Texturierung und Visualisierung virtueller 3D-Stadtmodelle

Dissertation
zur Erlangung des akademischen Grades
„doctor rerum naturalium“
(Dr. rer. nat.)
in der Wissenschaftsdisziplin Informatik

eingereicht an der
Mathematisch-Naturwissenschaftlichen Fakultät
der Universität Potsdam

von
Haik Lorenz, M.Sc.

Potsdam, den 10. November 2010

Online veröffentlicht auf dem
Publikationsserver der Universität Potsdam:
URL http://opus.kobv.de/ubp/volltexte/2011/5387/
URN urn:nbn:de:kobv:517-opus-53879
http://nbn-resolving.de/urn:nbn:de:kobv:517-opus-53879

Kurzfassung

Im Mittelpunkt dieser Arbeit stehen *virtuelle 3D-Stadtmodelle*, die Objekte, Phänomene und Prozesse in urbanen Räumen in digitaler Form repräsentieren. Sie haben sich zu einem Kernthema von Geoinformationssystemen entwickelt und bilden einen zentralen Bestandteil geovirtueller 3D-Welten. Zum einen werden virtuelle 3D-Stadtmodelle von Experten in Bereichen wie Stadtplanung, Funknetzplanung oder Lärmanalyse eingesetzt. Zum anderen verwenden auch allgemeine Nutzer virtuelle 3D-Stadtmodelle, häufig in Form von realitätsnah dargestellten virtuellen Städten in Bereichen wie Bürgerbeteiligung, Tourismus oder Unterhaltung. Entsprechende Anwendungen, z. B. GoogleEarth, ermöglichen die intuitive Erkundung einer räumlichen Umgebung und Erweiterung durch eigene 3D-Modelle oder zusätzliche Informationen.

Die Erzeugung und Darstellung virtueller 3D-Stadtmodelle besteht aus einer Vielzahl von Prozessschritten, von denen in der vorliegenden Arbeit zwei näher betrachtet werden: Texturierung und Visualisierung. Im Bereich der *Texturierung* werden Konzepte und Verfahren zur automatischen Ableitung von Fototexturen aus georeferenzierten Schrägluftbildern sowie zur Speicherung oberflächengebundener Daten in virtuellen 3D-Stadtmodellen entwickelt. Im Bereich der *Visualisierung* werden Konzepte und Verfahren für die multiperspektivische Darstellung sowie für die hochqualitative Darstellung nichtlinearer Projektionen virtueller 3D-Stadtmodelle in interaktiven Systemen vorgestellt.

Die *automatische Ableitung von Fototexturen aus georeferenzierten Schrägluftbildern* ermöglicht die Veredelung vorliegender virtueller 3D-Stadtmodelle durch Ergänzung flächendeckender Fototexturen. Schrägluftbilder bieten sich zur Texturierung an, da sie einen Großteil der Oberflächen einer Stadt, insbesondere Gebäudefassaden, mit hoher Redundanz erfassen. Das Verfahren extrahiert aus dem verfügbaren Bildmaterial alle Ansichten einer Oberfläche und fügt diese pixelpräzise zu einer Textur zusammen. Durch Anwendung auf alle Oberflächen wird das virtuelle 3D-Stadtmodell flächendeckend texturiert. Der beschriebene Ansatz wurde am Beispiel des offiziellen Berliner 3D-Stadtmodells sowie der in GoogleEarth integrierten Innenstadt von München erprobt.

Die *Speicherung oberflächengebundener Daten*, zu denen auch Texturen zählen, wird im Kontext von CityGML, einem international standardisierten Datenmodell und Austauschformat für virtuelle 3D-Stadtmodelle, untersucht. Es wird ein Datenmodell auf Basis computergrafischer Konzepte entworfen und in den CityGML-Standard integriert. Dieses Datenmodell richtet sich dabei an praktischen Anwendungsfällen aus und lässt sich domänenübergreifend verwenden.

Die *interaktive multiperspektivische Darstellung virtueller 3D-Stadtmodelle* ergänzt die gewohnte perspektivische Darstellung nahtlos um eine zweite Perspektive mit dem Ziel, den

Informationsgehalt der Darstellung zu erhöhen. Diese Art der Darstellung ist durch die Panoramakarten von H. C. Berann inspiriert; Hauptproblem ist die Übertragung des multiperspektivischen Prinzips auf ein interaktives System. Die Arbeit stellt eine technische Umsetzung dieser Darstellung auf Basis von 3D-Grafikhardware vor und demonstriert die Erweiterung von Vogel- und bodennaher Perspektive.

Die *hochqualitative Darstellung nichtlinearer Projektionen* beschreibt deren Umsetzung basierend auf 3D-Grafikhardware, wobei neben der Bildwiederholrate die Bildqualität das wesentliche Kriterium ist. Insbesondere erlauben die beiden vorgestellten Verfahren, *dynamische Geometrieverfeinerung* und *stückweise perspektivische Projektionen*, die uneingeschränkte Nutzung aller hardwareseitig verfügbaren, qualitätssteigernden Funktionen wie z. B. Bildraumgradienten oder anisotroper Texturfilterung. Beide Verfahren sind generisch und unterstützen verschiedene Projektionstypen. Sie ermöglichen die anpassungsfreie Verwendung gängiger computergrafischer Effekte wie Stilisierungsverfahren oder prozeduraler Texturen für nichtlineare Projektionen bei optimaler Bildqualität.

Die vorliegende Arbeit beschreibt wesentliche Technologien für die Verarbeitung virtueller 3D-Stadtmodelle: Zum einen lassen sich mit den Ergebnissen der Arbeit Texturen für virtuelle 3D-Stadtmodelle automatisiert herstellen und als eigenständige Attribute in das virtuelle 3D-Stadtmodell einfügen. Somit trägt diese Arbeit dazu bei, die Herstellung und Fortführung texturierter virtueller 3D-Stadtmodelle zu verbessern. Zum anderen zeigt die Arbeit Varianten und technische Lösungen für neuartige Projektionstypen für virtuelle 3D-Stadtmodelle in interaktiven Visualisierungen. Solche nichtlinearen Projektionen stellen Schlüsselbausteine dar, um neuartige Benutzungsschnittstellen für und Interaktionsformen mit virtuellen 3D-Stadtmodellen zu ermöglichen, insbesondere für mobile Geräte und immersive Umgebungen.

Abstract

This thesis concentrates on virtual 3D city models that digitally encode objects, phenomena, and processes in urban environments. Such models have become core elements of geographic information systems and constitute a major component of geovirtual 3D worlds. Expert users make use of virtual 3D city models in various application domains, such as urban planning, radio-network planning, and noise immision simulation. Regular users utilize virtual 3D city models in domains, such as tourism, and entertainment. They intuitively explore photorealistic virtual 3D city models through mainstream applications such as GoogleEarth, which additionally enable users to extend virtual 3D city models by custom 3D models and supplemental information.

Creation and rendering of virtual 3D city models comprise a large number of processes, from which texturing and visualization are in the focus of this thesis. In the area of *texturing*, this thesis presents concepts and techniques for automatic derivation of photo textures from georeferenced oblique aerial imagery and a concept for the integration of surface-bound data into virtual 3D city model datasets. In the area of *visualization*, this thesis presents concepts and techniques for multiperspective views and for high-quality rendering of nonlinearly projected virtual 3D city models in interactive systems.

The *automatic derivation of photo textures from georeferenced oblique aerial imagery* is a refinement process for a given virtual 3D city model. Our approach uses oblique aerial imagery, since it provides a citywide highly redundant coverage of surfaces, particularly building facades. From this imagery, our approach extracts all views of a given surface and creates a photo texture by selecting the best view on a pixel level. By processing all surfaces, the virtual 3D city model becomes completely textured. This approach has been tested for the official 3D city model of Berlin and the model of the inner city of Munich accessible in GoogleEarth.

The *integration of surface-bound data*, which include textures, into virtual 3D city model datasets has been performed in the context of CityGML, an international standard for the exchange and storage of virtual 3D city models. We derive a data model from a set of use cases and integrate it into the CityGML standard. The data model uses well-known concepts from computer graphics for data representation.

Interactive multiperspective views of virtual 3D city models seamlessly supplement a regular perspective view with a second perspective. Such a construction is inspired by panorama maps by H. C. Berann and aims at increasing the amount of information in the image. Key aspect is the construction's use in an interactive system. This thesis presents an approach to create multiperspective views on 3D graphics hardware and exemplifies the extension of bird's eye and ground-level views.

High-quality rendering of nonlinearly projected virtual 3D city models focuses on the implementation of nonlinear projections on 3D graphics hardware. The developed concepts and techniques focus on high image quality. This thesis presents two such concepts, namely *dynamic mesh refinement* and *piecewise perspective projections*, which both enable the use of all graphics hardware features, such as screen space gradients and anisotropic texture filtering under nonlinear projections. Both concepts are generic and customizable towards specific projections. They enable the use of common computer graphics effects, such as stylization effects or procedural textures, for nonlinear projections at optimal image quality and interactive frame rates.

This thesis comprises essential techniques for virtual 3D city model processing. First, the results of this thesis enable automated creation of textures for and their integration as individual attributes into virtual 3D city models. Hence, this thesis contributes to an improved creation and continuation of textured virtual 3D city models. Furthermore, the results provide novel approaches to and technical solutions for projecting virtual 3D city models in interactive visualizations. Such nonlinear projections are key components of novel user interfaces and interaction techniques for virtual 3D city models, particularly on mobile devices and in immersive environments.

Danksagungen

Eine Reihe von Personen hat maßgeblich zum Gelingen des Vorhabens „Promotion“ beigetragen, bei denen ich mich an dieser Stelle bedanken möchte. In ganz besonderem Maße gilt mein Dank Prof. Dr. Jürgen Döllner, der meine Forschung betreut und überhaupt ermöglicht hat.

Für die Bereitschaft zur Übernahme eines Gutachtens der Arbeit bedanke ich mich bei Prof. Dr. Thomas H. Kolbe (Technische Universität Berlin), Univ.-Prof. Dr. Georg Gartner (Technische Universität Wien) und Prof. Dr.-Ing. Monika Sester (Universität Hannover).

Meine aktuellen und ehemaligen Kollegen haben durch fachlichen Austausch wesentliche Unterstützung geleistet. Besonders hervorheben möchte ich Dr. Henrik Buchholz, der mich mit Wissen und Wortwitz durch die Promotionszeit begleitet hat, und Dr. Konstantin Baumann, der bei fachlichen Fragen immer die richtigen Tipps parat hatte. Zusätzlich möchte ich Matthias Trapp für die Zusammenarbeit bei den multiperspektivischen Projektionen und den kreativen Austausch danken. Mein Dank gilt natürlich auch den weiteren verdienten wie frischen Kollegen im Fachgebiet Computergrafische Systeme, auch wenn ich sie nicht namentlich nenne. Ein ganz besonderer Dank geht an Sabine Biewendt, die Sekretärin des Fachgebiets. Sie hält uns allen mit viel Herzlichkeit und Unermüdlichkeit den Kopf von bürokratischen Sorgen frei.

Ich bedanke mich bei der AG Modellierung der SIG 3D (GDI DE), die mir im Bereich CityGML einen wesentlichen Beitrag meiner Arbeit ermöglicht hat.

Bei meiner Arbeit hatte ich das große Glück, mit Datensätzen und Systemen aus der nicht-akademischen Welt arbeiten zu können. Die Daten wurden von den Firmen virtualcitySYSTEMS GmbH (Berlin), Berlin Partner GmbH (Berlin), COWI A/S (Silkebørg, DK) und Infoterra Ltd. (Leicester, UK) zur Verfügung gestellt. Der Firma Autodesk Inc. (Potsdam) danke ich für den Zugang zur LandXplorer-Plattform als Grundlage meiner Arbeit. Darüber hinaus haben die fachlichen Diskussionen mit Mitarbeitern zur Weiterentwicklung meiner Arbeit beigetragen. Namentlich seien Chris Andrews (Autodesk Inc.), Thomas Woge (virtualcitySYSTEMS GmbH) und Rasmus Lindeneg Johansen (COWI A/S) genannt.

Nicht zuletzt möchte ich mich bei meiner Familie bedanken. Bei meinen Eltern Angelika und Hilmar für die großartige Unterstützung und Förderung; sie sind mir ein beständiges Vorbild in einer bewegten Zeit. Bei meiner Frau Alexandra für fortwährende konstruktive fachliche und künstlerische Kritik genauso wie für die Sicherstellung eines Lebens abseits der Promotion. Und bei meiner Tochter Lucia, die zwar noch nicht fachlich mitwirken konnte, aber die Fertigstellung der Arbeit maßgeblich beschleunigt hat.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Programmausschnittverzeichnis

1 Einleitung

Im Mittelpunkt der vorliegenden Arbeit stehen virtuelle 3D-Stadtmodelle, die Objekte, Phänomene und Prozesse in urbanen Räumen in digitaler Form repräsentieren. Solche virtuellen 3D-Stadtmodelle haben sich zu einem Kernthema von Geoinformationssystemen (GIS) entwickelt (Zlatanova, 2000; Zlatanova u. a., 2002) und bilden einen zentralen Bestandteil geovirtueller 3D-Welten (Döllner und Buchholz, 2005). Virtuelle 3D-Stadtmodelle werden dabei nicht nur in Systemen für Experten verwendet, sondern auch zum Beispiel in GoogleEarth (Google, Inc., 2010), das in einem virtuellen Globus (Höffken, 2009) neben anderen Elementen komplexe, mit vielfältigen Informationen angereicherte virtuelle 3D-Stadtmodelle verbindet und für allgemeine Nutzer auf einer Vielzahl von Geräten mit Internetverbindung zugänglich macht. Dabei gehen derartige Anwendungen virtueller 3D-Stadtmodelle weit über einfache Visualisierungen hinaus und erlauben GIS-typische raumbezogene Anfragen wie Abstandsmessungen, Routenplanung oder die umkreisbezogene Suche von Objekten anhand von Attributen. Ein Beispiel für ein flächenmäßig großes und inhaltlich stark ausformuliertes virtuelles 3D-Stadtmodell ist das amtliche Berliner Stadtmodell (Berlin Partner GmbH, 2009; Kada, 2009). Es basiert auf der amtlich geführten automatischen Liegenschaftskarte (ALK) und enthält neben der dreidimensionalen Form und detaillierten Erscheinung von Gebäuden auch gebäudebezogene thematische Daten wie z. B. solare Eignung oder Verfügbarkeit als Gewerbeimmobilie. Hauptanwendungen des Berliner Stadtmodells liegen in den Bereichen Stadtentwicklung und Wirtschaftsförderung. Darüber hinaus haben virtuelle 3D-Stadtmodelle allgemein vielfältige Anwendungen in Bereichen wie Stadtplanung, Bürgerbeteiligung, Funknetzplanung, Lärmanalyse, Gefahrenabwehr, Katastrophenschutz, Geschichtsdokumentation, Tourismus oder Unterhaltung.

Die konkrete Ausprägung eines 3D-Stadtmodells hängt dabei von den Anforderungen und Zielsetzungen der jeweiligen Anwendung ab. Für Zwecke der Präsentation im Rahmen von Planung, Dokumentation oder Bürgerbeteiligung können physische, also direkt anfassbare und erlebbare 3D-Stadtmodelle sinnvoll sein (Höffken, 2009). Solche oft aus Holz oder Gips gefertigten 3D-Stadtmodelle lassen sich jedoch nur eingeschränkt bzw. mit hohem Aufwand, z. B. durch Austausch einzelner Gebäude oder ganzer Kacheln, verändern und kaum für andere Anwendungen einsetzen. Die Anwendungsmöglichkeiten virtueller 3D-Stadtmodelle sind dagegen wie oben aufgeführt breit gefächert, da diese Modelle eine auf die jeweilige Anwendung abgestimmte Menge relevanter Informationen in Form digitaler raumbezogener Daten, sogenannter Geodaten (Bollmann und Koch, 2005), beinhalten können. Der Raumbezug wird entweder explizit durch Angabe der dreidimensionalen geografischen Lage in einem Koordinaten- oder

Adresssystem oder implizit durch Bezug zu einem Objekt mit explizitem Raumbezug hergestellt. Der Begriff des virtuellen 3D-Stadtmodells setzt sich aus mehreren Teilen zusammen:

Modell: bezeichnet eine Abstraktion und Beschreibung konkreter Objekte, Phänomene und Prozesse der realen Welt; im vorliegenden Fall durch entsprechende Datenobjekte mit jeweiligen spezifischen Attributen (Bollmann und Koch, 2005). Es ist keine Beschreibung von Wirkungszusammenhängen im Sinne mathematischer Formeln oder Algorithmen. Es ist nicht mit der Beschreibung der Struktur der Datenobjekte (Datenmodell) gleich zu setzen.

Stadt: das Modell bildet räumliche Bereiche ab, in denen menschliche Einflussnahme, z. B. durch Bauwerke oder Infrastruktur, vorhanden ist. Solche urbanen Räume setzen sich aus typischen Komponenten wie z. B. Gebäuden, Straßen oder Vegetation zusammen (Gröger u. a., 2008).

3D: wesentliches Kennzeichen ist eine geometrische Repräsentation in einem definierten dreidimensionalen Koordinatensystem (Cooper, 2008) als Bezugsgrundlage aller erfassten Aspekte der Stadt.

virtuell: wird hier als Abgrenzung zu „physisch“ gebraucht. Ein virtuelles Modell liegt in Form strukturierter digitaler Daten vor und ist nur durch geeignete computergestützte Ausgabegeräte, Ausgabetechniken und Interaktionstechniken erlebbar. Die Bandbreite der möglichen Darstellungen reicht von abstrakten, tabellenhaften Darstellungen über kartenhafte 2D-Darstellungen bis hin zu interaktiven 3D-Darstellungen. In immersiven geovirtuellen Umgebungen mit fotorealistischer Ausgabe können sie den Eindruck einer realen Welt vermitteln (Fuhrmann und MacEachren, 2001).

Im weitesten Sinne ist ein virtuelles 3D-Stadtmodell ein komplexer Datensatz, der Informationen zu real existierenden oder imaginären, im geografischen Raum angesiedelten Objekten, Phänomenen und Prozessen kodiert. Die Komplexität und Ausdehnung eines virtuellen 3D-Stadtmodells ist dabei nicht gleichmäßig in allen drei Dimensionen, da die repräsentierten geografischen Räume typischerweise auf, in geringer Höhe über und geringer Tiefe unter der Erdoberfläche liegen. Damit ist die Komplexität parallel zur Erdoberfläche (horizontal) deutlich größer als orthogonal dazu (vertikal). Die klassische Kartografie nutzt diese Eigenschaft, indem sie die gleichen geografischen Räume mittels einer orthografischen Abbildung auf zwei Dimensionen reduziert darstellt und einfache Mehrdeutigkeiten in der Vertikalen, z. B. bei Brücken oder Tunneln, durch eindeutige grafische Gestaltung auflöst. Für Darstellungen von Geländeoberflächen, Verkehrsnetzen oder georeferenzierten demografischen Daten ist diese Art der Repräsentation typischerweise ausreichend. Dieser Ansatz ist nicht mehr ausreichend, wenn beispielsweise

- die Gebäudeform relevant ist (z. B. für 3D-Darstellungen oder bei der Prüfung für solare Eignung),
- Informationen in der Vertikalen feingranular benötigt werden (z. B. stockwerksgenau oder raumgenau),
- eine größere Komplexität in der Vertikalen berücksichtigt werden muss (z. B. Verkehrswege in Megacities) oder
- komplexe dreidimensionale Strukturen zusammengeführt werden sollen (z. B. mehrere Versorgungsnetzwerke).

In diesen beispielhaften Fällen muss die dritte Dimension explizit erfasst sein und somit ein virtuelles 3D-Stadtmodell verwendet werden. Es gilt jedoch weiterhin die für diese Arbeit zentrale Annahme, dass die Komponenten einer Stadt topologischen hauptsächlich in einer Ebene angeordnet sind (Buchholz, 2006). Zeitlich veränderliche Geodaten wie tageszeitabhängige Verkehrsaufkommen können auch Teil virtueller 3D-Stadtmodelle sein, werden in dieser Arbeit jedoch nicht gesondert betrachtet.

Die Erzeugung und Nutzung virtueller 3D-Stadtmodelle setzt sich typischerweise aus einer Kette klar unterscheidbarer und getrennter Prozessschritte zusammen. Im Allgemeinen ist diese Prozesskette iterativ und besteht aus Erfassung bzw. Fortführung, Qualifikation, Veredelung, Zusammenführung, Spezialisierung und fachspezifischer Nutzung (Kolbe u. a., 2009). Die Nutzung schließt häufig eine Visualisierung mit ein. Im Verlauf der Prozessierung erhöht sich die Wertigkeit des virtuellen 3D-Stadtmodells, wobei die Anforderungen an die Art, Qualität und Menge der Daten in einem virtuellen 3D-Stadtmodell anwendungsabhängig sind. Die Prozesskette erstreckt sich über verschiedene Domänen, Beteiligte und Systeme. Zur Realisierung werden zunehmend Geodateninfrastrukturen (GDI bzw. engl. Spatial Data Infrastructure, SDI) eingesetzt (Bill, 1999). Ein möglichst verlust- und reibungsfreier Datenaustausch ist hierbei essentiell. Kern eines solchen Datenaustausches sind klar definierte Schnittstellen und Datenformate, sowohl im Sinne der Datenstrukturen als auch in deren Bedeutung. Die international anerkannte Institution zur Definition von Schnittstellen und Datenformaten im Geoinformationsbereich ist das OpenGeospatialConsortium (OGC), ein Zusammenschluss von Firmen, Institutionen und Privatpersonen mit dem Ziel, Interoperabilität für Geodaten sicherzustellen. Das Leitbild der OGC ist die Abbildung der Prozesskette auf Webservices (Open Geospatial Consortium, 2008), deren Kommunikations- und Datenschnittstellen offenen Standards genügen.

Aus der angesprochenen Prozesskette ergibt sich eine Vielzahl interessanter Forschungsfragen. Diese Arbeit konzentriert sich auf zwei ausgewählte Aspekte:

Veredelung: bezeichnet die Ergänzung, Vervollständigung oder Homogenisierung von Daten. In dieser Arbeit wird die Ergänzung eines virtuellen 3D-Stadtmodells um reale, aus Schrägluftbildern abgeleitete Texturen sowie deren Speicherung beschrieben.

Visualisierung bzw. Geovisualisierung: bezeichnet in der für diese Arbeit relevanten Ausprägung die Erzeugung von Bildern zum Zweck der Vermittlung bestimmter Sachverhalte. In dieser Arbeit wird speziell auf die zur Bilderzeugung eingesetzte Projektion eingegangen. Es werden die standardmäßige und technisch optimal unterstützte perspektivische Projektion hinterfragt und multiperspektivische sowie nichtlineare Darstellungen zusammen mit ihrer technischen Umsetzung als Alternative vorgestellt.

1.1 Beitrag und Aufbau der Arbeit

Diese Arbeit konzentriert sich auf die Texturierung und Visualisierung als wesentliche Technologien für die Entwicklung innovativer Systeme und Anwendungen virtueller 3D-Stadtmodelle. Dafür werden jeweils Konzepte erarbeitet und Implementierungen zu deren Überprüfung bereitgestellt. Nach einer Einführung relevanter Grundlagen und der Einordung der vorliegenden Arbeit in Kap. 2 werden in den darauf folgenden Kapiteln die Hauptbeiträge vorgestellt:

1. Ein Konzept und Verfahren zur automatischen Texturierung virtueller 3D-Stadtmodelle aus Schrägluftbildern (Kap. 3).

2. Ein Konzept zur Speicherung oberflächengebundener Daten in virtuellen 3D-Stadtmodellen, das in den CityGML-Standard integriert wurde (Kap. 4).
3. Ein Konzept und Verfahren zur multiperspektivischen Darstellung virtueller 3D-Stadtmodelle, das den Informationsgehalt in Stadtmodellansichten erhöht (Kap. 5).
4. Ein computergrafisches Konzept zur qualitativ hochwertigen Erzeugung nichtlinearer Darstellungen in interaktiven Systemen und Verfahren zur effizienten Umsetzung auf 3D-Grafikhardware (Kap. 6 und 7).

Jedes Kapitel bildet eine Einheit, in der jeweils das Konzept vorgestellt, verwandte Arbeiten besprochen, die technische Umsetzung beschrieben, Anwendungsbeispiele präsentiert sowie Ergebnisse aufgelistet und kritisch diskutiert werden. Kap. 8 fasst die Arbeit zusammen und zeigt Anknüpfungspunkte für zukünftige Arbeiten auf.

Automatische Texturierung virtueller 3D-Stadtmodelle aus Schrägluftbildern

Konzept und Verfahren konzentrieren sich auf die Zusammenführung heterogener Daten, indem sie ein geometrisch vorliegendes virtuelles 3D-Stadtmodell um Texturen ergänzen, die aus dazu räumlich passenden Schrägluftbildern extrahiert werden. Die wesentlichen Kennzeichen sind:

- Texelpräzise Wahl der Quellbilder: Aus allen verfügbaren Ansichten einer Fläche wird pro Texel die optimale ausgewählt.
- Hoher Automatisierungsgrad: Nach einer manuellen Aufbereitung der Eingabedaten läuft die Extraktion der Texturen vollautomatisch ab.
- Hohe Flexibilität: Es werden gängige Ein- und Ausgabeformate einschließlich CityGML unterstützt.
- Praxistauglichkeit: Das Verfahren stellt Werkzeuge zur Kontrolle und Korrektur fehlerhafter Eingabedaten sowie der Ergebnistexturen in einem definierten Arbeitsablauf bereit. Es eignet sich auch für große Datensätze massiver virtueller 3D-Stadtmodelle.

Das Verfahren wird mittlerweile von mehreren Firmen aktiv zur Veredelung virtueller 3D-Stadtmodelle eingesetzt.

Speicherung oberflächengebundener Daten in virtuellen 3D-Stadtmodellen

Das Konzept umfasst ein flexibles, an praktisch relevanten Nutzungsbeispielen ausgerichtetes Datenmodell für die Speicherung und den Transport oberflächengebundener Daten in virtuellen 3D-Stadtmodellen. Dazu wird auf das Material- und Texturkonzept aus der Computergrafik zurückgegriffen, um die Daten als neuartige, feingranulare thematische Daten in das virtuelle 3D-Stadtmodell einzubinden. Ein Hauptzweck oberflächengebundener Daten ist die realitätsnahe Darstellung virtueller 3D-Stadtmodelle mit Hilfe aus Bildern abgeleiteter Texturen. Das Datenmodell ist jedoch nicht auf diesen Zweck beschränkt, sondern kann auch zur Repräsentation beliebiger oberflächengebundener Daten herangezogen werden. Wesentliche Eigenschaften sind:

- Kompatibilität mit GML: Die Erweiterung zentraler GML-Klassen wird vermieden, um die Abwärtskompatibilität trotz vorhandener oberflächenbezogener Daten sicher zu stellen.

- Klare Abgrenzung zur Präsentationsspezifikation: Das Datenmodell beinhaltet ausschließlich oberflächenbezogene Daten. Es werden keine Annahmen zu deren Verwendung in Visualisierungen getroffen und keine diesbezüglichen Einschränkungen impliziert.
- Fokussierung auf praktisch relevante Anwendungsfälle: Die Grundanforderungen an das Datenmodell wurden durch sechs in der Praxis wichtige Anwendungsfälle definiert.

Das Datenmodell ist in den internationalen Standard CityGML integriert und wird aktiv genutzt. Es wurde in Gröger u. a. (2008) und Lorenz und Döllner (2008b) beschrieben.

Multiperspektivische Darstellung virtueller 3D-Stadtmodelle

Die Arbeit klassifiziert perspektivische Ansichten virtueller 3D-Stadtmodelle und analysiert sie bezüglich ihres Orientierungswertes und der Erkennbarkeit von Objekten. Daraus abgeleitet werden zwei multiperspektivische Darstellungsvarianten vorgestellt, die Ansichten der einzelnen Klassen um zusätzliche Elemente ergänzen. Für die Erzeugung wird ein für die Implementierung auf der 3D-Grafikhardware optimiertes, echtzeitfähiges Verfahren basierend auf globalen Deformationen beschrieben. Es zeichnet sich durch folgende Eigenschaften aus:

- Effektivität: Die resultierenden Abbildungen verbinden den potentiell hohen Orientierungswert ausgewählter dreidimensionaler Darstellungen mit dem Informationswert kartenhafter Darstellungen.
- Interaktivität: Das Verfahren lässt sich auf der 3D-Grafikhardware implementieren und erreicht so für interaktive Anwendungen ausreichende Bildwiederholraten.
- Universalität: Das Verfahren lässt sich in bestehende Geovisualisierungssysteme integrieren und schränkt die Gestaltungsmöglichkeiten bzw. darstellbaren Informationen nicht ein.

Die multiperspektivischen Ansichten wurden prototypisch in die Autodesk LandXplorer-Plattform integriert und erstmals in Lorenz u. a. (2008) vorgestellt.

Hochwertige nichtlineare Darstellungen mit 3D-Grafikhardware

Das Konzept approximiert eine Klasse nichtlinearer Projektionen und ist optimal für die Umsetzung auf 3D-Grafikhardware geeignet. Kern der stückweisen perspektivischen Projektionen ist die Zerlegung der Projektion in nichtüberlappende Teile, die jeweils eine individuelle perspektivische Projektion verwenden. Das Konzept arbeitet somit ausschließlich mit perspektivischen Projektionen, welche optimal von 3D-Grafikhardware berechnet werden können. Daraus ergeben sich folgende Eigenschaften:

- Direkte Bilderzeugung: Es ist kein Zwischenbild und keine Verzerrung desselben nötig.
- Optimale Qualität: Alle Hardwaremerkmale für höchste Qualität wie anisotrope Texturfilterung, Gradientenoperationen oder Multisample-Antialiasing arbeiten wie unter einer regulären perspektivischen Projektion.
- Universalität: Es werden keine Annahmen über die Modellgeometrie getroffen; insbesondere werden keine topologischen Informationen benötigt. Bei unsauberer Geometrie erzeugen stückweise perspektivische Projektionen keine zusätzlichen Artefakte im Vergleich zu regulären perspektivischen Projektionen.

- Einfache Verwendung: Darstellungseffekte für perspektivische Darstellungen, auch nichtfotorealistische Stilisierung oder prozedurale Texturen, können ohne Änderung und ohne Auswirkungen auf den visuellen Eindruck direkt für nichtlineare Darstellungen übernommen werden.

Für die Umsetzung des Konzepts wurden Verfahren zur Replikation von Grafikprimitiven auf der 3D-Grafikhardware für verschiedene Hardwaregenerationen mit den jeweils vorhandenen Merkmalen entwickelt. Die Verfahren sind unabhängig von der stückweise perspektivischen Projektion auch für die Umsetzung anderer replikationsbasierter Techniken, z. B. dynamische Geometrieinstantiierung, verwendbar. Die stückweise perspektivischen Projektionen wurden in Lorenz und Döllner (2009) und Lorenz und Döllner (2010b) vorgestellt.

Zusätzlich wurde ein effizientes und universelles Verfahren zur Verfeinerung von Dreiecken auf älterer 3D-Grafikhardware entwickelt, das als Ersatz für die auf neuester 3D-Grafikhardware vorhandene Verfeinerungseinheit dienen kann. Dieses Verfahren wurde in Lorenz und Döllner (2008a) beschrieben.

2 Grundlagen

Dieses Kapitel beschreibt wesentliche Grundlagen der vorliegenden Arbeit und ordnet die entwickelten Verfahren und Konzepte darin ein.

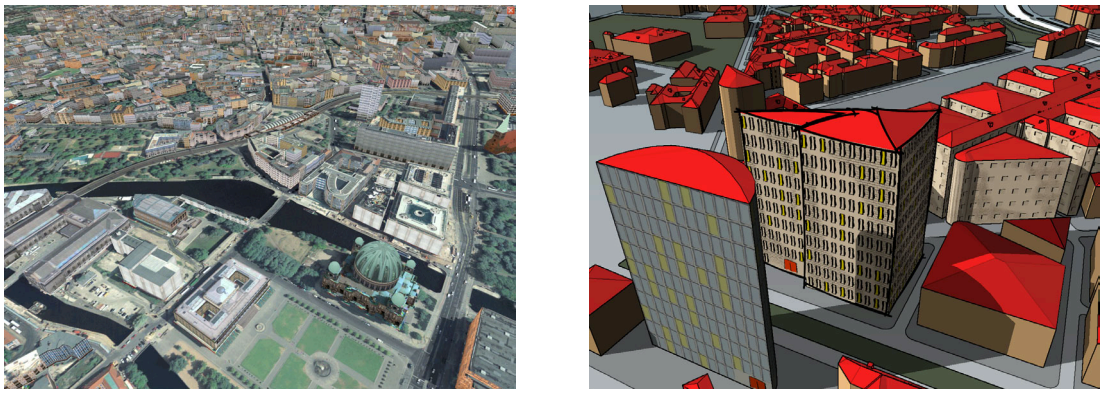
2.1 CityGML

CityGML (Gröger u. a., 2008) ist ein offenes, XML-basiertes Speicher- und Austauschformat für virtuelle 3D-Stadtmodelle. Es ist als Anwendungsschema der Geography Markup Language (GML, Portele, 2007) des Open Geospatial Consortium implementiert. Ziel von CityGML ist die Definition grundlegender Entitäten der Modelle urbaner Räume mit ihren Attributen und Beziehungen. Durch die standardisierte Beschreibung wird Kompatibilität in heterogenen Umgebungen ermöglicht und damit die Verwendung virtueller 3D-Stadtmodelle in verschiedenen Fachgebieten und Anwendungsfeldern vereinfacht sowie die systematische Konstruktion, Fortführung und Zusammenführung komplexer virtueller 3D-Stadtmodelle unterstützt.

CityGML beschreibt nicht nur die geometrische Form, sondern auch explizit semantische themenbezogene Informationen eines virtuellen 3D-Stadtmodells. Es unterscheidet dazu zwischen einem geometrischen und einem thematischen Modell, die beide kohärent verbunden sind. Damit unterstützt CityGML das gesamte Spektrum virtueller 3D-Stadtmodelle, begonnen bei rein geometrischen Modellen für die grafische Darstellung über für spezielle Aufgaben erzeugte Modelle bis hin zu „intelligenten“, geometrisch-semantisch voll ausmodellierten und kohärenten sowie mit fachspezifischen Daten angereicherten Modellen für verschiedene Anwendungszwecke (Stadler und Kolbe, 2007).

Das *geometrische Modell* beinhaltet die konsistente und homogene Beschreibung räumlicher Eigenschaften, sowohl bezüglich Geometrie als auch Topologie, aller Objekte in einem virtuellen 3D-Stadtmodell auf Basis von GML. Es ermöglicht auch die Verwendung von Prototypen für häufig auftretende Objekte gleicher Form (z. B. Bäume).

Das *thematische Modell* differenziert zwischen Geländemodellen, Gebäuden, Vegetation, Gewässer, Verkehrsräumen und Stadtmöbeln. Weiterhin gibt es die Möglichkeit, noch nicht von CityGML explizit erfasste Objekt- bzw. Informationskategorien als generische Objekte oder Attribute abzulegen. Jede Kategorie bildet ein eigenes Modul, das nur von einem Kernmodul abhängig ist. Anwendungen brauchen nur die für sie interessanten Module implementieren. Klassen des thematischen Modells sind über Fachschalen (Application Domain Extension, ADE) für konkrete Anwendungen strukturiert erweiterbar. Objekte lassen sich beliebig, auch rekursiv,



(a) Fotorealistische Darstellung.

(b) Skizzenhafte Darstellung (aus Döllner u. a., 2005).

Abb. 2.1: Zwei Beispiele texturierter virtueller 3D-Stadtmodelle.

gruppieren um hierarchische Beziehungen zwischen den Objekten auszudrücken. Besonderheiten stellen die Konzepte der ClosureSurfaces (Verschlussflächen) und TerrainIntersectionCurves (Geländeschnittlinien) dar. *ClosureSurfaces* werden dazu verwendet, geometrisch nicht geschlossene Volumen (z. B. Tunnel) virtuell zu verschließen und so Analysen wie Volumenberechnungen zu ermöglichen. *TerrainIntersectionCurves* definieren die exakte Verbindung zwischen einem Objekt und dem Gelände. Damit kann zum einen die grafische Darstellung aufgewertet und zum anderen die konsistente Zusammenführung verschiedener Datensätze ermöglicht werden.

Kernbestandteil von CityGML ist die Unterscheidung von fünf Detail- und Genauigkeitsstufen (Level of Detail, LoD). LoDs beziehen sich dabei nicht nur auf das Geometriemodell, sondern auch auf das thematische Modell. Jedes thematische Objekt kann für jedes LoD parallel eine eigene geometrische Repräsentation beinhalten. Generalisierungsbeziehungen zwischen Objekten können explizit angegeben werden. LoD 0 definiert dabei im Wesentlichen ein 2,5D-Geländemodell, LoD 1 Blockmodelle, LoD 2 Blockmodelle mit korrekter Dachform, LoD 3 detaillierte Modelle und LoD 4 detaillierte Modelle mit ausdifferenziertem Innenraum. In allen LoDs kann die Geometrie thematisch differenzierten Oberflächen (z. B. Wand, Dach) zugeordnet werden.

2.2 Texturierung und Texturen

Texturierung dient derzeit im Bereich virtueller 3D-Stadtmodelle hauptsächlich der grafischen Gestaltung, üblicherweise zur Vermittlung eines fotorealistischen Eindrucks (Buchholz, 2006). Die verwendeten Texturen werden meist aus Fotos abgeleitet und erzeugen auf Grund ihrer Detailierung ein glaubwürdiges Erscheinungsbild (Abb. 2.1(a)). Teilweise wird Texturierung auch im Rahmen nichtfotorealistischer Darstellungen (Döllner u. a., 2005) verwendet, wobei mit computergrafischen Verfahren abstrahierte Texturen generiert werden, um beispielsweise einen skizzenhaften Gesamteindruck zu erzeugen (Abb. 2.1(b)).

Texturierung bezeichnet „ein Verfahren zur effizienten Modellierung von Oberflächeneigenschaften“ (Akenine-Möller u. a., 2008) und ist ein „grundlegendes computergrafisches Primitiv“ (Haeberli und Segal, 1993). Anschaulich werden in Form eines 2D-Rasters vorliegende Daten

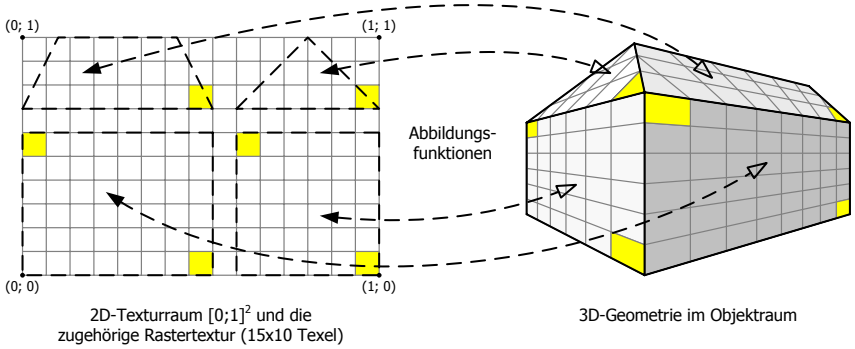


Abb. 2.2: Texturierung einer Oberfläche mittels eines 2D-Rasters (*Textur*). Mit Hilfe frei definierbarer, üblicherweise nicht umkehrbarer Abbildungsfunktionen werden Oberflächenpunkten Punkte im kanonischen Texturraum $[0;1]^2$ zugeordnet (*Texturparametrisierung*). Im Falle eindeutiger Texturparametrisierung sind die Abbildungsfunktionen umkehrbar. Die Textur überdeckt unabhängig von der Rasterauflösung den gesamten kanonischen Texturraum.

(*Textur*) auf eine Oberfläche abgebildet (Abb. 2.2) und zur Modifikation der Oberflächendarstellung eingesetzt. Der Texturierungsbegriff besitzt zwei Facetten:

Texturauswertung: Die Bestimmung eines Texturwertes zu einem gegebenen Pixel während der Darstellung, d. h. die Auswertung der Abbildungsvorschrift.

Texturdefinition: Die Erzeugung von Texturen und Spezifikation der Abbildungsvorschriften für gegebene Oberflächen, d. h. die Definition eines texturierten 3D-Modells.

Die Texturauswertung wird in Akenine-Möller u. a. (2008) detailliert beschrieben. In der vorliegenden Arbeit wird die Texturdefinition betrachtet. Dafür sind die drei Elemente Oberfläche, Textur und Abbildungsvorschrift relevant. Die Oberfläche wird durch ein 3D-Modell vorgegeben und liegt im *3D-Objektraum*. Die Textur ist eine Funktion mit einem eigenen Definitionsbereich, dem *Texturraum*, der meist ein bis drei Dimensionen besitzt und dem jeweiligen Einheitswürfel entspricht. Die Funktionswerte sind diskret in Form eines entsprechenden, aus *Texeln* bestehenden Rasters gespeichert oder mittels einer Prozedur berechenbar. Die Abbildungsfunktion beschreibt die Abbildung vom Objektraum in den Texturraum (*Texturparametrisierung*). Die Abbildung wird über Stützstellen (*Texturkoordinaten*) und geeignete Interpolation oder mittels parametrisierter Funktionen beschrieben. Ist die Abbildung bijektiv, d. h. umkehrbar, spricht man von *eindeutiger Texturparametrisierung* (Buchholz, 2006).

Die Texturierung wird in der Computergrafik als generisches Werkzeug eingesetzt (Haeberli und Segal, 1993). Dementsprechend werden Anwendungsmöglichkeiten und Varianten im Rahmen vielfältiger Forschungsarbeiten ausgelotet und technologisch untersucht. Mit der Textursynthese und Texturparametrisierung gibt es beispielsweise zwei Forschungszweige, die sich mit speziellen Aspekten der Texturdefinition beschäftigen. Im Rahmen der Geoinformatik wird die Texturierung hauptsächlich zur Kodierung der äußeren Erscheinung bzw. Feinstruktur realer oder geplanter Objekte und damit als Hilfsmittel zur Modellierung eingesetzt (Göbel und Freiwald, 2008). Für diesen Anwendungsfall lassen sich die generisch definierten Begriffe Textur und Abbildungsvorschrift deutlich einschränken. Im Folgenden werden o. B. d. A. 2D-Raster, die Farben kodieren, als Texturen angenommen; als Abbildungsvorschrift kommen Texturkoordinaten oder affine Abbildungen zum Einsatz. Die zentrale Eigenschaft einer texturierten Oberfläche ist in der Geoinformatik ihre metrische Texturauflösung, beschrieben durch die Größe eines

Texels im Objektraum. Sie bestimmt die Eignung für verschiedene Betrachtungsabstände, d. h. den Abstand der virtuellen Kamera von einem Objekt, und damit die Bandbreite möglicher Visualisierungsnutzungen.

Texturen lassen sich nach Entstehungsart und Verwendung klassifizieren (Göbel und Freiwald, 2008). Bzgl. der Entstehungsart wird zwischen Fototexturen, synthetischen Texturen und prozeduralen Texturen unterschieden. *Fototexturen* sind aus fotografischen Aufnahmen abgeleitet, wobei der fotografische Eindruck erhalten bleibt, *synthetische Texturen* werden manuell gezeichnet oder durch Verfremdung aus Fotos erzeugt, wobei ein stilisierter Eindruck erzeugt wird, und *prozedurale Texturen* werden algorithmisch auf Basis von Parametern bestimmt. Die Trennung zwischen Fototexturen und synthetischen Texturen ist unscharf, insbesondere bei Ableitung beider aus Fotos unter Verwendung von Methoden der Bildbearbeitung. Bzgl. der Verwendung wird zwischen objektspezifischen und prototypischen Texturen unterschieden. *Objektspezifische Texturen* werden für eine Fläche individuell erfasst und geben die wirkliche Erscheinung wieder. *Prototypische Texturen* generalisieren und typisieren Familien von Flächen. Sie werden evtl. auf Basis von Parametern an die jeweilige Fläche angepasst und nähern die wirkliche Erscheinung an.

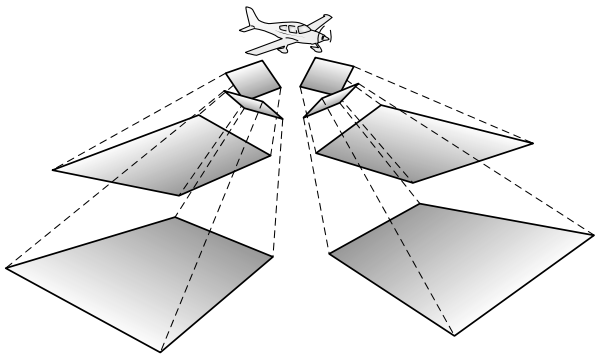
Das in dieser Arbeit vorgestellte Verfahren zur automatischen Texturierung virtueller 3D-Stadtmodelle (Kap. 3) erzeugt flächendeckende objektspezifische Fototexturen. Für das Ergebnis dieser Fototexturierung ist die Qualität des Bildmaterials entscheidend. Einen wesentlichen Anteil an der Qualität hat die Abbildungsschärfe (Kraus, 1997), die eng mit dem Auflösungsbegriff verbunden ist. In der Digitalfotografie sind dabei verschiedene Ausprägungen zu unterscheiden, für die es in der Literatur keine einheitliche Nomenklatur gibt. In dieser Arbeit werden unterschieden:

Physische Auflösung: Anzahl der Abtastpunkte (Pixel / Texel) eines Bildes, typischerweise als Anzahl Spalten mal Anzahl Zeilen gegeben.

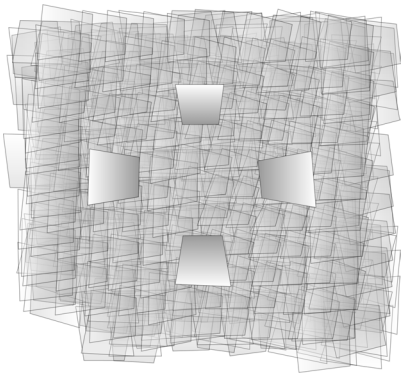
Metrische Auflösung: Abmessungen eines Pixels im Objektraum bezogen auf eine konkrete Oberfläche. Die überdeckte Fläche wird typischerweise durch ein Rechteck approximiert (z. B. $10\times10\text{cm}^2$ pro Pixel). Bei quadratischen Flächen wird nur eine Kantenlänge (z. B. 10cm pro Pixel) angegeben.

Effektive Auflösung: Maximale Anzahl unter praktischen Bedingungen im Bild unterscheidbarer Details, in der Fotografie in Linienpaaren pro mm, in der digitalen Bildverarbeitung in Linien pro Bildhöhe gemessen. Dieser Begriff berücksichtigt das optische Auflösungsvermögen des bildgebenden Systems (begrenzt bspw. durch Tiefenschärfe und Beugungsunschärfe) und nachfolgende Bildtransformationen. Die effektive Auflösung kann nicht höher als die Zeilenanzahl der physischen Auflösung sein.

Das in Kap. 4 entworfene Datenmodell zur Integration oberflächengebundener Daten in virtuelle 3D-Stadtmodelle orientiert sich in Bezug auf Texturen an den Anforderungen der Geoinformatik. Im Unterschied zu computergrafisch motivierten Datenmodellen und Austauschformaten wie z. B. X3D (ISO/IEC FDIS 19775-1:2008, 2008) und Collada (Barnes und Finch, 2001) konzentrieren sich die Betrachtungen auf den oben genannten vereinfachten Texturierungsfall. Komplexere Aspekte der Texturierung wie 3D- oder prozedurale Texturen oder frei definierte Abbildungsfunktionen werden aus praktischen Erwägungen nicht berücksichtigt.



(a) Sichtvolumina eines Schrägluftbildsystems mit 4 Kameras. Die Trapeze stellen die Schnitte mit der Erdoberfläche dar.



(b) Beispielhafte Gebietsabdeckung. Vier zu einem Zeitpunkt aufgenommene Bilder sind hervorgehoben.

Abb. 2.3: Aufnahmeprinzip eines Schrägluftbildsystems mit vier Kameras. Zu einem Zeitpunkt werden vier Aufnahmen erfasst: in Flugrichtung vorn, hinten, rechts und links (Abb. 2.3(a)). Eine Befliegung mit hoher Überlappung führt zu einer lückenlosen Abdeckung eines Gebiets für alle vier Aufnahmerichtungen (Abb. 2.3(b)).

2.2.1 Schrägluftbilder

Im Allgemeinen erzeugen aktuelle Erfassungssysteme für Schrägluftbilder neben klassischen *Nadiraufnahmen*, d. h. Aufnahmen mit vertikaler Blickrichtung, *Schrägaufnahmen* für vier Richtungen: relativ zur Flugrichtung nach vorn, hinten, links und rechts (Abb. 2.3). Für die Weiterverarbeitung ist eine *Georeferenzierung*, d. h. Lage und Orientierung der Bilder im Raum (äußere Orientierung, Kap. 3.2), erforderlich. Dazu werden die Bilder während des Bildfluges zusammen mit einem Zeitstempel gespeichert. Zusätzlich werden Datenströme einer GPS-Einheit (Global Positioning System) und einer INS-Einheit (engl.: inertial navigation system, dt.: Trägheitsnavigationssystem) aufgezeichnet. GPS liefert die absolute 3D-Position im WGS84-Koordinatensystem. INS liefert den Kurs, die Neigung und den Rollwinkel des Flugzeugs. Das Gesamtsystem ist auf einer Stabilisierungsplattform montiert. Nach dem Bildflug werden die GPS- und INS-Datenströme mit Hilfe bekannter Kalibrierungsdaten oder zusätzlicher flugspezifischer Referenzmessungen aufbereitet. Anschließend können die Schrägluftbilder auf Basis der relativen Lage und Ausrichtung der Kameras zu GPS- und INS-Einheit direkt, d. h. ohne Nutzung zusätzlicher fotogrammetrischer Verfahren, georeferenziert werden. Die Genauigkeit der direkten Georeferenzierung ist dabei etwas schlechter, aber vergleichbar mit der der klassischen Aerotriangulierung auf Basis bekannter bzw. markierter Passpunkte (Kraus, 1997; Cramer, 2003). Die durch die direkte Georeferenzierung ermittelten äußeren Orientierungsdaten haben den Vorteil, eine Messung der Realität zu sein. Durch Aerotriangulierung bestimmte äußere Orientierungsdaten sind dagegen Näherungslösungen, die aus einer Menge von Beobachtungen (Passpunkte) abgeleitet werden und nicht notwendigerweise mit der tatsächlichen Lage und Ausrichtung der Kamera übereinstimmen müssen.

Tab. 2.1 listet einige aktuelle Erfassungssysteme und deren Haupteigenschaften auf. Der Ablenkungswinkel der Schrägluftbilder variiert je nach System und Anforderung. Typischerweise liegt er zwischen 35° und 45°. Bei gegebener Brennweite hängt die am Boden erreichte metrische Auflösung (engl.: ground sample distance, GSD) von der Flughöhe ab, die sich wiederum

Name	*Kameras für Schrägansichten*	*Physische Bildauflösung*	*Besonderheiten*
Track'Air MIDAS (MultiVision)	4x Canon EOS-1Ds Mark III	5616×3744	Eine fünfte austauschbare Spezialkamera dient für Nadiraufnahmen.
Pictometry Imaging System (Wang u. a., 2008)	4x Canon EOS-1Ds	4008×2672	Eine fünfte Kamera gleichen Typs dient für Nadiraufnahmen.
Trimble AOS (Wiedemann, 2009)	2x Rollei Metric AIC 39	7228×5428	Der Kamerakopf wird nach jeder Aufnahme um 90° geschwenkt, um 4 Richtungen abzudecken. Eine dritte Kamera gleichen Typs dient für Nadiraufnahmen.
PFIFF (Grenzdörffer u. a., 2009)	1x Rollei AIC 45	5436×4080	Forschungsplattform der Uni Rostock. Für Abdeckung aller vier Richtungen muss die Befliegung entsprechend ausgelegt sein.
HRSC-AX (Deutsches Zentrum für Luft- und Raumfahrt)	8x Linienkamera	12000 Pixel Breite	Acht monochrome Zeilenscanner mit jeweils festem Spektrum und Neigung, die sich eine gemeinsame Optik teilen. Deckt nur Vor- und Rückansicht ab; für Abdeckung aller vier Richtungen muss die Befliegung entsprechend ausgelegt sein.

Tab. 2.1: Übersicht ausgewählter Schrägluftbilderfassungssysteme mit ihren Haupteigenschaften.

aus der geforderten Überlappung, der Fluggeschwindigkeit und der Aufnahmefrequenz ergibt. Typischerweise sind mit aktuellen Systemen zwischen 5cm und 15cm pro Pixel, gemessen am unteren Bildrand, möglich. Auf Grund der Neigung reduziert sich die metrische Auflösung in anderen Bildbereichen deutlich.

2.3 Visualisierung und Projektionen

Visualisierung ist in ihrer grundlegenden Definition die Erzeugung eines mentalen Modells bzw. Bildes zu einem in Daten enthaltenen, „verborgenen" Zusammenhang (Spence, 2001). Das Ziel einer Visualisierung ist die Transformation verfügbarer Daten in für den Menschen nützliche Informationen und die Ermöglichung der Ableitung von Wissen. Visualisierung ist dabei weder auf grafische Darstellungen beschränkt, noch setzt sie einen Computer voraus. Nichtsdestotrotz hat die computerbasierte grafische Visualisierung einen wesentlichen Anteil an der Visualisierung. Sie wird weiter in

- *wissenschaftlich-technische Visualisierung* für die Darstellung gemessener oder berechneter Werte physikalischer Größen im Raum (Schumann und Müller, 2000),
- *Informationsvisualisierung* für die Darstellung nichträumlicher Daten (Spence, 2001),
- *Geovisualisierung* für die Darstellung raumbezogener Daten (Dykes u. a., 2005) und
- *Visual Analytics* für die Zusammenführung und visuelle Auswertung großer Datenmengen (Keim u. a., 2009)

unterteilt. Grundsätzlich umfassen diese Visualisierungen nicht nur die Bilderzeugung sondern auch Interaktion als wichtigen Teil des Wissensgewinnungsprozesses.

Die in dieser Arbeit behandelte Visualisierung virtueller 3D-Stadtmodelle fällt in die Kategorie der Geovisualisierung. Geovisualisierung thematisiert die Visualisierung raumbezogener bzw. speziell erdbezogener Daten. Solche Geodaten sind entweder selbst geometrische Primitive, z. B. Polygone, haben eine explizite geografische Position, z. B. ein Pixel eines Luftbildes, oder sind einer Entität mit Raumbezug zugeordnet, z. B. die Höhe eines Baums. Ziel der Geovisualisierung ist die Vermittlung von Eigenschaften und Zusammenhängen räumlicher Objekte und Phänomene. Der Raum kann dabei sowohl in zwei als auch drei Dimensionen abgebildet werden. Die folgenden Erörterungen beziehen sich auf die für virtuelle 3D-Stadtmodelle wesentlichen 3D-Darstellungen.

Aus einer technischen Perspektive ergeben sich verschiedene Anforderungen an 3D-Geovisualisierungssysteme. Solche Systeme arbeiten vorrangig mit Bildschirmen oder Projektoren als Ausgabemedium, die eine große aber begrenzte Menge Pixel aufweisen, und müssen gleichzeitig Geodaten mit großer horizontaler Ausdehnung visualisieren. Typische Darstellungen zeigen deshalb nur kleine Ausschnitte mit hoher Genauigkeit oder größere Ausschnitte mit geringerer Genauigkeit. Unter solchen Bedingungen sind out-of-core Verfahren, die immer nur aktuell benötigte Daten von einem Massenspeicher laden, und Level-of-Detail (LoD) Verfahren, die Daten in unterschiedlichen Genauigkeitsstufen darstellen, essentiell. Für 3D-Geländedarstellungen gibt es verschiedene Verfahren, die im Allgemeinen beide Ansätze kombinieren (Baumann, 2000; Asirvatham und Hoppe, 2005; Hwa u. a., 2004; Duchaineau u. a., 1997; Livny u. a., 2009). Auf virtuelle 3D-Stadtmodelle spezialisierte Verfahren werden in Buchholz und Döllner (2005) und Cignoni u. a. (2007) vorgestellt. Im Bereich der reinen LoD-Verfahren gibt es zwei Herangehensweisen: computergrafisch motivierte Verfahren (Hoppe, 1996; Erikson u. a., 2001; DeCoro und Tatarchuk, 2007), die jedoch für virtuelle 3D-Stadtmodelle auf Grund deren Struktur nur begrenzt anwendbar sind, und kartografisch motivierte Verfahren (Stüber, 2005; Glander und Döllner, 2008; Thiemann, 2002; Chang u. a., 2008), die Generalisierungsstrategien aus der Kartografie auf virtuelle 3D-Stadtmodelle übertragen. Daneben ist die Bestimmung sichtbarer Objekte eine zur Optimierung der Bilderzeugung wesentliche Anforderung an 3D-Geovisualisierungssysteme. Insbesondere in virtuellen 3D-Stadtmodellen erlaubt der u. U. extrem hohe Verdeckungsgrad deutliche Geschwindigkeitsgewinne (Wonka u. a., 2001; Mattausch u. a., 2008; Pantazopoulos und Tzafestas, 2002; Cohen-Or u. a., 2003). In interaktiven 3D-Geovisualisierungssystemen ist zusätzlich die Navigation, d. h. die Steuerung der virtuellen Kamera, eine zentrale Funktion. Das umfasst zum einen die verfügbaren Navigationsmetaphern (Hand, 1997; Darken und Sibert, 1993), und zum anderen die gezielte Lenkung und Unterstützung des Benutzers bei der Navigation (Galyean, 1995; Buchholz u. a., 2005; Ropinski u. a., 2005).

Aus einer inhaltlichen Perspektive gibt es zwei Anforderungen an Visualisierungen und damit auch an 3D-Geovisualisierungen: Expressivität und Effektivität (Schumann und Müller, 2000). *Expressivität* bezeichnet die unverfälschte Wiedergabe der Daten. Jede Visualisierungstechnik

eignet sich nur für bestimmte Arten von Daten; wird sie auf ungeeignete Daten angewendet, wird der Betrachter am Ziehen der richtigen Schlüsse gehindert. Die *Effektivität* einer Visualisierungtechnik drückt die Verständlichkeit bzw. Klarheit der resultierenden Visualisierung für den Betrachter aus. Eine effektive Visualisierungstechnik ermöglicht dem Betrachter eine intuitive und direkte Informationsgewinnung. Dabei ist neben den Gewohnheiten des Betrachters vor allem die Zielstellung der Visualisierung für die Effektivität einer bestimmten Visualisierungstechnik entscheidend. Schumann und Müller (2000) definieren zusätzlich *Angemessenheit* als eine allgemeine Anforderung. Damit wird die Relation zwischen Aufwand zur Erzeugung und dem Nutzen einer Visualisierung bezeichnet. Die Angemessenheit verbindet demnach die technische und inhaltliche Seite der Visualisierung.

Die Arbeit konzentriert sich auf ein grundlegendes Element der 3D-Geovisualisierung: die Projektion der 3D-Welt auf ein 2D-Bild. In der interaktiven 3D-Geovisualisierung wird dazu standardmäßig die perspektivische Projektion verwendet. Auf der technischen Seite ist sie in den meisten Grafikbibliotheken direkt verfügbar und erfordert keinerlei zusätzlichen Programmieraufwand. Auf der inhaltlichen Seite ist die Expressivität gegeben, da die perspektivische Projektion dem gewohnten Seheindruck entspricht und damit eine intuitive Abbildung der 3D-Welt liefert. Die Effektivität für die jeweilige Aufgabenstellung wird jedoch nur selten hinterfragt.

Eine Forschungsrichtung, in der die Veränderung der Projektion zur Effektivitätssteigerung ein Kernthema ist, ist die Fokus&Kontext-Visualisierung aus dem Bereich der Informationsvisualisierung. Der Grundansatz solcher Visualisierungen ist die gleichzeitige Darstellung der gerade relevanten Daten (*Fokus*) und ihrer Umgebung (*Kontext*), um dem Betrachter die Einordnung des Gesehenen insbesondere bei Verschiebungen des Fokus zu erleichtern. Die Unterscheidung zwischen Fokus und Kontext kann durch stilistische Mittel erfolgen, durch zeitlich oder räumlich getrennte Darstellung oder durch Manipulation der Projektion (Cockburn u. a., 2008). Letztere hat das Ziel, einen kontinuierlichen Übergang bei gleichzeitig variabler Aufteilung des Bildraums zu erreichen. Die Manipulation der Projektion bietet sich somit als ein Mittel zur Effektivitätssteigerung einer Darstellung an.

Mit diesem Hintergrund wird in dieser Arbeit die Effektivität und damit die inhaltliche Seite der perspektivischen Projektion für die Geovisualisierung virtueller 3D-Stadtmodelle untersucht (Kap. 5). Die dazu angenommene Aufgabe ist die Wissensgewinnung über die räumliche Struktur einer städtischen Umgebung wie sie auch mit traditionellen 2D-Stadtplänen erfolgt. Sich daraus ergebende mögliche Anwendungen sind die Verortung eines Punktes in der Stadt (z. B. Betrachterstandort oder Kamerastandort) oder die Wegfindung in einer lokalen Umgebung. Als Ergebnis der Untersuchung werden zwei Projektionen vorgestellt, die jeweils zwei Perspektiven kontinuierlich verbinden, wobei eine Perspektive einen kartenähnlichen Charakter hat, während die andere 3D-Elemente betont.

Weiterhin werden Projektionen von der technischen Seite her untersucht (Kap. 6 und 7). Es wird mit stückweise perspektivischen Projektionen ein Konzept entwickelt, das die Nutzung aller 3D-Grafikhardwareeigenschaften und damit optimale Bildqualität für nichtlineare Projektionen erlaubt, obwohl die 3D-Grafikhardware nur für lineare Projektionen entworfen und optimiert ist. Für dieses Konzept werden Verfahren beschrieben, die unter Ausnutzung neuer Möglichkeiten moderner 3D-Grafikhardware (Kap. 2.3.2) dieses Konzept effizient umsetzen.

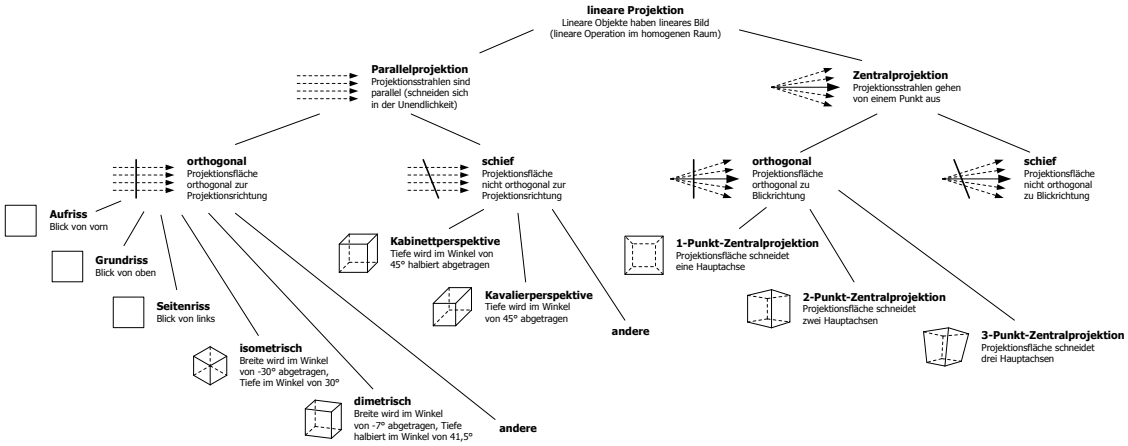


Abb. 2.4: Klassifikation linearer Projektionen basierend auf Foley u. a. (1995).

2.3.1 Klassifikation von Projektionen

Der im Kontext der Computergrafik gebrauchte Begriff der Projektion bezeichnet die Abbildung eines n-dimensionalen Vektorraums $\mathbb{O}$ (*Objektraum*), auf einen Vektorraum $\mathbb{B}$ (*Bildraum*) mit $m < n$ Dimensionen (Foley u. a., 1995). Die Computergrafik betrachtet dabei insbesondere den Fall $\mathbb{O} = \mathbb{R}^3$ und $\mathbb{B} = \mathbb{R}^2$, auf den sich im Folgenden beschränkt werden soll.

Eine allgemeine Projektion bildet alle Punkte o eines in $\mathbb{O}$ eingebetteten eindimensionalen Objekts p, dem Projektionsstrahl, auf einen dazugehörigen Punkt $b \in \mathbb{B}$ ab:

$$o \to b : o \in p(b, r) \text{ mit } b \in \mathbb{B} \text{ und } r \in \mathbb{R}_+^* \tag{2.1}$$

Die Projektion ist eine allgemeine mathematische Relation von $\mathbb{O}$ auf $\mathbb{B}$, die weder linkstotal (allen $o \in \mathbb{O}$ wird ein Bild $b \in \mathbb{B}$ zugeordnet), noch funktional (alle $o \in \mathbb{O}$ haben höchstens ein Bild $b \in \mathbb{B}$), surjektiv (alle $b \in \mathbb{B}$ sind Bild eines $o \in \mathbb{O}$) oder injektiv (alle $b \in \mathbb{B}$ sind Bild höchstens eines $o \in \mathbb{O}$) sein muss. Die Klassifikation der Projektionen erfolgt über die Eigenschaften der Projektionsstrahlen p, sowie die Eigenschaften des Bildraums $\mathbb{B}$, das im Allgemeinen eine in $\mathbb{O}$ eingebettete parametrisierte Oberfläche Q darstellt. Anschaulich unterscheiden sich Projektionen in vier Aspekten, die im Folgenden zur Klassifikation herangezogen werden sollen:

- der Form der Projektionsstrahlen p,
- der Verteilung bzw. Lage der Projektionsstrahlen in $\mathbb{O}$,
- der Form der Projektionsfläche Q und
- der Parametrisierung der Projektionsfläche, die Elemente von $\mathbb{B}$ auf Punkte der Projektionsfläche Q abbildet.

Eine erste, grobe Unterscheidung basiert auf der Linearität von p und Q. Sind die Projektionsstrahlen gerade, spricht man von geometrischen Projektionen; ist Q eine Ebene, von planaren Projektionen. Eine Teilmenge der planaren geometrischen Projektionen sind die für die Praxis besonders relevanten linearen bzw. Lochkamera-Projektionen (Foley u. a., 1995), die lineare Objekte wie Linien oder Polygone in $\mathbb{O}$ auf lineare Objekte in $\mathbb{B}$ abbilden. Diese Klasse ist von Interesse, da nur die End- bzw. Eckpunkte eines linearen Objektes projiziert werden müssen. Punkte dazwischen können durch lineare Interpolation im Bildraum bestimmt werden, was die Berechnungskomplexität deutlich senkt. Die linearen Projektionen sind auf Grund ihres

Stellenwertes gut untersucht und ausführlich in der Literatur beschrieben (Foley u. a., 1995; Salomon, 2006). Ihre Unterteilung lässt sich hierarchisch darstellen (Abb. 2.4). Grundsätzlich werden sie in Parallelprojektionen, bei denen die Projektionsstrahlen parallel verlaufen, und Zentralprojektionen, bei denen alle Projektionsstrahlen von einem Punkt (Projektionszentrum) ausgehen, unterschieden. Die verschiedenen Parallelprojektionen sind einfach zu konstruieren und ermöglichen das direkte Messen von Abständen und Winkeln, weshalb sie hauptsächlich bei technischen Zeichnungen oder Skizzen eingesetzt werden. Zentralprojektionen sind eine gute Approximation der Geometrie des menschlichen Sehens bzw. der technischen Bildaufzeichnung nach dem Lochkameraprinzip; zentralprojizierte Bilder sind daher gut verständlich.

Für nichtlineare Projektionen gibt es keine allgemein anerkannte Klassifikation analog jener für lineare Projektionen (Salomon, 2006). Vielmehr werden solche Projektionen für konkrete Anwendungsfälle entworfen, z. B.:

- für die Kompensation von Verzerrungen in nichtplanaren Ausgabegeräten wie z. B. Zylinder- oder (Halb-)Kugelprojektoren (Max, 1983; Jo u. a., 2008),
- für die Modellierung natürlicher Phänomene wie z. B. von Kaustiken, Reflexionen und Brechungen an nichtplanaren Oberflächen oder von relativistischer Lichtausbreitung in starken Gravitationsfeldern (Wei u. a., 2007; Gröller, 1995),
- in der Visualisierung zur Verbesserung bzw. Lenkung der Wahrnehmung der virtuellen Welt, bspw. durch Erweiterung des Sichtfeldes, Implementierung von Linseneffekten oder Minderung von Verzerrungen (Brosz u. a., 2007; Kopf u. a., 2009),
- in Kunst und Nichtfotorealismus für gestalterische oder erzählerische Effekte (Wood u. a., 1997; Agrawala u. a., 2000; Glassner, 2004a,b) oder
- als Speicher für Teile der plenoptischen Funktion (Rademacher und Bishop, 1998) zur Vereinfachung der Bildsynthese (z. B. Reflexionen, Brechungen, bildbasiertes Rendering Heidrich und Seidel, 1998; Wan u. a., 2007; Popescu und Aliaga, 2006).

Tab. 2.2 führt verschiedene lineare und nichtlineare Projektionen mit ihren Eigenschaften bezüglich der oben genannten vier Aspekte auf.

2.3.2 Die GPU-Renderingpipeline

Die Rasterisierung ist ein Hauptansatz zur Bilderzeugung in der 3D-Computergrafik. Dabei werden alle Szenenobjekte in Grundelemente (Primitive) zerlegt, für jedes Primitiv durch die Scankonvertierung überdeckte Pixel ermittelt (Fragmente), die Fragmente eingefärbt und nach erfolgreicher Sichtbarkeitsbestimmung in den Ausgabepuffer geschrieben. Das Bild ergibt sich nach Verarbeitung aller Objekte. Der Gesamtprozess lässt sich durch eine Sequenz von Schritten, die Renderingpipeline (Foley u. a., 1995), beschreiben. Die Renderingpipeline besteht aus:

Anwendungsstufe: Ist für die Nutzereingaben, Organisation der Szene, Steuerung von Szenenobjekten und Sichtbarkeitsbestimmung von Szenenobjekten verantwortlich.

Geometriestufe: Ist für die Verarbeitung ganzer Primitive und ihrer Eckpunkte verantwortlich und umfasst Transformation, Projektion, Beleuchtung, Animation und Sichtbarkeit.

Rasterisierungsstufe: Konvertiert die Primitive in Fragmente, d. h. ein Pixel große Primitivteile, bestimmt deren Farbe über Texturierung oder Beleuchtungsberechnungen, bestimmt deren Sichtbarkeit und schreibt sie gegebenenfalls in den Ausgabepuffer.

Projektionsname	*Form der Projektionsstrahlen*	*Verteilung der Projektionsstrahlen*	*Form der Projektionsfläche*	*Parametrisierung der Projektionsfläche*	*Verwendungsbeispiel*
Parallelprojektion	linear	parallel (kein PZ)	planar	uniform	technische Zeichnungen
Dreitafelprojektion	linear	drei separate Strahlenrichtungen	stückweise planar	uniform	Konstruktionszeichnungen
Perspektivprojektion	linear	ein PZ	planar	uniform	Standardprojektion der 3D-Echtzeitcomputergrafik
Kubische Projektion	linear	ein PZ	kubisch	uniform	Cubemaptexturen
Sphärische Projektion	linear	ein PZ	Halbkugeloberfläche	uniform in xy-Ebene	Environmentmapping
Paraboloide Projektion	linear	ein PZ	Paraboloidoberfläche	uniform in xy-Ebene	Environmentmapping
Zylinderprojektion	linear	ein PZ	Zylinderoberfläche	uniform	Darstellung auf zylinderförmigen Bildschirmen
Projektion für omnidirektionales Stereo (Peleg u. a., 2001)	linear	ringförmiges PZ	Zylinder-/Kugeloberfläche	uniform	Stereodarstellung auf zylinder-/kugelförmigen Bildschirmen
Zeilenprojektion	linear	ein lineares PZ	planar	uniform	Zeilenkamera (z. B. HRSC-AX)
Lochkamera mit imperfekter Linse	linear	ein PZ	planar	radial verzerrend	Simulation realer Kameralinsen
Gravitationslinsen (Gröller, 1995)	gekrümmt	ein PZ	planar	uniform	Astronomie
Magic lenses (Yang u. a., 2005) / Kameratexturen (Spindler u. a., 2006)	linear	ein PZ	planar	nichtuniform	Fokus&Kontext-Darstellungen
Projection Tile Screens (Trapp und Döllner, 2008)	linear	ein PZ	planar/kubisch	nichtuniform	Fokus&Kontext-Darstellungen
Multiperspektivische Projektionen (Glassner, 2004a,b)	linear	mehrere PZ	nichtplanar	nichtuniform	Künstlerische Effekte (z. B. Storyboarding)
SCFP (Brosz u. a., 2007)	Bezierkurve	unbestimmt	Bezierfläche	beliebig	Künstlerische Effekte
Verdeckungskamera (Popescu und Aliaga, 2006)	stückweise linear	ein PZ	planar	uniform	Bildbasiertes Rendering
General Linear Cameras (Yu und McMillan, 2004b)	linear	Linearkomb. dreier Referenzstrahlen	planar	uniform	Verallgemeinertes Kameramodell

Tab. 2.2: Eigenschaften verschiedener Projektionen (PZ = Projektionszentrum).

Da sich insbesondere Geometrie- und Rasterisierungsstufe auf wenige generische Entitäten (i. A. Dreiecke, Linien, Punkte und Fragmente) und dafür spezialisierte Operationen konzentrieren, es einen gerichteten Datenfluss gibt und die einzelnen Entitäten in den beiden Stufen weitgehend unabhängig voneinander sind, bieten sie sich für eine Implementierung in datenparalleler Hardware an. Die Steuerung dieser 3D-Grafikhardware (Graphics Processing Unit, GPU) erfolgt über systemnahe und (hardware-)herstellerunabhängige 3D-Grafikbibliotheken, bei denen sich OpenGL und Direct3D durchgesetzt haben. Beide Bibliotheken definieren eine im Wesentlichen identische GPU-Renderingpipeline als abstraktes Modell der Datenverarbeitung auf der GPU und umfassen die vollständige Geometrie- und Rasterisierungsstufe. Die Anwendungsstufe obliegt weiterhin dem Anwendungsprogrammierer bzw. höheren 3D-Grafikbibliotheken wie z. B. VRS (Döllner und Hinrichs, 2002) und wird nicht von der GPU-Renderingpipeline abgebildet.

Die Entwicklung der systemnahen 3D-Grafikbibliotheken und damit der GPU-Renderingpipeline spiegelt direkt die Entwicklung der GPUs von einer hochspezialisierten, konfigurierbaren Funktionseinheit zu einem hochparallelisierten, programmierbaren Datenprozessor wieder. Die aktuelle GPU-Renderingpipeline hat nur noch wenig mit der ursprünglich von OpenGL 1.5 (Segal und Akeley, 2003) oder Direct3D 7 als Zustandsmaschine definierten gemein. Heute sind große Teile der fest verdrahteten Funktionalität durch programmierbare Einheiten ersetzt und um neue Verarbeitungsstufen ergänzt worden (Luebke und Humphreys, 2007). Diese Entwicklung der von aktuellen GPUs implementierten Renderingpipeline wird am Beispiel von Direct3D 7 bis 11 beschrieben. Die in dieser Arbeit vorgestellten Verfahren sind hauptsächlich mit OpenGL umgesetzt. An dieser Stelle wird jedoch Direct3D verwendet, da anders als bei OpenGL Änderungen in der Renderingpipeline mit Versionsänderungen einhergingen.

Direct3D 7 markierte die letzte Version einer ausschließlich fest verdrahteten GPU-Renderingpipeline. Bis zur Version 9c wurde diese konfigurierbare Form der Pipeline als Programmiermodell unterstützt und um Spezialfunktionen wie z. B. Geometrieverblendung und zusätzliche Textureinheiten erweitert. Eine echte Programmierbarkeit wurde erstmals in Direct3D 8 Ende des Jahres 2000 mit Vertex- und Pixelshadern eingeführt. Diese Shader wurden in Assembler geschrieben, hatten getrennten Funktionsumfang und ersetzten bei Verwendung ausgewählte Abschnitte der fest verdrahteten, lediglich konfigurierbaren Pipeline. Im Jahr 2002 wurde mit Direct3D 9 die C-ähnliche Sprache HLSL (High-Level Shader Language) zur Shaderprogrammierung eingeführt. Bis zur Version 9c aus dem Jahr 2004 wurden in mehreren Schritten Programmbeschränkungen (z. B. Programmlänge, Registeranzahl) gelockert, neue Programmkonstrukte ergänzt (z. B. dynamische Flusskontrolle) und der Funktionsumfang von Vertex- und Pixelshadern angeglichen (z. B. Rechengenauigkeit, Texturzugriffe). In der Version 10 (2006) wurde die GPU-Renderingpipeline um den Geometryshader erweitert und gleichzeitig eine gemeinsame Ausführungsumgebung für alle Shaderstufen definiert. Zusätzlich wurden alle Funktionen der fest verdrahteten Pipeline, die durch Shader implementierbar sind, ersatzlos gestrichen. In der aktuellen Version 11 (2009) wurde die Pipeline um eine Verfeinerungseinheit erweitert, die aus zwei neuen Shaderstufen und einer komplexen konfigurierbaren, aber nicht programmierbaren, Einheit besteht.

Die aktuelle GPU-Renderingpipeline ist eine Sequenz fest verdrahteter Einheiten und programmierbarer Einheiten, die Shader ausführen (Abb. 2.5). Aus Entwicklersicht sind Shader einzelne Funktionen, die automatisch für jeweils eine Entität (Eckpunkt, Primitiv oder Fragment) aufgerufen werden und diese exklusiv prozessieren. Tatsächlich führt die GPU Shader hochparallelisiert und ineinander verschachtelt auf einem SIMD (Single Instruction Multiple

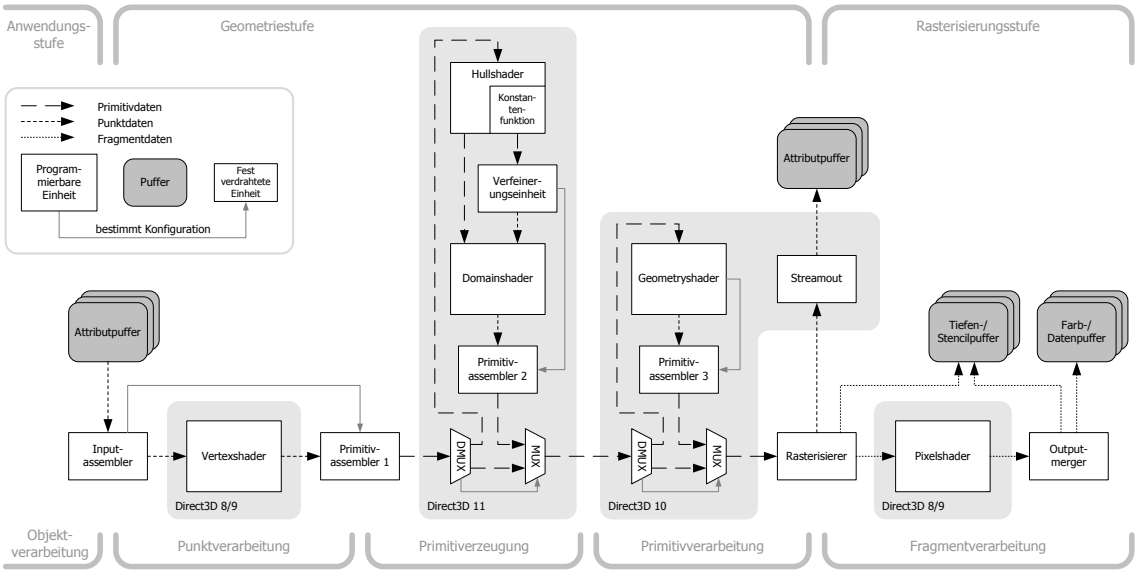


Abb. 2.5: GPU-Renderingpipeline von Direct3D 11. Die Anwendungsstufe wird nicht von Direct3D abgedeckt. Die Geometriestufe lässt sich in Punktverarbeitung, Primitiverzeugung und Primitivverarbeitung gliedern. Die Rasterisierungsstufe verarbeitet die vom Rasterisierer erzeugten Fragmente.

Data) Prozessor aus, um Wartezeiten für Speicherzugriffe oder Synchronisation zu überbrücken (Crassin, 2008). Die Aufgaben der einzelnen Einheiten sind:

Inputassembler: Erzeugung von Punkten aus den von der Anwendung spezifizierten Eingabedaten.

Vertexshader: Operationen auf einzelnen Punkten, z. B. Transformation, Normalenberechnung, Texturkoordinatenberechnung oder Beleuchtung.

Primitivassembler 1: Zusammenführung transformierter Punkte zu Primitiven; beinhaltet die Auflösung von Indizes und effizientes Caching.

Hullshader: Hat Zugriff auf alle transformierten Punkte eines Primitivs und besteht aus zwei Teilen: der Kontrollpunktfunktion, die für jeden Kontrollpunkt des Primitivs ausgeführt wird und diesen nochmals transformieren kann, und der Konstantenfunktion, die einmal für den gesamten Primitiv ausgeführt wird um die Verfeinerungsfaktoren zu bestimmen.

Verfeinerungseinheit: Erzeugt anhand der Verfeinerungsfaktoren baryzentrische Eckpunkte.

Domainshader: Hat Zugriff auf alle Ausgaben des Hullshaders und wird (konzeptionell) für jeden von der Verfeinerungseinheit ausgegebenen Eckpunkt einmal aufgerufen. Transformiert den baryzentrischen Eckpunkt in einen dem Primitiv entsprechenden Punkt.

Primitivassembler 2: Zusammenführung der vom Domainshader transformierten Punkte zu verfeinerten Primitiven (anderer Primitivtyp als bei Primitivassembler 1).

Geometryshader: Hat Zugriff auf alle Ausgaben des Vertex- bzw. Domainshaders für alle Eckpunkte eines Primitivs. Kann Operationen auf diesem Primitiv ausführen, z. B.

Culling, Transformation, Erzeugung neuer Primitiven oder Änderung des Primitivtyps.

Primitivassembler 3: Zerlegung der vom Geometryshader generierten Strips in Einzelprimitive.

Rasterisierer: Führt perspektivische Division sowie Clipping und Culling im Bildraum durch, konvertiert alle Primitive in Fragmente (Scanlinekonvertierung) und prüft Weiterverarbeitung der einzelnen Fragmente mittels eines Tiefen- und Stencilpuffers.

Streamout: Schreibt Eingaben (Punktdaten) des Rasterisieres als Datenstrom in einen oder mehrere Puffer.

Pixelshader: Bestimmt Farbe und/oder andere Fragmenteigenschaften, z. B. Tiefe oder Normale. Die Position im Bildpuffer kann nicht verändert werden. Kann ein Fragment auch verwerfen.

Outputmerger: Schreibt das Ergebnis des Pixelshaders in den Bildpuffer. Beinhaltet Blending, einen nochmaligen Tiefen- und Stenciltest.

Für die in dieser Arbeit vorgestellten Verfahren sind insbesondere die Erweiterungen der GPU-Renderingpipeline seit Direct3D 10 interessant, da sie erstmals die einfache und flexible Generierung neuer Primitive direkt in der GPU-Renderingpipeline erlauben. Der Geometryshader ermöglicht die Manipulation ganzer Primitive, indem er für jedes Eingabeprimitiv ausgeführt wird und diese durch eine (möglicherweise leere) Menge von Ausgabeprimitiven ersetzt. Dabei darf sich zum einen der Typ der Ausgabeprimitive von dem der Eingabeprimitive unterscheiden und zum anderen die Anzahl der Ausgabeprimitive von Aufruf zu Aufruf ändern. Wesentliches Merkmal des Geometryshaders ist die Erhaltung der Ordnung, d. h. die vom Geometryshader generierten Primitive müssen entsprechend der Reihenfolge der Eingabeprimitive angeordnet werden, bevor sie vom Rasterisierer verarbeitet werden können. Die notwendige Zwischenspeicherung macht eine hardwareseitige Limitierung der Ausgabekapazität des Geometryshaders pro Aufruf notwendig, die derzeit lediglich 4096 Byte beträgt (Blythe, 2006). Zudem nimmt die Effizienz potentiell ausgabeintensiver Shader dramatisch ab (Kap. 6.5), was den Geometryshader auf den ersten Blick ungeeignet für Geometrieverfeinerung macht. Seine Hauptzwecke sind stattdessen bspw. die Transformation von Punktprimitiven in Dreiecke (Splatting), die effiziente Erzeugung von Cubemaps (z. B. für Shadowmapping) oder die Silhouettenbestimmung (z. B. für nichtfotorealistische Effekte; Tariq, 2006).

Die stark begrenzte Ausgabekapazität des Geometryshaders wird durch die Verfeinerungseinheit kompensiert. Diese Einheit ist speziell auf die Generierung großer Mengen von Primitiven (bis zu 8192 Dreiecke) ausgelegt. Die eigentliche Erzeugung der Primitive erfolgt in der fest verdrahteten Verfeinerungseinheit, die jedoch konfigurierbar ist. Ihre Konfiguration bezieht sie vom vorgeschalteten Hullshader, der pro Eingabeprimitiv eigene Verfeinerungsfaktoren bestimmen kann. Passend zu den jeweiligen Faktoren generiert die Verfeinerungseinheit ein generisches Geometrienetz, dessen Punkte baryzentrische Koordinaten $(u, v) \in [0;1] \times [0;1]$ verwenden. Erst der nachgeschaltete Domainshader konvertiert diese generischen Punkte in an das ursprüngliche Eingabeprimitiv angepasste Punkte. Das Eingabeprimitiv ist nicht beschränkt auf Punkte, Linien und Dreiecke, sondern kann aus bis zu 32 Kontrollpunkten bestehen. Die Verfeinerungseinheit unterliegt nicht der Ordnungserhaltung, weshalb hier keine Effizienzeinbußen beobachtbar sind. Ein Beispiel für den Einsatz der Verfeinerungseinheit ist die vollständig auf der GPU ausgeführte adaptive Verfeinerung animierter Bezierflächen aus 16 Kontrollpunkten (Tatarchuk, 2007).

2.3.3 Nichtlineare Projektionen auf der GPU

In interaktiven Systemen erfolgen die Bilderzeugung und damit die Projektion auf der GPU. Die in diesem Zusammenhang wesentliche Komponente der GPU ist der Rasterisierer (Kap. 2.3.2). Er ist Kernkomponente der Rasterisierungsstufe, die als eines der ersten Elemente der Renderingpipeline hardwarebeschleunigt wurde (Seitz, 2004). Der Rasterisierer führt die Scankonvertierung von 2D-Primitiven durch und umfasst das Clipping, die Attributinterpolation und die Fragmenterzeugung. Er ist nicht programmierbar und nur minimal über nutzerdefinierte Clippingebenen und wählbare Attributinterpolationsarten konfigurierbar. Rasterisierbare 2D-Primitive sind Punkte, gerade Strecken und geradlinig begrenzte Dreiecke (Akenine-Möller u. a., 2008). Der Rasterisierer funktioniert damit lediglich für lineare Projektionen korrekt. Andere Projektionen müssen approximiert oder ohne Hilfe des Rasterisierers implementiert werden.

Für die Umsetzung nichtlinearer Projektionen auf der GPU haben sich verschiedene technische Ansätze etabliert:

Bildbasierte Umsetzung: Dabei werden zuerst ein oder mehrere perspektivische Zwischenbilder gerendert, so dass mindestens der gesamte sichtbare Bereich der beabsichtigten Projektion abgedeckt ist. Diese Zwischenbilder werden dann mittels Texture Mapping verzerrt auf dem Bildschirm dargestellt, um den gewünschten nichtlinearen Projektionseffekt zu erreichen. Dieser Ansatz ist einfach umzusetzen und wird sehr gut von aktueller 3D-Grafikhardware unterstützt (Yang u. a., 2005; Trapp und Döllner, 2008). Hauptnachteil ist die Bildqualität. Der Verzerrungsschritt verursacht unweigerlich Schärfeverluste, die sich an Kanten oder Bilddetails bemerkbar machen.

Punktbasierte Umsetzung: Hier wird die nichtlineare Projektion für jeden Eckpunkt im Vertexshader exakt berechnet. Die Rasterisierung erfolgt aber weiterhin durch die GPU auf Basis linear begrenzter 2D-Primitive. Die dabei verwendete lineare Interpolation korrespondiert nicht mit der Projektion und führt zu Ungenauigkeiten. Die Größe der Ungenauigkeiten hängt von der Größe der 2D-Primitive ab. Für ein akzeptables Ergebnisbild muss das Ausmaß der Ungenauigkeiten und damit die Größe der 2D-Primitive begrenzt werden. In interaktiven Anwendungen müssen 3D-Primitive deshalb dynamisch zerlegt werden können. Dazu gibt es eine große Palette an Verfahren, wie zum Beispiel vorberechnete statische Multiresolutionsmodelle (Sander und Mitchell, 2005), Progressive Meshes (Hoppe, 1996), adaptive Geometrieverfeinerung (Boubekeur und Schlick, 2008; Tatarinov, 2008), Rendern in Punktpuffer (Yu u. a., 2009) oder Hardwareverfeinerungseinheiten (Tatarchuk, 2007; Castaño, 2008). Diese Verfahren unterscheiden sich in der Lastverteilung zwischen CPU und GPU. Für den Einsatz bei nichtlinearen Projektionen darf das verfeinerte Geometriegitter keine T-Kreuzungen enthalten, da an solchen auf Grund der inkorrekten Rasterisierung sichtbare Löcher entstehen können. Vorteil des punktbasierten Ansatzes ist die vergleichsweise einfache Umsetzung. Je nach Wahl des Verfeinerungsverfahrens kann die Implementierungskomplexität jedoch steigen. Nachteil dieses Ansatzes ist die Anfälligkeit für Z-Pufferartefakte wie ungenaues Clipping an der vorderen Clippingebene oder Durchdringungen paralleler Flächen. Abhilfe schafft die Berechnung korrekter Tiefenwerte im Fragmentshader, was aber wiederum Leistungseinbußen nach sich zieht (Persson, 2007).

Nichtlineare Rasterisierung bzw. Raycasting: Hierbei wird der GPU-Rasterisierer lediglich genutzt, um für eine konservative Schätzung der projizierten Fläche eines 3D-

Primitivs Fragmente zu erzeugen. Im Fragmentshader wird dann ein Strahlentest durchgeführt, ob das jeweilige Fragment tatsächlich im Bild des 3D-Primitivs liegt, und falls nicht verworfen. Ist das Fragment gültig, werden alle Attribute anhand des Strahlentests berechnet. Hou u. a. (2006) implementieren dieses Verfahren für zusammengesetzte Projektionen. Gascuel u. a. (2008) spezialisieren es für verschiedene analytische nichtplanare Projektionen. Wei u. a. (2007) verbessern das Verfahren von Hou u. a. (2006) durch den Einsatz von Beamtracing. Vorteil des Raycastingansatzes ist die Exaktheit. Nachteil ist der erhebliche Mehraufwand während des Renderns, da vorhandene fest verdrahtete Funktionseinheiten wie die Attributinterpolation nicht genutzt werden können, sondern explizit nachprogrammiert werden müssen. Zusätzlich müssen durch die Flächenschätzung deutlich mehr Fragmente als bei anderen Ansätzen verarbeitet werden.

Raytracing: Hierbei wird die GPU nicht als Implementierung der Renderingpipeline verwendet, sondern als massiv paralleler Prozessor. Anfangs wurden Raytracingalgorithmen manuell auf die programmierbaren Stufen der Renderingpipeline abgebildet (Purcell u. a., 2002; Carr u. a., 2002). Später entwickelte sich mit General Purpose Computation on Graphics Processing Units (GPGPU Harris, 2010) eine von der Renderingpipeline losgelöste Sicht auf die GPU. Die Programmierung erfolgt mit eigens dafür konzipierten Sprachen und APIs wie Brook (Buck u. a., 2004), AMDs CAL (ehemals CTM, AMD, 2009; Houston, 2008), NVidias CUDA (NVidia, 2009; Luebke, 2008), OpenCL (Khronos OpenCL Working Group, 2009; Munshi, 2008) oder DirectCompute (Corp., 2008; Boyd, 2008). Raytracing ist dabei nur eine mögliche Anwendung unter vielen (Owens u. a., 2007, 2008). Einen spezialisierten Ansatz für neuere GPUs stellt NVidia OptiX (Parker u. a., 2010) dar. OptiX ist eine generische GPU-basierte Raytracingbibliothek, bei der ausgewählte Stufen shaderähnlich programmiert werden können. Mit jeder dieser Techniken lassen sich nichtlineare Projektionen wie in Löffelmann und Gröller (1996) oder Yu und McMillan (2004b) beschrieben für interaktive Anwendungen umsetzen.

Eine völlig andere Herangehensweise an die Bildsynthese wird in Whitted und Kajiya (2005) entwickelt: Statt der bisherigen Beschränkung auf Dreiecke schlagen die Autoren eine prozedurale Geometrierepräsentation vor. Der Rasterisierer wird dafür durch eine Hardwareeinheit ersetzt, die die prozedurale Geometrie punktbasiert auswertet, bis keine Löcher mehr im Bild existieren. Dieser theoretische Ansatz würde spezielle Algorithmen für nichtlineare Projektionen überflüssig machen.

Die in dieser Arbeit entwickelten Visualisierungstechniken sollen nichtlineare Projektionen interaktiv mit bestmöglicher Bildqualität darstellen. Raytracing oder Raycasting erfüllen diese Anforderungen nicht, da dadurch wesentliche, in Hardware verfügbare Funktionen wie Antialiasing, qualitativ hochwertige anisotrope Texturfilterung oder Gradienten nicht genutzt werden können. Ebenso können bildbasierte Ansätze die qualitativen Vorteile dieser Funktionen nicht vollständig zur Geltung bringen, da die nachfolgende Verzerrung des Zwischenbildes während des Renderings nicht berücksichtigt wird. Insbesondere nichtfotorealistische Effekte oder prozedurale Texturen müssen für gute Bildqualität (z. B. gleichmäßige Strichstärken oder homogene Maserung) an die jeweilige nichtlineare Projektion angepasst werden. Das ideale Verfahren kann die für perspektivische Projektionen verwendeten Renderingeffekte direkt und ohne Qualitätseinbußen für nichtlineare Projektionen übernehmen. Dazu sind punktbasierte Umsetzungen (Kap. 5 und 6) und nichtlineare Rasterisierungen (Kap. 7) geeignet.

3 Automatische Texturierung mit Schrägluftbildaufnahmen

Viele Anwendungen erfordern virtuelle 3D-Stadtmodelle mit spezifischen Fototexturen, um die Erscheinung realistisch wiederzugeben. Um ein vollständig fototexturiertes virtuelles 3D-Stadtmodell zu erzeugen, werden als Ausgangsbasis ein geometrisches virtuelles 3D-Stadtmodell und flächendeckende, georeferenzierte Bildinformationen benötigt. Im hier vorgestellten Verfahren werden die Bildinformationen, die aus theoretischer Sicht einen Ausschnitt der plenoptischen Funktion (Adelson und Bergen, 1991) speichern, geeignet auf das virtuelle 3D-Stadtmodell projiziert. In einem einmalig ausgeführten Verarbeitungsschritt wird pro Fläche des virtuellen 3D-Stadtmodells eine Textur auf Basis dieser Projektion erzeugt. Anschließend ist das vollständig fototexturierte virtuelle 3D-Stadtmodell wie ein anderweitig texturiertes virtuelles 3D-Stadtmodell in interaktiven Anwendungen einsetzbar.

Im Hinblick auf die verschiedenen Anwendungsmöglichkeiten und unterschiedlichen Erfassungsmethoden ist es bei der Fototexturierung entscheidend, dass zu einem gegebenen geografischen Gebiet verfügbare virtuelle 3D-Stadtmodelle und Bildinformationen technisch und konzeptionell unabhängig sind. So lassen sich aus verschiedenen Bildinformationen (z. B. verschiedene Erfassungszeitpunkte oder Spektralbereiche) verschiedene Texturen für das gleiche virtuelle 3D-Stadtmodell und umgekehrt aus den gleichen Bildinformationen Texturen für verschiedene virtuelle 3D-Stadtmodelle (z. B. verschiedene Generalisierungsstufen) gewinnen. Bedingungen sind lediglich

1. die *räumliche Überschneidung* von virtuellem 3D-Stadtmodell und Bildinformationen,
2. eine der beabsichtigten Anwendung *angemessene Bildauflösung* sowie
3. eine *ausreichende Korrespondenz* von Bild- und Geometriedaten, d. h. eine weitgehende Übereinstimmung der im Bild sichtbaren Gebäudeform mit der korrespondierenden Geometrie im virtuellen 3D-Stadtmodell.

Komplexe, fotogrammetrische Verfahren (Gruber u. a., 2008) stellen die Korrespondenzbedingung nicht, da sie sowohl das geometrische Modell als auch die Texturen direkt aus den gleichen Bildinformationen gewinnen. Allerdings können diese Verfahren nicht für die Texturierung anderweitig erzeugter virtueller 3D-Stadtmodelle verwendet werden. Dieser Anwendungsfall ist jedoch relevant, da das zu texturierende virtuelle 3D-Stadtmodell z. B.

- nichtfotogrammetrisch abgeleitet wurde,

- mit Fachinformationen angereichert wurde, deren Neueingabe unverhältnismäßig hohe Kosten erfordern würde,
- festgelegte Genauigkeitsanforderungen erfüllt, z. B. im öffentlichen Sektor die Entsprechung mit einem Kataster oder in der Computerspielebranche eine festgelegte Polygonanzahl, oder
- im Rahmen der Fortführung mit aktuellen Bilddaten texturiert werden soll.

Bei der Texturierung vorhandener virtueller 3D-Stadtmodelle treten eine Reihe praktischer Probleme auf, z. B.:

- Das virtuelle 3D-Stadtmodell kann insbesondere bei automatischer Erfassung Ungenauigkeiten (Abweichungen zwischen realer und virtueller Gebäudeform) enthalten.
- Das virtuelle 3D-Stadtmodell kann Fehler (fehlende oder falsche Gebäude) enthalten.
- Die Lagegenauigkeit der Schrägluftbilder kann stark variieren.
- Die Qualität der Schrägluftbilder kann durch atmosphärische Einflüsse oder Beleuchtungsbedingungen mangelhaft sein.
- Die Abdeckung des zu texturierenden Gebiets kann unvollständig sein.
- Bildinformationen und virtuelles 3D-Stadtmodell können auf Grund unterschiedlicher Erfassungszeitpunkte punktuell inkonsistent sein.

Ziel dieser Arbeit ist der Entwurf und die Umsetzung eines automatisierten Arbeitsablaufs und der dazugehörigen Verfahren für die Texturierung gegebener großflächiger virtueller 3D-Stadtmodelle aus gegebenen Schrägluftbildern. Dabei steht die praktische Anwendbarkeit unter Berücksichtigung der genannten Probleme und die weitgehende Unabhängigkeit von der Erfassungstechnologie sowohl des virtuellen 3D-Stadtmodells als auch der Bildinformationen im Vordergrund. Die Einsatzfähigkeit wird durch die Texturierung mehrerer virtueller 3D-Stadtmodelle verschiedener Herkunft und Dimension belegt. Diese Arbeit wurde in einem Pilotprojekt mit der Firma Autodesk GmbH auf Basis der Autodesk LandXplorer-Plattform erprobt; die Technologie wird mittlerweile im Produkt „Autodesk CityFactory“ eingesetzt.

3.1 Verwandte Arbeiten

Die Fototexturierung von 3D-Geometrie ist ein aktives Forschungsfeld in Computergrafik und Fotogrammetrie. Prinzipiell kann zwischen zwei Bereichen unterschieden werden: Dem Scannen von üblicherweise kleinen Objekten unter kontrollierten Bedingungen und der Erfassung von üblicherweise großen Objekten oder Objektgruppen im Rahmen von Nahbereichs- oder Fernerkundung. Die vorliegende Arbeit ist mit dem zweiten Bereich befasst.

Zu den ersten Arbeiten, die Texturen betrachten, gehört View-dependent Texture Mapping (Debevec u. a., 1996). Dieses nicht echtzeitfähige Verfahren arbeitet bildbasiert, indem es passend zum Blickwinkel der virtuellen Kamera ein georeferenziertes Bild mit ähnlichem Blickwinkel auswählt bzw. mehrere verblendet. Unter Einbeziehung von stereoskopischen Prinzipien lassen sich auch Parallaxenfehler korrigieren. Mit Hilfe projektiver Texturierung ist das Verfahren echtzeitfähig (Debevec u. a., 1998).

Im Rahmen eines fotogrammetrischen Modellerzeugungsprozesses wurde von Bornik u. a. (2001) ein Texturextraktionsmechanismus entwickelt. Die Extraktion beruht auf einer Quadtreedatenstruktur, die sich an der metrischen Auflösung der Bildaufnahmen orientiert (Ofek u. a.,

1997). Zusätzlich zur modellgeometriebezogenen Verdeckungsdetektion integrieren Bornik u. a. (2001) die Entfernung von Störobjekten durch die Überlagerung der verfügbaren Ansichten und Verwendung eines Medianfilters.

Im Kontext der Kulturgutdokumentation beschreiben Grammatikopoulos u. a. (2004) einen Ansatz zur Texturierung von aus Laserscans gewonnenen 3D-Oberflächen. Ihr Verfahren bestimmt modellbasiert Verdeckungen in den Quellbildern und führt Farbinterpolationen zwischen verschiedenen Ansichten einer Fläche zur Texturgenerierung durch. Im Unterschied zu der hier vorgestellten Methode benötigt es präzise innere und äußere Orientierungsdaten für die Bilder, die durch Bündelblockausgleichung sicherstellt werden. Ein ähnlicher Ansatz findet sich bei Alshawabkeh und Haala (2005).

Ein Verfahren zur Texturierung geometrischer virtueller 3D-Stadtmodelle mit Hilfe von Schrägluftbildern, das eine wichtige Grundlage für die vorliegende Arbeit darstellt, wird von Früh u. a. (2004) beschrieben. Ein erster Schritt ist die Georeferenzierung der aufgenommenen Schrägluftbilder. Danach wird für jedes Dreieck lokal das optimale Quellbild anhand einer Maßzahl bestimmt, die aus metrischer Auflösung, Verdeckungsgrad und Blickwinkel berechnet wird. Anschließend wird die Quellbildauswahl anhand einer Mehrheitsentscheidung in umliegenden Dreiecken optimiert, um visuelle Artefakte durch Texturübergänge zu minimieren. Eine texelpräzise Komposition erfolgt nicht.

Die Verwendung programmierbarer 3D-Grafikhardware zur Projektion terrestrischer Aufnahmen auf ein 3D-Gebäudemodell wird in Kada u. a. (2005) beschrieben. Bei der Kombination mehrerer Quellbilder beschränkt sich das Verfahren jedoch auf die Erkennung von Selbstverdeckungen. Der Ansatz ist deshalb vorwiegend für freistehende Gebäude bestimmt.

Ein weiteres Verfahren verwendet luftgestützte Videoaufnahmen für die Texturierung von 3D-Gebäudenmodellen (Wu u. a., 2007). Aus dem Videostrom werden für jede Fläche zwei verdeckungsarme rektifizierte Kandidatentexturen ausgewählt, von denen die mit der höheren metrischen Auflösung als finale Textur verwendet wird. Dabei erfolgt die Verdeckungserkennung über die maximale paarweise Korrelation der Kandidatentexturen und nicht unter Zuhilfenahme des virtuellen 3D-Stadtmodells selbst. So kann die Texturauswahl auch nicht explizit modellierte Details berücksichtigen.

Einen im Hinblick auf die verwendeten Eingabedaten besonderen Ansatz verfolgen Stilla u. a. (2009). Das Bildmaterial ist ein Videostrom aus dem Spektrum thermaler Infrarotstrahlung. Das Verfahren basiert grundsätzlich auf einem pixelweisen Verdeckungstest. Durch das besondere Bildmaterial werden andere Kalibrierungsverfahren und Verzerrungsmodelle benötigt.

Neben den Ergebnissen aus der Forschung gibt es bereits auch eine Reihe kommerzieller Systeme, die jedoch kaum vertrieben werden, da sie Eigenentwicklungen von Datenerzeugern sind. Die Funktionsweise oder Leistungsparameter dieser Systeme lassen sich schwer ermitteln. Aktive Firmen sind insbesondere Google und Microsoft, die für ihre jeweiligen Produkte GoogleEarth und VirtualEarth voll texturierte virtuelle 3D-Stadtmodelle aufbauen. Beispielhaft genannt sei zudem die Firma BLOM, die Texturierung auf Basis von Pictometry-Bildern anbietet (Jurisch und Mountain, 2008; Karbo und Schroth, 2009). Daneben finden sich meist wenig automatisierte Funktionen zur Texturierung in Fotogrammetriesystemen, wie zum Beispiel der TexelMapper im ERDAS Stereo Analyst von Leica (Smith u. a., 2009).

Für die Rekonstruktion von Einzelgebäuden aus Fotos gibt es verschiedene Softwaresysteme, wie z. B. den Autodesk ImageModeler (Autodesk, 2010) oder den Eos Systems PhotoModeler (EOS Systems, 2010), die auf einfache Datenerfassung ausgelegt sind. Sie verlangen keine

georeferenzierten Fotos, sondern erlauben die manuelle Identifikation von Passpunkten zur automatischen Bestimmung der relativen inneren und äußeren Orientierungsdaten aller Fotos. Das so aufbereitete Bildmaterial wird zur Unterstützung der manuellen Gebäuderekonstruktion verwendet. Das erzeugte 3D-Modell lässt sich mit dem Bildmaterial texturieren.

3.2 Theoretische Grundlagen

Das für die Fototexturierung wesentliche Element ist die Rückprojektion der aufgezeichneten Bildinformationen auf das virtuelle 3D-Stadtmodell. Der hier vorgestellte Ansatz arbeitet mit Schrägbildaufnahmen, die z. B. während eines Bildfluges mit in geeigneten Aufhängungssystemen installierten Digitalkameras aufgenommen werden können. Die für die hier betrachteten Schrägluftbilder gültigen mathematischen Grundlagen in Form von Kamera- und Linsenmodellen wurden in den Bereichen Computer Vision (Forsyth und Ponce, 2002) und Fotogrammetrie (Kraus, 1997) entwickelt. Die aufnehmenden Kameras unterliegen dem Lochkameramodell und folgen in erster Näherung den Gesetzen der perspektivischen Projektion. Das Lochkameramodell wird durch die innere und äußere Orientierung beschrieben (Kraus, 1997).

Die *äußere Orientierung* beschreibt die Transformation eines Punkts P mit den Objektkoordinaten $(x, y, z)^{\mathrm{T}}$ in Kamerakoordinaten $(x_{\mathrm{c}}, y_{\mathrm{c}}, z_{\mathrm{c}})^{\mathrm{T}}$. Die Parameter der äußeren Orientierung sind die Position des Projektionszentrums C und die Lage der Kamera, hier durch die Eulerwinkel ω, ϕ und κ beschrieben. Die Transformation lautet:

$$R_{\omega\phi\kappa} = \begin{pmatrix} \cos\phi\cos\kappa & -\cos\phi\sin\kappa & \sin\phi \\ \cos\omega\sin\kappa + \sin\omega\sin\phi\cos\kappa & \cos\omega\cos\kappa - \sin\omega\sin\phi\sin\kappa & -\sin\omega\cos\phi \\ \sin\omega\sin\kappa - \cos\omega\sin\phi\cos\kappa & \sin\omega\cos\kappa + \cos\omega\sin\phi\sin\kappa & \cos\omega\cos\phi \end{pmatrix} \tag{3.1}$$

$$\begin{pmatrix} x_{\mathrm{c}} \\ y_{\mathrm{c}} \\ z_{\mathrm{c}} \end{pmatrix} = R^{-1}_{\omega\phi\kappa} \cdot \left(\begin{pmatrix} x \\ y \\ z \end{pmatrix} - C \right) \quad [\mathrm{m}] \tag{3.2}$$

Die *innere Orientierung* beschreibt die eigentliche Projektion in die Bildebene. Dabei wird ein Punkt in Kamerakoordinaten in Bildkoordinaten $(x_{\mathrm{i}}, y_{\mathrm{i}})^{\mathrm{T}}$ bzw. im Falle einer Digitalkamera in Pixelkoordinaten $(x_{\mathrm{p}}, y_{\mathrm{p}})^{\mathrm{T}}$ transformiert. Die Parameter der inneren Orientierung umfassen die Kamerakonstante bzw. Brennweite c (in mm), den Bildhauptpunkt bzw. das optische Bildzentrum H (in mm), die Sensorgröße $s_{\mathrm{x}} \times s_{\mathrm{y}}$ (in mm) und die physische Bildauflösung $r_{\mathrm{x}} \times r_{\mathrm{y}}$ (in Pixeln). Die Transformation wird berechnet durch:

$$\begin{pmatrix} x_{\mathrm{i}} \\ y_{\mathrm{i}} \end{pmatrix} = H - c \begin{pmatrix} x_{\mathrm{c}}/z_{\mathrm{c}} \\ y_{\mathrm{c}}/z_{\mathrm{c}} \end{pmatrix} \quad [\mathrm{mm}] \tag{3.3}$$

$$\begin{pmatrix} x_{\mathrm{p}} \\ y_{\mathrm{p}} \end{pmatrix} = \begin{pmatrix} r_{\mathrm{x}}\left(x_{\mathrm{i}}/s_{\mathrm{x}} + 0{,}5\right) \\ r_{\mathrm{y}}\left(y_{\mathrm{i}}/s_{\mathrm{y}} + 0{,}5\right) \end{pmatrix} \quad [\mathrm{Pixel}] \tag{3.4}$$

Dieses vereinfachte mathematische Modell setzt die Verwendung einer idealen Linse voraus. Eine reale Linse erzeugt verschiedene Verzerrungen in der Abbildung (Verzeichnungen), die sich durch zusätzliche Korrekturfunktionen formulieren lassen. Geometrische Verzerrungen lassen sich mathematisch durch ein Modell mit mehreren Komponenten beschreiben. Den

gewöhnlich größten Einfluss hat die radiale Verzerrung (Kissen- oder Tonnenverzerrung), welche den Abstand r eines projizierten Punktes zum optischen Bildzentrum nichtlinear beeinflusst. Der korrigierte Abstand r_{rad} ergibt sich aus dem gemessenen, verzerrten Abstand r als:

$$r_{\text{rad}}(r) = r + k_1 r^3 + k_2 r^5 + k_3 r^7 \tag{3.5}$$

Die Parameter k_1, k_2 und k_3 werden durch Näherungsverfahren aus einer gemessenen Wertetabelle bestimmt. Andere geometrische Verzerrungskomponenten sind Trapez- und Parallelverzerrung. Zusätzlich erzeugt eine reale Linse auch farbliche Verzerrungen wie Vignettierung durch Lichtabsorbtion im Objektivgehäuse und Farbabberation durch die Wellenlängenabhängigkeit der Lichtbrechung (Forsyth und Ponce, 2002). Es existieren verschiedene Verfahren zur Kompensation, die in gängigen Bildverarbeitungsprogrammen (z. B. Adobe Photoshop) verfügbar sind und hier nicht näher betrachtet werden sollen.

Für die Bestimmung der inneren und äußeren Orientierung sowie der Verzerrungsparameter haben Computer Vision und Fotogrammetrie verschiedene Verfahren entwickelt. Die Verzerrungsparameter und innere Orientierung werden für gewöhnlich für jede Kamera durch Kalibrierung an einem Teststand gewonnen und regelmäßig überprüft (Forsyth und Ponce, 2002). Die äußere Orientierung muss für jedes Bild durch die Georeferenzierung bestimmt werden (Kap. 2.2.1).

3.3 Texelpräzise Texturerzeugung

Die Texturerzeugung benötigt zwei Eingaben: ein virtuelles 3D-Stadtmodell und flächendeckendes, georeferenziertes Bildmaterial. Im Rahmen dieser Arbeit wird o. B. d. A. angenommen, dass das virtuelle 3D-Stadtmodell geometrisch aus Dreiecken aufgebaut ist (Abb. 3.1). Polygonale virtuelle 3D-Stadtmodelle können immer in diese Form überführt werden, Stadtmodelle in anderen geometrischen Repräsentationsformen, z. B. Constructive Solid Geometrie (CSG, Foley u. a., 1995) oder Freiformflächen (Foley u. a., 1995), müssen geeignet konvertiert bzw. approximiert werden. Das Bildmaterial besteht im Folgenden aus Schrägluftbildern (Abb. 2.3). Die Kalibrierungsdaten der Aufnahmekameras sind bekannt und jedes Bild ist einzeln georeferenziert. Das Bildmaterial sollte das gesamte virtuelle 3D-Stadtmodell vollständig abdecken. Zur Sicherstellung dieser Eigenschaft werden Schrägluftbilder typischerweise aus vier oder mehr Richtungen mit hoher Überlappung aufgenommen. Der Neigungswinkel von 35° bis 45° ermöglicht die weitgehend verdeckungsfreie Erfassung der Fassaden.

Die hohe Dichte des Bildmaterials führt dazu, dass ein großer Teil des virtuellen 3D-Stadtmodells in mehreren Bildern sichtbar ist. Einem Dreieck des virtuellen 3D-Stadtmodells können daher mehrere Bildausschnitte zugeordnet werden, in denen es potentiell sichtbar ist. Idealisiert zeigen alle Bildausschnitte eine identische Fläche unter verschiedenen Bedingungen, wie z. B. Blickwinkel oder Beleuchtung. Ziel ist nun die Rekonstruktion der zu Grunde liegenden visuellen Eigenschaften der Fläche. Detaillierte visuelle Eigenschaften werden in einer positionsabhängigen bidirektionalen Reflexionsverteilungsfunktion (spatially varying bidirectional reflection distribution function, SVBRDF, Akenine-Möller u. a., 2008) kodiert. Die Bestimmung der SVBRDF ist unter kontrollierten Bedingungen möglich (Bernardini u. a., 2001; Weyrich u. a., 2008); entsprechende Scanner werden praktisch eingesetzt (Akenine-Möller u. a., 2008). Im vorliegenden Fall sind die Bedingungen nicht kontrollierbar und ein Rückschluss auf die tatsächlichen Oberflächeneigenschaften kaum möglich. Eine gute Näherung ist die Komposition einer Farbtextur

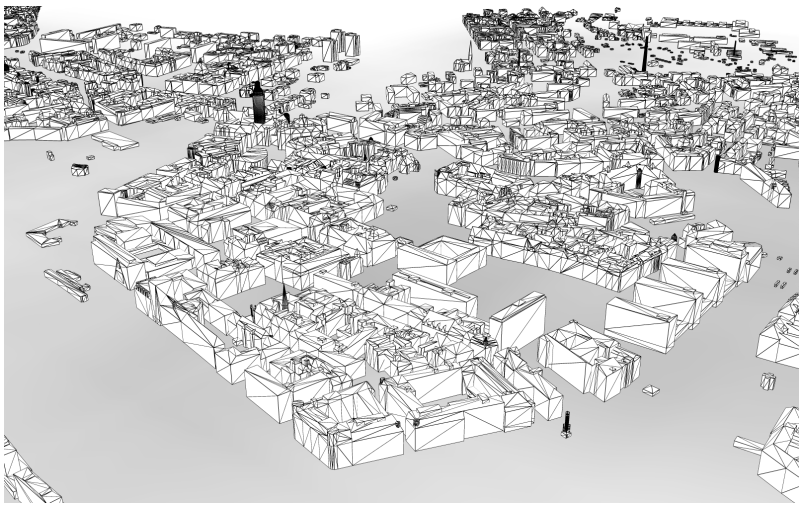


Abb. 3.1: Beispielhafte, aus Dreiecken aufgebaute Geometrie eines virtuellen 3D-Stadtmodells.

aus dem verfügbaren Bildmaterial. Dazu müssen die verschiedenen Bildausschnitte, die eine konkrete Fläche zeigen, extrahiert und anschließend zu einer einzigen Textur zusammengeführt werden. Die Bildausschnitte unterscheiden sich in ihrem Informationsgehalt, der von vielen Aspekten wie z. B. Abbildungsschärfe, Belichtungsqualität, spekularen Effekten, Verdeckungen oder flächenbezogener metrischer Auflösung beeinflusst wird. Da diese Aspekte auch lokal variieren, muss die Komposition auf einem lokal bestimmten Qualitätskriterium beruhen.

Die Extraktion der Bildausschnitte und deren Komposition werden für eine effiziente Verarbeitung großer Datensätze getrennt. Bei der Extraktion werden die zu einer Fläche gehörenden, je nach Blickwinkel unterschiedlich dimensionierten Bildausschnitte zur Vereinfachung der nachfolgenden Schritte in *Rohtexturen*, d. h. Texturraster mit einheitlicher, der finalen Textur entsprechenden Parametrisierung, überführt. Für jedes Texel der Rohtexturen kann dann das lokale Qualitätskriterium bestimmt und in Form zusätzlicher *Qualitätsraster* gespeichert werden. Die Komposition der Textur einer Fläche erfolgt anschliessend texelpräzise direkt aus den entsprechenden Rohtexturen und zugehörigen Qualitätsrastern. Durch die einheitliche Parametrisierung ist keine weitere Transformation der Bilddaten notwendig. Die Schritte der Texturerzeugung für ein gegebenes Dreieck des virtuellen 3D-Stadtmodells sind somit:

1. Texturzuordnung und Texturparametrisierung
2. Extraktion aller relevanten Bildausschnitte (Rohtexturen)
3. Texelpräzise Berechnung des Qualitätskriteriums pro Rohtextur (Qualitätsraster)
4. Komposition der Textur auf Basis der zugehörigen Rohtexturen und Qualitätsraster

Im Folgenden wird auf die einzelnen Schritte aus einer algorithmischen Sicht eingegangen.

3.3.1 Texturzuordnung und Texturparametrisierung

Mit der Texturzuordnung wird die Texturierung vorbereitet. Dabei muss jedem Dreieck eindeutig ein Texturausschnitt (Buchholz, 2006) zugeordnet werden, der sich in einem Texturraster befindet. Bei einer naiven Zuordnung wird für jedes Dreieck ein Texturraster erzeugt. Da Texturraster

notwendigerweise rechteckig sind, entsteht bei diesem Ansatz immer mindestens 50% Verschnitt, d. h. es wird mehr als doppelt so viel Speicher für die Texturraster belegt wie nötig wäre. Ein Ansatz zur Verschnittreduktion sind Texturatlanten (Wloka, 2005). Dabei werden mehrere Texturausschnitte zu einem größeren Bild zusammengefasst sowie die Texturparametrisierung und Texturzuordnung angepasst. Eine einfache Zusammenfassung ganzer Texturraster führt nicht zu einer Verschnittreduktion, eher im Gegenteil zu einer Verschnitterhöhung, da normalerweise nicht der gesamte Platz im Atlas genutzt werden kann (Buchholz, 2006). Zusätzlich ergeben sich bei der naiven Zuordnung visuelle Artefakte. An den Grenzen der Texturausschnitte werden bei der Texturfilterung (Foley u. a., 1995) auch außerhalb des Dreiecksbereichs liegende Texel mit einbezogen. Sind diese Texel unbelegt, entstehen Farbfehler an den Dreieckskanten. Besonders deutlich sind diese Artefakte, wenn räumlich zusammenhängende Dreiecke nicht zusammenhängende Texturbereiche nutzen.

Die hier gewählte Lösung zur gleichzeitigen Verschnitt- und Artefaktreduktion nutzt eine Eigenschaft virtueller 3D-Stadtmodelle: Gebäude bestehen zu großen Teilen aus ebenen, meist rechteckigen Flächen. Liegen zusammenhängende Dreiecke in einer Ebene, können sie verzerrungsfrei auf zusammenhängende Texturausschnitte in einem gemeinsamen Texturraster abgebildet werden. Dies entspricht einer verschnittreduzierenden Zusammenfassung i. S. v. Texturatlanten. Zudem beschränken sich Texturkanten auf deutlich wahrnehmbare Kanten im virtuellen 3D-Stadtmodell und Artefakte sind weniger auffällig.

Für die Texturzuordnung muss das virtuelle 3D-Stadtmodell in Mengen koplanarer, zusammenhängender Dreiecke, sog. *Zusammenhangskomponenten*, unterteilt werden. Hierzu wird ein zweistufiges Verfahren eingesetzt:

1. Alle Eckpunkte werden indiziert, wobei zwei Eckpunkte, deren Abstand kleiner als ein gegebener Mindestabstand ist (z. B. 5cm), den gleichen Punktindex bekommen. Zur Beschleunigung dieser Operation wird ein Octree verwendet (Akenine-Möller u. a., 2008).
2. Es werden Zusammenhangskomponenten auf Basis der Punktindizes gesucht. Eine Zusammenhangskomponente wird durch eine Menge von Punktindizes, eine Menge von Dreiecksindizes und eine Normale repräsentiert. Initial stellt jedes Dreieck eine eigene Komponente dar. In einer Schleife werden nun Komponenten zusammengefasst, die gleiche Normalen und mindestens zwei gemeinsame Punktindizes haben. Die Forderung nach zwei gemeinsamen Eckpunkten verbessert die Texturauslastung, da so ungünstige Konstellationen mit nur einem Verbindungspunkt vermieden werden. Die Schleife terminiert, wenn keine Zusammenfassungen mehr möglich sind.

Dieses Verfahren hat auf Grund der zweiten Stufe eine ungünstige Laufzeitkomplexität von mindestens $O(n^3)$ mit n als Eckpunktanzahl. Es muss jedoch nur auf Einzelgebäude angewendet werden, die aus relativ kleinen Punktmengen bestehen, da Gebäude als separate Objekte erhalten bleiben sollen und eine übergreifende Zusammenfassung daher nicht möglich ist. Die Dreieckszusammenfassung stellt somit keinen wesentlichen Leistungsengpass dar.

Nach Abschluss wird jeder Zusammenhangskomponente ein Texturraster zugeordnet und die Parametrisierung bestimmt. Hierzu wird das minimale orientierte Begrenzungsrechteck der Dreiecke in der gemeinsamen Ebene ermittelt. Die Abmessungen des Begrenzungsrechtecks bestimmen die physische Auflösung des Texturrasters, da die gewünschte metrische Auflösung durch den Benutzer vorgegeben wird. Die in der Erprobung genutzten Schrägluftbilder liefern

typischerweise eine metrische Auflösung von bestenfalls 15cm pro Pixel, die jedoch je nach Flächenorientierung und Position im Bild stark variiert. Eine sinnvolle metrische Auflösung des Texturrasters zur Vermeidung von Qualitätsverlusten ist deshalb 10cm pro Texel. Nach der Definition des Texturrasters wird die Parametrisierung in Form von Texturkoordinaten berechnet. Die Texturkoordinaten entsprechen den baryzentrischen Koordinaten der Dreieckseckpunkte innerhalb des Begrenzungsrechtecks.

3.3.2 Extraktion aller relevanten Bildausschnitte

Für die Sammlung des Bildmaterials müssen alle Schrägluftbilder identifiziert werden, in denen eine konkrete Zusammenhangskomponente sichtbar ist. Anschliessend können die Rohtexturen durch Rückprojektion der Schrägluftbilder auf die Zusammenhangskomponente extrahiert werden.

Ein wesentliches Element der Extraktion ist die Kompensation von Linsenverzerrungen. Die nötigen Parameter (Kap. 3.2) sind aus der Kalibrierung der zugehörigen Kameras bekannt. Eine einfache Möglichkeit ist eine separate Entzerrung, bei der alle Bilder mittels geeigneter Softwarepakete (z. B. Adobe Photoshop) oder Bildverarbeitungsbibliotheken (z. B. Intel Performance Primitives) entzerrt und gespeichert werden. Bei diesem Ansatz kommt es jedoch zu Qualitätsverlusten durch Interpolation und erneute evtl. verlustbehaftete Bildkompression. Die gewählte Alternative ist eine direkt in den Extraktionsprozess integrierte Entzerrung ohne Erzeugung und Speicherung eines Zwischenbildes.

Die Extraktion erfolgt mit Hilfe von 3D-Grafikhardware, wobei das Schrägluftbild als Textur ausgelesen wird. Aus der inneren und äußeren Orientierung wird eine perspektivische Transformationsmatrix bestimmt, mit deren Hilfe die Texturkoordinaten zu einem 3D-Punkt entsprechend dem idealen Lochkameramodell berechnet werden können. Anschliessend werden diese Texturkoordinaten zur Kompensation der radialen Verzerrung angepasst. Dabei muss die Inverse $r(r_{\mathrm{rad}})$ der bei der Kalibrierung bestimmten Radialverzerrung $r_{\mathrm{rad}}(r)$ (Gl. 3.5) eingesetzt werden. Eine analytische Invertierung des zugehörigen Polynoms bzw. die Verwendung lokaler Näherungsverfahren (z. B. Horner-Schema oder Newton-Verfahren) sind ineffizient. Stattdessen wird die Inverse durch ein Polynom gleichen Grades approximiert, dessen Koeffizienten durch die Kleinste-Quadrate-Methode auf Basis einer aus $r_{\mathrm{rad}}(r)$ abgeleiteten Wertetabelle bestimmt werden. Um die typischerweise sehr kleinen Koeffizienten k_1 bis k_3 in einem für eine genaue Berechnung im 32-Bit-Fließkommaformat sinnvollen Wertebereich zu halten, wird die Bildgröße vorher normalisiert.

Die OpenGL-Grafikpipeline ermöglicht diese Extraktion auf einfache Art und Weise. Es ist ausreichend, eine Kamera zu spezifizieren, deren Sichtvolumen mit einer Orthogonalprojektion das oben bestimmte Begrenzungsrechteck der Zusammenhangskomponente umschließt und senkrecht über der gemeinsamen Ebene platziert ist. Werden dann alle Dreiecke der Zusammenhangskomponente unter Verwendung des beschriebenen Texturzugriffs gerendert, ergibt das Bild eine Rohtextur. Zur Vermeidung von Texturartefakten am Komponentenrand wird zusätzlich das gesamte Begrenzungsrechteck gerendert, um so nicht abgedeckte Texturbereiche mit sinnvollen Bilddaten zu füllen. Das erzeugte Bild wird ausgelesen und als Rohtextur in einer Bilddatenbank gespeichert.

Faktor	*Beschreibung*
	bildbezogen
Physische Auflösung	Obere Grenze für die effektive Auflösung des Schrägluftbildes.
Tiefenschärfe	Güte der Fokussierung. Bei Autofokusproblemen kann bei einzelnen Schrägluftbildern die erreichte effektive Auflösung von der physischen erheblich abweichen.
Beugungsunschärfe	Durch Lichtbeugung im Objektiv bedingte blendenabhängige Unschärfe. Reduziert die effektive Auflösung.
Bewegungsunschärfe	Durch zu lange Belichtungszeit bei Flugzeugbewegung und Vibration bedingte Unschärfe. Reduziert die effektive Auflösung.
	transformationsbezogen
Flächenausrichtung	Winkel einer Fläche zur Kameraebene. Beeinflusst die metrische Auflösung auf der Fläche.
Lage im Bild	Position der Fläche im Bild bei gleicher Ausrichtung. Beeinflusst die metrische Auflösung auf der Fläche.

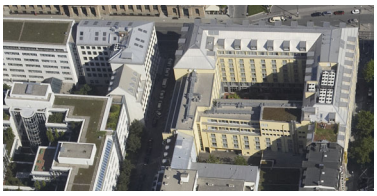
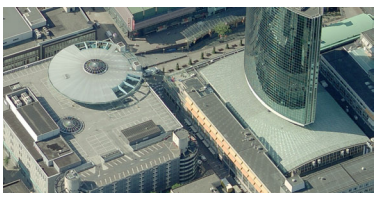
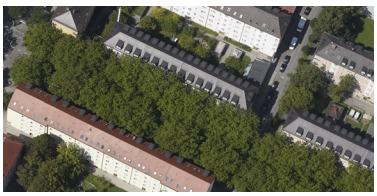
Tab. 3.1: Wesentliche Einflussfaktoren für die effektive Auflösung von Rohtexturen.

3.3.3 Texelpräzise Berechnung des Qualitätskriteriums

Mit Hilfe des Qualitätskriteriums wird das optimale Rohtexturtexel aus einer Menge von Rohtexturtexeln gewählt. Idealisiert beschreiben alle Rohtexturtexel dieser Menge ein identisches Flächenstück, werden jedoch aus Bilddaten unter verschiedenen Aufnahmebedingungen generiert. Damit fließt unterschiedlich viel Information in die Erzeugung des jeweiligen Rohtexturtexels ein, die zusätzlich eine variable Menge an Störeinflüssen enthält. Die Leitidee für die Ableitung des Qualitätskriteriums ist die Bestimmung der tatsächlich für das Flächenstück verfügbaren Informationsmenge.

Die für die Erzeugung eines Rohtexturtexels theoretisch verfügbare Informationsmenge wird durch die lokale effektive Auflösung der Rohtextur beschrieben, die sich aus der effektiven Auflösung des Quellschrägluftbildes und den Eigenschaften der Extraktionstransformation ableitet. Tab. 3.1 führt wesentliche Einflussfaktoren auf. Die praktisch verfügbare Informationsmenge reduziert sich durch eine Reihe von Störeinflüssen. Hierbei bleibt die effektive Auflösung unverändert, die Daten enthalten jedoch nicht nur Informationen über das zum Rohtexturtexel gehörende Flächenstück, sondern auch zu Umwelteinflüssen. Wesentliche Einflüsse sind in Tab. 3.2 aufgeführt.

Für eine umfassende Qualitätsbewertung müssten alle aufgeführten Einflüsse gemessen und unter Berücksichtigung möglicher Korrekturen bewertet werden. Eine objektive und verlässliche Messung ist jedoch nur für eine geringe Zahl der Faktoren möglich. Insbesondere die Messungen der Umwelteinflüsse gehen über einfache Bildverarbeitung hinaus und erfordern die Interpretation der Bildinhalte. Die automatische, maschinelle Interpretation von Bilddaten ist dabei ein aktives Forschungsfeld (Computer Vision, maschinelles Sehen). Verfügbare Ansätze setzen meist kontrollierte Bedingungen voraus oder haben große Bewertungsunsicherheiten

Faktor	*Beschreibung*	*Beispiel*
Belichtung	Konsistenz der Belichtung. Bei automatischer Belichtung kommt es zu deutlichen Farbabweichungen, bei fester Belichtung evtl. zu Über- oder Unterbelichtung.	
Dunst	Trübung des gesamten Bildes auf Grund von Lichtstreuung an mikroskopischen atmosphärischen Schwebepartikeln.	
Schattenwurf	Schlagschatten von benachbarten Objekten auf Gebäudeoberflächen.	
Beleuchtung	Veränderliche Lichtbedingungen während des Bildflugs, z. B. Wolkenschatten oder Sonnenstand.	
Lichthöfe	Lokale Überstrahlung verursacht durch punktuelle intensive Reflexion der Sonne direkt in die Kamera.	
Reflexionen	Unerwünschte Bestrahlung von Oberflächen durch reflektiertes Sonnenlicht.	
Verdeckungen	Verdeckung einer Oberfläche durch andere Objekte, z. B. Häuser, Vegetation, Baukräne.	

Tab. 3.2: Typische Störeinflüsse in Rohtexturen.

(z. B. Farag u. a., 2008; Kim u. a., 2009). Folglich ist es derzeit fraglich, ob die Einbeziehung automatischer Bildinterpretation die Qualität der Fassadentexturen erhöht. Nicht zuletzt ist die Bildinterpretation vergleichsweise rechenaufwändig, was einer Nutzung für große Datensätze zum jetzigen Zeitpunkt noch entgegen spricht.

Aus praktischen Erwägungen bezüglich Verlässlichkeit und Berechnungsaufwand konzentriert sich die hier verwendete Qualitätsbewertung auf vier Einflüsse: die physischen Schrägluftbildauflösung, die Lage im Bild, die Flächenausrichtung und die Verdeckung. Aus den ersten drei Einflüssen lässt sich die effektive Auflösung der Rohtextur nach oben abschätzen. Diese Abschätzung wird durch den Flächeninhalt A der Rückprojektion eines Rohtexturtexels in das zugehörige Schrägluftbild ausgedrückt. Die Verdeckung lässt sich in Bezug auf das virtuelle 3D-Stadtmodell bestimmen und liefert damit eine konservative Abschätzung für die Gültigkeit eines Rohtexturtexels, ausgedrückt in einer Gültigkeitsklasse k.

Der Flächeninhalt A misst die Anzahl der Pixel und somit die theoretische maximale Informationsmenge, die für die Erzeugung eines Rohtexturtexels in einem Schrägluftbild zur Verfügung steht. In der Praxis verringert sich die tatsächliche Informationsmenge durch die oben genannten bildbezogenen Einflüsse. Im Folgenden bezeichnen P einen Punkt auf einer Fläche, dx die horizontale und dy die vertikale Ausdehnung eines Rohtexturtexels in dieser Fläche. Das Rechteck $R = (P, P+dx, P+dx+dy, P+dy)$ liegt in der Fläche und umrahmt ein Rohtexturtexel. Die 4x4-Matrix M bezeichnet die projektive Transformationsmatrix eines Schrägluftbildes (Kap. 3.3.2), die sich aus der inneren und der äußeren Orientierung zusammensetzt. Die einzelnen Zeilenvektoren von M sind mit m_1 bis m_4 bezeichnet. Unter der vereinfachenden Annahme, die Projektion des Rechtecks ergebe ein Parallelogramm, ergibt sich der Flächeninhalt A des projizierten Rechtecks aus dem Absolutbetrag des Kreuzproduktes der in die Bildebene projizierten Vektoren dx und dy in Abhängigkeit von P:

$$A = \left\| \left(\frac{1}{\langle m_4, P+dx \rangle} \begin{pmatrix} \langle m_1, P+dx \rangle \\ \langle m_2, P+dx \rangle \\ 0 \end{pmatrix} - \frac{1}{\langle m_4, P \rangle} \begin{pmatrix} \langle m_1, P \rangle \\ \langle m_2, P \rangle \\ 0 \end{pmatrix} \right) \times \left(\frac{1}{\langle m_4, P+dy \rangle} \begin{pmatrix} \langle m_1, P+dy \rangle \\ \langle m_2, P+dy \rangle \\ 0 \end{pmatrix} - \frac{1}{\langle m_4, P \rangle} \begin{pmatrix} \langle m_1, P \rangle \\ \langle m_2, P \rangle \\ 0 \end{pmatrix} \right) \right\| \tag{3.6}$$

$\langle \cdot, \cdot \rangle$ bezeichnet das Skalarprodukt. Die getroffene Annahme ist korrekt, wenn die Fläche parallel zur Bildebene ist. Andernfalls bewirkt die perspektivische Verkürzung eine Abweichung, die bei gegebener Brennweite vom Verhältnis zwischen Tiefendifferenz und Betrachtungsabstand abhängt. Im betrachteten Fall beträgt die Tiefendifferenz typischerweise weniger als 10cm (metrische Texturauflösung) bei Betrachtungsabständen von 500m bis 2000m (in der Größenordnung der Flughöhe). Die dabei auftretenden Abweichungen liegen deutlich unter einem Promille und sind vernachlässigbar.

Die Gültigkeit eines Rohtexturtexels setzt sich aus drei Komponenten zusammen:

1. der Verfügbarkeit von Quellbildpixeln,
2. der Zugewandtheit einer Fläche zur Kamera und
3. der messbaren Verdeckung im virtuellen 3D-Stadtmodell.

Die *Verfügbarkeit von Quellbildpixeln* betrifft Flächen, die nur teilweise in einem Schrägluftbild sichtbar sind. Rohtexturtexel, deren Projektion nicht mehr in das Quellbild fällt, haben

Klasse	*Beschreibung*
ungültig	Das Rohtexturtexel besitzt keine Quellbildpixel. Somit können keine Bilddaten extrahiert werden.
abgewandt	Das Rohtexturtexel fällt nicht in die Klasse „ungültig“ und ist Teil einer der Kamera abgewandten Fläche. Somit sind Bilddaten verfügbar, sie zeigen jedoch definitiv nicht die Fläche.
verdeckt	Das Rohtexturtexel fällt nicht in die Klassen „ungültig“ oder „abgewandt“ und ist durch andere Teile des virtuellen 3D-Stadtmodells verdeckt. Somit sind Bilddaten verfügbar, sie zeigen jedoch höchstwahrscheinlich nicht die Fläche.
gültig	Das Rohtexturtexel fällt nicht in die Klassen „ungültig“, „abgewandt“ oder „verdeckt“. Somit zeigen die Bilddaten höchstwahrscheinlich die zum Rohtexturtexel gehörende Fläche.

Tab. 3.3: Gültigkeitsklassen für Rohtexturtexel.

keine verfügbaren Quellbildpixel. Die *Zugewandtheit zur Kamera* wird aus dem Skalarprodukt zwischen Flächennormale und Blickrichtung bestimmt. Die *messbare Verdeckung* wird über Standardschattenverfahren aus der Computergrafik (Akenine-Möller u. a., 2008) bestimmt. Damit lassen sich alle Verdeckungen durch das virtuelle 3D-Stadtmodell mit der dem virtuellen 3D-Stadtmodell eigenen Präzision erkennen. Verdeckungen durch nicht erfasste Gebäudeteile wie z. B. Gauben oder durch andere, nicht im verwendeten virtuellen 3D-Stadtmodell enthaltene Objekte wie z. B. Vegetation, Baukräne oder Straßenmöbel können nicht detektiert werden. Aus allen drei Komponenten ergeben sich 4 Klassen, die in Tab. 3.3 aufgeführt sind.

Für die folgende Komposition wird pro Rohtextur ein sogenanntes Qualitätsbild mit gleichen Dimensionen erzeugt, das in jedem Pixel den Flächeninhalt A und die Klasse k des korrespondierenden Rohtexturtexels speichert. Die Erzeugung des Qualitätsbilds erfolgt auf der 3D-Grafikhardware, wobei A und k in einem Fragmentshader berechnet werden.

3.3.4 Komposition aller relevanten Bildausschnitte

Nach der Qualitätskriteriumsberechnung liegen für jede Zusammenhangskomponente eine Menge von Rohtexturen und dazugehörigen Qualitätsbildern vor. Unter der Annahme einer konsistenten Ausrichtung der Schrägluftbilder kann die Textur nun aus den Rohtexturen zusammengesetzt werden, indem aus allen Rohtexturtexeln an einer Position dasjenige mit der besten Qualitätsbewertung gewählt und in die finale Textur geschrieben wird.

Für eine korrekte Texturierung können einzig Texel der Klasse „gültig“ herangezogen werden. Um den visuellen Eindruck eines vollständig texturierten virtuellen 3D-Stadtmodells im Falle nicht erfasster Bereiche (z. B. Ladenzeilen oder Innenhöfen) zu verbessern, ist es jedoch sinnvoll, auch Texel der Klassen „verdeckt“ und „abgewandt“ zu verwenden. Nicht erfasste Bereiche werden damit zwar falsch texturiert, aber sie fallen so weniger ins Auge als sie es zum Beispiel bei Füllung mit einer konstanten Farbe tun (Abb. 3.2).

(a) Verwendung ungültiger Bilddaten.

(b) Verwendung konstanter Farbe (grau).

Abb. 3.2: Behandlung nicht bildlich erfasster Modelloberflächen. Die Füllung mit ungültigen Bilddaten stört den Eindruck eines vollständig texturierten virtuellen 3D-Stadtmodells weniger als die Verwendung einer konstanten Farbe.

Die Umsetzung erfolgt mittels der von DirectX-10-kompatibler 3D-Grafikhardware zur Verfügung gestellten *texture arrays*, die die Speichung gleichartiger 2D-Texturen in einer über die dritte Dimension indizierbaren Schichtung erlauben. Rohtexturen und Qualitätsbilder werden in zwei strukturell identischen texture arrays abgelegt. In einem Fragmentshader muss nun über alle Schichten iteriert werden um die Rohtextur mit der besten Qualität für das aktuelle Fragment zu identifizieren. Dieser Shader wird durch Rendern eines texturrasterüberdeckenden Rechtecks zur Ausführung gebracht. Die vollständige Ergebnistextur wird in einer Bilddatenbank gespeichert.

3.3.5 Datenanforderungen

Die texelpräzise Komposition der Texturen trifft die idealisierende Annahme, dass alle zu einem Texel korrespondierenden Rohtexturtexel dasselbe Oberflächenstück zeigen, in anderen Worten die Rohtexturinhalte konsistent sind. In der Praxis kommt es durch verschiedene Faktoren (Tab. 3.4) zu Abweichungen von dieser Annahme. Die Faktoren lassen sich in stadtmodellbezogene und bildbezogene unterteilen.

Stadtmodellbezogene Faktoren werden nicht im Rahmen des vorgestellten Ansatzes korrigiert, da die Geometrie des virtuellen 3D-Stadtmodells nicht notwendigerweise für die Texturierung, sondern für andere Anwendungen mit spezifischen Anforderungen erstellt wurde. Aus dem gleichen Grund kann auch keine automatische Fehlerkorrektur in der Geometrie (z. B. Schließung von Lücken, Korrektur von Flächenorientierungen, Generalisierung o. ä.) erfolgen. Vielmehr trifft das hier vorgestellte Verfahren die Annahme, dass es Aufgabe der Anwender ist, stadtmodellbezogene Fehler ihren Anforderungen entsprechend zu bereinigen. Das in Kap. 3.4 vorgestellte System bietet die Möglichkeit, erkannte Fehler zu dokumentieren, um die Korrektur zu erleichtern.

Bei den *bildbezogenen Faktoren* wird die Kamerakalibrierung, d. h. die Parameter der radialen Verzerrung und die Brennweite, als gegeben vorausgesetzt. Dementsprechend müssen die Parameter bei festgestellten Abweichungen extern neu bestimmt werden. Der einzige im Rahmen des vorgestellten Ansatzes direkt beeinflussbare Faktor ist die Ausrichtungsgenauigkeit der

Faktor	*Beschreibung*
	stadtmodellbezogen
Generalisierung	Beschränkung der Erfassung auf Objekte mit gewisser Mindestgröße. Kleinere Details wie z. B. Gauben, Schornsteine oder Fassadenvor- bzw. -rücksprünge fehlen.
Geometriefehler	Falsche Erfassung von Gebäudeform, -höhe oder Dachform.
Positionierungs-genauigkeit	Genauigkeit der Gebäudeposition. Kann durch Koordinatensystem-wechsel oder ungenaue Einrichtung der Erfassungssysteme schwanken.
	bildbezogen
Ausrichtungs-genauigkeit	Genauigkeit der äußeren Orientierungsdaten der Bilder, d. h. die Größe des Versatzes zwischen einer geometrischen Fläche und ihrem rückprojizierten Abbild.
Kamera-kalibrierungs-genauigkeit	Genauigkeit der Kamerakalibrierung, insbesondere der radialen Verzerrungsparameter. Die Kalibrierung kann sich bei längerem Betrieb verändern.

Tab. 3.4: Typische texturqualitätsminderde Störfaktoren. Die aufgeführten Faktoren stören die Grundannahme der konsistenten Rohtexturinhalte für die texelpräzise Texturkomposition und sollten minimiert werden.

Schrägluftbilder. Relevant ist hierbei nicht die Übereinstimmung der ermittelten äußeren Orientierung mit der wahren Lage und Ausrichtung der Kamera, sondern die korrekte Rückprojektion eines Oberflächenabbildes auf die korrespondierende Oberfläche im virtuellen 3D-Stadtmodell. Die Abweichung sollte für gute Ergebnisse weniger als 3 Pixel, d. h. typischerweise maximal 50cm, betragen. Der Fehler ist eine Kombination der Genauigkeiten von Kameraposition und -ausrichtung. Erste Schrägluftbildsysteme lieferten insbesondere auf Grund von verhältnismäßig ungenauen Winkelmessungen Abweichungen in der Größenordnung von 5 bis 10m am Boden. Verbesserungen und Kalibrierungen der Schrägluftbildsysteme haben diese Toleranz deutlich verkleinert, sodass inzwischen Abweichungen von unter 2m gängig sind. Bildbasierte automatische Verfahren zur Verbesserung der Positionierungsgenauigkeit (Früh u. a., 2004; Drauschke u. a., 2006) sind nicht sinnvoll einsetzbar, da die Toleranzen in Bezug auf die verfügbare physische Auflösung der Schrägluftbilder häufig in der Größenordnung des erwarteten Fehlers liegen. Deshalb ist immer noch eine manuelle Nachbearbeitung erforderlich um die Lageabweichungen auf weniger als 50cm zu reduzieren. Je nach vorliegender Winkelgenauigkeit gibt es dafür zwei verschiedene Ansätze:

1. Die Winkelgenauigkeit ist ausreichend hoch. In diesem Fall reicht eine einfache Verschiebung der Kameraposition.
2. Die Winkelgenauigkeit ist nicht ausreichend. In diesem Fall muss eine projektive Korrektur erfolgen, die durch vier Passpunkte definiert wird (Heckbert, 1989).

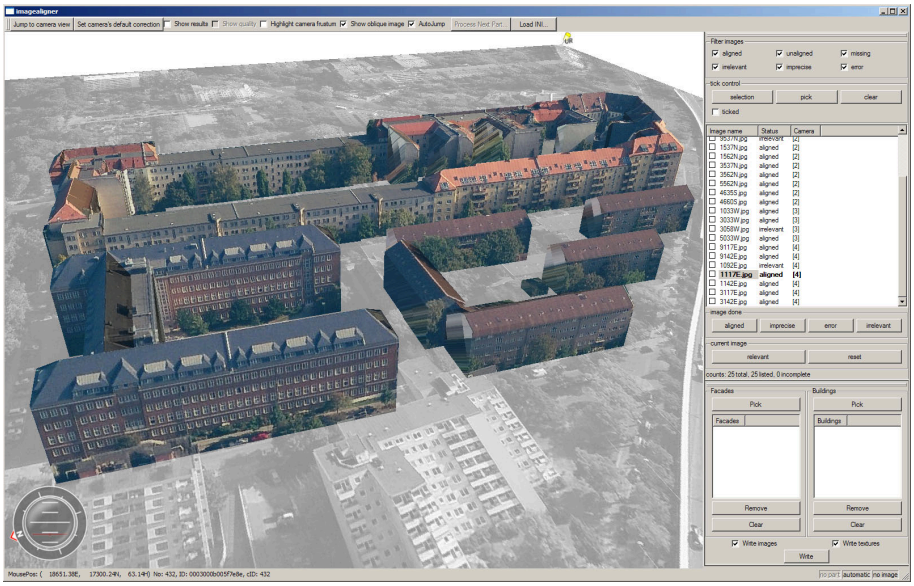


Abb. 3.3: Korrekturwerkzeug für äußere Orientierungsdaten von Schrägluftbildern. Das gerade aktive Schrägluftbild kann mittels direkter Manipulation verschoben oder verzerrt werden, um die Passgenauigkeit mit der Stadtmodellgeometrie zu erhöhen.

Für die Durchführung der Korrektur wurde ein interaktives Programm entwickelt, welches jeweils ein Schrägluftbild auf das untexturierte virtuelle 3D-Stadtmodell projiziert und die Lagekorrektur auf einfache Weise visuell interaktiv ermöglicht (Abb. 3.3). Mit Hilfe dieses Werkzeugs kann ein Schrägluftbild durch einen geübten Anwender in weniger als einer Minute ausgerichtet werden.

Eine wesentliche Anforderung über die genannten Faktoren hinaus ist die Verwendung des gleichen Koordinatensystems für alle raumbezogenen Daten. Diese Forderung ist an sich selbstverständlich, muss jedoch in der Praxis immer wieder geprüft werden. Insbesondere sind die verwendeten Koordinatensysteme nicht immer klar bzw. können in Details wie zum Beispiel dem Referenzdatum abweichen. Eine evtl. notwendige Transformation muss mit hoher Genauigkeit erfolgen. Werden die äußeren Orientierungsdaten der Schrägluftbilder transformiert, müssen neben der Position unbedingt auch die Winkel berücksichtigt werden. Entsprechende Transformationsfunktionen sind in gängigen Georeferenzierungs- und Geometrieerfassungssystemen enthalten.

3.3.6 Verbesserungen

Die texelpräzise Komposition der Flächentexturen fügt eine Textur aus mehreren Rohtexturen zusammen. Die Texelwahl berücksichtigt den Bildinhalt nicht, weshalb im Ergebnis störende Kanten entstehen können, die häufig von inkonsistenter Positionierung, Parallaxenfehlern, Beleuchtungsunterschieden oder nicht detektierbaren Verdeckungen herrühren. Da sich solche Artefakte auf Basis der derzeitigen Auswertung nicht gezielt vermeiden lassen, ist zumindest die Reduktion solcher zusätzlicher Fehler sinnvoll.

Ein Ansatz zur optischen Verringerung von Kanten ist Verblendung. Dabei wird ein sanfter Übergang zwischen angrenzenden Texturbereichen erzeugt. Das Innere der Texturbereiche

(a) Ohne Vermeidung von Rohtexturwechseln: Das Dach sind Texturfehler auf Grund von Lageungenauigkeiten und Parallaxenfehlern sichtbar.

(b) Mit Vermeidung von Rohtexturwechseln: Es wird nur ein einziges Schrägluftbild zur Texturerzeugung verwendet. Texturfehler werden vermieden.

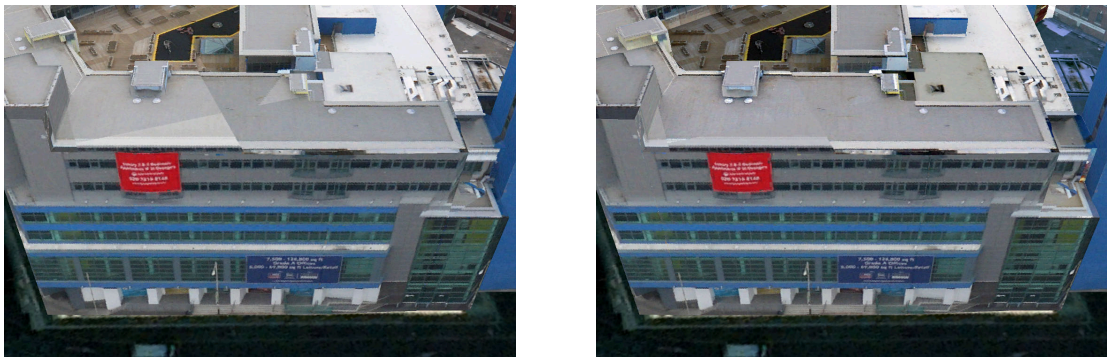
Abb. 3.4: Vergleich der Texturierung mit und ohne Vermeidung von Rohtexturwechseln.

bleibt allerdings unverändert, weshalb die Zusammensetzung immer noch deutlich sichtbar ist. Eine bessere Strategie ist die Vermeidung von Rohtexturwechseln. Damit wird zwar eine suboptimale Texturqualität i. S. d. Qualitätskriteriums (Kap. 3.3.3) akzeptiert, jedoch der Gesamteindruck deutlich konsistenter (Abb. 3.4).

Die Minimierung von Rohtexturwechseln erfordert an sich eine texturweite Optimierung, die nicht mit der lokalen Verarbeitung während der texelweisen Komposition vereinbar ist. Als Alternative wurde eine Heuristik entwickelt, bei der entweder eine komplette Rohtextur direkt als Textur übernommen oder die normale Komposition durchgeführt wird. Grundlage für diese Heuristik ist die Beobachtung, dass Artefakte insbesondere auf großen Flächen mit wenig Verdeckung, z. B. Dachflächen, auffällig und somit störend sind. Ohne Kenntnis des Bildinhalts sollten die eher kleinen, nicht verdeckungsbedingten Fehler in einer einzelnen Rohtextur akzeptiert werden, statt möglicherweise auffälligere Artefakte bei einer Komposition zu erzeugen.

Der Nutzer muss dazu einen sichtbarkeitsbezogenen prozentualen Schwellwert $t_\mathrm{v} \in [0;1]$ und einen auflösungsbezogenen prozentualen Schwellwert $t_\mathrm{r} \in [0;1]$ angeben. Die Sichtbarkeit $V(T)$ einer Rohtextur bezeichnet das Verhältnis von Texeln der Klasse „gültig“ zur Gesamttexelzahl. Die Auflösung $R(T)$ einer Rohtextur wird hier durch den Flächeninhalt der in das jeweilige Schrägluftbild projizierten Fläche gemessen. Nun wird für eine Fläche die Rohtextur $T_\mathrm{b} \in \{T\}$ gesucht, die die höchste Auflösung bei ausreichender Sichtbarkeit ($V(T_\mathrm{b}) > t_\mathrm{v}$) aufweist. Gibt es eine solche Textur und ist ihre Auflösung ausreichend hoch ($R(T_\mathrm{b}) > t_\mathrm{r} \cdot \max(\{R(T) : T \in \{T\}\})$, wird diese Rohtextur ohne weitere Komposition als finale Textur übernommen. Wird keine solche Rohtextur gefunden, wird der normale Kompositionsschritt mit allen Rohtexturen durchgeführt.

Wird die Komposition verwendet, sind Farbunterschiede zwischen verschiedenen Rohtexturen besonders auffällig. In diesem Fall ist ein Histogrammausgleich zwischen den Rohtexturen sinnvoll. Als Referenz wird die Rohtextur mit dem größten Kontrast in die Texeln der Klasse „gültig“ genutzt. Diese auf einzelne Flächen beschränkte, lokale Form des Farbausgleichs ist gut geeignet, in farblich bereits sehr homogenem Bildmaterial geringe Fehler auszugleichen. Weist das Bildmaterial allerdings größere Farbverschiebungen auf, führt der lokale Ansatz zwischen benachbarten Flächen auf Grund unterschiedlich gewählter Referenzbilder zu teils sehr



(a) Ohne Farbausgleich: Einzelne Flächen (z. B. Dach) enthalten deutliche Farbunterschiede.

(b) Mit Farbausgleich: Die Färbung innerhalb der Flächen ist homogener. Farbunterschiede zwischen Flächen können jedoch verstärkt werden.

Abb. 3.5: Vergleich der Texturierung mit und ohne histogrammbasiertem Farbausgleich.

deutlichen Farbunterschieden und somit zu störenderen Artefakten (Abb. 3.5). In solchen Fällen ist eine vorherige globale Farbkorrektur der Schrägluftbilder mit Spezialprogrammen zur Luftbildkorrektur sinnvoll.

3.4 Systemarchitektur

Die im Kap. 3.3 beschriebenen Algorithmen müssen für die effiziente Prozessierung massiver virtueller 3D-Stadtmodelle zu einem Gesamtprozess verknüpft werden. Wesentlich ist dabei die Trennung manueller und automatischer Arbeitsschritte. Zudem muss das System verschiedene Ein- und Ausgabeschnittstellen bereitstellen, um mit heterogenen Datensätzen und verschiedenen Einsatzumgebungen umgehen zu können. Basierend auf der Autodesk LandXplorer-Bibliothek werden für virtuelle 3D-Stadtmodelle folgende Eingabeformate unterstützt: 3DS, OBJ, X3D, PolygonZ-SHP, CityGML und CityDB. Die Schrägluftbilder können in gängigen Bildformaten (z. B. JPEG oder TIFF) vorliegen. Für die Speicherung der inneren Orientierung einer Kamera gibt es kein Standardformat. Die relevanten Daten müssen manuell eingegeben werden. Auch für äußere Orientierungsdaten gibt es kein Standardformat. Die erforderlichen Angaben können jedoch meist als strukturierte Textdatei zur Verfügung gestellt werden. Die Interpretation der Textdateien erfolgt über einen interaktiv konfigurierbaren Importer. Für die Weiterverwendung des vollständig texturierten virtuellen 3D-Stadtmodells ist die Unterstützung verschiedener Ausgabeformate ebenso notwendig. Es werden 3DS, KMZ, CityGML, CityDB und die Autodesk LandXplorer-internen Formate CityStreaming (CS) und TextureAtlasTree (TTT) unterstützt.

Das System besteht aus einer Serie von Arbeitsschritten (Abb. 3.6 und Tab. 3.5). Die Architektur orientiert sich an der Annahme, das zu texturierende virtuelle 3D-Stadtmodell bestehe aus ein oder mehreren Kacheln und werde von möglicherweise aus mehreren Quellen stammendem Schrägluftbildmaterial überdeckt. Jede Kachel soll individuell prozessiert werden können. Dementsprechend sind die Arbeitsschritte in kachelbezogene und übergreifende Schritte gegliedert. Übergreifende Schritte müssen vor der Kachelprozessierung durchgeführt werden. Kachelbezogene Schritte müssen für jede Kachel durchgeführt werden und sind im Allgemeinen unabhängig von anderen Kacheln; sie können somit getrennt und parallel ausgeführt werden. Einen Spezialfall bildet die manuelle Schrägluftbildausrichtung. Da ein Bild mehrere Kacheln

Arbeitsschritt	*Beschreibung*
	kachelübergreifend
Verdeckerberechnung	Zusammenfassung des gesamten Modells zu einem Objekt. Wird zur korrekten Verdeckungsberechnung benötigt, die auf benachbarte Kacheln zugreifen muss.
Import Kamerakalibrierung	Manuelle Eingabe der Kalibrierungsdaten aller verwendeten Kameras.
Import Externe Orientierungen	Import der externen Orientierungsdaten aller Schrägluftbilder.
	kachelbezogen
Import 3D-Geometrie	Import der 3D-Geometrie einer Kachel. Umfasst auch die Texturzuordnung und -parametrisierung (Kap. 3.3.1).
Schrägluftbildausrichtung	Manuelle Anpassung der externen Orientierungsdaten kachelrelevanter Schrägluftbilder. Die Anpassung wird kachelübergreifend gespeichert, sodass jedes Schrägluftbild nur einmal angepasst werden muss, auch wenn es mehrere Kacheln überdeckt.
Rohtextur- und Qualitätsrastererzeugung	Automatische Erzeugung aller Rohtexturen aller Flächen der Kachel (Kap. 3.3.2 und 3.3.3). Zur Effizienzsteigerung wird jedes relevante Schrägluftbild nur einmal geladen und daraus Rohtexturen und Qualitätsraster für alle darin sichtbaren Flächen erzeugt.
Farbkorrektur	Optionaler automatischer Schritt zum Farbausgleich (Kap. 3.3.6). Berechnet die Ausgleichsparameter für alle Rohtexturen. Der eigentliche Farbausgleich erfolgt bei der Komposition.
Texturkomposition	Automatische Komposition aller Texturen der Kachel (Kap. 3.3.4).
Inspektion	Manuelle Inspektion des texturierten virtuellen 3D-Stadtmodells. Fehlerhafte Schrägbildausrichtungen können direkt korrigiert werden und führen zu erneuter automatischer Prozessierung.
Texturnachbearbeitung	Optionaler manueller Schritt. Alle erzeugten Texturen lassen sich in externen Programmen nachbearbeiten und wieder in das virtuelle 3D-Stadtmodell integrieren.
Export texturierte 3D-Geometrie	Export des Prozessierungsergebnisses in verschiedene Ausgabeformate.

Tab. 3.5: Beschreibung der Arbeitsschritte zur automatischen Texturierung (Abb. 3.6).

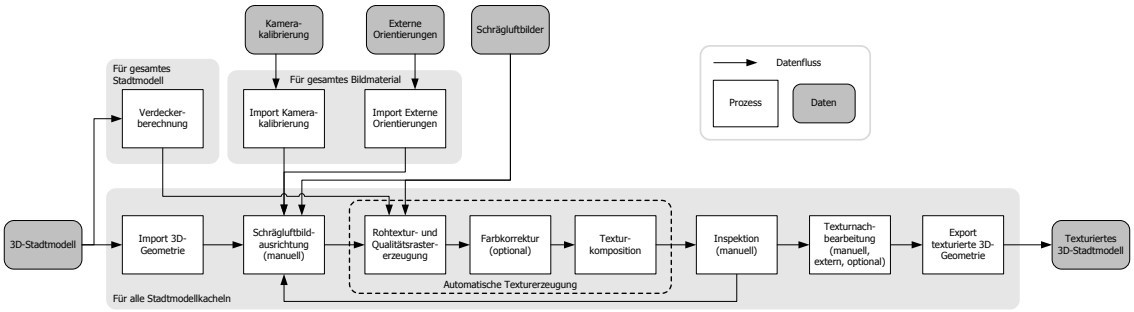


Abb. 3.6: Systemarchitektur zur automatischen Texturierung. Das System bettet die automatischen Arbeitsschritte in die erforderliche manuelle Vor- und Nachbearbeitung ein. Tab. 3.5 beschreibt alle Arbeitsschritte.

überdecken kann, aber nur eine „korrekte“ Lage besitzt, braucht es nur ein einziges Mal ausgerichtet werden. Die Korrektur wird dann kachelübergreifend gespeichert.

Jeder in Abb. 3.6 gezeigter Schritt ist als eigenständiges Programm implementiert. Damit wird eine deutlich höhere Stabilität insbesondere bei den zeitaufwändigen automatischen Prozessierungsschritten erreicht, da mögliche Speicherlecks oder Fehler in einem Prozess keine Auswirkungen auf andere Prozesse haben. Ebenso wird die Fragmentierung des Hauptspeichers begrenzt. Der Datenaustausch zwischen den Schritten erfolgt dateibasiert. Die Koordination erfolgt über eine separate Benutzerschnittstelle (Abb. 3.7), die Zugriff auf alle Parameter gibt und die Ausführung einzelner Schritte bzw. einer ganzen Gruppe von Schritten erlaubt.

Ein wesentlicher Aspekt für den praktischen Einsatz des Systems ist die Möglichkeit zur Fehlerkontrolle und manuellen Nachbearbeitung der erzeugten Texturen. Fehlerquellen sind zum einen mangelnde Ausrichtungsqualität der Schrägluftbilder und zum anderen Artefakte in den erzeugten Texturen. Eine automatische Bewertung der Qualität ist nicht Teil des Systems. Stattdessen muss eine visuelle Kontrolle erfolgen. Diese Kontrolle des texturierten virtuellen 3D-Stadtmodells ist in das Ausrichtungswerkzeug integriert, um fehlerhafte Bildausrichtungen sofort korrigieren zu können. Dabei kann zu jeder Textur das verwendete Schrägluftbild texelpräzise bestimmt werden. Zusätzlich besteht die Möglichkeit, sowohl Texturen als auch Rohtexturen für einzelne Flächen zu exportieren. Das Bildmaterial lässt sich dann mit einem beliebigen Bildbearbeitungsprogramm zu einer verbesserten Textur zusammenfügen und wieder in das System integrieren.

3.5 Anwendungsbeispiele

Das beschriebene Verfahren wurde in mehreren Projekten in Zusammenarbeit mit verschiedenen Firmen eingesetzt. Im Folgenden werden zwei Beispiele vorgestellt, Tab. 3.6 gibt einen Überblick über die jeweiligen technischen Informationen.

Das erste Beispiel ist ein virtuelles 3D-Stadtmodell der Innenstadt von München, bestehend aus ca. 21.000 Gebäuden auf einer Fläche von ca. 30km^2 und unterteilt in neun Kacheln. Die Eingabedaten wurden von der Firma COWI A/S zur Verfügung gestellt. Das Modell wurde fotogrammetrisch aus Nadiraufnahmen abgeleitet und wies eine sehr hohe Modellierungsqualität und sehr wenige Geometriefehler auf. Der Großteil der Gebäude lag im LoD 2 (Kap. 2.1) vor; ca. 350 Gebäude im Zentrum waren in LoD 3 modelliert. Die Schrägluftbilder wurden mit dem

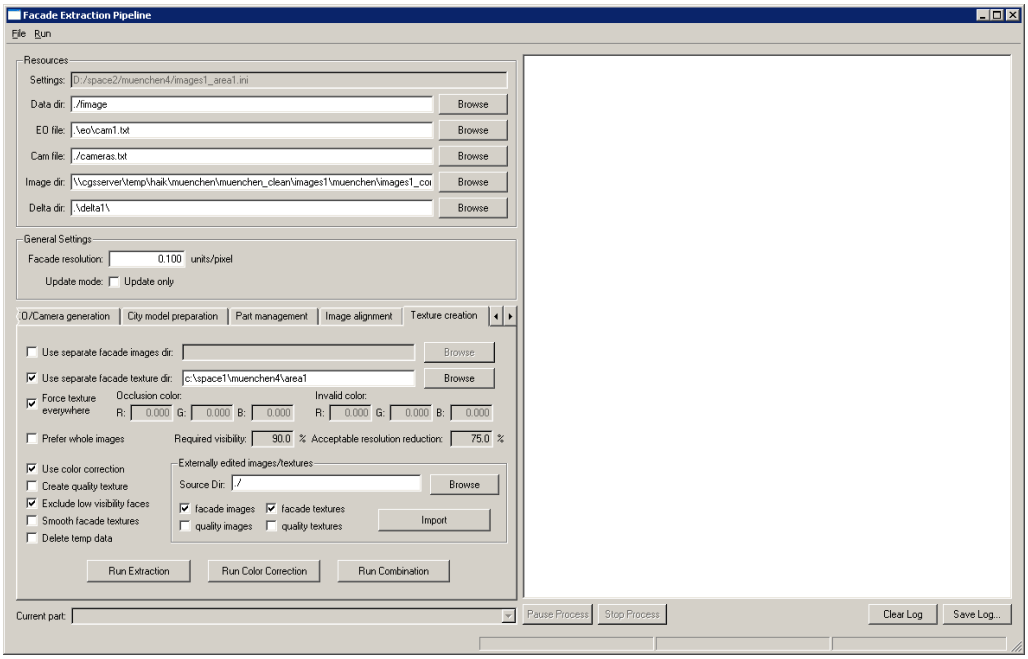


Abb. 3.7: Benutzeroberfläche zur Steuerung der automatischen Texturierung.

Track'Air MIDAS System aufgenommen. Für das Projektgebiet wurden ca. 3.000 Bilder verwendet. Die manuelle Ausrichtung der Schrägluftbilder benötigte ca. 35 Arbeitsstunden, d. h. ca. 41s pro Bild. Die automatische Prozessierung benötigte eine Rechenzeit von ca. 22h auf einem einzelnen PC mit AMD Athlon64 X2 Prozessor (2,3 GHz), 2GB RAM und einer NVidia GeForce 8800GTS Grafikkarte mit 640MB Grafikspeicher. Das texturierte virtuelle 3D-Stadtmodell umfasst 5,5GB im KMZ-Format bei einer metrischen Texturauflösung von 10x10cm^2 pro Texel, was einer unkomprimierten Texturmenge von ca. 16GB entspricht. Das Ergebnis (Abb. 3.8) wurde in GoogleEarth integriert und ist dort nach Aktivierung des 3D-Gebäudelayers erlebbar.

Das zweite Beispiel ist die Texturierung des offiziellen Berliner 3D-Stadtmodells. Die Modellgeometrie wurde in zwei Abschnitten, Ost und West, auf Basis der automatischen Liegenschaftskarte (ALK) aus Lidar-Daten automatisch abgeleitet. Es besteht aus zusammen ca. 475.000 Gebäuden auf einer Gesamtfläche von ca. 800km^2, die jedoch nicht gleichmäßig bebaut ist. Für die Erstellung war das Modell in regelmäßige Kacheln von 1×1km^2 unterteilt. Auf Grund der automatischen Ableitung enthielt dieses Modell vergleichsweise häufig Geometriefehler, die während der manuellen Schrägluftbildausrichtung protokolliert und anschließend extern korrigiert wurden. Als Bildmaterial standen Pictometryaufnahmen der Jahre 2006 bis 2008 mit stark variierender Farbqualität zur Verfügung, da auf Grund der großen Flughöhe von über 1.500m teilweise starke atmosphärische Streuung auftrat. Für die Texturierung wurden ca. 25.000 Bilder verwendet, die in ca. 225 Arbeitsstunden ausgerichtet wurden. Auf Grund von Verbesserungen im Ausrichtungswerkzeug reduziere sich die durchschnittliche Ausrichtungszeit auf ca. 32s pro Bild. Die automatische Prozessierung benötigte ca. 600 Rechenstunden und wurde auf Kachelbasis auf 8 Rechner aufgeteilt. Die Rechner waren ähnlich dem Rechner für das Münchenprojekt konfiguriert. Die unkomprimierte Texturmenge beträgt ca. 150GB, für Out-of-Core-Rendering aufbereitet beträgt die komprimierte Texturmenge ca. 33GB. Die erzeugten Texturen wurden

Abb. 3.8: Ein vollständig texturiertes virtuelles 3D-Stadtmodell der Münchener Innenstadt.

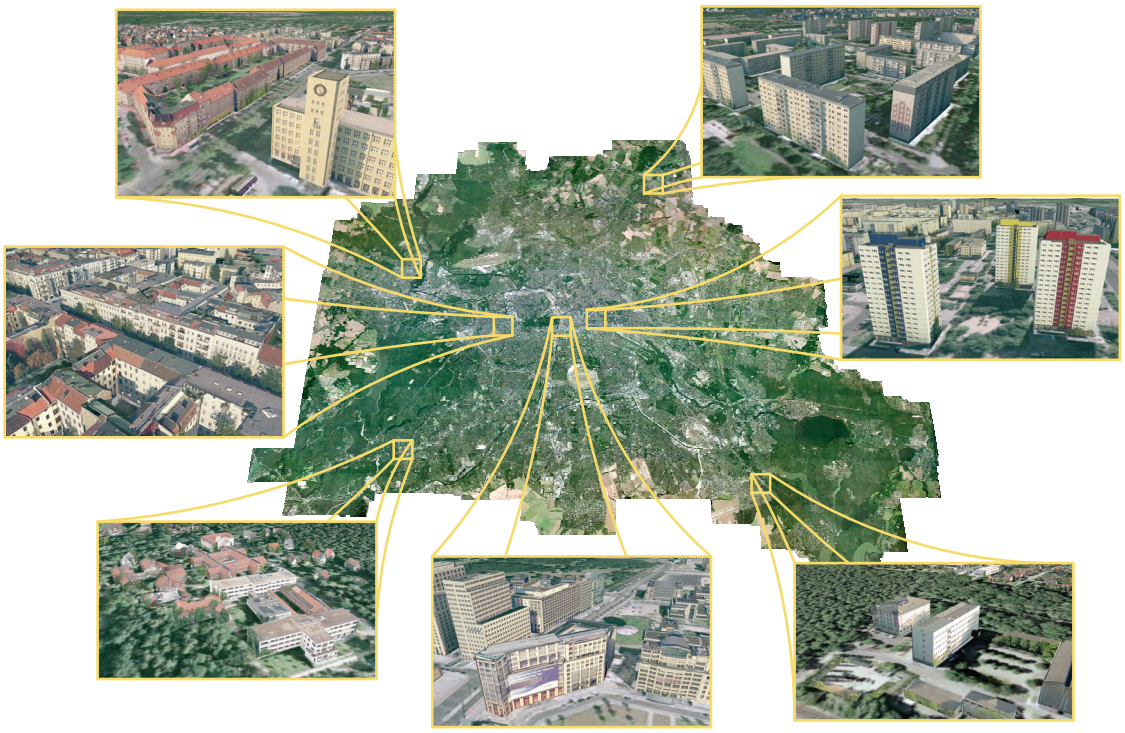


Abb. 3.9: Ein vollständig texturiertes virtuelles 3D-Stadtmodell von Berlin.

Projekt	*Dateneigenschaften*	*Dimensionen*	*Bearbeitungszeit*	*Verwendung*
München (Abb. 3.8)	Fotogrammetrisch abgeleitetes virtuelles 3D-Stadtmodell; MIDAS Schrägluftbilder	ca. 21.000 Gebäude; ca. 30km^2; ca. 3.000 Bilder; ca. 16GB Texturen	ca. 35h Verschiebung; ca. 22h Berechnung	Das Ergebnis wurde in GoogleEarth integriert und ist dort als volltexturiertes virtuelles 3D-Stadtmodell erlebbar.
Berlin (Kada, 2009) (Abb. 3.9)	Aus LiDAR und Grundrissen abgeleitetes virtuelles 3D-Stadtmodell; Pictometry Schrägluftbilder	ca. 470.000 Gebäude; ca. 775km^2; ca. 25.000 Bilder; ca. 150GB Texturen	ca. 225h Verschiebung; ca. 600h Berechnung	Das Ergebnis ist Teil des von der Berlin Partner GmbH verwalteten offiziellen Berliner 3D-Stadtmodells.

Tab. 3.6: Beispiele für aus Schrägluftbildern texturierte virtuelle 3D-Stadtmodelle.

in eine CityGML-basierte Datenbank (Stadler u. a., 2008) eingespielt; das gesamte virtuelle 3D-Stadtmodell (Abb. 3.9) wird von der Berlin Partner GmbH verwaltet.

Das beschriebene Verfahren fand Eingang als Kern des Autodesk-Produkts CityFactory. Dieses Produkt wird derzeit bei drei Firmen aktiv eingesetzt: Infoterra, Ltd., COWI A/S und virtualcity-SYSTEMS GmbH.

3.6 Diskussion

Der vorgestellte Ansatz ermöglicht bei ausreichender Genauigkeit der Eingabedaten eine vollautomatische Texturierung virtueller 3D-Stadtmodelle aus georeferenzierten Schrägluftbildern. Das virtuelle 3D-Stadtmodell wird dabei als fix angesehen. Evtl. notwendige Korrekturen müssen durch externe Werkzeuge durchgeführt werden. Ungenauigkeiten in der Geometrie beeinflussen das Ergebnis auf zwei Arten negativ: Zum einen ist die Textur des fehlerhaften Gebäudes selbst gestört, was sich in typischen Texturfehlern äußert (Tab. 3.7). Zum anderen wird die Verdeckungsberechnung verfälscht, sodass auch Texturen in der Umgebung des Gebäudes verfälscht werden können.

Der zentrale Mangel aktueller Erfassungssysteme ist jedoch die Genauigkeit der Georeferenzierung der Schrägluftbilder. Für eine hochwertige Texturierung ist eine Abweichung von unter 50cm zwischen Geometrie und korrespondierendem Bildinhalt notwendig (Kap. 3.3.5); andernfalls zeigen sich typische Texturfehler (Tab. 3.8). Die Georeferenzierung der Schrägluftbilder muss somit kontrolliert und ggf. korrigiert werden. Die Automatisierung dieser Korrektur ist ein offenes Problem. Es gibt sehr gute Verfahren, z. B. basierend auf SIFT (Lowe, 1999), die eine automatische Ausrichtung eines Bildverbandes erlauben. Die korrekte Ausrichtung der Schrägluftbilder untereinander ist jedoch im vorliegenden Fall zweitrangig. Wesentlich ist die Ausrichtung der Bilder zum virtuellen 3D-Stadtmodell. Das Problem wird durch die vorhandene

Fehler	*Fehlerbild*
Gebäudehöhe	Obere Stockwerke und Dach falsch texturiert.
Dachform	Dach falsch texturiert.
Fassadenausrichtung	Stockwerkslinien nicht horizontal.
Approximation gekrümmter Oberflächen durch Ebenen	Bogenförmige Verzerrungen.
Fehlende Dachüberstände (typisch für extrudierte Grundrisse)	Dachkante auf Fassaden projiziert.
Vergrößerter Grundriss (typisch für extrudierte Dachumringe)	Versatz der Fassadentexturen, Ränder mit ungültigem Inhalt gefüllt.
Fehlende strukturelle Details (z. B. Dachaufbauten, Fassadendetails)	Projektion der Elemente auf die jeweilige Fläche, evtl. aus verschiedenen Perspektiven.

Tab. 3.7: Typische geometrische Modellungenauigkeiten. Die Ungenauigkeiten zeigen sich im texturierten virtuellen 3D-Stadtmodell durch die beschriebenen sichtbaren Fehler.

Georeferenzierung der Schrägluftbilder als sehr gute initiale Schätzung vereinfacht. Die stark schwankende Bildqualität, die vergleichsweise geringe metrische Bildauflösung, die große Zahl der Störeinflüsse (Tab. 3.2) und die variable Zuverlässigkeit der Gebäudemodelle erschweren das Problem jedoch. Im Hinblick auf den Einsatz für reale Datensätze rechtfertigt die zu erwartende Verbesserung der Ausrichtungsgenauigkeit den benötigten Rechenaufwand einer automatischen Ausrichtung nicht. Derzeit ist deshalb eine manuelle Ausrichtung vor und manuelle Inspektion nach der Texturierung notwendig, für die ein separates Werkzeug entwickelt wurde. Ein Automatismus für die Ausrichtung der Schrägluftbilder ist nichtsdestotrotz ein wichtiges Ziel. Die damit implizit mögliche automatische Schätzung der Ausrichtungsgenauigkeit (z. B. durch Differenz zwischen originaler und nachbearbeiteter Ausrichtung) würde zum einen direkt Informationen zur Korrektheit des virtuellen 3D-Stadtmodells liefern, zum anderen ließe sich damit die manuelle Inspektion des texturierten virtuellen 3D-Stadtmodells gezielt leiten und unterstützen.

Aus Schrägluftbildern texturierte virtuelle 3D-Stadtmodelle sind nur eingeschränkt für Darstellungen aus der Fußgängerperspektive geeignet. Zum einen ist dies in der beschränkten metrischen Texturauflösung begründet. Fußgängerperspektiven benötigen Auflösungen von deutlich unter 5cm pro Texel (Göbel und Freiwald, 2008), Schrägluftbilder erreichen jedoch derzeit bestenfalls 15cm. Zum anderen lassen sich insbesondere Ladenzeilen bzw. das unterste Stockwerk prinzipiell schlecht erfassen, da sie häufig durch andere Gebäude oder Vegetation verdeckt werden. Durch Anpassung des Neigungswinkels der Schrägluftbilder kann die Sichtbarkeit zu Lasten der metrischen Auflösung verbessert werden. Eine attraktive Lösung ist die Kombination mit bodengestützter Erfassung. Die Machbarkeit einer flächendeckenden Erfassung wird durch verschiedene Anbieter demonstriert. Die erzeugten Bilddaten sind wie Schrägluftbilder direkt georeferenziert. Liegt eine ausreichende Genauigkeit der äußeren Orientierungsdaten vor, lässt sich dieses Bildmaterial direkt in den hier beschriebenen Ablauf integrieren und insbesondere der bodennahe Bereich deutlich aufbessern. Bei der Verwendung solcher Bilddaten stellen

Fehler	*Fehlerbild*
Ein einzelnes stark fehlerhaft ausgerichtetes Bild	Verschobene Texturen an der gleichen Gebäudeseite in einer ganzen Gruppe von Gebäuden.
Inkonsistent ausgerichtete Bilder	Versätze an Rohtexturwechseln innerhalb einer Textur in einzelnen Bereichen des virtuellen 3D-Stadtmodells.
Unzureichende Kamerakalibrierungsgenauigkeit	Versätze an Rohtexturwechseln innerhalb einer Textur im gesamten virtuellen 3D-Stadtmodell. Gute Ausrichtung der Schrägluftbilder schwierig.

Tab. 3.8: Typische Ungenauigkeiten der Schrägluftbildgeoreferenzierung. Die Ungenauigkeiten zeigen sich im texturierten virtuellen 3D-Stadtmodell durch die beschriebenen sichtbaren Fehler.

sich jedoch neue Probleme, wie das der Privatsphäre oder der deutlichen Verdeckungen durch Passanten, Fahrzeuge oder Vegetation, die behandelt werden müssen.

Das Ergebnis der beschriebenen Texturierung ist ein explorierbares „3D-Foto“ einer Stadt. Eine realitätsnahe Wiedergabe ist allerdings nicht immer die effektivste Darstellungsmöglichkeit, wie die deutliche Trennung der Anwendungsgebiete von Luftbildern und topographischen Karten zeigt. Analog zur Ableitung topographischer Karten aus Luftbildern können auch im 3D-Fall die aus Schrägluftbildern erzeugten Texturen Vorlagen für die Ableitung abstrahierter Texturen und Gebäudeeigenschaften wie Stockwerksanzahl oder Grundfarbe (Karbo und Schroth, 2009; Müller u. a., 2007) sein. Die Detailreduktion ist insbesondere im Bereich der mobilen Navigation interessant, bei der auf Grund von Hardwareeinschränkungen zwischen Realismus und Speicherbedarf abgewogen werden muss.

4 Modellierung oberflächenbezogener Daten in CityGML

Virtuelle 3D-Stadtmodelle werden nicht allein für Präsentationszwecke, sondern auch als Mittel zur raumbezogenen Informationsvisualisierung eingesetzt – sie agieren in Verbindung mit entsprechender Analyse- und Simulationsfunktionalität als „computational tools“. Neben der visuellen Präsentation, wie zum Beispiel durch Google Earth, entwickeln sich Anwendungen virtueller 3D-Stadtmodelle in den Gebieten der Planung, Simulation, Monitoring, Marketing und Sicherheit, etc. Auf Präsentation beschränkte virtuelle 3D-Stadtmodelle, die ausschließlich entsprechende Daten wie z. B. Geometrie- und Farbdaten enthalten, sind hier nicht mehr ausreichend. Vielmehr müssen virtuelle 3D-Stadtmodelle Möglichkeiten zur Speicherung fachspezifischer, semantischer und topologischer Daten beinhalten, um so die für die jeweiligen Anwendungen notwendigen Daten direkt in die Objekte des virtuellen 3D-Stadtmodells einzubetten.

Eine Kategorie fachspezifischer Daten sind *oberflächenbezogene Daten*, die als *„durch Sensoren erfassbare Eigenschaften einer Oberfläche“* (Gröger u. a., 2008) definiert sind. Das schließt neben Fotografien auch Infrarotaufnahmen, physikalische Parameter (z. B. Wärmedurchgangskoeffizient bzw. k-Wert) oder Simulationsergebnisse (z. B. Windgeschwindigkeiten) ein. Grundsätzlich wird dabei einem Flächenstück ein (Mess-)Wert zugeordnet. Die Größe des Flächenstücks ist abhängig von der Erfassung der Messwerte und kann von einem ganzen Oberflächenstück (z. B. konstanter k-Wert für eine ganze Wand) über wenige Zentimeter große Kacheln (z. B. ein Pixel in einer Farbfotografie der Wand) bis hin zu infinitesimal kleinen Bereichen (z. B. bei pro Punkt auf der Wand berechenbaren Beleuchtungsdaten) reichen. Oberflächenbezogene Daten bestehen somit aus drei Bestandteilen: den Daten selbst, der Oberfläche sowie der Abbildungsvorschrift von der Oberfläche auf die Daten.

Im Folgenden wird die Modellierung oberflächenbezogener Daten in CityGML (Gröger u. a., 2008), einem XML-basierten Standard für Austausch und Speicherung virtueller 3D-Stadtmodelle, beschrieben. Für die Modellierung oberflächenbezogener Daten wird dabei auf Konzepte der Parametrisierung von Objektoberflächen, wie sie insbesondere für die Texturierung in der 3D-Computergrafik entwickelt wurden, zurückgegriffen. Es werden die konkreten Ziele beschrieben, unterschiedliche Modellierungsansätze bewertet, das Datenmodell erläutert und das Ergebnis diskutiert. Dabei wird insbesondere auf Entwicklungsentscheidungen eingegangen, um das Datenmodell nachvollziehbar und transparent zu machen.

4.1 Zielstellungen

Oberflächenbezogene Daten werden in CityGML als *Appearances* bezeichnet. Der Name leitet sich aus der ursprünglichen Intention ab: In älteren Entwürfen von CityGML wurden lediglich Texturierung und Färbung polygonaler Oberflächen im computergrafischen Sinne unterstützt. Dies diente vor allem zur visuellen Ausgestaltung der virtuellen 3D-Stadtmodelle und zur verlustfreien Konvertierung auf Präsentation beschränkter virtueller 3D-Stadtmodelle nach CityGML. Das neu definierte Modul „Appearance“ soll diesen eingeschränkten Mechanismus um die Speicherung allgemeiner oberflächengebundener Daten erweitern und für fachspezifische Anwendungen öffnen. Dabei stehen Appearances zwischen dem geometrischen und thematischen Modell von CityGML. Sie ergänzen thematische Objekte um Informationen, die jedoch fest mit deren geometrischen Eigenschaften verbunden sind.

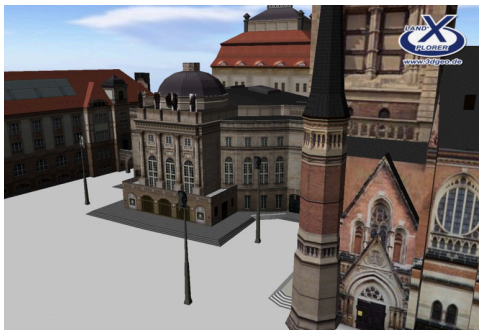
Für die Modellierung steht die einfache Verwendbarkeit für gängige Gebrauchsmuster im Vordergrund, nicht die möglichst generische Abdeckung aller Aspekte solcher Daten. Die Einstiegshürde für die Verwendung von CityGML soll nicht unnötig erhöht werden, wenngleich auch komplexe Anwendungen nicht ausgeschlossen werden dürfen. Dazu wurden sechs (kombinierbare) Anwendungsfälle identifiziert (Abb. 4.1):

1. *Appearance-freie virtuelle 3D-Stadtmodelle (Abb. 4.1(a)):* Virtuelle 3D-Stadtmodelle, die keine oberflächenbezogenen Daten benötigen, sollen ohne Zusatzaufwand beschreibbar sein.
2. *Auf Präsentation beschränkte virtuelle 3D-Stadtmodelle (Abb. 4.1(b)):* Gängige texturierte und gefärbte virtuelle 3D-Stadtmodelle sollen verlustfrei nach CityGML konvertierbar sein.
3. *Georeferenzierte Luftbilder (Abb. 4.1(c)):* Georeferenzierte Luftbilder sollen als Quelle für oberflächenbezogene Daten verwendbar sein.
4. *Transport von Simulationsdaten bzw. -ergebnissen (Abb. 4.1(d)):* Die speicherbaren Daten sollen nicht auf Bilddaten beschränkt sein.
5. *LoD-abhängige Daten (Abb. 4.1(e)):* Jede Detailstufe eines Objekts soll unabhängig mit oberflächenbezogenen Daten überziehbar sein.
6. *Mehrere Datenschichten pro Objekt (Abb. 4.1(f)):* Für ein Objekt sollen gleichzeitig oberflächenbezogene Daten unterschiedlicher und beliebiger Kategorien speicherbar sein.

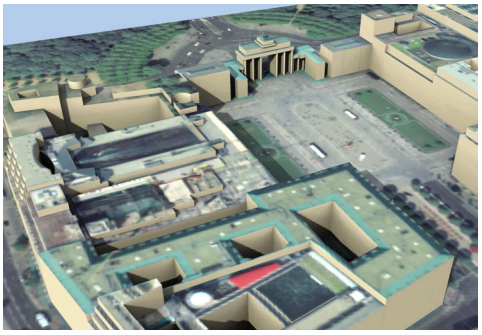
CityGML beschreibt ausschließlich stadtmodellbezogene Daten. Demzufolge sollen Details der computergrafischen Präsentation, z. B. die Definition von Lichtquellen oder Kameras, nicht abgedeckt werden. Die Präsentation obliegt entweder der Anwendung oder wird in einer zukünftigen separaten Beschreibung (Neubauer und Zipf, 2009) ähnlich zu OGC Styled Layer Descriptors (SLD; Lalonde, 2002) bzw. Symbology Encoding (SE; Müller, 2005) definiert. Es wird auch nicht auf die Kombination mehrerer Appearances eingegangen. Sind mehrere Appearances für ein Objekt definiert, entscheidet die Anwendung bzw. der Nutzer über die Auswahl und die Art der Darstellung. Generische Ansätze aus der Computergrafik, Darstellungsalgorithmen in Form von Code zu transportieren (z. B. Shaders in Collada: Arnaud und Barnes, 2006), liegen außerhalb des Aufgabenbereichs von CityGML.

Auf der technischen Seite sollen Appearances ohne Erweiterung der GML-Geometrieklassen auskommen, um bestehende GML-Implementierungen für CityGML weiterverwenden zu

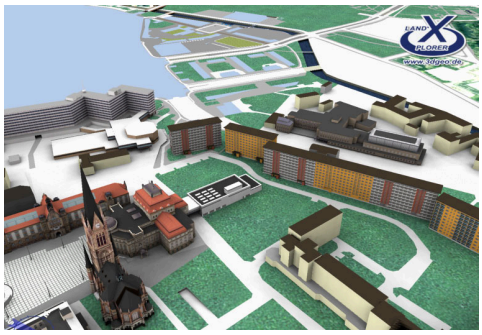
(a) Appearance-freie virtuelle 3D-Stadtmodelle.



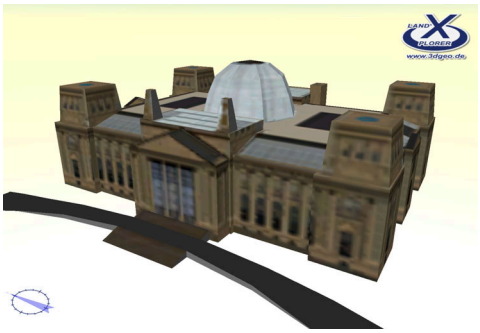
(b) Auf Präsentation beschränkte virtuelle 3D-Stadtmodelle.

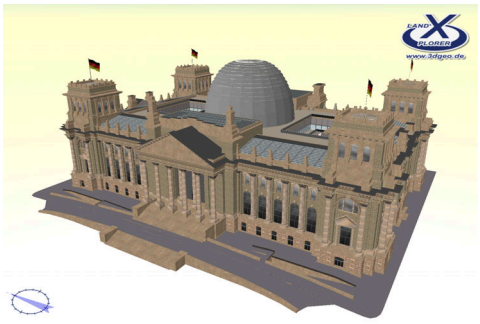


(c) Georeferenzierte Luftbilder.



(d) Transport von Simulationsdaten bzw. -ergebnissen, hier aus einer Beleuchtungssimulation.





(e) LoD-abhängige Daten. Links: LoD2, rechts: LoD3.

(f) Mehrere Datenschichten pro Objekt. Links: RGB, rechts: Infrarot.

Abb. 4.1: Anwendungsfälle für oberflächenbezogene Daten in CityGML.

können. Zudem sollen alle Informationen zu einer Appearance gemeinsam gespeichert werden, um sie zum Beispiel in einer separaten Datei speichern oder über einen Webservice (z. B. Web Feature Service (WFS); Vretanos, 2005) abrufen zu können. Schließlich wird auf die interne Speicherung der oberflächenbezogenen Daten selbst, d. h. die Einbettung der Texturdaten in das CityGML-Dokument, verzichtet. Die entstehende Datenmenge ist nicht sinnvoll in einem XML-Format handhabbar, weshalb zur Speicherung auf existierende Bilddatenformate zurückgegriffen wird. Bilddateien müssen nicht ausschließlich lokal vorliegen, sondern sollen auch als Ressource im Web oder als Webserviceabruf beschreibbar sein.

4.2 Modellierungsansätze

Für eine verständliche Modellierung sollen CityGML Appearances so weit wie möglich auf gängigen Ansätzen aufbauen. Dazu wurden drei Ansätze auf ihre Verwendbarkeit hin untersucht.

Erweiterung von OGC SLD/SE

Bei diesem Modellierungsansatz werden oberflächenbezogene Daten als Teil der Präsentation verstanden und dementsprechend nicht direkt in CityGML gespeichert. Stattdessen sollen die existierenden OGC-Standards Styled Layer Descriptors (Lalonde, 2002) oder Symbology Encoding (Müller, 2005) entsprechend erweitert werden.

Beide Standards dienen der Präsentation von 2D-Daten. Dazu bieten sie die Möglichkeit, Features anhand von definierbaren Kriterien Signaturierungen zuzuweisen. Diese Signaturierungen können die originale Feature-Geometrie vollständig ersetzen oder ihr einen Präsentationsstil zuweisen. Eine Erweiterung auf die Präsentation von 3D-Daten inkl. des Transports oberflächenbezogener Daten erfordert tiefgreifende Änderungen. Anders als Signaturierungen müssen oberflächenbezogene Daten einzelnen Flächen, nicht ganzen Features zugeordnet werden. Zudem kann die Positionierung nicht wie für Signaturierungen algorithmisch bestimmt werden, sondern muss frei definierbar sein. Neubauer und Zipf (2007) stellen einen Ansatz zur Erweiterung von SLD für thematische 3D-Visualisierung vor, schließen die Verwendung von Rasterdaten als oberflächenbezogene Daten jedoch explizit aus. Dieser Vorschlag wurde kürzlich als Diskussionsvorschlag beim OGC eingereicht (Neubauer und Zipf, 2009).

Änderungen an SLD oder SE würden einen langwierigen Standardisierungsprozess erfordern, weshalb sie für die Nutzung in CityGML nicht zur Verfügung stehen. Zudem sind oberflächenbezogene Daten nicht Teil der Präsentation, sondern integraler Bestandteil eines virtuellen 3D-Stadtmodells und müssen folglich Teil von CityGML werden.

Nutzung von GML Coverages

Die Norm ISO 19123:2005 definiert eine Coverage generisch als beliebige Abbildung eines räumlichen Definitionsbereiches auf Attributwerte und beinhaltet damit die Definition oberflächenbezogener Daten. Deren drei Bestandteile (Bezugsoberfläche, Daten und Abbildungsvorschrift) finden sich direkt in der Definition der Coverages. Es wird zwischen diskreten Coverages, die einen für ein ganzes Objekt konstanten Attributwert definieren, und kontinuierlichen Coverages, die mit der Raumposition variierende Attributwerte definieren, unterschieden.

GML 3.2 (Portele, 2007) stellt eine Implementierung dieses Konzepts zur Verfügung, welche den generischen Charakter der Coverages mit einem abstrakten Datenmodell umsetzt. Diskrete Coverages werden mit der trivialen Abbildung von Geometrieobjekt auf Wert unterstützt.

Kontinuierliche Coverages, welche für oberflächenbezogene Daten wichtig sind, werden nicht konkret definiert. Für eine Verwendung ist die Spezialisierung aller drei Aspekte, des Definitionsbereiches, des Wertebereiches und der Abbildungsvorschrift, notwendig. Somit bringt die Verwendung von Coverages nur wenig Reduzierung des Modellierungsaufwandes und kommt einer kompletten Neumodellierung nahe.

Materialien und Texturen

Materialien und Texturen sind Konzepte aus der 3D-Computergrafik zur Beschreibung von Oberflächeneigenschaften und -details. Sie sind nicht auf Farbinformationen limitiert, sondern können beliebige Größen bzw. Attribute auf eine Oberfläche abbilden. In der 3D-Computergrafik werden mit den Werten pro Oberflächenpunkt Darstellungsvorschriften wie z. B. Beleuchtungsmodelle ausgewertet. Dabei definieren Materialien für eine Oberfläche konstante Werte und Texturen variierende Werte.

Diese Konzepte sind wohlverstanden und zu einer kanonischen Form gereift, was die Gemeinsamkeiten unterschiedlicher Implementierungen, z. B. OpenGL (Segal und Akeley, 2009) oder Direct3D (Corp., 2006), und verschiedener Austauschformate, z. B. X3D (ISO/IEC FDIS 19775-1:2008) oder Collada (Barnes und Finch, 2001), unterstreichen. Eine direkte Verwendung existierender Datenmodelle wird jedoch erschwert, da eine strikte Trennung zwischen geometrischen Elementen der Oberfläche und oberflächenbezogenen Daten in CityGML erforderlich ist. Üblicherweise wird zumindest die Abbildungsvorschrift als Teil der geometrischen Elemente modelliert.

Gewählter Modellierungsansatz

Obwohl die Verwendung computergrafischer Konzepte im Kontext der raumbezogenen Modellierung durchaus kontrovers diskutiert wird, wurde für CityGML das Material- und Texturkonzept zur Modellierung oberflächenbezogener Daten gewählt. Diese Wahl ist in der Verständlichkeit und Einfachheit des Ansatzes begründet, der sowohl die Verwendung von CityGML als auch dessen Anbindung an bestehende Software erleichtert. Dabei schließt CityGML explizit die Definition einer Präsentation aus und verwendet diese Konzepte einzig als technisches Mittel zur Datenmodellierung. Komplexe Aspekte wie Multitexturing, d. h. das Verrechnen mehrerer Texturschichten, werden daher nicht behandelt; sie liegen im Aufgabenbereich der 3D-Präsentationssoftware.

4.3 Datenmodell des CityGML Appearance-Moduls

Das Datenmodell für Appearances ist auf die geforderten Anwendungsfälle reduziert. Es baut auf Material- und Texturdefinitionen von X3D, einem XML-basierten Computergrafik-Austauschformat, auf und bettet diese in das Appearance-Konzept von CityGML ein. An dieser Stelle soll ein Überblick über das Datenmodell gegeben werden. Für Details sei auf die Spezifikation (Gröger u. a., 2008) verwiesen.

Abb. 4.2 zeigt das UML-Klassendiagramm des Appearance-Moduls. Das Datenmodell lässt sich in drei Bereiche gliedern:

1. Anbindung an Features (Klasse `Appearance`)
2. Datendefinition (Klasse `_SurfaceData` und Ableitungen)
3. Anbindung an Geometrie (`target`-Elemente bzw. -Assoziationen)

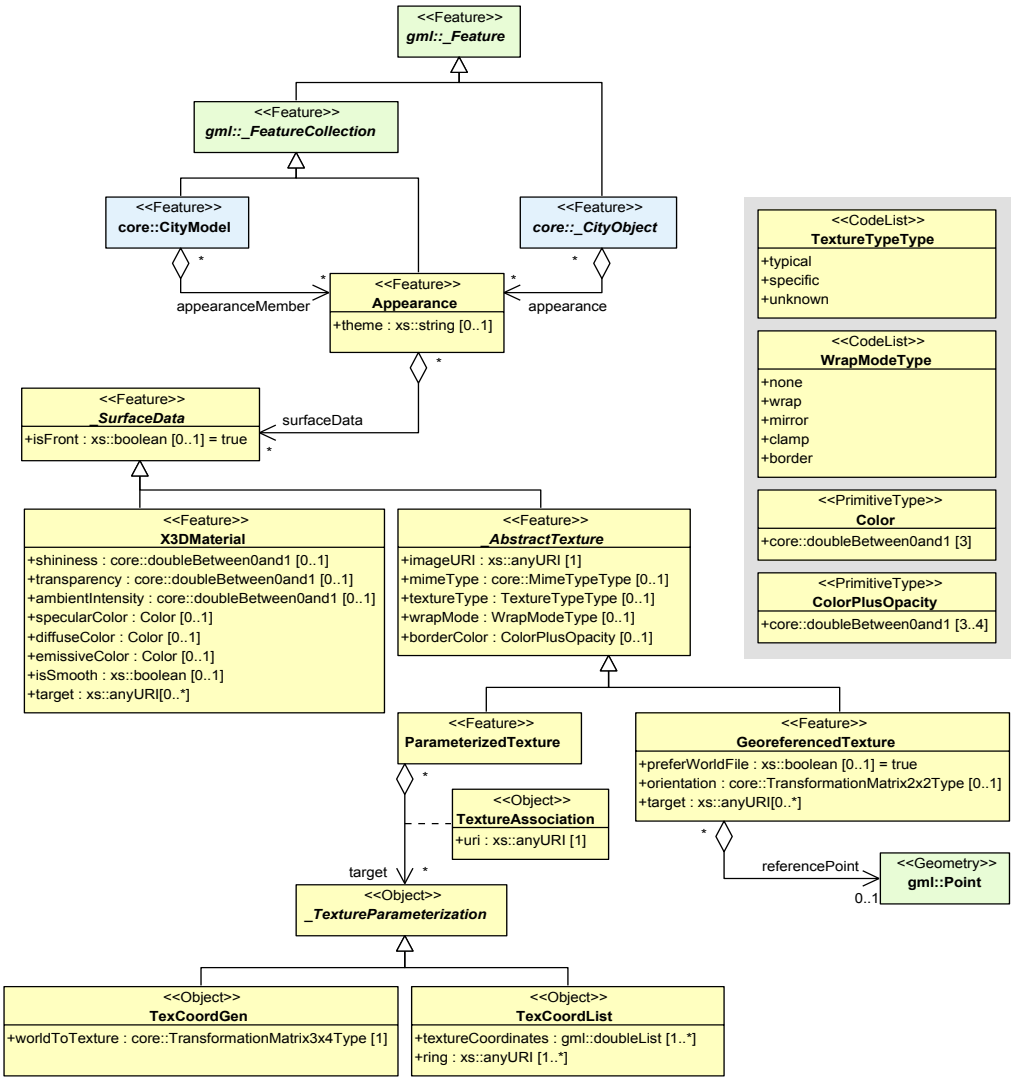


Abb. 4.2: UML-Klassendiagramm des CityGML-Appearancemoduls (aus Gröger u. a., 2008).

Generell sind alle zentralen Klassen des Datenmodells Features und somit durch einen Web Feature Service (WFS) abfragbar. Außerdem implementieren diese Klassen das Fachschalenkonzept, sind also strukturiert erweiterbar.

Die Klasse `Appearance` bettet oberflächenbezogene Daten in die Featurehierarchie von CityGML ein. Jedes `_CityObject` sowie das `CityModel` können eine beliebige Menge von (auch externen, über XLinks referenzierten) `Appearances` besitzen. Jede `Appearance`-Instanz ist einem Thema zugeordnet. So lassen sich zusammengehörende oberflächenbezogene Daten verschiedener Features identifizieren und z. B. gemeinsam in einer Darstellung an- und abschalten. Das Thema ist auch ein Anknüpfungspunkt für externe Präsentationsdefinitionen z. B. durch SLD. In jeder `Appearance`-Instanz sind beliebig viele `_SurfaceData`-Instanzen gespeichert.

Die Klasse `_SurfaceData` ist die Elternklasse für alle Datendefinitionen. Sie unterscheidet zwischen vorder- und rückseitigen Daten. Bezüglich einer Oberfläche, d. h. für ein GML-

Geometrieprimitiv, konstante Daten werden durch die Klasse `X3DMaterial` abgebildet, variable Daten durch `_AbstractTexture` mit den Ableitungen `ParameterizedTexture` und `GeoreferencedTexture`. Die Verbindung zwischen Oberfläche und oberflächenbezogenen Daten wird klassenspezifisch jeweils über ein `target`-Element bzw. eine `target`-Assoziation hergestellt.

`X3DMaterial` wurde direkt aus der X3D-Spezifikation übernommen und einzig um das `target`-Element erweitert. Damit ist eine einfache und verlustfreie Konvertierung bestehender, auf Präsentation beschränkter virtueller 3D-Stadtmodelle möglich, da die Informationen in einem `X3DMaterial` einen Quasi-Standard für die Definition konstanter Oberflächeneigenschaften in der Computergrafik darstellen. Es umfasst Farben und Parameter für verschiedene Beleuchtungskomponenten. Das `target`-Element listet die `gml:ids` der Geometrieobjekte auf, die dieses Material verwenden sollen.

Das Texturkonzept wird zur Abbildung oberflächenpunktabhängiger Daten verwendet. Eine Textur im Sinne von CityGML ist ein regelmäßiges 2D-Raster, an dessen Gitterpunkten (Mess-)Werte vorliegen. Das Raster füllt im 2D-Texturraum unabhängig von der Anzahl der Gitterpunkte immer das Intervall $[0;1] \times [0;1]$ (Foley u. a., 1995; Akenine-Möller u. a., 2008). Es wird technisch als Bild in einem gängigen Bildformat gespeichert. Bereiche außerhalb des Rasterintervalls werden durch das sogenannte Wrapping festgelegt. Damit lassen sich zum Beispiel Bildwiederholungen oder Randfarben definieren. Im Vergleich zu den in der Computergrafik verfügbaren Parametern und Varianten ist diese Texturdefinition deutlich eingeschränkt, erfüllt jedoch die Ansprüche der beabsichtigten Anwendungsfälle. Die Klasse `_AbstractTexture` fasst alle Informationen zur Definition einer Textur zusammen. Dabei wird das Texturbild durch einen Uniform Resource Identifier (URI) beschrieben, kann also auch aus dem Web oder von einem Webservice bezogen werden.

Der gesamte Texturraum wird dann auf die 3D-Oberfläche über eine Texturparametrisierung abgebildet. CityGML unterscheidet drei verschiedene Parametrisierungen:

1. Georeferenzierung (Klasse `GeoreferencedTexture`),
2. homogene lineare Abbildung (Klasse `ParameterizedTexture` mit `TexCoordGen`) und
3. explizite Texturkoordinaten (Klasse `ParameterizedTexture` mit `TexCoordList`).

Georeferenzierung und *homogene lineare Abbildung* verwenden eine lineare Berechnungsvorschrift in Form einer Matrix, um aus den Koordinaten eines Punktes im 3D-Raum direkt dessen Position im Texturraum zu bestimmen. Die Georeferenzierung beschreibt immer eine planimetrische Projektion, während die homogene lineare Abbildung auch perspektivische Projektionen zulässt. Der für die Modellierung entscheidende Unterschied ist die Eindeutigkeit der Abbildung. Eine georeferenzierte Textur hat genau eine gemeinsame Parametrisierung für alle Oberflächen, auf die sie angewendet wird, während eine allgemeine Textur für jede Zieloberfläche eine separate Parametrisierung besitzen kann. Aus diesem Grund speichert die Klasse `GeoreferencedTexture` die Parametrisierung direkt zusammen mit einer einfachen Liste der `gml:ids` der Zielgeometrien. Alternativ können auch nach dem GeoTIFF-Standard oder mittels eines ESRI world files georeferenzierte Bilder verwendet werden. Die Klasse `ParameterizedTexture` jedoch muss die Parametrisierung, d. h. die Matrix, als Eigenschaft des Links zur Oberfläche (`target`-Assoziation) in jeweils einer Instanz der Klasse `TexCoordGen` speichern.

Auf die gleiche Art und Weise unterstützt die Klasse `ParameterizedTexture` auch die Fixierung einer Textur mittels *expliziter Texturkoordinaten*. Dabei wird jedem Eckpunkt einer Zieloberfläche ein Punkt im Texturraum zugeordnet. Die Texturkoordinaten von Punkten innerhalb der Oberfläche werden aus den Eckkoordinaten interpoliert. Dieses Verfahren wird insbesondere bei auf Präsentation beschränkten virtuellen 3D-Stadtmodellen häufig verwendet. Ein Problem ist die Speicherung der Texturkoordinaten, da sie nicht wie in der Computergrafik üblich zusammen mit den Eckpunktkoordinaten abgelegt werden können. Stattdessen wird ein Verweis von den Texturkoordinaten zum entsprechenden Eckpunkt benötigt. Für höhere Effizienz sollen jedoch nicht einzelne Punkte zugeordnet werden, sondern Texturkoordinatenlisten auf ganze Geometrieelemente verweisen. Die kleinsten effizient adressierbaren Elemente einer Oberflächenbeschreibung in GML sind sogenannte Ringe, die die Grenzen polygonaler Oberflächen definieren. Ein GML-Polygon besteht aus mindestens einem Ring, dem Außenring. Es kann jedoch auch beliebig viele zusätzliche, innere Ringe enthalten, die Löcher definieren und deren Eckpunkte ebenfalls Texturkoordinaten benötigen. Würde einem Polygon eine einzige Texturkoordinatenliste zugeordnet werden, würde dies die Zusammenfassung aller Eckpunkte der Ringe erfordern und so eine Softwareimplementierung deutlich erschweren.

Wird eine `ParameterizedTexture` einer Oberfläche zugewiesen (mittels der `target`-Assoziation), speichert die zugehörige Instanz der Klasse `TexCoordList` für jeden Ring der Zielgeometrie eine korrespondierende Liste von Texturkoordinaten. Da so nur Ringe mit Texturkoordinaten versehen werden können, lässt sich zum Beispiel ein Gelände (mit Ausnahme einer `gml:TriangulatedSurface`) nicht mittels expliziter Texturkoordinaten texturieren. In einem solchen Fall muss auf eine der beiden verbleibenden Parametrisierungsvarianten ausgewichen werden, die für beliebige (auch gekrümmte) Oberflächen anwendbar sind.

Neben den aus dem UML-Diagramm ersichtlichen Eigenschaften gibt es eine Reihe von Konsistenzbedingungen, die sich nicht mit UML und auch nicht im XML-Schema ausdrücken lassen. Eine Oberfläche kann beliebig viele Daten zugewiesen bekommen, pro Seite und Thema jedoch maximal ein Material und eine Textur, da Multitexturing von vornherein ausgeschlossen sein soll. Die `gml:id` einer Oberfläche darf demnach insgesamt maximal vier Mal in allen `Appearance`-Instanzen eines Themas verwendet werden. Im Falle hierarchisch zusammengesetzter Geometrie (z. B. mittels `gml:CompositeSurface`) ist es trotzdem möglich, einer Seite einer Oberfläche indirekt mehrere oberflächenbezogene Daten des gleichen Themas zuzuweisen, indem jeder Hierarchiestufe eigene Daten zugewiesen werden. In diesem Fall überschreiben die oberflächenbezogenen Daten der inneren Geometrie die der äußeren. Es findet keine Verrechnung statt. Wird beispielsweise die gesamte Geometrie eines LoD1-Hauses im Thema „Wärmedämmung“ mit einem `X3DMaterial` versehen, das 0,3 als k-Wert (Wärmedurchgangskoeffizient) angibt, und das darin enthaltene Polygon des Daches mit einem `X3DMaterial` für einen k-Wert von 0,5 versehen, so ist 0,5 der korrekte k-Wert des Dachpolygons.

Im Folgenden werden die Anwendungsmöglichkeiten des Appearancemoduls beispielhaft gezeigt. In Abb. 4.1(c) wird ein georeferenziertes Orthofoto als gemeinsame Textur für Gelände und Dächer verwendet. Dazu ist nur eine einzige Instanz der Klasse `GeoreferencedTexture` notwendig, die alle zu texturierenden Oberflächen auflistet. Abb. 4.3 zeigt eine projektive Texturierung. Hier wird das Foto aus Abb. 4.3(a) in das entsprechende virtuelle 3D-Stadtmodell projiziert (4.3(b)). Dazu müssen die Aufnahmeparameter wie Brennweite, Standpunkt und Blickrichtung bekannt sein bzw. über Methoden der Kamerakalibrierung bestimmt werden und in eine entsprechende Projektionsmatrix umgewandelt werden. Abb. 4.4 zeigt die Verwendung mehrerer Themen in einem CityGML-Datensatz und LoD-abhängige oberflächenbezogene

(a) Originalfoto.

(b) Projektion des Fotos auf ein entsprechendes virtuelles 3D-Stadtmodell.

Abb. 4.3: Projektive Texturierung in CityGML. Das Foto (Abb. 4.3(a)) wird als `ParameterizedTexture` in CityGML repräsentiert und mittels einer projektiven Matrix allen relevanten Flächen zugewiesen (Abb. 4.3(b)). Die Matrix kodiert innere und äußere Orientierung des Fotos.

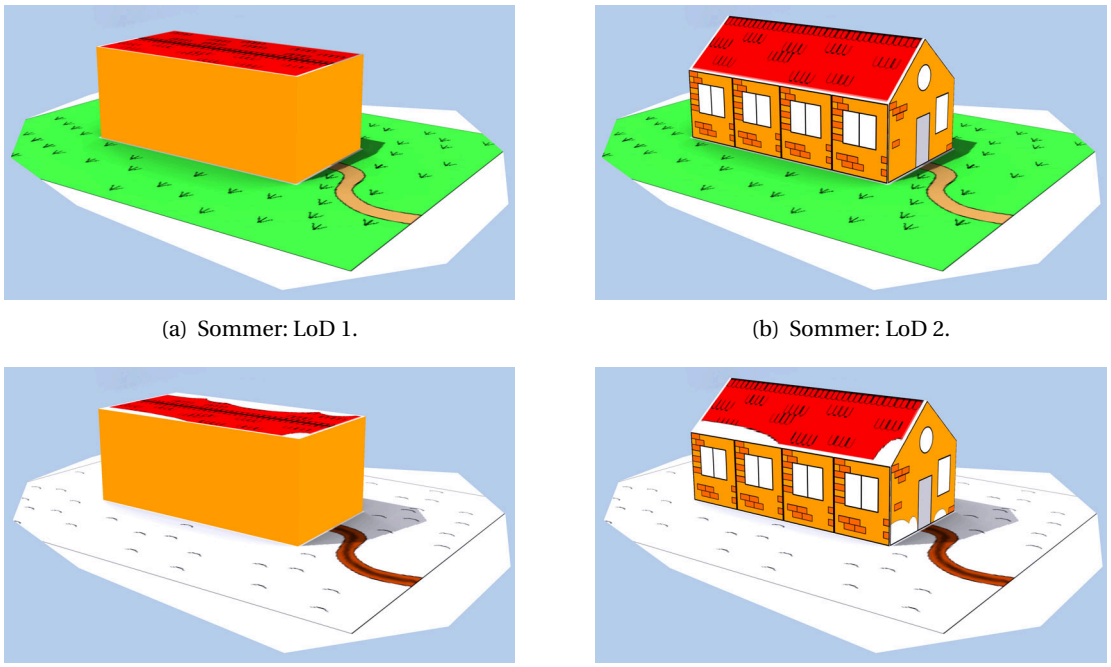
Daten. Es werden zwei Themen „Sommer“ (Abb. 4.4(a) und 4.4(b)) und „Winter“ (Abb. 4.4(c) und 4.4(d)) definiert. Das Gelände und die Dachflächen sind immer mit einer themenabhängigen georeferenzierten Textur versehen. Die Fassade ist in LoD1 nur gefärbt (Abb. 4.4(a) und 4.4(c)), während sie in LoD2 mit einer sich wiederholenden Textur versehen ist (Abb. 4.4(b) und 4.4(d)).

4.4 Anwendungsbeispiele

Das Appearance-Modul wird von verschiedenen CityGML-Implementationen unterstützt und hat sich als flexibles Werkzeug etabliert. Die Akzeptanz wird durch aus unterschiedlichen Quellen stammende, mit Appearancedaten angereicherte CityGML-Datensätze demonstriert (CityGML Wiki, 2010). Hier werden zwei Anwendungsbeispiele gezeigt, die im Rahmen dieser Arbeit entstanden sind.

Das erste Beispiel ist die Speicherung vorberechneter Beleuchtungsinformationen in Texturen. Das zu Grunde liegende Beleuchtungsmodell ist Ambient Occlusion (Döllner u. a., 2006), eine stark vereinfachte Approximation globaler Beleuchtung. Da jede Fläche unterschiedlich beleuchtet wird, können Texturen nicht mehrfach verwendet werden. Es muss demnach eine eindeutige und für einen artefaktfreien visuellen Eindruck möglichst verzerrungsfreie Parametrisierung der Oberfläche gefunden werden. Ein generisches Verfahren ist „Least squares conformal maps“ (Lévy u. a., 2002). Im Falle virtueller 3D-Stadtmodellen reicht jedoch eine Heuristik wie in Kap. 3.3.1 beschrieben aus. Die auf Basis der Parametrisierung berechneten Beleuchtungstexturen können direkt in CityGML unter Verwendung von Texturkoordinatenlisten in einem eigenen Thema gespeichert werden. Abb. 4.5 zeigt ein entsprechendes virtuelles 3D-Stadtmodell. Da die Bedeutung und Verwendung der Appearances nicht in CityGML spezifiziert werden kann, muss der Benutzer in der jeweiligen Visualisierung die Verwendung sofern möglich als vorberechnete Beleuchtungstextur oder alternativ als multiplikative Textur festlegen.

Das zweite Beispiel ist die Speicherung der aus Schrägluftbildern abgeleiteten Fototexturen in CityGML. Im Rahmen der Texturextraktion wurden drei verschiedene Anbindungen an CityGML geschaffen:



(a) Sommer: LoD 1.

(b) Sommer: LoD 2.

(c) Winter: LoD 1.

(d) Winter: LoD 2.

Abb. 4.4: Themen- und LoD-abhängige Texturierung in CityGML. Der Beispiel-Datensatz definiert zwei Themen „Sommer“ und „Winter“, die für ein Gebäudemodell in den LoDs 1 und 2 oberflächengebundene Daten definieren.

1. *Erzeugung texturierter CityGML-Datensätze aus in anderen Formaten gespeicherten virtuellen 3D-Stadtmodellen:* Die in Dreiecke zerlegte texturierte Geometrie wird nach Gebäuden getrennt gespeichert. Weitergehende semantische Informationen werden nicht übertragen.
2. *Hinzufügen eines Themas zu einem existierenden CityGML-Datensatz:* Die Geometrie wird aus einem CityGML-Datensatz gelesen und texturiert. Die Ergebnistexturen werden an den bestehenden CityGML-Datensatz angefügt und den jeweiligen Geometrieobjekten zugewiesen. Die Struktur des Quelldatensatzes bleibt dabei unverändert; alle semantischen Informationen bleiben erhalten. Es können beliebig viele Themen mit jeweils einem Texturierungsdurchlauf zu einem Datensatz hinzugefügt werden.
3. *Hinzufügen eines Themas zu einer CityGML-basierten Datenbank (Stadler u. a., 2008):* Analog dem Hinzufügen eines Themas zu einem CityGML-Datensatz wird bei dieser Anbindung die Geometrie aus der Datenbank gelesen, texturiert und die Texturen als eigenes Thema ohne sonstige Änderungen in der Datenbank ergänzt. Die Kommunikation mit der Datenbank erfolgt über eine Teilbibliothek der Autodesk LandXplorer-Plattform.

Mit der dritten Anbindung wurde die Texturierung des offiziellen Berliner 3D-Stadtmodells (Kap. 3.5) in eine vorhandene Datenbank übertragen. Abb. 3.9 zeigt einen kleinen Ausschnitt dieses virtuellen 3D-Stadtmodells.

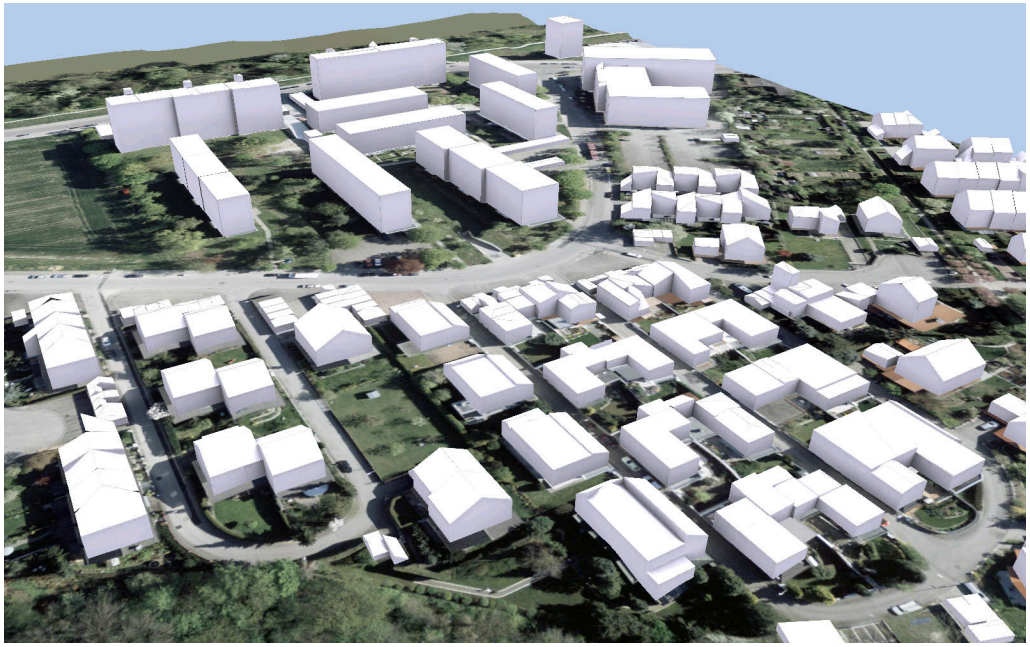


Abb. 4.5: Speicherung von Beleuchtungsinformationen in CityGML. Für das virtuelle 3D-Stadtmodell wurde die Beleuchtung auf Basis von Ambient Occlusion (Döllner u. a., 2006) vorberechnet und in Form von Texturen gespeichert.

4.5 Diskussion

Das Appearance-Modul erweitert CityGML um Möglichkeiten, die vordergründig der Darstellung dienen und verwendet dazu Konzepte aus der Computergrafik. Deshalb gibt es sowohl seitens der Geoinformatik als auch der Computergrafik Kritikpunkte, die hier aufgeführt und diskutiert werden.

„Vermischung von Präsentation und Daten“

Appearances in CityGML definieren keine Präsentation, sondern Daten, die häufig für eine Präsentation genutzt werden. Ein passender Vergleich ist das Luftbild, das im 2D-Kontext als spezieller Datentyp akzeptiert ist, obwohl es hauptsächlich der Präsentation dient. Da es als Messung nicht durch Signaturierung entstehen kann, muss es gespeichert werden.

Eine Definition der Präsentation i. S. v. Kameras, Lichtquellen oder Verrechnungsvorschriften von Themen ist nicht Teil des Appearance-Datenmodells. Es werden ausschließlich oberflächenbezogene Daten modelliert, deren Verwendung zur Präsentation offen bleibt. Derzeit müssen Viewer-Anwendungen eigene Präsentationen definieren. 3D-Entsprechungen zu den 2D-Präsentationsstandards wie SLD oder SE befinden sich derzeit in Entwicklung (Neubauer und Zipf, 2009).

„Texturen sind computergrafische Mittel“

Texturen werden in CityGML als technisches Mittel zur Kodierung und Assoziation raumbezogener Rasterdaten verwendet. Sie sind nicht auf Bilddaten beschränkt, sondern können beliebige Daten in Rasterform auf Oberflächen abbilden. Das Texturkonzept wurde gewählt, da es in Theorie und Praxis sehr gut verstanden und dementsprechend für Entwickler und Nutzer leicht

zugänglich ist. Coverages sind als verwandtes Konzept aus der Geoinformatik prinzipiell für Appearancemodellierung geeignet. Existierende Umsetzungen konzentrieren sich jedoch auf massive, auf eine Ebene bezogene Rasterdaten. Oberflächengebundene Daten mit ihrer dreidimensionalen, stark fragmentierten Bezugsoberfläche lassen mit diesen Umsetzungen nur sehr ineffizient beschreiben. Die zukünftige Verwendung von Coverages als Ersatz für Texturen und Materialien ist jedoch nicht ausgeschlossen.

„Zusammengehörende Daten werden getrennt“

Diese Kritik seitens der Entwickler umfasst zwei Probleme, welche die Konsistenzprüfung eines Datensatzes deutlich erschweren. Zum einen „kennt“ eine Oberfläche nicht alle auf sie bezogenen Daten. Zum anderen werden Texturkoordinaten und Eckpunkte umständlich getrennt, obwohl es eine 1:1-Beziehung gibt. Beides ist das Ergebnis technischer Notwendigkeiten. Da die GML-Geometrieklassen aus Kompatibilitätsgründen durch CityGML nicht erweitert werden sollen, ist es unmöglich, zusätzliche Informationen direkt in der Geometrie zu speichern. Gleichzeitig beziehen sich oberflächenbezogenen Daten immer auf einzelne Oberflächen, d. h. Geometrieobjekte, nicht auf die Gesamtoberfläche eines Features. Somit ist auch die Zuweisung auf Feature-Ebene ausgeschlossen. Es bleibt nur der umgekehrte Weg, bei dem die oberflächenbezogenen Daten selbst auf die Zieloberfläche verweisen.

Die Trennung der Eckpunkte und Texturkoordinaten ergibt sich analog aus der Vermeidung einer Erweiterung der GML-Punktklasse. Es ist durchaus vorstellbar, mit 5- oder 7-dimensionalen Punkten zu arbeiten, eine Zuordnung der einzelnen Dimensionen zu Themen ist jedoch nicht offensichtlich. Die derzeitige Lösung mit zu Ringen korrespondierenden Texturkoordinatenlisten erlaubt eine flexible und einfach zu manipulierende Verwendung von Texturen. Zum Beispiel lässt sich ein Thema mit allen enthaltenen Daten ohne Veränderung der Geometriedaten nur durch Entfernen der korrespondierenden `Appearance`-Instanzen löschen.

5 Interaktive multiperspektivische Ansichten

Perspektivische Ansichten stellen eine virtuelle oder reale 3D-Szene von einem einzelnen Blickpunkt aus mit Hilfe gerader Projektionsstrahlen in einer Bildebene dar, mathematisch beschrieben durch die perspektivische Projektion. Diese Ansichten entsprechen der alltäglichen menschlichen Wahrnehmung der realen Umgebung. Perspektivische Projektionen werden seit langem von der 3D-Grafikhardware explizit unterstützt, ermöglichen 3D-Darstellungen in Echtzeit und decken die Anforderungen vieler Anwendungsfelder im 3D-Bereich ab. Da technisch gesetzt, wurde diese Projektionsform jedoch kaum (Brosz u. a., 2007) grundlegend von Seite der 3D-Computergrafik hinterfragt.

Perspektivische Projektionen führen im Allgemeinen zu Verdeckungen, stark variierenden Objektskalierungen sowie einem begrenzten nutzbaren Sichtfeld (Jobst und Döllner, 2008) – Nachteile, die durch die Interaktionsmöglichkeiten des Nutzers teilweise kompensiert werden können. Ein anderer Weg, diese Probleme zu adressieren, besteht darin, die Projektionsmethode zu erweitern. In dem hier vorgestellten Ansatz werden multiperspektivische Projektionen verwendet, die unterschiedliche perspektivische Betrachtungsstandpunkte in einem einzigen Ergebnisbild vereinen (Yu und McMillan, 2004a). Dadurch können insbesondere Verdeckungen besser aufgelöst, Objektskalierungen gezielt gesteuert sowie zusätzliche Bereiche einer 3D-Welt sichtbar gemacht werden. Multiperspektivische Ansichten ermöglichen eine detaillierte visuelle Repräsentation ausgewählter Bereiche bei gleichzeitiger Erweiterung des Sichtfelds. Darüber hinaus tragen sie dazu bei, den vorhandenen Bildraum effektiver zu nutzen, d. h. den Informationsgehalt zu erhöhen (Keahey, 1998).

Multiperspektivische Ansichten wurden bereits im 11. Jahrhundert von chinesischen Landschaftsmalern verwendet (Vallance und Calder, 2001). Die kartographische Nutzung dieses Ansatzes führte zur Entwicklung sogenannter Panoramakarten, wobei der österreichische Künstler H. C. Berann als Wegbereiter für einen speziellen Typ dieser Karten gilt (Troyer, 2008). Anfang der 30er Jahre kreierte er einen Deformations- und Malstil (Abb. 5.1), der heute als „Berann-Panorama“ bekannt ist. Dieser Stil kombiniert eine vogelperspektivische Darstellung wichtiger Bereiche nahtlos mit einem aus niedriger Perspektive gezeichneten Horizont. Die gesamte Landschaft ist im „natürlichen Realismus“ (Patterson, 2000) dargestellt, während Schlüsselinformationen abstrakt illustriert sind. Die unterschiedliche grafische Gestaltung trägt dazu bei, dass Schlüsselinformationen, die zusätzlich durch die Vogelperspektive weitgehend frei von Verdeckung dargestellt werden, besser erfasst werden können. Gleichzeitig kann sich der Kartenleser anhand der Landmarken, die am Horizont sichtbar werden, orientieren.

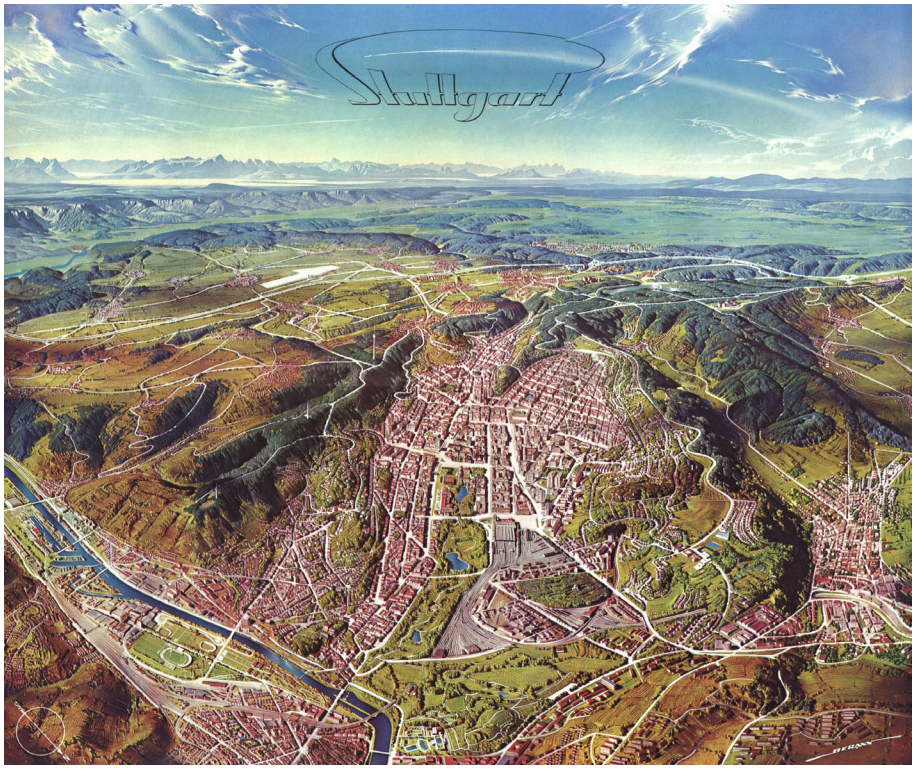


Abb. 5.1: Panoramakarte von Stuttgart (H. C. Berann, 1971, verwendet mit Erlaubnis).

Ziel dieser Arbeit ist die Übertragung der Grundidee der Berann-Panoramakarten auf interaktive Darstellungen geovirtueller 3D-Welten. Im Sinne der Visualisierung nutzen Berann-Panoramakarten den Bildraum effektiver als perspektische Darstellungen, da sie wesentliche Informationen durch Hinzufügen einer zweiten Perspektive ergänzen. Im Gegensatz zu einfachen Formen wie der Einblendung einer 2D-Übersichtskarte gibt es bei Berann-Panoramakarten keinen Bruch zwischen beiden Perspektiven sondern einen kontinuierlichen Übergang. Dieser Übergang verringert den kognitiven Aufwand bei der Aufnahme der Bildinformationen (Cockburn u. a., 2008).

Im Gegensatz zu statischen Abbildungen kann der Nutzer bei interaktiven Darstellungen in der geovirtuellen 3D-Welt navigieren, d. h. den Blickpunkt und die Blickrichtung beeinflussen. Während der Navigation muss der Nutzer die Bildfolge mit Bewegungen im virtuellen Raum assoziieren, um seine Orientierung zu erhalten. Gleichzeitig muss die wahrgenommene Bewegung mit der beabsichtigten Bewegung übereinstimmen, um Konfusion zu vermeiden. Für interaktive perspektivische Darstellungen gibt es sehr viele Arbeiten zur Effektivität und Verbesserung unterschiedlicher Navigationsmetaphern (z. B. Buchholz u. a., 2005; Fuhrmann und MacEachren, 2001; Hand, 1997; Darken und Sibert, 1993). Für multiperspektivische Darstellungen müssen die eingesetzten Navigationsmetaphern auf notwendige Anpassungen geprüft werden.

5.1 Verwandte Arbeiten

Berann-Panoramakarten und die beabsichtigten multiperspektivischen Darstellungen bedienen sich eines Phänomens, das als Fokus&Kontext in der Visualisierung bekannt ist. Dabei enthält eine Darstellung nicht nur die eigentlich relevanten Daten, sondern auch deren Umgebung oder Kontext, mit dem Ziel, den Interpretationsprozess für den Betrachter zu erleichtern. Projektionsbasierte Fokus&Kontext-Ansätze verzerren die Darstellung von 2D- oder 3D-Daten, um mittels Vergrößerung oder Verkleinerung die Relevanz kenntlich zu machen. Leung und Apperley (1994) und Cockburn u. a. (2008) geben eine Übersicht und Carpendale und Montagnese (2001) eine allgemeine Definition. Vallance und Calder (2001) verwenden multiperspektivische Ansichten für Fokus&Kontext-Darstellungen und diskutieren verschiedene Erzeugungstechniken. Eine interessante Anwendung des Fokus&Kontext-Ansatzes für Geodaten wird in Böttger u. a. (2008) vorgestellt, bei der eine Position in einem Orthofoto in den Kontext der gesamten Erde gestellt wird.

Für Berann-Panoramakarten beschreibt Patterson (2000) den manuellen Herstellungsprozess Beranns. Premože (2002) beschreibt darauf aufbauend einen ersten Ansatz, mittels interaktiver computergrafischer Techniken Berann-Panoramakarten zu erzeugen. Multiperspektivische Elemente lässt er jedoch außen vor. Bratkova u. a. (2009) geben eine genaue Analyse der in Berann-Panoramakarten verwendeten grafischen und künstlerischen Techniken. Aus dieser Analyse leiten sie ein automatisches, nicht echtzeitfähiges Verfahren ab, das Bilder im Stil einer Berann-Panoramakarte erzeugt. Schwerpunkt liegt hier auf stilistischen Aspekten, die mittels nicht-fotorealistischer Renderingverfahren im Bildraum implementiert sind. Abgesehen von einer Krümmung des Geländes werden geometrische Manipulationen, die der Minimierung von Verdeckungen dienen, noch nicht berücksichtigt.

Zur Auflösung solcher Verdeckungen beschreiben Takahashi u. a. (2002) ein interaktives System, mit dem sich ein Gelände lokal verformen lässt. Dabei werden die Verformungen in der Bildebene beschrieben und auf die 3D-Geometrie des Geländes abgebildet. Die Arbeit zeigt auch erste Ansätze, notwendige Verformungen automatisiert aus der Geländeform abzuleiten. Takahashi u. a. (2006) vervollständigen die Automatisierung, indem sie die Sichtbarkeit ausgewählter Objekte als Kontrollgröße verwenden. Mit Hilfe dieses Ansatzes implementieren und analysieren sie ein Autonavigationssystem, das Straßen in Echtzeit verdeckungsfrei darstellt, Verformungen des zugrundeliegenden 3D-Geländes jedoch minimiert. Falk u. a. (2007) konzentrieren sich auch auf die Verdeckungsauflösung und verwenden nichtlineares Raytracing, um Berann-Panoramaeffekte zu erreichen. Der Weg der Sichtstrahlen wird durch 2D- und 3D-Kraftfelder beeinflusst. Dabei müssen die Kraftfelder jedoch für den gewünschten Effekt manuell justiert werden. Degener und Klein (2009) stellen eine komplett automatische, wenn auch nicht echtzeitfähige, Technik vor, die eine Darstellung für einen gegebenen Blickpunkt anhand verschiedener Metriken durch Geländeverformung optimiert.

Ein interaktives multiperspektivisches Renderingsystem ähnlich dem hier vorgestellten wird in Möser u. a. (2008) beschrieben. Auch ihrem System liegt als Ziel die verbesserte Nutzung des Bildraums zu Grunde. Dazu beschreiben sie neben einer Verformung des Geländes eine lokale Skalierung zur Hervorhebung wichtiger Objekte und eine lokale isometrische Perspektive zur Darstellung von Fassadentexturen in Vogelperspektiven. Im Vergleich zu dem hier beschriebenen System wird das Profil der Geländeverformung durch eine kubische Hermitekurve beschrieben. Über Positionierung oder Steuerung der Hermitekurve in Abhängigkeit von der Kamera wird keine Aussage getroffen. Ebenso wird die grafische Unterscheidung der

(a) Vogelperspektive. (b) Schrägansicht. (c) Bodennahe Perspektive.

Abb. 5.2: Ausgewählte perspektivische Ansichten einer geovirtuellen 3D-Welt.

verschiedenen Bildbereiche nicht diskutiert. Die Arbeit enthält eine qualitative Auswertung durch Befragung einer kleinen, nicht repräsentativen Testgruppe.

5.2 Analyse ausgewählter perspektivischer Ansichten

Es werden drei typische perspektivische Ansichten geovirtueller 3D-Welten qualitativ in Bezug auf möglichen Orientierungswert (Buchholz u. a., 2005) und die Erkennbarkeit von Objekten analysiert. Beide Aspekte sind unabhängig von der eigentlichen Aufgabe wesentliche Grundeigenschaften einer Geovisualisierung. Der Orientierungswert bezeichnet den Wert der Darstellung zur Bestimmung der Position und Blickrichtung der virtuellen Kamera und hat direkte Auswirkungen auf das Raumbewusstsein und die Navigation (Sebok u. a., 2004). Die Erkennbarkeit von Objekten bezeichnet die Informationsmenge, die für die Zuordnung und Verortung konkreter Objekte zur Verfügung steht und ist Voraussetzung für Informationstransfer, da die Objekte Träger für räumliche Informationen sind.

Für eine sinnvolle Ansicht muss die Kamera grundsätzlich überirdisch positioniert und mindestens ein Teil der Umgebung sichtbar sein. Als Extrempunkte werden die Vogelperspektive, die aus großer Höhe (fast) senkrecht nach unten schaut, und die bodennahe Perspektive, die in geringer Höhe (fast) waagerecht schaut, untersucht. Zusätzlich wird die Schrägansicht, die einige Eigenschaften aufweist, die nicht in den Extremen zu finden sind, analysiert. Dabei lassen sich die drei typischen Ansichten nicht strikt trennen.

Neben Blickpunkt und -richtung beeinflusst auch der Öffnungswinkel der Kamera die Eigenschaften einer Ansicht. In der Analyse wird er jedoch nicht betrachtet, da Veränderungen des Öffnungswinkels immer ähnliche Effekte haben: Vergrößert sich der Öffnungswinkel, wird ein größerer Teil der Umgebung sichtbar, gleichzeitig lassen sich einzelne Objekte aber schwerer erkennen, da der ihnen zustehende Platz im Bild kleiner wird. Verkleinert sich der Öffnungswinkel, werden einzelne Objekte besser erkennbar, aber der sichtbare Teil der Umgebung verkleinert sich.

Vogelperspektive

Die Vogelperspektive sieht aus großer Höhe (annähernd) senkrecht nach unten auf die geovirtuelle 3D-Welt (Abb. 5.2(a)). Die kartenähnliche Darstellung minimiert Verdeckungen und erlaubt

auf Grund geringer perspektivischer Verzerrungen Entfernungsmessungen. Im sichtbaren Bereich lassen sich Informationen mit konstanter Dichte darstellen. Die Vogelperspektive eignet sich damit vor allem für Routen- und Kartendarstellungen.

Der Hauptnachteil ist die schlechte Abbildung von Fassaden- und Höheninformationen. Deshalb lassen sich Orientierungspunkte und Sehenswürdigkeiten nur anhand ihres Grundrisses oder durch Hervorhebungen (z. B. Einfärbung) identifizieren. Ebenso lassen sich Gebäudehöhen nur schwer einschätzen. Außerdem ist der sichtbare Bereich im Vergleich zur Schrägansicht stark eingeschränkt, weshalb evtl. zur Orientierung notwendige Objekte nicht sichtbar sind (Lynch, 1960).

Schrägansicht

Diese Ansicht zeigt einen gerichteten Überblick von einem erhöhten Blickpunkt (Abb. 5.2(b)). Sie stellt Objekte mit einem Teil der Fassadeninformationen und mit vom Sichtabstand abhängigem Maßstab dar. Entfernungsmessungen sind nur eingeschränkt möglich, Schätzungen dagegen schon. Insbesondere das Bestimmen des näheren von zwei Objekten ist einfach.

Die Schrägansicht ermöglicht die Schätzung von Gebäudehöhen und zeigt Fassadeninformationen auf Kosten der Überdeckungsfreiheit. Zudem führen perspektivische Verzerrungen zu vergleichweise kleinen Abbildungen und damit erschwerter Erkennbarkeit entfernter Objekte. Bei ausreichend flachen Blickwinkeln ist auch der Himmel in Schrägansichten sichtbar, der nur einen sehr geringen Orientierungswert besitzt. Für eine aufrechte Kamera verringert sich somit die Informationsmenge im oberen Bildbereich (Jobst und Döllner, 2008).

Im Allgemeinen verhalten sich Orientierungswert und Erkennbarkeit gegensätzlich in Abhängigkeit von der Höhe des Blickpunkts: Desto höher der Blickpunkt, desto besser die Erkennbarkeit aber auch schlechter der Orientierungswert.

Bodennahe Perspektive

Die Blickrichtung der bodennahen Perspektive ist nahezu parallel zum Gelände und der Blickpunkt nicht weiter als ein kleines Vielfaches der durchschnittlichen Gebäudehöhe vom Boden entfernt (Abb. 5.2(c)). Der Begriff reicht damit von der klassischen Fußgängerperspektive in ca. Augenhöhe bis hin zum Tiefflug. Die bodennahe Perspektive wird häufig an Stelle der Fussgängerperspektive verwendet, da heutige virtuelle 3D-Welten meist eine unzureichende Detaillierung (z. B. Texturauflösung) aufweisen.

Die bodennahe Perspektive zeigt die direkte Umgebung mit hohem Detailgrad und hat damit einen hohen Orientierungswert, insbesondere wenn die Umgebung fotorealistisch dargestellt wird. Zusätzlich ist der Horizont mit der Stadtsilhouette sichtbar. Hauptnachteil dieser Perspektive ist die starke Verdeckung. Abgesehen von den nächstliegenden sind alle Gebäude (teilweise) verdeckt. Straßen sind nur sichtbar, wenn sie in einer Sichtachse liegen. Zudem nimmt der Himmel als informationsarmes Element viel Bildraum in Anspruch.

Diskussion

Aus der qualitativen Betrachtung (Tab. 5.1) ist ersichtlich, dass keine der untersuchten Perspektiven in Hinblick auf Orientierungswert und Erkennbarkeit optimal ist. Stattdessen ergänzen sich Vogel- und bodennahe Perspektive, wenn es gelingt, die Vorteile beider zu vereinen.

Berann-Panoramakarten geben ein Beispiel für eine erfolgreiche Vereinigung, da die Vogelperspektive fließend um den Horizont als Element der bodennahen Perspektive ergänzt wird.

Ansicht	*Orientierungswert*	*Erkennbarkeit*
Vogelperspektive	gering	sehr gut
Schrägansicht	gering - gut	gut - gering
Bodennahe Perspektive	sehr gut	gering

Tab. 5.1: Qualitative Bewertung ausgewählter perspektivischer Ansichten geovirtueller 3D-Welten.

Damit wird bei geringer Beeinträchtigung der Erkennbarkeit der Orientierungswert deutlich gesteigert und so der Gesamtwert der Darstellung erhöht bzw. der Bildraum effektiver genutzt. Ein wesentlicher Grund für die Steigerung ist der fließende Übergang zwischen beiden Perspektiven, da damit der kognitive Aufwand zur Korrelation beider Ansichten deutlich kleiner ist als bei traditionellen Ansätzen wie eingeblendeten Übersichtskarten (z. B. bei Google Maps; Hornbæk u. a., 2002). Für die bodennahe Perspektive lässt sich eine ähnliche Vereinigung erzielen: Hier bietet sich der Bildbereich, der den Himmel zeigt, für die Einblendung einer Vogelperspektive an. Auch hier ist ein fließender Übergang beabsichtigt, sodass die zusätzliche Vogelperspektive voraus liegende, normalerweise verdeckte Gebiete zeigt. Damit wird die Erkennbarkeit gesteigert, auch wenn sie die der alleinigen Vogelperspektive nicht erreicht, da Teile der Vogelperspektive durch die bodennahe Perspektive verdeckt werden. Für die Schrägansicht lassen sich beide beschriebenen Varianten verwenden, je nach dem welcher Perspektive die konkrete Ansicht ähnlicher ist.

Durch die Verbindung einer Vogel- mit einer bodennahen Perspektive erhöht sich die Effektivität der Visualisierung, aber gleichzeitig reduziert sich die Effizienz, da die Darstellung nicht der normalen Wahrnehmung entspricht und erst interpretiert werden muss. Der erforderliche Interpretationsaufwand muss so klein wie möglich gehalten werden. Dem Betrachter muss die Multiperspektivität der Darstellung immer klar und deren Konstruktion offensichtlich sein. Hierbei helfen Interaktivität und zeitliche Kohärenz sowie unterschiedliche grafische Gestaltung der beiden Teilperspektiven. Für die beabsichtigte Anwendung sind zusätzlich folgende Aspekte wesentlich:

Distanzschätzung: Wird durch einen zentralperspektivischen Charakter großer Bildbereiche erleichtert.

Objekterkennung: Wird durch Erhaltung von Objektformen und relativen Positionen erleichtert.

5.3 Deformation der geovirtuellen 3D-Welt

Im Allgemeinen erfordert die Erstellung von Berann-Panoramakarten einen erfahrenen Künstler und viel Zeit. Der Erstellungsprozess umfasst die Wahl geeigneter Blickpunkte, eine geeignete Generalisierung der Landschaft, die Identifikation und Integration von Landmarken sowie die Erzeugung eines gleichmäßigen Übergangs zwischen Bereichen mit unterschiedlichen Perspektiven (Patterson, 2000). Selbst wenn digitale Modellierungswerkzeuge und 3D-Geodaten eingesetzt werden, ergibt sich ein aufwendiger, manueller Herstellungsprozess (Premože, 2002). Für interaktive multiperspektivische Ansichten ist ein automatisiertes, in Echtzeit ablaufendes computergrafisches Verfahren notwendig. An dieser Stelle sollen lediglich für die Projektion

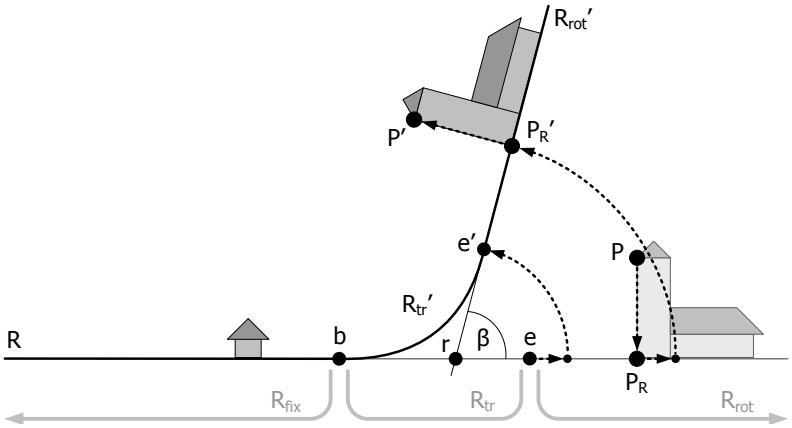


Abb. 5.3: Konstruktion zur deformationsbasierten Erzeugung multiperspektivischer Darstellungen. Eine verformbare Referenzebene R wird in drei Zonen unterteilt: R_{fix} und R_{rot} bilden die beiden unverzerrten Perspektiven. R_{tr} ist eine biegsame, nicht dehnbare Verbindungszone für einen fließenden Übergang zwischen R_{fix} und R_{rot}.

wesentliche Teile betrachtet werden. Weitere Verfahrensaspekte, wie z. B. die Generalisierung der Landschaftselemente, werden nicht untersucht.

Die in Kap. 5.2 entwickelten Anforderungen erfordern die Erhaltung der Wesen der beiden Teilprojektionen und die Minimierung von Verzerrungen. Für die Implementierung der multiperspektivischen Ansicht wird auf eine punktbasierte Umsetzung zurückgegriffen (Kap. 2.3.3). Die Projektion lässt sich dabei einfach und anschaulich durch eine globale Deformation Barr (1984) modellieren. Es wird eine Referenzebene R eingeführt, die in einem schmalen Streifen biegsam, nicht aber dehnbar ist. Die geovirtuelle 3D-Welt wird relativ zur Referenzebene definiert und folgt somit jeder Verformung der Referenzebene. Verzerrungen sind nun auf den Streifen begrenzt, während der Großteil der 3D-Welt unverzerrt, aber unter verschiedenen Perspektiven dargestellt wird. Diese Konstruktion garantiert den fließenden Übergang beider Perspektiven.

Abb. 5.3 zeigt die Konstruktion im Profil. Die hier verwendete Notation weicht geringfügig von der in Lorenz u. a. (2008) ab. Die Referenzebene R ist dreigeteilt: in zwei starre Sektionen R_{fix} und R_{rot} sowie eine biegsame Übergangszone R_{tr}. Die Übergangszone ist begrenzt durch zwei parallele Geraden b und e. Die Verformung von R_{rot} wird als Rotation beschrieben. Dabei fungiert eine zu b parallele Gerade r innerhalb der Übergangszone als Rotationsachse. Der Rotationswinkel β kann positiv oder negativ sein. Durch die Rotation entsteht die Sektion R'_{rot}, deren Beginn durch e' gekennzeichnet ist. Das Profil der dann verformten biegsamen Übergangszone R'_{tr} folgt dabei für C^1-Stetigkeit einer symmetrischen quadratischen Bezierkurve, d. h. $\|b,r\| = \|e',r\| = l_{\text{R}}$. Diese Kurve hat eine Länge l_{Q}. Die Gerade e ergibt sich damit als Parallelverschiebung der Gerade b in Richtung r um den Abstand l_{Q}.

Für die Transformation eines Punktes P ist es notwendig, einen Bezugspunkt $P_{\text{R}} \in R$ zu definieren. Im einfachsten Fall wird dazu der Lotpunkt verwendet. Sollen Verformungen innerhalb von Objekten verhindert werden, kann auch ein einziger Bezugspunkt pro Objekt verwendet werden. In Abhängigkeit der Position des Bezugspunktes lassen sich drei Fälle unterscheiden:

$P_{\text{R}} \in R_{\text{fix}}$: Dieser Bereich soll nicht verformt werden; die Transformation ist die Identität.

$P_R \in R_{rot}$: Dieser Bereich soll als Ganzes um r mit dem Winkel β rotiert werden. Auf Grund der Längenverkürzung der Bezierkurve ($l_Q < 2l_R$) muss eine zusätzliche Korrekturverschiebung beachtet werden: Vor der Rotation muss von r um die Länge $2l_R - l_Q$ in Richtung e verschoben werden.

$P_R \in R_{tr}$: In der Übergangszone soll P in den tangentialen Bezugsrahmen an der zu P_R gehörenden Position P'_R transformiert werden. Das Profil ist dabei als 3D-Bezierkurve $Q(t)$ mit $t \in [0;1]$ gegeben. Zur Vorbereitung wird Q entlang b so verschoben, dass es in einer Ebene mit P_R liegt. Dann ist P'_R ein Punkt auf der Bezierkurve. Die Forderung nach einer dehnungsfreien Verformung innerhalb der Übergangszone erfordert eine Längenparametrisierung von Q. Für quadratische Bezierkurven ist eine analytische Reparametrisierung möglich (Małczak, 2010), erfordert jedoch rechenintensive mathematische Operationen. Die notwendige Reparametrisierung lässt sich auch über eine Tabelle L realisieren, die die Kurvenlänge auf den dazugehörigen Parameter t abbildet und bei Änderungen von Q, d. h. von b, r oder β, aktualisiert wird. Der tangentiale Bezugsrahmen ist nun durch den Richtungsvektor von b, den Tangentenvektor von $Q(L(\|P_R, b\|))$, d. h. deren erste Ableitung $Q'(L(\|P_R, b\|))$, und das Kreuzprodukt der beiden gegeben. Mit diesen drei Basisvektoren lässt sich ein Koordinatensystemwechsel durchführen. Die Gesamttransformation für P ergibt sich damit aus:

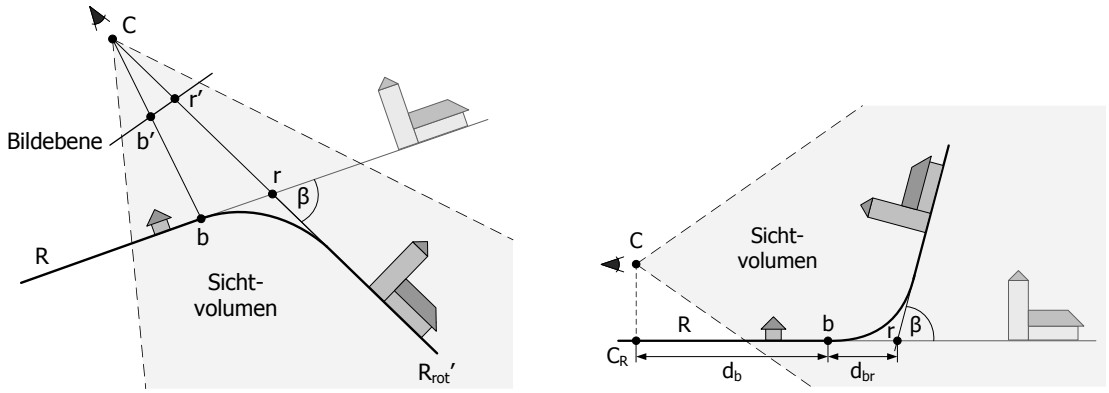
1. Translation um $-P_R$,
2. Koordinatensystemwechsel und
3. Translation um P'_R.

Für die grafische Unterscheidung wird eine Funktion $v(P) \in [0;1]$ definiert. Diese Funktion basiert auf der Zuordnung von P_R zu den einzelnen Abschnitten:

$$v(P) := \begin{cases} 0 & \text{für } P_R \in R_{fix} \\ 1 & \text{für } P_R \in R_{rot} \\ L(\|P_R, b\|) & \text{für } P_R \in R_{tr} \end{cases} \tag{5.1}$$

Mit dieser Definition kann zwischen zwei verschiedenen grafischen Darstellungen (z. B. Texturierung, Material, o. ä.) eines Objekts interpoliert werden.

Bei bekannten Parametern (Referenzebene R, die Geraden b und r, Rotationswinkel β) lässt sich zu einem gegebenen Punkt P das Abbild P' in konstanter Zeit berechnen. Damit ist die Verformung effizient im Vertexshader der GPU implementierbar. Ein Problem ist hierbei jedoch die Beschränkung der 3D-Grafikhardware auf das Zeichnen gerader Kanten. Bei einer Implementierung im Vertexshader lassen sich keine zusätzlichen Punkte erzeugen, weshalb in der Übergangszone bei zu geringer Punktdichte Artefakte sichtbar werden können. Das Problem kann elegant gelöst werden, indem für Objekte in der Übergangszone eine detailliertere Beschreibung verwendet wird (LoD). Alternativ lassen sich auch Verfahren zur dynamischen Verfeinerung (Boubekeur und Schlick, 2008; Lorenz und Döllner, 2008a) einsetzen. Die grafische Unterscheidung erfolgt im Fragmentshader. Hier wird für beide möglichen grafischen Darstellungen ein Farbwert berechnet und das Ergebnis aus der Interpolation mit dem Faktor $v(P)$ bestimmt.



(a) Vogelperspektive.

(b) Bodennahe Perspektive.

Abb. 5.4: Parametrisierung der generischen Verformung für Vogelperspektive und bodennahe Perspektive.

5.4 Einsatzszenarien

Die in Kap. 5.3 beschriebene Konstruktion ist die Basis für die hier entwickelten multiperspektivischen Ansichten. Wesentlich ist dabei die ständige Anpassung der Verformungsparameter an die aktuelle Kameraposition, um den Eindruck der Multiperspektivität zu erhalten. Dazu werden für die Vogelperspektive und bodennahe Perspektive unterschiedliche Parametrisierungen verwendet, aus denen die oben beschriebenen Parameter Bild für Bild berechnet werden. Die Parametrisierungen und Gestaltungen hängen im Wesentlichen vom beabsichtigten Einsatzzweck ab. Grundvoraussetzung ist jeweils die Definition der Referenzebene R, die im Allgemeinen horizontal ungefähr auf Höhe des Geländeverlaufs liegt.

5.4.1 Vogelperspektive

Ziel dieser multiperspektivischen Ansicht ist, wie in Kap. 5.2 beschrieben, die Einbettung des Horizonts in eine Vogelperspektive (Abb. 5.5). Damit wird der Orientierungswert der Darstellung erhöht. Wie in vielen Berann-Panoramakarten soll dabei der Horizont und Himmel nur einen kleinen Teil der Darstellung einnehmen. Der Großteil zeigt weiterhin die möglichst unveränderte Vogelperspektive.

Parametrisierung

In einer interaktiven Anwendung hat der Nutzer u. a. die Möglichkeit den Blickwinkel zu ändern. Um das Wesen der Ansicht für veränderliche Blickwinkel zu erhalten, erfolgt die Parametrisierung im Bildraum. Es werden zwei Geraden b' und r' definiert, deren Rückprojektion auf R die Geraden b und r ergeben (Abb. 5.4(a)). Der Winkel β ergibt sich implizit als Schnittwinkel der Ebene durch den Kamerastandort C und r mit der Ebene R. Damit wird der Horizont im Bild immer an der Gerade r' dargestellt und Verzerrungen bleiben auf den Bereich zwischen r' und b' beschränkt. Der untere Bildbereich bis b' zeigt die unverzerrte Vogelperspektive.

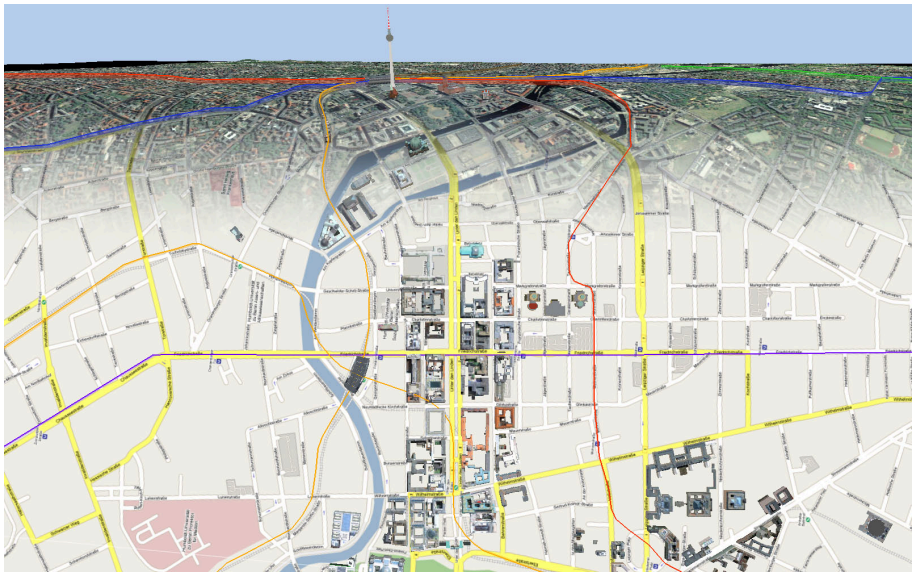


Abb. 5.5: Multiperspektivische Darstellung für Vogelperspektiven. Eine kartenhafte vogelperspektivische Darstellung wird durch den Horizont ergänzt.

Gestaltung

Die Gestaltung der beiden Perspektiven richtet sich nach deren Visualisierungsziel. Die Vogelperspektive dient zur Überblicksgewinnung und soll wesentliche Informationen in der Umgebung zeigen. Fotorealistische Darstellungen sind dafür wenig, kartenähnliche bzw. abstrahierte Darstellungen dagegen eher geeignet (Döllner, 2006). Dreidimensionale Elemente sind auf Grund des überwiegend steilen Blickwinkels eher nachrangig. Stattdessen sollte Wert auf Beschriftungen und Symbole gelegt werden. Das Ziel der bodennahen Perspektive ist die Unterstützung der Orientierung. Wesentliches Element ist die Silhouette, d. h. die dreidimensionale Form der Objekte, die der Betrachter insbesondere bei markanten Objekten relativ einfach mit der Realität vergleichen kann. Zur Unterstützung lassen sich auch Verfahren zur Hervorhebung von Landmarken einsetzen (z. B. Glander u. a., 2007). Die Darstellung der Landmarken darf deren Wiedererkennungswert nicht beeinträchtigen. Sinnvoll sind deshalb fotorealistische oder leicht abstrahierte Darstellungen. Zusätzlich lässt sich der Bereich der Himmelsdarstellung für die Einbettung von Beschriftungen, z. B. Landmarkbezeichnungen oder Richtungen, nutzen.

5.4.2 Bodennahe Perspektive

Die Konstruktion der multiperspektivischen Ansicht für die bodennahe Perspektive ermöglicht die Darstellung im Voraus liegender Gebiete (Abb. 5.6). Damit lassen sich z. B. Routen trotz der hohen Verdeckung einer bodennahen Perspektive einsehen. Die dafür verwendete Vogelperspektive ersetzt die Himmelsdarstellung im Bild.

Parametrisierung

Das Wesen der Ansicht ist nicht vom Blickwinkel abhängig. Es soll lediglich die direkte Umgebung aus bodennaher Perspektive dargestellt werden. Deshalb erfolgt die Parametrisierung dieser Ansicht im Objektraum. Es werden zwei Größen d_{b} und d_{br} definiert (Abb. 5.4(b)). Sie geben den Abstand vom Fußpunkt C_{R} der Kameraposition in R zur Geraden b bzw. von b zu

Abb. 5.6: Multiperspektivische Darstellung für bodennahe Perspektiven. In einer Darstellung aus der bodennahen Perspektive wird der Himmel durch eine kartenhafte vogelperspektivische Darstellung ersetzt.

r an. Die Richtung beider Geraden ist implizit durch die Blickrichtung gegeben, da sie dazu senkrecht in R verlaufen. Der Winkel β ist explizit gegeben.

Gestaltung

Die Gestaltung des Vordergrundes sollte fotorealistisch oder nur leicht abstrahiert erfolgen, da dieser Teil der Orientierung dient und somit der Wiedererkennungswert die höchste Priorität hat. Auf Grund der hohen Verdeckung lassen sich nur wenige zusätzliche Informationen in diesem Teil einbetten. Lediglich Fassaden in direkter Nähe bieten Platz für Beschriftungen oder Symbole (z. B. Maass und Döllner, 2006, 2008). Eine Beschriftung über den jeweiligen Objekten ist wegen der Verdeckung des vogelperspektivischen Teils (Hintergrund) problematisch. Für die Gestaltung des Hintergrunds gilt das in Kap. 5.4.1 Gesagte: Es sollte eine kartenähnliche Darstellung bevorzugt werden.

5.4.3 Navigation

Die Navigation ist größtenteils unbeeinflusst von den hier vorgestellten multiperspektivischen Ansichten. Da ein substantieller Teil des Bildes die originale Perspektive zeigt und sich die Kamera immer im unverformten Teil befindet, können *egozentrische* Navigationsmetaphern, bei denen die Kamera im Zentrum der Aufmerksamkeit steht (Hand, 1997), direkt verwendet werden. Bei *exozentrischen* Metaphern steht ein Objekt im Zentrum der Aufmerksamkeit, welches sich auch im verformten Teil der geovirtuellen 3D-Welt befinden kann. Einige exozentrische Metaphern, z. B. die click-and-fly Metapher („fliege zu einem bestimmten Objekt“, Mackinlay u. a., 1990), können unter Berücksichtigung der Verformung zur korrekten Objektbestimmung weiter verwendet werden. Andere exozentrische Metaphern, z. B. die Fokusnavigation („schaue zu einem bestimmten Objekt ohne Veränderung des Standorts“, Buchholz u. a., 2005), müssen auf ihre Intuitivität geprüft und evtl. angepasst oder ausgeschlossen werden.

Die multiperspektivischen Ansichten haben direkte Auswirkungen auf intelligente bzw. unterstützende Navigationstechniken (Buchholz u. a., 2005), da sich grundlegende Annahmen zur Qualität einer Darstellung ändern und neue Bedingungen hinzukommen. So sollte eine intelligente Navigation für die Vogelperspektive den Nutzer auf sinnvolle Blickwinkel und Flughöhen einschränken. Genauso ist bei der bodennahen Perspektive ein gewisser Blickwinkel nach oben tolerierbar und sogar nützlich, da so mehr von der kartenähnlichen Darstellung im Hintergrund sichtbar wird. Ist der Blick jedoch zu steil nach oben gerichtet, verliert der Betrachter den multiperspektivischen Eindruck. Da dann das Bezugssystem egozentrischer Navigationsmetaphern nicht mehr sichtbar ist, wird die Navigation stark erschwert und die Orientierung geht leicht verloren.

5.5 Anwendungsbeispiele

Der hier vorgestellte Ansatz wurde prototypisch in das Autodesk LandXplorer-Renderingsystem integriert und dann auf ein texturiertes virtuelles 3D-Stadtmodell von Berlin angewendet (Abb. 5.5, 5.6). Für die erfolgreiche Integration mussten zwei projektionsbezogene Details des Renderingsystems angepasst werden:

Culling: Das Culling muss die Deformation der 3D-Welt beachten, damit keine nun sichtbaren Objekte fälschlich entfernt werden. Dazu wird die Konstruktion der gebogenen Referenzebene aus Geschwindigkeitsgründen auf zwei ebene Bereiche, getrennt durch die Rotationsachse r und ohne gebogene Übergangszone, vereinfacht. Für beide Bereiche wird jeweils ein Sichtvolumen bestimmt. Je nach Lage eines Objekts in der 3D-Welt wird der Cullingtest nun für eines der beiden Sichtvolumen durchgeführt. Liegt das Objekt in beiden Teilbereichen, werden beide Tests durchgeführt und das Objekt bei Sichtbarkeit in mindestens einem Bereich gerendert.

Metriken für Gelände- und Texturrendering: Das Renderingsystem setzt für das Gelände- und Texturrendering Multiresolutionsverfahren ein (Baumann, 2000). Dabei wird sichtabhängig die jeweils benötigte Geometrie- bzw. Texturauflösung auf der CPU berechnet und nur diese an die 3D-Grafikhardware zum Rendern gesendet. Für diesen Vorgang muss die Größe eines Objekts auf dem Bildschirm in Pixeln geschätzt und deshalb die Deformation berücksichtigt werden. Auch hier wird die gleiche Vereinfachung wie beim Culling verwendet, so dass sich der Mehraufwand gegenüber der perspektivischen Schätzung auf die Auswahl der gültigen Perspektive beschränkt.

Mit dem so veränderten Renderingsystem lassen sich die multiperspektivischen Ansichten interaktiv verwenden. Die Leistung wurde auf einem Desktop-PC mit AMD Athlon64 X2 Prozessor (2,3 GHz), 2GB RAM und einer NVidia GeForce 7900GT Grafikkarte mit 256MB Grafikspeicher gemessen. Dabei verwendet das System nur einen der beiden Prozessorkerne. Der Datensatz beschreibt die Innenstadt von Berlin mit ca. 16.000 Gebäuden, die mit ca. 270 prototypischen Texturen texturiert sind, ca. 100 objektspezifisch texturierten Landmarken, einem 3GB großen Orthofoto, einer 250MB großen Graustufenkarte (Bebauungsplan) und einem digitalen Geländemodell mit 25m Auflösung.

Tab. 5.2 zeigt, dass die Bildrate der unverformten Darstellung größtenteils unabhängig von der Fenstergröße ist. Demnach könnte der Texturzugriff der Flaschenhals der Anwendung sein. Um die Auswirkungen dieses Problems auf die verformte Darstellung abzuschätzen, zeigt Tab.

Fenstergröße	*Konfiguration*	*Kamerapfad*	*Bilder/s*	*Bilder/s ohne Verformung*
1600×1200	Vogelperspektive	1	8,35	29,86
		2	6,73	17,64
	Bodennahe Perspektive	3	11,72	12,95
		4	19,33	15,69
1024×768	Vogelperspektive	1	8,87	27,24
		2	5,42	16,07
	Bodennahe Perspektive	3	17,85	15,63
		4	22,75	17,69
800×600	Vogelperspektive	1	8,74	27,26
		2	8,48	19,52
	Bodennahe Perspektive	3	20,54	16,49
		4	23,94	18,42

Tab. 5.2: Geschwindigkeitsvergleich multiperspektivischer und perspektivischer Darstellungen für verschiedene Fenstergrößen und Konfigurationen für verformte und unverformte Darstellungen.

Konfiguration	*Kamerapfad*	*Bytes/Bild*	*Bytes/Bild ohne Verformung*
Vogelperspektive	1	5.720.803	190.824
	2	2.555.602	243.067
Bodennahe Perspektive	3	260.207	407.822
	4	122.729	215.398

Tab. 5.3: Vergleich der durchschnittlichen Texturmenge, die pro Bild von der Festplatte geladen werden muss (Auflösung 1600×1200).

5.3 die durchschnittliche Texturmenge, die pro Bild von der Festplatte nachgeladen werden muss.

Die sehr große Texturmenge für die Vogelperspektive (Kamerapfade 1 und 2 in Tab. 5.3) wird durch den sichtbaren Horizont verursacht. Im Vergleich zur unverformten Perspektive ist mehr Gelände sichtbar, womit mehr Texturdaten nachzuladen sind (wenn auch mit geringer Auflösung). Zusätzlich werden bei Blickrichtungsänderungen mehr Texturdaten ungültig und die Cacheeffektivität geringer. Dementsprechend sinkt die Bildrate für die verformte Vogelperspektive deutlich ab, z. B. bei Kamerapfad 1 von ca. 30 auf ca. 8 Bilder pro Sekunde (Kamerapfade 1 und 2 in Tab. 5.2). Bei der verformten bodennahen Perspektive verringert sich dagegen der sichtbare Teil des Geländes. Da sich aber gleichzeitig die erforderliche Auflösung für den nach oben geklappten Teil erhöht, verringert sich die Texturmenge vergleichsweise wenig (Kamerapfade 3 und 4 in Tab. 5.3). Die Veränderung zeigt sich nichtsdestotrotz in der leicht gestiegenen Bildrate (Kamerapfade 3 und 4 in Tab. 5.2).

5.6 Diskussion

Quantitative Untersuchungen in Form von Nutzerstudien wurden im Rahmen der vorliegenden Arbeit nicht durchgeführt; sie werden jedoch in Zusammenarbeit mit Nokia auf Basis deren interaktiver Kartenanwendung OVI Maps (Nokia GmbH, 2010) vorbereitet.

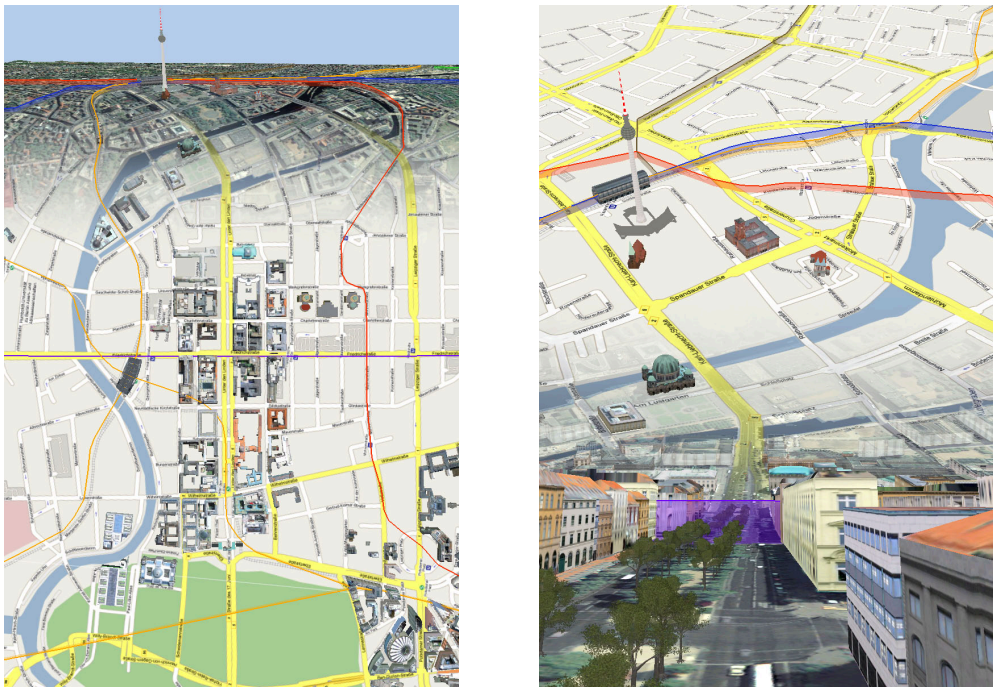
Die vorgestellte Integration in die Autodesk LandXplorer-Plattform ermöglicht die multiperspektivische Darstellung virtueller 3D-Stadtmodelle in interaktiven Anwendungen, deren Hauptziel die Exploration ist. Der Betrachter hat dabei weitreichende Kontrolle über die virtuelle Kamera. Er kann somit auch problemlos und fliessend zwischen den in Kap. 5.2 beschriebenen typischen Ansichten wechseln. Im Zuge dessen müsste sich die Art der multiperspektivischen Darstellung anpassen. Soll die Bewegungsfreiheit des Betrachters erhalten bleiben, muss demnach ein Übergang zwischen beiden Arten in Abhängigkeit von Kameraparametern wie Höhe über Grund und Blickwinkel definiert werden. Zudem müssen atypische Ansichten berücksichtigt werden. Alternativ kann die Bewegungsfreiheit mit Hilfe intelligenter Navigationstechniken (Buchholz u. a., 2005) auf die beschriebenen typischen Ansichten beschränkt werden. Ein Übergang zwischen den Ansichten und damit Darstellungsarten kann durch eine vordefinierte Animation erfolgen.

Aktuelle mobile Geräte wie z. B. Navigationsgeräte, das Apple iPhone oder Android-basierte Smartphones vereinen ausreichend Systemleistung für interaktive Anwendungen mit interessanten Sensoren wie GPS-basiertem Positionsgeber und Kompass. Auf solchen Systemen lässt sich aktive Interaktion durch eine Kopplung der Kamerasteuerung an diese Sensoren minimieren. Dies entlastet den Betrachter bei bestimmten Aufgaben, wie z. B. dem Folgen einer berechneten Route. Nichtsdestotrotz erscheint ein hoher Orientierungswert der Darstellung als Bestätigung und Rückversicherung für den Betrachter sinnvoll. Als zusätzliche Richtungsinformation lässt sich bei der Vogelperspektive auf solchen Geräten die aus der Position und Uhrzeit bestimmte korrekte Sonnenrichtung in die Himmelsdarstellung einbetten.

Die Wirkung multiperspektivischer Darstellungen hängt von verschiedenen Aspekten ab, auf die im Rahmen quantitativer Untersuchungen eingegangen werden sollte:

Bildschirmformat: Einen wesentlichen Einfluss auf die Effektivitätssteigerung hat das Format des Bildschirms. Hierbei ist zu beobachten, dass in hochformatigen Darstellungen der kartenartige Charakter der beiden Multiperspektiven besser zur Geltung kommt, da die jeweilige kartenartige Perspektive im Vergleich zum Querformat (Abb. 5.5, 5.6) mehr und die orientierungsgebende Perspektive weniger Bildraum einnimmt (Abb. 5.7). Der Wert der orientierungsgebenden Perspektive wird dadurch kaum reduziert. Da insbesondere bei der Routenverfolgung die Karte der wesentliche Informationsträger ist, scheint das Hochformat bei einer solchen Nutzung sinnvoll.

Bildschirmauflösung: Die Bildaufteilung zwischen beiden Perspektiven wird wesentlich von der Bildschirmauflösung bestimmt. Für die Erkennung von 3D-Objekten ist eine vom Hintergrund abhängige Mindestanzahl Pixel notwendig (Häberling, 2003). Damit relevante Objekte diese Forderung erfüllen können, benötigt der Teil aus bodennaher Perspektive eine entsprechende Menge Pixel. Bei einer geringen Auflösung muss der bodennahen Perspektive deshalb ein größerer Anteil des Bilds zur Verfügung stehen als bei einer höheren Auflösung. Mit sinkender Auflösung verringert sich folglich der Bildanteil der kartenähnlichen Vogelperspektive.



(a) Vogelperspektive.

(b) Bodennahe Perspektive.

Abb. 5.7: Hochformatige Darstellung multiperspektivischer Ansichten.

Art der Umgebung: Der Nutzen der orientierungsgebenden Perspektive ist nicht konstant, sondern hängt von der darin abgebildeten Umgebung ab. Gibt es zum Beispiel bei der Vogelperspektive keine hohen, markanten Gebäude am Horizont oder bei der bodennahen Perspektive nur eine Wiese als nähere Umgebung, ist der Orientierungswert stark eingeschränkt. Bei der Nutzung im städtischen Raum (Vogel- und bodennahe Perspektive) oder in bergigen Regionen (Vogelperspektive) ist anzunehmen, dass der Orientierungswert hoch ist. In ländlichen oder flachen Regionen wird der Orientierungswert jedoch eher gering sein.

Orientierungsgewohnheiten des Betrachters: Der bodennahen Perspektive ist immer richtungsweisend und kann nicht, wie die Vogelperspektive, beständig nordweisend gehalten werden. Dies kollidiert möglicherweise mit den Orientierungsgewohnheiten des Betrachters und muss bei der Bestimmung der Nutzbarkeit berücksichtigt werden.

6 GPU-basierte dynamische Geometrieverfeinerung

In diesem Kapitel wird ein generisches, 3D-grafikhardwarebeschleunigtes Verfahren vorgestellt, welches nichtlineare Projektionen in interaktiven Anwendungen, z. B. zur Darstellung virtueller 3D-Stadtmodelle, auf Basis von Geometrieverfeinerung ermöglicht. Solche grafikhardwarenahen Geometrieverfeinerungsverfahren zerlegen und ersetzen ein gegebenes geometrisches Primitiv durch eine Menge zusammenhängender Primitive meist einfacheren Typs. Sie lassen sich für zwei Klassen von Problemen einsetzen:

Geometriesynthese: Erzeugt ein von der Renderingpipeline verarbeitbares Polygonnetz aus einer (meist kompakteren) geometrischen Beschreibung. Beispiele sind subdivision surfaces (Akenine-Möller u. a., 2008), displacement mapping (Akenine-Möller u. a., 2008) oder parametrisierte Oberflächen (Foley u. a., 1995) wie Spline- oder Bezierflächen.

Punktbasierte Approximation: Modelliert bzw. simuliert ein Phänomen, wie z. B. globale Beleuchtung (Foley u. a., 1995) oder nichtlineare Projektionen, indem es für diskrete Punkte berechnet bzw. genähert und für weitere Punkte interpoliert wird.

Für beide Klassen liefern Geometrieverfeinerungsverfahren Stützstellen für die Auswertung einer Berechnungsvorschrift. Das Wesen dynamischer Geometrieverfeinerungsverfahren ist die Anpassung der Anzahl der Stützstellen an die jeweilige Situation, z. B. in Abhängigkeit von der aktuellen Ansicht oder von benutzerdefinierten Genauigkeitsanforderungen.

In der Literatur sind bereits eine Reihe generischer (z. B. Boubekeur und Schlick, 2008; Tatarinov, 2008; Lenz u. a., 2009) und anwendungsspezifischer (u. a. Shiue u. a., 2005; Bunnell, 2005) GPU-unterstützter Geometrieverfeinerungsverfahren beschrieben worden. Der Flaschenhals dieser Verfahren ist die Kommunikation zwischen CPU und GPU, da für jedes Bild und jedes Eingabeprimitiv Daten übertragen werden müssen. Damit eignen sich diese Verfahren vornehmlich für Anwendungen mit hohen Verfeinerungsraten. Punktbasierte Approximationen nichtlinearer Projektionen erfüllen diese Eigenschaft nicht. Einzelne Eingabeprimitive können zwar hohe Verfeinerungsraten benötigen (z. B. weil sie eine große Fläche des Bildes füllen), im Durchschnitt ist die Verfeinerungsrate jedoch eher klein. Um die 3D-Grafikhardware trotzdem effizient zu nutzen, muss der Kommunikationsaufwand zwischen CPU und GPU minimiert werden.

In der vorliegenden Arbeit wird ein generisches Verfahren beschrieben, das die speziellen Anforderungen nichtlinearer Projektionen berücksichtigt und vollständig auf der GPU abläuft.

Es beruht auf der Grundidee der baryzentrischen Verfeinerungsmuster und der dazugehörigen Vorgehensweise (Boubekeur und Schlick, 2005a, 2008). Allerdings werden die Auswahl und Instantiierung der Verfeinerungsmuster auf die GPU verlagert. Dazu wird auf Geometryshader zurückgegriffen, deren effiziente Nutzung auf Grund der in Kap. 2.3.2 beschriebenen Restriktionen jedoch nicht offensichtlich ist. Insbesondere muss die verfeinerte Geometrie notwendigerweise zwischengespeichert werden (Kap. 6.3). Damit ist dieses Verfahren weniger effizient für große Verfeinerungsraten und komplementär zu Boubekeur und Schlick (2008).

Das hier vorgestellte Verfahren zeichnet sich durch fünf Haupteigenschaften aus. Es ist

generisch: Es operiert auf Dreiecksnetzen mit beliebiger Topologie und Konnektivität. Es werden keine zusätzlichen Informationen wie Halbkantendatenstrukturen (Akenine-Möller u. a., 2008) benötigt. Es können beliebige Verfeinerungsstrategien und Renderingeffekte verwendet werden.

musterbasiert: Jedes Eingabedreieck wird durch ein vorberechnetes Verfeinerungsmuster ersetzt. Die Muster können beliebig groß sein und jede problembezogene Struktur haben.

nichtuniform: Unterschiedliche Eingabedreiecke können unterschiedliche Verfeinerungen verwenden.

dynamisch: Für jedes Bild wird das angemessene Verfeinerungsmuster zu einem Eingabedreieck bestimmt und verwendet.

minimal und konstant im Kommunikationsaufwand: Es lastet die GPU unabhängig von Eingabe oder Ausgabe stets voll aus.

Die ersten vier Eigenschaften ergeben sich aus dem zugrundeliegenden musterbasierten Ansatz. Der Hauptunterschied zu anderen Verfahren liegt in der fünften Eigenschaft. Anders als dort erfolgt das Rendering trotz einer nichtuniformen Verfeinerung für eine beliebige Anzahl Eingabedreiecke immer mit genau der gleichen Anzahl API-Aufrufe. Zentrale Idee ist die bildweise Aktualisierung der verfeinerten Geometrie statt einer ständigen Neuerzeugung.

6.1 Verwandte Arbeiten

Im Bereich der geometrischen Verfeinerung gibt es eine Reihe von Arbeiten, die die Idee der musterbasierten, generischen Verfeinerung aus Boubekeur und Schlick (2005a) fortführen und verbessern. Boubekeur und Schlick (2005a) stellt eine einfache und elegante Herangehensweise für das Ersetzen von Primitiven durch baryzentrische Muster vor, die zum Beispiel in Dyken u. a. (2008) für die Glättung polygonaler Silhouetten verwendet werden. Die wenigen uniformen Muster wurden in Boubekeur und Schlick (2008) durch adaptive Muster, d. h. mit unterschiedlichen Verfeinerungen pro Seite, ersetzt. Mit einer neuen hardwarebeschleunigten Funktion, dem Instancing, wurden beide Verfahren beschleunigt: Tatarinov (2008) konzentriert sich auf uniforme Muster, Lenz u. a. (2009) auf adaptive. Dyken u. a. (2009) beschreibt die Verwendung semiuniformer Verfeinerungsmuster und greift bei der Implementierung auch auf Instancing zurück. Die in dieser Arbeit entwickelte dynamische Geometrieverfeinerung beruht auch auf adaptiven Verfeinerungsmustern, zeigt jedoch eine neue, alternative Umsetzung.

Ebenfalls auf Mustern beruht ein Verfahren von Guthe u. a. (2005) für NURBS-Oberflächen. Es verwendet vordefinierte Gitter ähnlich den baryzentrischen Verfeinerungsmustern, mit denen bikubische Oberflächen approximiert und gerendert werden. Im Fokus von Guthe u. a.

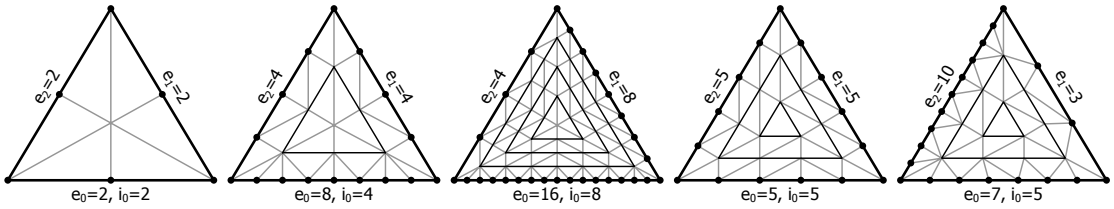
(2005) steht jedoch die effiziente und artefaktfreie Beschneidung der Oberflächen, nicht die Verfeinerung.

Eine zweite Gruppe von Arbeiten beruht auf der Idee der Geometriebilder (Gu u. a., 2002). Diese kodieren Punktattribute in einem Bild, auf das Bildoperationen anwendbar sind und das sich wieder in renderbare Geometrie zurück konvertieren lässt. Wird die Auflösung eines Geometriebildes erhöht, erhöht sich auch die Geometriemenge. Durch geeignete Filter ist so eine Verfeinerung möglich. Losasso u. a. (2003), Bunnell (2005) und Shiue u. a. (2005) nutzen diesen Ansatz für subdivision surfaces. Während die ersten beiden Ansätze die Oberfläche als 2D-Textur darstellen, verwenden Shiue u. a. (2005) ein Spiralschema, um die Oberflächenstücke in einer 1D-Textur zu erfassen. Bunnell (2005) integriert auch Displacementmapping in die Verfeinerung. Einen allgemeinen Ansatz verfolgen Bokeloh und Wand (2006), die gitterförmig repräsentierte Oberflächen hierarchisch unterteilen und so Gitternetze glätten oder fraktale Geländeoberflächen erzeugen können.

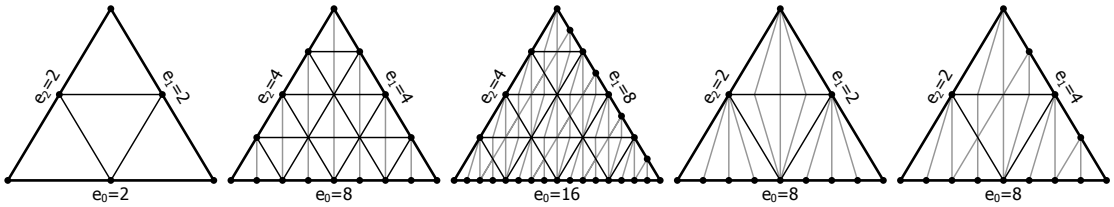
6.2 Verfeinerungsmuster

Verfeinerungsmuster sind generische Dreiecksnetze, die ein Eingabeprimitiv, hier ein Dreieck, ersetzen. Sie sind demnach Zerlegungen einer dreieckigen Domäne. Um sie unabhängig von konkreten Dreiecken zu kodieren, werden die Punktpositionen in baryzentrischen Koordinaten $(u; v; w)^T$ ausgedrückt (Akenine-Möller u. a., 2008), wobei für Punkte in einem Dreieck $u+v+w = 1$ gilt. Bei der Speicherung kann die dritte Koordinate weggelassen werden, da $w = 1-u-v$ gilt. Die Übertragung eines Musters auf ein konkretes Dreieck erfolgt anwendungsspezifisch. Im einfachsten Fall wird eine baryzentrische Kombination verwendet. Dabei ergeben sich die Position und andere Punktattribute als Summe der entsprechenden, mit den baryzentrischen Koordinaten gewichteten Dreieckseckpunktattribute. Falls das Muster auf das projizierte Dreieck übertragen werden soll, z. B. für gleichmäßige Punktdichte im Bild, müssen die baryzentrischen Koordinaten zuerst durch hyperbolische Interpolation (Blinn, 1992) konvertiert werden. Für parametrisierte Oberflächen können die baryzentrischen Koordinaten direkt als Parameter in der Berechnungsvorschrift verwendet werden.

Die Struktur der Verfeinerungsmuster hängt von der Anwendung ab. Sie können zum Beispiel eine homogene Verteilung von Punkten im Raum erzeugen oder eine maximale Kantenlänge garantieren. Gleichmäßige Verfeinerungen werden durch die Anzahl der Unterteilungen der Außenkanten und des Innenbereichs bestimmt. Erfolgt die Verfeinerung immer durch Kantenhalbierung, spricht man von dyadischer Verfeinerung. Abb. 6.1 zeigt beispielhaft gleichmäßige Verfeinerungsmuster, die auf verschiedene Arten algorithmisch erzeugt wurden. Die Direct3D-11-Verfeinerungseinheit verwendet vier Parameter i_0, e_0, e_1 und e_2 (Abb. 6.1(a)). Der Parameter i_0 bestimmt die innere Verfeinerung. Dazu werden $\lfloor i_0/2 \rfloor$ konzentrische innere Dreiecke mit korrespondierender, aus i_0 abgeleiteter Unterteilung erzeugt. Das innerste Dreieck kann zu einem Punkt degeneriert sein. Die Kanten des Originaldreiecks werden den Parametern e_0 bis e_2 entsprechend unterteilt. Die Bereiche zwischen den Dreiecken werden anschließend mit Dreiecksstreifen gefüllt. Diese Art der Verfeinerung ist auf flexible Parametrisierung ausgelegt. Sie ermöglicht beliebige Parameterkombinationen und ist nicht auf dyadische Verfeinerungen limitiert. Es sind auch nichtganzzahlige Parameterwerte gültig, wobei der Nachkommaanteil zwischen zwei Verfeinerungsstufen interpoliert. Eine OpenGL-Erweiterung (OpenGL ARB, 2010) beschreibt den kompletten Algorithmus für alle drei unterstützten Domänen Linen, Dreiecke und Vierecke.



(a) DirectX-11-Verfeinerungsmuster (fett: Originaldreieck, schwarz: innere Dreiecke, grau: Dreiecksstreifen).



(b) Verfeinerungsmuster mit begrenzter Kantenlänge (fett: Originaldreieck, schwarz: rekursiv erzeugte Teildreiecke, grau: Dreiecksnetze).

Abb. 6.1: Beispiele algorithmisch erzeugter Verfeinerungsmuster. e_0, e_1 und e_2 spezifizieren die Verfeinerung der Kanten, i_0 die innere Verfeinerung.

Für nichtlineare Projektionen ist die maximale Kantenlänge des verfeinerten Eingabedreiecks wesentliches Kriterium. Innere Kanten des Musters dürfen nicht länger als Kanten auf der korrespondierenden Außenkante sein. Die Direct3D-11-Verfeinerungsmuster erfüllen diese Eigenschaft nicht. Eine alternative, dyadische Mustererzeugung, die auch im Folgenden verwendet wird, ist in Abb. 6.1(b) dargestellt. Sie verwendet drei Parameter e_0, e_1 und e_2. Zunächst wird das Dreieck rekursiv durch Kantenhalbierung unterteilt, bis alle Außenkanten die Verfeinerung $e = \min(e_0, e_1, e_2)$ erreicht haben. Dann werden alle Teildreiecke mit einem Dreiecksnetz gemäß der Parameter e_0/e, e_1/e und e_2/e gefüllt.

Während die Verfeinerungseinheit Verfeinerungsmuster aus den gegebenen Parametern direkt berechnet, müssen sie für ältere 3D-Grafikhardware vorberechnet werden. Dazu werden für jede erforderliche Parameterkombination baryzentrische Dreiecksnetze erzeugt und in Puffern auf der 3D-Grafikhardware hinterlegt. Das Speicherformat hängt dabei vom verwendeten Verfahren ab: Boubekeur und Schlick (2008) lassen optimierte indizierte Dreiecksstreifen zu; das hier beschriebene Verfahren kann nur mit nichtindizierten Dreieckslisten arbeiten. Für effiziente Ressourcennutzung lassen sich alle Muster in einem gemeinsamen Puffer speichern. Eine zusätzliche Tabelle speichert zu jeder Parameterkombination den zugehörigen Pufferausschnitt. Im Falle dyadischer, indizierter Verfeinerungsmuster ist eine zusätzliche Optimierung durch musterübergreifende gemeinsame Nutzung der baryzentrischen Punkte möglich.

6.3 Technische Umsetzung

Für die Umsetzung der dynamischen Verfeinerung ist es notwendig, sowohl Auswahl als auch Instantiierung der Verfeinerungsmuster auf der GPU auszuführen. Die Auswahl erzeugt pro Eingabedreieck genau einen Parametersatz, weshalb sie sich unkompliziert in einem Vertexshader umsetzen lässt. Die Instantiierung erzeugt jedoch eine variable und möglicherweise sehr große

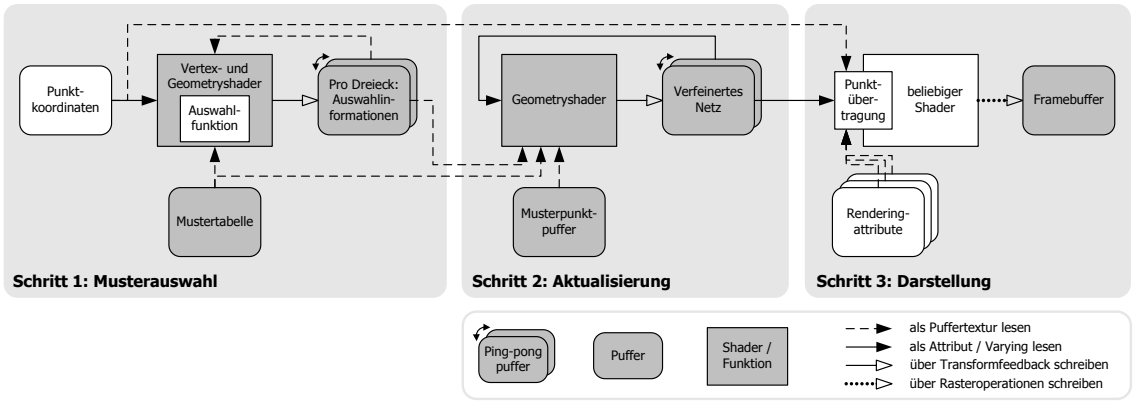


Abb. 6.2: Ablaufdiagramm zur dynamischen Primitivverfeinerung auf der GPU. Weiß dargestellte Elemente sind anwendungsspezifisch.

Menge an Ausgabepunkten pro Eingabedreieck. Die notwendige Flexibilität lässt sich nur mit Geometryshadern auf der GPU erreichen.

Auf Grund der Ausgabelimitierung der Geometryshader ist die Instantiierung nicht in einem Schritt möglich. Ein gängiger Ansatz ist die schrittweise Verfeinerung durch Wiederholung. Dabei gibt ein Geometryshader eine gewisse Höchstmenge von Punkten pro Eingabedreieck aus, z. B. zwölf Punkte für maximal vier neue Dreiecke. Die Ausgabe wird in einen Puffer geschrieben und als Eingabe für den nächsten Verfeinerungsschritt verwendet. Dies wird so lange wiederholt, bis alle Dreiecke die gewünschte Verfeinerung erreicht haben. Für uniforme und selten veränderliche Verfeinerung ist dieses Vorgehen effizient, für den angestrebten Einsatzzweck jedoch nicht. Im Allgemeinen gibt es im vorliegenden Fall nur eine kleine Zahl stark verfeinerter Dreiecke, während der Großteil nur unwesentlich verfeinert werden muss. Da sich die wenigen Dreiecke nicht isoliert betrachten lassen, müssen alle Dreiecke wiederholt prozessiert werden, bis auch das letzte Dreieck korrekt verfeinert wurde.

Die Alternative zur Neuerzeugung ist die Aktualisierung eines gespeicherten verfeinerten Netzes. Da Darstellungen in interaktiven Anwendungen zeitlich kohärent sind, ist das verfeinerte Netz eines Bildes eine gute Näherung für das verfeinerte Netz des nächsten Bildes. Es sind deshalb typischerweise nur wenige Änderungen am verfeinerten Netz notwendig. Operiert ein Geometryshader auf den Dreiecken des verfeinerten Netzes, reicht im Normalfall ein einziger Aktualisierungsschritt pro Bild. Dabei begrenzt das Ausgabelimit nun das Wachstum der Verfeinerung von Bild zu Bild und nicht mehr von Wiederholung zu Wiederholung.

Für die Ausführungsgeschwindigkeit von Geometryshadern ist neben der Anzahl der ausgegebenen Punkte auch die ausgegebene Datenmenge in Bytes von zentraler Bedeutung. Aus diesem Grund speichert das verfeinerte Netz nicht die auf die Eingabedreiecke angewendeten Muster, sondern deren generische Fassung. Das gespeicherte verfeinerte Netz besteht also lediglich aus Kopien der Verfeinerungsmuster, die pro Punkt nur 2 baryzentrische Koordinaten benötigen. Erst in einem anschließenden Schritt wird das generische verfeinerte Netz auf die korrespondierenden Eingabedreiecke angewendet und gerendert.

Insgesamt ergibt sich aus Sicht der CPU ein dreischrittiger Algorithmus, wie er in Abb. 6.2 dargestellt ist. Alle Daten befinden sich im Grafikspeicher. Jeder Schritt wird durch einen API-Renderaufruf gestartet und läuft danach vollständig auf der GPU ab. Der Kommunikationaufwand zwischen CPU und GPU beschränkt sich somit auf genau 3 Zustandsänderungen pro zu

verfeinerndem Dreiecksnetz und ist unabhängig von der Anzahl der Eingabedreiecke. Der grobe Ablauf ist anwendungsunabhängig, einzig die Musterauswahl und das Rendering müssen für konkrete Anwendungen angepasst werden. Beispiele für konkrete Anwendungen sind in Kap. 6.4 beschrieben.

6.3.1 Musterauswahl

Der erste Schritt ist die Musterauswahl, die für jedes Eingabedreieck eines der vorberechneten Verfeinerungsmuster auswählt. Diese Funktion muss sowohl lokal berechenbar sein als auch ein global bruchfreies Dreiecksnetz sicherstellen. Zusätzlich sollte sie die Anzahl der Ausgabedreiecke minimieren.

Die lokale Berechenbarkeit ergibt sich aus der Implementierung in einem Shader. Dieser hat keinen Zugriff auf die Berechnungsergebnisse benachbarter Dreiecke, es sei denn, er berechnet sie selbst. Im Sinne der Ausführungsgeschwindigkeit sind solche Ansätze jedoch nicht effizient. Insbesondere schließt diese Forderung rekursive Verfeinerungsverfahren wie Progressive Meshes (Hoppe, 1996) oder ROAM-basiertes Geländerendering (Duchaineau u. a., 1997) aus.

Im direkten Gegensatz zur lokalen Berechenbarkeit steht die Forderung nach einem global bruchfreien Dreiecksnetz. Dies lässt sich nur durch identische Unterteilung gemeinsamer Kanten der Eingabedreiecke sicherstellen. Werden die Verfeinerungsmuster durch die Unterteilung der Außenkanten beschrieben, muss die Funktion zur Berechnung der Kantenverfeinerung unabhängig von der Kantenorientierung sein.

Die Minimierung der Ausgabedreiecksanzahl ist eine Möglichkeit zur Geschwindigkeitsoptimierung. Zentraler Ansatzpunkt ist die sichtabhängige Musterauswahl. Da das Muster für jedes Bild neu gewählt werden kann, ist eine Optimierung für die jeweilige Sicht über angemessene Metriken möglich und sinnvoll. Der zweite Ansatzpunkt für Minimierung ist das Culling. Da immer ganze Netze verarbeitet werden, ist ein anwendungsseitiges feingranulares Culling auf Dreiecksebene nicht möglich. Stattdessen kann die Musterauswahl Culling insoweit berücksichtigen, unsichtbare Dreiecke nicht zu verfeinern. Prüfungen für Viewport-Culling und Backface-Culling (Persson, 2007) lassen sich direkt integrieren. Der erforderliche Mehraufwand für unsichtbare Dreiecke wird so minimiert.

Um die Konsistenz des gespeicherten verfeinerten Netzes sicher zu stellen, darf ein Dreieck niemals weggelassen werden, d. h. auf null Dreiecke verfeinert werden. Außerdem muss das Verfeinerungswachstum auf das eingestellte Geometryshaderausgabelimit begrenzt werden, da eine Detektion derartiger Überläufe nicht ohne signifikanten Mehraufwand (Auslesen der Geometryshaderausgabe und Prüfung durch die CPU) möglich ist. Sollte eine Verfeinerung stärker wachsen müssen, müssen kurzfristige Brüche im verfeinerten Netz akzeptiert werden.

Die konkrete Umsetzung der Auswahlfunktion erfolgt in einem Geometryshader für Dreiecke. Die notwendigen Eingaben hängen von der Auswahlfunktion ab, umfassen aber meist nur einen Teil der verfügbaren Punktattribute. Zusätzlich werden die Ergebnisse des vorherigen Bildes benötigt, um die Konsistenzbedingungen prüfen zu können. Das Ergebnis besteht aus 3 Informationen, die zusammen als Punktprimitiv ausgegeben werden: dem Index des gewählten Musters m, der Anzahl Dreiecke im gewählten Muster c und der Anzahl Dreiecke im vorherigen Muster c_p. Diese Punktprimitive werden nicht rasterisiert, sondern mittels Transform Feedback in einen Pingpongpuffer geschrieben. Pingpongpuffer bestehen aus zwei Teilen, wobei aus einem Teil gelesen und in den anderen Teil geschrieben wird. Nach einem Schritt werden die Rollen der Teile

getauscht. Durch die Einführung von Puffertexturen ist ein wahlfreier Zugriff auf den lesenden Teil des Pingpongpuffers möglich. Hier werden aus dem lesenden Teil mittels der vordefinierten Systemvariable `primitive_id` die Ergebnisse des vorherigen Bildes ausgelesen.

6.3.2 Aktualisierung des verfeinerten Netzes

Wie bereits beschrieben, besteht das verfeinerte Netz aus hintereinander gehängten Kopien von Verfeinerungsmustern und jedes Muster wiederum aus einer Liste von baryzentrischen Dreiecken. Die Aufgabe des Aktualisierungsschrittes ist es, für jedes originale Eingabedreieck die existente Musterkopie durch eine Kopie des für das aktuelle Bild ausgewählten Musters zu ersetzen. Dieser Schritt ist größtenteils anwendungsunabhängig und muss lediglich bei einer Musterkompression angepasst werden. Für optimale Effizienz muss die Arbeit gleichmäßig auf alle Geometryshaderaufrufe, d. h. auf alle im verfeinerten Netz enthaltenen baryzentrischen Dreiecke, verteilt werden. Dazu müssen die Anzahl der Geometryshaderaufrufe, d. h. der vorhandenen baryzentrischen Dreiecke, und der benötigten baryzentrischen Dreiecke bekannt sein. Während die Anzahl der vorhandenen baryzentrischen Dreiecke bekannt ist, ist es auf Grund der parallelen Ausführung der Musterauswahl nicht möglich, die Gesamtzahl der benötigten baryzentrischen Dreiecke zu ermitteln. Somit kann die Gleichmäßigkeit nicht für das gesamte verfeinerte Netz erreicht werden. Für eine einzelne Musterkopie ist jedoch die Anzahl der vorhandenen und benötigten baryzentrischen Dreiecke bekannt. Somit lässt sich die Arbeit zumindest für einen Kopiervorgang gleichmäßig verteilen. Das verfeinerte Netz ist wie die Ausgabe des ersten Schrittes in einem Pingpongpuffer gespeichert, da es gleichzeitig Ein- und Ausgabe ist.

Eingaben des Aktualisierungsschritts sind das verfeinerte Netz in einem Attributpuffer sowie die Ergebnisse des Auswahlschrittes, die Mustertabelle und das Gesamtmusternetz als Puffertextur. Zum Start wird auf der CPU ein Renderingaufruf mit der Gesamtanzahl an baryzentrischen Dreiecken im vorhandenen verfeinerten Netz ausgeführt. Der zugehörige Geometryshader führt folgende Schritte aus:

1. Identifikation des baryzentrischen Dreiecks und des zugehörigen Eingabedreiecks
2. Identifikation des zugehörigen Musters
3. Bestimmung des zu kopierenden Musterausschnitts
4. Ausgabe des Musterausschnitts

Der erste Schritt ist die Voraussetzung für den Zugriff auf die verschiedenen Puffertexturen. Die notwendigen Informationen müssen deshalb im verfeinerten Netz kodiert sein. Dazu speichert jeder Punkt neben den beiden baryzentrischen Koordinaten u und v auch die laufende Nummer des baryzentrischen Dreiecks i und das zugehörige Eingabedreieck j. Im zweiten Schritt werden die während der Musterauswahl erzeugten Informationen m, c und c_p geladen. Der dritte Schritt bestimmt das erste zu kopierende baryzentrische Dreieck und die zu kopierende Anzahl. Die ersten $c \bmod c_p$ baryzentrischen Dreiecke geben dazu jeweils $\lceil c/c_p \rceil$ und die verbleibenden baryzentrischen Dreiecke $\lfloor c/c_p \rfloor$ Dreiecke aus. Das erste zu kopierende Dreieck ergibt sich aus $\min(i, c \bmod c_p) \cdot \lceil c/c_p \rceil + \max(0, i - c \bmod c_p) \cdot \lfloor c/c_p \rfloor$. Im vierten Schritt werden die Eckpunkte der neuen baryzentrischen Dreiecke aus dem Muster gelesen und zusammen mit einer neuen laufenden Nummer i' sowie der Eingabedreiecksnummer j ausgegeben. Die Ausgabe wird wiederum mittels Transform Feedback im Pingpongpuffer gespeichert und nicht rasterisiert. Der GLSL-Programmausschnitt 6.1 zeigt den zugehörigen Geometryshader.

```
#version 120
#extension GL_EXT_gpu_shader4 : enable
#extension GL_EXT_geometry_shader4 : enable

// x, y = barycentric coord
// z = mainTriID (should be int)
// w = subTriID (should be int)
//varying in vec4 vertexBary[]; // now in gl_PositionIn
varying out vec4 newBary;

// subdivision information about each triangle
// x = number of subtriangles
// y = prev number of subtriangles
// z = start index of pattern
isamplerBuffer currTriInfo;

// vertex buffer with
samplerBuffer subTriangles;

void emitTri(int mainID, int subID, int startIndex)
{
    // find sub triangle
    vec2 sp0 = texelFetchBuffer(subTriangles, startIndex+subID*3+0).ra;
    vec2 sp1 = texelFetchBuffer(subTriangles, startIndex+subID*3+1).ra;
    vec2 sp2 = texelFetchBuffer(subTriangles, startIndex+subID*3+2).ra;
    newBary = vec4(sp0, float(mainID), float(subID));
    EmitVertex();
    newBary = vec4(sp1, float(mainID), float(subID));
    EmitVertex();
    newBary = vec4(sp2, float(mainID), float(subID));
    EmitVertex();
    EndPrimitive();
}

void main()
{
    // STEP 1: get triangle information
    ivec2 triID = ivec2(gl_PositionIn[0].zw);

    // STEP 2: find corresponding pattern triangle
    ivec3 subdivInfo = texelFetchBuffer(currTriInfo, triID.x).xyz;

    // kill all excess triangles
    if (triID.y>=subdivInfo.x) return;

    // STEP 3: determine the number of outputs
    int prevSubTriCount = subdivInfo.y;
    int output = subdivInfo.x/prevSubTriCount + 1;
    int outputLimit = subdivInfo.x%prevSubTriCount;
    int base = triID.y*output;
    if (triID.y>=outputLimit) {
        base = outputLimit * output + (triID.y-outputLimit)*(output-1);
        output -= 1;
    }

    // STEP 4: create each subtriangle
    for (int i = 0; i<output; ++i) {
        emitTri(triID.x, base+i, subdivInfo.z);
    }
}
```

Progr. 6.1: GLSL-Code des Geometryshaders aus Schritt 2 der dynamischen Verfeinerung.

Für die korrekte Funktion des Aktualisierungsschritts muss das verfeinerte Netz vor jeder Aktualisierung drei Konsistenzbedingungen erfüllen:

1. Zu jedem Eingabedreieck muss mindestens ein baryzentrisches Dreieck existieren.
2. Die Anzahl der baryzentrischen Dreiecke pro Eingabedreieck muss mit der im Auswahlschritt gespeicherten (c_p) übereinstimmen.
3. Die zu einem Eingabedreieck gehörenden baryzentrischen Dreiecke müssen fortlaufend durchnummeriert sein.

Bedingung 1 stellt sicher, dass zu jedem Eingabedreieck eine Verfeinerung existiert. Sollte ein Eingabedreieck aus dem verfeinerten Netz verschwinden, wird der beschriebene Aktualisierungsschritt nie mehr eine Verfeinerung dafür erzeugen. Die Bedingungen 2 und 3 stellen vollständige Kopien der ausgewählten Muster sicher. Bei der Ressourcenerzeugung müssen diese Konsistenzbedingungen durch geeignete Initialisierung sichergestellt sein. Während der Ausführung stellt das beschriebene Shaderprogramm die Erhaltung der Bedingungen 2 und 3 sicher. Bedingung 1 muss im Auswahlschritt gesichert werden, indem ein Eingabedreieck niemals zu null Dreiecken verfeinert wird ($c \neq 0$).

Durch die GPU als Ausführungsumgebung gibt es zwei Ausnahmesituationen: Erstens kann jeder Geometryshaderaufruf nicht mehr als eine vorbestimmte Anzahl baryzentrischer Dreiecke ausgeben. Liegt $\lceil c/c_p \rceil$ oder $\lfloor c/c_p \rfloor$ höher als diese Grenze, werden einzelne baryzentrische Ausgabedreiecke stillschweigend unterdrückt und die Bedingungen 2 und 3 verletzt. Demzufolge muss der Auswahlschritt das Verfeinerungswachstum begrenzen und so diese Ausnahmesituation verhindern. Zweitens kann es zu einem Ausgabepufferüberlauf kommen, wodurch wiederum baryzentrische Ausgabedreiecke unterdrückt werden. Diese Ausnahme lässt sich nicht rein GPU-basiert vermeiden. Stattdessen stellt die GPU eine Pufferüberlaufdetektion zur Verfügung, die nach Abschluss des Aktualisierungsschritts von der CPU über API-Funktionen ausgelesen werden kann. Kam es zu einem Pufferüberlauf, muss der Ausgabepuffer vergrößert und die Aktualisierung wiederholt werden. Im Rahmen der Überlaufdetektion wird auch die Gesamtzahl der erzeugten baryzentrischen Dreiecke bestimmt, um sie für den anschließenden Darstellungsschritt und den nächsten Aktualisierungsschritt zu verwenden.

6.3.3 Darstellung

Der abschließende Schritt ist die Darstellung. Er bekommt das aktualisierte verfeinerte Netz als Eingabe, überträgt die darin enthaltenen baryzentrischen Punkte auf alle Punktattribute der Eingabedreiecke und prozessiert die resultierenden Punkte, Dreiecke und Fragmente exakt wie die nichtverfeinerten Eingabedreiecke. Insgesamt ist dieser Schritt vollständig anwendungsspezifisch. Lediglich die initiale Übertragung zu Beginn des Vertexshaders unterscheidet das Rendering der verfeinerten Darstellung von dem einer unverfeinerten.

Für die Übertragung sind die Punktattribute der Eingabedreiecke in Puffertexturen abgelegt. Der Zugriff darauf erfolgt mit dem in den Punkten des verfeinerten Netzes gespeicherten Eingabedreiecksindex j. Die Übertragung der baryzentrischen Koordinaten erfolgt je nach Anwendung wie in Kap. 6.2 beschrieben durch einfache oder hyperbolische baryzentrische Kombination (meist im Kontext punktbasierter Approximationen) oder durch komplexere Algorithmen (meist für Geometriesynthese). Nach der Übertragung sind die Punkte nicht mehr von originalen Eckpunkten der Eingabedreiecke unterscheidbar.

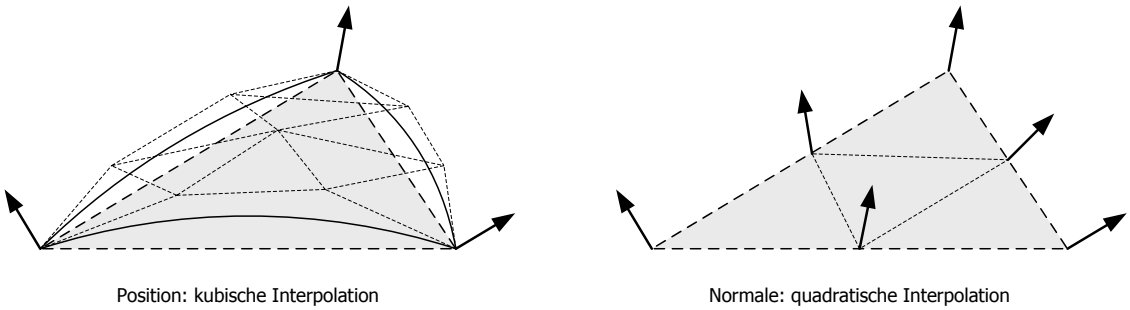


Abb. 6.3: Konstruktion gekrümmter PN-Dreiecken. Die Oberfläche ergibt sich aus einer kubischen Interpolation der Eckpunkte und Punktnormalen. Die zugehörigen Normalen folgen einer quadratischen Interpolation.

Im Falle von Geometriesynthese kann die Geometrie nun direkt rasterisiert werden. Im Falle einer punktbasierten Approximation kann nun das jeweilige Phänomen am aktuellen Punkt berechnet und das Ergebnis für die Rasterisierung verwendet werden. In beiden Fällen können sowohl Geometryshader als auch Fragmentshader unverändert für Darstellungen mit und ohne Verfeinerung eingesetzt werden.

6.4 Anwendungsbeispiele

Im Folgenden werden zwei Anwendungsbeispiele vorgestellt. Dabei wird auf die beiden anwendungsspezifischen Teile, die Musterauswahl und die Darstellung, eingegangen. Beide Anwendungen verwenden die dyadischen Verfeinerungsmuster aus Kap. 6.2.

6.4.1 Gekrümmte PN-Dreiecke

Gekrümmte PN-Dreiecke (Vlachos u. a., 2001) sind eine einfache Heuristik zur Glättung von Dreiecksnetzen und ein Beispiel für Geometriesynthese. Sie lassen sich direkt auf bestehende Inhalte anwenden und im Zusammenspiel mit spezieller Hardware (ATI Inc., 2001) einfach in bestehende Produkte (z. B. Spiele) integrieren. Es werden lediglich Punktposition und -normalen verwendet, um dreieckige Bezieroberflächen über den originalen Dreiecken zu konstruieren. Diese Oberflächen werden im Rahmen eines Verfeinerungsverfahrens ausgewertet. Für grobe Dreiecksnetze, wie sie häufig in Spielen verwendet werden, lässt sich so die visuelle Qualität deutlich steigern. Der Originalansatz wurde in verschiedenen Arbeiten weiter untersucht (Choi u. a., 2004; Boubekeur und Schlick, 2005b; Boubekeur und Alexa, 2008).

In diesem Beispiel wird der originale Ansatz umgesetzt (Abb. 6.3). Die Musterauswahl basiert auf der Länge der geraden Dreieckskanten im Bildraum. Diese Näherung ist akzeptabel, da PN-Dreiecke nur geringfügig von den zugrunde liegenden Dreiecken abweichen und die Berechnung der gekrümmten Kantenlänge deutlich aufwendiger ist. Es wird ein Längenschwellwert definiert, z. B. 10 Pixel, den eine verfeinerte Kante nicht überschreiten darf. Da an der Silhouette Veränderungen in der Verfeinerung deutlicher sichtbar sind, wird ein zweiter, kleinerer Längenschwellwert definiert, wobei je nach Normalenorientierung zwischen beiden Schwellwerten interpoliert wird. Für jede Kante ergibt sich so eine Verfeinerungsstufe und alle drei Stufen zusammen ergeben den Musterindex. Die Darstellung verwendet hier lediglich Phongschattierung.

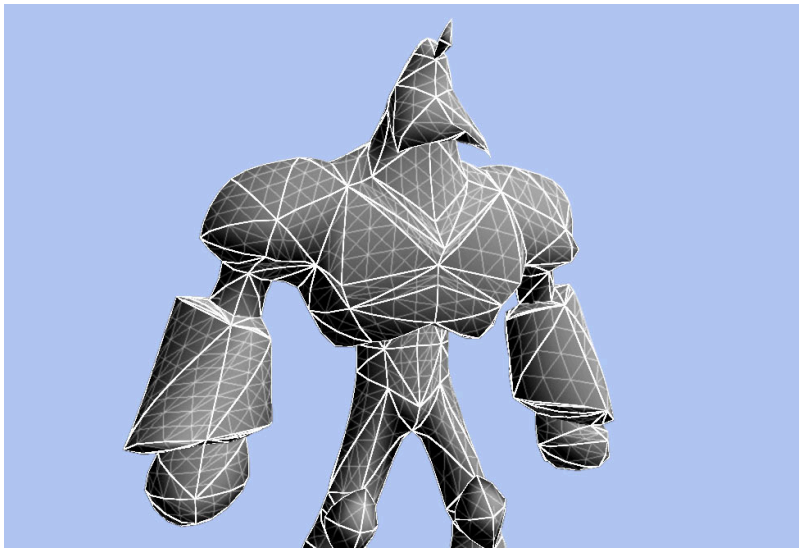


Abb. 6.4: Beispiel für die Verwendung gekrümmter PN-Dreiecke. Fette weiße Linien kennzeichnen das unverfeinerte Dreiecksnetz. Graue Linien kennzeichnen das für die Darstellung verwendete dynamisch verfeinerte Dreiecksnetz.

Die Übertragung der baryzentrischen Koordinaten erfolgt mittels der Formeln aus Boubekeur und Schlick (2005a). Abb. 6.4 zeigt ein Ergebnisbeispiel.

6.4.2 Zylinderprojektion

Die Zylinderprojektion ist ein Beispiel für nichtlineare Projektionen. Sie wird hier als punktbasierte Approximation umgesetzt. Im Folgenden wird auf die Projektion auf einen vertikalen Zylinder eingegangen, die Projektion auf einen horizontalen Zylinder erfolgt analog.

Für die Musterauswahl wird wie bei den gekrümmten PN-Dreiecken pro Dreieckskante die Zahl der Unterteilungen bestimmt. Die hier betrachtete Projektion lässt sich in zwei Teile separieren: eine planare Zentralprojektion in y-Richtung und eine Zentralprojektion auf einen Kreis in x-Richtung (Abb. 6.5). Nichtlineare Effekte, die eine Verfeinerung nötig machen, sind damit lediglich im horizontalen Anteil einer Kante begründet. Die Kantenunterteilung muss demnach den Horizontalwinkel bzw. die horizontale projizierte Länge der resultierenden Segmente begrenzen.

Die Übertragung der baryzentrischen Koordinaten im Darstellungsschritt erfolgt durch einfache baryzentrische Kombination. Anschließend wird die Zylinderprojektion auf den berechneten 3D-Punkt angewendet. Für eine korrekte Darstellung muss ein Sonderfall auf Dreiecksebene, also im Geometryshader, berücksichtigt werden: Die gleichzeitige Sichtbarkeit eines Dreiecks am rechten und linken Bildrand. Bei horizontalen Öffnungswinkeln größer 180° kann ein (verfeinertes) Dreieck hinter der Kamera gleichzeitig am rechten und am linken Bildrand sichtbar sein (Abb. 6.6). In diesem Fall muss das Dreieck zwei Mal an den entsprechenden Bildpositionen dargestellt werden, indem der Geometryshader es doppelt mit unterschiedlichen Eckpunktkoordinaten an den Rasterisierer sendet. Werden Öffnungswinkel größer 360° unterstützt, können Dreiecke sogar an mehr als zwei Bildpositionen sichtbar sein. Abb. 6.7 zeigt eine 160°-Zylinderprojektion.

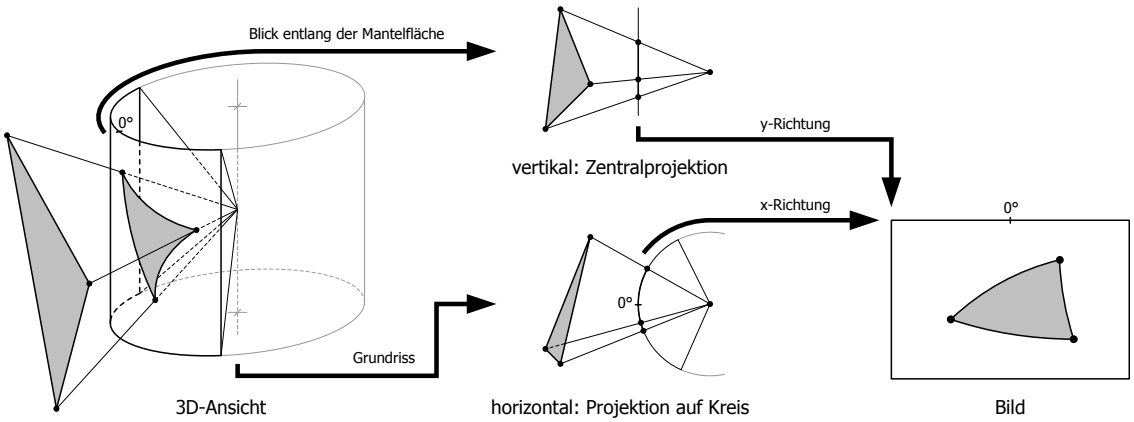


Abb. 6.5: Schema einer vertikalen Zylinderprojektion. Die Projektion setzt sich aus einer Zentralprojektion in vertikaler Richtung und einer Projektion auf einen Kreis in horizontaler Richtung zusammen.

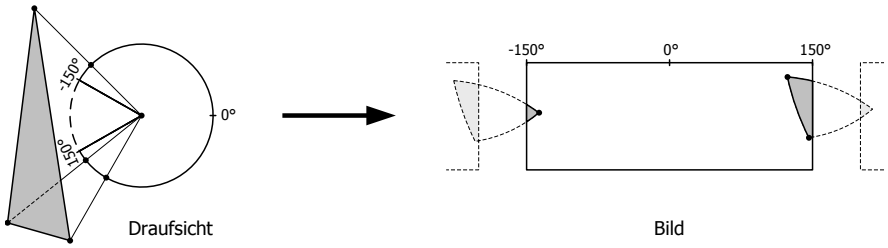


Abb. 6.6: Spezialfall der zylindrischen Projektion eines Dreiecks. Das Dreieck ist sowohl an der rechten als auch linken Bildseite sichtbar und muss deshalb zwei Mal gerendert werden.

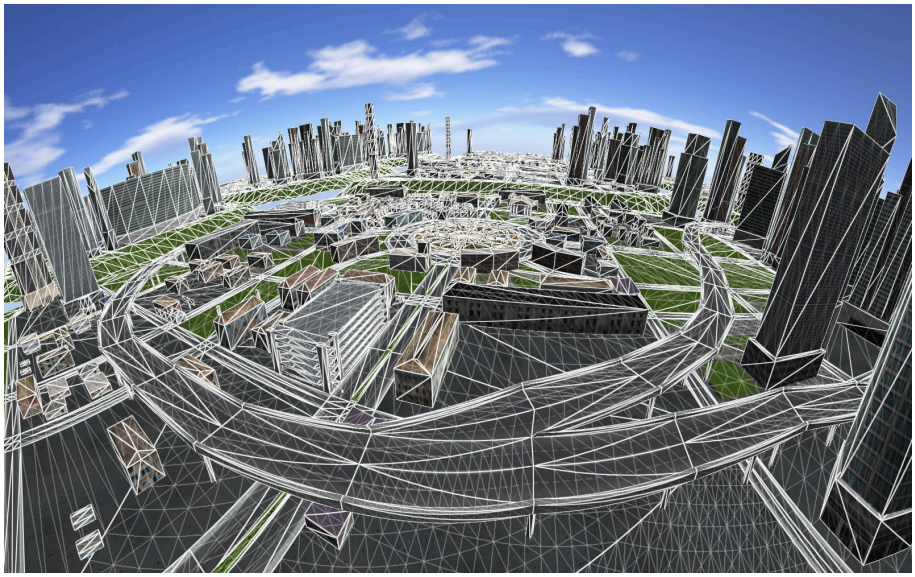


Abb. 6.7: Beispiel für eine Zylinderprojektion. Fette weiße Linien kennzeichnen das unverfeinerte Dreiecksnetz. Graue Linien kennzeichnen das für die Darstellung verwendete dynamisch verfeinerte Dreiecksnetz.

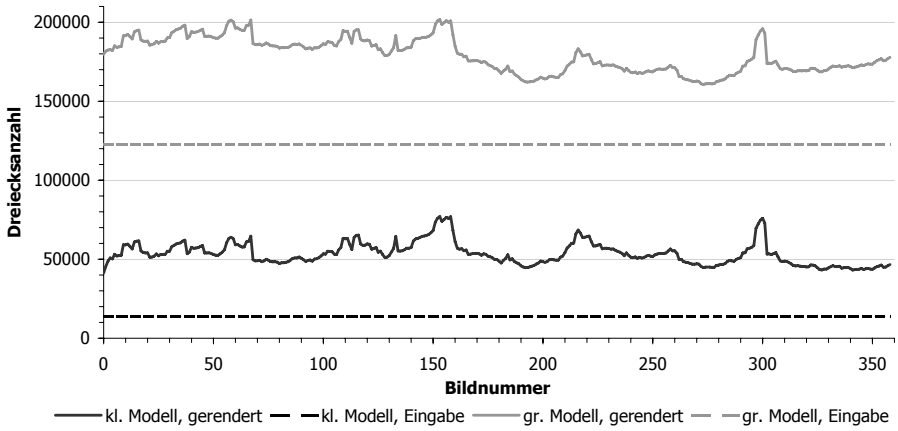


Abb. 6.8: Dreiecksmenge während eines Kamerapfads (Zylinderprojektion). Die Anzahl der Eingabedreiecke ist konstant. Die Anzahl der gerenderten Dreiecke variiert auf Grund der dynamischen Verfeinerung. Verfeinerungskriterium ist eine max. horizontale Kantenlänge von 10 Pixeln.

6.5 Ergebnisse

Die dynamische Verfeinerung wird mit der adaptiven Verfeinerung (Boubekeur und Schlick, 2008) verglichen. Beide Verfahren beruhen auf den gleichen Grundlagen und unterscheiden sich lediglich in der Erzeugung der verfeinerten Geometrie. Zum Vergleich wurde die in Kap. 6.4 beschriebene zylindrische Projektion implementiert. Damit beide Verfahren identische Bilder für gleiche Kameraeinstellungen erzeugen, wird der Musterauswahlschritt der dynamischen Verfeinerung auch für die adaptive Verfeinerung verwendet. Vor der adaptiven Verfeinerung müssen die Ergebnisse des Auswahlschritts in den CPU-Speicher zurück kopiert werden. Für die adaptive Verfeinerung werden die Verfeinerungsmuster mit Hilfe der FStrip-Bibliothek (Reuter u. a., 2005) optimiert. Trotz gewissenhafter Implementierung war es nicht möglich, die in Boubekeur und Schlick (2008) gezeigten Geschwindigkeiten zu reproduzieren[1]. Dies ändert jedoch nichts am qualitativen Geschwindigkeitsverhalten der adaptiven Verfeinerung.

Alle Messungen wurden auf einem PC mit AMD Athlon 64 X2 4400+ Prozessor, 2GB RAM und einer nVidia GeForce 8800GTS Grafikkarte mit 640MB RAM durchgeführt. Die Bildgröße war auf 1600×1200 eingestellt. Die Messungen nutzen den gleichen Flugpfad durch ein kleines (13.639 Dreiecke) und großes (122.751 Dreiecke) texturiertes virtuelles 3D-Stadtmodell.

Abb. 6.8 zeigt die Anzahl der Eingabedreiecke (konstant) und Ausgabedreiecke (variierend) für eine maximale horizontale Kantenlänge von 10 Pixeln. Der Einfluss der sichtabhängigen Verfeinerung ist klar zu erkennen. Ebenso zeigt sich wie erwartet eine eher kleine, wenig variierende durchschnittliche Verfeinerungsrate. Da das kleine virtuelle 3D-Stadtmodell ein Ausschnitt des größeren ist, sind beide Kurven sehr ähnlich.

Abb. 6.9 zeigt die durchschnittliche Dreiecksanzahl und Bildwiederholrate für verschiedene maximale horizontale Kantenlängen. Bei den hier durchgeführten Tests ist die Geschwindigkeit der adaptive Verfeinerung fast unabhängig von der Anzahl der gerenderten Dreiecke. Sie hängt lediglich von der Anzahl der Eingabedreiecke ab, was für die CPU-GPU-Kommunikation als Flaschenhals spricht. Nur bei sehr großen Verfeinerungen mit max. 1 Pixel Kantenlänge ist die

[1] Lt. den Autoren hat der Grafikkartentreiber erheblichen Einfluss auf die Geschwindigkeit der adaptiven Verfeinerung. Sie haben Unterschiede bis zum Faktor fünf festgestellt.

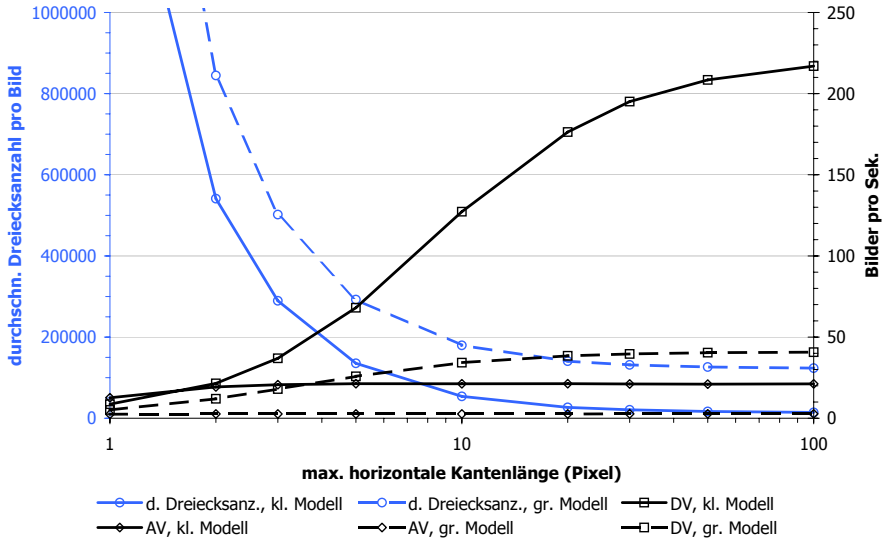


Abb. 6.9: Vergleich der adaptiven (AV) und dynamischen (DV) Verfeinerung. Die weitgehend konstante Bildwiederholrate von AV spricht für einen CPU-seitigen Flaschenhals. DV kann die GPU dagegen voll auslasten. Nur bei sehr starken Verfeinerungen erreicht AV eine höhere Bildwiederholrate als DV. Für sinnvolle max. horizontale Kantenlängen von 10 Pixeln ist DV deutlich schneller.

adaptive Verfeinerung schneller als die dynamische Verfeinerung. Die dynamische Verfeinerung skaliert sowohl mit der Anzahl der Eingabedreiecke als auch mit der der gerenderten Dreiecke. Durch die Minimierung der CPU-GPU-Kommunikation kann die 3D-Grafikhardware immer voll ausgelastet werden, auch bei kleinen Verfeinerungsraten. Dementsprechend ist die dynamische Verfeinerung für sinnvolle Kantenlängengrenzen trotz des komplexen Algorithmus schneller.

Abb. 6.10 analysiert den Einfluss des Geometryshaderausgabelimits im Aktualisierungsschritt auf die Geschwindigkeit. Sie zeigt die Bildwiederholrate für das kleine virtuelle 3D-Stadtmodell bei max. 10 Pixeln horizontaler Kantenlänge für verschiedene Ausgabelimits. Trotz des veränderlichen Ausgabelimits sind die verfeinerten Netze immer identisch (Abb. 6.8). Die zusätzliche Ausgabekapazität wird durch die Limitierung im Musterauswahlschritt nicht genutzt. Nichtsdestotrotz sinkt die Bildwiederholrate deutlich mit zunehmendem Ausgabelimit, da die GPU weniger Geometryshaderinstanzen parallel ausführen kann. Bei einem Limit von 81 Punkten fällt die Kurve plötzlich auf bis hin zu Maximum von 1024 Punkten (nicht mehr dargestellt) konstante 5,75 Bilder pro Sekunde. Vermutlich verwendet der Grafikkartentreiber bei solch hohen Limits stillschweigend eine Softwareemulation des Geometryshaders. Diese Kurve zeigt die Wichtigkeit der Minimierung des Ausgabelimits. In den gezeigten Tests wurden 12 Punkte, d. h. 4 Dreiecke, als Balance zwischen hohen Bildwiederholraten und schnellem Verfeinerungswachstum gewählt.

6.6 Diskussion

Die vorgestellte dynamische Geometrieverfeinerung ist im Allgemeinen für die Implementierung nichtlinearer Projektionen geeignet. Dieser Ansatz unterstützt alle Projektionen, bei denen ein Objektpunkt höchstens einen Bildpunkt besitzt. Sind einem Punkt mehrere Bilder

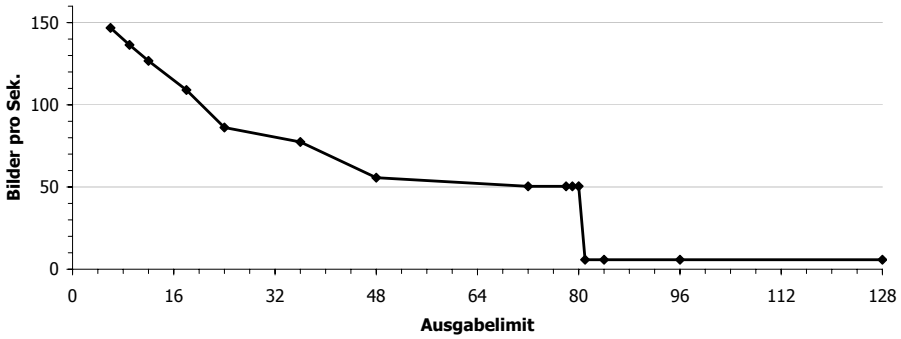


Abb. 6.10: Einfluss des Ausgabelimits des Geometryshaders. Die tatsächlich ausgegebene Datenmenge bleibt in diesem Test bei variierendem Ausgabelimit konstant. Die Bildwiederholrate sinkt bei Erhöhung des Limits und fällt ab 81 Punkten auf konstante 5,75Hz.

zugeordnet, ist eine zusätzliche Replikation, z. B. mittels des Geometryshaders, notwendig. Da zwischen den projizierten Punkten linear interpoliert wird, ist für eine gute Approximation eine ausreichend hohe Punktdichte im Bild notwendig, die über eine dynamische Verfeinerung sichergestellt werden muss. In den vorliegenden Beispielen wurde eine maximale Kantenlänge als Verfeinerungskriterium gewählt. Es sind jedoch auch adaptive Kriterien ähnlich der bei der Multiresolutionsgeländedarstellung eingesetzten Schätzungen des visuellen Geometrie- oder Texturfehlers (Baumann, 2000) denkbar.

Das Verfahren ist für beliebige Dreieckssuppen ohne topologische Bedingungen universell einsetzbar. In der Realität anzutreffende unsauber modellierte Polygonnetze führen jedoch zu vermehrten Bildfehlern. Typische Modellierungsprobleme sind unter anderem nah beieinander liegende koplanare oder sich durchdringende Flächen. Bei perspektivischen Darstellungen führen solche Probleme auf Grund der begrenzten Tiefenpuffergenauigkeit zu Z-Fighting (Akenine-Möller u. a., 2008). Beim Einsatz von Geometrieverfeinerung für nichtlineare Projektionen wird das Z-Fighting verstärkt, da zur begrenzten Tiefenpuffergenauigkeit noch mit Interpolationsfehlern behaftete Tiefenwerte hinzukommen. Am besten lässt sich dies an einem sichtbaren Eckpunkt demonstrieren, der exakt projiziert wird, dessen zugehöriges Fragment somit den korrekten Tiefenwert erhält und auf Grund der Sichtbarkeit im Pixel gespeichert wird. Wird für dasselbe Pixel ein zweites Fragment aus dem Inneren einer anderen Fläche erzeugt, so ist dessen Tiefenwert linear interpoliert und daher mit einem Fehler behaftet. Ist der Interpolationsfehler größer als der tatsächliche Tiefenabstand der beiden Fragmente, kann es zu einer falschen Entscheidung und damit zu Z-Fighting kommen. Eine Lösung ist die korrekte Berechnung des Tiefenwertes für jedes Fragment, was jedoch zusätzliche Fragmentoperationen erfordert und zur Abschaltung hardwareseitiger Optimierungen des Tiefentests (z. B. des early-z-Tests) führt (Persson, 2007).

Die dynamische Geometrieverfeinerung ist auch unabhängig von nichtlinearen Projektionen universell für Geometriesynthese und punktbasierte Approximationen einsetzbar. Mit geeigneten Verfeinerungsmustern kann sie als Ersatz der Direct3D-11-Verfeinerungseinheit auf Direct3D-10-GPUs dienen.

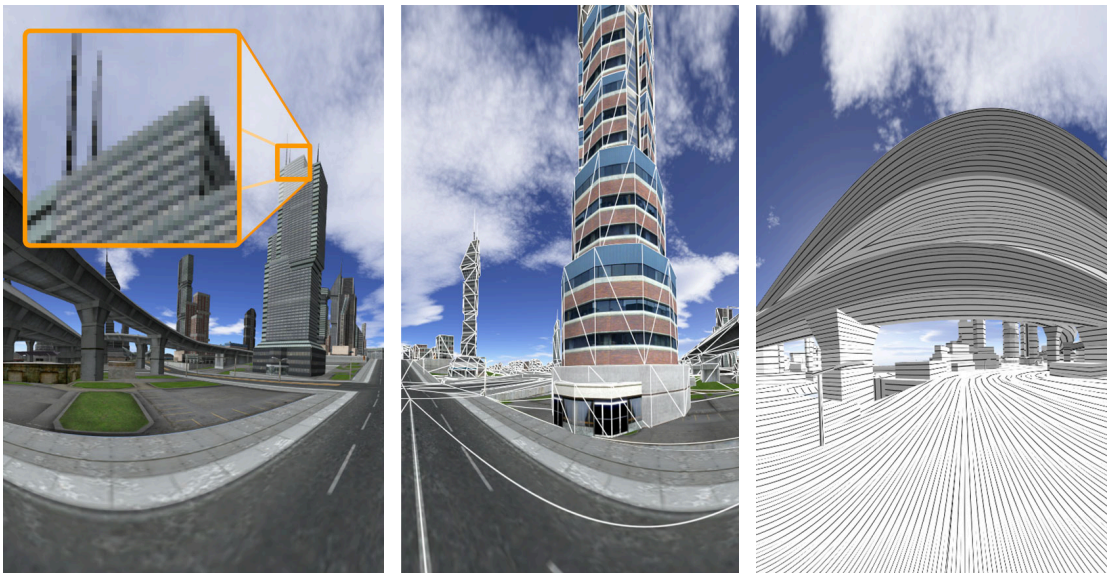
7 GPU-basierte stückweise perspektivische Projektionen

Die in Kap. 6 beschriebene dynamische Geometrieverfeinerung erlaubt die Darstellung nichtlinearer Projektionen ohne Verwendung eines Zwischenbildes, führt jedoch u. U. zu Interpolationsfehlern. Diese Fehler lassen sich nur vermeiden, wenn die Rasterisierung ausschließlich auf linearen Projektionen basiert. Dabei ist es nicht erforderlich, eine einzige lineare Projektion für das gesamte Bild zu verwenden. Für die korrekte Funktion des Tiefenpuffers muss lediglich für jedes Pixel genau eine lineare Projektion gelten. Lässt sich eine nichtlineare Projektion als Zusammensetzung nichtüberlappender linearer Projektionen annähern, so ist auch eine interpolationsfehlerfreie Rasterisierung möglich. Anschaulich werden bei dieser Idee nicht, wie bei der dynamischen Geometrieverfeinerung, die einzelnen Primitive zerlegt, sondern stattdessen das Bild, wobei jedes Teilbild einer eigenen Projektion unterliegt. Deshalb fällt diese Idee in den Bereich der nichtlinearen Rasterisierung. Ein entsprechendes, auf perspektivischen Projektionen basierendes Konzept wird in diesem Kapitel vorgestellt.

Solche *stückweise perspektivischen Projektionen* lassen sich unter voller Ausnutzung der 3D-Grafikhardware und all ihrer qualitätssteigernden Funktionen direkt rendern. Während des Renderns tritt die originale nichtlineare Projektion nicht mehr in Erscheinung, weshalb sogar eine Umsetzung ohne programmierbare Hardware möglich ist. Die Funktion des Tiefenpuffers wird nicht durch zusätzliche Interpolationsfehler beeinträchtigt, weshalb die Darstellungsqualität unsauber modellierter Dreiecksnetze nicht von der unter linearen Projektionen abweicht. Zusätzlich können alle existierenden Renderingeffekte ohne Änderungen und Qualitätseinbußen auch für nichtlineare Projektionen verwendet werden (Abb. 7.1).

7.1 Verwandte Arbeiten

Verwandte Verfahren zur GPU-basierten Darstellung nichtlinearer Projektionen wurden in Kap. 2.3.1 und Kap. 2.3.3 vorgestellt. Das in diesem Kapitel verwendete Konzept der zusammengesetzten Projektionen wurde in Hou u. a. (2006) beschrieben und fällt in die Kategorie der nichtlinearen Rasterisierung. Hou u. a. (2006) verwenden Zusammensetzungen einfacher nichtlinearer Projektionen, sogenannter Dreieckskameras, um komplexe Projektionen anzunähern. Eine Dreieckskamera wird dabei durch eine dreieckige Projektionsfläche und genau einen beliebigen Projektionsstrahl pro Eckpunkt beschrieben. Die Projektionsstrahlen im Inneren des Dreiecks ergeben sich durch baryzentrische Interpolation. Vorteil der Dreieckskameras ist



(a) Einfache Texturierung. (b) Kantenhervorhebung. (c) Schraffureffekt.

Abb. 7.1: Eine mittels stückweise perspektivischer Projektionen erzeugte 360°-Zylinderprojektion unter Verwendung von Multisample-Antialiasing und verschiedenen Darstellungseffekten.

mathematische Einfachheit, die eine fragmentweise Auswertung ermöglicht. Nachteil ist die Nichtlinearität, weshalb eine direkte Rasterisierung nicht möglich ist. Beschränkt man sich statt der Dreieckskameras auf perspektivische Projektionen, ist eine direkte Rasterisierung möglich.

7.2 Konzept

Stückweise perspektivische Projektionen approximieren das Sichtvolumen nichtlinearer Projektionen durch eine Menge von verbundenen, nichtüberlappenden perspektivischen Projektionsvolumina. Der Inhalt jedes Teilvolumens unterliegt einer perspektivischen Projektion. Das Abbild der nichtlinearen Projektion ergibt sich aus der Kombination der Abbilder der Teilprojektionen. Alle nichtlinearen Eigenschaften der Projektion werden durch die Struktur der stückweise perspektivischen Projektion kodiert. Die Anzahl der Teilprojektionen bestimmt die Güte der Approximation.

Die Beschränkung auf perspektivische Projektionen als Elemente der zusammengesetzten Projektion führt zu einer Einschränkung der approximierbaren nichtlinearen Projektionen. Jede perspektivische Teilprojektion hat genau ein punktförmiges Projektionszentrum. Für eine nahtlose Darstellung müssen die Projektionsstrahlen an den Projektionsgrenzen kohärent sein, was für alle Teilprojektionen das gleiche punktförmige Projektionszentrum erzwingt. Es können also nur Darstellungen mit einem einzigen punktförmigen Projektionszentrum erzeugt werden. Zudem müssen die Projektionsstrahlen gerade sein.

Konkret lassen sich mit stückweise perspektivischen Projektionen die gleichen Projektionen abbilden wie mit dem bildbasierten Ansatz von Yang u. a. (2005). Deren Ansatz erzeugt ein perspektivisches Zwischenbild, dessen Projektion durch eine Matrix M_{P} beschrieben wird. Danach wird dieses Bild durch Texturemapping auf einem Dreiecksnetz verzerrt. Dabei definiert jedes

Dreieck implizit eine affine Transformation $M(t)$ eines Zwischenbildausschnitts in den Bildpuffer. Eine äquivalente stückweise perspektivische Projektion lässt sich aus der Kombination aller Projektionen $M(t) \cdot M_{\mathrm{P}}$ erzeugen, die jeweils auf die Bildpuffergrenzen des zugehörigen Dreiecks begrenzt werden.

Die Technik von Hou u. a. (2006) kann eine größere Klasse nichtlinearer Projektionen darstellen. Diese Technik ist lediglich auf gerade Projektionsstrahlen festgelegt und kann ansonsten jede nichtlineare Projektion, selbst mit überlappenden Projektionsstrahlen, approximieren. Damit ist dieses Verfahren auch für das Rendering physikalischer Phänomene wie Spiegelungen, Brechungen oder Kaustiken an beliebigen Oberflächen geeignet. Stückweise perspektivische Projektionen können solche Effekte nur für spezielle Körperformen darstellen.

Das Rendering kann direkt in den Bildpuffer erfolgen, da sich die Bilder der Teilprojektionen nicht überlappen. Dazu müssen drei Aspekte betrachtet werden:

Approximation der nichtlinearen Projektion durch eine zusammengesetzte Projektion: Das Ergebnis dieses Schritts ist eine Menge von Projektionsmatrizen, die die Teilprojektionen beschreiben. Die Approximation muss nur bei Änderung der Projektion durchgeführt werden. Wesentliches Kriterium ist die Kohärenz und somit Bruchfreiheit an Teilprojektionsgrenzen. Die konkrete Bestimmung der Approximation ist projektionsabhängig. In Kap. 7.6 werden zwei typische Ansätze vorgestellt.

Rendering aller Primitive in alle relevanten Teilprojektionen: Während der Rasterisierung eines Primitivs kann nur eine perspektivische Projektion aktiv sein. Ist ein Primitiv in mehreren Teilprojektionen sichtbar, muss es auch mehrfach rasterisiert werden, was durch das Rendering sichergestellt werden muss.

Clipping der Renderingausgaben auf die Grenzen der jeweiligen Teilprojektion: Die Renderingausgabe muss auf den jeweiligen Teilprojektionsbereich begrenzt werden. Dazu können hardwareseitig verfügbare benutzerdefinierte Clippingebenen verwendet werden. Eine fehlerträchtige explizite Clippingimplementierung in einem Shader ist nicht notwendig.

Im Folgenden werden drei generische Umsetzungen für Rendering und Clipping beschrieben. Sie sind in ihrem Funktionsumfang identisch und unterscheiden sich nur in den genutzten 3D-Grafikhardwarefunktionen und damit erzielbaren Geschwindigkeiten.

7.3 Klassische Umsetzung

Da die stückweise perspektivische Projektion keinerlei nichtlineare Anteile mehr enthält, lässt sie sich in bestehenden Grafiksystemen einfach mittels Multipassrendering umsetzen. Dazu muss jede Teilprojektion durch eine systemspezifische Kamera und dazugehörige Clippingebenen beschrieben werden. Zur Erzeugung eines Bildes kann dann die unveränderte Originalrenderingfunktion verwendet werden, indem für jede Teilprojektion die entsprechende Kamera und Clippingebenen aktiviert werden und damit die Szene gerendert wird. Der Programmausschnitt 7.1 zeigt dieses Vorgehen in Pseudocode.

Für die korrekte Funktion darf die Originalrenderingfunktion die Kameraeinstellungen und Clippingebenen nicht verändern. Typische Effekte wie Schatten oder Reflexionen, die in temporäre Puffer rendern, müssen vor der stückweise perspektivischen Projektion durchgeführt

```
function render_ppp()
begin
  foreach projection piece p do
    activate camera of p
    activate clip planes of p
    render()
  end
end
```

Progr. 7.1: Pseudocode der Renderingfunktion der klassischen Umsetzung.

werden. Ebenso darf die Originalrenderingfunktion keine Bildumschaltung bei Doppelpufferung durchführen. Für optimale Geschwindigkeit sollte das in der Originalrenderingfunktion verwendete Culling die zusätzlichen Clippingebenen berücksichtigen, da die Teilprojektionen jeweils nur einen sehr kleinen Bildbereich einnehmen.

Diese Umsetzung erfordert durch das häufige Culling vergleichsweise viel CPU-Rechenleistung. Zusätzlich werden sehr viele Zustandsänderungen und Renderingaufrufe ausgeführt. Teilprojektionsübergreifende zustandsbasierte Optimierungen der Szenenbeschreibung sind nicht möglich. Aus diesen Gründen ist dieser Ansatz CPU-limitiert. Die GPU wird dagegen kaum ausgelastet. Interaktive Bildraten sind nur für relativ kleine Teilprojektionsanzahlen möglich.

7.4 Umsetzung auf Direct3D-10-GPUs

Die Struktur des Renderingvorgangs ist besser an die 3D-Grafikhardware angepasst, wenn statt der Bestimmung aller Primitive für eine Teilprojektion alle relevanten Teilprojektionen für ein Primitiv bestimmt werden. Wird jedes Primitiv entsprechend oft durch die GPU repliziert, können alle Primitive ohne Zustandsänderung und mit nur einem einzigen Renderingaufruf von der 3D-Grafikhardware verarbeitet werden. Die Teilprojektionen werden dazu als einfache Matrizen in einem Puffer gespeichert und im Shader ausgelesen. Die Clippingebenen lassen sich nicht aus einem Shader heraus verändern. Da das Clipping jedoch in einem beliebigen Koordinatensystem erfolgen kann, werden ein standardisierter Clippingraum mit festen Clippingebenen und eine teilprojektionsabhängige Transformation in diesen definiert.

Das zentrale Problem dieses Ansatzes ist die Replikation. Mit der Einführung von Geometryshadern ist eine stark begrenzte Erzeugung neuer Primitive auf der GPU möglich. Da die Anzahl der Teilprojektionen und damit die maximale Anzahl der Replikationen eines Primitivs diese Grenze bei weitem übersteigt, ist die direkte Verwendung der Geometryshader nicht möglich. Stattdessen muss ein ähnlicher Ansatz wie bei der dynamischen Geometrieverfeinerung (Kap. 6.3) verfolgt werden. Da für die stückweise perspektivischen Projektionen lediglich Replikationen erzeugt werden müssen, kann auf baryzentrische Verfeinerungsmuster verzichtet und der Algorithmus deutlich vereinfacht werden.

Der Renderingvorgang (Abb. 7.2) ähnelt dem für die dynamische Geometrieverfeinerung. Der zentrale Unterschied liegt in der Form des zwischengespeicherten Netzes. Während für die Verfeinerung generische baryzentrische Dreiecke erforderlich waren, brauchen jetzt nur noch Replikationen der Primitive gespeichert werden. Die Replikation lässt sich am effizientesten über eine Indizierung der Primitive umsetzen, da dann nicht mehr alle Punktattribute, sondern einzig der Index kopiert werden muss. Dadurch wird auf der 3D-Grafikhardware Speicherplatz,

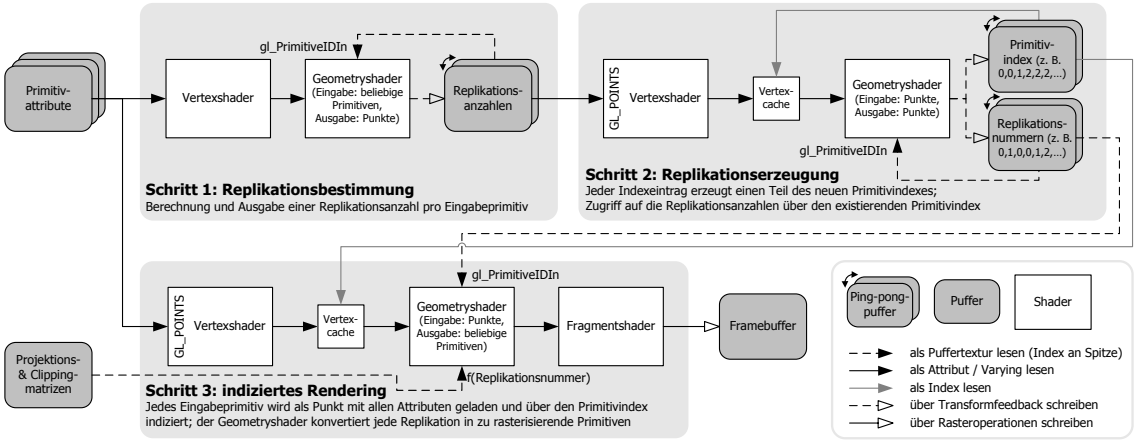


Abb. 7.2: Ablaufdiagramm der DirectX-10-Umsetzung der SPP. Der Algorithmus läuft analog der dynamischen Geometrieverfeinerung (Abb. 6.2) ab.

Bandbreite und Rechenaufwand gespart. Da die Replikationen beim Rendering unterscheidbar sein müssen – sie sollen in unterschiedlichen Teilprojektionen dargestellt werden – muss jeder Index um eine Replikationsnummer ergänzt werden. Diese Nummer nummeriert alle Wiederholungen eines Indexes fortlaufend durch. Für diesen Primitivindex gelten die gleichen Konsistenzbedingungen wie für das verfeinerte Netz in Kap. 6.3.

Das Rendering erfolgt analog Kap. 6.3 wieder in drei Schritten. Zuerst muss in einem Geometryshader die Anzahl der relevanten Teilprojektionen pro Primitiv berechnet bzw. geschätzt werden. Das Ergebnis wird mittels Transform Feedback in einen Pingpongpuffer geschrieben. Der zweite Schritt verwendet den existierenden Primitivindex samt der zugehörigen Replikationsnummern als Eingabe und erzeugt daraus einen neuen Primitivindex inklusive Replikationsnummern. Der GLSL-Programmausschnitt 7.2 zeigt den dazugehörigen Geometryshader. Der dritte Schritt übernimmt schließlich das Rasterisieren. Die Eingabe sind Primitivindex und Replikationsnummer, die als Attribute eines Punktes aufgefasst werden. In einem Vertexshader werden mit Hilfe des Indexes alle Attribute der eigentlichen Primitive aus Puffern geladen. Im nachfolgenden Geometryshader erfolgen dann die Transformation entsprechend der zu verwendenden Teilprojektion, die Transformation in den Clippingraum und die Konvertierung in den korrekten Primitivtyp. Der Pixelshader hat keine spezifische Aufgabe und kann jeden beliebigen Effekt implementieren.

Diese generische Umsetzung muss an zwei Stellen für eine konkrete nichtlineare Projektion angepasst werden:

1. In Schritt eins die müssen die relevanten Teilprojektionen ermittelt werden. Dabei ist eine konservative Schätzung ausreichend, da das Rendering eines Primitivs in irrelevante Teilprojektionen keine Bildfehler erzeugt. Der unnötige Aufwand mindert jedoch die Geschwindigkeit, weshalb gute Schätzungen zu bevorzugen sind.
2. In Schritt drei muss die Replikationsnummer eines Primitivs auf eine konkrete Teilprojektion abgebildet werden. Diese Abbildung muss mit lokalen Informationen in konstanter Zeit ausführbar sein, da sie in einem Shader abläuft. Es ist möglich, in Schritt eins erzeugte Informationen an Schritt drei weiterzuleiten, um die Abbildungsfunktion zu unterstützen.

```
#version 120
#extension GL_EXT_gpu_shader4 : enable
#extension GL_EXT_geometry_shader4 : enable

// x = number of subtriangles
// y = prev number of subtriangles
// z = triIdx
flat varying in ivec3 cInfo[]; // from pass 1

// repetition information about each triangle
flat varying out int newTriIdx;
flat varying out int newRepNo;

// repetition number per triangle
uniform isamplerBuffer repNoTex;

void emitTri(int mainID, int subID)
{
    // emit each vertex with updated status info
    newTriIdx = mainID;
    newRepNo = subID;
    EmitVertex();
    EndPrimitive();
}

void main()
{
    // get triangle repetition number
    int repNo = texelFetchBuffer(repNoTex, gl_PrimitiveIDIn).r;

    // kill all excess triangles
    if (repNo>=cInfo[0].x) return;

    // determine the number of outputs
    int prevRepCount = cInfo[0].y;
    int output = cInfo[0].x/prevRepCount + 1;
    int outputLimit = cInfo[0].x%prevRepCount;
    int base = repNo*output;
    if (repNo>=outputLimit) {
        base = outputLimit * output + (repNo-outputLimit)*(output-1);
        output -= 1;
    }

    // create each subtriangle
    for (int i = 0; i<output; ++i) {
        emitTri(cInfo[0].z, base+i);
    }
}
```

Progr. 7.2: GLSL-Code des Geometryshaders für Schritt 2 der Direct3D-10-Umsetzung. Der wesentliche Unterschied zu Progr. 6.1 liegt in der vereinfachten Funktion emitTri().

Die Anwendungsbeispiele in Kap. 7.6 zeigen typische Lösungsansatze für diese beiden Teilprobleme.

7.5 Umsetzung auf Direct3D-11-GPUs

Mit der Einführung der konfigurierbaren Verfeinerungseinheit in Direct3D 11 und den dazugehörigen GPUs gibt es eine zusätzliche Möglichkeit, neue Geometrie auf der 3D-Grafikhardware zu erzeugen. Mit dieser neuen Einheit lassen sich bis zu 8192 Dreiecke pro Eingabeprimitiv erzeugen. Sofern die Anzahl der Teilprojektionen diese Grenze nicht übersteigt, entfällt damit die Notwendigkeit zur Speicherung des Primitivindexes und der gesamte Renderingablauf aus Kap. 7.4 kann in einem Schritt abgearbeitet werden.

Grundidee für die neue Umsetzung ist die Übertragung aller Prozessschritte aus Abb. 7.2 auf die jetzt verfügbaren GPU-Renderingpipelineabschnitte. Die Verfeinerungseinheit ist auf die Erzeugung geschlossener Netze ausgelegt, was sich in der Geometrieverarbeitung durch die einzelnen Shaderstufen widerspiegelt. Der Hullshader wird pro Eingabeprimitiv (z. B. eine Bezierfläche) ausgeführt und liefert transformierte Kontrollpunkte sowie Verfeinerungsfaktoren. Die Verfeinerungseinheit erzeugt passend zu den Faktoren, aber unabhängig von konkreten Eingabeprimitivattributen, neue generische Punkte mit UV-Koordinaten im Intervall $[0;1] \times [0;1]$. Erst der Domainshader passt die generischen Punkte an das Primitiv an (z. B. durch Auswertung der Bezierfläche an der Stelle (u,v)) und wird somit ein Mal pro erzeugtem Punkt aufgerufen. Er ist somit ähnlich dem Vertexshader, da er nur auf Punkten ohne deren topologische Einbettung operiert. Erst für die Verarbeitung im Geometryshader werden die tatsächlichen verfeinerten Primitive auf Basis der vorgegebenen Topologie der Verfeinerung aus den Ausgaben des Domainshaders zusammengesetzt (Primitivassembler 2 in Abb. 2.5, OpenGL ARB, 2010).

Im vorliegenden Fall soll jedoch kein zusammenhängendes Netz sondern eine Menge unabhängiger Replikationen erzeugt werden. Die Unterscheidung der Replikationen ist dabei essentiell, eine Nummerierung der erzeugten Punkte oder Primitive analog der Eingabeprimitivkennung (`primitive_id`) gibt es allerdings nicht. Im Domainshader ist die Unterscheidung anhand der UV-Koordinaten einfach möglich, durch die nachfolgende Zusammensetzung ist eine eindeutige Zuordnung im Geometryshader jedoch nicht gewährleistet. Die Lösung ist die Verwendung des Punktmodus der Verfeinerungseinheit, da dann einzelne Punkte statt Dreiecken oder Linien an den Geometryshader geliefert werden. Es kommt also zu einer 1:1-Entsprechung zwischen Domainshaderaufrufen und Geometryshaderaufrufen.

Die exakte Topologie der Verfeinerung ist unter diesen Umständen nebensächlich. Wesentlich ist die einfache Abbildung der UV-Koordinaten auf eine fortlaufende Replikationsnummer sowie die einfache Bestimmung von Verfeinerungsfaktoren für eine gewünschte Anzahl von Replikationen. Dafür bietet sich die Isolinientopologie an, die durch 2 Verfeinerungsfaktoren in Längst- und Querrichtung beschrieben wird. Die erzeugten Punkte bilden ein regelmäßiges 2D-Gitter im Intervall $[0;1] \times [0;1]$. Die UV-Koordinaten lassen somit als 2D-Index auffassen und wie üblich in eine fortlaufende Nummer umrechnen.

Der gesamte Ablauf stellt sich folgendermaßen dar (Abb. 7.3): die Berechnung der Anzahl der benötigten Replikationen erfolgt pro Eingabedreieck im Hullshader, der das Ergebnis in Verfeinerungsfaktoren für die Verfeinerungseinheit konvertiert und an diese weitergibt. Sodann erzeugt die Verfeinerungseinheit die angefragte Anzahl Punkte. Der nachfolgende Domainshader bestimmt aus den UV-Koordinaten der erzeugten Punkte die jeweiligen Replikationsnummern

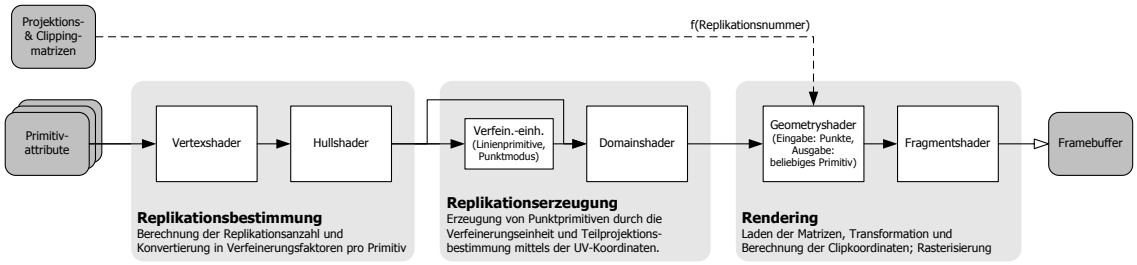


Abb. 7.3: Ablaufdiagramm der DirectX-11-Umsetzung der SPP. Die GPU-Verfeinerungseinheit erlaubt die Zusammenfassung aller drei Schritte der DirectX-10-Umsetzung (Abb. 7.2) zu einem Schritt.

und ergänzt gleichzeitig den erzeugten Punkt um alle Attribute der Eckpunkte des Eingabedreiecks. Der Geometryshader lädt über die Replikationsnummer die korrekten Projektions- und Clippingmatrizen. Nach der Transformation aller drei im Punkt enthaltenen Eckpunkte des Eingabedreiecks wird die so an die Teilprojektion angepasste Replikation als Dreieck an den Rasterisierer übergeben. Der Pixelshader hat wiederum keine spezielle Aufgabe.

Die Anpassung des generischen Ablaufs an konkrete nichtlineare Projektionen erfolgt analog der Direct3D-10-Variante durch eine Funktion zur Schätzung der Anzahl der benötigten Replikationen (im Hullshader ausgewertet) und eine Abbildung von Replikationsnummer auf Teilprojektion (im Geometryshader ausgewertet).

Im Vergleich zur geometryshaderbasierten Umsetzung hat dieser neue Ansatz mehrere Vorteile:

1. Primitive können vor der Replikation gecullt werden,
2. es werden keine zusätzlichen Puffer und Überlaufkontrollen benötigt und
3. es gibt keine Pingpongpuffer und somit keine Abhängigkeiten zwischen aufeinanderfolgenden Bildern.

Damit ist die Direct3D-11-Umsetzung deutlich besser für die Integration in bestehende 3D-Grafiksysteme und Mehr-GPU-Systeme geeignet. Nachteile ergeben sich aus den Beschränkungen der Verfeinerungseinheit. Besteht eine Projektionsapproximation aus mehr als 8192 Teilen, muss der Geometryshader mehr als ein Dreieck ausgeben, um das Gesamtschema beibehalten zu können. Außerdem ist es möglich, dass einzelne Replikationszahlen nicht exakt als Kombination von Verfeinerungsfaktoren erzeugt werden können. In dem Fall müssen zusätzliche Replikationen in Kauf genommen werden, die zwar keine Bildfehler aber unnötigen Aufwand verursachen.

7.6 Anwendungsbeispiele

Im Folgenden werden zwei nichtlineare Projektionen und deren Implementierung als stückweise perspektivische Projektionen beschrieben. Die Projektionen wurden wegen ihres beispielhaften Charakters gewählt: Die Zylinderprojektion ist eine Projektion mit geschlossener analytischer Form während texturbasierte Bildverzerrung auf einer frei manipulierbaren Beschreibung aufbaut. Beide Projektionen besitzen jeweils genau ein punktförmiges Projektionszentrum bei geraden Projektionsstrahlen und sind damit für die stückweise perspektivische Approximation geeignet.

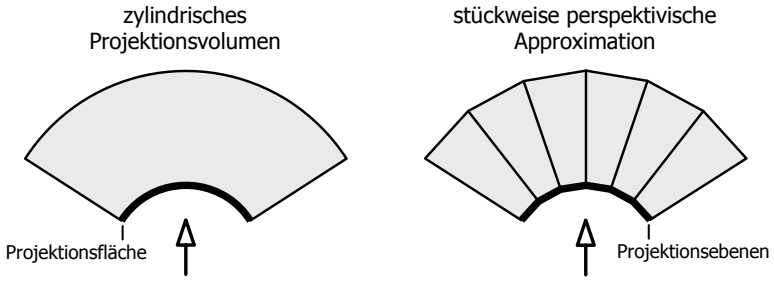


Abb. 7.4: Stückweise perspektivische Approximation einer Zylinderprojektion (Draufsicht). Das zylindrische Projektionsvolumen wird fächerförmig in Pyramidenstümpfe zerlegt.

7.6.1 Zylinderprojektion

Die nachfolgenden Betrachtungen beziehen sich auf einen vertikalen Zylinder als Projektionsfläche. Projektionen auf horizontale Zylinder erfolgen analog. Ähnlich wie in Kap. 6.4 ist auch hier relevant, dass lediglich die horizontale Komponente der Projektion Nichtlinearitäten hervorruft. Es genügt also, das Projektionsvolumen in der Horizontalen zu zerlegen, um eine gute stückweise perspektivische Approximation zu erreichen. Bildlich wird dazu das Projektionsvolumen durch einen Fächer gleich schmaler Pyramidenstümpfe angenähert. Jeder Pyramidenstumpf ist das Projektionsvolumen einer perspektivischen Projektion. Abb. 7.4 illustriert dieses Prinzip.

Soll eine Zylinderprojektion mit dem horizontalen Öffnungwinkel ϕ_c durch n perspektivische Teilprojektionen approximiert werden, lässt sich die i-te Teilprojektion mit $i \in [0; n-1]$ als Verkettung mehrerer Transformationen in einer Matrix $M(i)$ darstellen:

$$M(i) = M_{tx}(i) \cdot M_{sx} \cdot M_P \cdot M_{ry}(i) \tag{7.1}$$

$M_{ry}(i)$ rotiert die Mittelachse der Teilprojektion um die y-Achse auf die negative z-Achse. Der Rotationswinkel $\alpha(i)$ ergibt sich aus:

$$\alpha(i) = \phi_c / n \cdot \big(i - (n-1)/2\big) \tag{7.2}$$

M_P ist die Standardmatrix für eine perspektivische Projektion mit dem horizontalen Öffnungswinkel $\phi_p = \phi_c / n$. Vertikaler Öffnungswinkel sowie vordere und hintere Clippingebene werden von der Zylinderprojektion übernommen.

M_{sx} skaliert das aus M_P resultierende kanonische Sichtvolumen in der Horizontalen mit dem Faktor $1/n$ auf die eigentliche Breite der Teilprojektion im Bild.

$M_{tx}(i)$ bewegt das Abbild der Teilprojektion durch horizontale Verschiebung an den korrekten Platz im Bild. Der Verschiebungsvektor $d(i)$ ergibt sich aus

$$d(i) = \Big(1/n \cdot \big(i - (n-1)/2\big), 0, 0\Big)^T \tag{7.3}$$

Die Teilprojektionen sind durch diese Konstruktion im Bild von links nach rechts durchnummeriert.

Für die klassische Umsetzung müssen nun n Kameras erzeugt werden, die als Orientierungsmatrix die der Zylinderprojektion übernehmen und als Projektionsmatrix die jeweilige Matrix $M(i)$. Zusätzlich werden pro Kamera zwei benutzerdefinierte Clippingebenen benötigt, die

jeweils die rechte und linke Begrenzung der Teilprojektion sicherstellen. Die Begrenzung nach oben, unten, vorn und hinten wird implizit durch das kanonische Sichtvolumen sichergestellt.

Für die Umsetzung auf Direct3D-10- und -11-GPUs ist zusätzlich die Definition eines Standardclippingraums und eine teilprojektionsabhängige Transformation in diesen notwendig. Da die GPU kein Clipping in homogenen Koordinaten unterstützt, wird das Standardsichtvolumen der perspektivischen Projektion – eine Pyramide mit 90° Öffnungswinkel – verwendet. Die Transformation $M_{\text{clip}}(i)$ kann direkt aus der Projektionsmatrix abgeleitet werden:

$$M_{\text{clip}}(i) = M_{\text{s}}(M_{\text{P11}}, M_{\text{P22}}, 1) \cdot M_{\text{ry}}(i) \tag{7.4}$$

M_{s} ist dabei eine Skalierungsmatrix, die die Skalierungsfaktoren aus der Diagonalen der Projektionsmatrix M_{P} bezieht. Die dazugehörigen Clippingebenen haben die Normalen $n_1 = (-1,0,-1)^T$, $n_2 = (1,0,-1)^T$, $n_3 = (0,-1,-1)^T$ und $n_4 = (0,1,-1)^T$ und schneiden den Ursprung.

Nachdem alle Transformationen bekannt sind, muss das generische Verfahren and die konkrete Projektion angepasst werden. Wie in Kap. 7.4 beschrieben, muss dazu für ein Primitiv die Anzahl der relevanten Teilprojektionen bestimmt werden und die Replikationsnummer auf eine konkrete Teilprojektion abgebildet werden.

Die Anzahl der relevanten Teilprojektionen lässt sich über die horizontale Ausdehnung eines Primitivs nach oben begrenzen. Im Allgemeinen überdeckt ein sichtbares Primitiv eine zusammenhängende Folge von Teilprojektionen. Da bei der Zylinderprojektion nur die Eckpunkte horizontale Extrempunkte der Abbildung eines Primitivs bilden können, genügt zur Bestimmung der relevanten Teilprojektionen die exakte Projektion der Eckpunkte. Ein Spezialfall ergibt sich, falls die Projektionszylinderachse das Primitiv schneidet. In dem Fall ist es potentiell in allen Teilprojektionen sichtbar und die Anzahl der relevanten Teilprojektionen ist n. Hat die Zylinderprojektion einen Öffnungswinkel größer als 180°, kann es zu einem weiteren Spezialfall kommen, bei dem ein hinter der Kamera befindliches Primitiv auf beiden Seiten des Bildes sichtbar ist. Zur Unterstützung der Replikationsnummernabbildung speichert der erste Schritt zusätzlich zur Anzahl der relevanten Teilprojektionen c auch den Index der ersten relevanten Teilprojektion i_{s}.

Die Replikationsnummernabbildung lässt sich nun einfach als Summe darstellen. Der Index i der konkreten Teilprojektion ergibt sich aus der Replikationsnummer r durch:

$$i = (i_{\text{s}} + r) \bmod n \tag{7.5}$$

Mit der modulo-Operation wird der zweite Spezialfall abgedeckt.

Abb. 7.5 zeigt ein Beispielbild mit hervorgehobenen Primitivkanten (weiß) und Teilprojektionsgrenzen (schwarz). Zur Verdeutlichung wurden für diese 360°-Zylinderprojektion nur 32 Teilprojektionen mit jeweils 50 Pixeln Breite verwendet. In Versuchen wurde eine Breite von 10-20 Pixeln als gute Näherung für die Zylinderprojektion gefunden, bei der die gemessene durchschnittliche Replikationsrate bei ca. 2 und die gemessene maximale Replikationsrate bei n liegt.

7.6.2 Texturbasierte Bildverzerrung

Bildverzerrung (Trapp und Döllner, 2008) verwendet ein oder mehrere perspektivische Darstellungen (z. B. zusammengefasst in einer Cubemap), die sie für das finale Bild verzerrt. Die

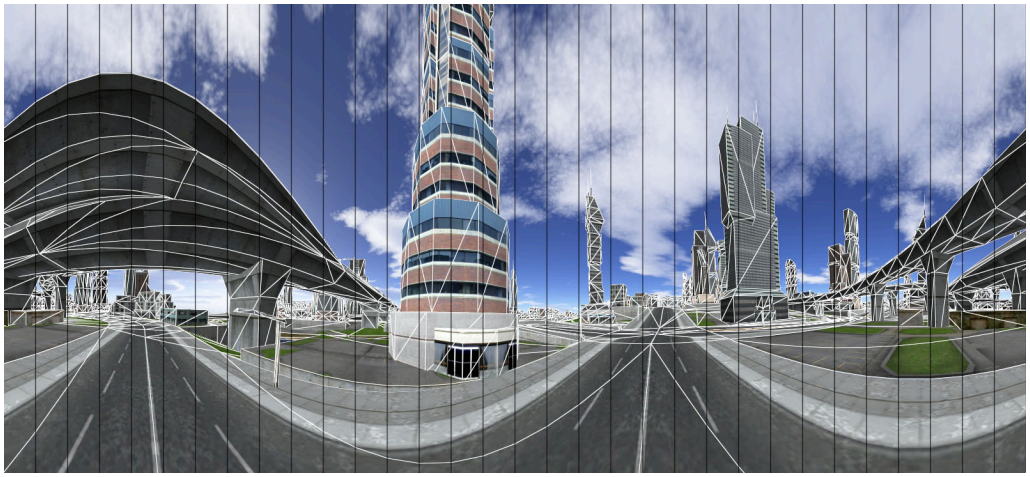


Abb. 7.5: Beispiel einer stückweise perspektivisch approximierten 360°-Zylinderprojektion. Die perspektivischen Teilprojektionen sind durch schwarze Linien gekennzeichnet. Weiße Linien kennzeichnen das Dreiecksnetz. Zur Verdeutlichung beträgt die Teilprojektionsbreite 50 Pixel.

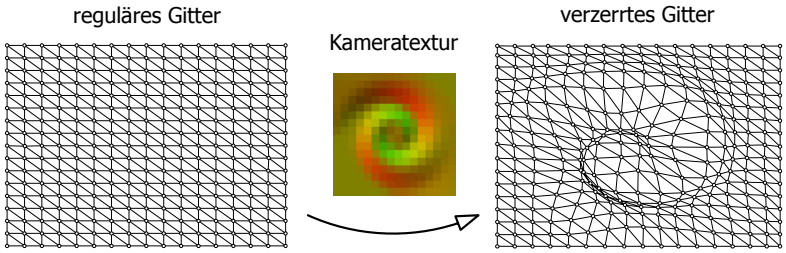


Abb. 7.6: Bildverzerrung mittels einer Kameratextur. Die Kameratextur kodiert Verschiebungsvektoren, die auf die Punkte eines regulären Gitters angewendet werden.

Verzerrung lässt sich entweder analytisch beschreiben, z. B. eine paraboloidische Abbildung, oder frei definieren, z. B. mittels einer Textur (Spindler u. a., 2006). In der Umsetzung verwenden beide Varianten ein reguläres, rechtwinkliges Gitter in der perspektivischen Darstellung, dessen Gitterpunkte für das Ergebnisbild verschoben werden.

Im Folgenden wird eine Implementierung der Kameratexturtechnik (Spindler u. a., 2006) als stückweise perspektivische Projektion beschrieben. Bei dieser Technik wird jedem Gitterpunkt ein beliebiger Verschiebungsvektor zugeordnet. Alle Verschiebungsvektoren werden in einer Textur gespeichert (Abb. 7.6). Spindler u. a. (2006) verwenden eine punktbasierte Umsetzung, bei der ein Punkt zuerst perspektivisch projiziert wird, mit dieser 2D-Position die Kameratextur ausgelesen wird und der resultierende Verschiebungsvektor auf die 2D-Position addiert wird. Damit unterliegt das Verfahren allen in Kap. 2.3.3 beschriebenen typischen Problemen.

Für die Formulierung als stückweise perspektivische Projektion muss der Übergang vom regulären zum verzerrten Gitter als stückweise lineare Transformation beschrieben werden. Jede Gitterzelle muss dazu in zwei dreieckige Zellen zerlegt werden. Zwar lässt sich die Transformation eines Rechtecks in ein beliebiges 2D-Viereck als projektive Transformation beschreiben, allerdings kann damit keine Bruchfreiheit zwischen benachbarten Gitterzellen sichergestellt werden. Für diese Eigenschaft ist eine bilineare Abbildung vonnöten (Heckbert, 1989), die nicht vom Rasterisierer unterstützt wird. Die Abbildung eines 2D-Dreiecks auf ein anderes ist eine

affine und damit lineare Abbildung, die auch die Bruchfreiheit garantiert. Somit lässt sich für jede dreieckige Zelle $i = (x, y)$ eine Projektionsmatrix $M(i)$ definieren:

$$M(i) = M_{\mathrm{A}}(i) \cdot M_{\mathrm{P}} \tag{7.6}$$

M_{P} bezeichnet die Projektionsmatrix der unverzerrten perspektivischen Abbildung und $M_{\mathrm{A}}(i)$ die affine Abbildung vom unverzerrten zum verzerrten Gitter. Die Berechnungsvorschrift von $M_{\mathrm{A}}(i)$ aus Verschiebungsvektoren ist in Heckbert (1989) zu finden.

Für die klassische Umsetzung muss für jede dreieckige Zelle eine Kamera mit der Originalorientierungsmatrix und der jeweiligen Projektionsmatrix $M(i)$ definiert werden. Zusätzlich werden drei benutzerdefinierte Clippingebenen benötigt, die die Zellgrenzen im verzerrten Bild definieren. Einzig die vordere und hintere Clippingebene wird implizit von M_{P} übernommen.

Die Umsetzung für Direct3D-10-GPUs erfordert wieder die Definition eines Standardclippingraums. Das Clipping erfolgt im unverzerrten regulären Gitter, indem das Sichtvolumen jeder rechteckigen Zelle in ein kanonisches Sichtvolumen transformiert wird. Ein Teildreieck dieser Zelle verwendet diese Transformation direkt, für das zweite Teildreieck wird eine zusätzliche 180°-Drehung um die z-Achse durchgeführt. Die dazugehörigen Clippingebenen haben die Normalen $n_1 = (-1, 0, -1)^T$, $n_2 = (0, -1, -1)^T$ und $n_3 = (1, 1, -1)^T$ und schneiden den Ursprung.

Bei der Anpassung des generischen Algorithmus werden zunächst nur reguläre Gitterzellen, also Dreieckspaare, betrachtet. Für die Bestimmung relevanter Teilprojektionen pro Primitiv wird ein einfacher Boundingbox-Test im unverzerrten Gitter verwendet. Alle von der Boundingbox überdeckten regulären Zellen werden als relevant eingestuft. Zur Unterstützung der Replikationsnummernabbildung werden die linke untere Zelle c_{ll} der Boundingbox sowie deren Breite w in Gittereinheiten gespeichert. Der Replikationsschritt erzeugt sodann pro relevanter, regulärer Gitterzelle eine Replikation.

Die Abbildung der Replikationsnummern erfolgt durch Umwandlung der Replikationsnummer r in einen 2D-Index. Der Index i der regulären Gitterzelle ergibt sich als:

$$i = c_{\mathrm{ll}} + (r \bmod w, \lfloor r / w \rfloor) \tag{7.7}$$

Erst im Geometryshader des dritten Schrittes wird pro Replikation das Primitiv in beide Teildreiecke der jeweiligen regulären Zelle ausgegeben. Die Projektions- und Clippingmatrizen werden dazu mit Hilfe des Index i aus Texturen geladen. Da der Relevanztest sehr konservativ ist – im Schnitt sind nur die Hälfte aller detektierten Zellen tatsächlich relevant – wird im Geometryshader zusätzlich Culling verwendet.

Die Umsetzung für Direct3D-11-GPUs kann auf der eben vorgestellten Konstruktion aufbauen und sie vereinfachen. Der Hullshader bestimmt die Anzahl der relevanten Zellen auch über die Boundingbox-Methode. Die Breite und Höhe in Zelleneinheiten übermittelt er als Verfeinerungsfaktoren an die Verfeinerungseinheit. Der Domainshader kann dann aus den UV-Koordinaten der neu erzeugten Punkte den 2D-Index der regulären Gitterzelle durch Skalierung mit den Boundingbox-Dimensionen und Addition von c_{ll} bestimmen. Ein Umweg über die lineare Replikationsnummer ist nicht nötig. Auch hier müssen zwei Primitive pro Geometryshaderaufruf ausgegeben werden.

Abb. 7.7 zeigt ein Beispielbild mit hervorgehobenen Primitivkanten (weiß) und Teilprojektionsgrenzen (schwarz). Darin wird eine Kameratextur mit Auflösung 64x64 ähnlich der Abb. 7.6 verwendet. In dieser Implementierung lässt sich der Verzerrungseffekt auch animieren, da dies

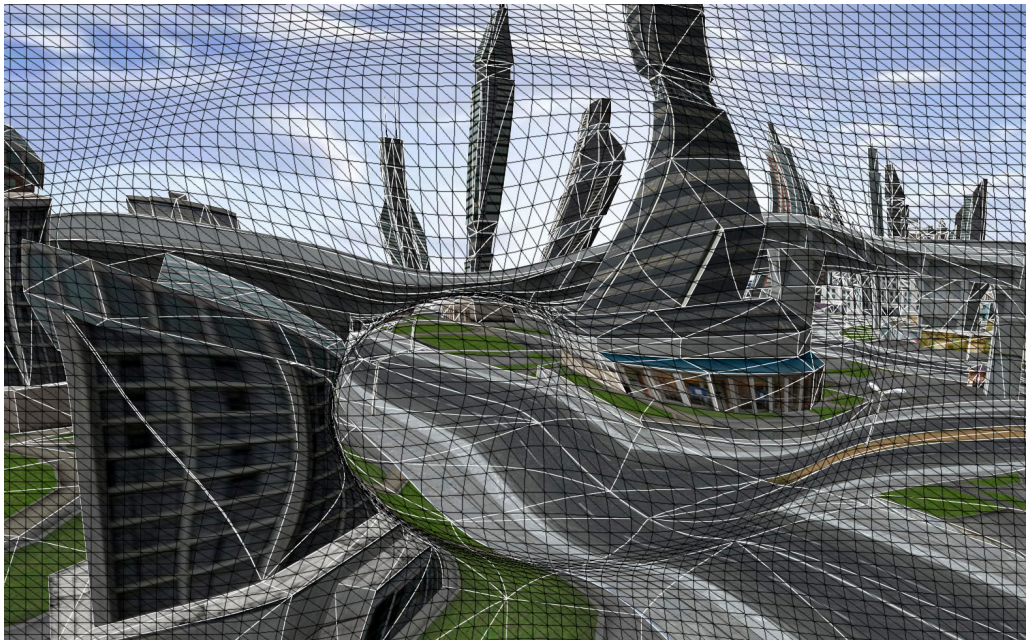


Abb. 7.7: Beispiel einer stückweise perspektivisch approximierten texturbasierten Bildverzerrung. Das verwendete Gitter für die spiralförmige Verzerrung besteht aus 64×64 Punkten. Die resultierenden 7938 perspektivischen Teilprojektionen sind durch schwarze Linien gekennzeichnet. Weiße Linien kennzeichnen das Dreiecksnetz.

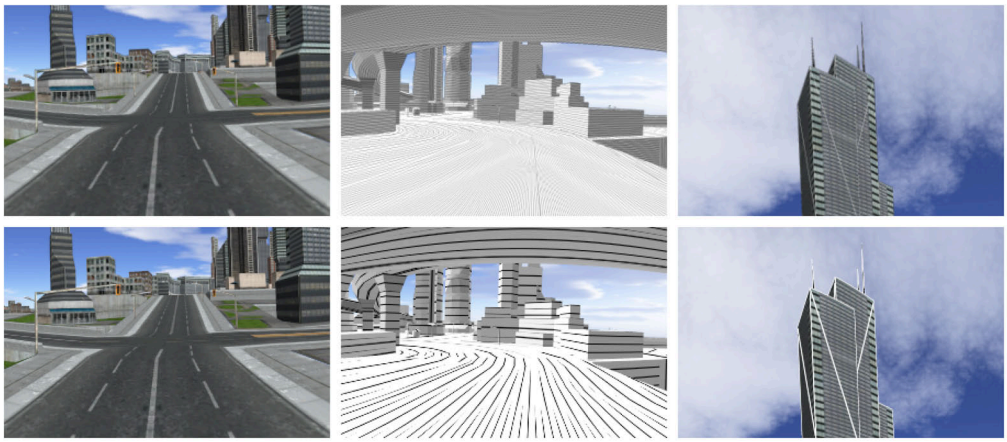
lediglich eine bildweise Aktualisierung der Matrizen erfordert. Solche Animationen lassen sich beispielsweise für Bildlinsen verwenden, die festgelegten Objekten wie Landmarken folgen.

7.7 Ergebnisse

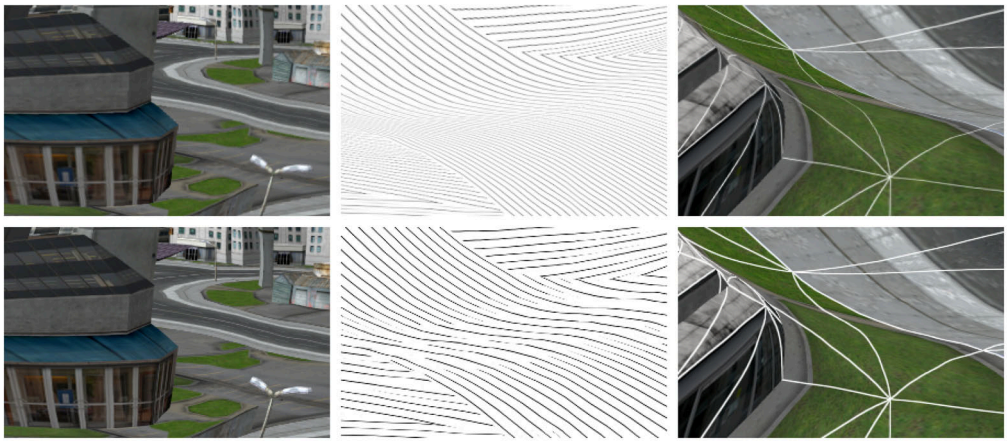
Die Tests der vorgestellten Techniken gliedern sich in zwei Abschnitte. Im ersten Abschnitt wurde die Direct3D-10-Umsetzung der stückweise perspektivischen Projektionen (SPP) mit einer bildbasierten Umsetzung (BB) für beide beschriebenen Anwendungsfälle verglichen. Hauptaugenmerk lag dabei auf der erzielten Bildqualität. Im zweiten Abschnitt wurden die drei Implementierungsvarianten untereinander in Bezug auf die Renderinggeschwindigkeit verglichen.

Der erste Testabschnitt verwendete als Umgebung einen PC mit AMD Athlon 64 X2 4400+ Prozessor, 2GB RAM und einer nVidia GeForce 8800GTS Grafikkarte mit 640MB RAM. Die Bildgröße war auf 1600×1200 eingestellt. Die Messungen nutzten einen Kamerapfad durch das in den Abb. 7.5 und 7.7 dargestellte texturierte virtuelle 3D-Stadtmodell. Dieses Modell besteht aus ca. 35.000 Dreiecken in 14 Gruppen.

Für den Test der Bildqualität wurden drei verschiedene Renderingeffekte verwendet: normales Texturemapping, ein Schraffureffekt (Freudenberg u. a., 2001) und ein Kantenhervorhebungseffekt (Bærentzen u. a., 2006). Das Wesen des Schraffureffekts ist der konstante Abstand der einzelnen Striche im Bildraum. Dies wird durch Texturierung mit einer Strichtextur erreicht, die eine speziell generierte Mipmappyramide besitzt. Der Kantenhervorhebungseffekt zeichnet sich durch eine entfernungsabhängige Kantenbreite aus. Die Breite wird mit Hilfe der Gradientenoperationen im Fragmentshader gesteuert. Abb. 7.8 zeigt einen direkten Vergleich von SPP



(a) Zylinderprojektion (vergrößerter Ausschnitt aus Abb. 7.5).



(b) Texturbasierte Bildverzerrung (vergrößerter Ausschnitt aus Abb. 7.7).

Abb. 7.8: Qualitätsvergleich von BB und SPP für normales Texturemapping (links), einen Schraffureffekt (Freudenberg u. a., 2001, mitte) und einen Kantenhervorhebungseffekt (Bærentzen u. a., 2006, rechts). BB ist jeweils oben dagestellt, SPP jeweils unten. SPP erhält das Wesen der Renderingeffekte unter den nichtlinearen Projektionen.

und BB für alle drei Effekte mit aktiviertem Antialiasing und anisotroper Texturfilterung. Der Fragmentshader ist für die jeweiligen Vergleichsbilder identisch. Bei der bloßen Texturierung ist kein Qualitätsunterschied zu sehen, da die verwendeten Texturen durch die zusätzliche Verzerrung der BB-Variante kaum gestört werden. Schraffur und Kantenhervorhebungseffekt weisen jedoch deutliche Unterschiede auf. Der Strichabstand bzw. die Kantenstärke variieren in der BB-Variante deutlich, während in der SPP-Variante das Wesen trotz Verzerrungen exakt erhalten bleibt. Der deutliche Unterschied erklärt sich durch die in den Effekten verwendeten bildabhängigen Operationen (Mipmapping bzw. Gradienten). Bei der BB-Variante werden diese Operationen auf das unverzerrte Zwischenbild angewendet und nachträglich verzerrt. Bei der SPP-Variante werden sie direkt für das verzerrte Zielbild – und somit korrekt – ausgewertet.

Der Qualitätsgewinn muss natürlich in Relation zur Geschwindigkeit gesetzt werden. Hierzu wurde die Bildwiederholrate (Bilder (Hz)), die mittlere Anzahl der gerenderten Dreiecke (⌀

Variante	*Bildwiederholrate (Bilder pro Sek.)*							
	360° Zylinderprojektion (160 Teilp.)				*texturbas. Bildverzerrung (9.322 Teilp.)*			
	Bilder (Hz)	*⌀ D.-menge*	*⌀ Repl.-rate*	*Puffer-gr. (kB)*	*Bilder (Hz)*	*⌀ D.-menge*	*⌀ Repl.-rate*	*Puffer-gr. (kB)*
BB	41,7	21.151	0,61	1.081	206,2	34.596	1	1.081
SPP	84,7	67.675	1,96	2.672	22,1	351.954	10,17	7.661
BB HQ	33,8	21.151	0,61	1.081	95,8	34.596	1	1.081
SPP HQ	54,8	67.675	1,96	2.672	20,9	351.954	10,17	7.661

Tab. 7.1: Geschwindigkeitsvergleich zwischen bildbasierten Verzerrungen (BB) und stückweise perspektivischen Projektionen (SPP). Die Tabelle zeigt die Bildwiederholrate (Bilder (Hz)), die durchschnittliche Dreiecksmenge (⌀ D.-menge), die die durchschnittliche Replikationsrate (⌀ Repl.-rate) und die notwendige Punktpuffergröße (Puffergr. (kB)). SPP erreicht bei der Zylinderprojektion höhere Bildwiederholraten als BB, bei der texturbasierten Bildverzerrung jedoch nicht.

D.-menge), die mittlere Replikationsrate (⌀ Repl.-rate) und die Gesamtgröße der verwendeten Punktpuffer (Puffergr. (kB)) gemessen. Zusätzlich wurde der Einfluss hoher Qualitätseinstellungen, d. h. aktiviertem 16-fachem Multisample-Antialiasing und 16-facher anisotroper Texturfilterung, untersucht (HQ). Der Vergleich erfolgt zwischen der Direct3D-10-Umsetzung der SPP und einer auf aktuellen Techniken basierenden Implementierung der BB-Variante, die beide mit OpenGL 2.0 und entsprechenden Erweiterungen implementiert wurden. Die Messungen sind in Tab. 7.1 aufgelistet.

Die bildbasierte Implementierung der 360°-Zylinderprojektion verwendet eine 6×2048×2048 Pixel große Cubemap, die für jedes Bild in einem einzelnen Pass mittels Layered Rendering aktualisiert wird (Persson, 2007). Dabei wird im Geometryshader jedes Dreieck für jede relevante Seite der Cubemap repliziert. Die Relevanz wird mittels Frustum- und Backfaceculling bestimmt, was das Replikationsverhältnis von weniger als 1 erklärt. SPP teilt die Projektion in Streifen von 10 Pixeln Breite, d. h. 160 Teilprojektionen, ein. Im Schnitt ist jedes Dreieck dabei nur in zwei Teilprojektionen sichtbar. Der erhöhte Speicherbedarf ergibt sich aus der Notwendigkeit der Zwischenspeicherung der Replikationsinformationen, was 16 Byte pro Replikation erfordert. In der Direct3D-11-Variante ist diese Zwischenspeicherung nicht erforderlich. Insgesamt sind für die Zylinderprojektion mit SPP trotz höherer Qualität größere Bildwiederholraten möglich als mit BB. Hauptgrund ist der enorme Aufwand für die Cubemapaktualisierung und die notwendige Mipmapstufenerzeugung bei BB. Auch wenn die Größe der Cubemap und damit die Qualität reduziert werden, bleibt SPP schneller.

Bei der texturbasierten Bildverzerrung kehren sich die Verhältnisse um. Die BB-Variante wird deutlich schneller, da nur noch eine 2D-Textur statt der Cubemap aktualisiert werden muss. Andererseits muss SPP deutlich kleinere Teilprojektionen für eine gute Approximation der Verzerrung verwenden. Da jedes Dreieck nun im Schnitt 10 Teilprojektionen überdeckt, steigt der Replikations- und Berechnungsaufwand deutlich an und die Bildwiederholrate sinkt. Die Replikation (Schritte 1 und 2) haben am Gesamtaufwand nur einen Anteil von weniger als 10%. Der Hauptaufwand entsteht demnach in Schritt 3 bei der Transformation und Rasterisierung der replizierten Dreiecke. Eine Verbesserung ließe sich durch eine bessere Schätzung als den

Umsetzung	*Bildwiederholrate (Bilder pro Sek.)*					
	Kameratexturauflösung 16×16			*Kameratexturauflösung 80×60*		
	640×480	*1600×1200*	*MSAA + AI*	*640×480*	*1600×1200*	*MSAA + AI*
klassisch	27,56	26,1	26,1	2,26	2,26	2,26
Direct3D 10	293,7	275,5	244,7	71,3	69,7	67,7
Direct3D 11	455	412	303,5	82,8	80,9	78,1

Tab. 7.2: Geschwindigkeitsvergleich der verschiedenen SPP-Umsetzungen für die Auflösungen 640×480 und 1600×1200. MSAA + AI verwendet bei einer Auflösung von 1600×1200 zusätzlich 8-faches Multisample-Antialiasing und 16-fache anisotrope Texturfilterung. Alle angegebenen Werte in Bildern pro Sekunde.

Boundingbox-Test erzielen. Zusätzlich sollte die Auflösung der Verzerrungstextur an die Bildauflösung angepasst werden. Meist genügt eine mittlere Größe von 20×20 Pixeln pro Teilprojektion. In unserem Test haben wir entsprechend eine 80×60 Pixel große Verzerrungstextur verwendet. Bei einer 128×128 Pixel großen Textur steigt das Replikationsverhältnis von 10 auf 21, d. h. ca. 750.000 Replikationen pro Bild, und die Bildwiederholrate sinkt auf 8,2 Bilder pro Sekunde.

Der zweite Testabschnitt vergleicht die Geschwindigkeit der verschiedenen SPP-Varianten. Dazu wurde ein PC mit Intel Xeon W3520 Prozessor (2,66 GHz), 6 GB RAM und einer AMD Radeon HD5850 Grafikkarte mit 1 GB Grafikspeicher verwendet. Das Betriebssystem war Windows 7 64 Bit, während alle Implementierungen eine 32-Bit-Umgebung mit Direct3D 11 als Grafik-API verwendeten. Das Testmodell und der Kamerapfad blieben gegenüber dem ersten Testabschnitt unverändert, die Anwendung ist die texturbasierte Bildverzerrung.

Die drei Umsetzungen wurden wie beschrieben implementiert, außer dass die klassische Umsetzung auch den Standardclippingraum statt variabler Clippingebenen verwendet. Diese Anpassung ist erforderlich, da Direct3D 11 keine expliziten Clippingebenen mehr unterstützt. View-Frustum-Culling wurde mittels einer kd-Baum-basierten Begrenzungsvolumenhierarchie implementiert. Es wird sowohl von der klassischen als auch von der Direct3D-11-Umsetzung verwendet. Die Direct3D-10-Umsetzung kann aus den beschriebenen Gründen kein applikationsseitiges Culling verwenden. Tab. 7.2 listet die Messergebnisse für verschiedene Rahmenbedingungen auf. Die von den einzelnen Umsetzungen erzeugten Bilder sind dabei jeweils identisch.

Die Ergebnisse decken sich klar mit den Erwartungen, wonach die Direct3D-11-Umsetzung vor Direct3D-10- und klassischer Umsetzung die höchste Bildwiederholrate liefert. Interessant ist der vergleichsweise geringe Unterschied zwischen Direct3D-11- und Direct3D-10-Umsetzung, insbesondere da die Verfeinerungseinheit im Gegensatz zum Geometryshader für Anwendungsfälle wie den Vorliegenden geschaffen wurde und die Umsetzung deutlich einfacher ist. Eine mögliche Antwort liegt im unterschiedlichen Reifegrad der Hardware. Die verwendete 3D-Grafikhardware ist bereits die dritte Generation mit Geometryshadern, jedoch die erste mit Verfeinerungseinheit. Der zweite, wesentlichere Grund liegt in dem recht kleinen Anteil der eigentlichen Replikation am Gesamtaufwand. Wie beschrieben, hat die Transformation und Rasterisierung der replizierten Dreiecke den Hauptanteil an der Bilderzeugung. Da beide Umsetzungen diesen Teil im Wesentlichen identisch abarbeiten, kann auch eine dramatische Beschleunigung der Replikation nach Amdahls Gesetz (Amdahl, 1967) nur zu einer geringen

Beschleunigung des Gesamtvorgangs führen. Die klassische Umsetzung kann in keiner Weise mit den anderen beiden Umsetzungen konkurrieren. Sie ließe sich durch Multithreading und verbessertes Culling, das z. B. Kohärenzeffekte benachbarter Teilprojektionen ausnutzt oder andere Datenstrukturen einsetzt, optimieren. Eine Verbesserung um Faktor 30, um zur Direct3D-10-Umsetzung aufzuschließen, ist jedoch unrealistisch.

7.8 Diskussion

Die SPP behebt die qualitativen Probleme (z. B. Z-Fighting) der dynamischen Geometrieverfeinerung (Kap. 6), indem das Bild ausschließlich aus perspektivischen Projektionen aufgebaut wird. Damit funktionieren ausnahmslos alle GPU-Merkmale mit optimaler Qualität. Als Konsequenz können alle für perspektivische Projektionen entworfenen Effekte direkt und ohne Änderung für die nichtlineare Projektion übernommen werden, ohne dass sich ihr Charakter ändert.

Der zentrale Nachteil der SPP ist die durch die Approximation bedingte Einschränkung der möglichen nichtlinearen Projektionen. Die Projektion sollte durch zusammenhängende, nicht überlappende perspektivische Projektionen approximiert werden. Dies ist zum Beispiel für Projektionen mit geraden Projektionsstrahlen, die sich in genau einem punktförmigen Projektionszentrum treffen und ansonsten schnittfrei sind, möglich. Ein Grenzfall ist die Projektion für omnidirektionale Stereopanoramen (Peleg u. a., 2001), die eng mit der Zylinderprojektion verwandt ist, jedoch kein punkt- sondern ein kreisförmiges Projektionszentrum besitzt. Für diese lässt sich keine bruchfreie Approximation spezifizieren, was allerdings nur für sehr nahe Objekte zu sichtbaren Bildfehlern führt. Ein Gegenbeispiel ist die Projektion durch eine Linienkamera (Pushbroom-Projektion; Lorenz und Döllner, 2006), für die es keine geeignete SPP-Approximation gibt.

Ein wenig praktikabler Extremfall ist die Reduktion jeder Teilprojektion auf ein einzelnes Pixel. Bei einer solchen Konstruktion fallen Überlappungen in der Approximation nicht mehr ins Gewicht, da pro Teilprojektion nur ein einziger Strahl ausgewertet wird. SPP nähert sich damit dem Raycasting an, wobei die Effizienz der GPU jedoch dramatisch sinkt, da deren massive Parallelität bei den dann erzeugten 1-Pixel-Dreiecken nur beschränkt ausgenutzt werden kann (Crassin, 2008).

Die im Rahmen der SPP entwickelten technischen Umsetzungen sind auch unabhängig von nichtlinearen Projektionen universell einsetzbar. Die Replikation mittels Geometryshader (Kap. 7.4) oder Verfeinerungseinheit (Kap. 7.5) lässt sich für primitivscharfe dynamische Geometrieinstanziierung verwenden. GPUs unterstützen derzeit nur die Instanziierung ganzer Geometrienetze mit vorgegebener Instanzenzahl. Mit der beschriebenen Replikation ließen sich Teile eines Netzes von der GPU gesteuert unterschiedlich oft instanziieren, was z. B. für ein LoD-Verfahren beim Rendering von Charaktergruppen interessant wäre.

8 Zusammenfassung und Ausblick

8.1 Zusammenfassung

Die vorliegende Arbeit stellt Verfahren und Konzepte zur Texturierung und Visualisierung virtueller 3D-Stadtmodelle vor. Im Bereich der Texturierung wird ein automatisches Verfahren zur Ableitung von Gebäudetexturen aus Schrägluftbildern entwickelt (Kap. 3). Dieses Verfahren zeichnet sich durch eine texelpräzise Auswahl der Quellbilder für eine Textur aus, wobei Verdeckungen und Auflösungsunterschiede berücksichtigt werden. Das Verfahren ist nicht auf die geometrische Erfassung virtueller 3D-Stadtmodelle ausgelegt, sondern auf deren „Veredelung“ durch fotorealistischen Texturen. Es konzentriert sich deshalb auf die Verarbeitung heterogener Datensätze. Zusätzlich wurden Probleme des praktischen Einsatzes berücksichtigt und Möglichkeiten zur manuellen Korrektur von Ungenauigkeiten der Schrägluftbildpositionierung, automatischen Korrektur von Farbabweichungen und manuellen Nachbearbeitung der erzeugten Texturen geschaffen. Das Verfahren ist in einen praktisch einsetzbaren Arbeitsfluss integriert, der auch größte virtuelle 3D-Stadtmodelle bewältigt, und wird von verschiedenen Firmen aktiv zur Veredelung virtueller 3D-Stadtmodelle genutzt. Ein mit Hilfe des vorgestellten Verfahrens texturiertes virtuelles 3D-Stadtmodell ist das offizielle Berliner 3D-Stadtmodell mit ca. 475.000 Gebäuden auf ca. 800 km^2 Fläche.

Der zweite betrachtete Aspekt der Texturierung virtueller 3D-Stadtmodelle ist die Speicherung oberflächengebundener Daten (Kap. 4). Zur Beschreibung der Daten werden modifizierte Konzepte aus der Computergrafik verwendet. Das abgeleitete Datenmodell richtet sich dabei an praktischen Anwendungsfällen aus, um eine einfache und breite Verwendbarkeit ohne unnötige technische Komplexität sicherzustellen. Es ist in den CityGML-Standard integriert, der sich seit seiner Verabschiedung 2008 international durchgesetzt hat und zum Austausch und zur Beschreibung virtueller 3D-Stadtmodelle eingesetzt wird.

Im Bereich der Visualisierung konzentriert sich diese Arbeit auf die Projektion. Auf einer inhaltlichen Ebene wird die perspektivische Projektion bezüglich ihrer Effektivität bei der Vermittlung von Umgebungswissen für verschiedene typische Ansichten untersucht (Kap. 5). Daraus werden zwei multiperspektivische Ansichten abgeleitet, die jeweils eine kartenähnliche Ansicht mit einer egozentrischen Sicht kombinieren. Der Übergang beider Bildteile ist kontinuierlich, um eine Korrelation beider Perspektiven visuell zu unterstützen. Das Ergebnis sind kartenähnliche Darstellungen mit zusätzlich hohem Orientierungswert. Für die Umsetzung wurde ein echtzeitfähiges Verfahren basierend auf einer globalen Deformation entwickelt und erfolgreich in eine Geovisualisierungsplattform integriert.

Auf einer technischen Ebene wird die qualitativ hochwertige, echtzeitfähige Umsetzung nichtlinearer Projektionen untersucht (Kap. 6 und 7). Solche Projektionen werden zum Beispiel bei der Fokus&Kontext-Visualisierung eingesetzt, um dem Betrachter zusätzliche Daten zeigen zu können. Das zentrale Problem besteht dabei in der Grundannahme einer linearen Projektion durch die zur Echtzeitvisualisierung eingesetzte 3D-Grafikhardware. Nichtlineare Projektionen erfordern deshalb spezielle Verfahren für den Einsatz in interaktiven Anwendungen. In dieser Arbeit wird mit stückweise perspektivischen Projektionen ein Konzept erarbeitet, das die Approximation einer Klasse nichtlinearer Projektionen durch ausschließlich lineare Projektionen erlaubt. Solche Approximationen lassen sich direkt und mit optimaler Qualität auf 3D-Grafikhardware rendern. Zur Umsetzung werden Verfahren zur Replikation und Verfeinerung entwickelt, die an die technischen Möglichkeiten verschiedener 3D-Grafikhardware angepasst sind.

Die vorliegende Arbeit erweitert die Prozesskette zur Erzeugung und Nutzung virtueller 3D-Stadtmodelle durch wesentliche Technologien: Zum einen lassen sich mit den Ergebnissen der Arbeit Texturen in virtuelle 3D-Stadtmodelle integrieren und nahtlos austauschen. Somit trägt diese Arbeit dazu bei, die Herstellung und Fortführung texturierter virtueller 3D-Stadtmodelle zu verbessern. Zum anderen zeigt die Arbeit Varianten und technische Lösungen für neuartige Projektionstypen für virtuelle 3D-Stadtmodelle in interaktiven Visualisierungen. Solche nichtlinearen Projektionen stellen Schlüsselbausteine dar, um neuartige Benutzungsschnittstellen für und Interaktionsformen mit virtuellen 3D-Stadtmodellen zu ermöglichen, insbesondere für mobile Geräte und immersive Umgebungen. Damit ergänzt die vorliegende Arbeit die genannte Prozesskette um neue Glieder und ermöglicht innovative Anwendungen und Systeme auf Basis virtueller 3D-Stadtmodelle.

8.2 Ausblick

Die in dieser Arbeit vorgestellten Ansätze im Bereich der Texturierung virtueller 3D-Stadtmodelle bilden einen Startpunkt für die Integration oberflächengebundener Daten in Geoinformationssysteme. Mit der Möglichkeit, solche Daten flächendeckend automatisiert zu erfassen und auszutauschen, werden sie zu einer praktisch verfügbaren Informationsquelle. Derzeit werden oberflächengebundene Daten jedoch hauptsächlich zur Visualisierung, nicht aber bei der Prozessierung und Analyse eingesetzt. Es mangelt dabei an generischen Operationen, auf denen Basis Anwendungen, die von der hohen Dichte oberflächengebundener Daten profitieren, entwickelt werden können (Lorenz und Döllner, 2010a). Beispiele für solche Anwendungen sind Solarpotentialanalyse oder der visuelle Kontakt (Engel und Döllner, 2009). Mit einer solchen Ergänzung würden oberflächengebundene Daten zu einem flexiblen und generischen Werkzeug in Geoinformationssystemen werden.

Im Bereich der Visualisierung ist der nächste Schritt die Verifikation der theoretischen Effektivitätssteigerung der multiperspektivischen Ansichten durch eine Nutzerstudie. Darin ließe sich der tatsächliche Nutzen, der auch von den Gewohnheiten des Nutzers beeinflusst wird, für verschiedene Einsatzszenarien bestimmen. Eine mögliche Einsatzumgebung sind mobile Geräte, die inzwischen ausreichend Rechenleistung für die 3D-Darstellung bieten. Die betrachtete Aufgabe – Erlangung von Wissen um die räumliche Struktur zur Orientierung in einer Stadt – ist im Kontext mobiler Kartografie ein wesentliches Nutzungsszenario (Kölmel und Hubschneider, 2002). Ein Vergleich zwischen 2D- und 3D-Karten (Oulasvirta u. a., 2009) zeigte Vorteile der 2D-Karten, von denen möglicherweise auch multiperspektivische Ansichten profitieren. Im Rahmen

der nichtlinearen Projektionen bestehen zwei Entwicklungsrichtungen: Zum einen sollten die Beschränkungen der beschriebenen Konzepte untersucht und gelockert werden, um zusätzliche nichtlineare Projektionen zu unterstützen. Zum anderen sollten neue Anwendungsfälle identifiziert werden. Eine nutzbare Eigenschaft ist beispielsweise die kompakte Beschreibung stückweise perspektivischer Projektionen, die eine interaktive Veränderung in Echtzeit ermöglicht. Die Projektion lässt sich dadurch dynamisch an den aktuellen Bildinhalt anzupassen und bspw. zur Umsetzung einer einem Objekt folgenden Vergrößerungslinse verwenden.

Literaturverzeichnis

Edward H. Adelson und James R. Bergen (1991). *The Plenoptic Function and the Elements of Early Vision.* In: Michael S. Landy und Anthony J. Movshon (Eds.), Computational Models of Visual Processing, MIT Press, Cambridge, MA. Seiten 3–20. [23]

Maneesh Agrawala, Denis Zorin und Tamara Munzner (2000). *Artistic Multiprojection Rendering.* In: Proc. of the Eurographics Workshop on Rendering Techniques 2000. Springer-Verlag, Seiten 125–136. [16]

Tomas Akenine-Möller, Eric Haines und Naty Hoffman (2008). Real-time Rendering. Peters, Wellesley, 3. Aufl.. [8, 9, 21, 27, 29, 34, 53, 74, 75, 76, 88]

Yahya Alshawabkeh und Norbert Haala (2005). *Automatic Multi-Image Photo-Texturing of Complex 3D Scenes.* In: XVIII CIPA Int. Symposium. ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, vol. XXXIV, part 5/C34, IAPRS, Seiten 68–73. [25]

AMD (2009). *AMD Stream Computing SDK.* `http://ati.amd.com/technology/streamcomputing/`. Zuletzt abgerufen: 09.07.2010. [22]

Gene M. Amdahl (1967). *Validity of the Single Processor Approach to Achieving Large Scale Computing Capabilities.* In: AFIPS '67 (Spring): Proceedings of the Spring Joint Computer Conference. ACM, New York, NY, USA, Seiten 483–485. [104]

Remi Arnaud und Mark C. Barnes (2006). Collada: Sailing the Gulf of 3d Digital Content Creation. AK Peters, Ltd., Wellesley. [48]

Arul Asirvatham und Hugues Hoppe (2005). *Terrain Rendering Using GPU-Based Geometry Clipmaps.* In: Matt Pharr (Ed.), GPU Gems 2, Addison-Wesley, Kap. 2. Seiten 27–45. [13]

ATI Inc. (2001). *TRUFORM white paper.* Tech. Rep., AMD. [83]

Autodesk (2010). *ImageModeler.* `http://usa.autodesk.com/adsk/servlet/pc/index?id=11390028&siteID=123112`. Zuletzt abgerufen: 09.07.2010. [25]

Andreas Bærentzen, Steen L. Nielsen, Mikkel Gjøl, Bent D. Larsen und Niels Jørgen Christensen (2006). *Single-pass wireframe rendering.* In: ACM SIGGRAPH 2006 Sketches. ACM, Seite 149. [101, 102]

Mark Barnes und Ellen Levy Finch (Eds.) (2001). COLLADA - Digital Asset Schema. Khronos Group, release 1.4.1 Aufl.. [10, 51]

Alan H. Barr (1984). *Global and local deformations of solid primitives.* In: SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques. ACM, New York, NY, USA, Seiten 21–30. [65]

Konstantin Baumann (2000). Modellierung, Texturierung und Rendering digitaler Geländemodelle. Dissertation, Universität Münster. [13, 70, 88]

Berlin Partner GmbH (2009). *Das amtlich Berlin.* `http://www.3d-stadtmodell-berlin.de`. Zuletzt abgerufen: 09.07.2010. [1]

Fausto Bernardini, Ioana M. Martin und Holly Rushmeier (2001). *High-Quality Texture Reconstruction from Multiple Scans.* IEEE Transactions on Visualization and Computer Graphics, Bd. 7 (4), Seiten 318–332. [27]

Ralf Bill (1999). Grundlagen der Geo-Informationssysteme, Band 2 - Analysen, Anwendungen und neue Entwicklungen. Wichmann Verlag, Heidelberg, Germany, 2 Aufl.. [3]

Jim Blinn (1992). *Hyperbolic Interpolation.* IEEE Computer Graphics and Applications, Bd. 12 (4), Seiten 89–94. [76]

David Blythe (2006). *The Direct3D 10 system.* ACM Trans. Graph., Bd. 25 (3), Seiten 724–734. [20]

Martin Bokeloh und Michael Wand (2006). *Hardware accelerated multi-resolution geometry synthesis.* In: I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games. ACM, New York, NY, USA, Seiten 191–198. [76]

Jürgen Bollmann und Wolf G. Koch (Eds.) (2005). Lexikon der Kartographie und Geomatik. Spektrum Akademischer Verlag. [1, 2]

Alexander Bornik, Konrad F. Karner, Joachim Bauer, Franz Leberl und Heinz Mayer (2001). *High-quality texture reconstruction from multiple views.* Journal of Visualization and Computer Animation, Bd. 12 (5), Seiten 263–276. [24, 25]

Joachim Böttger, Martin Preiser, Michael Balzer und Oliver Deussen (2008). *Detail-In-Context Visualization for Satellite Imagery.* Comput. Graph. Forum, Bd. 27 (2), Seiten 587–596. [61]

Tamy Boubekeur und Marc Alexa (2008). *Phong Tessellation.* ACM Trans. Graph., Bd. 27 (5), Seiten 1–5. [83]

Tamy Boubekeur und Christophe Schlick (2005a). *Generic mesh refinement on GPU.* In: Proceedings of the HWWS. ACM, Seiten 99–104. [75, 84]

Tamy Boubekeur und Christophe Schlick (2005b). *Scalar Tagged PN Triangles.* In: Eurographics (Short Paper). Blackwell Publishing, Seiten 17–20. [83]

Tamy Boubekeur und Christophe Schlick (2008). *A Flexible Kernel for Adaptive Mesh Refinement on GPU.* Computer Graphics Forum, Bd. 27 (1), Seiten 102–114. [21, 66, 74, 75, 77, 86]

Chas Boyd (2008). *The DirectX 11 Compute Shader.* In: Beyond Programmable Shading. ACM SIGGRAPH 2008 Course Notes. [22]

Margarita Bratkova, Peter Shirley und William B. Thompson (2009). *Artistic rendering of mountainous terrain.* ACM Trans. Graph., Bd. 28 (4), Seiten 1–17. [61]

John Brosz, Faramarz F. Samavati, M. S. T. Carpendale und Mario Costa Sousa (2007). *Single camera flexible projection.* In: Proc. of NPAR '07. ACM, Seiten 33–42. [16, 17, 59]

Henrik Buchholz (2006). Real-Time Visualization of 3D City Models. Dissertation, Universität Potsdam. [3, 8, 9, 28, 29]

Henrik Buchholz, Johannes Bohnet und Jürgen Döllner (2005). *Smart and Physically-Based Navigation in 3D Geovirtual Environments.* In: 9th International Conference on Information Visualization. IEEE Computer Society Press, Seiten 629–635. [13, 60, 62, 69, 70, 72]

Henrik Buchholz und Jürgen Döllner (2005). *View-Dependent Rendering of Multiresolution Texture-Atlases.* In: IEEE Visualization 2005. IEEE Computer Society Press, Seiten 215–222. [13]

Ian Buck, Tim Foley, Daniel Horn, Jeremy Sugerman, Kayvon Fatahalian, Mike Houston und Pat Hanrahan (2004). *Brook for GPUs: stream computing on graphics hardware.* In: SIGGRAPH '04: ACM SIGGRAPH 2004 Papers. ACM, New York, NY, USA, Seiten 777–786. [22]

Michael Bunnell (2005). *Adaptive Tesselation of Subdivision Surfaces with Displacement Mapping.* In: Matt Pharr (Ed.), GPU Gems 2, Addison-Wesley, Kap. 7. Seiten 109–122. [74, 76]

M. S. T. Carpendale und Catherine Montagnese (2001). *A framework for unifying presentation space.* In: UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology. ACM, New York, NY, USA, Seiten 61–70. [61]

Nathan A. Carr, Jesse D. Hall und John C. Hart (2002). *The ray engine.* In: HWWS '02: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, Seiten 37–46. [22]

Ignacio Castaño (2008). *Tesselation of Displaced Subdivision Surfaces in DX11.* In: XNA Gamefest 2008. [21]

Remco Chang, Thomas Butkiewicz, Caroline Ziemkiewicz, Zachary Wartell, Nancy Pollard und William Ribarsky (2008). *Legible Simplification of Textured Urban Models.* IEEE Comput. Graph. Appl., Bd. 28 (3), Seiten 27–36. [13]

Yun-Seok Choi, Kyu-Sik Chung und Lee-Sup Kim (2004). *Adaptive Tessellation of PN Triangles Using Minimum-Artifact Edge Linking.* IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences, Bd. E87-A (10), Seiten 2821–2828. [83]

Paolo Cignoni, Marco Di Benedetto, Fabio Ganovelli, Enrico Gobbetti, Fabio Marton und Roberto Scopigno (2007). *Ray-Casted BlockMaps for Large Urban Models Streaming and Visualization.* Computer Graphics Forum, Bd. 26 (3), Seiten 405–413. [13]

CityGML Wiki (2010). *Examples and WFSs.* `http://www.citygmlwiki.org/index.php/Examples_and_WFSs`. Zuletzt abgerufen: 09.07.2010. [55]

Andy Cockburn, Amy Karlson und Benjamin B. Bederson (2008). *A review of overview+detail, zooming, and focus+context interfaces.* ACM Comput. Surv., Bd. 41 (1), Seiten 1–31. [14, 60, 61]

Daniel Cohen-Or, Yiorgos L. Chrysanthou, Cláudio T. Silva und Frédo Durand (2003). *A Survey of Visibility for Walkthrough Applications.* IEEE Transactions on Visualization and Computer Graphics, Bd. 9, Seiten 412–431. [13]

Paul Cooper (Ed.) (2008). The OpenGIS Abstract Specification, Topic 2: Spatial referencing by coordinates. Version 4.0.0, doc.no. 08-015r2, OGC. [2]

Microsoft Corp. (2006). *Direct3D 10 Reference.* `http://msdn.microsoft.com/directx`. Zuletzt abgerufen: 09.07.2010. [51]

Microsoft Corp. (2008). *Direct3D 11 Reference.* `http://msdn.microsoft.com/directx`. Zuletzt abgerufen: 09.07.2010. [22]

Michael Cramer (2003). *Erfahrungen mit der direkten Georeferenzierung.* Photogrammetrie - Fernerkundung - Geoinformation, Heft 4/2003, Seiten 267–278. [11]

Cyril Crassin (2008). *Understanding G80 behavior and performances.* `http://www.icare3d.org/GPU/CN08`. Zuletzt abgerufen: 09.07.2010. [19, 105]

Rudy P. Darken und John L. Sibert (1993). *A toolset for navigation in virtual environments.* In: UIST '93: Proceedings of the 6th annual ACM symposium on User interface software and technology. ACM, New York, NY, USA, Seiten 157–165. [13, 60]

Paul Debevec, Yizhou Yu und George Boshokov (1998). *Efficient View-Dependent Image-Based Rendering with Projective Texture-Mapping.* Tech. Rep., Berkeley, CA, USA. [24]

Paul E. Debevec, Camillo J. Taylor und Jitendra Malik (1996). *Modeling and rendering architecture from photographs: a hybrid geometry- and image-based approach.* In: SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques. ACM, New York, NY, USA, Seiten 11–20. [24]

Christopher DeCoro und Natalya Tatarchuk (2007). *Real-time Mesh Simplification Using the GPU.* In: Proceedings of the I3D. Seiten 161–166. [13]

Patrick Degener und Reinhard Klein (2009). *A variational approach for automatic generation of panoramic maps.* ACM Trans. Graph., Bd. 28 (1), Seiten 1–14. [61]

Jürgen Döllner (2006). *Non-Photorealistic 3D Geovisualization.* In: William Cartwright, Michael P. Peterson und Georg Gartner (Eds.), Multimedia Cartography, second edition, Springer, Kap. 16. Seiten 229–240. [68]

Jürgen Döllner und Henrik Buchholz (2005). *Non-Photorealism in 3D Geovirtual Environments*. In: AutoCarto Proceedings Papers. Online. [1]

Jürgen Döllner, Henrik Buchholz und Haik Lorenz (2006). *Ambient Occlusion - ein Schritt zur realistischen Beleuchtung von 3D-Stadtmodellen*. GIS - Zeitschrift für Geoinformatik, Seiten 7–13. [55, 57]

Jürgen Döllner, Henrik Buchholz, Marc Nienhaus und Florian Kirsch (2005). *Illustrative Visualization of 3D City Models*. In: Robert F. Erbacher, Matti T. Roberts, Jonathan C.and Gröhn und Katy Börner (Eds.), Visualization and Data Analysis. International Society for Optical Engine (SPIE), *Proceedings of the SPIE*, Bd. 5669, Seiten 42–51. [8]

Jürgen Döllner und Klaus Hinrichs (2002). *A Generic Rendering System*. IEEE Transactions on Visualization and Computer Graphics, Bd. 8 (2), Seiten 99–118. [18]

Martin Drauschke, Hanns-Florian Schuster und Wolfgang Förstner (2006). *Detectibility of Buildings in Aerial Images over Scale Space*. In: Wolfgang Förstner und Richard Steffen (Eds.), Symposium of ISPRS Commission III: Photogrammetric Computer Vision. ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, vol. XXXVI, part 3, ISPRS, IAPRS, Seiten 7–12. [36]

Mark Duchaineau, Murray Wolinsky, David E. Sigeti, Mark C. Miller, Charles Aldrich und Mark B. Mineev-Weinstein (1997). *ROAMing terrain: real-time optimally adapting meshes*. In: VIS '97: Proceedings of the 8th conference on Visualization '97. IEEE Computer Society Press, Los Alamitos, CA, USA, Seiten 81–88. [13, 79]

Christopher Dyken, Martin Reimers und Johan Seland (2008). *Real-Time GPU Silhouette Refinement using Adaptively Blended Bezier Patches*. Computer Graphics Forum, Bd. 27 (1), Seiten 1–12. [75]

Christopher Dyken, Martin Reimers und Johan Seland (2009). *Semi-uniform Adaptive Patch Tessellation*. Computer Graphics Forum, Bd. 28 (8), Seiten 2255–2263. [75]

Jason Dykes, Alan. M. MacEachren und Menno-Jan Kraak (2005). Exploring Geovisualization. Pergamon. [13]

Juri Engel und Jürgen Döllner (2009). *Approaches Towards Visual 3D Analysis for Digital Landscapes and Its Applications*. Digital Landscape Architecture Proceedings 2009, Seiten 33–41. [107]

EOS Systems (2010). *PhotoModeler*. `http://www.photomodeler.com/index.htm`. Zuletzt abgerufen: 09.07.2010. [25]

Carl Erikson, Dinesh Manocha und William V. Baxter, III (2001). *HLODs for faster display of large static and dynamic environments*. In: I3D '01: Proceedings of the 2001 symposium on Interactive 3D graphics. ACM, New York, NY, USA, Seiten 111–120. [13]

Martin Falk, Tobias Schafhitzel, Daniel Weiskopf und Thomas Ertl (2007). *Panorama maps with non-linear ray tracing*. In: GRAPHITE '07: Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia. ACM, New York, NY, USA, Seiten 9–16. [61]

Amal A. Farag, Shireen Y. Elhabian, Abdelrehim H. Ahmed und Aly A. Farag (2008). *Noise Analysis of a SFS Algorithm Formulated under Various Imaging Conditions*. In: ISVC '08: Proceedings of the 4th International Symposium on Advances in Visual Computing. Springer-Verlag, Berlin, Heidelberg, *Lecture Notes in Computer Science*, Bd. 5358, Seiten 793–802. [33]

James D. Foley, Andries VanDam, Steven K. Feiner und John F. Hughes (1995). Computer Graphics: Principles and Practice in C. Addison-Wesley Longman. [15, 16, 27, 29, 53, 74]

David A. Forsyth und Jean Ponce (2002). Computer Vision: A Modern Approach. Prentice Hall Professional Technical Reference. [26, 27]

Bert Freudenberg, Maic Masuch und Thomas Strothotte (2001). *Walk-through illustrations: Frame-coherent pen-and-ink in game engine*. In: Proc. of Eurographics 2001. Seiten 184–191. [101, 102]

Christian Früh, Russell Sammon und Avideh Zakhor (2004). *Automated Texture Mapping of 3D City Models With Oblique Aerial Imagery*. In: Proceedings of the 2nd International Symposium on 3D Data

Processing, Visualization and Transmission (3DPVT 2004). IEEE Computer Society, Seiten 396–403. [25, 36]

Sven Fuhrmann und Alan M. MacEachren (2001). *Navigation in desktop geovirtual environments: Usability assessment.* In: Proceedings of the 20th ICA/ACI International Cartographic Conference. ICA, Beijing, China, August 6-10, 2001, Seiten 2444–2453. [2, 60]

Tinsley A. Galyean (1995). *Guided navigation of virtual environments.* In: I3D '95: Proceedings of the 1995 symposium on Interactive 3D graphics. ACM, New York, NY, USA, Seiten 103–ff. [13]

Jean-Dominique Gascuel, Nicolas Holzschuch, Gabriel Fournier und Bernard Péroche (2008). *Fast non-linear projections using graphics hardware.* In: Symposium on Interactive 3D graphics and games SI3D '08. ACM, Seiten 107–114. [22]

Tassilo Glander und Jürgen Döllner (2008). *Techniques for Generalizing Building Geometry of Complex Virtual 3D City Models.* In: Peter van Oosterom, Sisi Zlatanova, Friso Penninga und Elfriede M. Fendel (Eds.), Advances in 3D Geoinformation Systems. Lecture Notes in Geoinformation and Cartography, Springer, Seiten 381–400. [13]

Tassilo Glander, Matthias Trapp und Jürgen Döllner (2007). *A Concept of Effective Landmark Depiction in Geovirtual 3D Environments by View-Dependent Deformation.* In: 4th International Symposium on LBS and Telecartography. CD proceedings. [68]

Andrew S. Glassner (2004a). *Digital Cubism.* IEEE Computer Graphics and Applications, Bd. 24 (3), Seiten 82–90. [16, 17]

Andrew S. Glassner (2004b). *Digital Cubism, Part 2.* IEEE Computer Graphics and Applications, Bd. 24 (4), Seiten 84–95. [16, 17]

Rüdiger Göbel und Nicolai Freiwald (2008). *Texturen für 3D-Stadtmodelle - Typisierung und Erhebungsmethodik.* In: Manfred Schrenk, Vasily V. Popovich, Dirk Engelke und Pietro Elisei (Eds.), 13th International Conference on Urban Planning, Regional Development and Information Society (REAL CORP). CORP - Competence Center of Urban and Regional Planning, Seiten 659–663. [9, 10, 45]

Google, Inc. (2010). *GoogleEarth.* `http://earth.google.com`. Zuletzt abgerufen: 09.07.2010. [1]

Lazaros Grammatikopoulos, Ilias Kalisperakis, George Karras, T. Kokkinos und E. Petsa (2004). *Automatic Multi-Image Photo-texturing of 3D Surface Models Obtained With laser Scanning.* In: CIPA International Workshop on Vision Techniques Applied. [25]

Görres J. Grenzdörffer, Markus Guretzki und Ilan Friedlander (2009). *Photogrammetric image acquisition and image analysis of oblique imagery.* The Photogrammetric Record, Bd. 23 (124), Seiten 372–386. [12]

Gerhard Gröger, Thomas H. Kolbe, Angela Czerwinski und Claus Nagel (Eds.) (2008). OpenGIS City Geography Markup Language (CityGML) Implementation Specification. Version 1.0.0, doc.no. 08-007, OGC. [2, 5, 7, 47, 51, 52]

Eduard Gröller (1995). *Nonlinear Raytracing - Visualizing Strange Worlds.* The Visual Computer, Bd. 11 (5), Seiten 263–274. [16, 17]

Michael Gruber, Martin Ponticelli, Stefan Bernögger und Franz Leberl (2008). *Ultracamx, the Large Format Digital Aerial Camera System by Vexcel Imaging/Microsoft.* In: ISPRS Congress Beijing 2008. ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, vol. XXXVII, part B1, IAPRS, Seiten 665–670. [23]

Xianfeng Gu, Steven J. Gortler und Hugues Hoppe (2002). *Geometry images.* ACM Trans. Graph., Bd. 21 (3), Seiten 355–361. [76]

Michael Guthe, Aákos Balázs und Reinhard Klein (2005). *GPU-based trimming and tessellation of NURBS and T-Spline surfaces.* ACM Trans. Graph., Bd. 24 (3), Seiten 1016–1023. [75]

Christian Häberling (2003). „Topographische 3D-Karten“: Thesen für kartographische Gestaltungsgrundsätze. Dissertation, Eidgenössische Technische Hochschule ETH Zürich, Nr. 15379. [72]

Paul Haeberli und Mark Segal (1993). *Texture Mapping as a Fundamental Drawing Primitive.* In: Proceedings of the Fourth Eurographics Workshop on Rendering. Eurographics, Seiten 259–266. [8, 9]

Chris Hand (1997). *A Survey of 3D Interaction Techniques.* Computer Graphics Forum, Bd. 16 (5), Seiten 269–281. [13, 60, 69]

Mark Harris (2010). *GPGPU.org.* `www.gpgpu.org`. Zuletzt abgerufen: 09.07.2010. [22]

Paul S. Heckbert (1989). *Fundamentals of Texture Mapping and Image Warping.* Tech. Rep. UCB/CSD-89-516, EECS Department, University of California, Berkeley. [36, 99, 100]

Wolfgang Heidrich und Hans-Peter Seidel (1998). *View-independent environment maps.* In: HWWS '98: ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware. ACM, Seiten 39–45. [16]

Stefan Höffken (2009). Google Earth in der Stadtplanung – Die Anwendungsmöglichkeiten von Virtual Globes in der Stadtplanung am Beispiel von Google Earth. Diplomarbeit, TU Berlin. [1]

Hugues Hoppe (1996). *Progressive meshes.* In: Proc. of SIGGRAPH '96. ACM, Seiten 99–108. [13, 21, 79]

Kasper Hornbæk, Benjamin B. Bederson und Catherine Plaisant (2002). *Navigation patterns and usability of zoomable user interfaces with and without an overview.* ACM Trans. Comput.-Hum. Interact., Bd. 9 (4), Seiten 362–389. [64]

Xianyou Hou, Li-Yi Wei, Heung-Yeung Shum und Baining Guo (2006). *Real-time multi-perspective rendering on graphics hardware.* In: EUROGRAPHICS Symposium on Rendering. Blackwell Publishing. [22, 89, 91]

Mike Houston (2008). *Introduction to the AMD Stream SDK.* In: Beyond Programmable Shading. ACM SIGGRAPH 2008 Course Notes. [22]

Lok M. Hwa, Mark A. Duchaineau und Kenneth I. Joy (2004). *Adaptive 4-8 Texture Hierarchies.* In: VIS '04: Proceedings of the conference on Visualization '04. IEEE Computer Society, Washington, DC, USA, Seiten 219–226. [13]

ISO 19123:2005 (2005). Geographic information - Schema for coverage geometry and functions. ISO. [50]

ISO/IEC FDIS 19775-1:2008 (2008). Extensible 3D (X3D). Version 3.2, ISO/IEC. [10, 51]

Kensei Jo, Kouta Minamizawa, Hideaki Nii, Naoki Kawakami und Susumu Tachi (2008). *A GPU-based real-time rendering method for immersive stereoscopic displays.* In: ACM SIGGRAPH 2008 posters. ACM, Seite 1. [16]

Markus Jobst und Jürgen Döllner (2008). *3D City Model Visualization with Cartography-Oriented Design.* In: Manfred Schrenk, Vasily V. Popovich, Dirk Engelke und Pietro Elisei (Eds.), 13th International Conference on Urban Planning, Regional Development and Information Society (REAL CORP). CORP - Competence Center of Urban and Regional Planning, Seiten 507–516. [59, 63]

Astrid Jurisch und David Mountain (2008). *Evaluating the Viability of Pictometry®Imagery for Creating Models of the Built Environment.* In: ICCSA '08: Proceeding sof the international conference on Computational Science and Its Applications, Part I. Springer-Verlag, Berlin, Heidelberg, *Lecture Notes in Computer Science*, Bd. 5072, Seiten 663–677. [25]

Martin Kada (2009). *The 3D Berlin Project.* In: Dieter Fritsch (Ed.), Photogrammetric Week 2009. Wichmann Verlag, Heidelberg, Seiten 331–340. [1, 44]

Martin Kada, Darko Klinec und Norbert Haala (2005). *Facade Texturing for rendering 3D city models.* In: Proceedings of the ASPRS Conference. Seiten 78–85. [25]

Nils Karbo und Ralf Schroth (2009). *Oblique Aerial Photography: A Status Review.* In: Dieter Fritsch (Ed.), Photogrammetric Week 2009. Wichmann Verlag, Heidelberg, Seiten 119–126. [25, 46]

Alan Keahey (1998). *The Generalized Detail-In-Context Problem.* In: INFOVIS '98: Proceedings of the 1998 IEEE Symposium on Information Visualization. IEEE Computer Society, Washington, DC, USA, Seiten 44–51. [59]

Daniel A. Keim, Florian Mansmann und Jim Thomas (2009). *Visual analytics: how much visualization and how much analytics?* SIGKDD Explor. Newsl., Bd. 11 (2), Seiten 5–8. [13]

Khronos OpenCL Working Group (2009). *The OpenCL Specification, Version: 1.0.48*. `http://www.khronos.org/registry/cl/specs/opencl-1.0.48.pdf`. Zuletzt abgerufen: 09.07.2010. [22]

Sehwan Kim, Christopher Coffin und Tobias Höllerer (2009). *Relocalization using virtual keyframes for online environment map construction*. In: VRST '09: Proceedings of the 16th ACM Symposium on Virtual Reality Software and Technology. ACM, New York, NY, USA, Seiten 127–134. [33]

Thomas H. Kolbe, Claus Nagel und Alexandra Stadler (2009). *CityGML - OGC Standard for Photogrammetry?* In: Dieter Fritsch (Ed.), Photogrammetric Week 2009. Wichmann Verlag, Heidelberg, Seiten 265–277. [3]

Bernhard Kölmel und Martin Hubschneider (2002). *Nutzererwartungen an Location Based Services - Ergebnisse einer empirischen Analyse*. In: Alexander Zipf und Josef Strobl (Eds.), Geoinformation mobil. Wichmann Verlag, Heidelberg, Seiten 85–97. [107]

Johannes Kopf, Dani Lischinski, Oliver Deussen, Daniel Cohen-Or und Michael Cohen (2009). *Locally Adapted Projections to Reduce Panorama Distortions*. Computer Graphics Forum (Proceedings of EGSR 2009), Bd. 28 (4), Seiten 1083–1089. [16]

Karl Kraus (1997). Photogrammetrie, Band 1 - Grundlagen und Standardverfahren. Dümmlers Verlag, Bonn, Germany, 6 Aufl.. [10, 11, 26]

William Lalonde (Ed.) (2002). Styled Layer Descriptor Implementation Specification. Version 1.0.0, doc.no. 02-070, OGC. [48, 50]

Ricardo Lenz, Joaquim Bento Cavalcante-Neto und Creto Augusto Vidal (2009). *Optimized Pattern-Based Adaptive Mesh Refinement Using GPU*. In: Luis Gustavo Nonato und Jacob Scharcanski (Eds.), Proceedings. IEEE Computer Society, Los Alamitos. [74, 75]

Ying K. Leung und Mark D. Apperley (1994). *A review and taxonomy of distortion-oriented presentation techniques*. ACM Trans. Comput.-Hum. Interact., Bd. 1 (2), Seiten 126–160. [61]

Bruno Lévy, Sylvain Petitjean, Nicolas Ray und Jérome Maillot (2002). *Least squares conformal maps for automatic texture atlas generation*. ACM Trans. Graph., Bd. 21 (3), Seiten 362–371. [55]

Yotam Livny, Zvi Kogan und Jihad El-Sana (2009). *Seamless patches for GPU-based terrain rendering*. Vis. Comput., Bd. 25 (3), Seiten 197–208. [13]

H. Löffelmann und E. Gröller (1996). *Ray Tracing with Extended Cameras*. Journal of Visualization and Computer Animation, Bd. 7 (4), Seiten 211–227. [22]

Haik Lorenz und Jürgen Döllner (2006). *Towards Automating the Generation of Facade Textures of Virtual City Models*. ISPRS Commission II, WG II/5 Workshop, Vienna. [105]

Haik Lorenz und Jürgen Döllner (2008a). *Dynamic Mesh Refinement on GPU using Geometry Shaders*. In: Proc. of the 16th WSCG. [6, 66]

Haik Lorenz und Jürgen Döllner (2008b). *Modellierung oberflächenbezogener Informationen virtueller 3D Stadtmodelle in CityGML*. In: Mitteilungen des Bundesamtes für Kartographie und Geodäsie. Verlag des Bundesamtes für Kartographie und Geodäsie, Bd. 41, Seiten 69–82. [5]

Haik Lorenz und Jürgen Döllner (2009). *Real-time Piecewise Perspective Projections*. In: GRAPP 2009 - International Conference on Computer Graphics Theory and Applications. INSTICC Press, Seiten 147–155. [6]

Haik Lorenz und Jürgen Döllner (2010a). *3D Feature Surface Properties and Their Application in Geovisualization*. Computers, Environment and Urban Systems (CEUS), Bd. 34 (6), Seiten 476–483. [107]

Haik Lorenz und Jürgen Döllner (2010b). *High-Quality Non-Planar Projections Using Real-time Piecewise Perspective Projections*. In: Alpesh Kumar Ranchordas, João Madeiras Pereira, Hélder J. Araújo und João Manuel R. S. Tavares (Eds.), Computer Vision, Imaging and Computer Graphics. Theory and Applications. International Joint Conference, VISIGRAPP 2009, Lisboa, Portugal, February 5-8, 2009. Revised Selected Papers. Springer, *Communications in Computer and Information Science*, Bd. 68, Seiten 45–58. [6]

Haik Lorenz, Matthias Trapp, Markus Jobst und Jürgen Döllner (2008). *Interactive Multi-Perspective Views of Virtual 3D Landscape and City Models.* In: Lars Bernard, Anders Friis-Christensen und Hardy Pundt (Eds.), 11th AGILE International Conference on GI Science. Lecture Notes in Geoinformation and Cartography, Springer, Seiten 301–321. Best Paper Award. [5, 65]

Frank Losasso, Hugues Hoppe, Scott Schaefer und Joe D. Warren (2003). *Smooth geometry images.* In: SGP '03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing. Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, Seiten 138–145. [76]

David G. Lowe (1999). *Object Recognition from Local Scale-Invariant Features.* In: ICCV '99: Proceedings of the International Conference on Computer Vision-Volume 2. IEEE Computer Society, Washington, DC, USA, Seiten 1150–1157. [44]

David Luebke (2008). *CUDA Fundamentals.* In: Beyond Programmable Shading. ACM SIGGRAPH 2008 Course Notes. [22]

David Luebke und Greg Humphreys (2007). *How GPUs Work.* Computer, Bd. 40 (2), Seiten 96–100. [18]

Kevin Lynch (Ed.) (1960). The Image of the City. MIT Press. [63]

Stefan Maass und Jürgen Döllner (2006). *Dynamic Annotation of Interactive Environments using Object-Integrated Billboards.* In: Joaquim Jorge und Vaclav Skala (Eds.), 14th International Conference in Central Europe on Computer Graphics, Visualization and Computer Vision (WSCG). Seiten 327–334. [69]

Stefan Maass und Jürgen Döllner (2008). *Seamless Integration of Labels into Interactive Virtual 3D Environments Using Parameterized Hulls.* In: P. Brown, D. W. Cunningham, V. Interrante und J. MacCormack (Eds.), 4th International Symposium on Computational Aesthetics in Graphics, Visualization, and Imaging. The Eurographics Association, Seiten 33–40. [69]

Jock D. Mackinlay, Stuart K. Card und George G. Robertson (1990). *Rapid controlled movement through a virtual 3D workspace.* In: SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques. ACM, New York, NY, USA, Seiten 171–176. [69]

Mateusz Małczak (2010). *Quadratic Bezier curve length.* `http://segfaultlabs.com/docs/quadratic-bezier-curve-length`. Zuletzt abgerufen: 09.07.2010. [66]

Oliver Mattausch, Jiří Bittner und Michael Wimmer (2008). *CHC++: Coherent Hierarchical Culling Revisited.* Computer Graphics Forum (Proceedings Eurographics 2008), Bd. 27 (2), Seiten 221–230. [13]

Nelson Max (1983). *SIGGRAPH '84 call for Omnimax Films.* SIGGRAPH Comput. Graph., Bd. 17 (1), Seiten 73–76. [16]

Sebastian Möser, Patrick Degener, Roland Wahl und Reinhard Klein (2008). *Context Aware Terrain Visualization for Wayfinding and Navigation.* Computer Graphics Forum, Bd. 27 (7), Seiten 1853–1860. [61]

Markus Müller (Ed.) (2005). Symbology Encoding Implementation Specification. Version 1.1.0 (draft), doc.no. 05-077, OGC. [48, 50]

Pascal Müller, Gang Zeng, Peter Wonka und Luc Van Gool (2007). *Image-based procedural modeling of facades.* ACM Trans. Graph., Bd. 26 (3), Seite 85. [46]

Aaftab Munshi (2008). *OpenCL.* In: Beyond Programmable Shading. ACM SIGGRAPH 2008 Course Notes. [22]

Steffen Neubauer und Alexander Zipf (2007). *Suggestions for Extending the OGC Styled Layer Descriptor (SLD) Specification into 3D - Towards Visualization Rules for 3D City Models.* In: Urban Data Management Symposium. [50]

Steffen Neubauer und Alexander Zipf (Eds.) (2009). 3D-Symbology Encoding Discussion Draft. Version 0.0.1, doc.no. 09-042, OGC. [48, 50, 57]

Nokia GmbH (2010). *OVI maps.* `http://maps.ovi.com`. Zuletzt abgerufen: 09.07.2010. [72]

NVidia (2009). *NVidia CUDA Programming Guide, Version: 2.3.1*. `http://developer.download.nvidia.com/compute/cuda/2_3/toolkit/docs/NVIDIA_CUDA_Programming_Guide_2.3.pdf`. Zuletzt abgerufen: 09.07.2010. [22]

Eyal Ofek, Erez Shilat, Ari Rappoport und Michael Werman (1997). *Multiresolution Textures from Image Sequences*. IEEE Comput. Graph. Appl., Bd. 17 (2), Seiten 18–29. [24]

Open Geospatial Consortium (Ed.) (2008). OGC Reference Model. Version 2.0, doc.no. 08-062r4, OGC. [3]

OpenGL ARB (2010). *ARB_tessellation_shader*. `http://www.opengl.org/registry/specs/ARB/tessellation_shader.txt`. Zuletzt abgerufen: 09.07.2010. [76, 95]

Antti Oulasvirta, Sara Estlander und Antti Nurminen (2009). *Embodied interaction with a 3D versus 2D mobile map*. Personal Ubiquitous Comput., Bd. 13 (4), Seiten 303–320. [107]

John D. Owens, Mike Houston, David Luebke, Simon Green, John E. Stone und James C. Phillips (2008). *GPU Computing*. Proceedings of the IEEE, Bd. 96 (5), Seiten 879–899. [22]

John D. Owens, David Luebke, Naga Govindaraju, Mark Harris, Jens Krüger, Aaron E. Lefohn und Timothy J. Purcell (2007). *A Survey of General-Purpose Computation on Graphics Hardware*. Computer Graphics Forum, Bd. 26 (1), Seiten 80–113. [22]

Ioannis Pantazopoulos und Spyros Tzafestas (2002). *Occlusion Culling Algorithms: A Comprehensive Survey*. J. Intell. Robotics Syst., Bd. 35 (2), Seiten 123–156. [13]

Steven G. Parker, James Bigler, Andreas Dietrich, Heiko Friedrich, Jared Hoberock, David Luebke, David McAllister, Morgan McGuire, Keith Morley, Austin Robison und Martin Stich (2010). *OptiX: a general purpose ray tracing engine*. ACM Trans. Graph., Bd. 29 (4), Seiten 1–13. [22]

Tom Patterson (2000). *A View From on High: Heinrich Berann's Panoramas and Landscape Visualization Techniques For the US National Park Service*. Cartographic Perspectives, Bd. 36, Seiten 38–65. [59, 61, 64]

Shmuel Peleg, Moshe Ben-Ezra und Yael Pritch (2001). *Omnistereo: Panoramic Stereo Imaging*. IEEE Trans. Pattern Anal. Mach. Intell., Bd. 23 (3), Seiten 279–290. [17, 105]

Emil Persson (2007). *ATI Radeon HD2000 Programming Guide*. Tech. Rep., AMD, Inc. [21, 79, 88, 103]

Voicu Popescu und Daniel G. Aliaga (2006). *The depth discontinuity occlusion camera*. In: Proceedings of the 2006 Symposium on Interactive 3D Graphics. ACM, Seiten 139–143. [16, 17]

Clemens Portele (Ed.) (2007). OpenGIS Geography Markup Language (GML) Encoding Standard. Version 3.2.1, doc.no. 07-036, OGC. [7, 50]

Simon Premože (2002). *Computer Generated Panorama Maps*. In: Proceedings 3rd ICA Mountain Cartography Workshop. [61, 64]

Timothy J. Purcell, Ian Buck, William R. Mark und Pat Hanrahan (2002). *Ray tracing on programmable graphics hardware*. ACM Trans. Graph., Bd. 21 (3), Seiten 703–712. [22]

Paul Rademacher und Gary Bishop (1998). *Multiple-center-of-projection Images*. In: SIGGRAPH. Seiten 199–206. [16]

Patrick Reuter, Johannes Behr und Marc Alexa (2005). *An Improved Adjacency Data Structure for Fast Triangle Stripping*. ACM Journal of Graphics Tools, Bd. 10 (2), Seiten 41–50. [86]

Timo Ropinski, Frank Steinicke und Klaus Hinrichs (2005). *A constrained road-based VR navigation technique for travelling in 3D city models*. In: ICAT '05: Proceedings of the 2005 international conference on Augmented tele-existence. ACM, New York, NY, USA, Seiten 228–235. [13]

David Salomon (2006). Transformations and Projections in Computer Graphics. Springer-Verlag. [16]

Pedro V. Sander und Jason L. Mitchell (2005). *Progressive Buffers: View-dependent Geometry and Texture for LOD Rendering*. In: Symposium on Geometry Processing. Eurographics Association, Seiten 129–138. [21]

Heidrun Schumann und Wolfgang Müller (2000). Visualisierung - Grundlagen und allgmeine Methoden. Springer-Verlag. [13, 14]

Angelia Sebok, Espen Nystad und Stein Helgar (2004). *Navigation in desktop virtual environments: an evaluation and recommendations for supporting usability.* Virtual Real., Bd. 8 (1), Seiten 26–40. [62]

Mark Segal und Kurt Akeley (2003). *The OpenGL Graphics System: A Specification (Version 1.5).* `http://www.opengl.org/documentation/specs/version1.5/glspec15.pdf`. Zuletzt abgerufen: 09.07.2010. [18]

Mark Segal und Kurt Akeley (2009). *The OpenGL Graphics System: A Specification, Version 3.2 (Core Profile).* `http://www.opengl.org/registry/doc/glspec32.core.20091207.pdf`. Zuletzt abgerufen: 09.07.2010. [51]

Chris Seitz (2004). *Evolution of NV40.* `http://developer.nvidia.com/object/gpu_bbq_2004_presentations.html`. Zuletzt abgerufen: 09.07.2010. [21]

Le-Jeng Shiue, Ian Jones und Jörg Peters (2005). *A realtime GPU subdivision kernel.* ACM Trans. Graph., Bd. 24 (3), Seiten 1010–1015. [74, 76]

Martin J. Smith, Ahmed M. Hamruni und Allan Jamieson (2009). *3-D urban modelling using airborne oblique and vertical imagery.* In: ISPRS Hannover Workshop 2009: High-Resolution Earth Imaging for Geospatial Information. ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, vol. XXXVIII, part 1-4-7/W5, IAPRS. [25]

Robert Spence (2001). Information Visualization. Addison Wesley. [12, 13]

Martin Spindler, Marco Bubke, Tobias Germer und Thomas Strothotte (2006). *Camera textures.* In: Proceedings of the 4th International Conference on Computer Graphics and Interactive Techniques in Australasia and Southeast Asia (GRAPHITE). ACM, Seiten 295–302. [17, 99]

Alexandra Stadler und Thomas H. Kolbe (2007). *Spatio-semantic Coherence in the Integration of 3D City Models.* In: Proceedings of the 5th International Symposium on Spatial Data Quality. [7]

Alexandra Stadler, Claus Nagel, Gerhard König und Thomas H. Kolbe (2008). *Making Interoperability Persistent: A 3D Geo Database Based on CityGML.* In: Jiyeong Lee und Sisi Zlatanova (Eds.), 3D Geo-Information Sciences, Springer, Kap. 11. Seiten 175–192. [44, 56]

Uwe Stilla, Jakub Kolecki und Ludwig Hoegner (2009). *Texture Mapping of 3D Building Models with Oblique Direct Geo-referenced Airborne IR Image Sequences.* In: ISPRS Hannover Workshop 2009: High-Resolution Earth Imaging for Geospatial Information. ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, vol. XXXVIII, part 1-4-7/W5, IAPRS. [25]

Ralf Stüber (2005). Generalisierung von Gebäudemodellen unter Wahrung der visuellen Richtigkeit. Dissertation, Rheinische Friedrich-Wilhelms-Universität zu Bonn. [13]

Shigeo Takahashi, Naoya Ohta, Hiroko Nakamura, Yuriko Takeshima und Issei Fujishiro (2002). *Modeling Surperspective Projection of Landscapes for Geographical Guide-Map Generation.* Computer Graphics Forum, Bd. 21 (3), Seiten 259–268. [61]

Shigeo Takahashi, Kenichi Yoshida, Kenji Shimada und Tomoyuki Nishita (2006). *Occlusion-Free Animation of Driving Routes for Car Navigation Systems.* IEEE Transactions on Visualization and Computer Graphics, Bd. 12 (5), Seiten 1141–1148. [61]

Sarah Tariq (2006). *Next Generation Effects in DirectX10.* In: NVidia tutorials session at SIGGRAPH 2006. [20]

Natalya Tatarchuk (2007). *Real-Time Tessellation on GPU.* In: Course 28: Advanced Real-Time Rendering in 3D Graphics and Games. ACM SIGGRAPH 2007. [20, 21]

Andrei Tatarinov (2008). *Instanced Tessellation in DirectX10.* In: GDC '08: Game Developers' Conference 2008. [21, 74, 75]

Frank Thiemann (2002). *Generalization of 3D Building Data.* In: Proceedings of the Joint International Symposium on GeoSpatial Theory, Processing and Applications. ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, vol. XXXIV, part 4, IAPRS. [13]

Matthias Trapp und Jürgen Döllner (2008). *A Generalization Approach for 3D Viewing Deformations of Single-Center Projections.* In: Proc. of GRAPP 2008. INSTICC Press, Seiten 162–170. [17, 21, 98]

Matthias Troyer (2008). *The World of H. C. Berann.* `http://www.berann.com`. Zuletzt abgerufen: 09.07.2010. [59]

Scott Vallance und Paul Calder (2001). *Multi-perspective images for visualisation.* In: ACM International Conference Proceeding Series. Australian Computer Society, Inc., Bd. 147, Seiten 69–76. [59, 61]

Alex Vlachos, Jörg Peters, Chas Boyd und Jason L. Mitchell (2001). *Curved PN triangles.* In: Proceedings of the I3D. ACM, Seiten 159–166. [83]

Panagiotis A. Vretanos (Ed.) (2005). Web Feature Service Implementation Specification. Version 1.1.0, doc.no. 04-094, OGC. [50]

Liang Wan, Tien-Tsin Wong und Chi-Sing Leung (2007). *Isocube: Exploiting the Cubemap Hardware.* IEEE Trans. on Vis. and Comp. Graphics, Bd. 13 (4), Seiten 720–731. [16]

Yandong Wang, Steve Schultz und Frank Giuffrida (2008). *Pictometry's Proprietary Airborne Digital Imaging System and its Application in 3D City Modelling.* In: ISPRS Congress Beijing 2008. ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, vol. XXXVII, part B1, IAPRS, Seiten 1065–1070. [12]

Li-Yi Wei, Baoquan Liu, Xu Yang, Chongyang Ma, Ying-Qing Xu und Baining Guo (2007). *Nonlinear Beam Tracing on a GPU.* Tech. Rep., Microsoft, MSR-TR-2007-168. [16, 22]

Tim Weyrich, Jason Lawrence, Hendrik Lensch, Szymon Rusinkiewicz und Todd Zickler (2008). *Principles of appearance acquisition and representation.* In: SIGGRAPH '08: ACM SIGGRAPH 2008 classes. ACM, New York, NY, USA, Seiten 1–119. [27]

Turner Whitted und Jim Kajiya (2005). *Fully procedural graphics.* In: HWWS '05: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware. ACM, New York, NY, USA, Seiten 81–90. [22]

Albert Wiedemann (2009). *Photogrammetrische Schrägluftbilder mit dem Aerial Oblique System AOS.* In: Tagungsband der DGPF-Jahrestagung, Band 18. DGPF. [12]

Matthias Wloka (2005). *Improved Batching via Texture Atlases.* In: Wolfgang Engel (Ed.), ShaderX3, Charles River Media, Kap. 2.6. Seiten 155–167. [29]

Peter Wonka, Michael Wimmer und François X. Sillion (2001). *Instant Visibility.* Comput. Graph. Forum, Bd. 20 (3), Seiten 411–421. [13]

Daniel N. Wood, Adam Finkelstein, John F. Hughes, Craig E. Thayer und David H. Salesin (1997). *Multi-perspective panoramas for cel animation.* In: Proc. of ACM SIGGRAPH '97. ACM Press/Addison-Wesley Publishing Co., Seiten 243–250. [16]

Jun Wu, Guoqing Zhou, FengLi Ding und Zhigang Liu (2007). *Automatic Retrieval of Optimal Texture from Aerial Video for Photo-realistic 3D Visualization of Street Environment.* In: ICIG '07: Proceedings of the Fourth International Conference on Image and Graphics. IEEE Computer Society, Washington, DC, USA, Seiten 943–947. [25]

Yonggao Yang, Jim X. Chen und Mohsen Beheshti (2005). *Nonlinear Perspective Projections and Magic Lenses: 3D View Deformation.* IEEE Computer Graphics and Applications, Bd. 25 (1), Seiten 76–84. [17, 21, 90]

Jingyi Yu und Leonard McMillan (2004a). *A Framework for Multiperspective Rendering.* In: Alexander Keller und Henrik Wann Jensen (Eds.), Proceedings of Eurographics Symposium on Rendering 2004. EUROGRAPHICS Association, Seiten 61–68. [59]

Jingyi Yu und Leonard McMillan (2004b). *General Linear Cameras.* In: Proceedings of ECCV 2004, 8th European Conference on Computer Vision, Part II. Springer, *Lecture Notes in Computer Science*, Bd. 3022, Seiten 14–27. [17, 22]

Xuan Yu, Jingyi Yu und Leonard McMillan (2009). *Towards multi-perspective rasterization.* Vis. Comput., Bd. 25 (5-7), Seiten 549–557. [21]

Siyka Zlatanova (2000). 3D GIS for Urban Development. Dissertation, Technische Universität Graz, Österreich. [1]

Siyka Zlatanova, Alias Rahman und Morakot Pilouk (2002). *3D GIS: Current Status and Perspectives.* In: Proceedings of the ISPRS Commission IV Symposium "Geospatial Theory, Processing and Applications“. ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, vol. XXXIV, part 4, IAPRS. CDROM. [1]

Ehrenwörtliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Dissertation ohne Hilfe Dritter und ohne Zuhilfenahme anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe. Die den benutzten Quellen wörtlich oder inhaltlich entnommenen Stellen sind als solche kenntlich gemacht. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Potsdam, den 10. November 2010

Haik Lorenz