Ralph Stelzer · Karl-Heinrich Grote · Klaus Brökel
Frank Rieg · Jörg Feldhusen (Hrsg.)

ENTWERFEN ENTWICKELN **ERLEBEN**

Methoden und Werkzeuge in der Produktentwicklung

10. Gemeinsames Kolloquium Konstruktionstechnik
KT2012 | Residenzschloss Dresden | 14.–15. Juni 2012

Stelzer · Grote · Brökel · Rieg · Feldhusen (Hrsg.)

ENTWERFEN ENTWICKELN **ERLEBEN**

Methoden und Werkzeuge in der Produktentwicklung

10. Gemeinsames Kolloquium Konstruktionstechnik KT2012

TUD*press*

Ralph Stelzer · Karl-Heinrich Grote · Klaus Brökel
Frank Rieg · Jörg Feldhusen (Hrsg.)

ENTWERFEN ENTWICKELN **ERLEBEN**

Methoden und Werkzeuge in der Produktentwicklung

Entwickeln – Entwerfen – Erleben.
Methoden und Werkzeuge in der Produktentwicklung
10. Gemeinsames Kolloquium Konstruktionstechnik (KT2012)

Herausgeber:
Prof. Dr. Ralph Stelzer (Technische Universität Dresden)
Prof. Dr. Karl-Heinrich Grote (Otto-von-Guericke-Universität Magdeburg)
Prof. Dr. Klaus Brökel (Universität Rostock)
Prof. Dr. Frank Rieg (Universität Bayreuth)
Prof. Dr. Jörg Feldhusen (RWTH Aachen)

Wir bedanken uns für die Unterstützung bei
ma design, Tedata, Continental, xPLM, B.I.M. Consulting und Reiss Büromöbel

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.

ISBN 987-3-942710-80-0

Verlag der Wissenschaften GmbH
Bergstr. 70 | D-01069 Dresden
Tel.: 0351/47 96 97 20 | Fax: 0351/47 96 08 19
http://www.tudpress.de

Layout und Satz: Sandra Olbrich/Technische Universität Dresden.
Umschlaggestaltung: TU Dresden, Illustration Audi A6 Limousine © 2012 Audi AG

Markus Färber, Johannes Ghiletiuc,
Peter Schwarz & Beat Brüderlin

Echtzeit-Visualisierung von sehr großen Virtual- und Augmented-Reality-Szenen auf Smartphones und mobilen Tablet-Computern

1 Einführung und Motivation

Klassische VR/AR-Ausgabegeräte sind meist kopfgetragene Systeme oder 3D-Projektionsanlagen. In bestimmten Anwendungsfällen, zum Beispiel in der Wartung (Schreiber 2011) oder auch in Bauplanung und Bauausführung (Woodward 2011), verbieten sich diese Geräte aufgrund von Arbeitsschutzbestimmungen oder widrigen Einsatzbedingungen. Hier bieten sich Smartphones oder auf ähnlicher Technologie arbeitende Tablett-Computer als Alternative an, insbesondere dann, wenn stereoskopische Darstellung nicht benötigt wird.

In dieser Arbeit wird ein client-server-basiertes Renderingverfahren vorgestellt, das sehr große Polygonmodelle (im Bereich vieler Millionen bis einiger Milliarden an Dreiecken), wie sie aus komplexen CAD-Konstruktionszeichnungen oder Laserscan-Datensätzen abgeleitet werden, in Echtzeit darstellt. Bei diesem Verfahren entfällt die bisher notwendige Umwandlung und Detailreduzierung der Modelle, sodass alle in den Modellen enthaltenen Informationen bei Bedarf auf dem Client-System dargestellt werden können. Das Verfahren beansprucht nur geringe Ressourcen des Clients und stellt keine hohen Anforderungen an seine 3D-Grafikfähigkeiten.

Damit verbleibt genügend Verarbeitungskapazität für andere Aufgaben, wie zum Beispiel Tracking, Augmentierung der Daten, Interaktion, Datenkompression und ähnliches.

Die Interaktion mit dem Client-System erfolgt intuitiv mit Hilfe von Touchscreen-Gesten oder alternativ durch Bewegen des gesamten Geräts, wobei die Daten der Magnetfeld- und Beschleunigungssensoren ausgewertet werden.

2 Bildbasiertes Rendering mit Impostoren

Die Visualisierung einer Szene auf einem Rasterbildschirm ist immer ein approximierender Vorgang. Für jeden Bildpunkt wird ein Farbwert bestimmt, der alle Informationen wiederspiegelt, die auf dem Areal des Bildpunktquadrats von der virtuellen Kamera aus zu sehen sind. In der Regel berechnet ein Renderer auch den Abstand des Bildpunkts von der Kamera und speichert ihn in einem separaten Tiefenpuffer, um die Sichtbarkeit zu bestimmen. Die Komplexität des dargestellten Bildes ist konstant und daher für komplexe 3D-Szenen um mehrere Größenordnungen kleiner als die Szene selbst.

Weiterhin ist in vielen Fällen ein in einem Frame darzustellendes Abbild der Szene während der interaktiven Navigation ähnlich wie im vorherigen Frame. Diese zeitliche Kohärenz bewirkt meist, dass viele Szenenteile in beiden Bildern dargestellt werden und nur verhältnismäßig wenig neue Information visualisiert wird.

Immer dann, wenn das Erstellen eines Szenenabbildes sehr viel Aufwand in Anspruch nimmt und auch die Beleuchtung der Szene konstant bleibt, werden approximative Visualisierungstechniken, die die oben beschriebenen Tatsachen ausnutzen, interessant. Der in dieser Arbeit umgesetzte Ansatz beruht darauf, ein für eine Referenzkamerakonfiguration erstelltes 3D-Bild auch für nachfolgende Frames wiederzuverwenden. Dieses Referenzbild wird Impostor genannt, und es existieren viele 3D-Rendering-Systeme, die auf dieser Technik beruhen, zum Beispiel die von Décoret et al. (1999, 2003), Jeschke (2005) oder Wilson und Manocha (2003).

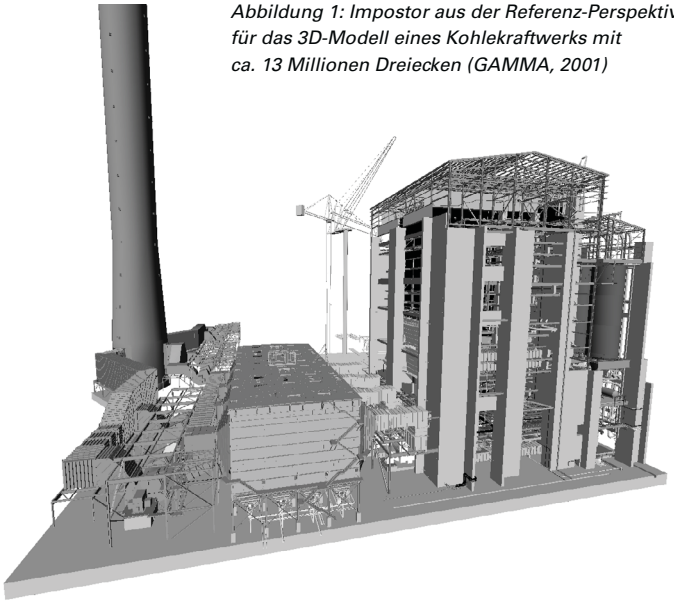


Abbildung 1: Impostor aus der Referenz-Perspektive für das 3D-Modell eines Kohlekraftwerks mit ca. 13 Millionen Dreiecken (GAMMA, 2001)

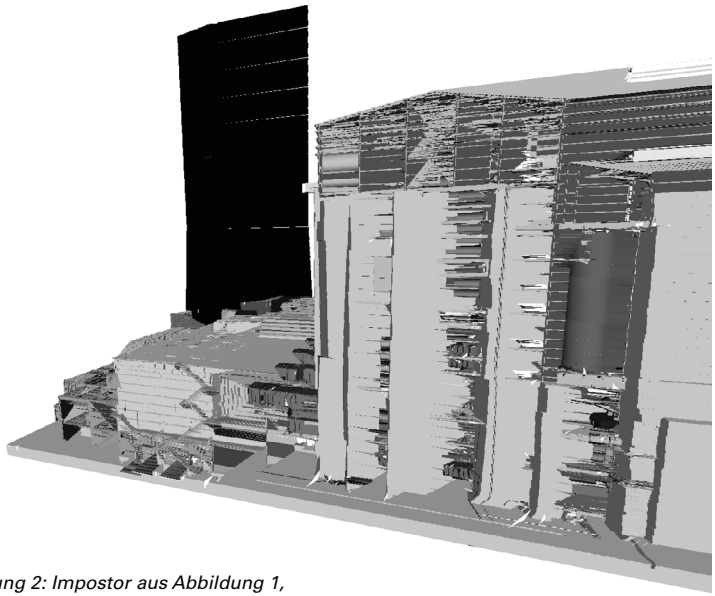


Abbildung 2: Impostor aus Abbildung 1, gesehen aus einer anderen Kamerakonfiguration

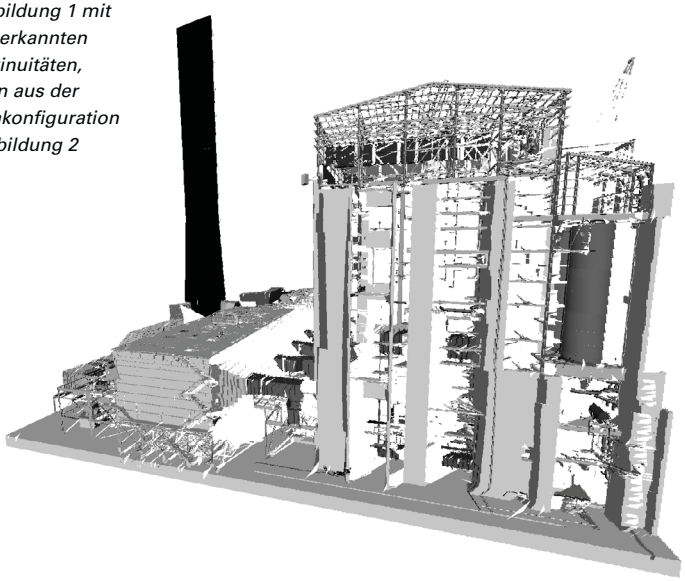
Die Grundidee ist hierbei, das Referenzbild und die zugehörigen Tiefenwerte in ein Dreiecksnetz im Objektraum umzuwandeln. Dieser 3D-Screenshot entspricht für die Referenzkamerakonfiguration exakt dem Original-Rendering (Abbildung 1).

Betrachtet man den Impostor aus einer anderen Kameraposition, so erkennt man, dass die Umwandlung des Referenzbildes in ein Dreiecksnetz an einigen Stellen Darstellungsfehler aufweist (Abbildung 2).

Man sieht viele lange, nach rechts gezogene Dreiecke (Skins), die Szenenteile miteinander verbinden, die in den Originaldaten nicht zusammenhängend sind. Diese Dreiecke befinden sich genau an den Stellen, an denen sich der Tiefengradient der Referenzansicht stark ändert (siehe hierzu auch Abbildung 6 weiter unten). An diesen Stellen muss das Impostor-Dreiecksnetz aufgetrennt werden. Dies kann zum Beispiel mit Hilfe von Kantenerkennungsoperatoren (Ghiletiuc et al. 2010) bewerkstelligt werden. Es zeigt sich jedoch, dass eine einfache Heuristik, die nur das Skalarprodukt von Referenz-Blickrichtung und aus den Tiefenwerten benachbarter Bildpunkten interpolierter Normalenrichtung gegen einen Schwellwert vergleicht, qualitativ bessere Resultate liefert. Abbildung 3 stellt den Impostor aus derselben Kameraposition und -orientierung dar wie Abbildung 2, jedoch wurde das Dreiecksnetz an den Unstetigkeitsstellen aufgetrennt. Die Originalszene wird dadurch wesentlich korrekter wiedergegeben.

In Abbildung 3 sieht man auch, dass nun leere Bereiche aufgedeckt werden, in denen sich Szenenteile befinden, die aus der Referenzansicht nicht sichtbar waren. Es existieren verschiedene Ansätze, diese fehlenden Szenenteile zu ergänzen (Ghiletiuc et al. 2011). Im Rahmen dieser Arbeit wird eine Impostor-Datenbank verwendet, die eine Anzahl bereits erstellter Impostoren zwischenspeichert und zur Anzeige bereithält. Jeder Impostor besteht aus Impostorfragmenten, und es wird in jedem Frame bestimmt, welche Fragmente aus allen in der Datenbank gehaltenen Impostoren für die aktuelle Kamerakonfiguration gültig sind. Diese werden dann gemeinsam dargestellt.

Abbildung 3: Impostor aus Abbildung 1 mit korrekt erkannten Diskontinuitäten, gesehen aus der Kamerakonfiguration von Abbildung 2



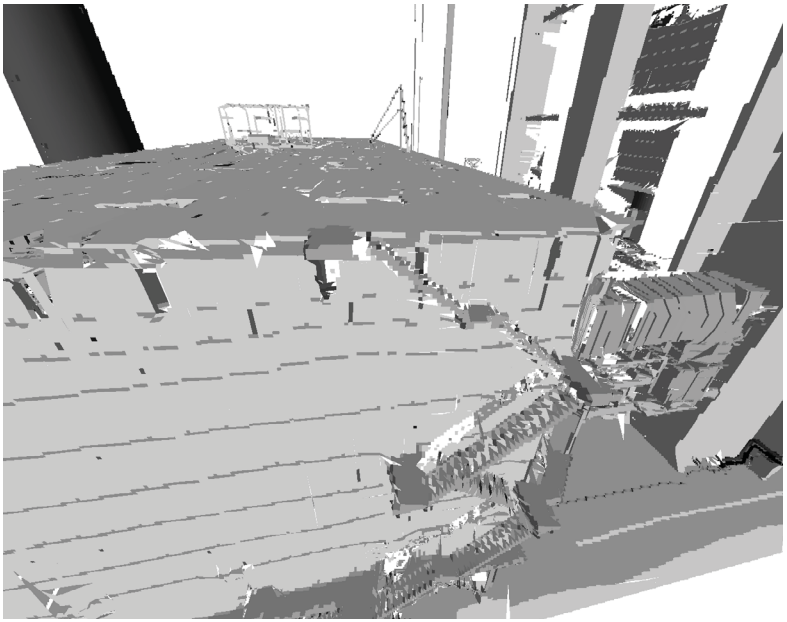


Abbildung 4: Visualisierung des Impostors aus Abbildung 1 aus zu naher Entfernung: Überabtastung des texturierten Dreiecksnetzes

Weiterhin ist die Darstellungsqualität des Impostors abhängig von der Entfernung der virtuellen Kamera. Insbesondere wirkt sich die Annäherung an den Impostor negativ aus. Während bei der Visualisierung des Originalmodells weitere Szenendetails sichtbar werden, bewirkt die Annäherung an den Impostor nur die Darstellung von Überabtastungsartefakten, da die hochaufgelöste Information im Impostor nicht verfügbar ist. Abbildung 4 zeigt hierfür ein Beispiel.

Der Auflösungsfehler lässt sich verringern bzw. ganz vermeiden, indem der gesamte Impostor für ungültig erklärt wird, wenn die aktuelle Kameraposition zu stark von der Referenzkameraposition abweicht. In diesem Fall wird ein neuer Impostor vom Server angefordert.

3 Architektur des Rendering-Systems

Das Rendering-System, genannt Incremental Impostor Streaming, besteht aus einem Server und einem Client, wie in Abbildung 5 dargestellt. Im Unterschied zu videostrombasierten Remote-Rendering-Systemen (Pajak et al. 2011) fordert das Incremental-Impostor-Streaming-System weniger Grafikleistung des Render-Servers, da

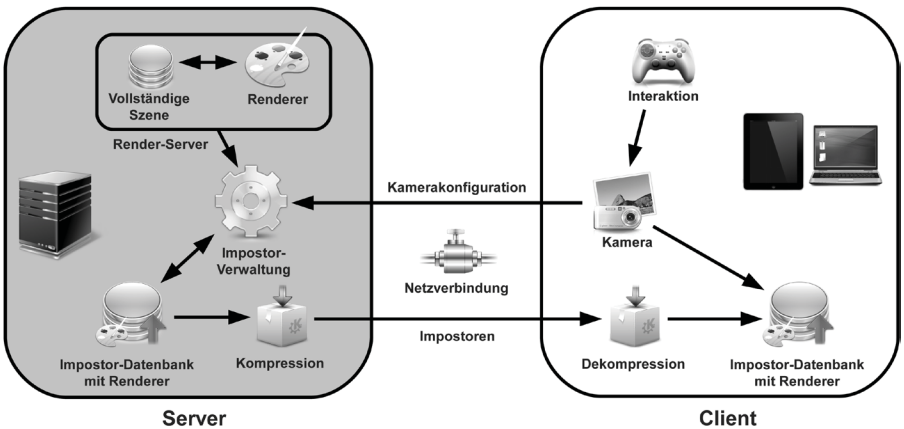


Abbildung 5: Architektur des Incremental-Impostor-Streaming-Systems

nicht in jedem Frame ein Bild erzeugt werden muss. Auch tritt keine Verzögerung zwischen Interaktion und Visualisierung auf, da eine eventuelle Interaktion des Nutzers direkt vom Client bearbeitet wird und nicht erst eine Antwort des Servers abgewartet werden muss. Das Impostor-Streaming-System eignet sich in erster Linie für statische Szenen.

3.1 Server

Auf dem Server werden die Ansichten aus Referenzkamerakonfiguration, die so genannten Impostoren, erstellt und in einer Datenbank verwaltet. Jeder Impostor wird in Form eines Dreiecksnetzes mit einer zugehörigen Textur (Textured Depth Mesh – TDM) gespeichert. Er gibt genau eine vollständige Ansicht der Szene aus einer bestimmten Kamerakonfiguration wieder, siehe Abbildung 6.

In der Wireframe-Darstellung, die mit dem Gradienten der Tiefenwerte unterlegt ist, kann man erkennen, dass das Dreiecksnetz nur in Bereichen hoher Tiefenkomplexität verfeinert wurde. Damit entsteht eine kompakte 3D-Repräsentation des Bildschirminhalts, deren Speicherbedarf durch Festlegung der maximalen Größe der kleinsten Dreiecke gesteuert werden kann. Die Impostorgröße wird entsprechend der Übertragungsbandbreite und der Verarbeitungsleistung des Clients angepasst, damit immer eine Mindestdarstellungsgeschwindigkeit bzw. -qualität erreicht wird. Die kleinsten Rechtecke des Dreiecksnetzes in Abbildung 6 bilden die so genannten Impostorfragmente.

Um einen Impostor zu erstellen, benötigt der Server ausschließlich Farb- und Tiefenpuffer der Szenenansicht. Diese werden mit Hilfe eines leistungsfähigen Visualisierungsservers (Brüderlin et al. 2006) erstellt, der als einziger Systembestandteil vollständigen Zugriff auf das gesamte Polygonmodell hat.

Alle zur Impostorerstellung notwendigen Algorithmen wurden auf der Grafikkarte implementiert. Einen Impostor zu erstellen dauert ca. 15 Millisekunden auf einer Grafikkarte des Typs NVIDIA GeForce GTX 470.

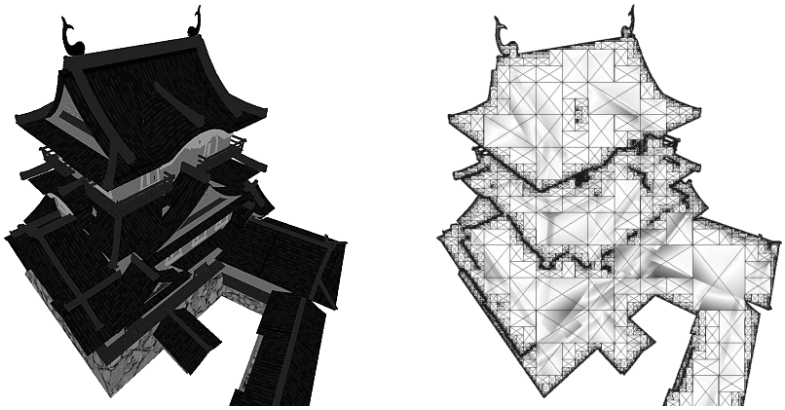


Abbildung 6: Impostor als texturiertes Dreiecksnetz (Textured Depth Mesh): Darstellung mit Texturierung (links) und Wireframe-Darstellung (rechts)

3.2 Client

Der Client wurde als Interactive-Walkthrough-Applikation realisiert und soll als Technologiedemonstrator für eine Virtual-und-Augmented-Reality-Lösung dienen. Das System besteht aus drei parallel laufenden Teilen: der kontinuierlichen Visualisierung der jeweils gültigen Impostorfragmente, dem Interaktionssystem zur Steuerung der virtuellen Kamera sowie der Impostordatenbank und -verwaltung mit angeschlossenem Netzwerk-Übertragungssystem. Der Client sendet die folgenden Informationen an den Server:

- Position der virtuellen Kamera
- Blickrichtung der virtuellen Kamera
- Andere Konfigurationsdaten der Kamera (Öffnungswinkel, Seitenverhältnis, Near und Far Plane)
- Dreiecksbudget für die Impostordatenbank auf Client-Seite

Der Client empfängt folgende Daten vom Server:

- Liste der Kennungen der jeweils anzuzeigenden Impostorfragmente
- Texturierte Dreiecksnetze, in Blöcken von Impostorfragmenten

Der Client ist allein für die Visualisierung des jeweils aktuellen Inhalts der Impostordatenbank zuständig. Die Datenbank wird kontinuierlich und asynchron vom Server mit neuen Impostoren versorgt, sofern sich die Kameraeinstellungen signifikant ändern. Zu jedem Zeitpunkt visualisiert der Client alle als gültig gekennzeichneten Impostorfragmente, wobei deren Gültigkeit basierend auf einer Reihe von Heuristiken auf Serverseite bestimmt und regelmäßig an den Client übertragen wird.

4 Implementierung des Smartphone-Clients

Die Implementierung des Client-Systems für Smartphones und Tablett-Computer basiert auf Android Version 2.2 und wurde in Java umgesetzt. Sie ist dafür ausgelegt, auf Geräten verschiedener Leistungklassen zu arbeiten. Die Lauffähigkeit auf dem Samsung Galaxy Tab ist nachgewiesen. Abbildung 7 zeigt ein Foto des auf diesem System laufenden Clients. Die visualisierten Impostoren bestehen aus ca. 500 Dreiecken. Diese Zahl wird im Mittel immer erreicht, unabhängig davon, aus wie vielen Dreiecken die Originalszene besteht.

4.1 Übertragung und Verwaltung der Impostoren

Die vom Server erstellten und dort in einer Datenbank gehaltenen Impostoren werden per WLAN an den Client übertragen. Hierzu wurde ein platzsparendes Übertragungsprotokoll entwickelt, das Dreiecksnetz und Textur getrennt überträgt, wobei ein verlustloser Kompressionsalgorithmus verwendet wird. Die Übertragung erfolgt in Blöcken von Impostorfragmenten.

Der aktuelle Arbeitssatz an Impostoren wird an den Client übertragen und im Hauptspeicher sowie im Flash-Speicher des Gerätes abgelegt, wobei ungültig gewordene Impostoren erst dann aus dem Speicher entfernt werden, wenn neu nachgeladene Impostoren den Platz beanspruchen. Die Impostorfragmente, die zum aktuellen Zeitpunkt visualisiert werden müssen, werden dann in den Grafikspeicher transferiert, visualisiert und weiter vorgehalten, bis sie durch neue Impostorfragmente ersetzt werden.

4.2 Rendering

Die verschiedenen Smartphones und Tablett-Computer besitzen unterschiedliche 3D-Grafikfähigkeiten. Ebenfalls unterscheiden sich die OpenGL-ES-Implementierungen der verschiedenen Android-Versionen. Aus diesem Grund wurden drei verschiedene Renderer implementiert, basierend jeweils auf OpenGL ES 1.0, OpenGL ES 1.1 und OpenGL ES 2.0. Rendering der Impostoren direkt aus dem Grafikspeicher (per Vertex Buffer Objects) wird verwendet, sofern es von Hardware und OpenGL-ES-Implementierung unterstützt wird.

Das Rendering selbst erfolgt auf die einfachste mögliche Weise: Es wird ausschließlich ein texturiertes Dreiecksnetz dargestellt. Alle Material- und Beleuchtungsinformationen werden ausschließlich auf dem Server berechnet. Sie sind in der Impostortextur enthalten. Der Renderer stellt in jedem Frame die jeweils gültigen Impostorfragmente dar. Zu Beginn sendet er die aktuelle Kamerakonfiguration (Position, Orientierung und andere Parameter, falls diese sich ändern) an den Server. Der Server antwortet umgehend mit einer Liste der Kennungen der für dieses Frame gültigen Impostorfragmente und beginnt, die auf dem Client nicht verfügbaren Impostoren zu senden. Gegebenenfalls müssen die Impostoren erst mit Hilfe des Render-Servers aus der Originalszene erzeugt werden.

Aufgrund der Kohärenz zwischen den verschiedenen Frames eines Interaktionsvorganges – Kameraposition sowie -orientierung ändern sich in der Regel nur in kleinen Schritten – bleibt ein Teil der Impostorfragmente gültig, und der Renderer beginnt sofort, diese darzustellen. Im Hintergrund empfängt ein Thread die neuen Impostoren, speist sie in die clientseitige Datenbank ein und stellt sie zum Rendern zur Verfügung, sodass sich das dargestellte Bild nach und nach vervollständigt. Impostoren sind aufgrund ihrer guten Approximationseigenschaften (Ghiletiuc et al. 2010) vergleichsweise lange gültig, sodass nur bei abrupter Kameraänderung ein völliger Neuaufbau des Bildes erforderlich ist.

4.3 Interaktion

Die Interaktion mit dem Client besteht im Wesentlichen aus einer Logik zur Steuerung der virtuellen Kamera. Zwei Steuerungsmög-

Abbildung 7: Incremental-Impostor-Streaming-Client auf einem Samsung Galaxy Tab: Darstellung des 3D-Modells eines Verkehrsflugzeugs vom Typ Boeing 777 (ca. 350 Millionen Dreiecke, Quell-3D-Daten zur Verfügung gestellt von The Boeing Company)

lichkeiten wurden implementiert, eine, die die Beschleunigungssensoren und den Magnetfeldsensor des Gerätes auswertet und einfache Zeigegesten ermöglicht, sowie eine Touchpad-Steuerung.

Mit Hilfe der in Android eingebauten Auswertungsalgorithmen wird bei der sensorbasierten Steuerung die Orientierung der Kamera im Raum bestimmt. Indem man das Smartphone rechtwinklig zur Erdoberfläche hält und nach links oder rechts dreht, wird die Blickrichtung der Kamera geändert, siehe Abbildung 8. Die Änderung der Kameraposition erfolgt durch Berühren des Touchpads (Abbildung 9). Inspiriert ist diese Art der Interaktion vom Kompassmodus in Google Maps Street View für Android (Google 2008, 2011).

Die von den Sensoren gelieferten Werte sind, abhängig von der Einsatzumgebung, deutlichen Schwankungen unterworfen. Insbesondere im Innenraum kann die Ausgabe des Magnetfeldsensors sehr stark gestört sein. Aus diesem Grund werden die Sensorwerte in einem parallel zum Renderer laufenden Thread mit sehr viel höherer Wiederholrate erfasst und mit Hilfe einer rollenden Durchschnittsberechnung geglättet. Die Glättung wird unterbrochen, sobald eine sehr große sprunghafte Änderung der Sensorwerte gemessen wird, mit dem Ergebnis, dass durch die Glättung schnelle Interaktionen nicht unnötig verzögert werden.

Zusätzlich ist eine reine Touchpad-Steuerung der Kamera implementiert. Einfache Wischgesten bewirken die Rotation der Kamera um die globale x- bzw. y-Achse, während die Verschiebung der Kameraposition mit Hilfe eines unsichtbaren Tastenkreuzes auf dem Touchscreen geschieht. Mit zwei Fingern ausgeführte Gesten bewirken die Bewegung der Kamera in Blickrichtung. In den Abbildungen 9 und 10 ist die Touchpad-Steuerung symbolisch dargestellt. Zur Erleichterung der Orientierung erlaubt das System die Darstellung einer Skybox.

5 Zusammenfassung und Ausblick

Es wurde ein Client-Server-System vorgestellt, mit dem sehr große Polygonmodelle, wie sie aus komplexen CAD-Modellen oder Laserscan-Datensätzen generiert werden, auf Smartphones und Tablett-Systemen dargestellt werden können. Das System kann als Basistechnologie für anspruchsvolle Virtual- und Augmented-Reality-Systeme dienen, da es nur geringe Ressourcen des Clients in Anspruch nimmt und dennoch hochdetaillierte Modelldaten visualisieren kann.

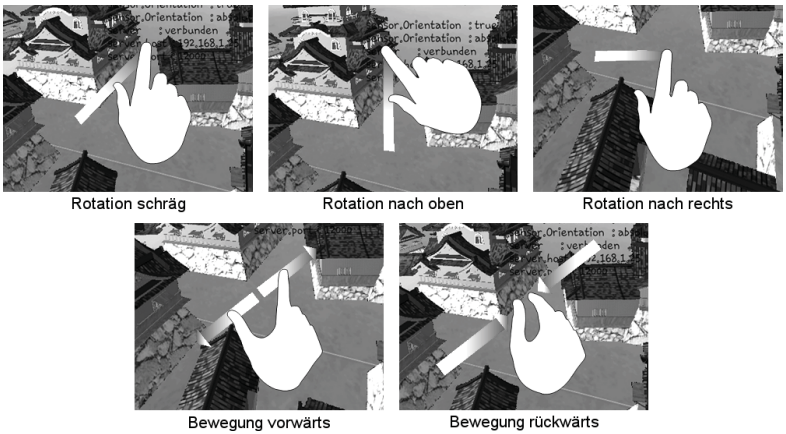
Während auf dem Server auf Anforderung so genannte Impostoren, das sind 3D-Screenshots der darzustellenden Szene, erstellt und verwaltet werden, visualisiert der Client diese Impostoren kontinuierlich und erlaubt so die interaktive Navigation durch die Szene. Einmal generierte Impostoren werden während der Interaktion wiederverwendet, sofern die Einstellungen der virtuellen Kamera nicht zu stark von der Referenzkonfiguration abweichen. Das Impostor-Rendering-Konzept nutzt die Tatsache, dass ein 3D-Screenshot nur einen Bruchteil des Speicherbedarfs und des Visualisierungsaufwandes der Gesamtszene hat und dass während der Navigation die meisten Frames zeitlich kohärent sind.

Das Impostor-Streaming-System wird stetig weiterentwickelt. Wichtigstes Ziel ist die Qualitätsverbesserung und damit Verlängerung der Wiederverwendungsdauer der Impostoren, zum Beispiel durch mehrschichtige texturierte Dreiecksnetze (Ghiletiuc et al. 2011) und intelligente Anordnung der Referenzkamerakonfigurationen. Langfristiges Ziel ist die Ertüchtigung des Systems für dynamische Szenen.

Abbildung 8 (oben): Rotation der virtuellen Kamera durch Veränderung der Geräteposition im Raum

Abbildung 9 (Mitte): Bewegung der virtuellen Kamera nach links, rechts, oben und unten durch Berühren des Touchscreens

Abbildung 10 (unten): Rotation u nd Bewegung in Blickrichtung der virtuellen Kamera per Wischgesten auf dem Touchscreen



Danksagung

Diese Arbeit wurde gefördert von der Graduiertenschule Bildverarbeitung und Bildinterpretation im ProExzellenz-Programm des Thüringer Ministeriums für Bildung, Wissenschaft und Kultur.

Literaturverzeichnis

Brüderlin, B., Heyer, M., Pfützner, S., 2006: Visibility-guided Rendering to Accelerate 3D Graphics Hardware Performance. In: Kasik, D., Manocha, D., Stephens, A. et al.: Real-time Interactive Massive Model Visualization, Eurographics 2006 Course Notes, Wien.

Décoret, X., Durand, F., Sillion, F. X., Dorsey, J., 2003: Billboard Clouds for Extreme Model Simplification. In: SIGGRAPH '03 Papers, 689–696.

Décoret, X., Sillion, F., Schaufler, G., Dorsey, J., 1999: Multi-layered Impostors for Accelerated Rendering. In: Computer Graphics Forum, 18(3), 61–73.

Ghiletiuc, J., Färber, M., Brüderlin, B., 2010: A Highly Scalable Image-Based Remote Rendering Framework. In: 9. Paderborner Workshop: Augmented und Virtual Reality in der Produktentstehung, 317–330, Paderborn.

Ghiletiuc, J., Färber, M., Brüderlin, B., 2011: Bildbasiertes Echtzeit-3D-Rendering mit Hilfe von mehrschichtigen Impostoren. In: 10. Paderborner Workshop: Augmented und Virtual Reality in der Produktentstehung, 171–184, Paderborn.

GAMMA Research Group, University of North Carolina, 2001: Power Plant Model. http://gamma.cs.unc.edu/POWERPLANT, 19.03.2001, abgerufen am 10.02.2012.

Google Inc., 2011: Google Maps für Handys in 3D. http://www.google.de/mobile/maps, abgerufen am 8.1.2012.

Google Inc., 2008: Google Maps on Android. http://www.youtube.com/watch?v=_YFw9p0TjT8, 30.09.2008, abgerufen am 8.1.2012

Heyer, M., Pfützner, S., Brüderlin, B., 2005: Visualization Server for Very Large Virtual Reality Scenes. In: 4. Paderborner Workshop: Augmented und Virtual Reality in der Produktentstehung, Paderborn.

Jeschke, S., 2005: Accelerating the Rendering Process using Impostors. Dissertation, Technische Universität Wien.

Klein, G., Murray, D., 2009: Parallel Tracking and Mapping on a Camera Phone. In: Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR 2009), Orlando, Florida.

Pajak, D., Herzog, R., Eisemann, E., Myszkowski, K., Seidel, H.-P., 2011: Scalable Remote Rendering with Depth and Motion-flow Augmented Streaming. In: Computer Graphics Forum, 30(2), 415–424.

Papagiannakis, G., Singh, G., Magnenat-Thalmann, N., 2008: A Survey of Mobile and Wireless Technologies for Augmented Reality Systems. In: Computer Animation and Virtual Worlds, 19(1), 3–22.
Schmalstieg, D., Wagner, D., 2007: Experiences with Handheld Augmented Reality. In: Proceedings of the International Symposium on Mixed and Augmented Reality (ISMAR 2007), Nara, Japan.
Schreiber, W., Zimmermann, P. (Hrsg.), 2011: Virtuelle Techniken im industriellen Umfeld: Das AVILUS-Projekt – Technologien und Anwendungen. Berlin: Springer.
Wilson, A., Manocha, D., 2003: Simplifying Complex Environments using Incremental Textured Depth Meshes. In: SIGGRAPH '03 Paper, 678–688.
Woodward, C., Hakkarainen, M., 2011: Mobile Augmented Reality System for Construction Site Visualization. In: Proceedings of the ISMAR 2011 Workshop on Enabling Large-Scale Outdoor Mixed Reality and Augmented Reality, Basel.

Kontakt

Dipl.-Inf. Markus Färber
Markus.Faerber@tu-ilmenau.de

Dipl.-Inf. Johannes Ghiletiuc
Johannes.Ghiletiuc@tu-ilmenau.de

Peter Schwarz
Peter.Schwarz@tu-ilmenau.de

Prof. Dr. sc. techn. Beat Brüderlin
Beat.Brüderlin@tu-ilmenau.de

Technische Universität Ilmenau
Fachgebiet Graphische Datenverarbeitung
Postfach 100 565
98684 Ilmenau
www.tu-ilmenau.de/gdv

Graduiertenschule Bildverarbeitung und Bildinterpretation
www.tu-ilmenau.de/gs-ipii