9 Stellen essentiell. Die Abweichung von bis zu 100 Winkelmillisekunden können ca. drei Meter ausmachen. Vor allem bei dynamischen Szenen fallen Artefakte, wie sich sprunghaft und unregelmäßig bewegende Objekte, auf.

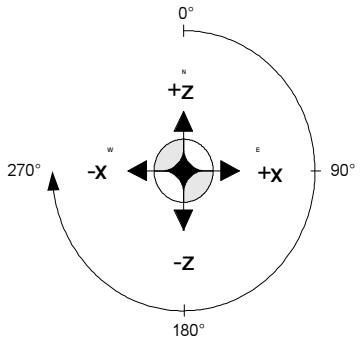

Abb. 3: Orientierung

Die Praxis zeigt, dass Stadtmodelle nur einen sehr kleinen Teil der Erdoberfläche überdecken. Demzufolge liegen die Längen- und Breitengrade aller Objekte in einem relativ schmalen Intervall. Dieses Intervall bringen wir durch eine Translation in die Nähe des Koordinatenursprungs.

$$vis = geo - Mittelpunkt \tag{3}$$

Somit werden die Werte in Winkelmillisekunden deutlich kleiner, sie liefern dann auch mit 7 Dezimalstellen eine ausreichende Genauigkeit.

2.1.3 Meter-Koordinatensystem

Das Walfish-Ikegami-Modell zur Berechnung der Funkwellenausbreitung verlangt seine Parameter in Metern. Dazu ist es erforderlich, dass wir den Längen- und Breitengrad eines Punktes jeweils als Abstand in Metern zu einem Bezugspunkt umrechnen (siehe Kapitel 4.1.7). Nebenbei berücksichtigt dieses Koordinatensystem auch die Erdkrümmung (siehe Abb. 4). Die Umsetzung mit Hilfe eines dreidimensionalen Vektors folgt der Form:

$$m = \begin{pmatrix} m_{Ost} \\ m_{Höhe} \\ m_{Nord} \end{pmatrix} \tag{4}$$

Die Komponenten des Vektors beziehen sich dabei jeweils auf den Bezugspunkt. Die x-Achse zeigt von ihm ausgehend nach Osten, die z-Achse nach Norden und die y-Achse nach oben, also vom Erdmittelpunkt weg.

Um von den Koordinaten eines Punktes im Gradnetz zu den beschriebenen Koordinaten zu kommen, sind zwei Schritte notwendig. Zuerst werden die Meterkoordinaten des Punktes mit dem Erdmittelpunkt als Bezugspunkt bestimmt. Danach wird eine Transformationsmatrix auf die ermittelten Koordinaten angewandt. Sie verschiebt den tatsächlichen Bezugspunkt in den Koordinatenursprung und rotiert zusätzlich entsprechend, so dass die Koordinatenachsen den oben beschriebenen Richtungen folgen.

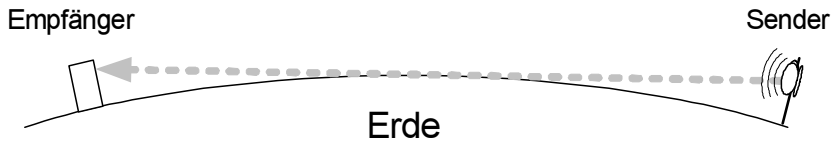

Abb. 4: Erdkrümmung

Die beschriebene Transformation lässt sich mit Hilfe einer orthonormalen Transformation multipliziert mit einer Translationsmatrix erreichen. Um zu den notwendigen drei orthonormalen Vektoren zu kommen, wird im Millisekundenkoordinatensystem jeweils ein kleines ε in der entsprechenden Richtung zum Bezugspunkt addiert. Danach wird dieser Vektor in das Meterkoordinatensystem (mit dem Erdmittelpunkt als Bezugspunkt) umgerechnet und der ebenso umgerechnete Bezugspunkt davon abgezogen.

$$zuMeter\begin{pmatrix} geo_{Längengrad} \\ geo_{Höhe} \\ geo_{Breitengrad} \end{pmatrix} = \begin{pmatrix} \sin(geo_{Längengrad}) \cdot \cos(geo_{Breitengrad}) \cdot (erdradius + geo_{Höhe}) \\ \sin(geo_{Breitengrad}) \cdot (erdradius + geo_{Höhe}) \\ -\cos(geo_{Längengrad}) \cdot \cos(geo_{Breitengrad}) \cdot (erdradius + geo_{Höhe}) \end{pmatrix} \quad (5)$$

$$ost = zuMeter\begin{pmatrix} geo_{Längengrad} + \varepsilon \\ geo_{Höhe} \\ geo_{Breitengrad} \end{pmatrix} - zuMeter(geo) \quad (6)$$

$$oben = zuMeter\begin{pmatrix} geo_{Längengrad} \\ geo_{Höhe} + \varepsilon \\ geo_{Breitengrad} \end{pmatrix} - zuMeter(geo) \quad (7)$$

$$nord = zuMeter\begin{pmatrix} geo_{Längengrad} \\ geo_{Höhe} + \varepsilon \\ geo_{Breitengrad} \end{pmatrix} - zuMeter(geo) \quad (8)$$

So ergeben sich drei Vektoren, die (nahezu) senkrecht aufeinander stehen. Sie zeigen zudem, ausgehend vom Bezugspunkt, nach Norden, nach Osten und nach oben. Diese werden normalisiert und ergeben die orthonormale Rotationsmatrix. Der umgerechnete Bezugspunkt ergibt die Translationsmatrix.

$$bezP = zuMeter(geo) \quad (9)$$

$$ostN = \frac{ost}{|ost|} \quad (10)$$

$$obenN = \frac{oben}{|oben|} \quad (11)$$

$$nordN = \frac{nord}{|nord|} \quad (12)$$

$$transform = \begin{pmatrix} ostN_x & ostN_y & ostN_z & 0 \\ obenN_x & obenN_y & obenN_z & 0 \\ nordN_x & nordN_y & nordN_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 & -bezP_x \\ 0 & 1 & 0 & -bezP_y \\ 0 & 0 & 1 & -bezP_z \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (13)$$

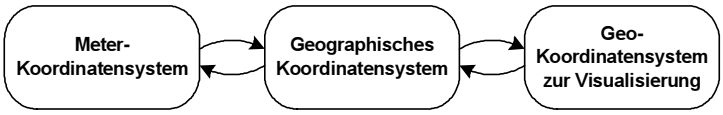

Abb. 5: Konvertierungsmöglichkeiten

Insgesamt ergibt sich die folgende Transformation vom Millisekunden in das Meterkoordinatensystem:

$$meter = transform \cdot zuMeter(geo) \tag{14}$$

2.2 Wellenphysik im Mobilfunkbereich

Für unser Bachelorprojekt Geryon sind grundlegende Kenntnisse über die Ausbreitung von Funkwellen erforderlich. Hierbei spielt insbesondere der Frequenzbereich von 900 MHz bis 2000 MHz eine zentrale Rolle, da in ihm GSM900, GSM1800 und UMTS operieren.

2.2.1 Dämpfungsfaktor

Der Dämpfungsfaktor ist eine dimensionslose Maßeinheit für die Dämpfung bezüglich einer Vergleichsgröße:

$$D = \frac{P_1}{P_2} \tag{15}$$

wobei D die Dämpfung zwischen einer gesendeten Leistung P_1 und einer empfangenen Leistung P_2 darstellt[9].

2.2.2 Dezibel

Das Dezibel, kurz dB, ist dem Dämpfungsfaktor nah verwandt. Es wird unter Verwendung des dekadischen Logarithmus ausgedrückt[10]. Allgemein gilt:

$$\begin{aligned} a &= 10 \cdot \log_{10} \frac{P_1}{P_2} \\ &= 10 \cdot \log_{10} D \end{aligned} \tag{16}$$

Eine Dämpfung von 50% entspricht damit etwa 3,3 dB[8]:

$$\begin{aligned} &P_1 = 2, \quad P_2 = 1 \\ &a = 10 \cdot \log_{10} 2 \\ &\approx 3{,}3 \text{ dB} \end{aligned} \tag{17}$$

Die Umrechnung von Dezibel in einen Dämpfungsfaktor[11]:

$$D = 10^{\frac{a}{10}} \tag{18}$$

Der Vorteil des Dezibel gegenüber dem Dämpfungsfaktor liegt darin, dass man die Gesamtwirkung von mehreren Einzeldämpfungen durch einfache Summation erhält:

$$a = \sum_i a_i \tag{19}$$

Eine Verstärkung entspricht einer negativen Dämpfung.

2.2.3 Berechnung der Dämpfungen

In der Praxis werden die Funkwellen durch verschiedenste Einflüsse gedämpft. In Geryon nutzen wir zur Simulation der Gebäudeeinflüsse in einer Stadt das Walfish-Ikegami-Modell (vgl. Kapitel 2.3). Dieses liefert uns eine Dämpfung in Dezibel zurück. Bezüglich der Dämpfungsauswirkungen von Gebäuden finden sich in einigen Quellen Messwerte für verschiedene Materialien[11],[12]. Finnische Messungen sprechen von einer durchschnittlichen Dämpfung von 18 dB innerhalb von Häusern[11].

Die Freiraumdämpfung $a_{Freiraum}$ in dB hängt nur von der Trägerfrequenz f (in MHz) und der Entfernung d (in m) ab[11], [12]:

$$\begin{aligned} a_{Freiraum} &= 10 \cdot \log\left(\frac{4\pi \cdot d}{\lambda}\right)^2 \\ \lambda &= \frac{0{,}3}{f} \\ a_{Freiraum} &= 32{,}44 + 20 \cdot \log f + 20 \cdot \log d \end{aligned} \tag{20}$$

Diese Gleichung setzen wir für die Darstellung des dreidimensionalen Antennendiagramms um.

2.2.4 Die elektrische Feldstärke

Ohne Dämpfungseinflüsse

Die elektrische Feldstärke unter der Annahme einer ungehinderten Ausbreitung in Hauptstrahlrichtung bestimmt sich aus den in Tabelle 1 aufgeführten Parametern.

Der Antennengewinnfaktor G entspricht einem negativen Dämpfungsfaktor der Antenne. Somit gilt für die elektrische Feldstärke:

Parameter	Einheit
Elektrische Feldstärke	E in V/m
Antennengewinnfaktor	G
Antennengewinn	g in Dezibel
Sendeleistung	P_1 in W
Empfangsleistung	P_2 in W
Entfernung Sender/Empfänger	d in m

Tabelle 1: Einflussfaktoren der elektrischen Feldstärke

$$E = \frac{\sqrt{30 \cdot G \cdot P}}{d} \tag{21}$$

Ausgedrückt mit Hilfe von Dezibel:

$$G = 10^{\frac{g}{10}}$$
$$E = \frac{\sqrt{30 \cdot 10^{\frac{g}{10}} \cdot P}}{d} \tag{22}$$

Mit Dämpfungseinflüssen

Für die Gesamtdämpfung ist zu beachten, dass der Antennengewinn eine negative Dämpfung darstellt. Somit folgt aus (15) und (18):

$$E = \frac{\sqrt{30 \cdot 10^{\frac{g-a}{10}} \cdot P}}{d} \tag{23}$$

Einfluss mehrerer Antennen

Wird ein Gebiet von mehreren Antennen überdeckt, so sind die dort auftretenden elektrischen Feldstärken zu addieren[13].

$$E = \sum_k E_k \tag{24}$$

2.2.5 Antennendiagramme

Die uns vorliegenden Antennendiagramme bestehen aus einem horizontalen und einem vertikalen Abstrahlmuster (vgl. Abb. 6). Diese Diagramme geben den relativen Gewinn verglichen mit einer isotropen Antenne (Kugelstrahler) an. Der Gewinn wird in dB gemessen, manchmal auch dBi genannt, um den Bezug zur isotropen Referenzantenne zu verdeutlichen.

Die vertikale Schnittebene verläuft durch das Maximum der horizontalen Schnittebene. Per Definition liegt dieses bei 0°. Eine Normalisierung sorgt dafür, dass von allen Werten der Maximalgewinn subtrahiert wird, d.h. alle Werte deshalb nicht-positiv sind.

Es ist nicht möglich, aus dem horizontalen und dem vertikalen Antennendiagramm das

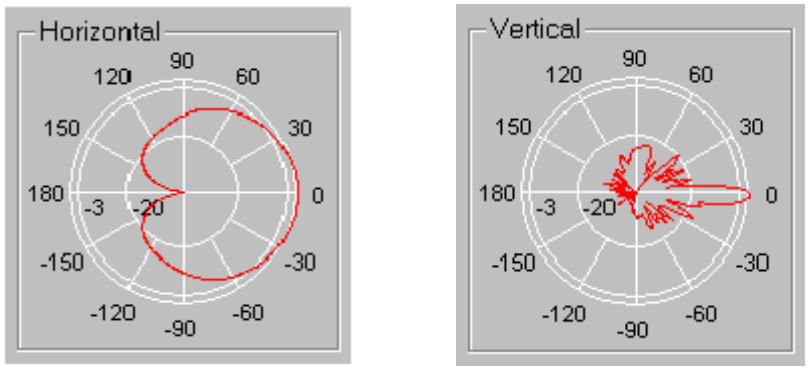

Abb. 6: Horizontales und vertikales Antennendiagramm[3]

dreidimensionale Abstrahlmuster perfekt zu rekonstruieren. Als Interpolationsverfahren kommen u.a. diese Gleichungen in Betracht[3]:

$$g_1(\alpha,\beta) = \min(h(\alpha), v(\beta)) \tag{11}$$

$$g_2(\alpha,\beta) = \frac{(g_{\max} + h(\alpha)) \cdot (g_{\max} + v(\beta))}{g_{\max}} - g_{\max} \tag{12}$$

$$g_3(\alpha,\beta) = h(\alpha) + v(\beta) \tag{13}$$

Zu g_1, g_2 und g_3 ist jeweils der Maximalgewinn zu addieren, um die Normalisierung auszugleichen. In Geryon kommt Formel 13 zum Einsatz.

2.3 Das Walfish-Ikegami-Modell

Mit der Installation von Mobilfunk-Sendeanlagen sind hohe Kosten verbunden. Aus diesem Grunde ist es wünschenswert, bereits während der Planung solcher Anlagen die resultierende Netzabdeckung bestimmen zu können.

Mehrere Wellenausbreitungsmodelle nehmen sich dieser Problematik an. Sie versuchen entweder mit physikalisch-basierten Ansätzen eine Vorhersage zu treffen oder beruhen auf empirischen Erkenntnissen. In die erste Kategorie fallen abgewandelte Raytracing-Algorithmen, die wellenphysikalische Gesetzmäßigkeiten nutzen und daher sehr realistische Ergebnisse liefern können. Leider besitzen sie eine hohe Laufzeitkomplexität und kommen daher kaum zum Einsatz. Die empirischen Modelle sind im Vergleich deutlich schneller zu berechnen und reagieren zugleich weniger sensibel auf Ungenauigkeiten in der modellierten Umwelt. Allerdings ist ihr Einsatzgebiet recht beschränkt: sie sind meist auf urbane Regionen ausgerichtet, bei anderen Szenarien scheitern sie.

Unser Bachelorprojekt Geryon erfordert eine schnelle Berechnung der Ausbreitung von Mobilfunkwellen in einem Stadtmodell. Die Trägerfrequenz liegt in der Regel bei 900 MHz (GSM), 1800 MHz (GSM) oder 2000 MHz (UMTS).

Alle untersuchten Verfahren können nur Ergebnisse in der Qualität liefern, in der uns die verwendeten Szenarien vorliegen. Aufgrund des Datenvolumens, des Vermessungsaufwands und der vielen Änderungen in der Bausubstanz bestehen starke qualitative Einschränkungen in Hinblick auf die uns bereitgestellten Stadtmodelle. Dort werden auch keine elektromagnetischen Einflüsse (Stromleitungen), biologische Faktoren (Bewuchs) und zeitliche Parameter (Aufenthalt der Menschen im Tagesverlauf) berücksichtigt.

Unsere Entscheidung fiel zugunsten des empirischen Walfish-Ikegami-Modells aus. Die verwendeten Formeln und Algorithmen sind verhältnismäßig einfach zu berechnen und daher schnell. Außerdem integriert das Programm Pegasos[1] der T-Mobile Deutschland GmbH dieses Verfahren[2], so dass ein guter Vergleich mit bereits existierenden Ergebnissen möglich ist.

2.3.1 Konzepte des Walfish-Ikegami-Modells

Das Walfish-Ikegami-Modell stellt ein approximatives Verfahren zur Berechnung von Funkwellenpfadverlusten in bebauten Gebieten dar. Es ist Bestandteil von COST231 (european COoperation in the field of Scientific and Technical research 231: Evolution of Land Mobile Radio Including Personal Communication). Die Ergebnisse dieses von 1989 bis

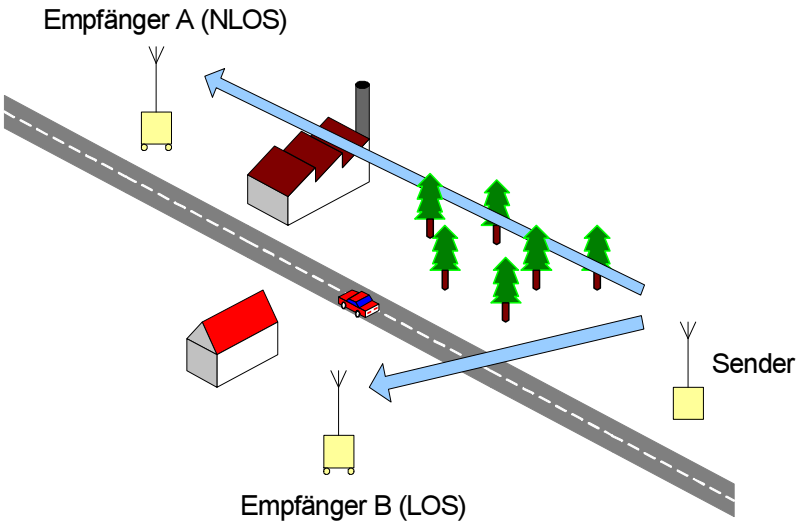

Abb. 7: Unterscheidung LOS / NLOS

1996 europaweit durchgeführten Forschungsvorhabens[6],[7] werden häufig in Wissenschaft und Industrie bei der Untersuchung von Mobilfunknetzen verwendet.

Die Idee beruht darauf, dass der überwiegende Teil der Funkwellen über die Hausdächer in die Straßen hinein gebeugt wird (over-roof propagation) [2],[3],[4],[5].

Die Walfish-Ikegami-Formeln liefern stets einen positiven Wert in Dezibel, der angibt, wie hoch der Verlust einer Funkwelle entlang eines Pfades ist. Dabei sind zwei Fälle zu unterscheiden (siehe Abb. 7):

1. Direkte Sichtverbindung (LOS – „Line Of Sight“):
   - Die Funkwelle gelangt auf direktem Wege vom Sender zum Empfänger
   - Es brauchen keine Reflexionen und Diffraktionen (Beugungen) betrachtet werden
   - Dämpfung der Funkwelle durch Partikel in der Luft

2. Keine direkte Sichtverbindung (NLOS – „No Line Of Sight“):
   - Es ist min. ein Hindernis zwischen Sender und Empfänger
   - Hindernisse erhöhen Dämpfung enorm
   - Starker Einfluss von Reflexionen und Diffraktionen
   - Ebenfalls Dämpfung der Funkwelle durch Partikel in der Luft

Die Parameter des Walfish-Ikegami-Modell unterliegen einigen Einschränkungen, wie Tabelle 2 zeigt.

Diese Werte bewegen sich im Rahmen unserer Anforderungen. Lediglich die maximale Höhe des Empfängers könnte kritisch werden, da wir auch eine Berechnung der Häuserwände durchführen wollen und deren Höhe meist fünf Meter deutlich übersteigt.

Es ist zu beachten, dass alle Längenangaben in Metern erfolgen, lediglich der Abstand d

Parameter	Wertebereich
Trägerfrequenz	f = 800 MHz ... 2000 MHz
Höhe des Senders	h_b = 4 m ... 50 m
Höhe des Empfängers	h_m = 1 m ... 5 m
Entfernung Sender/Empfänger	d = 0,010 km ... 5 km

Tabelle 2: Basisparameter für das Walfish-Ikegami-Modell

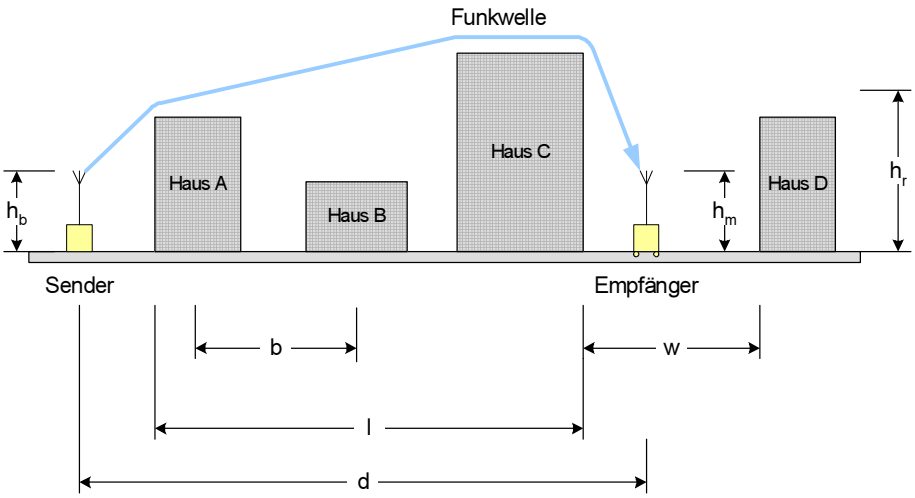

Abb. 8: Wellenausbreitung über Dächer

muss immer in Kilometern betrachtet werden.

2.3.2 Line Of Sight

Bei direkter Sichtverbindung kann der Pfadverlust L_{LOS} sehr einfach berechnet werden:

$$L_{LOS} = 42{,}6 + 20 \cdot \log_{10} f + 26 \cdot \log_{10} d \qquad (25)$$

Zu beachten sind die verwendeten Einheiten: die Frequenz f muss in Megahertz vorliegen, für den Abstand d kommt die Einheit Kilometer zum Zuge.

Beispiel für L_{Los}: In 0,5 km Entfernung erhält man bei 900 MHz (GSM) einen Pfadverlust von 93,8 dB.

2.3.3 No Line Of Sight

Dieser Teil ist der eigentliche Kern des Walfish-Ikegami-Modells. Die Strahlenausbreitung in urbanen Gebieten zeichnet sich aus durch eine Ausbreitung der Wellen über die Dächer und eine horizontale Beugung in die Straßen.

2.3.3.1 Parameter

Als Berechnungsgrundlage dient eine zum Boden vertikal orientierte Schnittebene, die Sender und Empfänger enthält.

Der vertikale Schnitt in Abb. 8 verdeutlicht, dass eine Beugung (Diffraktion) der Wellen an mehreren Dachkanten erfolgt. Um diese Phänomene erfassen zu können, sind zusätzlich zu d, h_b, h_m und f weitere Parameter notwendig, diese sind in Tabelle 3 aufgeführt.

Parameter	Einheit
Durchschnittliche Gebäudehöhe	h_r (in m)
Breite der Straße, in der sich der Empfänger befindet	w (in m)
Mittlerer Abstand der Gebäudemittelpunkte	b (in m)
Strecke, die die Wellen über Dächern zurücklegen	l (in km)

Tabelle 3: Zusätzliche Parameter für NLOS

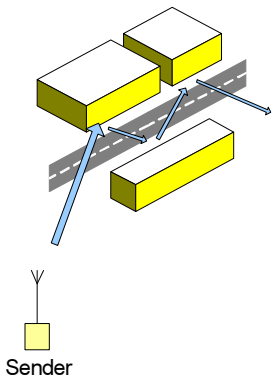

Abb. 9: Wellenreflexion entlang von Straßen

Reflexionen an den Hauswänden der Straße, in der der Empfänger sich aufhält, sind ebenfalls von Bedeutung. Aus diesem Grunde muss man den Winkel zwischen dem Wellenpfad und der Straße berücksichtigen. Dieser liegt stets zwischen 0° und 90°, in Abb. 9 sind es in etwa 30°.

Für eine genauere Justierung des empirisch-basierten Walfish-Ikegami-Modells unterscheidet man zwischen kleinen bis mittleren Städten und großen Metropolen. Somit erweitert sich die Menge von Parametern um die zwei in Tabelle 4 aufgeführten.

2.3.3.2 Formelwerk

Allgemein

Im Walfish-Ikegami-Modell berechnet sich der Pfadverlust L als Summe der Freiraumdämpfung L_{fs}, der Abschlussdämpfung L_{rts} und der Dämpfung über Dächern L_{msd}:

$$L = \begin{cases} L_{fs} + L_{rts} + L_{msd} & L_{rts} + L_{msd} > 0 \\ L_{fs} & L_{rts} + L_{msd} \leq 0 \end{cases} \tag{26}$$

Freiraumdämpfung (free-space loss)

Ähnlich dem Line-Of-Sight-Fall gibt es auch bei No-Line-Of-Sight eine Freiraumdämpfung L_{fs}, die lediglich von der Trägerfrequenz f und der Entfernung d abhängt:

$$L_{fs} = 32{,}4 + 20 \cdot \log_{10} d + 20 \cdot \log_{10} f \tag{27}$$

Da die Trägerfrequenz konstant ist, hängt die Dämpfung nur von der Entfernung d ab und erzeugt in der Visualisierung die Bildung eines radialen Farbverlaufs.

Beispiel für L_{fs}: In 0,5 km Entfernung erhält man bei 900 MHz (GSM) einen Pfadverlust von 85,5 dB.

Parameter	Wertebereich
Winkel zwischen Wellenpfad und Straßenachse	ϕ ($0° \leq \phi \leq 90°$)
Besiedlungsdichte	{ kleine/mittlere Stadt; große Metropole }

Tabelle 4: Zusätzliche Parameter für NLOS (Fortsetzung)

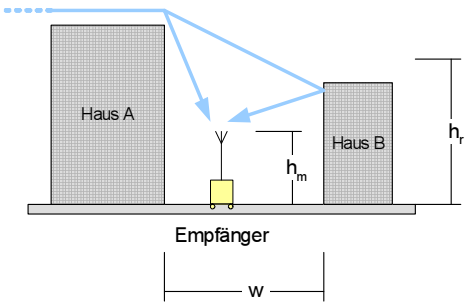

Abb. 10: Abschlussdämpfung

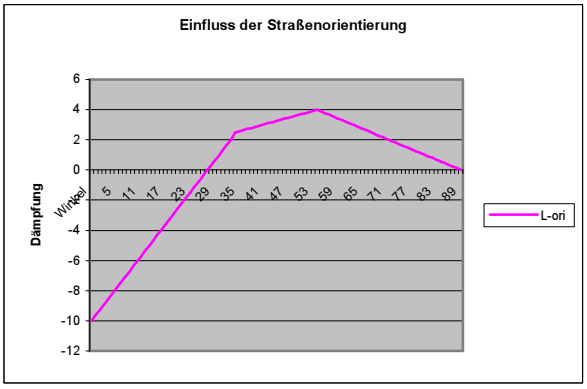

Abb. 11: Dämpfung aufgrund der Straßenorientierung

Abschlussdämpfung (roof-top-to-street diffraction and scatter loss)

Die Abschlussdämpfung ist das Resultat der Diffraktion (Beugung) der Funkwellen von den Dachkanten in die Straße hinein, in der der Empfänger steht (siehe Abb. 10). Neben der Trägerfrequenz f, der Straßenbreite w, der durchschnittlichen Gebäudehöhe h_r und der Höhe der Empfängers h_m spielt auch der Winkel φ zwischen dem Wellenpfad und der Straßenachse eine Rolle:

$$L_{rts} = -16{,}9 - 10 \cdot \log_{10} w + 10 \cdot \log_{10} f + 20 \cdot \log_{10}(h_r - h_m) + L_{ori} \tag{28}$$

$$L_{ori} = \begin{cases} -10 + 0{,}354 \cdot \varphi & 0° \le \varphi < 35° \\ 2{,}5 + 0{,}075 \cdot (\varphi - 35°) & 35° \le \varphi < 55° \\ 4{,}0 - 0{,}114 \cdot (\varphi - 55°) & 55° \le \varphi \le 90° \end{cases} \tag{29}$$

Eine breitere Straße bewirkt demzufolge eine geringere Dämpfung als eine schmale Straße. Ebenso zieht ein geringer Höhenunterschied zwischen dem Empfänger und den Gebäudeoberkanten nur eine geringe Dämpfung nach sich.

Die Orientierung φ der Straßenachse in Bezug zum Wellenpfad äußert sich in den in Abb. 11 dargestellten Dämpfungswerten.

Beispiel für L_{rts}: In einer 20 m breiten Straße, deren Achse einen Winkel von 30° zum Wellenpfad bildet und von durchschnittlich 12 m hohen Häusern umgeben wird, ermittelt man für einen Empfänger auf einer 900-MHz-Trägerfrequenz eine Abschlussdämpfung von 32,4 dB.

Dämpfung über Dächern (multiscreen diffraction loss)

Die Dämpfung über Dächern wird teilweise auch als „multi-obstacle diffraction“ bezeichnet.

$$L_{msd} = L_{bsh} + k_a + k_d \cdot \log_{10} d + k_f \cdot \log_{10} f - 9 \cdot \log_{10} b \tag{30}$$

L_{bsh} bestimmt die Dämpfung aufgrund der Höhe der Antenne über den Dächern.

$$L_{bsh} = \begin{cases} -18 \cdot \log_{10}(1 + h_b - h_r) & h_b > h_r \\ 0 & h_b \le h_r \end{cases} \tag{31}$$

k_a und k_d stellen Korrekturfaktoren hinsichtlich der Höhe der Sendeantenne h_b über der durchschnittlichen Gebäudehöhe h_r dar, k_f wird von der Stadtgröße beeinflusst.

$$k_a = \begin{cases} 54 & h_b > h_r \\ 54 - 0{,}8 \cdot (h_b - h_r) & h_b \le h_r \quad und \quad d \ge 0{,}5 \\ 54 - 1{,}6 \cdot (h_b - h_r) \cdot d & h_b \le h_r \quad und \quad d < 0{,}5 \end{cases} \tag{32}$$

$$k_d = \begin{cases} 18 & h_b \ge h_r \\ 18 - 15 \cdot (h_b - h_r) / h_b & h_b < h_r \end{cases} \tag{33}$$

$$k_f = \begin{cases} -4 + 0{,}7 \cdot f/925 - l & kleine / mittlere\ Städte \\ -4 + 1{,}5 \cdot f/925 - l & Großstädte \end{cases} \tag{34}$$

Abweichungen der T-Mobile-Formeln

Die hier vorgestellten Formeln sind in den Quellen [3], [4] und [5] identisch. Sie weichen aber von den von der T-Mobile zur Verfügung gestellten Gleichungen[2] leicht ab. Genauer gesagt existieren zwei Vorzeichenunterschiede in (24) und (25):

$$L_{rts} = -16{,}9 + 10 \cdot \log_{10} w + 10 \cdot \log_{10} f + 20 \cdot \log_{10}(h_r - h_m) + L_{ori} \tag{24a}$$

$$L_{ori} = \begin{cases} -10 + 0{,}354 \cdot \varphi & 0° \le \varphi < 35° \\ 2{,}5 + 0{,}075 \cdot (\varphi - 35°) & 35° \le \varphi < 55° \\ 4{,}0 + 0{,}114 \cdot (\varphi - 55°) & 55° \le \varphi \le 90° \end{cases} \tag{25a}$$

2.3.4 Implementation

In Geryon ist allein die Klasse `WavePropagation` für die Umsetzung des Walfish-Ikegami-Modells zuständig. Sie hat Zugriff auf das Stadtmodell und kennt die Position der Sendeantenne.

Ein Programm, das `WavePropagation` benutzt, muss folgende Schritte durchlaufen:

1. Mit `setModel` das Stadtmodell festlegen
2. Über `setAntennaPosition` den Standort der Sendeantenne definieren
3. Anhand von `getWalfishIkegamiPathLoss` den Pfadverlust an einem bestimmten Punkt bestimmen
4. Punkt 3 beliebig wiederholen

Die Methoden `getLineOfSightPathLoss` und `getNoLineOfSightPathLoss` werden intern von `getWalfishIkegamiPathLoss` benutzt.

Um alle Parameter von `getNoLineOfSightPathLoss` bestimmen zu können, schickt `getWalfishIkegamiPathLoss` virtuelle Strahlen per Raytracing vom Sender zum Empfänger durch das Stadtmodell. Dabei protokolliert die Methode alle Schnittpunkte mit Gebäuden. Da auch die Straße, in der sich der Empfänger befindet, wichtige Parameter liefert, verlängern wir den virtuellen Strahl über den Empfängerstandort hinaus (siehe Abb. 12).

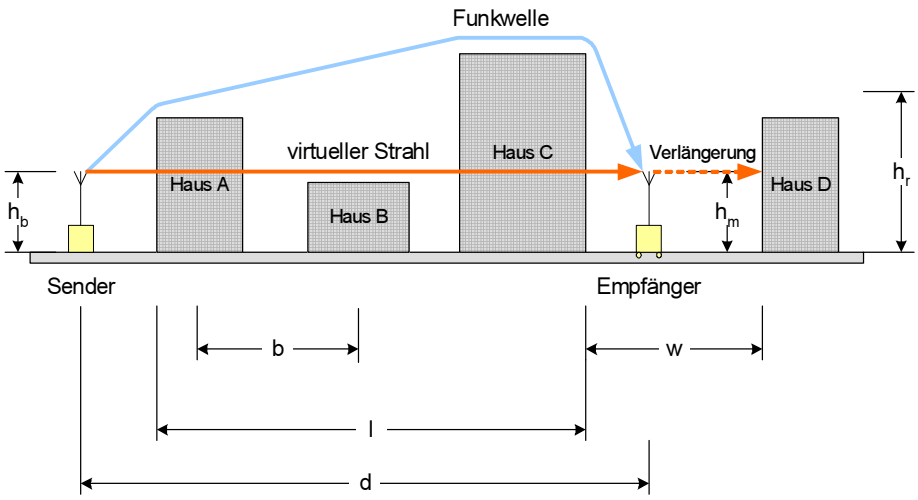

Abb. 12: Virtuelle Strahlen im Stadtmodell

Die Straßenachse liegt uns nicht explizit als Datensatz vor. Wir behelfen uns, indem wir davon ausgehen, dass die Häusernormalen orthogonal zur Straßenachse verlaufen.

3. Grobe Softwarearchitektur

Grundsätzlich lässt sich Geryon sehr leicht in zwei unabhängige Teile zerlegen: die eher technische Visualisierung und die Schnittstelle zum Benutzer. Diese beiden lassen sich getrennt betrachten und entwickeln. Sie können Abb. 13 entnommen werden. Der Visualisierungsberechner und der Renderer gehören zu ersterem, der GUI-Verwalter und der Kamera-Steuerer zu letzterem.

Die für die Visualisierung zuständigen Teile sind in der Toolbox zusammengefasst und profitieren in hohem Maße von der Grafikbibliothek VRS und dem 3D-Kartensystem LandExplorer oder kurz LDX. Letzteres übernimmt vollständig die Darstellung des Terrains und bietet darüber hinaus die Möglichkeit an, Rasterdaten einzublenden. Über diese Möglichkeit werden die Feldstärken visualisiert. Die Anzeige weiterer Elemente, wie zum Beispiel der Basisstation und einer idealisierten Funkwolke, setzt direkt auf VRS auf. Die Darstellung des Stadtmodells wurde vollständig neu entwickelt, allerdings als Erweiterung zu VRS. Dies erlaubt es, das Stadtmodell zusammen mit den anderen dargestellten Elementen in den Szenengraph zu integrieren. Große Teile des Renderers entstammen VRS. Er wird jedoch sowohl durch den LandExplorer, beispielsweise für das Terrain, als auch durch Geryon, beispielsweise für das Stadtmodell, erweitert.

Die für die Schnittstelle zum Benutzer erforderliche Funktionalität ist fast vollständig in der GUI-Toolbox untergebracht. Sie wurde auf der Basis der Qt-Bibliothek entwickelt, verwendet allerdings ebenfalls Teile von VRS, LDX und der Toolbox. So bietet LDX beispielsweise die von uns genutzte Möglichkeit an, dem Terrain Textmarkierungen hinzu zu fügen. Der aufgebaute Szenengraph entstammt VRS, einige notwendige Koordinatentransformationen werden durch die Toolbox übernommen.

Die Paketbeziehungen sind in Abb. 14 veranschaulicht. Auf der untersten Ebene liegt

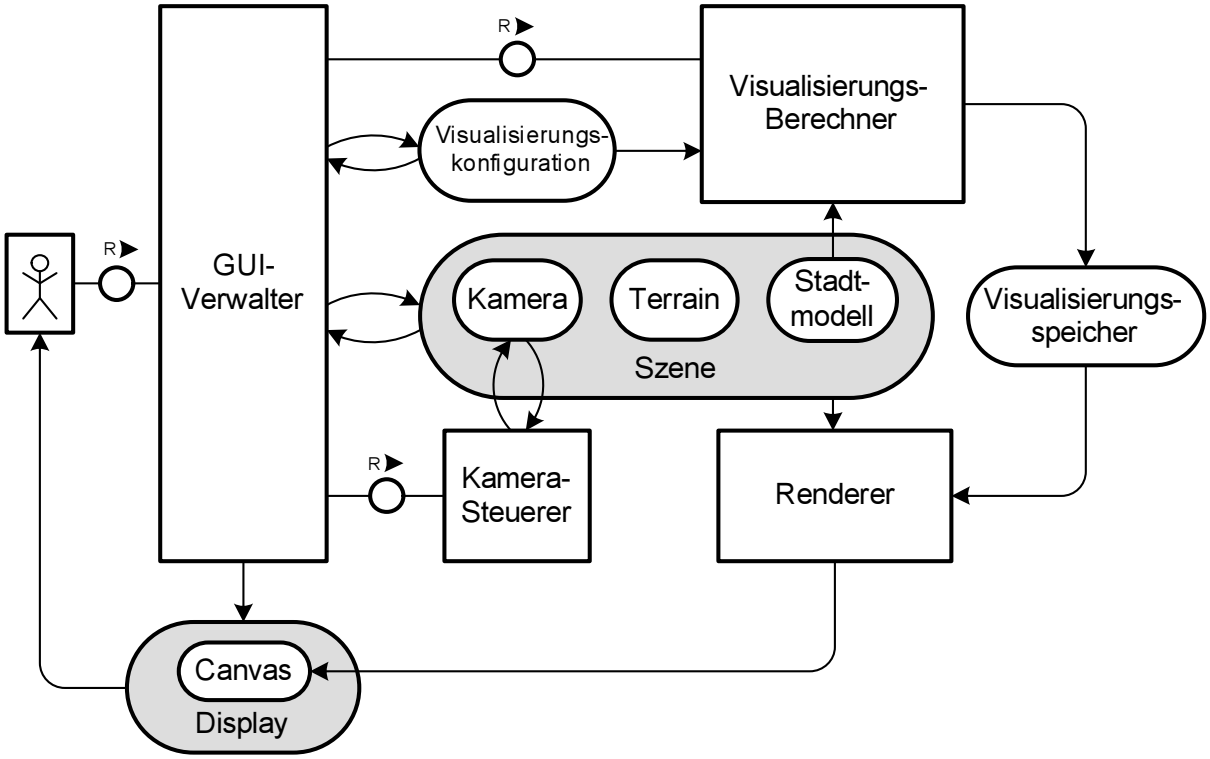

Abb. 13: Der Gesamtaufbau von Geryon

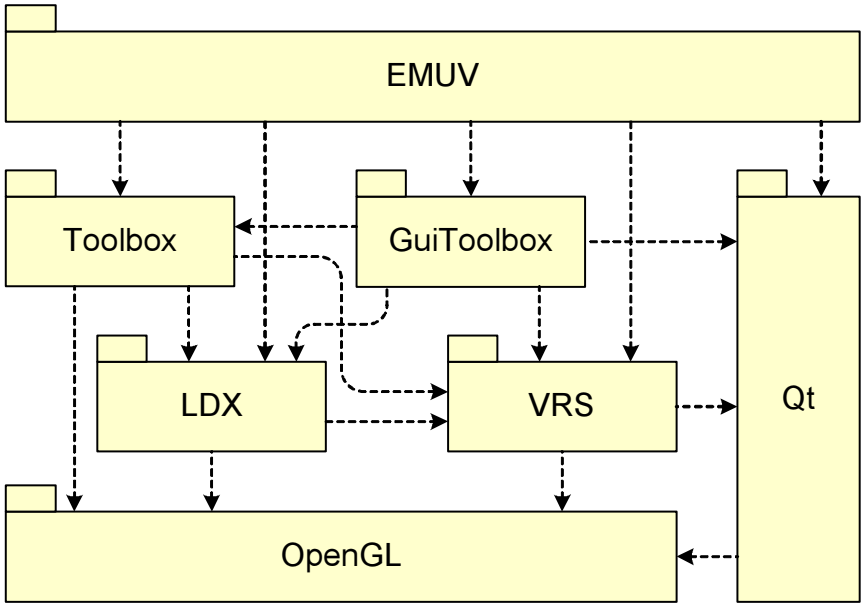

Abb. 14: Paketbeziehungen

OpenGL. Qt besitzt eine grundsätzliche Unterstützung für OpenGL. VRS baut zum großen Teil auf OpenGL auf und bietet eine Anbindung an Qt in Form eines Widgets. LDX stützt sich zum Großteil auf in VRS bereits angebotene Elemente, greift aber auch direkt auf OpenGL zurück. Ähnlich verhält sich die Toolbox, sie benutzt, wie schon erwähnt, Teile aus LDX, VRS und greift aus Performancegründen auch direkt auf OpenGL zu. Die GUI-Toolbox ist auf Basis von Qt realisiert. Ihre Bestandteile manipulieren Elemente aus VRS und benutzen dazu auch Funktionalität aus LDX und der Toolbox. EMUV fasst alles zusammen und greift dabei auf alle Schichten bis auf OpenGL zu.

3.1 Der Visualisierungsteil

Wie schon erwähnt, besteht der Visualisierungsteil aus zwei wesentlichen Komponenten: dem Berechner und dem Renderer. Beide werden ausführlich im Kapitel 4.2 erklärt. Im Gegensatz zum Berechner besteht der Renderer zu großen Teilen aus bereits vorhandenen Elementen, die VRS und LDX entstammen.

Um VRS und LDX reibungslos nutzen zu können, war es erforderlich, die von uns benötigten neuen Darstellungselemente in den VRS-Mechanismus zu integrieren. Dieser gibt eine Teilung von den zu rendernden Elementen, den sogenannten Shapes, und dem tatsächlichen Rendering-Algorithmus vor. Die Struktur des Renderers ist so von vorn herein mehr oder weniger vorgegeben.

Abb. 15 zeigt hauptsächlich die von uns entwickelten Erweiterungen. Der Terrain-Renderer zusammen mit dem Terrain als Element des Szenengraphen entstammen bereits dem LDX, der Canvas konnte ohne Änderungen aus VRS entnommen werden. Alle anderen Szenengraph-Elemente zusammen mit ihren Renderern und der Visualisierungsspeicher mussten neu entwickelt werden. Allerdings verwenden der FS-Textur-Auswerter, der Funkmast-Renderer und auch der Funkkeulen-Renderer intern wieder von VRS bereitgestellte Elemente.

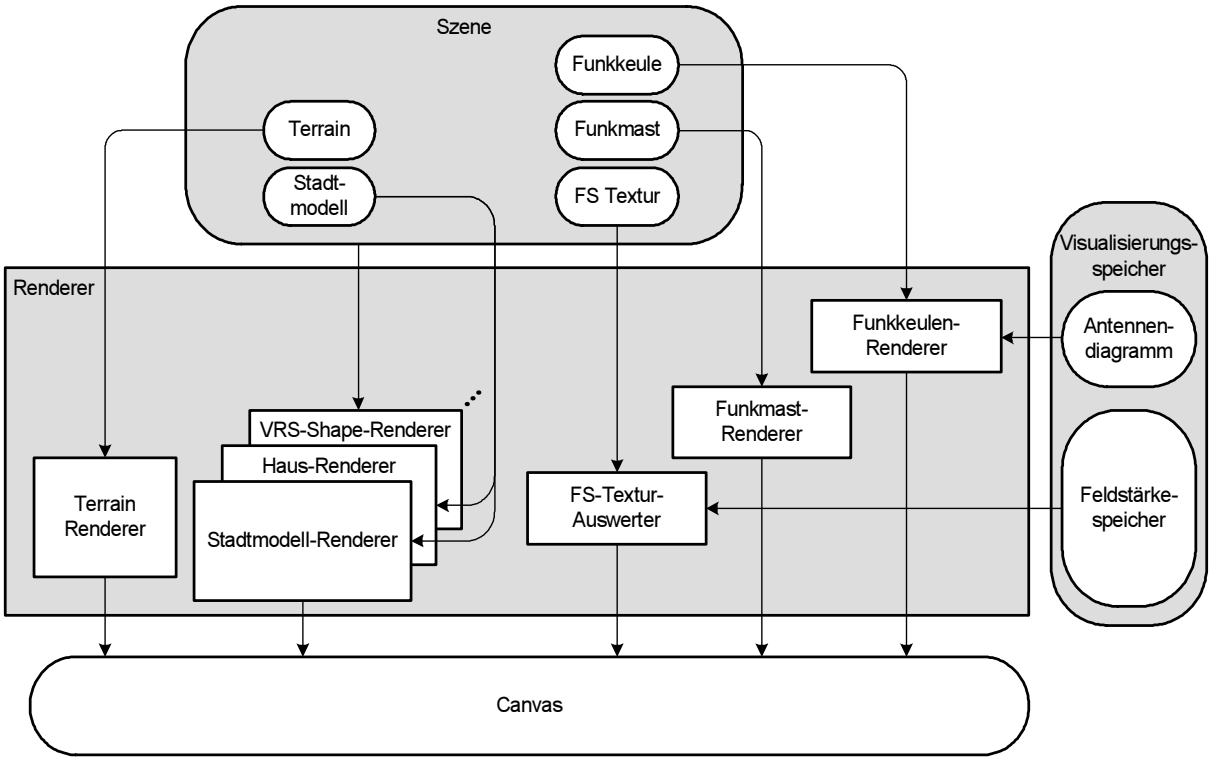

Abb. 15: Der Aufbau des Renderers

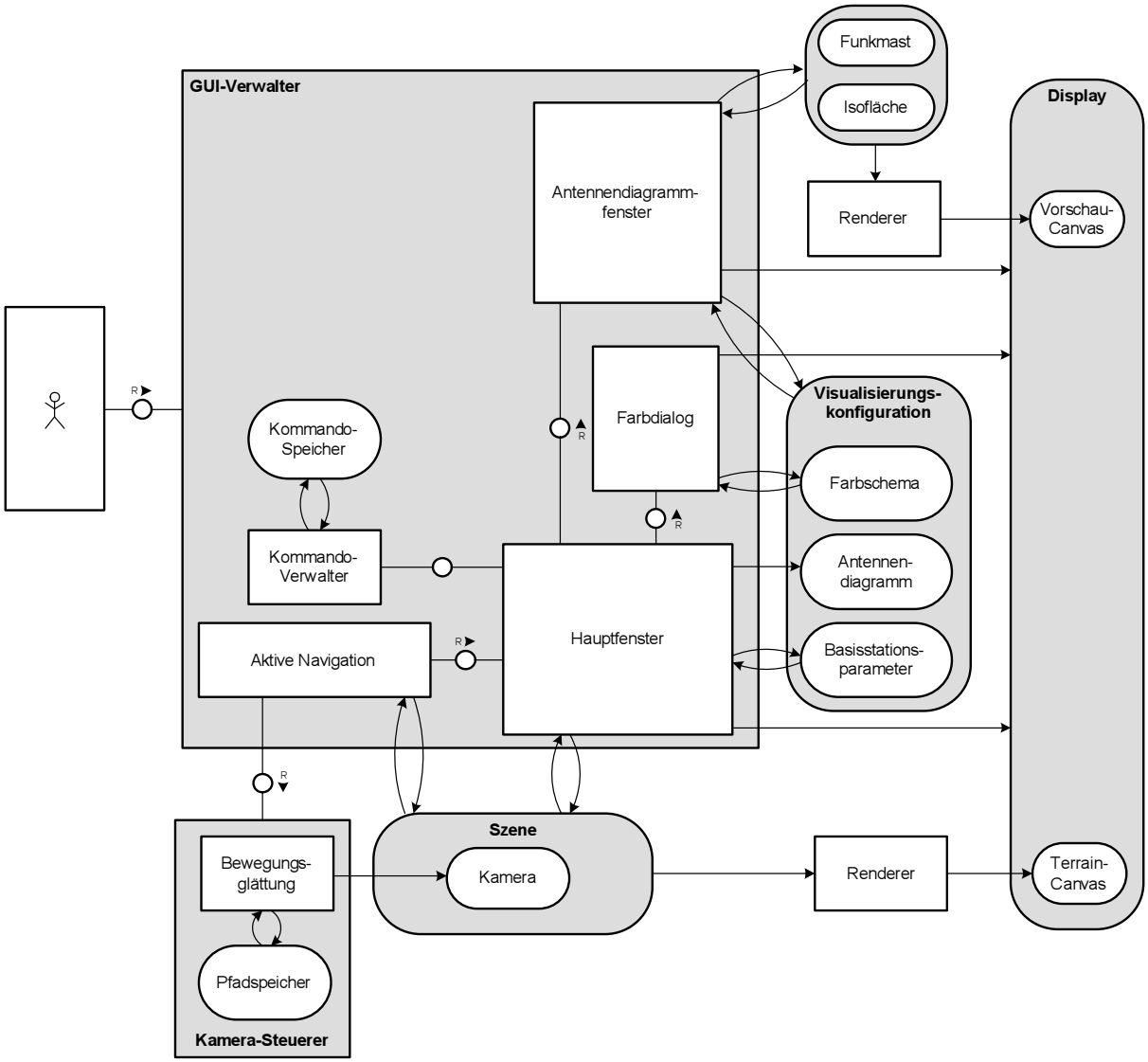

Abb. 16: Aufbau des GUI-Verwalters

3.2 Die Schnittstelle zum Benutzer

Die Struktur der Teile von Geryon, die sich mit der Benutzerschnittstelle befassen, ist wesentlich komplexer. Die Abläufe für einzelne, durch den Benutzer angestoßene Schritte sind an sich relativ simpel, aber eben durch diese vom Benutzer bestimmte Steuerung der Schritte sind sie weniger gut zu überschauen.

Abb. 16 zeigt die beteiligten Akteure, Speicher und Kommunikations- bzw. Anstoßbeziehungen. Sie werden im Kapitel 0 ausführlich beschrieben.

4. Visualisierung

4.1 Das Stadtmodell

Es ist offensichtlich, dass für eine sinnvolle Visualisierung der Funkwellenausbreitung auch die Gebäude und ergänzende Elemente (Bäume, Straßen, Gewässer usw.) eines Stadtbildes visualisiert werden müssen. In diesem Kapitel werden die Modellierung und das Rendering der Häuser behandelt.

Um zu einer Darstellung zu kommen, müssen zunächst einmal die geographischen Sachverhalte erfasst, d.h. die realen Gebäude vermessen, fotografiert, anhand von Bauplänen oder auf andere Art und Weise digital erfasst werden. Dabei entstehen für größeren Städte mitunter sehr umfangreiche Datenmengen.

Unsere Visualisierung muss vor allem zwei Forderungen genügen: Sie muss interaktiv sein und sie muss perzeptiv effektive Techniken einsetzen. Es sind mehr als 10 Bilder pro Sekunde notwendig, damit der Anwender das Gefühl hat, sich tatsächlich innerhalb der simulierten Umgebung zu befinden. Gleichzeitig will der Anwender Objekte intuitiv identifizieren können. Es hat sich gezeigt, dass reiner Photorealismus nicht unbedingt von Vorteil ist. Man denke etwa an den Versuch, den Puck bei der Übertragung eines Eishockey-Spiels nicht aus den Augen zu verlieren. Außerdem ist Photorealismus mit heutiger Hardware in Echtzeit nicht erreichbar. Als Kompromiss erzeugt ein comic-artiger Stil einen besseren Eindruck, da wesentliche Merkmale hervorgehoben werden können.

Die Forderungen nach Interaktivität und hoher optischer Qualität stehen im Widerspruch, denn sie stellen enorm hohe Anforderungen an das Grafiksystem. Eine der Kernaufgaben unseres Projekts war die sorgfältige Abwägung zwischen Geschwindigkeit und Qualität. Wir erforschten die Möglichkeiten und Grenzen der zugrundeliegenden Bibliotheken VRS[17] und LandExplorer[18], experimentierten mit verschiedenen Techniken in OpenGL[19] und führten Performance-Analysen mit Nvidia-Hardware durch[20].

Es mussten Wege gefunden werden, die teilweise riesigen Datenmenge für das Rendering zu reduzieren, ohne die Bildqualität erheblich zu verschlechtern. Bei sehr großen Städten wird darüber hinaus sogar die Ressourcenbelastung, insbesondere des Arbeitsspeichers, aktueller Rechner zum Problem.

4.1.1 Das Stadtmodell der T-Mobile

Die Erfassung der physikalischen Daten ist nicht Teil unseres Projekts, sie erfolgt durch die T-Mobile. In deren Datenbank sind polygonale Informationen abgelegt, welche die Gebäude repräsentieren. Diese ist über die PSD (Pegasos Spatial Data) Schnittstelle ansprechbar,

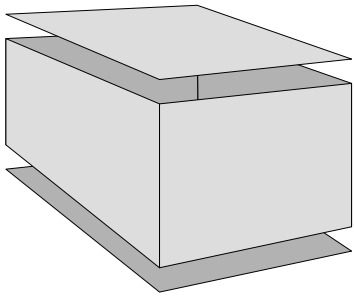

Abb. 17: Hausdarstellung


Abb. 18: Frankfurter Skyline und Altstadt

welche wir jedoch nicht direkt benutzen. Unser Projekt liest eine reine Textdatei im ASCII-Format ein, welche jedoch das Stadtmodell in derselben Form enthält.

Den wichtigsten Aspekt für das Rendering stellt der Aufbau eines einzelnen Gebäudes dar. Es wird vereinfacht durch eine Polygonmenge repräsentiert. Diese setzt sich aus drei unterschiedlichen Mengen für das Dach, für die Wände und für den Boden zusammen. Es wird eine indizierte Darstellung verwendet. Sämtliche Eckpunkte werden im geographischen Koordinatensystem mit Höhen über NN spezifiziert. Für die Spezifikation der Polygone werden dann nur noch Indizes verwendet. Weiterführende Informationen können Quelle [27] sowie dem Kapitel 4.1.6.1 entnommen werden. Abb. 17 enthält eine anschauliche Darstellung.

Die Dächer liegen uns nur als Flachdächer vor. Für viele Neubauten und vor allem für die Skyline (siehe Abb. 18) ist das akzeptabel und korrekt. In der Altstadt überwiegen dagegen Spitz- und Walmdächer. Hier leidet der Wiederkennungswert deutlich und nicht-technisches Publikum hat unter Umständen große Probleme, sich im Stadtmodell zu orientieren. Da wir aufgrund unserer Datenbasis keinerlei Information über die reale Dachform besitzen, können wir dieses Problem jedoch nicht adressieren

Konkret liegen uns zwei Modelle in Form von zwei Textdateien vor, zum einen Offenbach mit etwa 12.000 und Frankfurt mit etwa 44.000 Gebäuden.

4.1.2 Perzeptive Technik: Silhouette-Rendering

Geryon soll auch für Großstädte geeignet sein. In einer Stadtansicht sind dann meist mehrere tausend Häuser gleichzeitig sichtbar. Aufgrund der begrenzten Bildschirmauflösung heutiger Monitore umfassen die meisten Häuser nur wenige Pixel in der Darstellung.

Um dennoch einzelne Häuser voneinander abgrenzen zu können, setzen wir auf eine Akzentuierung der Häuserkanten, auch als Silhouette-Rendering bekannt. Eine besonders

Abb. 19: Verstärkte Silhouette bei Cartoons

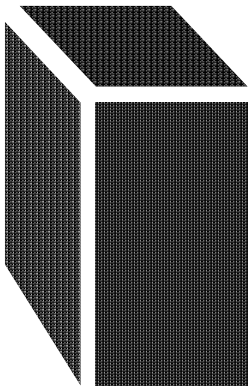

Abb. 20: Schritt 1 schematisch - Wände und Dach

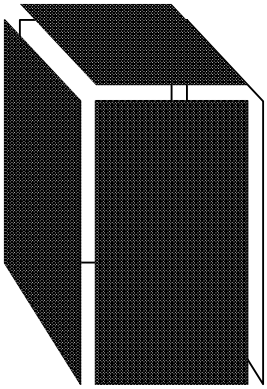

Abb. 21: Schritt 2 schematisch - Kanten nachzeichnen

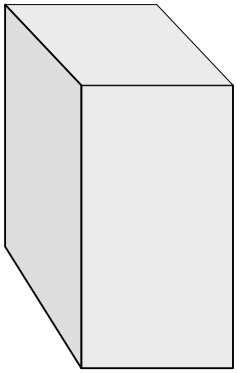

Abb. 22: Tatsächliches Ergebnis

einfach zu implementierende Technik nutzt die vorhandenen Häuserkantendaten und zeichnet sie mit einer kontrastierenden Farbe nach. Dafür gut geeignet sind dunkle Farben oder gar Schwarz, da die meisten Menschen diesen Zeichenstil aus Cartoons kennen. Es reicht aus, die Kanten der Wände zu betonen, wie in Abb. 20 bis Abb. 22 ersichtlich wird.

4.1.3 Technische Probleme für das Rendering

4.1.3.1 Die Menge der Eckpunkte

Probleme für das Rendering ergeben sich fast ausschließlich aus den hohen Gebäudezahlen. Gerade durch die polygonale Darstellung ist die Darstellung eines einzelnen Gebäudes ohne weiteres möglich. Überschlägt man jedoch die Zahl der notwendigen Eckpunkte für ein Stadtmodell mit 44.000 Häusern, erkennt man leicht, wo die Schwierigkeiten liegen.

Aus technischen Gründen müssen wir auf eine indizierte Darstellung verzichten (siehe Kapitel 4.2). Daher müssen selbst für ein einfaches würfelförmiges Gebäude 24 Eckpunkte übertragen werden. Zusätzlich müssen für jeden Eckpunkt pro Textureinheit zwei Texturkoordinaten übertragen werden. Außerdem ist es für die optische Qualität des Bildes notwendig, die Kanten verstärkt darzustellen. Daher müssen zwei Drittel der Eckpunkte zum Zeichnen von Linien ein weiteres Mal übertragen werden, allerdings ohne Texturkoordinaten. Geht man von einem Durchschnittswert von 40 Eckpunkten aus (konkret sind es beispielsweise zwei Millionen Eckpunkte für die 44.000 Gebäude des Frankfurter Stadtmodells), so müssen $40\cdot(3+n_t\cdot 2)+40\cdot{}^{2}/_{3}\cdot 3$ Werte pro Gebäude an die Graphikhardware gesendet werden. Dabei ist n_t die Anzahl der verwendeten Textureinheiten. Geryon verwendet für die Darstellung der Feldstärken bis zu vier Textureinheiten, wenn diese vorhanden sind. Dies ergibt im Schnitt 521 übertragene Werte. Handelt es sich dabei um mit acht Byte dargestellte Gleitkommawerte, so müssen also $521\cdot 8=4.168$ Byte übertragen werden. Für 44.000 Gebäude ergibt dies 183.392.000 Byte, das sind in etwa 175 Megabyte.

Aufgrund der Gesamtgröße der Daten können sie offensichtlich nicht ständig im Speicher der Graphikkarte vorgehalten werden. Dieser ist durch das restliche Terrain und Texturen noch zusätzlich belastet. Folglich muss die Datenmenge für jedes Bild übertragen werden. Nimmt man eine theoretische Datenübertragungsrate von 1.067 Mbyte/s des AGP4x-Bus (welche bezogen auf die Nutzdatenmenge nicht einmal annähernd erreicht wird) an, so würde man allein durch ihn die Bildrate auf etwa sechs pro Sekunde beschränken.

Auch die Verarbeitung von so vielen Eckpunkten lässt gängige Hardware an ihre Grenzen stoßen.

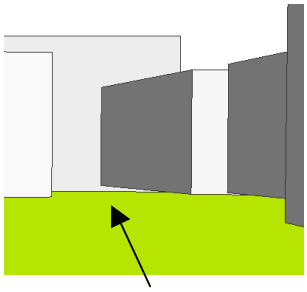

Abb. 23: Ein Haus schwebt über dem Terrain

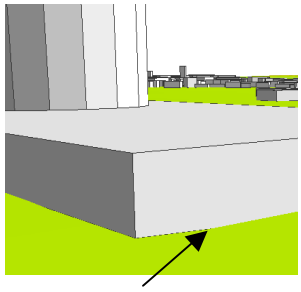

Abb. 24: Ein Gebäude taucht in das Terrain ein

4.1.3.2 Konflikte mit dem Terrain

Geryon ist in der Lage, beliebige Gelände, d.h. Terrains, zu laden. Dieser Aspekt ist Bestandteil vom LandExplorer, der sich um alle dazu notwendigen technischen Details, wie z.B. nebenläufiges Laden und Level-of-Detail, kümmert.

Es gibt jedoch Unterschiede hinsichtlich der Auflösung und Genauigkeit zwischen Terrains und Gebäudedaten. In der Regel sind letztere mit einer deutlich höheren Präzision erfasst worden. Dies führt im Rendering zu deutlich sichtbaren Artefakten, da Gebäude entweder ins Terrain eintauchen oder darüber schweben.

Zur Optimierung der Geschwindigkeit arbeitet der LandExplorer mit einem sichtabhängigen Level-of-Detail-Algorithmus. Dieser hat den Effekt, dass zwischen ohnehin schon ungenauen Daten noch interpoliert wird, somit verstärken sich die Artefakte.

4.1.4 Verschiedene Lösungsansätze

Die Menge von Eckpunkten muss reduziert werden. Nicht immer tragen alle Daten zum visuellen Gesamteindruck bei. Im Extremfall befindet sich der Betrachter mitten in der Stadt oder er schaut von weit oberhalb der Stadt. Im ersten Fall sind einige Gebäude groß im Vordergrund zu sehen, einige wenige klein im Hintergrund. Die meisten sind aber hinter dem Betrachter oder liegen verdeckt hinter den Gebäuden im Vordergrund. Im zweiten Fall sind zwar möglicherweise fast alle Gebäude zu sehen, sie sind jedoch so klein, dass man sie kaum erkennen kann. In beiden Fällen kann somit die Datenmenge erheblich verringert werden.

Für den Fall, dass der Betrachter sich weit außerhalb der Stadt befindet, kann die Menge der dargestellten Polygone aufgrund ihrer geringen Größe reduziert werden. Die Gebäude besitzen allerdings eine für das Rendering sehr unangenehme Eigenschaft: Sie sind nicht miteinander verbunden. Dadurch scheiden einschlägig untersuchte Lösungen für Level-Of-Detail-Terrain-Rendering aus. Sie basieren alle auf der Annahme, dass ein geschlossenes Netz von Höhenpunkten vorhanden ist. Ein künstliches Verbinden der Häuser zu einem solchen Netz führt sofort zu nicht akzeptierbaren Fehlern in der Darstellung. Übrig bleiben also nur die Reduzierung pro Haus und das Ersetzen ganzer Häusergruppen. Pro Gebäude kann allerdings nicht allzu viel reduziert werden, da ein einzelnes Gebäude relativ wenig Polygone besitzt.

Für den anderen Fall, bei dem sich der Betrachter mitten in der Stadt befindet, kann die Menge durch geometriebasiertes Clipping und Culling verringert werden.

4.1.4.1 Flaschenhals Datenübertragung

Da die Zeit für das Rendering des Stadtmodells hauptsächlich an der Übertragung der Daten auf die Grafikkarte hängt, lassen wir diesem Aspekt besondere Aufmerksamkeit zukommen.

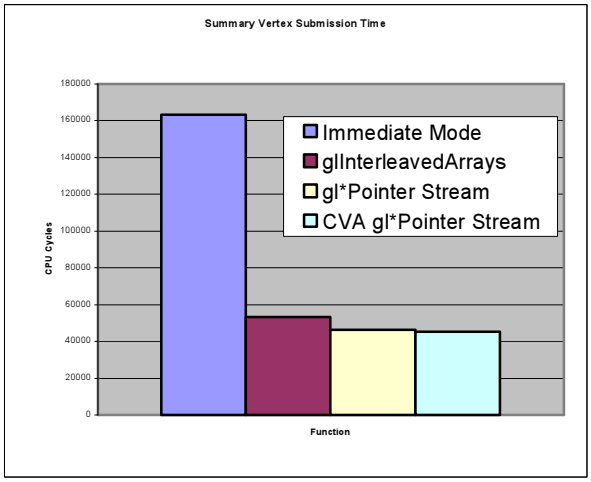

Abb. 25: Vergleich der Datenübertragungstechniken[6]

Die zugrundeliegende OpenGL-Bibliothek wird in der Regel im sogenannten Immediate Mode angesprochen, d.h. jedes geometrische Primitiv und dessen Attribute werden mit separaten Kommandos übertragen. Ein typischer Vertreter dieser Befehle ist `glVertex`. Dieser Modus hat den Vorteil, am flexibelsten zu sein. Er ist in der Lage, alle möglichen Visualisierungstechniken zu realisieren.

Allerdings steht der Immediate Mode in Konflikt zu der Pipeline-Architektur gängiger Grafikkarten. Diese arbeiten nur dann optimal, wenn sie Daten en bloc erhalten. Die meisten Änderungen des Renderingzustands erfordern ein kompletten Entleeren, einen sogenannten Flush, der mitunter extrem langen Pipelines. Ebenso erreichen viele interne Caches nicht ihre volle Effizienz.

Unter OpenGL existieren aus diesem Grunde Vertex Arrays. Ihr Zweck ist es, eine große Menge an Geometriedaten zu übertragen, die alle den gleichen Renderingzustand nutzen. Vertex Arrays übergeben dem Grafikkartentreiber lediglich Zeiger auf Daten für darzustellende Elemente, die im Hauptspeicher bereit stehen. Diese können dann vom Grafikkartentreiber sehr effizient über den AGP-Bus transferiert werden, was der CPU kaum Arbeit aufbürdet. Da jeder Bus-Transfer etwas Verwaltungsoverhead benötigt, sollten möglichst viele Daten auf einmal verarbeitet werden. Nvidia spricht aber davon, dass zu große Transfers auch wieder die Gesamtperformance reduzieren und empfiehlt daher etwa 1000 Eckpunkte (inkl. deren Normalen und Texturkoordinaten) als optimale Array-Größe.

Vertex Arrays sind der Grundbaustein für viele High-Performance-Renderer. Das für seine hohe technische Qualität bekannte Spiel Quake III (indiziert) wickelt 99% der Kommunikation mit dem Grafikkartentreiber über diese Schnittstelle ab[8].

4.1.5 Die Lösung in Geryon

4.1.5.1 Reduzierung der Eckpunkte

Geryon benutzt zur Reduzierung der Datenmenge das hierarchische Ersetzen von Häusergruppen und das ebenfalls hierarchische geometriebasierte Culling. Ganze Häusermengen können unter Umständen von vorn herein vom Rendering ausgeschlossen werden, da sie nicht sichtbar sind. Andere können durch ein einfacheres Objekt ersetzt werden und die einfachen Objekte ihrerseits können wiederum ersetzt werden usw.

Um dies bestmöglich zu unterstützen, wird das Stadtmodell intern in einen Quadtree unterteilt (siehe Kapitel 4.1.6). Anschaulich ergibt sich für ein einfaches Stadtmodell zum

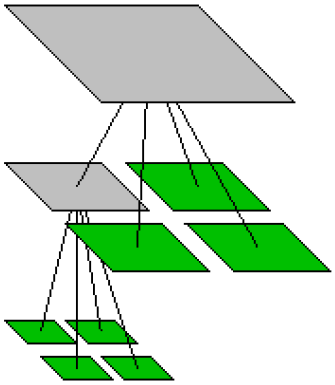

Abb. 26: Beispiel eines Quadtree

Beispiel die in Abb. 26 dargestellte Struktur. Die grauen Vierecke symbolisieren Knoten, die ihrerseits wieder Knoten enthalten, die grünen stellen Blätter des Baumes dar. Sie enthalten lediglich Häuser.

In jedem Knoten, also sowohl grauen als auch grünen, wird zusätzlich eine Polygonmenge gespeichert, die gegebenenfalls den Inhalt des Knoten ersetzen kann. Dies erfolgt, wenn der gesamte Inhalt des Knotens eine gewisse Größe auf dem Bildschirm nicht überschreitet (siehe Kapitel 4.1.6.5).

Bei der Polygonmenge handelt es sich gewissermaßen um den minimalen achsenparallelen Quader, der den gesamten Inhalt enthält. Optisch wirkt diese „Bounding Box“ allerdings nicht gut, da sich recht schnell eine geschlossene Fläche ergibt, wenn benachbarte Knoten vereinfacht dargestellt werden. Zudem ergibt sich schnell ein auffällig regelmäßiges Muster. Aus diesem Grund wird die „Box“ ein wenig verkleinert und die Ecken um einen zufällig bestimmten geringen Betrag verschoben. So wird die Regelmäßigkeit aufgehoben und es bleiben immer Zwischenräume, in denen das Terrain sichtbar ist.

Abb. 27 zeigt Beispiele der sich in Geryon ergebenden Darstellung. Das Bild ganz links oben zeigt die gesamte Stadt von einem sehr weit entfernten Standpunkt. Wie man sieht, werden nicht die einzelnen Häuser gezeichnet. Sie sind durch grobe Blöcke ersetzt worden. Bewegt man sich näher heran, so werden die groben Blöcke nach und nach immer feiner unterteilt. Dieser Prozess ist in dem Bild rechts oben zu sehen. Erst wenn sich der Betrachterstandpunkt der Stadt weit genug genähert hat, werden die tatsächlichen Häuser gezeichnet, wie im unteren Bild zu sehen ist.

Die Grenze, ab welcher Entfernung die Ersatzdarstellung verwendet werden darf, wurde für diese Bilder erhöht. Normalerweise ist sie derart festgelegt, dass die Verwendung der Ersatzdarstellung im Allgemeinen nicht auffällt.

4.1.5.2 Übertragung der Daten

Um die Anzahl an AGP-Bus-Transfers weiter zu reduzieren, emulieren wir Interleaved Vertex Arrays. Alle zu einem Eckpunkt gehörenden Daten, wie die Position, die Normale und die Texturkoordinaten, werden dabei im Speicher direkt hintereinander abgelegt. Leider können wir nicht auf die extra für diesen Zweck geschaffene OpenGL-Funktion `glInterleavedArrays` zurückgreifen, da diese nicht in der Lage ist, mit Multitexturing umzugehen. Stattdessen bietet OpenGL die Möglichkeit, bei `glVertexPointer`, `glNormalPointer` und `glTexCoordPointer` einen sogenannten `stride` anzugeben, der einen Offset zwischen zwei Elementen darstellt. Hiermit ist es möglich, alle verfügbaren

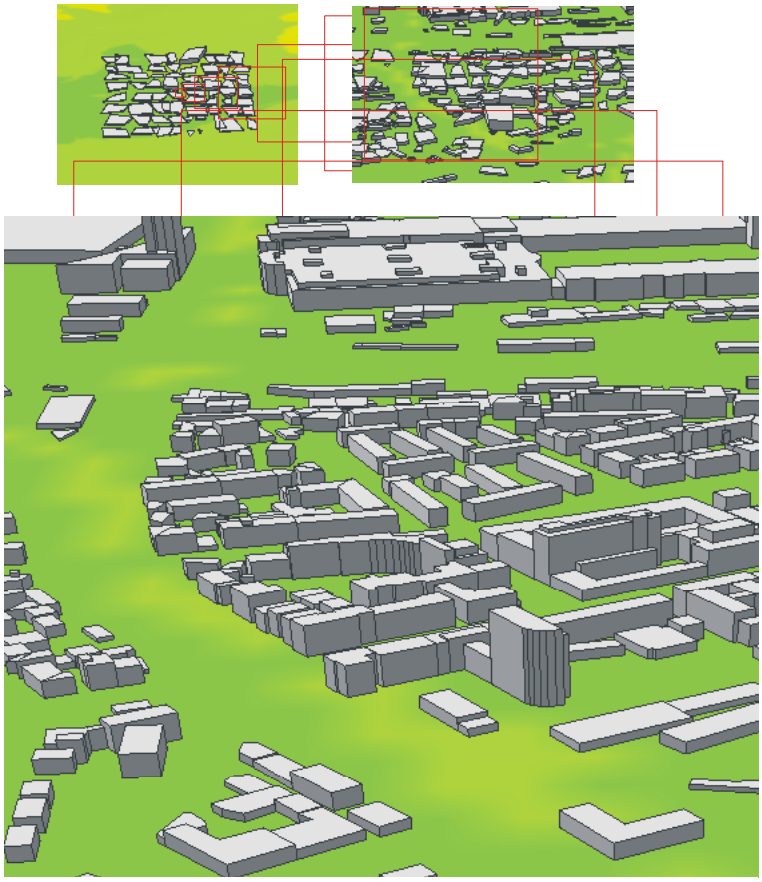

Abb. 27: Die Verfeinerung des Stadtmodells

Textureinheiten anzusprechen. In Abb. 25 ist ersichtlich, dass unsere Vorgehensweise keinen zusätzlichen Overhead erzeugt, eher das Gegenteil ist der Fall.

Aufgrund der Tatsache, dass alle für das Rendering notwendigen Daten nun direkt hintereinander im Speicher liegen, haben die Grafikkartentreiber wesentlich mehr Spielraum für die Optimierung der Übertragung über den AGP-Bus. Da wir auch weniger Speicherblöcke alloziieren, sinkt der Aufwand für die Speicherverwaltung und die Hauptspeicherfragmentierung wird reduziert. Das kann sich u.a. dann positiv auswirken, wenn Swapping notwendig wird, da dann semantisch zusammengehörige Daten auch zusammen ein- bzw. ausgelagert werden.

4.1.5.3 Lösung der Konflikte mit dem Terrain

Es muss sichergestellt werden, dass alle Häuser stets über dem Gelände gezeichnet werden. Dazu setzen wir den Stencil-Buffer ein. Nachdem dieser mit Null initialisiert wurde, erfolgt das Rendering der Häuser. Die Stencilfunktion ist „always“, die z-Buffer-Funktion bleibt unberührt bei „less“. Mit der Stenciloperation „replace“ und dem Referenzwert Eins erzielen wir den Effekt, dass im Stencilbuffer genau dort eine Eins steht, wo sich im Framebuffer auch ein Haus befindet. Die z-Buffer-Funktion sorgt für eine korrekte Tiefensortierung.

Im nächsten Schritt aktivieren wir die Stencilfunktion „equal“ unter Verwendung des Referenzwertes Null. Die Stenciloperation „keep“ verändert den Stencilbuffer nicht mehr. Diese Einstellungen resultieren darin, dass an keiner Stelle das Terrain ein Haus überzeichnen kann.

Wir beobachteten, dass die uns vorliegenden Terrains meist sehr eben sind. Nur aus sehr flachen Blickwinkeln sind Berge und Täler im Gelände zu beobachten. Mit unseren

Einstellungen würden auch Häuser, die eigentlich *hinter* Bergen liegen, *vor* diesen erscheinen. Dieser Fall konnte aber von uns im Frankfurter Gebiet nicht beobachtet werden, so dass der visuelle Gewinn, den wir durch die Unterdrückung der durch Rundungseffekte erzeugte Artefakte erreichten, überwiegt.

4.1.6 Die Implementierung

Die Implementierung beginnt mit dem Einlesen der Häuserdaten aus einer Textdatei. Zusätzlich zum reinen Einlesen der Daten müssen Verarbeitungsschritte zu ihrer Aufbereitung erledigt werden. Erst danach kann das Stadtmodell gerendert werden. Da sich verschiedene, immer vorhandene Teile von Geryon auf die Existenz eines Stadtmodells verlassen, sind bei Austausch oder Zerstörung desselben einige Dinge zu beachten. Sie werden neben anderen Dingen im Kapitel 4.1.7 beschrieben.

Hier wird zunächst der Gesamtaufbau des Systems anhand eines Aufbaudiagramms in Abb. 28 und dessen Realisierung durch Klassen mittels eines UML-Diagramms in Abb. 29 dargestellt. Der Textdateiverarbeiter liest die einzelnen Zeichen der Datei ein und konstruiert daraus die Häuser des Stadtmodells. Der Code, welcher den Textdateiverarbeiter beschreibt, befindet sich hauptsächlich in der Klasse `CityModelReader`, die Konstruktion eines Hauses ist jedoch in dessen Konstruktor und der Methode `createRenderingData` beschrieben. Aus den Häusern konstruiert der Modellbauer das interne Modell, welches aus dem beschriebenen Quadtree besteht (siehe Kapitel 4.1.5). Diese Konstruktion ist auf die Methoden `divideHouses`, `buildQuadTree` und `addBoundingBoxHouse` verteilt, welche alle zur Klasse `CityModel` gehören. Der Stadtmodellrenderer letztendlich beschäftigt sich mit dem tatsächlichen Zeichnen des Stadtmodells. Dazu sendet er entsprechende Befehle an OpenGL. Er ist auf die Methoden `render` und `evaluateThing` der Klasse `CityModelPainterGL` verteilt. Zusätzlich existiert ein Renderer für einzelne Häuser, welcher durch die Methode `render` der Klasse `HousePainterGL` realisiert wird. Außerdem können sowohl ein einzelnes Haus als auch das gesamte Stadtmodell ohne direkte Verwendung von OpenGL in einfachere geometrische Formen zerlegt werden. Dies wird durch die Methode `simplify` der Klasse `HouseSimplifier` bzw. der Klasse `CityModelSimplifier` getan.

Die Zerlegung in Klassen ist größtenteils durch VRS und den LandExplorer vorgegeben.

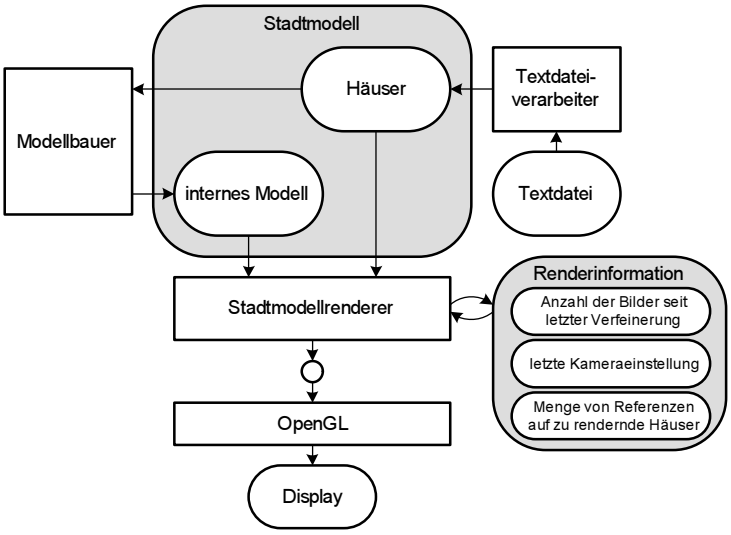

Abb. 28: Aufbau der Stadtmodellverarbeitung

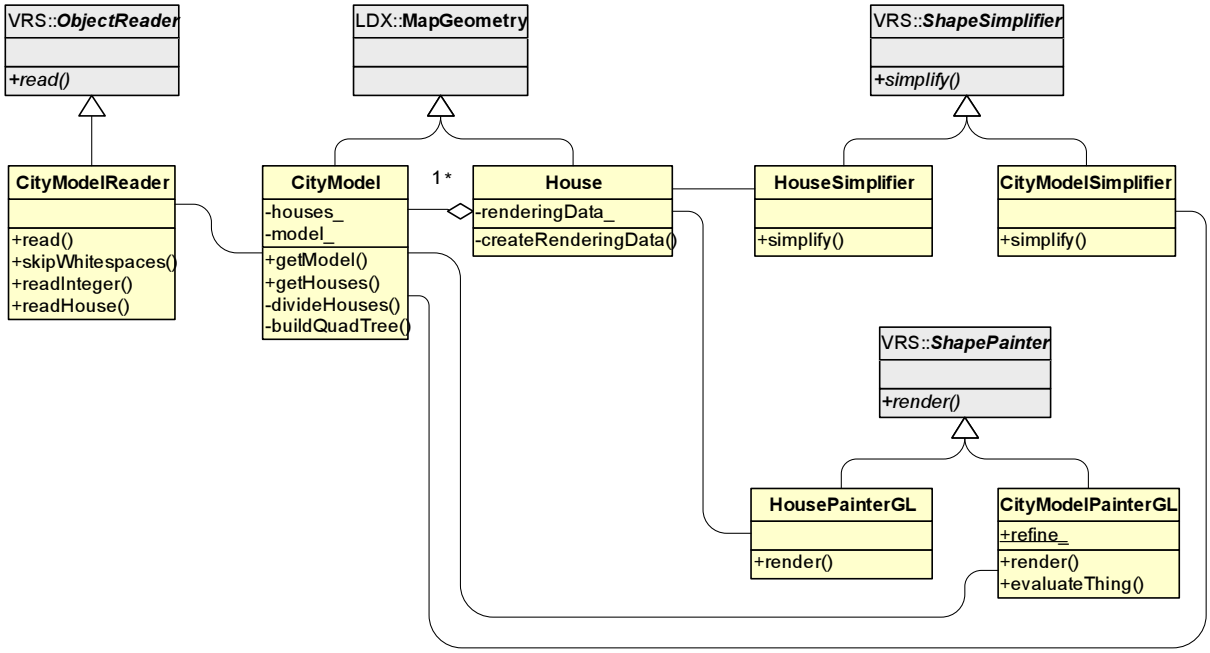

Abb. 29: Klassendiagramm der an der Stadtmodellverarbeitung beteiligten Klassen

Deutlich zu sehen ist die Trennung der Datenhaltung (`House`, `CityModel`) von der Verarbeitung für das Rendering (`HousePainterGL`, `HouseSimplifier`, `CityModel-PainterGL`, `CityModelSimplifier`), welche für VRS typisch ist.

4.1.6.1 Die Syntax

Die eingelesene Textdatei beginnt mit einem Eintrag für die Anzahl der Gebäude in der Datei, danach folgt die Beschreibung des ersten. Diese wiederum beginnt mit einem Eintrag für die Anzahl seiner Eckpunkte. Als nächstes werden entsprechend viele Eckpunkte durch jeweils drei Werte spezifiziert. Diese sind Längen- und Breitengrad in Millisekunden und die Höhe über NN in Zentimetern (in genau dieser Reihenfolge).

Darauf folgt die Anzahl der Bodenpolygone mit deren Beschreibung. Sie besteht immer aus der Angabe der Anzahl der Eckpunkte des Polygons gefolgt von entsprechend vielen Indizes. Auf dieselbe Art und Weise werden dann die Polygone der Wände und des Daches beschrieben.

Zusätzlich zur beschriebenen Struktur können Kommentarzeilen an beliebigen Stellen eingefügt werden. Im Anhang befindet sich eine Spezifikation der Syntax in Form einer Grammatik, die unser Reader einlesen kann (siehe Kapitel 8.1).

4.1.6.2 Das Einlesen

Die Eingabedatei muss nun Zeichen für Zeichen eingelesen und interpretiert werden. Das übernimmt ein Objekt der Klasse `CityModelReader`. VRS bietet ein allgemeines Konzept für das Einlesen von Dateien an. Um die Integration zu verbessern, fügt sich unser Reader in dieses Konzept ein, wie das UML-Diagramm zeigt. Es wird allerdings in Geryon nicht benutzt, da in VRS schon ein anderer Reader für den *bld*-Dateityp existiert. Um Schwierigkeiten vorzubeugen, die durch den Vorzug dieses Readers entstehen würden, wird direkt ein Objekt unserer Reader-Klasse erzeugt und verwendet.

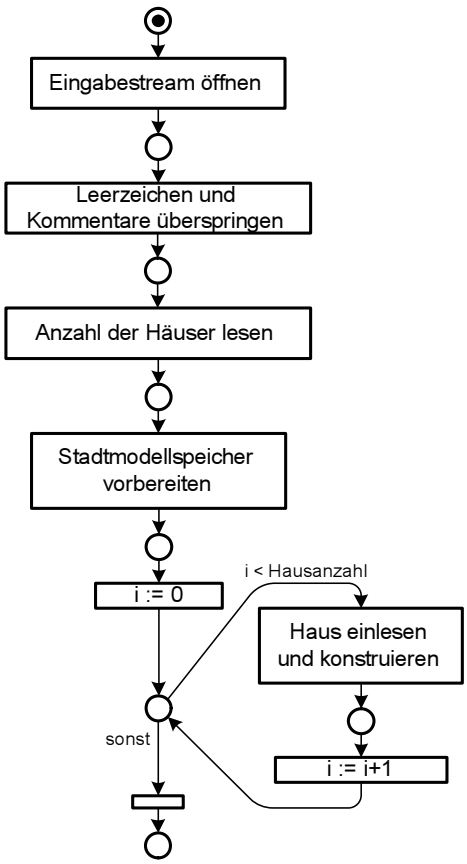

Abb. 30: Einlesen des Stadtmodells

Der grobe Ablauf wird in Abb. 30 veranschaulicht. Die Verfeinerung orientiert sich sehr stark an der Syntax der Textdatei und kann den Bildern im Anhang entnommen werden.

4.1.6.3 Die Haus-Konstruktion

Bei der Konstruktion eines Hauses wird die der Textdatei entnommene indizierte Darstellung verwendet. Diese wandelt der Textdateiverarbeiter zunächst in eine nicht indizierte Darstellung um. Dabei werden für jeden Eckpunkt Texturkoordinaten und Normale berechnet. Die Normale wird für die Beleuchtung bei der Visualisierung benötigt, die Verwendung der Texturkoordinate wird im Kapitel 4.2 erklärt. Zum Schluss werden noch die Bounding Box berechnet und eine renderingspezifische Datenstruktur erzeugt.

Alle Datenformate dieser Datenstruktur sind das Ergebnis einer sorgfältigen Abwägung zwischen zwei Faktoren: geringen Speicherbedarf und hohe Verarbeitungsgeschwindigkeit. Normalerweise kollidieren diese beiden Interessen, im Einzelfall kann sich aber gerade ein geringer Speicherbedarf positiv auf die Datenübertragung und damit die Gesamt-Performance auswirken.

In der Darstellung sind kleine Artefakte durch Rundungsfehler tolerierbar, solange keine stark sichtbaren Sprünge auftreten. Die Ersetzung des 64-Bit-Datentyps `double` durch mit 32 Bits dargestellte `floats` hat das visuelle Ergebnis in keiner Weise negativ beeinträchtigt, reduziert aber den Speicherbedarf um 50%.

Interessant ist die von Deering propagierte Normalen-Kompression[23]: Er stellte fest, dass mit 32-bittigen `float` 2^{96} unterschiedliche Normalen möglich sind, die sich gleichmäßig auf einer Einheitskugel verteilen. Der Abstand zwischen zwei Normalen beträgt damit nur noch 2^{-46} Radiant. Eine derart hohe Genauigkeit ist für die Beleuchtungsberechnung absolut über-

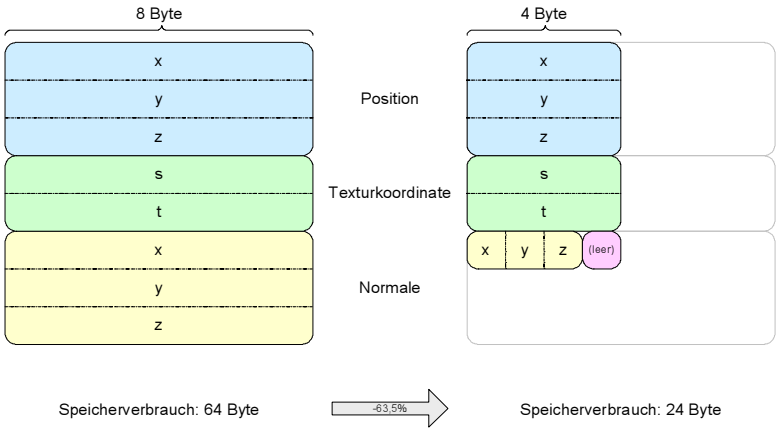

Abb. 31: Optimiertes Speicherlayout

flüssig, hier kann nach Deerings Angaben der Mensch ohnehin nur ca. 100.000 Normalen optisch voneinander unterscheiden. Wir setzen daher statt `float` auf den 8-bit-Datentyp `signed char`, was uns immerhin noch mehr als 16,7 Mio. Normalen erlaubt.

Statt ursprünglich 64 Bytes kann ein einzelner Eckpunkt in Geryon jetzt mit nur 23 Bytes gespeichert werden. Mit Rücksicht auf einen optimalen Speicherzugriff durch moderne Prozessoren füllen wir die Datenstruktur mit einem leeren Dummy-Byte auf 24 Bytes auf, damit die Eckpunkte stets an einer durch vier teilbaren Speicheradresse beginnen. Insgesamt kommen wir aufgrund der genannten Techniken mit nur 37,5 % des ursprünglichen Speicherbedarfes (siehe Abb. 31) aus.

4.1.6.4 Die Verarbeitung im Stadtmodell

Nach Abschluss des Einlesens aller Häuser kann das interne Modell berechnet werden. Schon während des Hinzufügens der einzelnen Häuser wird die Höhe jedes Hauses zu einer Gesamtsumme addiert. Dies befähigt das Stadtmodell später, die durchschnittliche Gebäudehöhe sehr einfach zu berechnen. Die Konstruktion des internen Modells ist an eine vorhandene Basisstation gebunden und wird erst durchgeführt, wenn das Modell das erstemal abgefragt wird. Sollten sich die Position der Basisstation oder andere relevante Parameter ändern, so muss das Modell neu konstruiert werden.

Das Ablaufdiagramm in Abb. 32 veranschaulicht vorbereitende Schritte für den Aufbau des Quadtrees, wie weiter unten ersichtlich wird. Die gesamte Häusermenge wird zunächst in drei Mengen aufgeteilt. Die erste Menge enthält diejenigen Häuser, die im Bereich für eine exakte Berechnung der Strahlungsausbreitung mittels des Walfish-Ikegami-Modells liegen. Die zweite enthält solche, die für eine Berechnung mit Standardparametern benötigt werden und die dritte Menge enthält den Rest.

Jede dieser Mengen wird dann in der in Abb. 33 dargestellten Art und Weise durch einen Quadtree strukturiert. Dieser enthält letztendlich auf seine Blätter verteilt wieder dieselben Häuser. Zunächst wird der aktuell zu bearbeitende Knoten in einen definierten Zustand gebracht. Wenn weiter verfeinert werden muss, enthält der Knoten danach vier weitere Knoten für die Aufteilung der Häuser. Ein ggf. vorhandenes `FilterTag` gefolgt von der Ersatzdarstellung für den aktuellen Knoten wird entfernt. Schließlich wird die Position des ersten einzusortierenden Hauses festgestellt und gespeichert.

Anschließend wird der Schwerpunkt aller einzusortierenden Häuser bestimmt, wenn es mehr als vier sind. Anhand dieses Schwerpunkts werden dann die Häuser auf die vier neuen Knoten verteilt und aus dem aktuellen Knoten entfernt. Dann wird rekursiv für jeden dieser

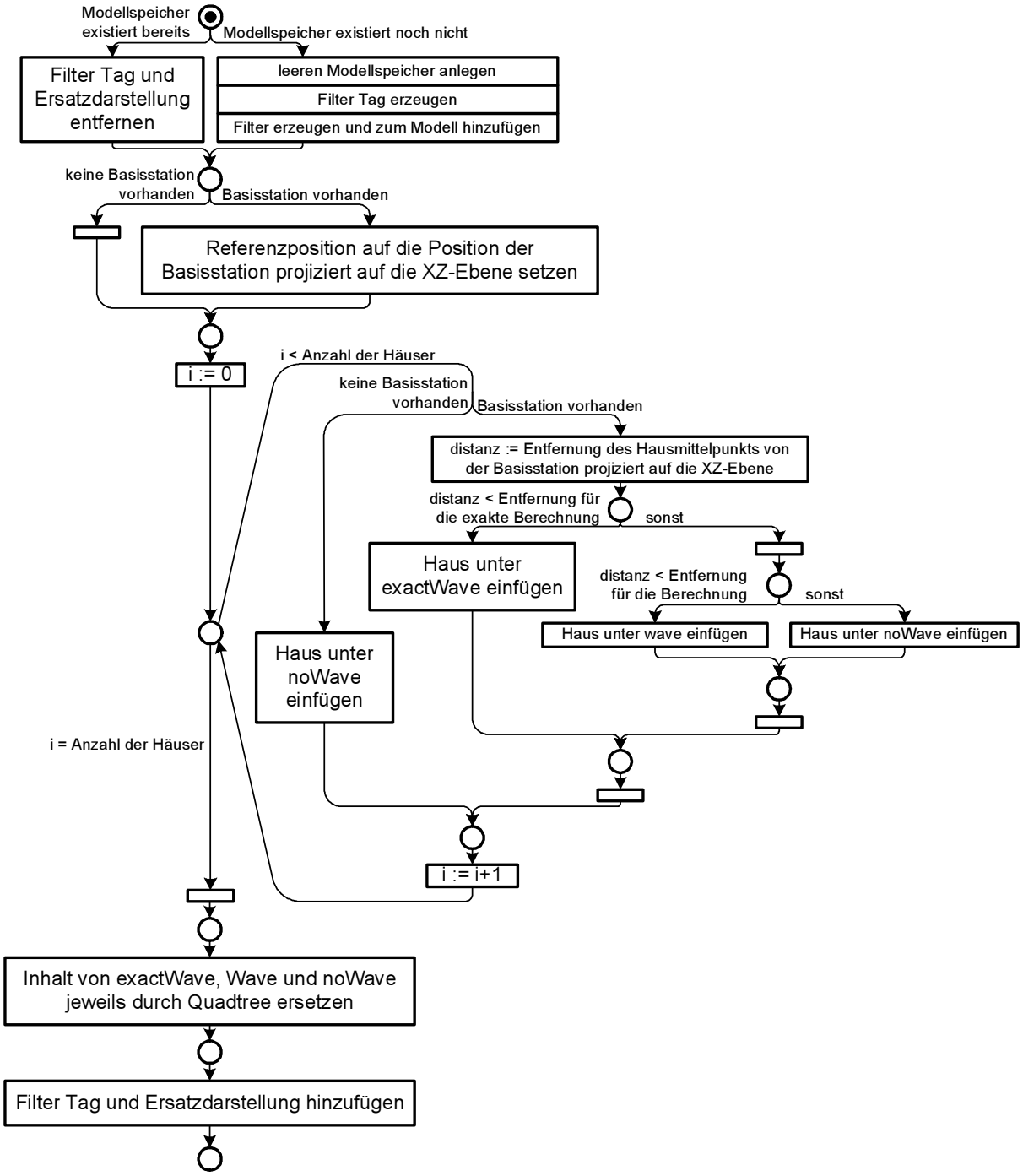

Abb. 32: Ablauf der Modellkonstruktion

neuen Knoten wiederum sein Inhalt durch einen Quadtree ersetzt. Zum Schluss wird noch das `FilterTag` und die Ersatzdarstellung hinzu gefügt.
Zum Ende der Knotenkonstruktion wird immer die im Kapitel 4.1.5 beschriebene Ersatzdarstellung in Form eines weiteren Hauses hinzu gefügt. Um die Verarbeitung des Quadtrees zu erleichtern, wird vor dieses Haus ein Filter-Tag gesetzt. Dieses Tag ist dasselbe, welches zuvor, wie im groben Ablauf gezeigt (siehe Abb. 32), erzeugt wurde. Zusammen mit dem Filter, der ganz zu Beginn in das Modell eingefügt wurde, bewirkt er, dass die zusätzlichen Häuser bei der Auswertung durch VRS ignoriert werden.

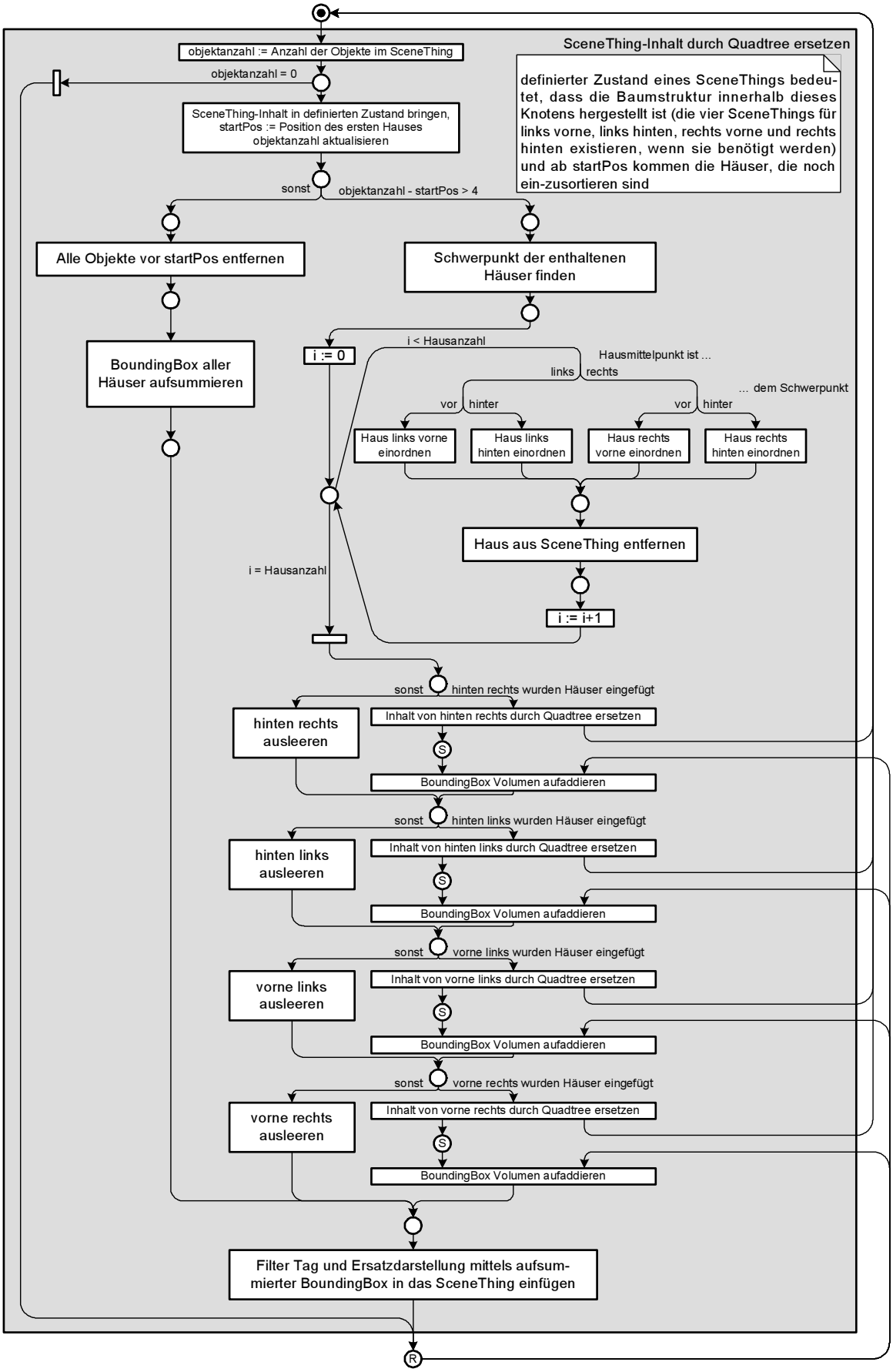

Abb. 33: Ablauf zur Konstruktion des Quadtree

4.1.6.5 Das Rendering

Die so gewonnene Repräsentation des Modells unterstützt den Rendering-Vorgang optimal. Der genaue Algorithmus ist in Abb. 34 dargestellt. Er ist in der `render`-Methode (bzw. `evaluateThing` und `boundingBoxVisibility`) der Klasse `CityModel-PainterGL` implementiert.

Zu Beginn wird, wenn bestimmte Bedingungen erfüllt sind, die Menge der zu rendernden Häuser bestimmt. Für die Überprüfung dieser Bedingungen sind einige Informationen notwendig, die das Rendering früher Bilder betreffen. Diese müssen also über den Aufruf von `render` hinaus aufbewahrt werden, was durch den Speicher Renderinformation geleistet wird (siehe Abb. 28). Um die zu rendernden Häuser zu ermitteln, wird der Wurzelknoten des internen Modells ausgewertet.

Bei der Auswertung eines Knoten wird sein Inhalt geprüft. Zuerst wird ggf. festgestellt, ob überhaupt etwas von diesem Inhalt sichtbar ist oder ob alle Elemente außerhalb des Kameravolumens liegen (Cullingtest). Danach wird die Ausdehnung des Knoteninhalts berechnet und mit der Entfernung zur Kamera verglichen. Je kleiner das Verhältnis ausfällt, desto kleiner ist der Inhalt auf dem Bildschirm. Wenn er eine gewisse Schwelle unterschreitet, bedeutet dies, dass der gesamte Inhalt ohnehin nur wenige Pixel einnimmt und es deswegen nicht notwendig ist, den Inhalt detailliert darzustellen. In diesem Fall wird die Ersatzdarstellung verwendet. Andernfalls wird der Inhalt des Knotens selbst ausgewertet.

Handelt es sich dabei um Häuser, so werden sie der zu rendernden Menge von Häusern hinzugefügt. Werden weitere Knoten des Baumes gefunden, so werden diese (rekursiv) ausgewertet. Mit Ausnahme von `Filter`-Objekten wird bei allen anderen Elementen des Szenengraphen die Auswertung des Inhalts abgebrochen. Dies basiert auf dem Wissen darum, wie das interne Modell des Stadtmodells aufgebaut ist. In diesem sind ein einziger `Filter`, Knoten, Häuser und `FilterTags` enthalten. Der genaue Aufbau des Modells ist im Kapitel 0 beschrieben. Hinter einem `FilterTag` befindet sich die Ersatzdarstellung eines Knotens, diese darf also bei der normalen Auswertung des Inhalts nicht berücksichtigt werden.

4.1.7 Besonderheiten der Implementierung

Die in der uns vorliegenden Textdatei enthaltenen Bodenpolygone sind von außerhalb des Hauses betrachtet immer rechts herum beschrieben. Daher wird die Punktreihenfolge beim Einlesen derselben vertauscht. Für die Darstellung in OpenGL ist es notwendig, dass für jedes Polygon die Eckpunkte von der Vorderseite betrachtet in einer einheitlichen Reihenfolge beschrieben werden. In unserem Fall ist das links herum.

Die Vereinfachungsrechnung benötigt die Bounding Box des Knotens, den sie gerade kontrollieren soll. Da unser Modell aus Exemplaren der VRS-Klasse `SceneThing` zusammen gesetzt ist und diese ihre Bounding Box nicht zwischenspeichern, wird der Einfachheit halber die Bounding Box der vereinfachten Darstellung verwendet. Außerdem wird aus Effizienzgründen auf die Wurzelbildung bei der Abstandsberechnung verzichtet. Dies fließt in den Vergleichsparameter für das Verhältnis zwischen Diagonale und Entfernung zum Betrachter ein und macht ihn relativ unverständlich.

Verschiedene Teile von Geryon verlassen sich auf eine bestimmte Struktur der internen Darstellung des Stadtmodells. Aus Effizienzgründen wird diese jedoch nicht geprüft. Insbesondere handelt es sich um die Strahlungsausbreitungsberechnung, das Rendering und die Berechnung des internen Modells selbst.

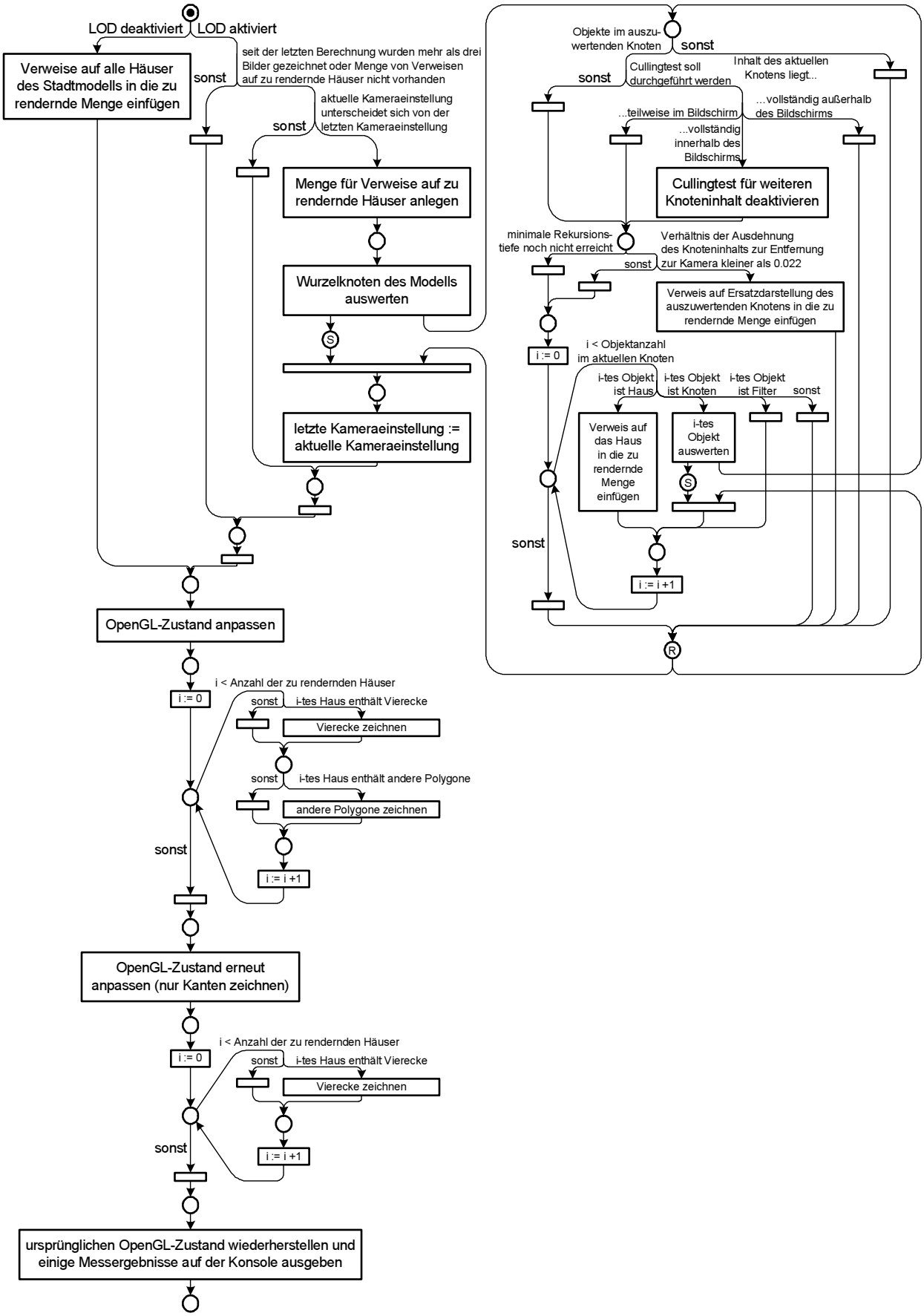

Abb. 34: Das Rendering des Stadtmodells

Geryon und der Landexplorer verwenden intern verschiedene Koordinatensysteme. In Geryon werden alle diesbezüglichen Angelegenheiten vom Singleton `Coordinate-Transformer` (siehe Kapitel 2.1) behandelt. Dieses Singleton muss genau einmal initialisiert werden. Es benötigt einen Referenzpunkt, welcher möglichst dicht an den anderen dargestellten Objekten liegen muss. Er legt den Ursprung des Meter-Koordinatensystems fest. Dieser Referenzpunkt wird beim Einlesen des Terrains auf den Mittelpunkt desselben gesetzt. Nach Möglichkeit sollte dieser Referenzpunkt während der Verwendung eines Stadtmodells nicht geändert werden.

Auch bei der persistenten Speicherung des Stadtmodells bzw. dem Einladen spielt dieser Referenzpunkt eine Rolle. Beim Einlesen und Erzeugen des Stadtmodells werden Punkte zwischen den unterschiedlichen Koordinatensystemen hin und her gerechnet. Diese Umrechnungen hängen von dem gesetzten Referenzpunkt ab. Wird das Stadtmodell über den VRS-Serialisierungsmechanismus eingelesen und der Referenzpunkt ist anders gesetzt, als bei der ursprünglichen Erzeugung, so sind die intern gespeicherten Punkte im Meter-Koordinatensystem nicht mehr gültig. Dies beeinflusst vor allem die Strahlungsberechnung, da sie im Meter-Koordinatensystem arbeitet. Das Rendering sollte nicht beeinflusst sein, es arbeitet mit Koordinaten im geographischen Koordinatensystem bzw. im visuellen Koordinatensystem.

4.2 Feldstärken

Die Darstellung der Strahlungsausbreitung bzw. der aus ihr resultierenden Feldstärke an einem bestimmten Ort ist der zentrale Punkt unseres Projekts. Für die Präsentation vor nicht-technischem Publikum oder Managern, aber auch für die Verbesserung der Planung ist die Unsichtbarkeit der Strahlung ein großes Problem. Eine möglichst real wirkende, intuitive Veranschaulichung ist wünschenswert.

Schon die Berechnung der Strahlungsausbreitung ist jedoch mit aktueller Hardware in akzeptabler Rechenzeit nur unter stark vereinfachenden Annahmen möglich. Auch die Darstellung stößt an diese Grenze, wenn sie in Echtzeit oder wenigstens interaktiv geschehen soll. Dieses Kapitel beschreibt sowohl die in Geryon erfolgenden Strahlungsberechnungen als auch die Visualisierung derselben.

4.2.1 Die Strahlungsberechnung

Bevor überhaupt etwas bezüglich des Mobilfunks dargestellt werden kann, ist es erforderlich, dass Daten vorliegen oder berechnet werden. Da uns lediglich die Antennencharakteristika vorliegen, müssen alle anderen Daten zur Visualisierung berechnet werden.

Dabei handelt es sich zum einen um die Abstrahlungsleistung der Basisstation in eine bestimmte Richtung. Diese ergibt sich aus der Summe der Abstrahlungsleistung der einzelnen Antennen der Basisstation.

Zum anderen muss die Feldstärke an einem konkreten Ort bestimmt werden. Diese hängt von der Abstrahlungsleistung in die entsprechende Richtung und der Entfernung zur Antenne ab. Dadurch allerdings, dass sich beliebige Hindernisse innerhalb der Fresnelzone befinden können, ist ihre Berechnung sehr komplex. Um Durchdringung, Beugung und Brechung zu berücksichtigen, greifen entsprechend komplexe Verfahren zum Beispiel auf Raytracing zurück. Dies ist für eine interaktive Visualisierung nicht möglich. Geryon verwendet hier das in Kapitel 2.3 beschriebene Walfish-Ikegami-Modell.

4.2.2 Visualisierungsgegenstände

Zunächst muss geklärt werden, was genau visualisiert werden soll. Bezogen auf die gesamte Funknetzplanung gibt es recht viele Sachverhalte, die dafür in Frage kommen. Neben der Feldstärke sind zum Beispiel die Aufteilung in Funkzellen, Nachbarschaftsbeziehungen, Interferenzen, Gesprächsabbrüche und die Ergebnisse von Messfahrten interessant.

Geryon, genauer gesagt GeryonEMUV, beschäftigt sich mit der Visualisierung einer einzelnen Basisstation. Damit bleiben nur noch die Feldstärken übrig.

4.2.3 Visualisierungsstrategien

Es bleibt die Frage, wie die Feldstärken dargestellt werden sollen. Da sie normalerweise nicht sichtbar sind, gibt es an sich keine aus der realen Welt stammende Vorgabe, wie eine Feldstärke aussehen soll.

Im 2D-Bereich wird sie normalerweise durch Farbe symbolisiert. Diese Möglichkeit bleibt natürlich für die dreidimensionale Darstellung erhalten. Es sind jedoch einige Schwierigkeiten damit verbunden. So entsteht der Eindruck der Dreidimensionalität hauptsächlich durch Beleuchtung. Die Farbe der dargestellten Gegenstände wird entsprechend ihrer Ausrichtung zur Lichtquelle moduliert. Dies ist bei gleichzeitiger Symbolisierung der Feldstärken durch eine bestimmte Farbe allerdings nicht mehr möglich. Die Farben würden dann verfälscht werden und nicht mehr mit der Legende übereinstimmen. Ohne die modulierten Farben erkennt jedoch der Betrachter die geometrische Struktur nicht mehr.

Weiterhin kann die Feldstärke in 2D durch Linien, die Orte gleicher Werte miteinander verbinden, veranschaulicht werden. Diese „Isolinien“ können ausgedehnt auf „Isoflächen“ auch in 3D Verwendung finden. Allerdings treten auch hier Schwierigkeiten auf. Zum einen verdecken sich die Isoflächen gegenseitig, was bei Isolinien nicht auftritt. Zum anderen verdeckt aber auch eine einzelne Isofläche vollständig das Gelände, die Gebäude und sonstige dargestellte Objekte.

Als dritte Möglichkeit bietet sich die Darstellung der Feldstärke als Nebel an. Dabei würde die Dichte des Nebels entsprechend der Feldstärke variieren.

Das Geryon-Visiondocument[15] zeigt einige Beispiele für bereits vorhandene Visualisierungen.

4.2.4 Umsetzungsmöglichkeiten

Die Darstellung der Feldstärken über Farben ist technisch sehr einfach zu realisieren. Vorausgesetzt die Feldstärken liegen vor, braucht man lediglich eine Abbildung von Feldstärken auf Farben. Diese können einzeln für bestimmte Bereiche festgelegt werden oder es kann ein Farbverlauf für einen Feldstärkenbereich definiert werden. Im einfachsten Fall kann der Verlauf linear sein. Da die Feldstärken jedoch sehr stark variieren, zum Beispiel vom Milliwatt-Bereich bis in den Mikrowatt-Bereich hinab, bietet sich auch eine exponentielle Abbildung an.

Insbesondere im 3D-Bereich stellt sich allerdings die Frage, was man denn mit der entsprechenden Farbe darstellen soll. Dafür zusätzliche geometrische Objekte in die Darstellung einzufügen ist wenig zweckmäßig. Sie verdecken viel zu stark die eigentlich darzustellenden Elemente wie Häuser beispielsweise. Würde man die zusätzlichen Elemente halbtransparent machen, so träte eine viel zu starke Verfälschung der Farbe auf, ähnlich wie bei der bereits erwähnten Modulation mit den Beleuchtungsergebnissen. Die Farbe muss also mit den vorhandenen geometrischen Objekten auf den Bildschirm gelangen. Um dabei noch

einigermaßen gut ihre Struktur erkennen zu können, können die Kanten durch Linien verstärkt bzw. überhaupt erst sichtbar gemacht werden.

Um das Problem der Verdeckung bei der Wahl von Isoflächen zu lösen, kann man die Isoflächen halbtransparent gestalten. Dies ist hier ohne weiteres möglich, da die Farbe der Isofläche zweitrangig ist. Sie symbolisiert die Feldstärke durch ihre geometrische Position. Dabei können lediglich eine Isofläche oder auch mehrere gleichzeitig angezeigt werden. Der Grad der Transparenz kann dabei gut als Einstellungsparameter dienen, um die Intensität der Darstellung im Vergleich zu den restlichen Gegenständen zu steuern.

Eine Visualisierung durch Nebel ist mit aktueller Hardware nicht in Echtzeit möglich. Sowohl die Berechnung der Feldstärken für ein dreidimensionales Raster mit akzeptabler Auflösung als auch die rendering-technische Umsetzung stoßen hier an ihre Grenzen.

4.2.5 Die Lösung in Geryon

Geryon benutzt sowohl Farben als auch Isoflächen. Die beiden Methoden werden dabei zur Darstellung von Feldstärken verwendet, die mit unterschiedlichen Berechnungsverfahren ermittelt werden.

Farblich dargestellt werden die Berechnungsergebnisse des Walfish-Ikegami-Modells. Eingefärbt wird dabei die Darstellung des Terrains und des Stadtmodells. Hauptsächlich durch die beiden Elemente entstehen in Geryon geometrische Objekte. Es wird dabei immer, beschränkt auf die begrenzte Auflösung der Berechnung allerdings, die Feldstärke für einen Punkt in anderthalb Metern Höhe über dem Boden berechnet. Die Höhe des Bodens ergibt sich aus dem Terrain.

Die geometrische Struktur des Terrains variiert im Bereich der Feldstärkenvisualisierung ohnehin recht wenig, daher verzichten wir auf eine Betonung derselben. Für die Gebäude jedoch ist sie von entscheidender Bedeutung, um überhaupt Gebäude erkennen zu können. Daher wird ihre geometrische Struktur durch Linien hervorgehoben. Diese wären prinzipiell nur im Bereich der Feldstärkenvisualisierung notwendig, heben aber unserer Meinung nach die Qualität der Darstellung auch außerhalb desselben.

Die Isoflächen stellen die idealisierte Feldstärke dar, die sich nur dann ergeben würde, wenn sich keinerlei Hindernisse zwischen Basisstation und Empfänger befinden würden. Die Position der Isoflächen ebenfalls nach dem Walfish-Ikegami-Modell zu berechnen wäre zu aufwendig. Wie oben beschrieben werden die Isoflächen halbtransparent dargestellt, um die restlichen Objekte nicht zu verdecken. Sie erhält dabei einen leicht bläulichen Schimmer und die Silhouette wird durch Linien hervorgehoben.

4.2.6 Die Implementierung

Um die Gründe für den Aufbau von Geryon bezüglich der Feldstärkenvisualisierung verstehen zu können, muss man zunächst die Rendering-spezifischen Anforderungen betrachten.

4.2.6.1 Die Farbdarstellung

Um die Feldstärken durch Farben darstellen zu können, müssen sie zum einen berechnet werden und zum anderen muss die entsprechende Farbe je nach Position auf die Geometrie aufgebracht werden.

Ersteres kann in der gewünschten Auflösung bei weitem nicht für jedes Bild in Echtzeit geschehen und es dauert auch als Vorberechnung zu lange. Daher geschieht es nebenläufig. Die Berechnung muss bei einer Änderung von Parametern der Basisstation, zum Beispiel ihrer Position oder ihrer Sendeleistung, und bei Änderungen am Stadtmodell neu angestoßen werden. Um nun parallel zur Berechnung ständig Bilder zeichnen zu können, muss also ein Speicher existieren, der von beiden Vorgängen verwendet wird. Da der Renderingvorgang ausschließlich lesend auf ihn zugreift, gibt es keine Schwierigkeiten mit der Synchronisation.

Es muss sichergestellt werden, dass bei jedem Bild, also insbesondere schon beim ersten, überall eine wenigstens halbwegs sinnvolle Farbe vorhanden ist. Daher wird zu Beginn und wenn die Berechnung neu angestoßen werden muss für ein sehr grobes Raster, konkret 32 x 32 Pixel, jeweils eine Farbe berechnet. Danach kann der Benutzer ganz normal navigieren und beliebige andere Aktionen durchführen. Im Hintergrund wird die berechnete Auflösung ständig verfeinert.

Daraus leitet sich mehr oder weniger ab, dass die Farbe nicht explizit für jeden Eckpunkt der Geometrie des Geländes und des Stadtmodells berechnet und angegeben werden kann. Dies wäre ohnehin nicht ohne tiefgreifende Änderungen am Quellcode für die Terraindarstellung möglich gewesen. Die Farbe wird für beide Geometrien über eine bzw. mehrere Texturen bestimmt.

Um eine sinnvolle Interpolation durch die Hardware zu erreichen, soll die Textur nach Möglichkeit dieselbe Auflösung wie die Berechnung besitzen. Daher werden mehrere Texturen für die Festlegung der Farbe verwendet, die sich gegenseitig überlagern. Maßgeblich ist dabei immer die Textur in der feinsten berechneten Auflösung. Für diesen Zweck existiert eignet sich der Decal-Modus hervorragend. Dabei wird immer die Farbe aus der Textur mit der bereits vorhandenen Fragmentfarbe gewichtet durch den Alphawert des Texel vermischt. Besitzt also die Textur an der relevanten Stelle den Alphawert 1, so ersetzt die Texturfarbe die Fragmentfarbe. Ist der Alphawert null, so bleibt die Texturfarbe ohne Wirkung. Wendet man also mehrere Texturen, in denen der Alphawert jedes Texel eins oder null ist, nacheinander an, so besitzt das Fragment die Farbe des zuletzt angewendeten Texel. Angewendet wird ein Texel nur, wenn der Alphawert ungleich null ist.

Initialisiert werden die „Feldstärketexturen“ mit dem Alphawert null. Jedes Mal, wenn die Farbe eines Texel berechnet wurde, wird sie eingetragen und der Alphawert dieses Texel wird auf eins gesetzt.

Es wird zunächst die Textur mit der gröbsten Auflösung angewandt, dann die nächst feinere und so weiter. So ist gewährleistet, dass immer das Ergebnis, mit der höchsten Auflösung, das bereits berechnet ist, zum Tragen kommt.

Abb. 35 verdeutlicht dieses Prinzip. Links sind die Texturen in ihrer unterschiedlichen Auflösung gezeigt. Kombiniert man die Texturen in der beschriebenen Art und Weise so ergibt sich eine Gesamttextur wie sie im mittleren Bild zu sehen ist. Im rechten Bild wurde vor der Kombination der Texturen interpoliert. Wie man sieht, fällt das optische Ergebnis

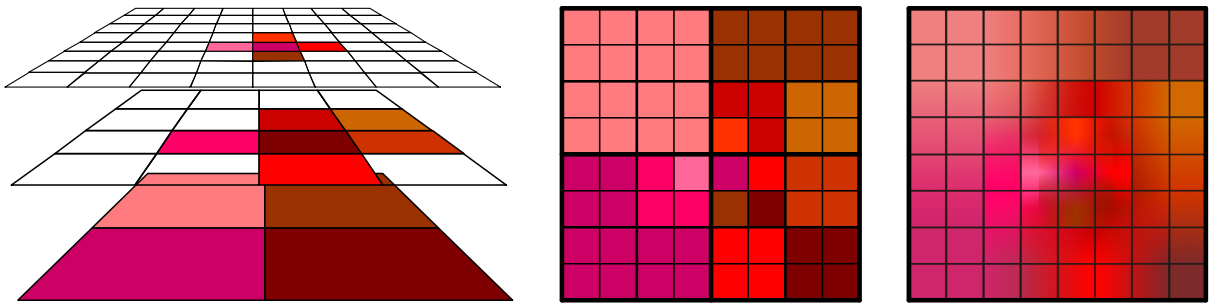

Abb. 35: Überlagerung der Feldstärketexturen

deutlich besser aus. Hier wurde allerdings nicht, wie es die Grafikhardware normalerweise tun würde, bilinear interpoliert sondern mit Hilfe eines anderen Verfahrens.

In Anlehnung an aktuelle Graphikhardware berechnet Geryon vier Texturen in jeweils verdoppelter Auflösung. Die Texturen werden auf das Gelände und auf die Stadt in unterschiedlicher Art und Weise aufgebracht. Für das Gelände verwendet Geryon den LDX-internen Mechanismus zum Multitexturing. Dieser verarbeitet den Szenengraphen notfalls mehrfach, um alle spezifizierten Texturen anwenden zu können.

Für das Stadtmodell wäre diese mehrfache Verarbeitung pro Bild zu teuer. Es werden bis zu vier Textureinheiten verwendet, um die Feldstärketexturen anzuzeigen. Wenn weniger vorhanden sind, werden die übrigen Texturen in höherer Auflösung weggelassen.

4.2.6.2 Die Isoflächen

Auch die Berechnung einer Isofläche wäre pro Bild wesentlich zu zeitintensiv. Glücklicherweise muss sie aber lediglich dann erfolgen, wenn sich bestimmte Parameter der Basisstation ändern, zum Beispiel die Sendeleistung. Folglich ist sie nur für das erste Bild notwendig und kann danach vorgehalten werden.

Die Isofläche wird mit einem bestimmten Tessellationsgrad, der VRS-intern verwaltet wird, berechnet. Der Tessellationsgrad bestimmt dabei sowohl die Anzahl der Punkte auf einer waagerechten als auch die auf einer senkrechten Schnittebene durch die Isofläche.

Ist die Isofläche berechnet, wird eine OpenGL-Displayliste angelegt, welche die entsprechenden Polygone enthält. Diese kann dann in den folgenden Bildern verwendet werden, solange bis die Isofläche neu berechnet werden muss.

4.2.6.3 Der Aufbau

Aus den in den Kapitel 4.2.6.1 und 4.2.6.2 beschriebenen Überlegungen folgt der in Abb. 36 dargestellte Aufbau. Der Feldstärkespeicher ist der erwähnte Speicher, der nebenläufig aktualisiert wird. Da das Schreiben in diesen Speicher von recht komplexer Natur ist, wird ein Akteur bereitgestellt, der dies übernimmt. Ein weiterer Akteur übernimmt die Kontrolle über die Berechnung der Feldstärken. Er kann die erwähnte erste Berechnung, die vor der Anzeige des ersten Bildes erfolgen muss, sowie die nebenläufige Berechnung durchführen. Diese kann begonnen, angehalten, fortgesetzt und neu gestartet werden. Zur Unterstützung der Nebenläufigkeit schafft der Feldstärkeberechner einen neuen Akteur, wenn die Berechnung angestoßen wird. Dieser Akteur wird wieder zerstört, wenn sie angehalten wird.

Der GUI-Verwalter kontrolliert die Feldstärkenberechnung, d. h. sie wird durch ihn angestoßen und ggf. wieder gestoppt. Er ist hier lediglich zum besseren Verständnis des Gesamtrahmens aufgeführt und für die Funktionsweise der Feldstärkenberechnung und -visualisierung eigentlich unerheblich.

Der VRS-Szenengraphauswerter arbeitet den gesamten Szenengraphen ab und stößt die entsprechenden registrierten Auswerter an, je nachdem, welche Objekte er vorfindet. Wie man sieht, ist die Funkkeule nicht direkt im Szenengraph sondern in einem Cache-Objekt enthalten. Der Szenengraphauswerter stößt nicht auf die Funkkeule sondern auf dieses Cache-Objekt. Der Auswerter des Cache-Objekts wiederum füllt bei der ersten Auswertung seinen Speicher mit Hilfe des Funkkeulenauswerters und erzeugt die Funkkeulenvisualisierung danach mit Hilfe dieses Speichers solange die Funkkeule unverändert bleibt. Das Anlegen und das Füllen dieses Speichers geschehen mit Hilfe von OpenGL. Prinzipiell lässt die Implementierung es auch zu, dass Funkkeulen direkt in den Szenengraphen eingefügt werden. Dies ist aber wenig sinnvoll, da die Berechnung der Funkkeule so viel Zeit in Anspruch nimmt, dass das Zeichnen eines einzelnen Bildes mehrere Sekunden dauern würde.

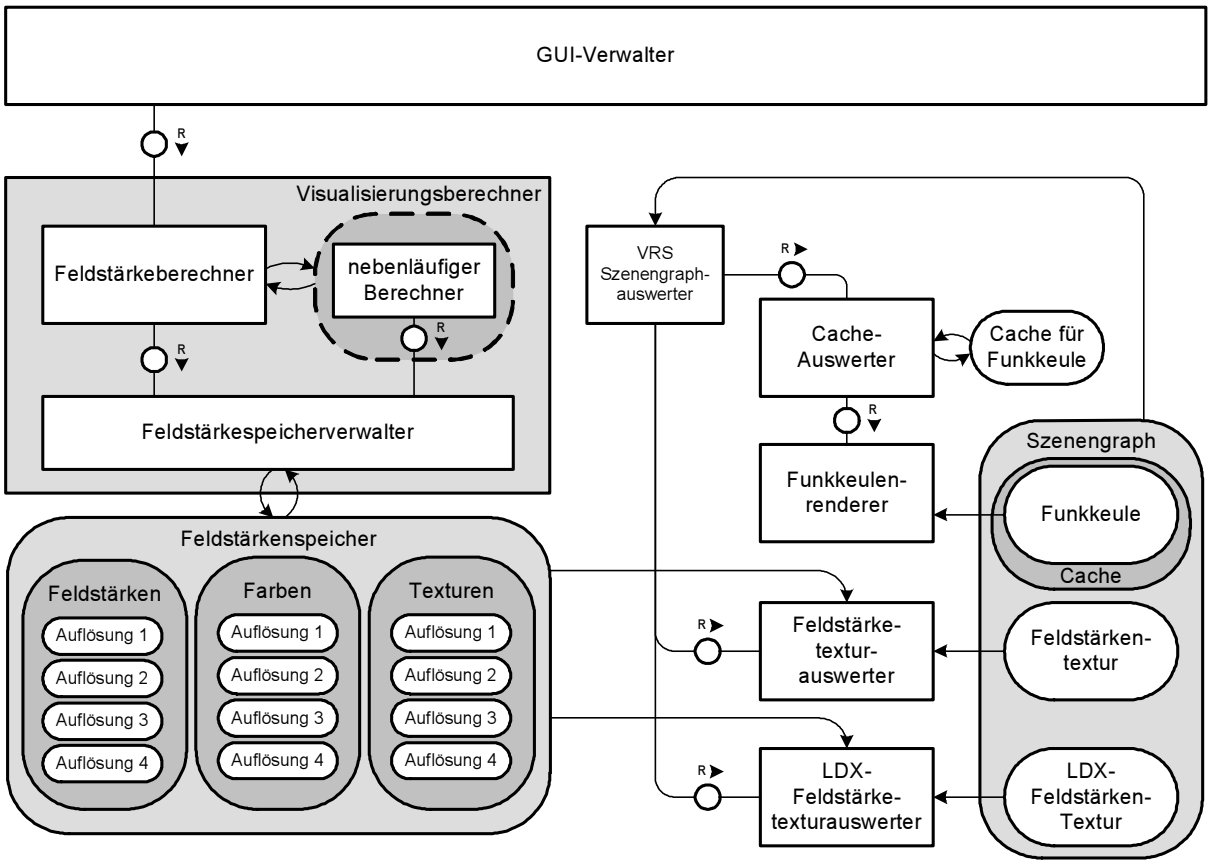

Abb. 36: Der Aufbau der Feldstärkenverarbeitung

In Abb. 37 ist die Umsetzung des Aufbaus durch Klassen veranschaulicht. Diese orientiert sich zum Teil an der durch VRS vorgegebenen Klassenstruktur, so zum Beispiel die Szenengraphelemente und ihre Auswerter. Der Feldstärkenspeicher und dessen Verwalter sind durch die Klasse `FSStorage` realisiert. Um die Farbberechnung leicht verändern zu können, ist der dazu notwenige Algorithmus in die Klassenhierarchie unter `ColorScheme` ausgelagert.

Der Feldstärkenberechner ist auf die Klassen `FSCalculator`, `WavePropagation`, `BaseStation`, `Antenna` und `RadioPattern` verteilt. Für die Berechnung ist der Zugriff auf viele verschiedene Attribute notwendig. Diese sind meistens nur innerhalb der entsprechenden Klasse direkt verfügbar. Der nebenläufige Berechner wird durch ein Exemplar der inneren Klasse `FSCThread` realisiert.

Die Klassen für die Realisierung des GUI-Verwalters und des VRS-Szenengraph-auswerters sind hier nicht aufgeführt.

4.2.6.4 Die Berechnung der Feldstärken

Beim Programmstart müssen der Feldstärkespeicher und der Berechnungsassistent initialisiert werden. Danach und wenn sich Parameter der Basisstation oder das Stadtmodell ändern, muss der Feldstärkespeicher geleert werden und der Berechnungsassistent die erste, grobe Berechnung durchführen. Dies stellt sicher, dass überall eine einigermaßen sinnvolle Farbe gesetzt ist und das erste Bild angezeigt werden kann. Der Berechnungsassistent startet dann die nebenläufige Verfeinerung und die berechneten Feldstärken fließen nach und nach in das dargestellte Bild ein. Dazu erzeugt er einen neuen Akteur, den nebenläufigen Berechner bzw. ein neues Exemplar der Klasse `FSCThread`. Dieser grobe Ablauf ist in Abb. 38 dargestellt.

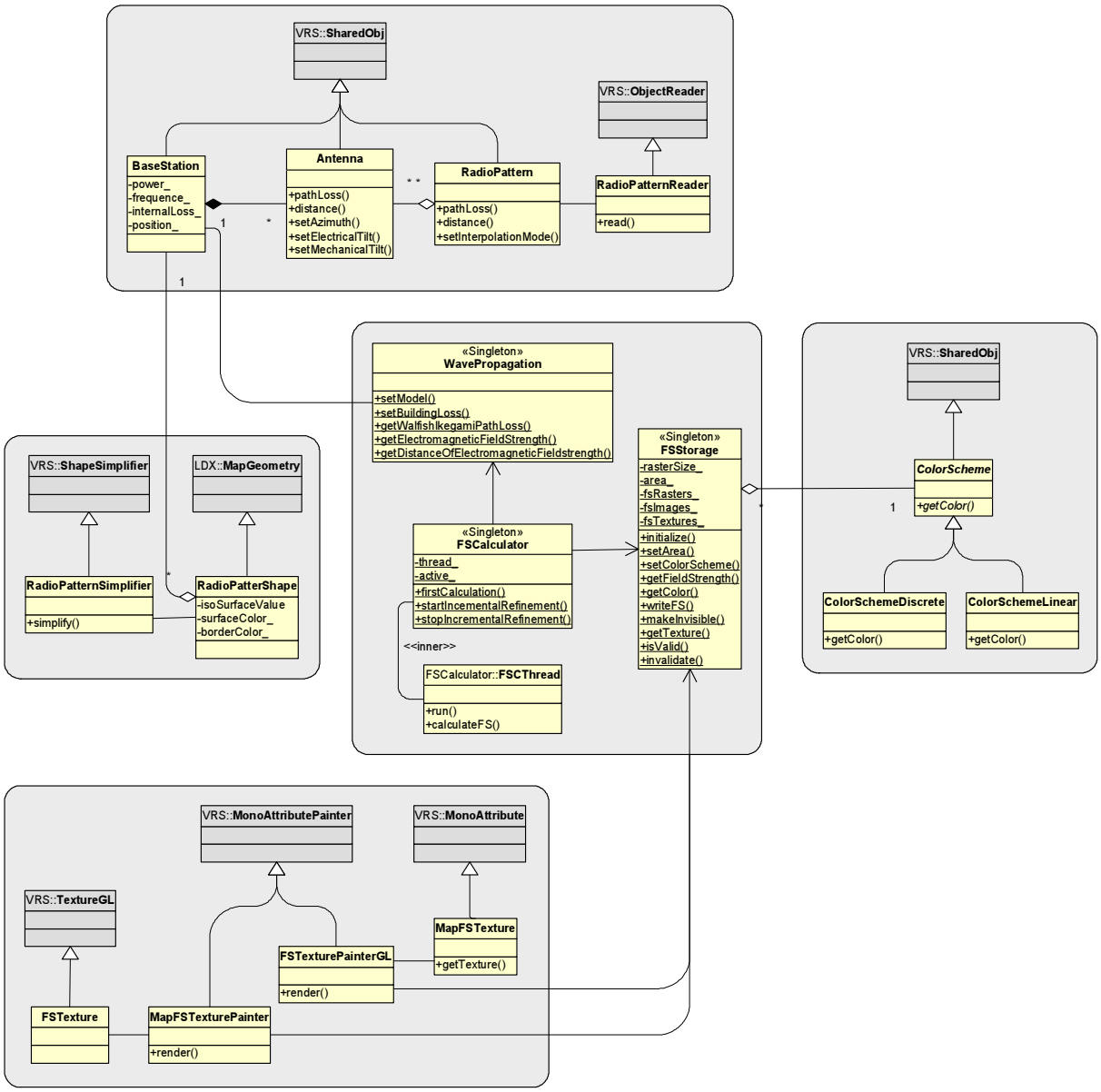

Abb. 37: An der Feldstärkenvisualisierung beteiligte Klassen

Das Gebiet, auf dem die Feldstärke dargestellt wird, ist oft wesentlich größer, als der aktuell dargestellte Bildausschnitt. Damit die verfeinernde Berechnung nicht ständig außerhalb des angezeigten Bildes stattfindet sondern dort, wo es von Interesse ist, kann man dem Berechnungsassistenten noch ein Gebiet „von besonderem Interesse“ bekannt geben. Für dieses Gebiet werden dann die Feldstärken zuerst berechnet. Dieser Parameter wird bei jedem Anstoßen oder Fortsetzen der nebenläufigen Berechnung übergeben.

Der Anstoß des gesamten Ablaufs geschieht durch den GUI-Verwalter wenn, bedingt durch die Aktionen des Nutzers, Feldstärken berechnet werden sollen bzw. alle Elemente vorhanden sind, die für eine Feldstärkenberechnung notwendig sind. Dies ist dann gegeben, wenn mindestens das Terrain geladen wurde und die Basisstation in selbigem platziert wurde. Die Abläufe im GUI-Verwalter sind im Kapitel 0 beschrieben.

Für die pixelweise Berechnung der Feldstärke werden je nach Entfernung zur Basisstation unterschiedliche Verfahren verwendet. Das Walfish-Ikegami-Modell wird nur für Pixel, die in einer gewissen Entfernung (d2, `setCalculateWavePropagationDistance`) liegen, angewendet. Für weiter entfernte Pixel wird automatisch angenommen, dass die ankommende

Feldstärke gleich 0 ist. Innerhalb dieser Distanz wird noch mal unterschieden. Für Pixel die weiter als eine zweite Entfernung (d1, `setExactCalculateWavePropagation-Distance`) entfernt liegen, wird das Modell nur mit Standardparametern angewandt. Nur für die verbleibenden Pixel werden exakt die durchschnittliche Gebäudehöhe, Straßenbreite usw. auf dem Weg von der Basisstation zum Pixel bestimmt.

4.2.6.5 Berechnung und Darstellung der Isofläche

Wie schon erwähnt wird die Isofläche nur einmal für das erste Bild berechnet und das Ergebnis dieser Berechnung dann gespeichert und bei den folgenden Bildern wiederverwendet. Diese Wiederverwendung basiert auf dem VRS-internen Caching-Mechanismus.

Für die Berechnung der Isofläche wird rund um die Antenne nach Punkten gesucht, an denen die Feldstärke der gewünschten entspricht. Dabei wird die Richtung von der Antenne aus festgelegt und nur nach der Entfernung gesucht. Da die meisten Antennen hauptsächlich in waagerechter Richtung ausstrahlen, wird die Genauigkeit der Abtastung in vertikaler Richtung nicht linear verteilt, sondern in waagerechter Richtung konzentriert. Da die Isofläche hier wesentlich weiter weg sein wird, bewirkt eine kleine Winkeländerung eine große Änderung der geometrischen Position des ermittelten Punktes. Damit würde die Auflösung des berechneten Polygonnetzes hier wesentlich verringert werden, wenn mit der selben Genauigkeit abgetastet werden würde. Der Algorithmus für die Abtastung ist direkt in der `render`-Methode der Klasse `RadioPatternPainterGL` enthalten.

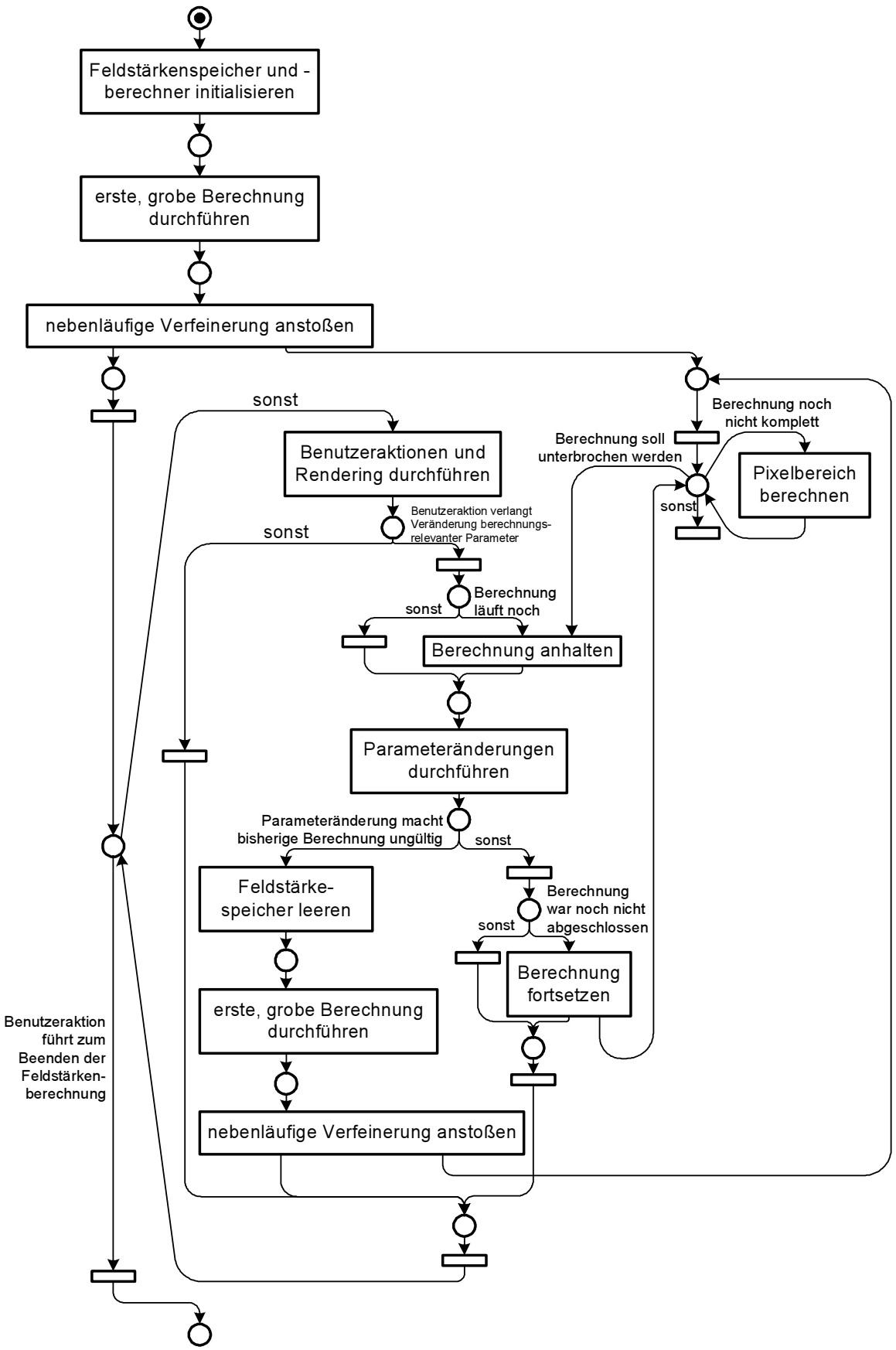

Abb. 38: Ablauf der Feldstärkenberechnung

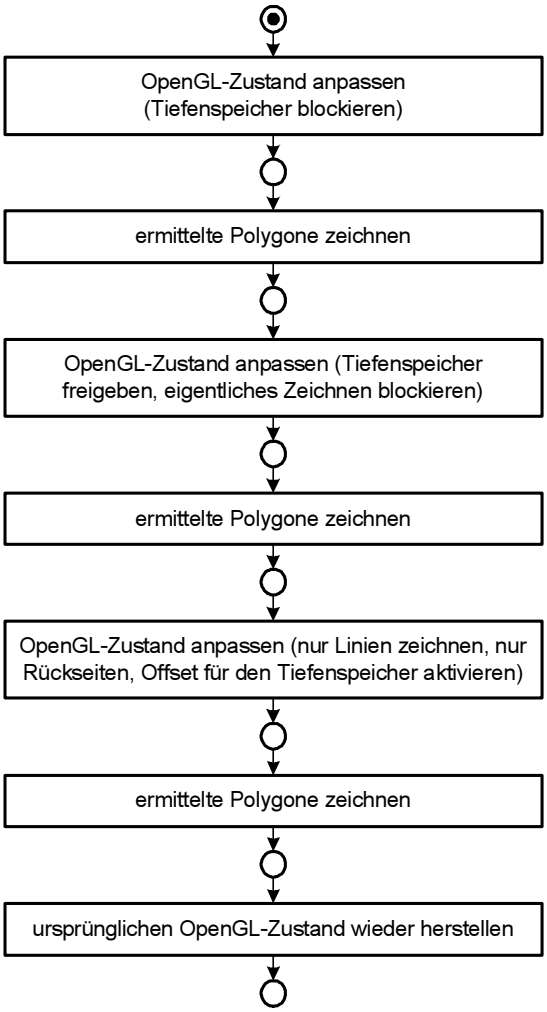

Abb. 39: Ablauf beim Rendering der Funkkeule

Die Entfernung des Punktes mit der gewünschten Feldstärke in gegebener Richtung wird über eine binäre Suche ermittelt. Gesucht wird zwischen 0 und 10.000 Metern mit einer Genauigkeit von etwa 0,1 Metern. Dieser Algorithmus ist in der Methode `getDistance` der Klasse `BaseStation` zu finden.

Alle gefundenen Punkte werden zu einem Polygon-Netz verbunden. Dieses Netz wird dann mehrfach im Rendering-Vorgang benötigt, wie Abb. 39 zeigt.

Die Isofläche selbst ist ein geschlossener Körper. Da sie aber halbtransparent, also quasi aus Glas, ist, sieht man nicht nur die Außenfläche sondern auch die normalerweise verdeckte Innenfläche. OpenGL arbeitet mit einem Tiefenspeicher für jeden Pixel, um festzustellen, welche Teile der Geometrie aktuell sichtbar sind. Es wird immer nur der vorderste Pixel gezeichnet, für transparente Gegenstände führt dies allerdings nicht immer zum gewünschten Ergebnis. Um nun zu verhindern, dass willkürlich in Abhängigkeit von der Zeichenreihenfolge bestimmte Teile der Innenfläche nicht gezeichnet werden, die eigentlich sichtbar sind, muss man zu einem kleinen Trick greifen. Geryon zeichnet die Isofläche zunächst einmal, wobei das Schreiben in diesen Tiefenspeicher blockiert wird. Die Innenfläche wird also auf jeden Fall komplett gezeichnet, wenn sie nicht durch sonstige Teile der Geometrie verdeckt ist. Danach wird die Isofläche noch mal gezeichnet, aber diesmal nur in den Tiefenspeicher, um auch hier sinnvolle Werte zu erhalten. Für die Betonung der Silhouette wird sie dann noch ein drittes Mal gezeichnet, wobei aber nur die Kanten der Polygone gezeichnet werden und auch nur diejenigen, die ganz am Rand liegen.

Der gezeigte Ablauf wird zusammen mit dem berechneten Polygon-Netz im Cache für die Funkkeule abgelegt. Dieser wird wie schon erwähnt von OpenGL selbst verwaltet. Bei nachfolgenden Bildern kann das Zeichnen der Funkkeule dann mit Hilfe dieses Speichers sehr viel effizienter erfolgen.

4.2.6.6 Die Auswertung der Texturen

Damit die berechneten Feldstärkewerte angezeigt werden können, liegen im `FSStorage` Texturen bereit. Diese Texturen enthalten Bilder, in denen die Feldstärkewerte übersetzt in Farben enthalten sind. Diese Übersetzung wird durch ein Exemplar einer Klasse, die von `ColorScheme` erbt, vorgenommen. Jedes Mal, wenn ein neuer Feldstärkewert in den `FSStorage` geschrieben wird, wird gleichzeitig die Farbe im Bild entsprechend gesetzt.

Die Texturen, die der `FSStorage` parat hält, setzen bei der Auswertung die Texturmatrix so, dass sie auf den vorgegebenen geometrischen Bereich wirken. Um die Texturen tatsächlich zu aktivieren, gibt es in Geryon zwei unterschiedliche Klassen, deren Exemplare in den Szenengraphen eingefügt werden können. Exemplare der Klasse `MapFSTexture` bewirken eine speziell auf das Terrain abgestimmte Auswertung der Texturen. Hier wird ein Exemplar der Klasse `MapDecalTexturingEffect` ausgewertet, welches die vier Feldstärketexturen enthält. Dies bewirkt, dass die Feldstärketexturen der Menge der Texturen, die auf die folgende Geometrie angewandt werden, hinzugefügt werden. Dabei wird als Texturmodus `GL_DECAL` gesetzt. Dies führt, da es sich alleine um vier Feldstärketexturen handelt, auch bei der Verwendung aktueller Hardware (GeForce4) zu mehrfachem Rendering der folgenden Geometrie, da in aller Regel wenigstens noch eine weitere Textur auf das Terrain aufgebracht werden soll.

Exemplare der Klasse `FSTexture` bewirken eine andere Form der Auswertung. Hier werden bis zu vier Textureinheiten mit Feldstärketexturen belegt. Der Modus wird dabei wiederum auf `GL_DECAL` gesetzt. Dies umgeht den Multitexturingmechnismus von VRS (siehe Kapitel 4.2.7). Dies ist speziell auf das Stadtmodell abgestimmt. Es verhindert, dass das Stadtmodell möglicherweise mehrfach gezeichnet wird. Das Rendering des Stadtmodells ist potentiell sehr aufwendig ist. Daher würde es wenig Sinn machen, es komplett mehrfach zu zeichnen, lediglich um die Feldstärken in verbesserter Auflösung darzustellen.

4.2.7 Besonderheiten der Implementierung

Für die Berechnung der Feldstärke für Pixel innerhalb der d1-Entfernung sind Parameter erforderlich, die von den konkreten Daten des Stadtmodells, der Position der Basisstation und der Position des Pixels abhängen. Daher muss für sie eine Strahlintersektion von der Basisstation zur Position des Pixels durchgeführt werden. Um dies effizient tun zu können, wird die Quadtree-Struktur des Stadtmodells genutzt. Dies kann nicht durch den VRS-internen Strahlintersektionsalgorithmus geschehen. In der Klasse `WavePropagation` wird ein eigener Algorithmus hierfür implementiert.

Bei der Auswertung der Feldstärketexturen für das Stadtmodell wird der VRS Multitexturingmechnismus umgangen. Die vorhandenen Textureinheiten, bis zu vier jedenfalls, werden explizit belegt. Im Kontext eines Subgraphen sind die Exemplare der Klasse `FSTexture` bei der Auswertung vollständig inkompatibel mit der Auswertung von Exemplaren der Klassen `TexturGL`, `MultiTexturingEffectGL` bzw. von ihnen ableitenden Klassen.

Um Artefakte bei der Darstellung zu vermeiden, müssen die Texturkoordinaten der Gebäude in einer bestimmten Art und Weise leicht von der geometrischen Position der

Eckpunkte abweichen. Um zu verhindern, dass eine Wand mit der Feldstärke, wie sie in dem zugehörigen Haus herrscht, eingefärbt wird, werden ihre Texturkoordinaten leicht nach außen verschoben. Dies ist, zumindest wenn die Texturauflösung fein genug ist, erfolgreich. Analog werden die Texturkoordinaten des Bodens und des Daches leicht nach innen verschobene, um zu verhindern, dass sie mit einer Feldstärke, wie sie außerhalb des Hauses herrscht, eingefärbt werden.

Daher können die Texturkoordinaten nicht automatisch generiert werden. Offensichtlich müssen auch für denselben Punkt unterschiedliche Texturkoordinaten angegeben werden, je nachdem, ob er gerade zu einer Wand, zu einer anderen Wand, zum Boden oder dem Dach gehört. Dies schließt eine indizierte Darstellung für das Rendering von vorn herein aus. Außerdem kann OpenGL Texturkoordinaten nicht replizieren, sie müssen für jede Textureinheit, die verwendet werden soll, explizit gesetzt werden. Dies geschieht beim Rendering des Stadtmodells. Bei der Anwendung der Feldstärketexturen auf das Terrain werden die Texturkoordinaten automatisch generiert.

5. Die Benutzerschnittstelle

Dieses Kapitel beschreibt die technischen Hintergründe der Benutzerschnittstelle zu Geryon. Hierbei liegen Schwerpunkte in der Architektur und wichtigen Abläufen, um das Verständnis der Quelltexte zu erleichtern.

5.1 Architekturgrundlagen

Die Architektur von Geryon ist durch zwei Bibliotheken geprägt: VRS und Qt. Beide bieten sowohl Framework-Funktionen wie Eventhandling als auch Programmiererleichterungen wie eine halbautomatische Garbagecollection an. Geryon hat damit die Aufgabe, zwischen beiden zu vermitteln aber auch zu trennen.

Neben der Vermittlerfunktion zwischen den Frameworks implementiert Geryon auch die Funktionalität der Benutzerschnittstelle. Es verarbeitet dazu sowohl direkte Manipulation in der 3D-Umgebung als auch Eingaben über klassische Oberflächenelemente, sogenannte Widgets.

Zusätzlich ist Geryon auch für die Datenhaltung zuständig. In Anbetracht der Größe einzelner Daten wie Terrains, Texturen oder Stadtmodelle, welche leicht die 100-MByte-Grenze sprengen können, werden sie in einen veränderbaren Teil, das sogenannte Szenario, und einen unveränderbaren Teil, die eigentlichen Rohdaten, unterteilt. Im Szenario werden einzig Verweise auf die entsprechenden Rohdaten abgelegt, um so Festplattenkapazität zu sparen.

5.1.1 Grobdarstellung

Abb. 40 zeigt den Aufbau von Geryon mit seinen zentralen Akteuren und Daten. Schaltzentrale ist Geryon, das alle weiß dargestellten Akteure und Daten verwaltet. Das Qt-Framework, das VRS/LDX-Framework und die Toolbox sind externe Komponenten. Das Qt-Framework stellt dabei die abstrahierte Verbindung zum Betriebssystem her, um Plattformunabhängigkeit zu gewährleisten. Das VRS/LDX-Framework liefert die nötige Funktionalität zur Darstellung und Animation der simulierten 3D-Umgebung. Die Toolbox,

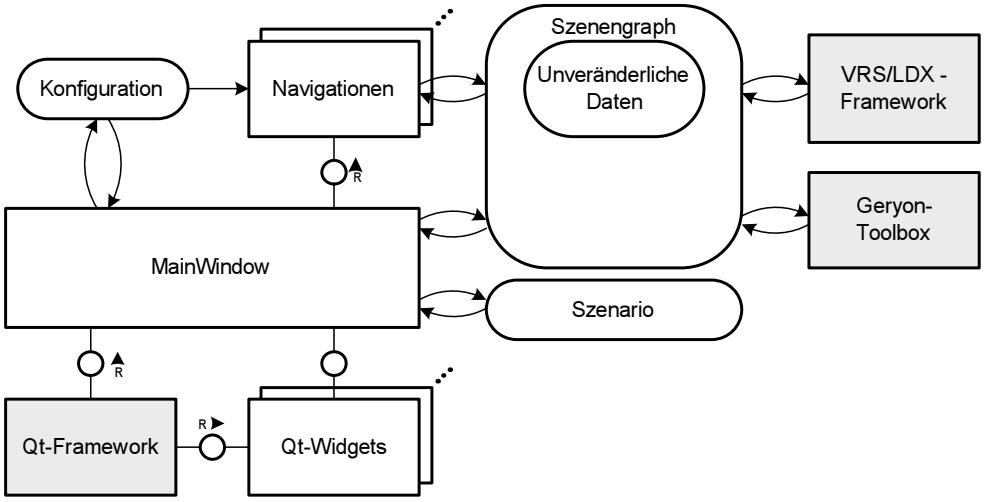

Abb. 40: Grober Aufbau

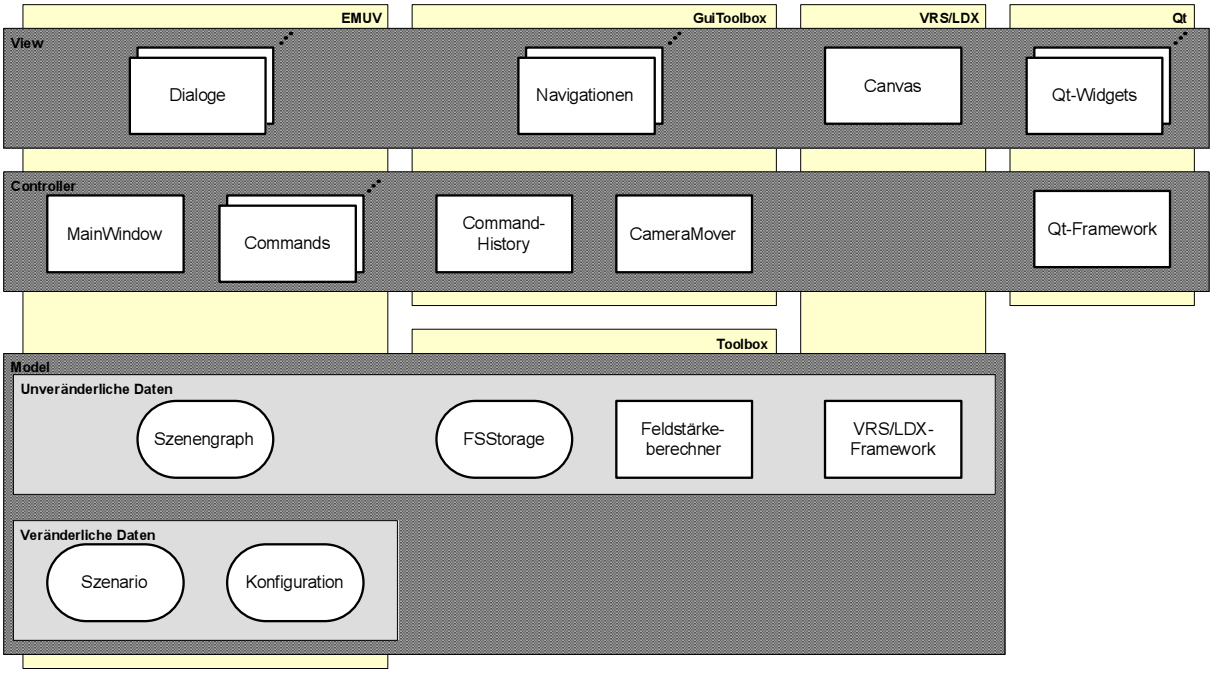

Abb. 41: Zuordnung der Akteure

die auch ein erheblicher Teil des Bachelorprojektes ist, sorgt für die Berechnung und Anzeige der Auswirkungen elektromagnetischer Strahlung auf das Gelände.

5.1.2 Einordnung in das MVC-Modell

Für Applikationen mit Benutzerschnittstellen ist das MVC-Modell das am häufigsten genutzte Architektur-Paradigma. Auch Geryon lässt sich in die drei Schichten Model, View und Controller einteilen. Abb. 41 zeigt die einzelnen Komponenten und ihre Einordnung in das MVC-Modell sowie die Zugehörigkeit zu den verschiedenen Bibliotheken des Quelltextes.

Qt bietet mit seinem Signal-Slot-Mechanismus eine einfache Möglichkeit, um die sonst häufig anzutreffende Verschränkung von View und Controller aufzuspalten. In diesem Framework senden View-Elemente Signale aus, die von Controller-Elementen empfangen werden. View-Elemente sind die Qt-Widgets, besonders hervorzuheben ist hier der QtCanvas des VRS-Frameworks sowie die Dialoge und Navigationen. In dieser Schicht finden sich dementsprechend, abgesehen von den EMUV-spezifischen Dialogen, wiederverwendbare Klassen.

Die Controller-Schicht wird klar von der Klasse `MainWindow` dominiert. Sie enthält den Großteil der Verarbeitungslogik von Geryon. Insbesondere für den Undo-Redo-Mechanismus (siehe Kapitel 5.3) wird auch Gebrauch von Commands gemacht, die jeweils eine Aufgabe kapseln und somit auch dem Controller zuzuordnen sind.

Die Model-Schicht teilt sich in 2 klar abgegrenzte Teile: veränderbare Daten, die durch eigene Klassen repäsentiert sind, und unveränderbare Daten, die in die Klasse `MainWindow` integriert sind. In den ersten Teil fallen das Szenario wie auch die Konfiguration, in den zweiten der Szenengraph und die darin enthaltenen Objekte. Dem unveränderbaren Teil des Models ist auch die Toolbox sowie das VRS-Framework zuzuordnen, da beide keinerlei Interaktionsanteile besitzen, sondern einzig von MainWindow kontrolliert werden.

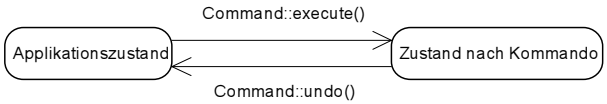

Abb. 42: Zustandsübergang mit Command

5.1.3 Genutzte Design Patterns

In Geryon kommen an verschiedenen Stellen Design Patterns zum Einsatz. Das Publisher-Subscriber- oder Observer-Pattern wird von den beiden Framesworks in verschiedenen Ausprägungen eingesetzt. Qt bezeichnet es als Signal-Slot-Mechanismus, der hauptsächlich zur bidirektionalen Nachrichtenübermittlung zwischen View und Controller dient und VRS als Callbacks, die bei Benachrichtigungen auf Grund von Zustandsänderungen von Objekten Einsatz finden. Beide Mechanismen erfüllen im Rahmen von Geryon den Zweck, View und Model mit dem einzigen Exemplar von MainWindow als Schaltzentrale synchron zu halten.

Das Singleton-Pattern ist ebenfalls in Geryon zu finden. Allerdings kommt es nur indirekt zum Einsatz, da die eigentlichen Singletons das Terrain aus der LDX-Bibliothek und die Feldstärkeberechnung aus der Toolbox-Bibliothek sind.

Das zentrale Pattern für die Funktion von Geryon ist das Command-Pattern. Dieses Pattern, das die Kapselung von Benutzeraktionen zum Ziel hat, dient der Realisierung der Undo- / Redo-Funktion, ohne die eine interaktive Applikation nicht mehr auskommt. Mehr Informationen zum Einsatz des Command-Patterns gibt Kapitel 5.3.

5.2 Signale und Callbacks

Wie bereits in Kapitel 5.1.3 ausgeführt, sind sowohl Qt-Signale als auch VRS-Callbacks Ausprägungen des Observers-Patterns.

Der Signal-Slot-Mechanismus bildet die Grundlage für die Verknüpfung einzelner Komponenten unter Qt. Er bietet die Möglichkeit, mittels in die Programmiersprache integrierten Mechanismen einfach eine anonyme Benachrichtigung zwischen Objekten zu implementieren, die nicht innerhalb des üblichen Rahmens der Objektorientierung liegt. Dabei ist sogar eine m:n-Beziehung realisierbar: ein Signal kann an beliebig viele Slots sowie ein Slot an beliebig viele Signale gebunden werden. Diese Bindungen sind zudem zur Laufzeit editierbar, was die Applikation flexibel macht. Der-Mechanismus kommt beispielsweise zum Einsatz, wenn ein Menüpunkt ausgewählt wurde. Dann wird durch das Menü ein Signal verschickt, auf das sich die MainWindow-Instanz beim Programmstart registriert hat. Der Empfang dieses Signals löst die entsprechende Funktion aus

Der Callback-Mechanismus dient auch zur anonymen Benachrichtigung fremder Objekte, die zur Compilierzeit nicht bekannt sein müssen. Ebenso wie Signale sind auch Callbacks nicht auf die Hierarchie der Objektorientierung festgelegt, erfordern jedoch mehr Aufwand vom Programmierer, da sie hauptsächlich über C++-Templates in VRS realisiert sind. In Geryon wird dieser Mechanismus hauptsächlich zur Benachrichtigung bei Objektänderungen eingesetzt. Dazu registriert sich die `MainWindow`-Instanz beispielsweise mittels eines Callbacks bei der Basisstation (vgl. Abb. 43), um bei Änderungen der Basisstation automatisch auch die aktuelle 3D-Ansicht zu aktualisieren..

5.3 Commands

Das Command-Pattern wird in interaktiven Anwendungen zur Kapselung von Benutzeraktionen eingesetzt. Ziel ist die Schaffung von atomaren Funktionsblöcken, die durch die festgelegte Funktionalität auch die Möglichkeit zur Reversibilität bieten, die Anwendung also von einem Zustand in einen anderen versetzen können und umgekehrt (siehe Abb. 42). Werden alle relevanten Benutzeraktionen durch Commands gekapselt, lässt sich eine Kette von Commands aufbauen, die eine schrittweise Wiederherstellung von vorherigen Zuständen erlaubt und dem Nutzer so zusätzliche Sicherheit gibt, da er Aktionen mit dem Wissen ausprobieren kann, dass im Falle unerwarteter Reaktionen Änderungen zurücknehmbar sind.

5.3.1 Umsetzung CommandHistory und Command-Mechanismus

In Geryon ist der Command-Mechanismus zweigeteilt. Wie Abb. 44 zeigt, ist die allgemeine, von Geryon unabhängige Funktionalität in das Paket `GuiToolbox` integriert. Das sind sowohl die Basisklasse für alle Commands, `Command`, als auch die Verwaltungsklasse für alle ausgeführten Commands, die `CommandHistory`. Im Paket `EMUV`, speziell sogar als private Subklasse von `MainWindow` sind die eigentlichen Commands für die Benutzeraktionen definiert. Die Abbildung zeigt beispielhaft `CmdChangeTexture`, mit dem das Laden einer neuen Textur gekapselt wird. Deren Attribute enthalten alle notwendigen Daten zum Ausführen der Aktion, d.h. den Dateinamen der neuen Textur, wie auch zum Zurücknehmen, d.h. den Dateinamen der alten Textur. Alternativ könnte dieses Command auch die bereits im Speicher befindliche Textur speichern. Da diese aber potentiell viel Speicher belegt, wurde darauf verzichtet.

Die `Command`-Klassen treffen gewisse Annahmen über den Zustand der Applikation. Die `execute`-Methode hat klassenspezifische Bedingungen für den Applikationszustand, die vom Erzeuger des Commands sichergestellt werden müssen. So kann beispielsweise `CmdMoveBaseStation` nur funktionieren, wenn auch eine Basisstation vorhanden ist. Die

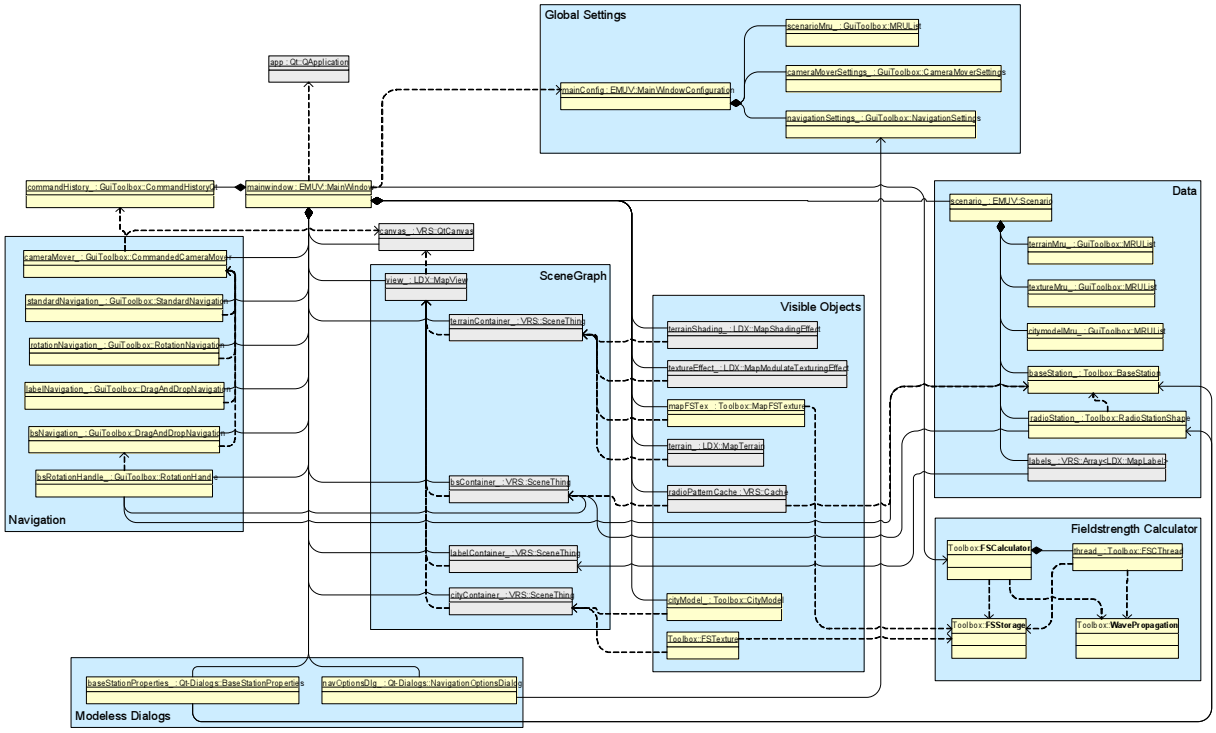

Abb. 43: Standardzustand von GeryonEMUV zur Laufzeit

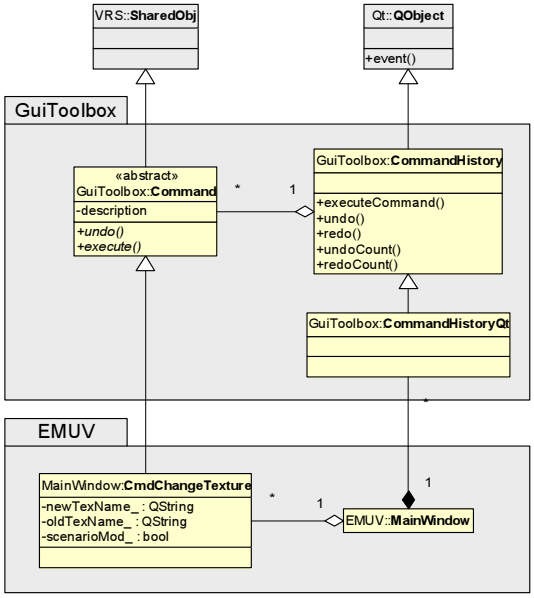

Abb. 44: Command-Mechanismus in Geryon

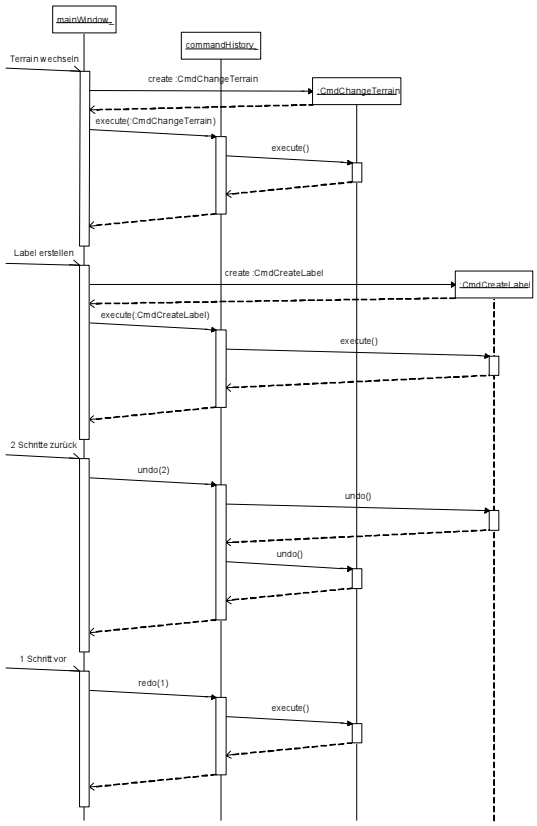

Abb. 45: Ablauf von Benutzeraktionen

`undo`-Methode jedoch hat eine einheitliche Bedingung: Sie darf nur aufgerufen werden, wenn sich die Applikation in genau dem Zustand befindet, in den sie durch `execute` überführt wurde. Für die Einhaltung dieser Annahme ist die Klasse `CommandHistory` verantwortlich. Dazu entzieht sie der Anwendung die Kontrolle über die Commands. Die Anwendung muss bei einer Benutzeraktion nur noch ein Exemplar der passenden `Command`-Klasse erzeugen und der `CommandHistory` übergeben, welche es sodann ausführt. Zusätzlich speichert sie die Commands in exakt der Reihenfolge, in der sie ausgeführt wurden. Ein Undo spezieller Commands ist nicht mehr möglich, nur eine Anzahl von Commands kann rückgängig gemacht werden. Damit kann die CommandHistory die Konsistenz des Anwendungszustandes sicherstellen, indem es die `undo`-Methoden der Commands in umgekehrter Reihenfolge aufruft (vgl. Abb. 45).

Die von `CommandHistory` abgeleitete Klasse `CommandHistoryQt` geht sogar soweit, dass sie der Anwendung nicht nur die Kontrolle über die Commands entzieht, sondern auch über den Zeitpunkt der `undo`- und `redo`-Aufrufe, da sie eigene Menüeinträge und Toolbarelemente definiert. Somit weiß die Anwendung nicht einmal, wann Commands rückgängig gemacht oder wiederholt werden.

Die Forderung nach Konsistenz des gesamten Anwendungszustandes ist oftmals zu restriktiv, da es damit erforderlich wäre, wirklich jeder möglichen Zustandsänderung ein Command zuzuordnen. Deshalb ist es sinnvoll, den Zustand in unabhängige Teile zu zerlegen. So ist die Kamerasteuerung in Geryon vom Command-Mechanismus abgekoppelt. Allerdings ist es damit nicht mehr möglich, über Commands Einfluß auf die Kamera zu nehmen, da ansonsten Inkonsistenzen zu befürchten sind. Ein Beispiel: Der Nutzer kann die Kamera mittels eines Kommandos an eine gespeicherte Stelle springen lassen. Das zugehörige Command speichert sich die Ursprungsposition, um sie für Undo wiederherstellen zu können. Bewegt der Nutzer nun nach dem Kamerasprung die Kamera (wobei kein Command erzeugt wird) und löst dann ein Undo aus, so springt die Kamera zu der Stelle, wo

Command	Funktion
`CmdRenameLabel`	Label umbenennen
`CmdMoveLabel`	Label verschieben
`CmdCreateLabel`	Label anlegen
`CmdDeleteLabel`	Label löschen
`CmdTransparentLabel`	Label-Transparenz umschalten (halbtransparent/opaque)
`CmdRotateBaseStation`	Basisstation drehen
`CmdMoveBaseStation`	Basisstation verschieben
`CmdChangeBaseStation`	Änderungen der Basisstation (in Eigenschaften) übernehmen
`CmdChangeTerrain`	Neues Terrain laden
`CmdChangeCityModel`	Neues Stadtmodell laden
`CmdChangeTexture`	Neue Textur laden
`CmdChangeColorScheme`	Neues Farbschema laden
`CmdSaveView`	Aktuelle Kameraposition speichern

Tabelle 5: Commands in Geryon

sie vor der Commandausführung war und nicht, wie eher zu erwarten wäre, zum Sprungziel, was die letzte Position der Kamera war.

5.3.2 Anwendung im Rahmen von Geryon

Geryon stellt Commands für alle Benutzeraktionen bereit, die Daten des Szenarios verändern. Sie sind in Tabelle 5 aufgeführt.

5.4 Daten in GeryonEMUV

Geryon unterscheidet nach veränderbaren und unveränderbaren Daten. Veränderbar sind alle Informationen, die durch unsere Applikation bearbeitet werden können. Unveränderbare Daten sind alle zur Laufzeit zur Darstellung des Szenarios notwendigen Daten, die jedoch aus anderen Quellen stammen. Hauptgrund für diese Unterscheidung sind das Gelände, die Texturen und Stadtmodelle. Da diese von Geryon nicht editiert werden und beträchtliche Größe erreichen können, ist jeweils nur ein Verweis, d.h. der Dateiname der Daten, im Szenario enthalten. Somit ist aber beim Laden eines Szenarios nicht nur die Szenario-Datei erforderlich, sondern auch alle zusätzlichen Dateien, auf die verwiesen wird.

5.4.1 Veränderbare Daten

Die veränderbaren Daten werden komplett von der Klasse `Scenario` gekapselt, die sich auch um das Laden und Speichern kümmert. Sie sind in Tabelle 6 aufgeführt.

Art	Bedeutung
Terrainname	Datei, die Geländedaten enthält (abs. Pfad)
Texturname	Datei, die Texturdaten enthält (abs. Pfad)
Stadtmodellname	Datei, die das Stadtmodell enthält (abs. Pfad)
Basisstationsparameter	Beschreibung der Basisstation, besteht aus mehreren Elementen
Stationsdarstellung	Art der Stationsdarstellung im der 3D-Umgebung
Labels	Auflistung aller gesetzten Labels, besteht aus mehreren Elementen
Sichten	Die gespeicherten Sichten
Kameraposition	Position der Kamera zur Speicherzeitpunkt
Isoflächenwert	Größe der idealisierten Funkwolke
Berechnungsparameter	Parameter, die Feldstärkenberechnung beeinflussen
Terrain-MRU	Liste der zuletzt verwendeten Terrains
Textur-MRU	Liste der zuletzt verwendeten Texturen
Stadtmodell-MRU	Liste der zuletzt verwendeten Stadtmodelle

Tabelle 6: Veränderbare Daten

5.4.2 Unveränderbare Daten

Unveränderbare Daten finden sich alle in der Klasse `MainWindow`. Sie dienen hauptsächlich dem Aufbau des Szenengraphen, der internen Repräsentation der 3D-Umgebung (vgl. Abb. 46).

5.4.3 Der Szenengraph

Der Szenengraph in Geryon besitzt eine relativ statische Struktur. Einige Teile werden je nach aktivierten Optionen ausgeblendet, das Grundgerüst bleibt jedoch immer identisch. Einzig der Fall, dass kein Terrain geladen ist, unterscheidet sich komplett vom Normalzustand, da dann kein Szenengraph vorhanden ist. Abb. 46 zeigt den Normalzustand des Graphs. Die angegebenen Variablennamen sind aus `MainWindow` entnommen. Elemente ohne Namen sind dynamisch angelegt, ohne dass darauf eine Referenz in `MainWindow` gespeichert wird.

5.5 Zentrale Abläufe

Geryon hat als interaktive Anwendung im Controller, d.h. der Klasse `MainWindow`, sehr viele verschiedene Abläufe, da es auf jede Benutzeraktion reagieren muß. Allerdings sind die meisten Abläufe primitiv, da sie lediglich mit einfachen Model-Änderungen auf Signale der View reagieren müssen. Einige Abläufe sind jedoch komplexer, so dass sie im folgenden näher erläutert werden sollen.

5.5.1 Szenario laden

Das Laden eines Szenarios ist die zentrale Funktion zum Laden der unveränderbaren Daten anhand der Verweise im Szenario. Die Funktion `setScenario(Scenario* scenario)` der Klasse `MainWindow`, deren Ablauf in Abb. 47 dargestellt ist, erfüllt diesen

Zweck. Sie sorgt für das Entfernen eines u.U. vorhandenen Szenarios (mit der dazugehörigen Nachfrage zum Speichern) und dem Laden aller zusätzlichen Daten. Dabei muß sie zusätzlich verschiedene Ausnahmesituationen beachten, wenn zum Beispiel kein Stadtmodell vorhanden ist oder nicht geladen werden kann.

5.5.2 Terrain laden

Abb. 48 zeigt den Ablauf zum expliziten Laden eines Terrains durch den Benutzer. Im Gegensatz zum Laden eines Szenarios, das definierte Ausgangsbedingungen hat bzw. sich schafft, müssen hier zusätzlich Abhängigkeiten zwischen den einzelnen unveränderbaren Daten überprüft und nur die betroffenen Teile des Szenengraphen aktualisiert werden.

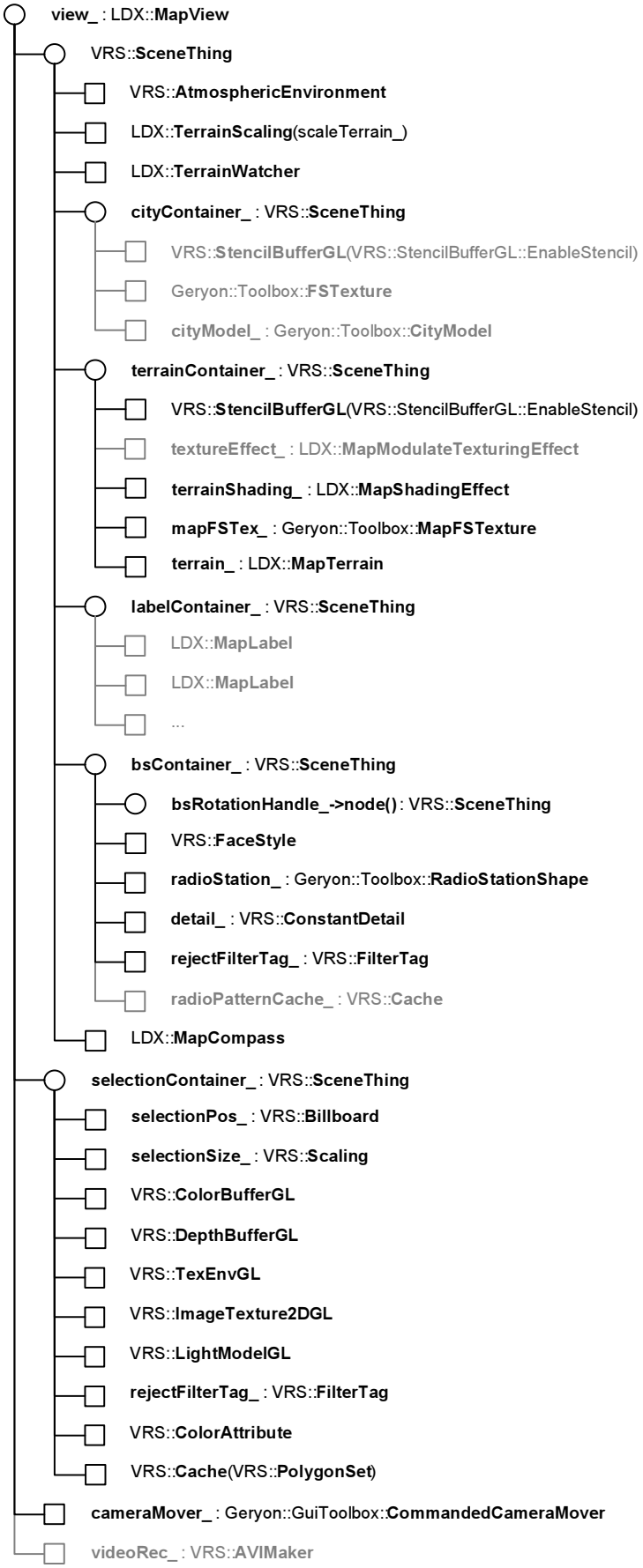

Abb. 46: Der Standard-Szenengraph von Geryon

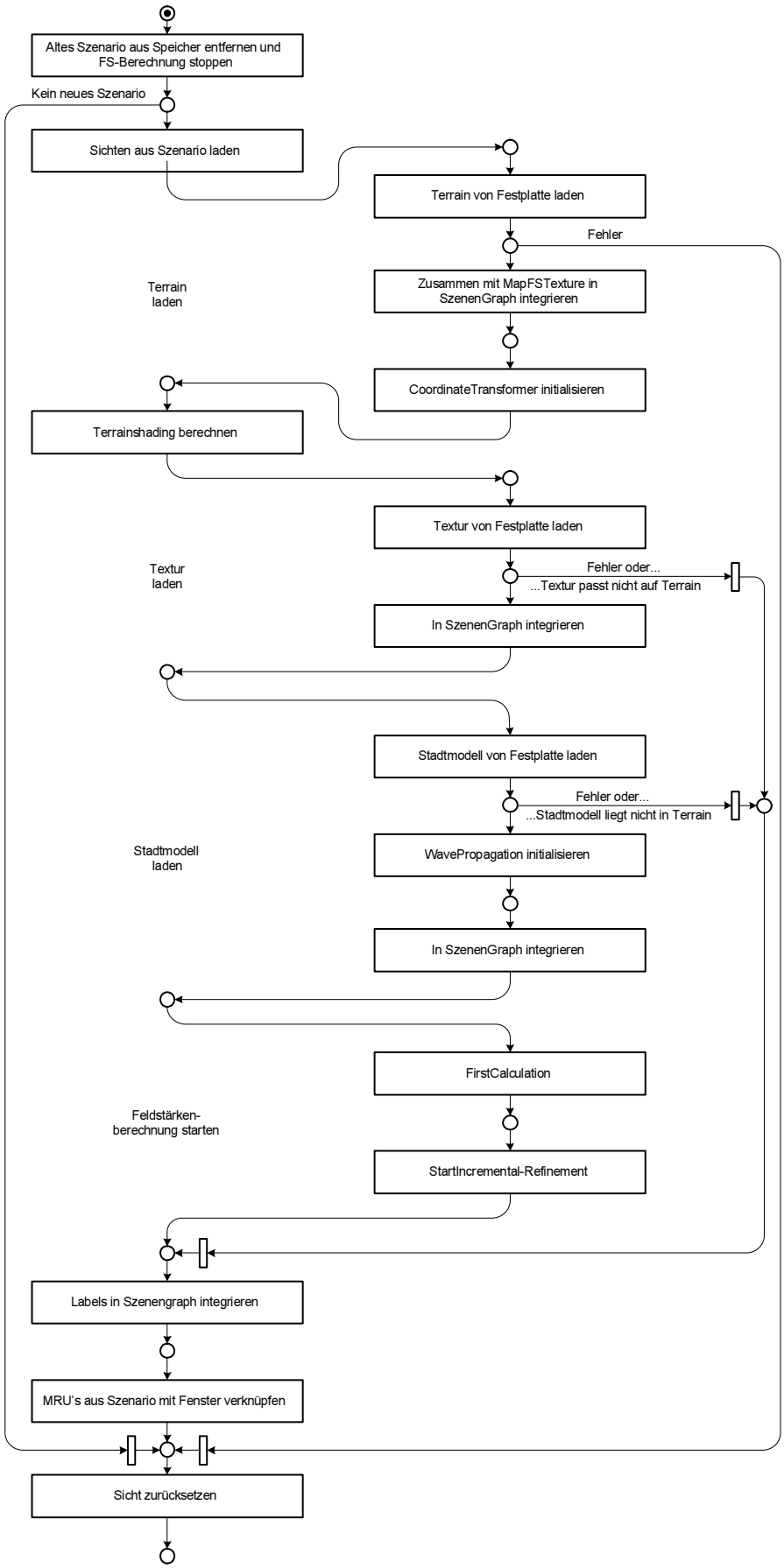

Abb. 47: Ablauf der Funktion "Szenario laden"

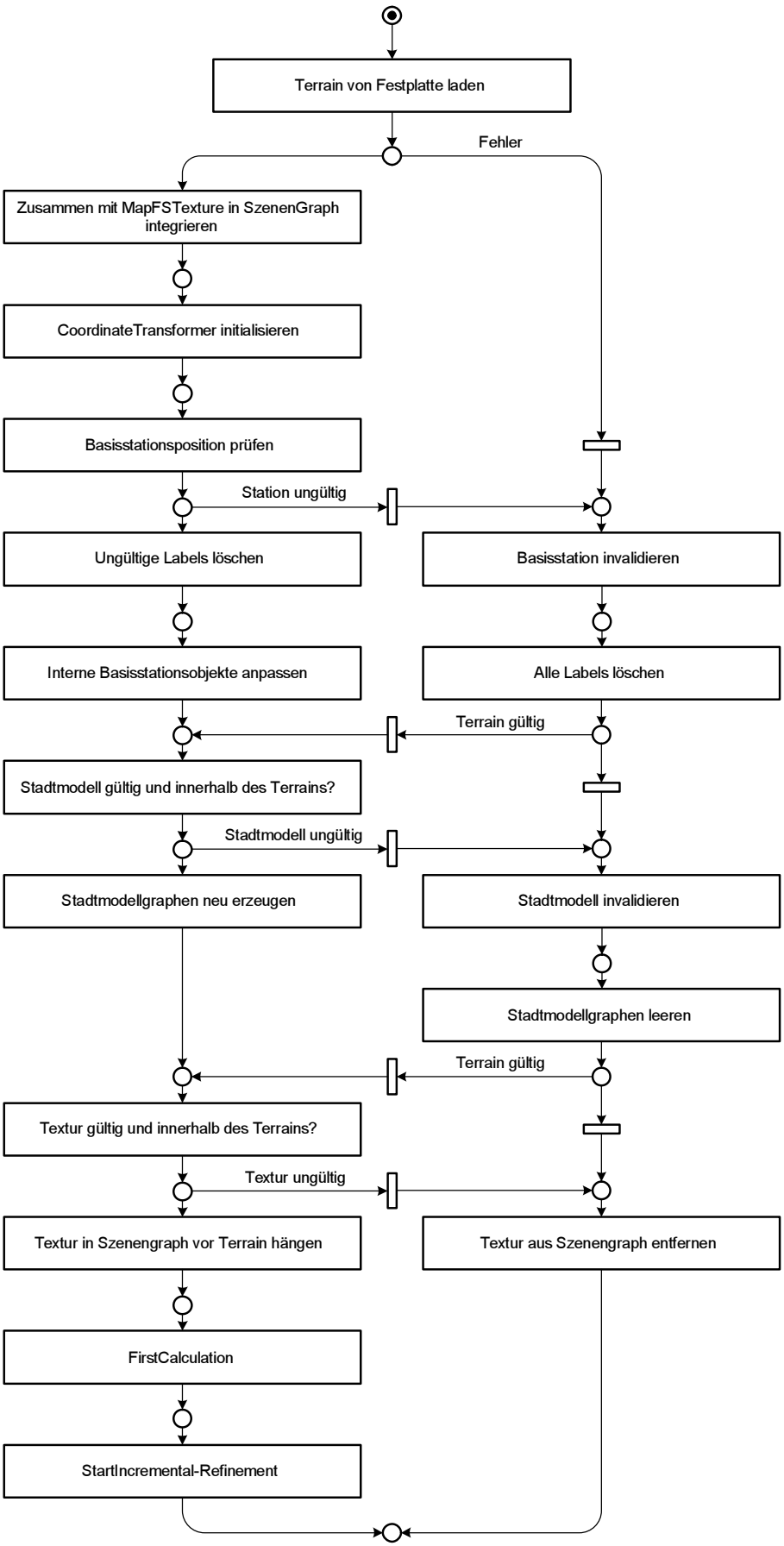

Abb. 48: Ablauf der Funktion "Terrain laden"

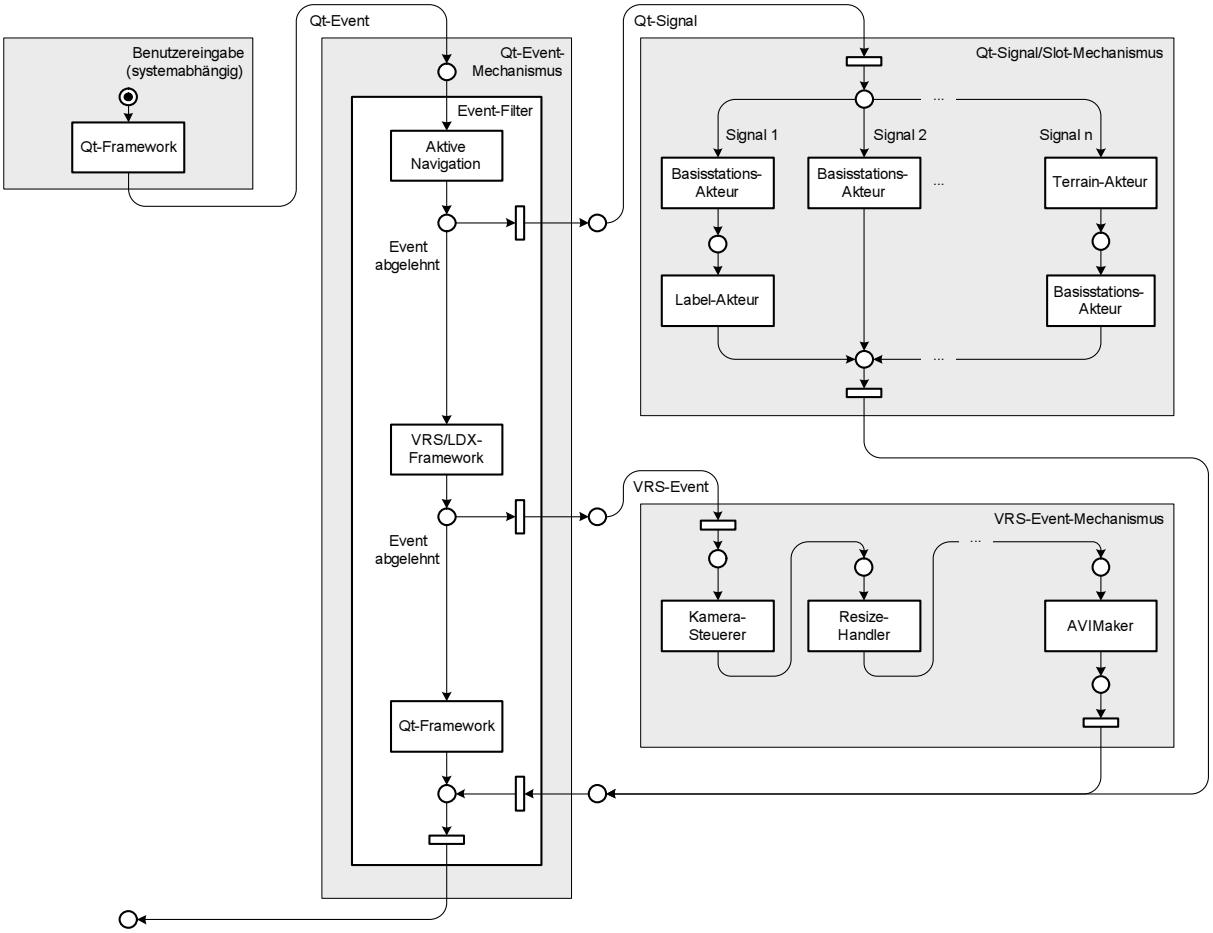

Abb. 49: Signalfluß in Geryon

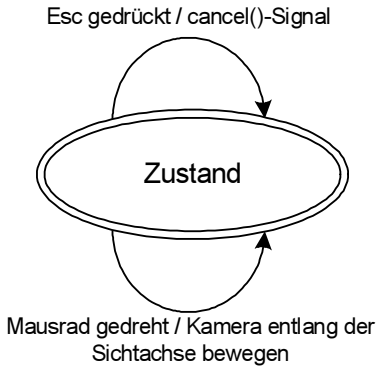

Abb. 50: Immer ausführbare Aktionen

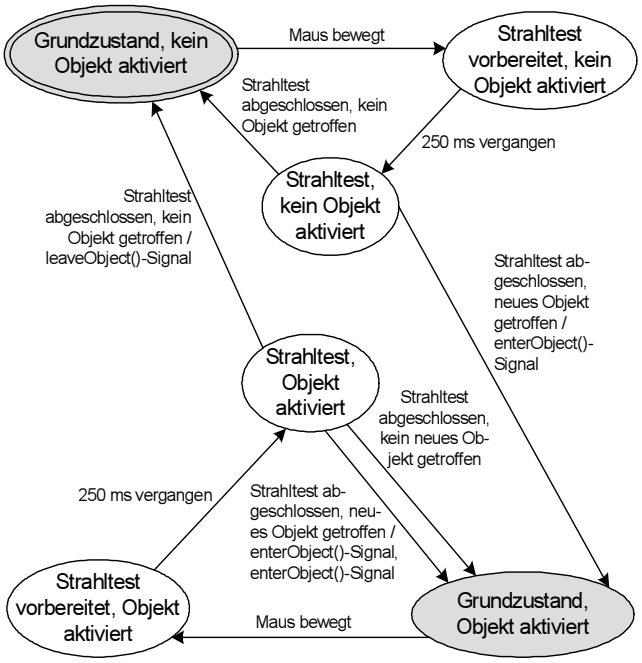

Abb. 51: Timergesteuerter Strahltest

Name	Funktion
`StandardNavigation` (ungenutzt)	Kamerabewegung mittels der Hovering-Metapher. Es wird kein Signal versandt, sondern direkt mit dem `CameraMover` kommuniziert.
`RotationNavigation` (ungenutzt)	Kamerabewegung mittels einer angepaßten Trackball-Metapher. Es wird kein Signal versandt, sondern direkt mit dem `CameraMover` kommuniziert.
`CameraNavigation`	Kombination der Kamerabewegung mittels der Hovering- und Trackball-Metapher. Es wird kein Signal versandt, sondern direkt mit dem `CameraMover` kommuniziert.
`DragAndDropNavigation`	Gleiche Funktion wie `CameraNavigation`. Zusätzlich wird Picking und Drap and Drop unterstützt. Dazu werden der Applikation Signale zur Reaktion gesandt.

Tabelle 7: Navigationen und deren Funktion

5.5.3 Arbeitsweise der Navigationen

Die Navigationen in Geryon übernehmen die Abstraktion der Benutzereingaben, um diese auf Programmaktionen abzubilden. Das LDX-Framework bietet bereits ausgefeilte Möglichkeiten zur Kamerasteuerung, jedoch wollten wir neben der Kamerasteuerung auch die Interaktion mit der Umgebung realisieren, was uns zur Entwicklung eines eigenen, auf Qt basierenden Konzepts veranlasste.

In diesem Konzept gibt es verschiedene Akteure, die die Eingaben in mehreren Stufen uminterpretieren und mit den jeweiligen Daten verknüpfen, um letztlich die Reaktion der Anwendung zu implementieren. Der Großteil des Codes sollte im Sinne der Objektorientierung wiederverwendbar sein, weshalb problemspezifische Annahmen wie die Existenz bestimmter Szenenelemente o.ä. so weit wie möglich abstrahiert wurden. Abb. 49 zeigt die beteiligten Akteure. Daneben ist auch der Unterschied der verschiedenen Benachrichtigungsmechanismen sichtbar. Während Events unabhängig von ihrem Typ immer durch die gleiche Kette von Akteuren geleitet werden, die jeweils entscheiden müssen, ob das Event für sie interessant ist, werden Signale schon anhand ihres Typs in unterschiedliche Akteurketten vermittelt.

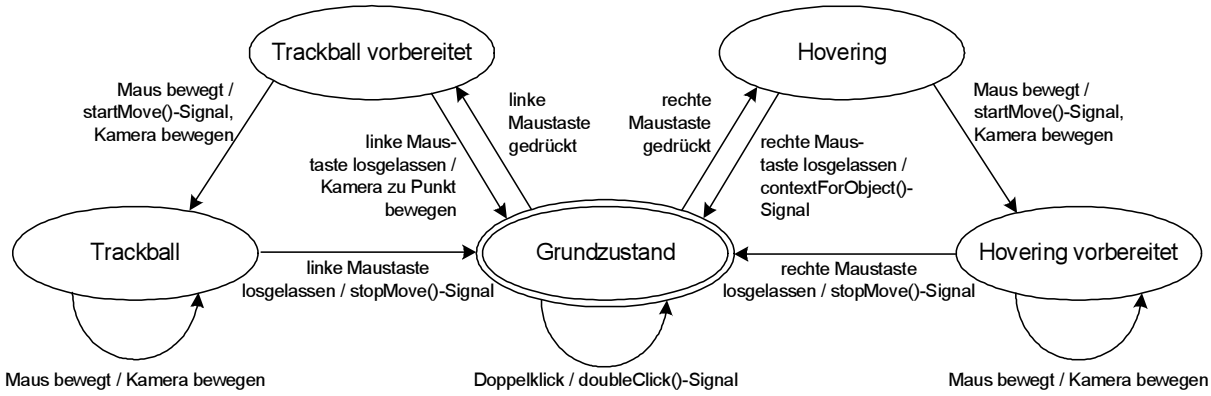

Abb. 52: Zustandsdiagramm `CameraNavigation`

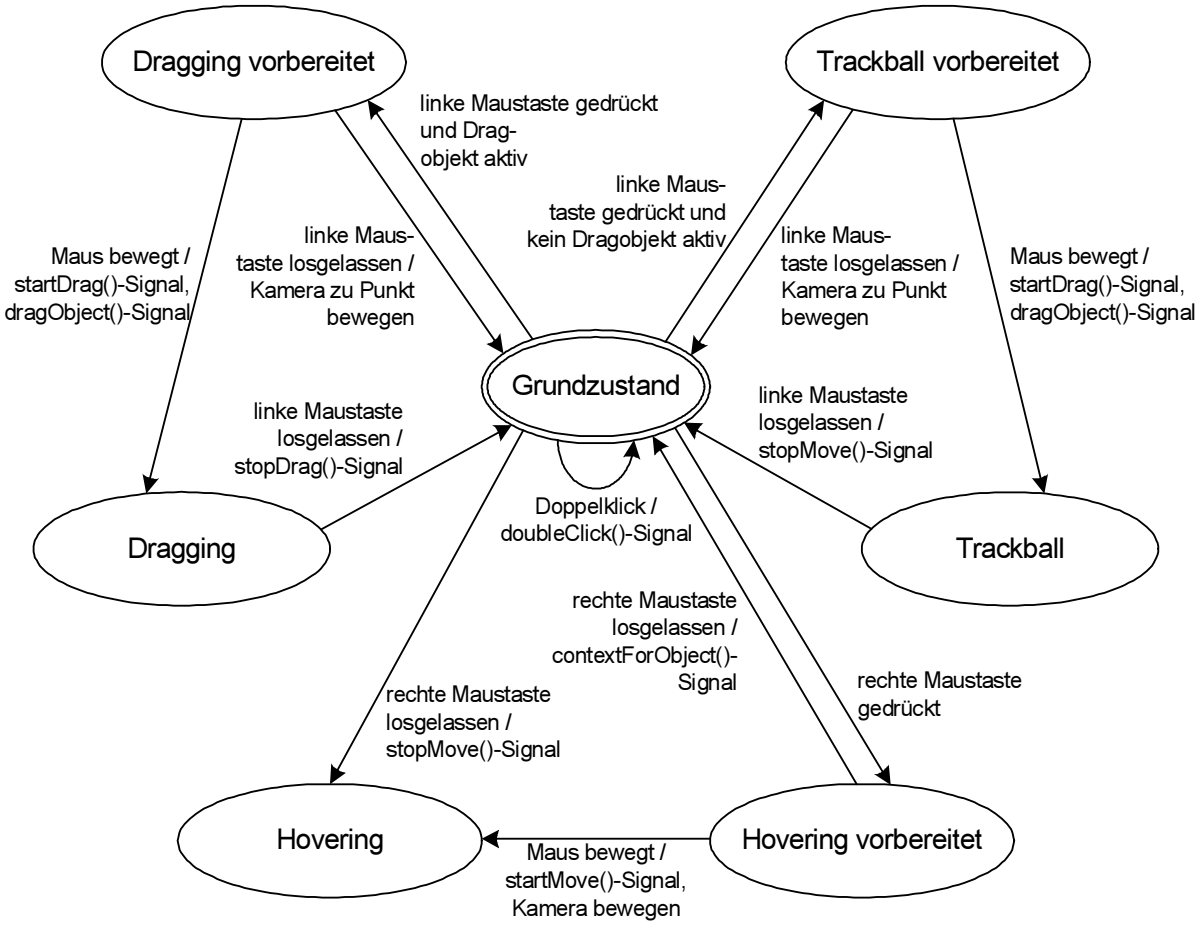

Abb. 53: Zustandsdiagramm `DragAndDropNavigation`

Die Navigationen übernehmen in dieser Struktur die Übersetzerfunktion zwischen der Qt-Eventwelt, die wenig abstrahiert alle Benutzereingaben, aber auch Systemereignisse, wie Timer u.ä., weiterleitet und der Qt-Signalwelt, die bereits auf applikationsspezifischem Level vorverarbeitete Nachrichten vermittelt. Die Navigationen sind somit Filter, die aus dem Eventstrom Signale erzeugen. Zudem erfüllen sie auch Funktionen, insbesondere die Kamerasteuerung, die keine Mitarbeit der Applikation erfordern, da sie als grundsätzlich betrachtet werden.

Für Geryon wurden vier verschiedene Navigationen entwickelt, von denen derzeit zwei genutzt werden. Tabelle 7 zeigt die Navigationen und deren Funktionen. Die Arbeitsweise der Navigationen lässt sich sehr gut durch Zustandsautomaten beschreiben. Dabei werden jedoch zwei parallele Automaten nötig, da es einige Aktionen gibt, die unabhängig vom aktuellen Zustand jederzeit ausführbar sind. Diesen Teil enthält Abb. 50. Die Arbeitsweise der `CameraNavigation` wird in Abb. 52 gezeigt.

`DragAndDropNavigation` ist deutlich komplexer, da es zusätzlich zur Kamerasteuerung auch noch Object-Picking und Drag-And-Drop beherrscht (Abb. 53). Das Picking ist mittels des MapRayQuery-Mechanismus des LDX-Frameworks realisiert. Da der Strahlentest viel Rechenzeit beansprucht, welche der Feldstärkeberechnung entzogen werden muss, wird die Frequenz der Tests durch einen Timer begrenzt. Dieser Mechanismus ist der Übersichtlichkeit halber getrennt in Abb. 51 dargestellt. In den Zustandsdiagrammen sind auch die Qt-Signale notiert, die jeweils verschickt werden, um die Applikation zu informieren. Die konkrete Verknüpfung der Signale mit Slots aus `MainWindow` ist in `MainWindow::setupNavigation()` zu finden.

5.6 Anforderungen und deren Umsetzung im GUI

Im Visionpaper für Geryon wurden verschiedene Anforderungen definiert, die sich direkt auf die Gestaltung des GUI und dessen Funktionalität auswirken. Hier ein Auszug aus den Features von Geryon:

Product Position Statement:
1) Präsentation der Auswirkungen einer einzelnen Basisstation auf die Umgebung für ein nicht-technisches Publikum. Dabei sollen Parameteränderungen sofort sichtbar werden.

Features Benutzerschnittstelle:
18) Interaktive Bewegung in der 3D-Umgebung in Echtzeit
19) Interaktive Änderung von Parametern
20) Verschiebung der Sendeanlage
21) Antennendiagrammvorschau
22) Grundlegende Aktionen auch ausschließlich mit Maus ausführbar (Einhandbedienung)
23) Klare, einfache Bedienung
24) Speichern und Wiederherstellen der Arbeitsumgebung
25) Mehrsprachige Oberfläche, Standardsprache Deutsch

Insbesondere der Punkt 1 stellt eine grundlegende Anforderung dar, die eine Zweiteilung der Oberfläche sinnvoll erscheinen lässt: zum einen sollte in einer Art Vorbereitungsmodus ein Datensatz erzeugt werden können, der auch gewisse schnell wählbare Voreinstellungen enthält, und zum anderen sollte ein Präsentationsmodus vorgesehen sein, der keinerlei störende Bedienelemente enthält, um den Blick auf das Wesentliche für das Publikum nicht zu verstellen. Die Bedienung erfolgt allerdings immer durch eine geschulte Person, nicht durch Außenstehende. Nichtsdestotrotz muss die Bedienung leicht verständlich sein, um auch während der Präsentation einen flüssigen Ablauf zu gewährleisten.

Für die Umsetzung gibt es zwei Möglichkeiten: zwei getrennte Applikationen, die eine gemeinsame Datenschnittstelle haben, oder eine Applikation, die sich zur Laufzeit umschalten lässt. Wir haben uns für den zweiten Ansatz entschieden. In einem Fenstermodus stehen neben der direkten Manipulation in der 3D-Umgebung Toolbars und ein Menü zur Interaktion zur Verfügung. Im Vollbildmodus werden alle Umrandungen ausgeblendet, so dass nur die 3D-Umgebung sichtbar bleibt. Zudem wurde die Navigation in der 3D-Umgebung entsprechend gestaltet. Während im Fenstermodus die direkte Manipulation des Geländes (z.B. Platzieren von Labels) möglich ist, bietet der Vollbildmodus einzig die Bewegung der Kamera als Navigation an. Diese beabsichtigte Reduzierung der Möglichkeiten im Vollbildmodus soll Fehlbedienungen während der Präsentation weitgehend ausschließen.

Neben diesem Hauptpunkt ist der Punkt 19 von erhöhtem Interesse. Eine interaktive Änderung von Parametern erfordert den Einsatz von nicht-modalen Dialogen mit der entsprechenden Logik, um die Veränderungen allen betroffenen Programmteilen bekannt zu machen. Der Signal-Slot-Mechanismus von Qt bietet hierfür eine gute Grundlage. Daneben wird jedoch auch der Callback-Mechanismus von VRS benötigt, da einige grundlegende Klassen wie die Repräsentation der Basisstation nicht auf Qt beruhen. Somit muss `MainWindow` als Vermittler zwischen beiden Welten dienen: der Datenrepräsentation innerhalb von VRS auf der einen Seite und den Qt-Dialogen auf der anderen.

6. Ausblick und Weiterentwicklungen

Während der Projektlaufzeit haben sich interessante Ideen entwickelt, sowohl seitens der Projektmitglieder als auch seitens T-Mobile, Geryon weiter zu entwickeln. Außerdem konnten einige Aspekte der Anforderungen nicht termingerecht fertiggestellt werden.

6.1 Feldstärkenberechnung

Bei einem Treffen mit der T-Mobile zur Sichtung des Projektstandes ergab sich, das die interaktive Berechnung der Feldstärke durch Berücksichtigung der Landnutzung in Gebieten, die nicht durch das Stadtmodell abgedeckt werden, deutlich verbessert werden kann.

6.2 Benutzeroberfläche

Die Benutzerschnittstelle erweist sich, trotz der gegebenen Benutzbarkeit, als ausbaufähig. Interessante Aspekte sind verbesserte Hilfestellungen wie kontextsensitive Hilfe, eine weitergehende Personalisierbarkeit der Oberfläche und die Verfeinerung der Mauseingaben.

Die Personalisierbarkeit der Oberfläche umfasst derzeit einzig die Fensterpositionen, die auch gespeichert werden sowie die Verschiebung der Toolbars zur Laufzeit, wobei bei Programmstart ein Standardlayout wiederhergestellt wird. Wünschenswert wäre neben der Speicherung der Toolbarpositionen auch die Änderung der Toolbarfunktionen bzw. die Erstellung eigener Toolbars. Die erweiterte Personalisierung beinhaltet auch eine individuelle Tastaturbelegung.

Die Mauseingabe erfordert noch etwas Feinschliff, um verschiedene Situationen besser erkennen zu können. Derzeit wird zum Beispiel bei jedem Doppelklick auch ein Einzelklick ausgelöst, was unerwünschte Aktionen zur Folge haben kann. In [33] gibt es Richtlinien zur Erkennung von Mausaktionen, die jedoch noch nicht umgesetzt sind.

6.3 Geryon

Das Modul zur Feldstärkenberechnung und Stadtmodelldarstellung kann losgelöst von Geryon benutzt werden. Es böte sich an, beide Module im Rahmen von Projekt B zu nutzen und so deren erweiterte Darstellungs- und Steuerungsmöglichkeiten mit unseren inhaltlichen Erweiterungen zu kombinieren.

Eine zentrale Anforderung, die wir bisher nicht testen konnten, ist die Portierung von Geryon auf Linux. Derzeit werden kaum plattformabhängige Mechanismen eingesetzt. Nichts desto trotz stehen Anpassungsarbeiten und Tests aus.

7. Der Entwicklungsprozess

7.1 Idee des Bachelorprojekts

7.1.1 Allgemein

Das Bachelorprojekt stellt einen abschließenden Höhepunkt des 7-semestrigen Bachelorstudiums dar. Es dient dazu, die erworbenen Kenntnisse in die Praxis umzusetzen und möglichst viele technische und soziale Facetten des Arbeitsalltags zu erleben. Welche dies im einzelnen sind, darüber klärt der Internetauftritt des Hasso-Plattner-Instituts auf:

Bachelor-Absolventen der Softwaresystemtechnik sind befähigt, in Teams technische und organisatorische Aufgaben im Kontext des Software Engineering zu bearbeiten. Sie besitzen folgendes Profil:

- Die Absolventen kennen mathematische und ingenieurwissenschaftliche Prinzipien.
- Die Absolventen verfügen über Kenntnisse zu Prinzipien, Methoden und Techniken zur organisatorischen Beherrschung großer softwareintensiver Systeme, z.B. Modellierung und Dokumentation, Projektplanung, Projektleitung, Projektmanagement.
- Die Absolventen können Prinzipien, Methoden und Techniken zur Softwareentwicklung bewerten, auswählen und anwenden, z.B. Analyse, Entwurf, Modellierung, Implementierung, Validation von Softwaresystemen.
- Die Absolventen verfügen über Fähigkeiten zur Kommunikation von Wissen über Softwaresysteme.
- Die Absolventen besitzen die Befähigung zum zügigen Einarbeiten in neue Informationstechniken.
- Die Absolventen besitzen Kenntnisse über Softwarebasissysteme sowie Spezialkenntnisse in den von ihnen gewählten Vertiefungsrichtungen.
- Die Absolventen besitzen praktische Erfahrungen im Bereich Software Engineering.
- Die Absolventen verfügen über soziale Kompetenz zur Arbeit in Teams.

7.1.2 Bachelorprojekt A: Geryon

Geryon hat das Ziel, eine interaktive dreidimensionale Visualisierung der Ausbreitung von Mobilfunkwellen in einem Stadtmodell zu bieten. Genaue Details finden sich im Kapitel 0. Der Lehrstuhl für Computergrafische Systeme am Hasso-Plattner-Institut der Universität Potsdam bietet dieses Projekt in Kooperation mit der T-Mobile an, da bereits seit längerer Zeit gemeinsame Forschungen in Hinblick auf Funknetzplanungstools laufen (siehe [1]).

7.2 Rahmenbedingungen

7.2.1 Auftraggeber

Mit dem Auftraggeber, der T-Mobile, bestehen Kontakte schon über einen größeren Zeitraum hinweg. Herr Dr. Konstantin Baumann war gemeinsam mit Herrn Prof. Jürgen Döllner unsere

Ansprechstelle am HPI, mit ihnen klärten wir viele organisatorische Belange und koordinierten unseren Zeitplan.

Technische Spezifikationen erhielten wir direkt von der T-Mobile, die uns stets unbürokratisch und kurzfristig durch Herrn Eckhard Oppermann und Herrn Stefan Hoffrichter persönlich oder per Email zur Seite stand.

7.2.2 Auftragnehmer

Unser Geryon-Team setzte sich aus drei Personen zusammen: Haik Lorenz, Stephan Kirsch und Stephan Brumme. Wir drei studieren seit Oktober 1999 Softwaresystemtechnik und sind seit der Einrichtung des Lehrstuhls Computergrafische Systeme als hilfswissenschaftliche Mitarbeiter aktiv an der Forschung[34] auf diesem Gebiet beteiligt.

Wir konnten bereits auf einen recht großen Erfahrungsschatz im Bereich Softwareentwicklung zurückblicken, den wir während des Studiums sowie außeruniversitärer Aktivitäten, wie z.B. Praktika, erwarben.

7.2.3 Arbeitsplatz

Den Großteil der Projekttätigkeit führten wir am HPI durch, wobei uns ein Laborraum zur Verfügung stand. Jeder Student verfügte dort über einen eigenen Rechner, den er individuell konfigurieren konnte. Die Beschaffung zusätzlicher Hard- und Software geschah über das HPI, insbesondere Herr Baumann half uns dabei.

7.3 Softwareprozess-Techniken

7.3.1 Der Rational Unified Process

Eine der wesentlichen Herausforderungen bestand in der termingerechten Fertigstellung eines funktional vollständigen und qualitativ hochwertigen Produkts. Um dieses Ziel erreichen zu können, bedurfte es einer sorgfältigen Planung und deren konsequenten Umsetzung.

Von den verschiedenen Vorgehensmodellen, die uns im Verlaufe des Studiums vorgestellt wurden, sagte uns der Rational Unified Process[35] am meisten zu. Er beruht auf erprobten Verfahren aus der industriellen Praxis, die man auch als Best Practices bezeichnet. Unter den Firmen, die den Rational Unified Process einsetzen, finden sich viele Branchengrößen wie Oracle, MCI, Intel und Merrill Lynch[36].

Die zu Grunde liegende Idee, ein Produkt iterativ zu entwickeln, orientiert sich an den Gegebenheiten der Praxis. Es ist nämlich leider nahezu unmöglich, bereits zu Beginn eines Projekts alle Anforderungen exakt zu formulieren. Man muss stets davon ausgehen, dass sich die Kundenwünsche und wirtschaftliche oder soziale Rahmenbedingungen ändern.

Wir konnten den RUP nur teilweise in die Praxis umsetzen. Zwar gelang es uns, das iterative Konzept und die jeweiligen Arbeitsschritte der Phasen zu adaptieren, allerdings fehlte aufgrund der geringen Teamgröße eine eindeutige Rollenzuteilung der Studenten. Hier ist es wünschenswert, dass zukünftig darauf geachtet wird, die kommenden Bachelorprojektgruppen mit deutlich mehr als nur drei Personen zu besetzen.

Der zeitliche Ablauf stimmte nicht exakt mit der idealisierten Abb. 54 überein. So trat die Inbetriebnahme erst deutlich nach der Konstruktion ein, zu einem Zeitpunkt, wo Implementierung, Test etc. bereits abgeschlossen waren. Dass Abweichungen auftauchen, halten wir aber für völlig normal, wir gehen ferner davon aus, dass sie in fast jedem Industrie-Projekt zu finden sind.

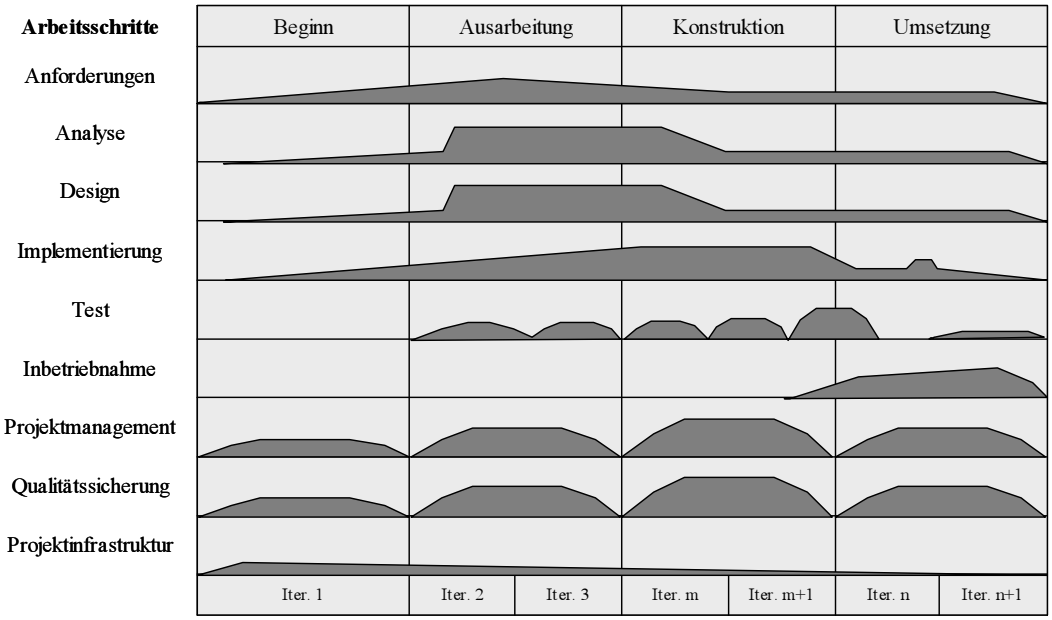

Abb. 54: Phasen im Unified Process[37]

Aufgrund interner Umstrukturierungen innerhalb der T-Mobile konnten nicht alle Projekttreffen wie gewünscht stattfinden. Dieser Umstand missfiel allen Beteiligten, war aber nicht zu ändern. Diese Tatsache dient jedoch gleichzeitig als gutes Beispiel dafür, dass nicht alles in einem Projekt planbar ist und unvorhersehbare Hindernisse jederzeit auftauchen können.

7.3.2 UML und FMC

Lange Zeit fehlte eine angemessene grafische Beschreibungssprache für Software. Erst Mitte der 90er Jahre entstand mit der Unified Modelling Language (UML) eine international standardisierte Notation, die OMG[38] verabschiedete 1997 dann UML 1.0. Die zur Zeit aktuelle Fassung UML 1.5 wird demnächst durch UML 2.0 ersetzt.

Der große Vorteil und gleichzeitig Nachteil von UML besteht in der starken Nähe zum Code. Man kann Programmierkonstrukte im Kleinen sehr schön darstellen und erläutern, jedoch ist es schwierig, mit dieser Notation abstrakte übergreifende Zusammenhänge zwischen Bauelementen klar und einfach zu verdeutlichen. Genau an dieser Stelle setzt FMC[39] an.

Während UML auf breiter Front durch Software (wie etwa Rational Rose, Microsoft Visio etc.) unterstützt wird, die auch in der Lage ist, direkt aus der Notation Code zu erzeugen, hat FMC an dieser Stelle noch Nachholbedarf. Für unser Projekt wiederum konnten wir keinen dieser UML-Codegeneratoren sinnvoll einsetzen, da die Qualität und vor allem die Verständlichkeit noch viel zu wünschen übrig ließ. Wahrscheinlich ändert sich dieser Zustand in den nächsten Jahren, so dass sich dann die Möglichkeit einer neuen Art von Softwareentwicklung eröffnet.

Da jede der beiden Notationen ihre eigenen Vor- und Nachteile besitzt, verwenden wir beide, um so aus der Summe der Stärken möglichst optimale Abbildungen zu gewinnen. In der Folge sind alle Grafiken, die den konzeptionellen Aufbau von Geryon auf einem hohen abstrakten Niveau zeigen (siehe Kapitel 0), unter Einsatz von FMC entstanden. Wenn es dann um die Realisierung auf Codeebene geht, sind UML-Bilder wesentlich effektiver.

Wir vertreten die Ansicht, dass Bilder in ihrer Aussagekraft dem geschriebenen Text deutlich überlegen sind. Allerdings kann nicht jeder Aspekt durch optische Mittel sinnvoll

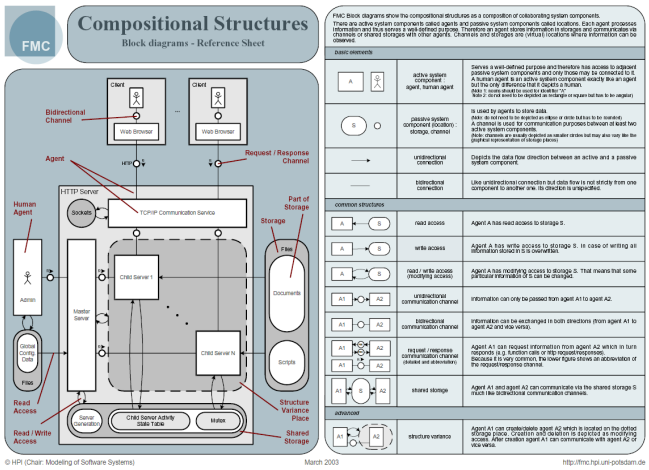

Abb. 55: Generelles FMC-Aufbaubild

visualisiert werden, weshalb wir als optimale Mischung auf eine Kombination von Wort und Bild setzen. Die Verständlichkeit und die Effektivität der Beschreibungen ist ein Resultat des Studiengangs Softwaresystemtechnik, sie ist unser Maßstab.

7.4 Eingesetzte Software

7.4.1 Betriebssystem

Der Auftraggeber stellte im Anforderungsdokument die Bedingung, dass Geryon auf jeden Fall unter Windows 2000 lauffähig sein soll. Im mündlichen Gespräch wurde weiterhin seitens der T-Mobile auch der Wunsch nach einer Linux-Version geäußert, jedoch konnten wir uns aus Zeitgründen nicht mehr darum kümmern. Der Code ist weitgehend portabel ausgelegt, es sollte mit nur geringem Aufwand möglich sein eine Linux-Version zu erstellen.

Fast alle Rechner am HPI laufen unter Windows 2000. Da wir drei Projektbearbeiter auch im privaten Umfeld am meisten Erfahrung mit diesem Microsoft-Betriebssystem besitzen, ließen wir diese Konfiguration unverändert bestehen.

7.4.2 Entwicklungsumgebung und Compiler

Die Bibliotheken VRS und LandExplorer werden vom Lehrstuhl ausschließlich mit dem Visual Studio 6.0 von Microsoft bzw. dem Intel C++ Compiler entwickelt. Um die Lauffähigkeit unter Linux zu gewährleisten, kommt ab und zu der frei verfügbare GCC-Compiler[40] zum Einsatz.

Wir sind mit dem Visual Studio 6.0 ebenfalls sehr gut vertraut. Der Umgang in privaten Projekten mit dem aktuellen Visual Studio .NET zeigte aber, dass wir damit noch effektiver und effizienter arbeiten können, so dass wir diese neue Version der älteren vorzogen.

Insbesondere der Editor weist sinnvolle Neuerungen auf, die wir bei anderen Konkurrenzprodukten vermissen. Hervorzuheben ist hier vor allem IntelliSense, das uns mit seiner Autovervollständigungsfunktion enorm bei Namensräumen und Bezeichnern half.

Die Migration verlief problemlos, da wir die vorhandenen Projekt-Workspaces ohne großen Aufwand importieren und konvertieren konnten. Beide Versionen des Visual Studios können übrigens auf einem System koexistieren, dieser Umstand ist gerade in der Migrationsphase von großem Vorteil.

In seltenen Fällen traten Abstürze auf, die uns bei der Arbeit aber nicht behinderten. Wir gehen davon aus, dass mit der Verfügbarkeit der ersten Service Packs diese Unzulänglichkeiten behoben sein werden.

7.4.3 Bibliotheken

Eine strikte Vorbedingung war, die 3D-Visualisierung mit Hilfe der Bibliothek LandExplorer durchzuführen. Sie setzt wiederum auf dem Virtual Rendering System 3.0 (VRS) auf, das ebenfalls hier am Hasso-Plattner-Institut entwickelt wird. Zwar ist VRS derart offen konzipiert worden, dass theoretisch beliebige Rendering-Systeme zugrunde liegen können, zur Zeit ist OpenGL 1.4 allerdings die einzige breit unterstützte Realtime-Schnittstelle.

Aus Performance-Gründen umgehen wir an einigen Stellen den LandExplorer und greifen auf VRS-Funktionalität zurück. Sehr kritische Routinen setzen gar direkt auf OpenGL auf.

Die Benutzeroberfläche entstand mit Qt 3.0[41] unter dem Gesichtspunkt einer späteren Portierung von Windows nach Linux. Die alternativ zur Auswahl stehenden GUI-Toolkits scheiterten entweder an der Portabilität (MFC), an mangelnder Objektorientierung (Gnome) oder nicht ausreichender Geschwindigkeit (Tcl/Tk).

7.4.4 Konfigurationsmanagement

Ein unabdingbares Werkzeug zur Softwareentwicklung ist ein Konfigurationsmanagement-Programm. Unsere Wahl fiel auf das Concurrent Versions System CVS[42], da ein solcher Server am Lehrstuhl bereits existierte und erfolgreich bei VRS und dem LandExplorer Verwendung findet. Wir kannten dieses System bereits zu Projektbeginn, so dass auch hier eine Einarbeitungsphase nicht notwendig war.

Die Administration des CVS-Servers übernahm Florian Kirsch, der uns bei sämtlichen Fragen und Problemen sofort hilfsbereit zur Seite stand. Als Client-Software griffen wir auf das bewährte Programm WinCVS[43] zurück.

Abschließend sind wir mit CVS aber nicht völlig zufrieden. Viele Schwierigkeiten, wie etwa fehlende Dateiumbenennung unter Beibehaltung der Versionierungsinformationen, treten bei anderen Produkten nicht auf. Jedoch hat CVS den unschätzbaren Vorteil der enorm hohen Verbreitung, welche für eine hohe Zuverlässigkeit sorgt. In Zukunft könnten neue Open-Source-Systeme, z.B. Subversion[44], CVS ersetzen.

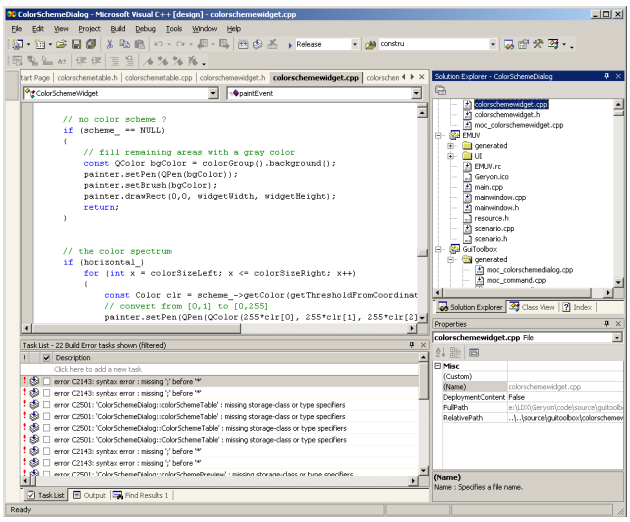

Abb. 56: Visual Studio .NET

7.4.5 Groupware zur Terminplanung

Von Anfang an wollten wir eine zentrale Verwaltung aller Projekttreffen und ausstehenden Tätigkeiten, um so unsere Aktivitäten besser koordinieren zu können. Wir zogen dabei die Fähigkeiten von Microsoft Outlook in Betracht, entschieden uns dann aber doch für eine web-basierte Lösung, die den Zugriff von einem beliebigen Rechner aus erlaubt, was besonders von Vorteil ist, wenn man auf das System von daheim aus zugreifen will.

Für unsere Ansprüche fanden sich mehrere gute Produkte aus dem Open-Source-Sektor auf professionellem Niveau. Nach einer kurzen Probephase ging dann MoreGroupware[45] in Betrieb, das sich als zuverlässig und stabil erwies.

7.4.6 Dokumentation

Bei der Dokumentation sind zwei wesentliche Varianten zu unterscheiden: direkt im Code oder extern. Erstere basiert auf Kommentaren, die zum Teil auch durch Tools analysierbar sind. Hiermit meinen wir insbesondere die Logs von CVS und Klassen- bzw. Methodenbeschreibungen für Doxygen[46]. Die Resultate sind zwar gut und ansprechend, allerdings mangelt es ihnen an einer Heraushebung wesentlicher Kernbestandteile. Sie sind demzufolge eher als Referenz denn als Lehrmaterial zu verstehen.

Für den Einstieg in Geryon eignet sich die externe Dokumentation viel besser. Sie entstand unter Verwendung von Microsoft Office, namentlich vorwiegend Word und Visio.

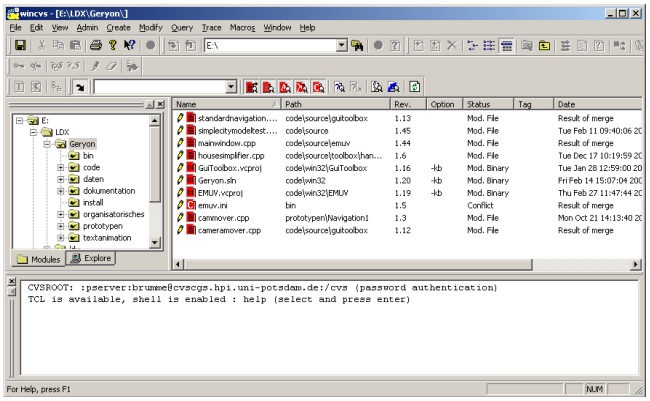

Abb. 57: WinCVS 1.2

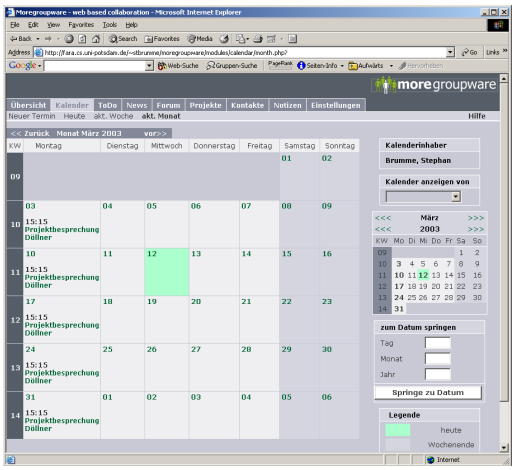

Abb. 58: MoreGroupware

8. Anhang

8.1 Spezifikation der Stadtmodell-Datei

Anmerkung: Terminale der Grammatik sind entweder kursiv dargestellt oder durch Anführungsstriche gekennzeichnet. Das Terminal *Zahl* steht für eine vorzeichenlose ganze Zahl in der Textdatei, das Terminal *Zeichen* repräsentiert ein beliebiges alpha-numerisches Zeichen.

(1) Hausdatei -> “# number of buildings” Trennsymbol Hausdateiinhalt

(2) Hausdateiinhalt -> *Zahl* Trennsymbol Häuser

(3) Häuser -> Haus Häuser
(4) Häuser -> Haus

(5) Haus -> Kommentar Haus
(6) Haus -> Ecken Fläche Fläche Fläche

(7) Ecken -> *Zahl* Eckpunkte

(8) Eckpunkte -> Eckpunkt Eckpunkte
(9) Eckpunkte -> Eckpunkt

(10) Eckpunkt -> *Zahl* Trennsymbol *Zahl* Trennsymbol *Zahl* Trennsymbol

(11) Fläche -> *Zahl* Trennsymbol Polygone

(12) Polygone -> Polygon Polygone
(13) Polygone -> Polygon

(14) Polygon -> *Zahl* Trennsymbol Indizes

(15) Indizes -> *Zahl* Trennsymbol Indizes
(16) Indizes -> *Zahl* Trennsymbol

(17) Trennsymbol -> “ “
(18) Trennsymbol -> *Tabulator*
(19) Trennsymbol -> *Zeilenumbruch*
(20) Trennsymbol -> Kommentar Trennsymbol

(21) Kommentar -> “#” Kommentarinhalt
(22) Kommentarinhalt -> *Zeichen* Kommentarinhalt
(23) Kommentarinhalt -> *Zeilenumbruch*

8.2 Zusätzliche Abbildungen

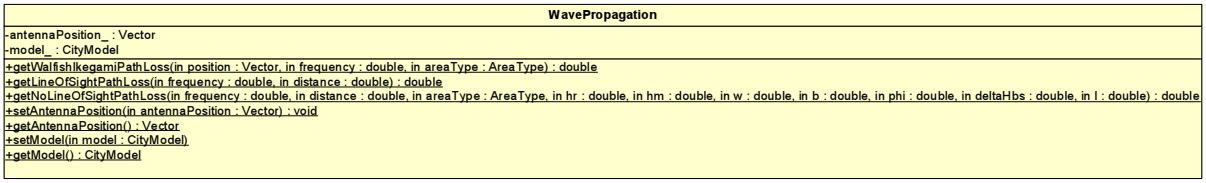

Abb. 59: UML-Beschreibung von WavePropagation

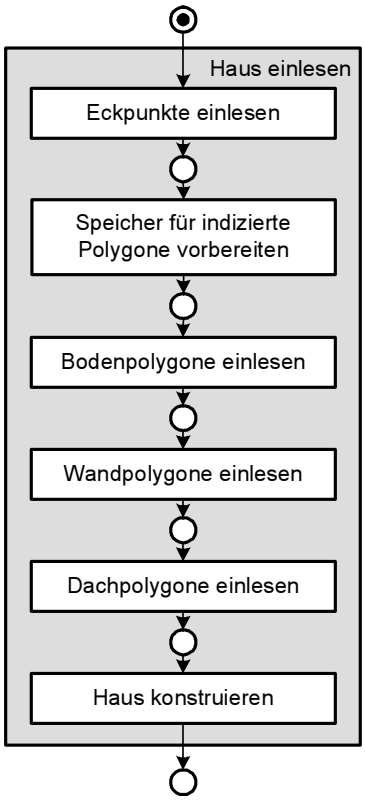

Abb. 60: Ablauf für das Einlesen eines Hauses

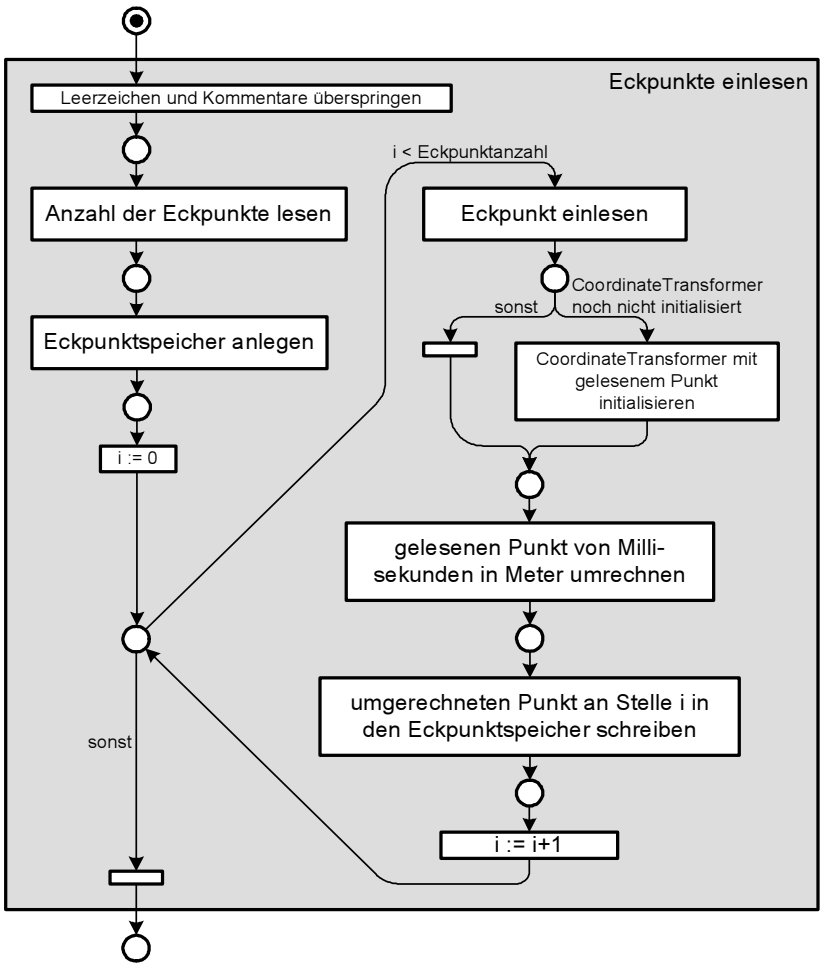

Abb. 61: Ablauf für das Einlesen der Eckpunte eines Hauses

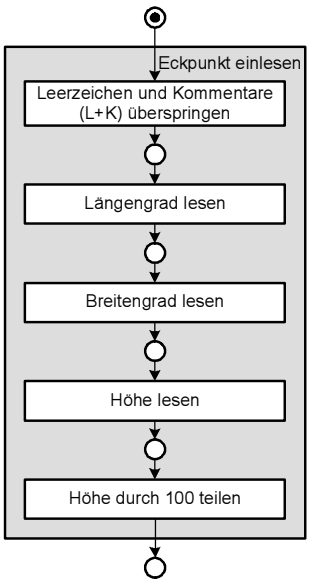

Abb. 62: Ablauf zum Einlesen eines einzelnen Eckpunkts

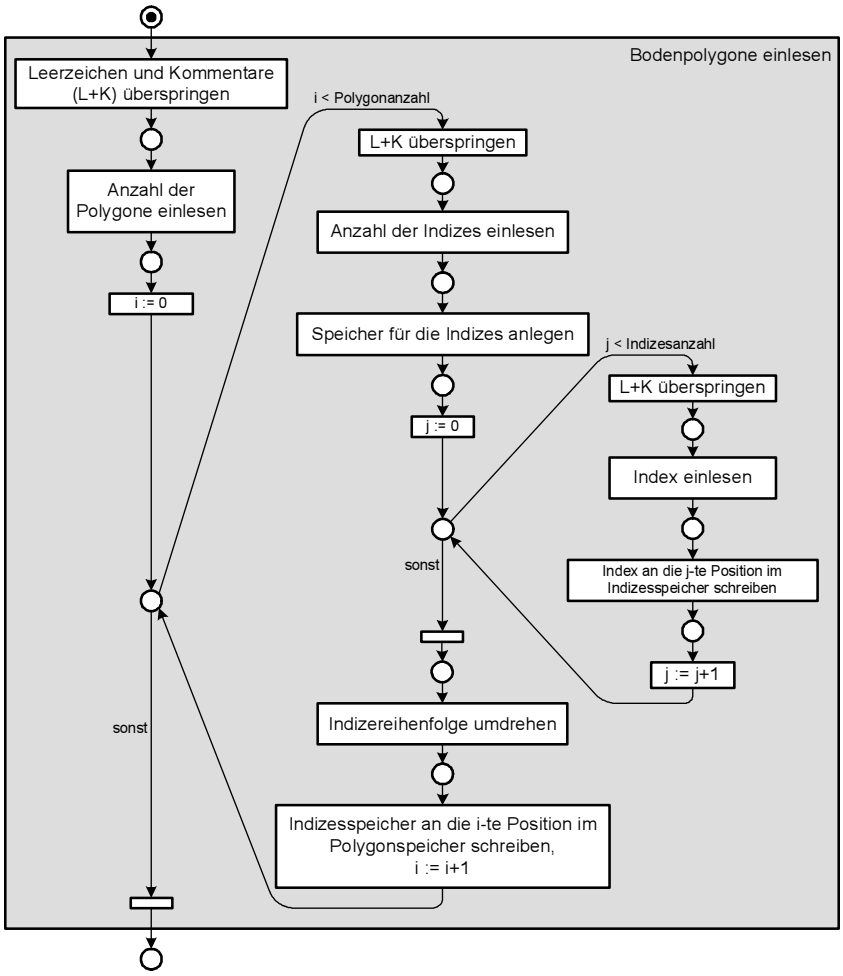

Abb. 63: Ablauf zum Einlesen der Polygone

Anmerkung: Zum Einlesen der anderen Polygone fällt lediglich der Schritt Indizereihenfolge umdrehen weg.

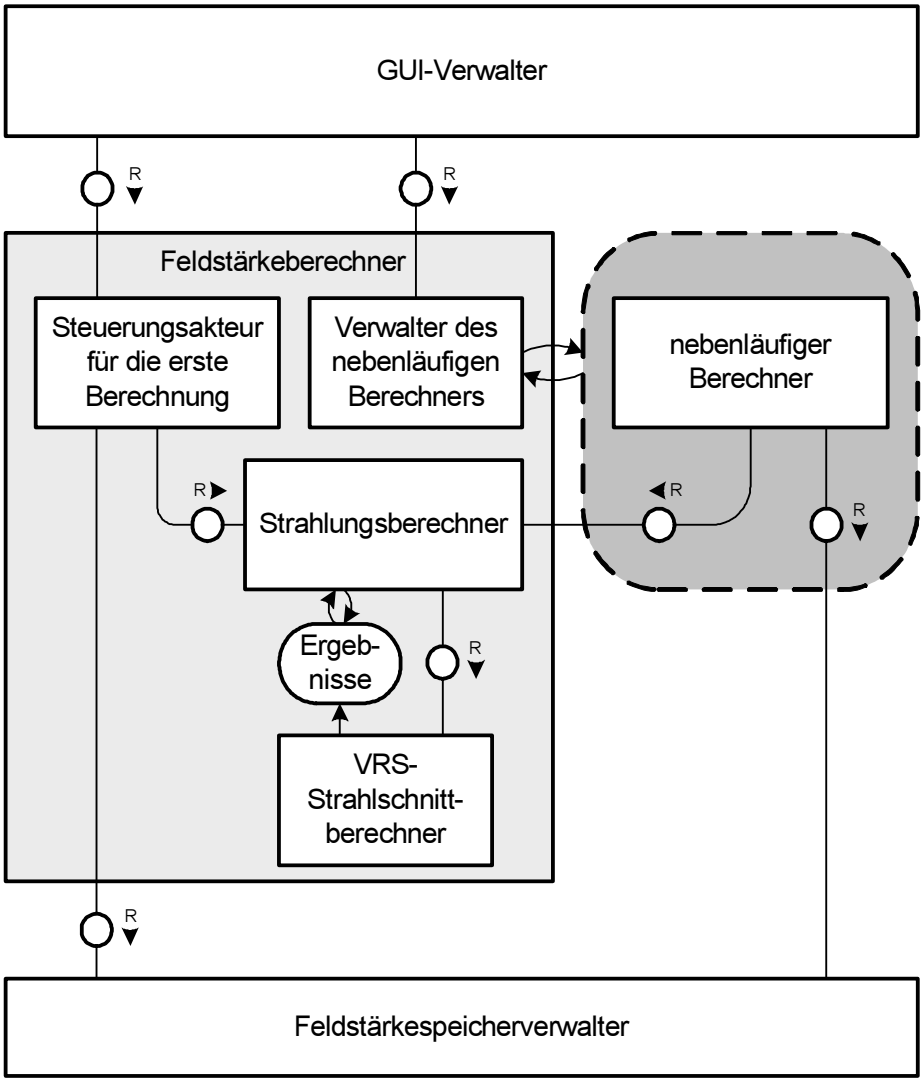

Abb. 64: Aufbau des Feldstärkebrechners

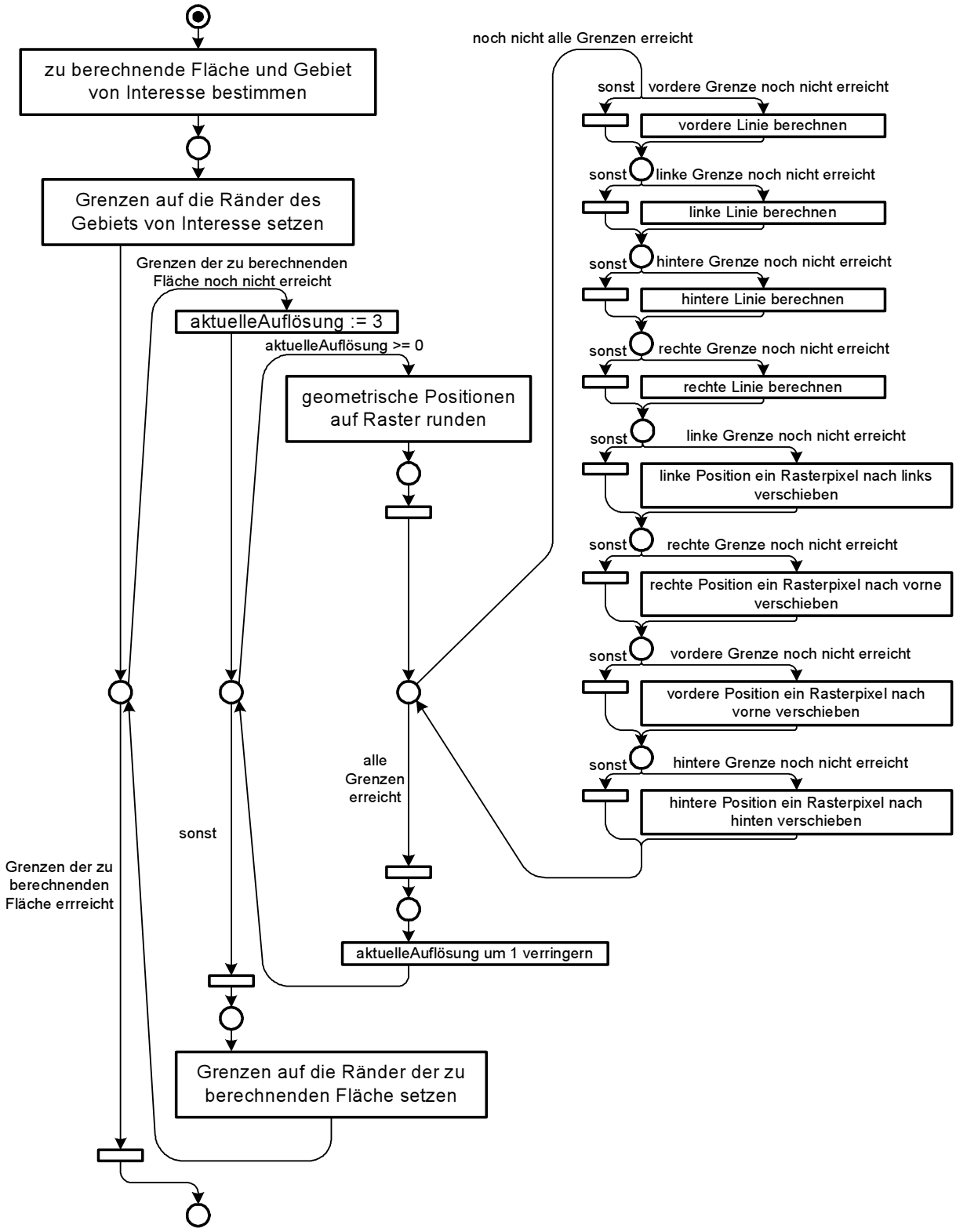

Abb. 65: Ablauf der Feldstärkepixelberechnung

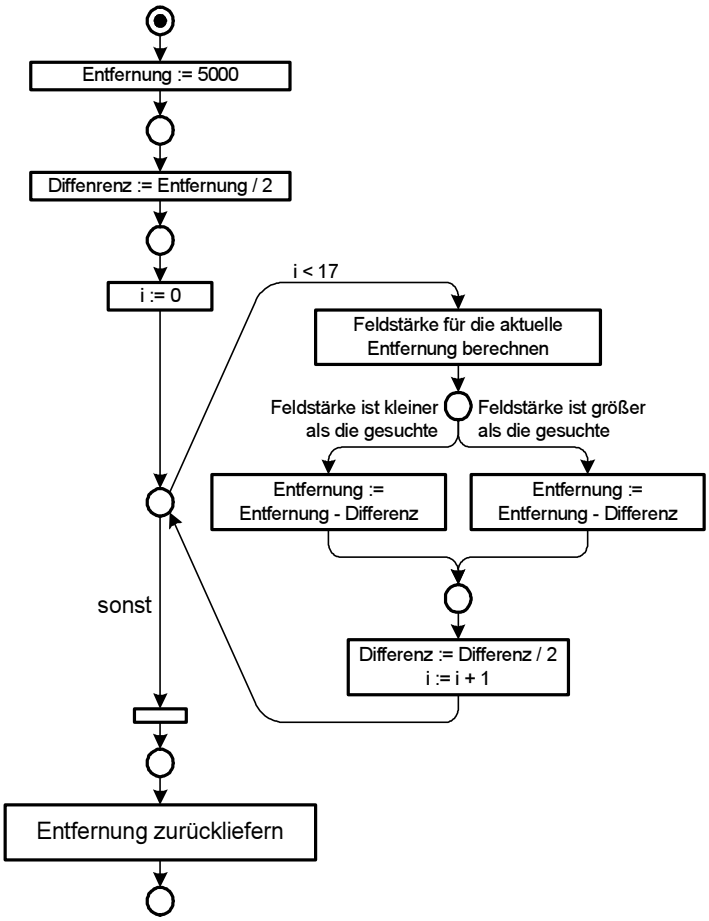

Abb. 66: Ablauf zur Bestimmung der Entfernung einer bestimmten Feldstärke

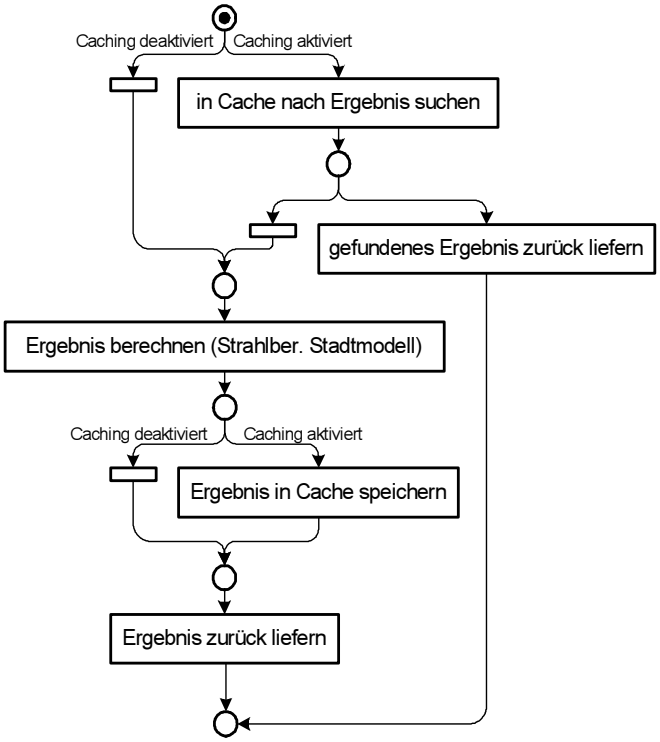

Abb. 67: Ablauf für den Strahlentest im Stadtmodell

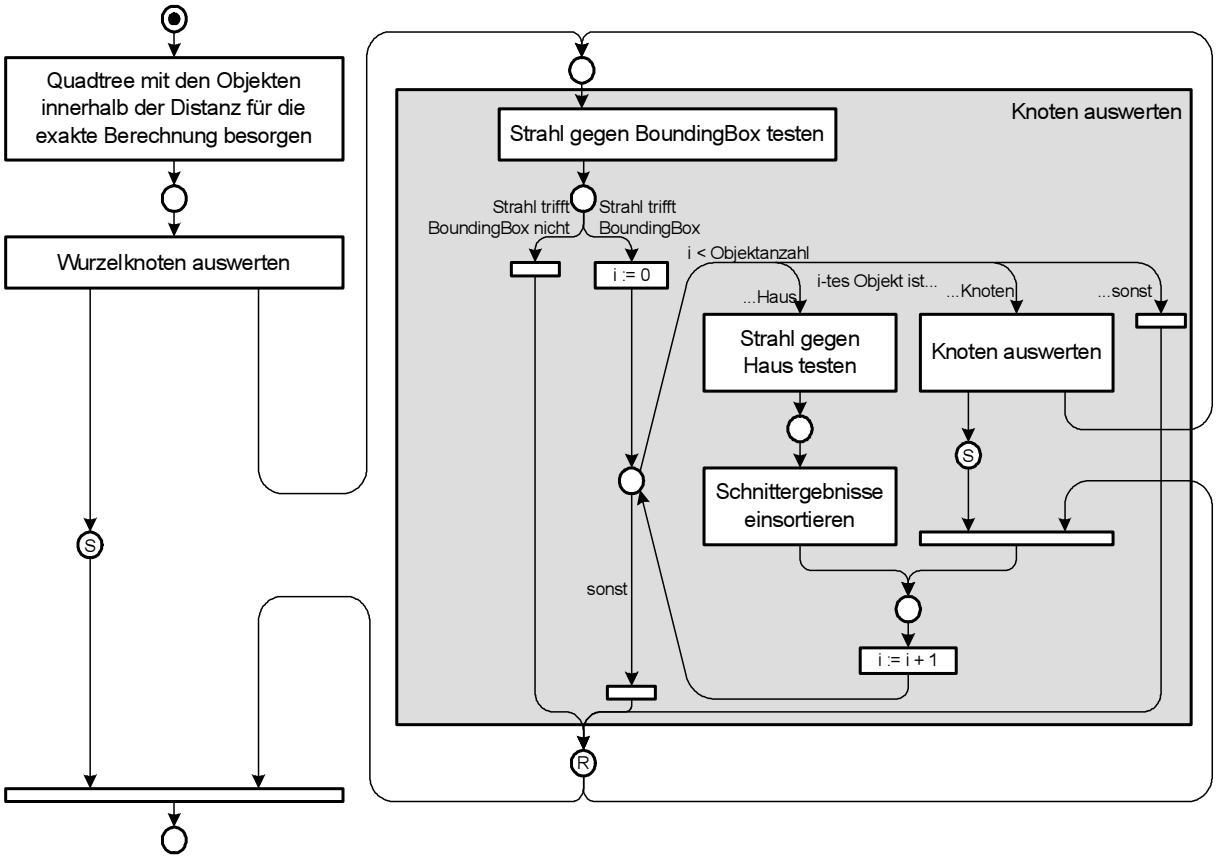

Abb. 68: Ablauf zur Berechnung eines Strahls durch das Stadtmodell

8.3 Glossar

Antenne	Sende- und Empfangseinheit, charakterisiert durch ein Antennendiagramm
Antennendiagramm	Dämpfung der elektromagnetischen Energie in Abhängigkeit von der Abstrahlrichtung
Basisstation, Basestation	Träger einer oder mehrer Antennen, Abkürzung BS
Feldstärke	Tatsächliche Funkleistung an einem bestimmten Punkt
CGS	Computer-Grafische Systeme, Lehrstuhl am HPI, geleitet durch Professor Döllner
Clipping	Entfernung außerhalb des Sichtvolumens liegender Bildteile auf Pixelebene
Culling	Entfernung nicht sichtbarer Bildteile auf Applikationsebene
Dämpfungsfaktor	Verhältnis der gesendeten zur empfangen Leistung
d1/d2	Im Kontext von Geryon eingeführte Parameter des Feldstärkeberechnungsmodells (siehe Kapitel 4.2.6.4)
Dezibel	Logarithmischer Dämpfungsfaktor (sieh Kapitel 2.2.2)
Echtzeit	Im Kontext der Computergraphik die ausreichend schnelle Generierung von Bildern (15 pro Sekunde)
Funkkeule	Idealisierte Abstrahlleistung einer Antenne
Funkmast	Teil des Versorgungsnetzes, der die physische Verbindung zu den Netzteilnehmern herstellt, kann mehrere Antennen besitzen, entspricht einer Basisstation
F	Frequency, Frequenz
FS	Field Strength, Feldstärke
Fresnelzone	Die für die ankommende Feldstärke relevante Zone zwischen Sender und Empfänger
Geländemodell	Virtuelle Repräsentation realer topografischer Formationen
Geryon	Produktname, aus der griechischen Mythologie: Neffe von Pegasos, Sohn des Chrysaor, drei-köpfiger Riese dessen Rinderherde von Herakles als eine seiner 12 Aufgaben gestohlen wurde
HPI	Hasso-Plattner-Institut, aus privaten Mitteln finanzierte Lehr- und Forschungseinrichtung in der Brandenburgischen Landeshauptstadt Potsdam
IF	Interference
IMEX	Import und Export von Daten zu Basisstationen
Interaktiv	Direktes visuelles Feedback (möglichst in Echtzeit)

LandExplorer	Bibliothek zur Visualierung von Geländemodellen und zusätzlicher Information, Abkürzung LDX
Messfahrt	Sequentielle Aufnahme von (Funknetz-)Daten entlang einer bestimmten Route
Metadaten	Zu einem Visualisierungsobjekt gehörige Informationen, die nicht zentraler Gegenstand der Visualisierung sind
NB	Handover Neighbourhood Relations, Relation, die beschreibt, zwischen welchen Basisstationen
OpenGL	Open Graphics Library, offener Industriestandard für die Entwicklung portabler, interaktiver 2D- und 3D-Grafikanwendungen
Pegasos2D	Bisherige Funknetzplanungssoftware der T-Mobile
PL	Path Loss, Dämpfung
PSD	Pegasos Spatial Data, Datenbank mit raumbezogenen, thematischen Daten zur Verwendung in Pegasos
Qt	Plattform-übergreifende Bibliothek für die Entwicklung grafischer Benutzerschnittstellen
Stadtmodell	Menge von dreidimensionalen Objekten, die eine virtuelle Repräsentation der realen Bauten und ausgewählter Vegetation (z. B. Bäume)
VIM	Visualization of Measurementdata
Visualisierung	Intuitive Darstellung großer, komplexer Datenmengen
Visualisierungsobjekt	Zu visualisierendes Element
Visualisierungstechnik	Methode zur Visualisierung
VRS	Virtual Rendering System, frei (open source) verfügbare Renderingbibliothek, entwickelt vom Lehrstuhl für Computer-Grafische Systeme am HPI

8.4 Autorenschaft

Alle drei Mitglieder des Geryon-Projekts haben an sämtlichen Teilen der Dokumentation gearbeitet. Allerdings lassen sich den Kapiteln Hauptautoren zuordnen:

Kapitel	Autor
Zielstellungen	Stephan Kirsch
Theoretische Grundlagen	Stephan Brumme
Grobe Softwarearchitektur	Stephan Kirsch, Haik Lorenz
Visualisierung	Stephan Brumme, Stephan Kirsch
Die Benutzerschnittstelle	Haik Lorenz
Ausblick und Weiterentwicklungen	Haik Lorenz
Der Entwicklungsprozess	Stephan Brumme

8.5 Abbildungsverzeichnis

Quellen

[1] www.pegasos.com

[2] T-Mobile, „Propagation in Macro and Microcells“, Pegasos-Dokumentation, S.39-40

[3] Christoph Stamm, „Algorithms and Software for Radio Signal Coverage Prediction in Terrains”, Dissertation 14283 der ETH Zürich, S. 90-92, www.inf.ethz.ch/personal/stamm/publications/stamm_thesis01.pdf, 2001

[4] G. Wölfle, R. Hoppe, F.M. Landstorfer, R.R. Collmann, „Vergleich deterministischer und empirischer Ausbreitungsmodelle für die Planung von Mikrozellen“, www.ihf.uni-stuttgart.de/forschung/pdf/woe98e_p.pdf

[5] Ari Viinikainen, „Digital Mobile Communication Systems”, Mandatory Assignment des Kurses TLI319 an der Universität Jyväskylä/Finnland, S. 2-3, www.mit.jyu.fi/arjuvi/opetus/tli319/Tli319_HE2_2001.pdf, 2001

[6] www.lx.it.pt/cost231/

[7] Coopération européenne dans le domaine de la recherche scientifique et technique, cost.cordis.lu/src/home.cfm, www.bbw.admin.ch/html/pages/forschung/cost/faq-d.html

[8] mitglied.lycos.de/radargrundlagen/grundlagen/db.html

[9] www.qsl.net/dj4uf/lehrg/a14/a14.htm

[10] www.wissen.de, Suchbegriff „Dezibel“

[11] www.elektrosmoginfo.de

[12] COST231, www.lx.it.pt/cost231/

[13] T-Mobile Deutschland GmbH, *Anforderungsdokument* Geryon

[14] T-Mobile, *Configuration Manual Pegasos*

[15] Geryon-Team, *„Visiondocument Geryon“*

[16] Scott McCloud, *Understanding Comics: The Invisible Art*, Harper Perenial, 1994

[17] http://www.vrs3d.org

[18] http://www.landex.org

[19] http://www.opengl.org

[20] http://www.nvidia.com und http://developer.nvidia.com

[21] Michael Garland und Paul S. Heckbert, *Surface Simplification Using Quadric Error Metrics*

[22] T-Mobile, *Short description of data structures for 3d-city-models*

[23] Michael Deering, *Geometry compression*, SIGGRAPH ´95 Proceedings, 1995

[24] http://www.gamers.org/dEngine/quake3/johnc_glopt.html

[25] http://www.3dlabs.com

[26] Michael Garland und Paul S. Heckbert, *Surface Simplification Using Quadric Error Metrics*

[27] T-Mobile, *Short description of data structures for 3d-city-models*

[28] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: *Design Patterns*, Addison Wesley, 1995

[29] Heide Balzert, *UML kompakt*, Spektrum, 2001

[30] Bernd Oestereich, *Objekorientierte Softwareentwicklung*, 4. Auflage, Oldenbourg Verlag, 1999

[31] Helmut Balzert, *Lehrbuch der Software-Technik*, Band 1 und 2, Spektrum Akademischer Verlag, 1998

[32] Glenn E. Krasner und Stephen T. Pope, *A cookbook for using the model-view controller user interface paradigm in Smalltalk-80*, Journal of Object-Oriented Programming, Aug. – Sep. 1988

[33] Helmut Herold: *Das Qt-Buch*, SuSE Press, 2001

[34] http://www.hpi.uni-potsdam.de/deu/forschung/cgs

[35] http://www.rational.com/products/rup/

[36] Philippe Kruchten, *The Rational Unfied Process – An Introduction, 2nd edition*, Addison-Wesley, 2000

[37] Wolfgang Zuser et al., *Software Engineering mit UML und dem Unified Process*, Pearson-Verlag, 2001

[38] Object Management Group, http://www.omg.org/uml

[39] Fundamental Modelling Concepts, http://www.fmc.hpi.uni-potsdam.de

[40] http://gcc.gnu.org

[41] http://www.trolltech.com

[42] http://www.cvshome.org

[43] http://www.wincvs.org

[44] http://subversion.tigris.org

[45] http://www.moregroupware.org/

[46] http://www.doxygen.org