Technische Universität Dresden

Fakultät Informatik
Institut für Software- und Multimediatechnik
Professur für Computergraphik und Visualisierung
Prof. Dr. Stefan Gumhold

Diplomarbeit

zur Erlangung des akademischen Grades
Diplom-Informatiker

Optimierung und Übertragung von Tiefengeometrie für Remote-Visualisierung

Josef Schulz
(Geboren am 20. Oktober 1989 in Naumburg (Saale), Mat.-Nr.: 3658867)

Betreuer: Dr. Sebastian Grottel

Dresden, 7. November 2016

Aufgabenstellung

In Big-Data-Szenarien in der Visualisierung spielt der Ansatz der Remote-Visualisierung eine zunehmende Rolle. Moderne Netzwerktechnologien bieten große Datenübertragungsraten und niedrige Latenzzeiten. Für die interaktive Visualisierung sind aber selbst kleinste Latenzzeiten problematisch. Um diese vor dem Benutzer maskieren zu können, kann eine Extrapolation der Darstellung durchgeführt. Diese Berechnungen erfordern zusätzlich zum normalen Farbbild weitere Daten, beispielsweise ein Tiefenbild und die Daten der verwendeten Kameraeinstellung. Für die Darstellungsextrapolation werden Farb- und Tiefenbild zusammen interpretiert, beispielsweise als Punktwolke oder Höhenfeldgeometrie. Im Rahmen dieser Arbeit soll untersucht werden, wie die Darstellung mittels Höhenfeldgeometrie optimiert werden kann. Ansätze sind hierfür Algorithmen aus der Netzvereinfachung. Zu erwarten sind sowohl harte Kanten als auch glatte Verläufe der Tiefenwerte, welche sich in der Netzgeometrie durch adaptive Vernetzung mit reduziertem Datenaufwand darstellen lassen.

Dem Szenario der Web-basierten Remote-Visualisierung folgend soll der Web-Browser als Klient-Komponente eingesetzt werden. Die einzusetzenden Technologien sind HTML5, Javascript, WebGL und WebSockets. Entsprechende Javascript-Bibliotheken sollen genutzt werden um die Qualität und Wartbarkeit des Quellcodes zu steigern. Für die Server-Komponente darf die Technologie vom Bearbeiter frei gewählt werden.

Zu Beginn der Arbeit wird eine Literatur-Recherche zu Web-basierter Visualisierung und Remote-Visualisierung erfolgen. Schwerpunkte sind hierbei die Bild-Extrapolation, Vernetzung und Rekonstruktion auf Basis von Tiefenbildern und die Netzoptimierung und -Vereinfachung. Im Anschluss an die Literaturrecherche wird ein Konzept für die Implementierung mit dem Betreuer abgesprochen und anschließend als prototypische Software umgesetzt. Folgendes Szenario dient als Grundlage für dieses Konzept:

Als Eingabedaten stehen mehrere Datensätze aus unterschiedlichen Szenarien der wissenschaftlichen Visualisierung zur Verfügung. Für jeden Datensatz sind mehrere Tripel aus Farbbild, Tiefenbild und Kamera-Parameter gegeben. Die Serverkomponente bereitet einen Datensatz auf und bietet ihn dem Klienten an. Diese Aufbereitung ist vor allem die Generierung einer optimierten Tiefennetzgeometrie aus den Tiefenbilddaten. Der Klient fordert Farbbilder, Kameraeinstellungen und Tiefengeometrie von Tripel-Paaren an. Konzeptuell wird ein Tripel als aktueller Zustand und das zweite Tripel als Ground-Truth einer Bildextrapolation verstanden. Diese können daher auch in dieser Reihenfolge angefordert werden. Die Tripel werden zwischen

Klient und Server direkt per Sockets/WebSockets übertragen. Die Daten des ersten Tripels werden anschließend genutzt um dessen Farbbild in die Ansicht des zweiten Tripels extrapoliert. Hierbei werden vom zweiten Tripel nur die Kameraeinstellung genutzt. Diese Extrapolation wird Klient-seitig in WebGL implementiert damit alle Berechnungen auf der GPU ausgeführt werden. Anschließend wird das extrapolierte Bild mit dem originalen Ground-Truth-Farbbild aus dem zweiten Tripel verglichen um die Qualität der Extrapolation zu bewerten, z.B. durch SSIM.

Die umgesetzte Lösung wird ausführlich evaluiert. Zentraler Wert ist hierbei die Bildqualität nach der Extrapolation abhängig vom Winkelunterschied zwischen den Kameraeinstellungen und den Parametern der Vereinfachung der Tiefennetzgeometrie. Hierfür werden Tripel-Paare aus den Datensätzen und Variationen der Parameter der Algorithmen systematisch und automatisiert vermessen. Untersuchungen zum Laufzeitverhalten der Netzoptimierung im Server und der Bildextrapolation im Klienten sind optional durchzuführen.

Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die von mir am heutigen Tag dem Prüfungsausschuss der Fakultät Informatik eingereichte Arbeit zum Thema:

Optimierung und Übertragung von Tiefengeometrie für Remote-Visualisierung

vollkommen selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Dresden, den 7. November 2016

Josef Schulz

Kurzfassung

Zusammenfassung Text Deutsch

Abstract

abstract text english

Inhaltsverzeichnis

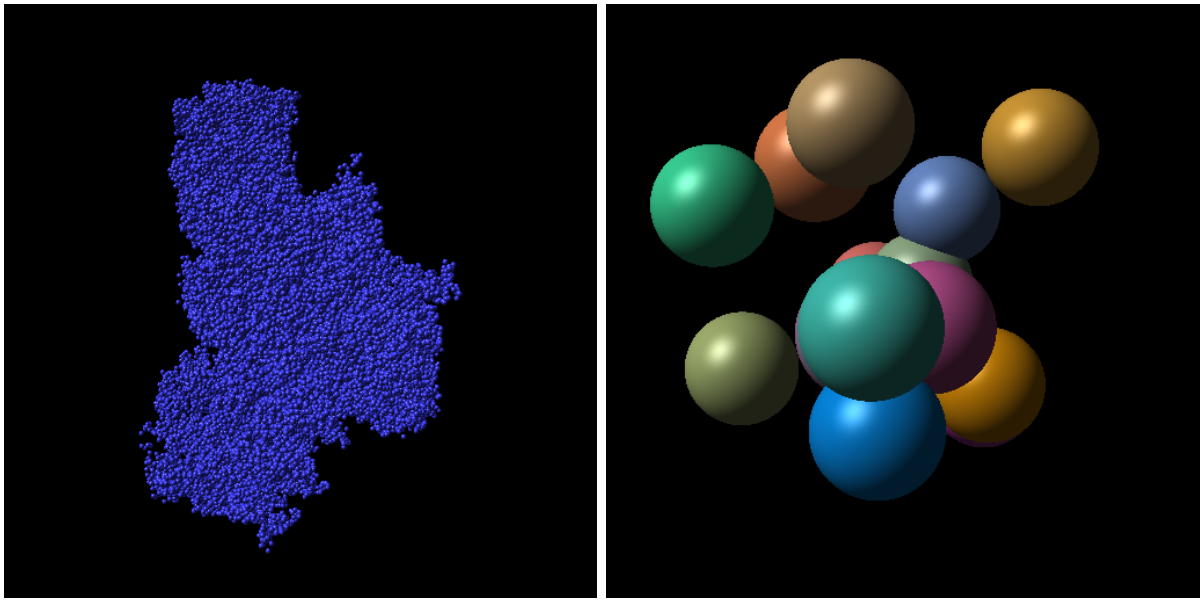
1 Einleitung

Bei der Remote-Visualisierung, wird die Bildsynthese und die eigentliche Darstellung voneinander getrennt. Der Server-Prozess erzeugt und kodiert jedes Bild zu einem kompakten Datenpaket, welches an den Klient-Prozess gesendet wird. Der Klient dekodiert die Informationen und gibt sie auf einem Bildschirm aus.

Remote-Visualisierung ist ein insbesondere für mobile Endgeräte interessantes Konzept, weil es die Visualisierung von komplexen Szenen auch auf Leistungsarmen Geräten ermöglicht. Neben Computerspielen, ist die wissenschaftliche Visualisierung ein wichtiges Anwendungsgebiet, da Datensätze Größenordnungen erreichen können, die den Speicher herkömmlicher Desktops, Laptops, Smartphones etc. bei weitem übersteigen. Auch wenn ausreichend Speicher zur Verfügung steht, kann die Übertragung dieser Daten viel Zeit in Anspruch nehmen.

Mit Hilfe der Remote-Visualisierung ist es möglich, dass der Server die Bildsynthese übernimmt und nicht der komplette Datensatz übertragen werden muss. Ein weiterer Vorteil der mit leistungsstarken Serversystemen einhergeht ist der, dass sich komplexe Visualisierungs- und Beleuchtungsmethoden verwenden lassen, die mit normalen Endgeräten nicht zu realisieren sind.

Die Latenz bezeichnet in der Netzwerktechnik die Übertragungszeit von einem zum anderen Gerät. Diese ist in modernen Netzwerken gering, für die Interaktive Visualisierung allerdings immer noch zu Groß um eine für den Menschen nicht wahrnehmbare Verzögerungsfreie interaktive Visualisierung zu gewährleisten. Ein möglicher Ausweg besteht darin, dass der Klient-Prozess ein bereits empfangenes Bild extrapoliert. Auf diese Weise, ist es möglich, dass der Klient-Prozess bereits ein neues Bild ausgeben kann, obwohl es noch nicht empfangen wurde. Bei der Bildsynthese des Server-Prozesses entsteht neben dem Farbbild zusätzlich ein Tiefenbild. Dieses kann genutzt werden, um geometrische Informationen an den Klient-Prozess weiter zureichen. Geometrisch entspricht das Tiefenbild einer 2.5D Ansicht der Szene, in Form einer Punktwolke. Die Extrapolation eines Bildes wird durchgeführt, indem diese Informationen aus einer neuen Kameraperspektive gezeichnet wird. Um die Qualität zu verbessern und Informationen einzusparen, wird aus dem Tiefenbild ein Dreiecksnetz erzeugt. Das Farbbild wird als Textur über das Netz gelegt. In dieser Arbeit werden Methoden zur Erzeugung von Dreiecknetzen vorgestellt und mit



(a) CoolRandom (b) TestSpheres

Abbildung 1.1: Datensätze

Hilfe von sechs Datensätzen evaluiert. Jeder Datensatz entspricht einer Kamerafahrt, durch eine Szene. Zur Verfügung standen die zwei Szenen, von denen jeweils ein Bild, eines Datensatzes, in der Abbildung 1.1 die beiden Szenen exemplarisch vorstellt.

Um die Algorithmen zu testen wurde eine Server- und eine Klient-Komponente entwickelt. Beide Komponenten tauschen Informationen über das auf TCP basierende Websocket-Protokoll aus. Im Gegensatz zum HTTP-Protokoll ist die Kommunikation bidirektional, der Vorteil besteht darin, dass sich beide Kommunikationsteilnehmer ohne *long-polling* direkt Nachrichten zu schicken können. Der Klient ist ein Browser basierter Web-Klient, damit die implementierten Algorithmen mit den Einschränkungen durch JavaScript und WebGl evaluiert werden können.

Zu diesem Zweck wurde eine Server- und eine Klient-Komponente entwickelt, die mit Hilfe des WebSocket-Protokolls Daten untereinander austauschen. Der Klient basiert auf JavaScript und nutzt für die Bildextrapolation WebGl. Die Qualität der Darstellungen werden mit Hilfe von Ground-Truth-Daten überprüft. Zum Vergleich wird der PSNR und der SSIM [WBSS04] der Bildpaare bestimmt und evaluiert.

Im Folgenden werden existierende Konzepte und Ideen vorgestellt.

2 Verwandte Arbeiten

Einen Überblick über Architekturen und Methoden der interaktiven Remote-Visualisierung geben Shu Shi et al. [SH15]. Zum zentralen Problem ihrer Arbeit, wird die Latenz und die effiziente Übertragung der Daten vom Server zum Klient. Lösungen hängen vom Anwendungsfall ab, in *THIN*-Systemen besteht die Aufgabe in der Übertragung von 2D Informationen, mit denen sich zum Beispiel Desktop-Anwendungen Fernsteuern lassen. Als Beispiele dienen SLIM [SLN99] und THiNC [BKN05]. Die Übertragung der Daten wurde in beiden Systemen für den Einsatz von 2D Grafiken optimiert. Ein Vorteil bei diesen Anwendungen besteht darin, dass nur relativ kleine Änderungen tatsächlich übertragen werden müssen, wenn sich zum Beispiel ein Fenster ändert muss auch nur diese Änderung zum Klient übertragen werden. Zur Verbesserung der Latenz wird von Shu Shi et al. die Bildextrapolation vorgeschlagen. Diese kann durch die zusätzliche Übertragung von Tiefenbildern oder Dreiecksnetzen erreicht werden. Bei der Bildsynthese auf dem Klient-System, entsteht eine Menge von Artefakten, diese kann durch Verzerrung und Verformung der Geometrieinformationen verkleinert werden, durch sogenanntes *Warping* [BG04], [SLBF09]. Wenn eine Extrapolation der Bilder, durch den Klient, aus Hardware-Gründen nicht möglich ist, können auch die nächst möglichen Bewegungen geschätzt werden und deren Ergebnisse werden mit übertragen. Zur Verbesserung der Qualität werden deshalb in vielen Anwendungen die Bewegungsmöglichkeiten des Nutzers auf Pfade oder Aussichtspunkte beschränkt.

2.1 Architektur

Ein spezielles Remote-Visualisierungssystem haben Peter Eisert und Philipp Fechteler für Computerspiele entwickelt [Eis07]. Ihr System kommt ohne Beschränkung der Bewegungsfreiheit aus. Sie haben sich dafür zwei Ansätze zunutze gemacht. Im ersten Ansatz erzeugt der Server die fertigen Bilder, welche mit Video-Codecs codiert und an den Klient gestreamt werden. Im zweiten Ansatz, werden Zeicheninstruktionen an den Klient gestreamt, welcher mit diesen das Ergebnissbild produziert. Ihr System ist für den Einsatz im lokalen Netzwerk konstruiert.

Wessels et al. stellen eine Konzeption für den Programmaufbau eines interaktiven Remote-

Visualisierungssystem basierend auf dem WebSocket-Protokoll vor [WPJR11]. In ihrem System besteht der Server-Prozess aus zwei Hauptkomponenten, der Visualisierungs-Engine und dem Deamon. Während die Visualisierungs-Engine für die Bildsynthese zuständig ist, übernimmt der Deamon die Kommunikation mit dem Klient-Prozess. Der Klient schickt dabei seine Eingabeinformationen von Maus und Tastatur direkt an den Server. Dieser wertet die Daten aus und erzeugt darauf hin ein mit JPEG komprimiertes Bild, das mit Base64 kodiert wird und schließlich an den Klient-Prozess geschickt wird. Dieser kann das Bild nativ mit Hilfe eines HTML5 Canvas dekodieren und darstellen. Ihr System wird zur Grundlage dieser Arbeit.

2.2 Extrapolation

Die Grundidee das Problem der Latenz mit Bildexploration in den Griff zu bekommen wurde in der Arbeit von Palomo et al. betrachtet [PG10].

Das erzeugte Tiefenbild lässt sich Pauly et al. [PGK02]

Simon Stegmaier [SME02] A Generic Solution for Hardware-Accelerated Remote Visualization

2.3 Kompression

Gabriel Taubin und Jarek Rossignac haben ein Algorithmus zur Erzeugung und effizienten Kodierung von Dreiecksstreifen aus Dreiecksnetzen entwickelt [TR98]. Dazu konstruiert ihr Algorithmus Spannbäume über dem Netz, die zur Erzeugung möglichst großer Dreiecksstreifen genutzt werden. Die Kompression kann wahlweise verlustfrei oder verlustbehaftet durchgeführt werden. Typische Kompressionsraten werden mit 1:50 angegeben.

Eine weitere Arbeit die sich mit der Kompression von Dreiecksnetzen und einer kompakten Repräsentation von diesen beschäftigt wurde Stefan Gumhold und Wolfgang Straßer geschrieben [GS98]. Kompression und Dekompression sind echtzeitfähig.

Michael Deering hat ebenfalls ein Geometrisches Kompressionsverfahren entwickelt. [Dee95]

Federico Ponchio und Matteo Dellepiane [PD15] Fast decompression for web-based view-dependent 3D rendering

Diplomarbeit mädcheninformatiker Effiziente Datenübertragung von Modellen und Texturen für die Verwendung in WebGL Stefan Wagner noch kein cite

WEB-BASED VISUALISATION OF ON-SET POINT CLOUD DATA Alun Evans et al [EAB14]

planarer scheiß [MWB+13]

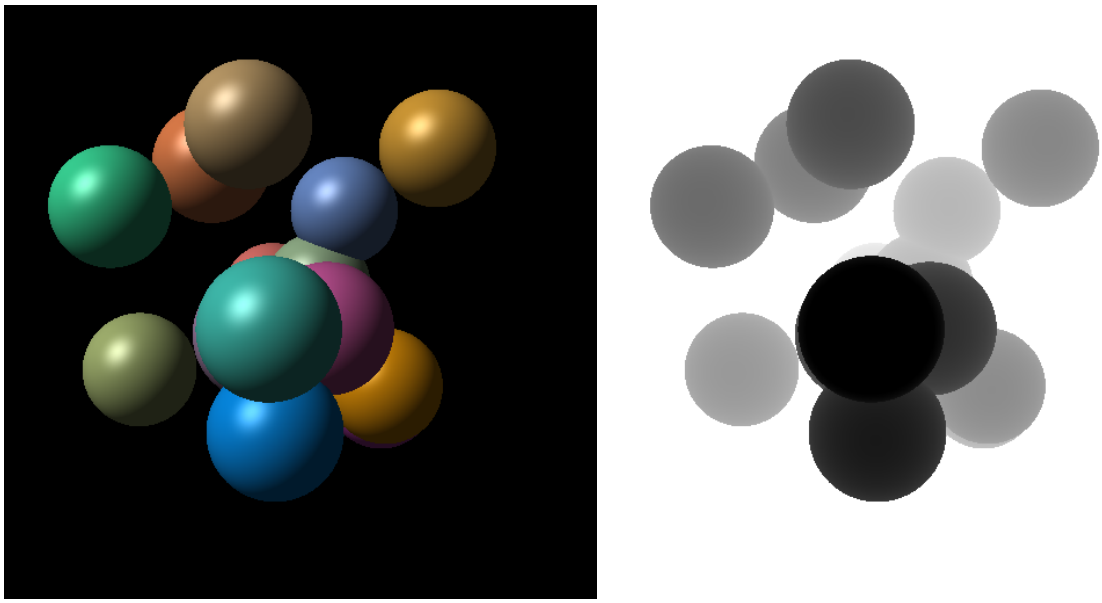
Ein Remote-Visualisierungssystem als ein verteiltes System betrachtet werden. Zhefan Jin stellt drei Klassen solcher Systeme vor [Jin06]. Diese unterscheiden sich anhand der Verteilung ihrer Daten auf unterschiedliche Host-Systeme. Dabei können Zeicheninstruktionen, Attribute, oder Bilder von einem Knoten zum anderen weiter gereicht werden. Auch bei einfache Remote-Visualisierungssysteme unterscheiden sich an Hand der Daten die übertragen werden.

3 Grundlagen

In diesem Kapitel werden die Datensätze im Detail vorgestellt und es wird gezeigt, wie die Bildextrapolation anhand der zur Verfügung stehenden Informationen durchgeführt wird. Am Ende dieses Kapitels werden zwei Metriken zum Vergleich von Bildern vorgestellt und diskutiert.

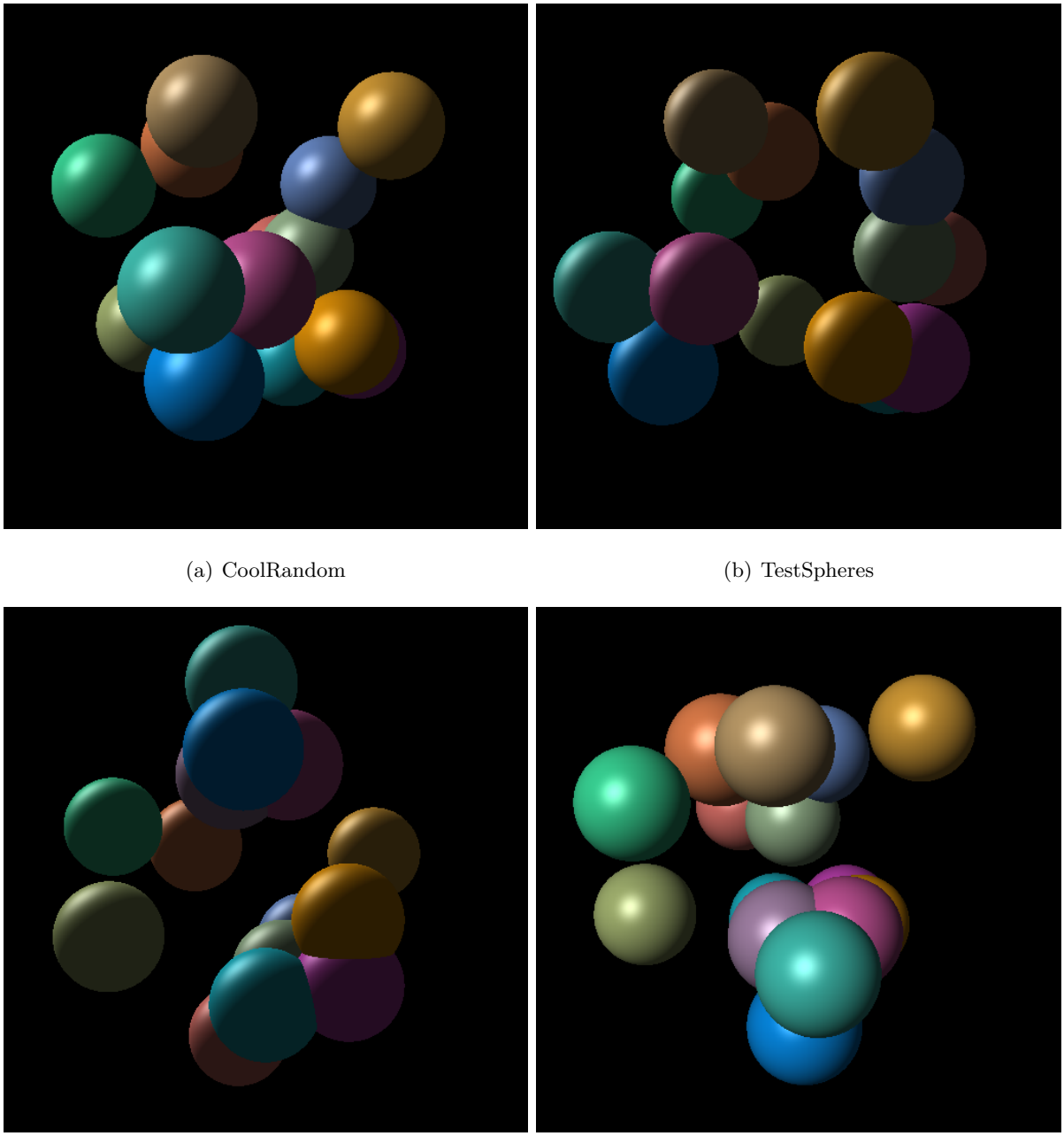
3.1 Datensätze

Zur Analyse der verwendeten Methoden stehen zwei Szenen zur Verfügung. Mit beiden Szenen wurden jeweils drei Kamerafahrten aufgenommen.



(a) CoolRandom (b) TestSpheres

Abbildung 3.1: Datensätze



(a) CoolRandom (b) TestSpheres

(c) CoolRandom (d) TestSpheres

Abbildung 3.2: Datensätze

#	Szene	Kamerafahrt	Auflösung	Anzahl Bilder	Maximaler Winkel
1	CoolRandom	1	512×512	241	5
2	CoolRandom	2	512×512	60	10
3	CoolRandom	3	512×512	192	90
4	TestSpheres	1	512×512	241	5
5	TestSpheres	2	512×512	60	10
6	TestSpheres	3	512×512	192	90

Tabelle 3.1: In der Tabelle sind alle Datensätze, die für die Evaluation zur Verfügung stehen aufgelistet.

3.2 Extrapolation

Bei dem Zeichenvorgang auf dem Server entsteht ein Farbbild und ein Tiefenbild. Das Tiefenbild wurde mit einer Farbtiefe von 16bit erzeugt und ist die Ausgangsbasis für die Bildextrapolation. Da aus diesem ein Dreiecksnetz erzeugt wird, welche das Farbbild als Textur verwendet, und zur Erzeugung neuer Ansichten genutzt wird. Um das Verfahren besser zu verständlich zu machen, wird zunächst die Vertex-Transformation näher betrachtet, zu Verdeutlichung der Zusammenhänge dient die folgende Abbildung:

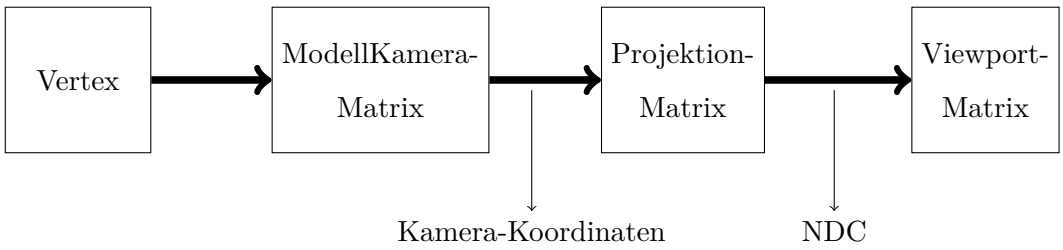


Abbildung 3.3: Die Abbildung zeigt die Transformationspipeline, mit deren Hilfe die Vertices auf den Bildschirm abgebildet werden.

Die Eingabe der Transformationspipeline ist ein Vertex. Jeder Vertex besteht aus einer x-, y- und einer z-Komponente. Die Vertices sind im Modell-Koordinatensystem definiert, das bedeutet, dass der Ursprung dieses Koordinatensystems das Zentrum des Modells ist. Durch die Multiplikation mit der Modell-Matrix werden die Vertices in das Weltkoordinatensystem projiziert. Um die Koordinaten aus dem Weltkoordinatensystem in das Kamera-Koordinatensystem abzubilden genügt die Multiplikation mit der Kameramatrix. In der Abbildung 3.2 wurde die Modell- und die Kamera-Matrix zur ModellKamera-Matrix zusammengefasst. Werden die Ver-

tices, die sich im Kamerakoordinatensystem befinden mit der Projektionsmatrix multipliziert, dann werden diese in *normalized-device-coordinates*, kurz NDC umgewandelt. Normalisierte-Geräte-Koordinaten haben für die x-,y- und z-Komponente den Wertebereich von -1.0 bis 1.0. Durch die Multiplikation mit der Viewport-Matrix werden die x- und y-Komponente auf die Bildschirmkoordinaten abgebildet.

Um aus dem Tiefenbild normalisierte Gerätekoordinaten zu erhalten, müssen die x- und y-Koordinaten der Pixel durch die Auflösung geteilt werden. Die Farbwerte des Tiefenbildes liegen im Bereich von 0.0 bis 1.0. Je größer der Wert ist, umso weiter ist ein Pixel von der Bildschirmebene entfernt. Die Farbwerte müssen ebenfalls in den Wertebereich von -1.0 bis 1.0 überführt werden.

Damit aus den normalisierten Gerätekoordinaten wieder Modellkoordinaten werden, reicht es aus diese mit der invertierten ModellKameraProjektionsmatrix zu multiplizieren. Die daraus resultierenden Vertices, lassen sich erneut mit der Transformationspipeline abbilden, auf diese Weise lassen sich die Koordinaten des einen Bildes in die eines anderen überführen.

3.3 Delaunay-Triangulierung

Die Delaunay-Triangulierung ist ein Verfahren um ein Dreiecknetz aus einer Menge von Punkten $p \in \mathbb{R}^2$ zu erzeugen. Dabei wird für jedes Dreieck ein Umkreis erzeugt, innerhalb dessen keine Punkte eines anderen Dreiecks enthalten sein dürfen. Jedes Dreieck des zu erzeugenden Netzes muss diese Bedingung erfüllen. Das Resultat dieser Forderung ist die maximierte Innenwinkelsumme aller Dreiecke. Für eine gegebene Punktmenge, ist die Lösung nicht eindeutig, es kann verschiedene Netzkonfigurationen geben, welche die Forderung erfüllen.

Es existieren verschiedene Algorithmen die Delaunay-Triangulierung durchzuführen, die besten erreichen eine Laufzeit von $O(n \log n)$ und sind damit tauglich für den Einsatz in Echtzeitanwendungen. Beispiele sind der Sweep-Algorithmus und die inkrementelle Konstruktion.

3.4 PSNR

Die Abkürzung PSNR in Englisch *Peak signal-to-noise ratio*, gibt das Verhältnis zwischen dem

Peak signal-to-noise ratio, kurz PSNR, gibt das Verhältnis zwischen dem Maximalwert und der maximalen Störung an. Da die meisten Signale sehr große Skalen haben, wird der PSNR häufig mittels einer logarithmischen Skala angeben.

Der PSNR wird zur Messung der Qualität von nicht verlustfreien Kompressionsalgorithmen verwendet. Dazu wird das Originalbild als Signal interpretiert und der Fehler, der durch die Kompression eingeführt wird als Rauschen. Ein größer der PSNR-Wert bedeutet eine besser Qualität des dekomprimierten Bildes.

Bei einer Farbtiefe von 8 Bit pro Kanal, stehen Werte von 30 - 40 dB für ein geringes Störsignal.

Der *mean squared error*, kurz MSE summiert einem Fenster der Größe $m \times n$ die quadratischen Abstände zwischen dem Original und dem rekonstruierten Bild auf.

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2 \tag{3.1}$$

Beim PSNR wird der maximal mögliche Wert MAX_I mit dem MSE ins Verhältnis gesetzt:

$$PSNR = 10 \times \log_{10} \left(\frac{MAX_I^2}{MSE} \right) \tag{3.2}$$

$$= 20 \times \log_{10} \left(\frac{MAX_I}{\sqrt{MSE}} \right) \tag{3.3}$$

$$= 20 \times \log_{10}(MAX_I) - 10 \times \log_{10}(MSE) \tag{3.4}$$

3.5 SSIM

Eine weitere Metrik, die zum Vergleich von Bilder eingesetzt wird, wurde von Wang *et al.* entwickelt [WBSS04]. Diese basiert auf der Idee das die Struktur der abgebildeten Objekte von Beleuchtung und Kontrast unabhängig ist.

Beleuchtung und Kontrast können im gesamten Bild variieren, aus diesem Grund werden beide Parameter lokal in einem Fenster bestimmt.

In ihrem System wird die Aufgabe, die Ähnlichkeit zu messen in drei Teile unterteilt: Beleuchtung, Kontrast und Struktur. Als Erstes wird die Beleuchtung bestimmt. Wenn die Signale, wie in diesem Fall diskret sind, lässt sich die mittlere Intensität, des Signals x mit dem Term

$$\mu_x = \frac{1}{N} \sum_{i=1}^{N} x_i \tag{3.5}$$

berechnen. Die Vergleichsfunktion für die Beleuchtung zwischen zwei Signalen x und y wird $l(x,y)$ und setzt die werte μ_x und μ_y wie folgt ins Verhältnis:

$$l(x,y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1}. \tag{3.6}$$

Als Nächstes wird die mittlere Intensität von den Signalen x und y abgezogen, so dass $x' = x - \mu_x$ und $y' = y - \mu_y$ entstehen.

Der Kontrast des Signals x' wird mit Hilfe der Standardabweichung $\sigma_{x'}$ approximiert. In diskreter Form lässt sich diese mit der folgenden Formel berechnen:

$$\sigma_{x'} = \left(\frac{1}{N-1} \sum_{i=1}^{N} (x_i' - \mu_{x'})^2 \right)^{\frac{1}{2}}. \tag{3.7}$$

Die Vergleichsfunktion für den Kontrast wird mit $c(x,y)$ bezeichnet und definiert sich wie folgt:

$$c(x,y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2}. \tag{3.8}$$

An dieser Stelle wird das Signal normalisiert, indem es durch seine eigene Standartabweichung geteilt wird, so dass die Signale $x'' = x' - \mu_{x'}/\sigma_x'$ und $y'' = y' - \mu_{y'}/\sigma_y'$ entstehen. Der Vergleich der strukturellen Eigenschaften $s(x,y)$ wird mit den normalisierten Signalen x'' und y'' durchgeführt. Für den Vergleich der Struktur sollen folgende Eigenschaften gelten:

Erstens, die Funktion $s(x,y)$ muss Symmetrisch sein $s(x,y) = s(y,x)$.

Zweitens soll die Funktion auf einen Wert kleiner oder gleich 1 beschränkt werden $s(x,y) \leq 1$.

Drittens es soll nur ein Maximum $s(x,y) = 1$, genau dann und nur dann, wenn gilt das $x = y$.

Eine Definition, die diese Forderungen erfüllt ist:

$$s(x,y) = \frac{\sigma_{xy} + C_3}{\sigma_x + \sigma_y + C_3}. \tag{3.9}$$

Die Kovarianz σ_{xy} berechnet sich folgendermaßen:

$$\sigma_{xy} = \frac{1}{N-1} \sum_{i=1}^{N} (x_i - \mu_x)(y_i - \mu_y). \tag{3.10}$$

Sie korreliert mit dem Kosinus des Winkels zwischen den beiden Vektoren $x - \mu_x$ und $y - \mu_y$. Die Konstanten C_1, C_2 und C_3 in den drei Vergleichsfunktionen sorgen dafür, das eine Division durch 0 nicht möglich wird, sollten die Werte in den Nennern zu klein werden. Letztlich kann die Gesamtqualität gemessen werden, indem Beleuchtung, Kontrast und Strukturvergleich kombiniert werden:

$$SSIM(x,y) = [l(x,y)]^{\alpha} \times [c(x,y)]^{\beta} \times [s(x,y)]^{\gamma}. \tag{3.11}$$

Mit den Parametern $\alpha > 0, \beta > 0, \gamma > 0$, kann die Gewichtung zwischen den Vergleichsfunktionen variiert werden. Im Rahmen dieser Arbeit gilt $\alpha = \beta = \gamma = 1$ und es gilt für die Konstanten: $C_3 = C_2/2$, mit $C_1 = 6.5025$ und $C_2 = 58.5225$. Der $SSIM$ wird lokal berechnet, in Fenstern der Größe 11×11, dabei erden die Signalwerte x_i und y_i mit einer Gaussianfunktion gewichtet. Damit lassen sich die mittlere Intensität, die Standardabweichung und die Kovarianz zu folgenden Gleichungen umschreiben, wobei w_i die Gewichtung an dem Punkt i bezeichnet:

$$\mu_x = \sum_{i=1}^{N} w_i x_i \tag{3.12}$$

$$\sigma_x = \left(\sum_{i=1}^{N} w_i (x_i - \mu_x)^2 \right)^{\frac{1}{2}} \tag{3.13}$$

$$\sigma_{xy} = \sum_{i=1}^{N} w_i (x_i - \mu_x)(y_i - \mu_y). \tag{3.14}$$

Die Gesamtgleichung für den $SSIM(x,y)$ lässt sich durch die Gleichung

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{3.15}$$

ausrechnen. Der $SSIM(x,y)$ misst die Güte jedoch nur lokal, um eine Aussage über das gesamte Bild treffen zu können, kann der Mittelwert über allen $x \in X$ und $y \in Y$ berechnet werden:

$$MSSIM(X,Y) = \frac{1}{M} \sum_{j=1}^{M} SSIM(x_j, y_j). \tag{3.16}$$

4 Methodik und Umsetzung

Das Ziel dieser Arbeit ist es, ein Remote-Visualisierungssystem zu entwickeln, und eine Klient-seitige Bildextrapolation durchzuführen. Grundlage sind jeweils ein Farbbild, ein Tiefenbild und die entsprechenden Kamerainformationen. Die Extrapolation wird durchgeführt, indem aus dem Tiefenbild wird ein Dreiecksnetz erzeugt wird, welches schließlich mit dem Farbbild als Textur, aus einer neuen Kameraperspektive gezeichnet wird. Mit Hilfe von Ground-Truth-Datensätzen wird die Qualität, der extrapolierten Bildern in Abhängigkeit der gewählten Parameter, mit dem PSNR und dem MSSIM gemessen und evaluiert. Die Kommunikation, zwischen den beiden Komponenten erfolgt mit Hilfe des WebSocket-Protokolls. Der Klient ist ein Webklient, basierend auf JavaScript, der die Bildextrapolation mit WebGL durchführt.

Die Zentrale Aufgabe ist die Optimierung und die Übertragung der Tiefeninformationen, vom Server zum Klient. Es gibt im wesentlichen zwei Möglichkeiten diese Aufgabe zu erfüllen. Zum einen, kann das Tiefenbild direkt komprimiert werden und im Anschluss daran wird es an den Klient gesendet, welcher das Dreiecksnetz aus dem Tiefenbild erzeugt und die Extrapolation durchführt. Die zweite Möglichkeit besteht in der Konstruktion des Dreiecksnetzes durch den Server. Beide Varianten werden in dieser Arbeit untersucht.

Im Folgenden werden beide zuerst Varianten konzeptionell vorgestellt. Anschließend wird die Architektur des Klient-Server-Systems vorgestellt und die Besonderheiten der Implementierung und die dazugehörigen Parameter im Detail erklärt. Im Anschluss daran werden die Parameterkonfigurationen und die erzeugten Ergebnisse objektiv dargestellt. Zum Schluss werden diese diskutiert und es wird auf Vor- und Nachteile der umgesetzten Lösungen eingegangen.

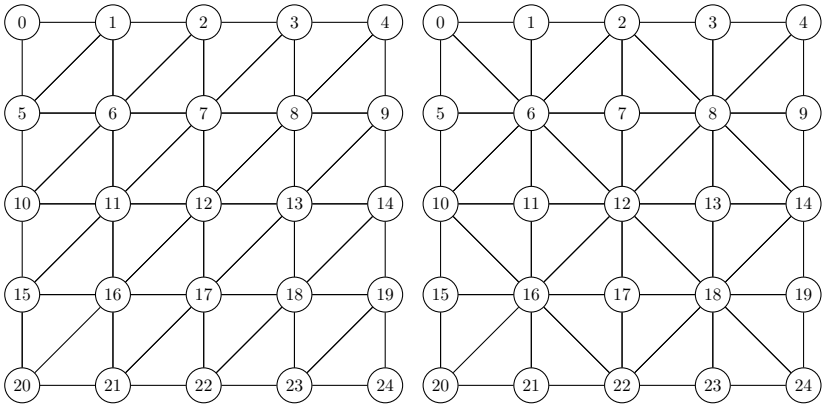
4.1 Erzeugung von Dreiecksnetzen

Grundlage für die Erzeugung der Dreiecksnetze ist das Tiefenbild D. Jeder Tiefenwert $d(x, y) \in D$ liegt in dem Intervall $[0, 1]$. Je kleiner der Wert von d ist, umso näher ist dieser an der *near-Plane*.

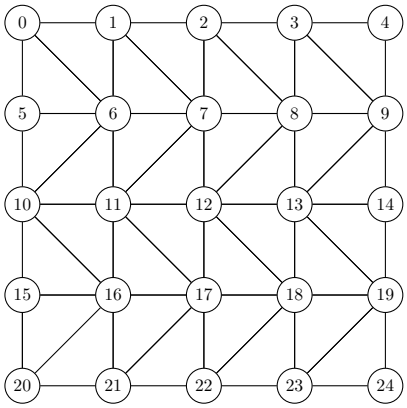
4.1.1 Vollvernetzung

In dieser Arbeit wird die Vollvernetzung vom Klient durchgeführt. Die Vernetzung des Dreiecksnetzes kann vorausberechnet werden und muss neu berechnet werden, wenn sich die Auflösung der Tiefenbilder ändert.

Bei der Vollvernetzung wird für jeden Pixel d aus dem Tiefenbild D ein Vertex v erzeugt. Es existieren verschiedene Möglichkeiten, diese miteinander zu einem Dreiecksnetz zu verknüpfen. Die Abbildung 4.1 zeigt drei unterschiedliche Varianten. Aus vier benachbarten Vertices werden jeweils zwei Dreiecke erzeugt. Hat das Tiefenbild D die Auflösung $w \times h$, dann entspricht die Anzahl aller Dreiecke $2(w-1)(h-1)$.



(a) regulär (b) cookie-cutter



(c) isometrisch

Abbildung 4.1: Darstellung von drei verschiedene Varianten der Vollvernetzung.

4.1.2 Delaunay-Triangulierung

Die zweite Variante die in dieser Arbeit untersucht wird, ist die eine adaptive Vernetzung des Tiefenbildes. Diese wird vom Server durchgeführt und das Dreiecksnetz wird anschließend als solches zum Klient gesendet. Dazu wurde der von Banno *et al.* [BGTB12] entwickelte Algorithmus implementiert.

Grundlage des Algorithmus ist die bereits in den Grundlagen erörterte Delaunay-Triangulierung. Zu beginn muss eine Menge von Punkten $P \subset D$ gewählt werden. Die Punkte $p \in P$ werden anschließend zu einem Dreiecksnetz mit Hilfe der Delaunay-Triangulierung vernetzt. Bevor das fertige Netz zum Klient gesendet wird, werden in einem Optimierungsschritt, die durch die Delaunay-Triangulierung erzeugten Kanten verbessert und gegeben falls neue Dreiecke zum Netz hinzugefügt.

Ziel ist es die Menge P so zu wählen, dass sie die Struktur des Tiefenbildes möglichst optimal approximiert wird. Punkte müssen an Stellen gesetzt werden, an denen Tiefensprünge auftreten oder die Oberfläche nicht planar ist. Die Dichte der gesetzten Punkte P ist dabei proportional zur lokalen Komplexität der Szene.

4.1.2.1 Besondere Punkte

Um die besonderen Punkte bestimmen zu können, müssen zuerst die Gradienten des Tiefenbildes ∇_x und ∇_y berechnet werden, dies geschieht mit Hilfe des Sobel-Operators. Dieser berechnet die erste Ableitung und glättet gleichzeitig dazu die orthogonale Richtung. Mit Hilfe einer Faltungsmatrix der Größe 3×3 können die Gradienten berechnet werden:

$$\nabla_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} * D \tag{4.1}$$

$$\nabla_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * D. \tag{4.2}$$

Die Punktemenge P wird mit Hilfe einer Quaternärbaum-Datenstruktur erzeugt. Ein Quaternärbaum unterteilt das Tiefenbild sukzessiv in vier rechteckige Regionen. Die maximale Tiefe des Quaternärbaums ist invers proportional zur Breite der Blattregionen. Mit diesem Parameter

kann die Qualität und die Kompression des erzeugten Netzes, sowie der Berechnungsaufwand skaliert werden.

Jeder Knoten des Baums entspricht einer rechteckigen Region R des Tiefenbildes, welche mit Hilfe der Sobel-Gradienten auf planarität getestet wird. Wenn die Region R nicht planar ist, dann werden die Eckpunkte von R zur Menge P hinzugefügt. Wenn die Gradienten in einer Region R nahezu gleich sind, dann wird diese Region R als planar bezeichnet. Ob eine Region R als planar bezeichnet werden kann, lässt sich mit Hilfe der Differenz, zwischen dem maximalen und dem minimalen Gradienten von R bestimmten:

$$c_x = \max_R \nabla_x - \min_R \nabla_x = \nabla_{H_x} - \nabla_{L_x} \tag{4.3}$$

$$c_y = \max_R \nabla_y - \min_R \nabla_y = \nabla_{H_y} - \nabla_{L_y}. \tag{4.4}$$

Wenn entweder c_x oder c_y größer als ein Schwellwert T_{planar} ist, dann ist die Region R nicht planar und die Eckpunkte werden der Menge P hinzugefügt.

Der Baum wird beginnend mit dem Blattknoten zum Wurzelkonten traversiert. Die Werte für c_x und c_y der inneren Konten lassen sich effizient mit Hilfe von ∇_H und ∇_L der vier Kindknoten berechnen.

Der Schwellwert T_{planar} ist entscheidend für die Anzahl der eingefügten Punkte in P. Der Wert von T_{planar} kann adaptiv gewählt werden, um die Komplexität des Dreiecksnetzes anzupassen. Für den eigentlichen Planaritäts-Test unterscheidet sich der Schwellwert für Blatt oder innere Knoten in T_{leaf} und T_{inner}. Die Idee dahinter ist, dass Blattknoten vorrangig Tiefenuntersprünge detektieren sollen bzw. die Kontur, während die inneren Knoten nicht-planare Oberflächen repräsentieren. Weil Tiefensprünge größere Gradienten bedeuten als nicht-planare Oberflächen, sollte $T_{internal}$ kleiner T_{leaf} als gewählt werden.

Durch das Traversieren des Quaternärbaums lässt sich die Menge P bestimmen und das Dreiecksnetz kann mit Hilfe der Delaunay-Triangulierung vernetzt werden.

Ähnlich zur Vollvernetzung kann die Datenstruktur des Quaternärbaums im vorausberechnet werden und wird nur im Falle einer Auflösungsänderung oder einer Änderung des maximalen Tiefenwertes des Quaternärbaums neu berechnet.

4.1.2.2 Optimierung des Netzes

Das auf diese Weise entstandene Netz enthält zwei arten von Artefakten. Zum einen können Dreiecke falsche Tiefenregionen approximieren, weil nur die Eckpunkte der jeweiligen Regionen R zur Konstruktion der Dreiecke genutzt werden. Zum anderen können Dreiecke über Tiefensprüngen liegen und dadurch werden Kanten nicht korrekt vom Netz abgebildet. Um die Artefakte zu reduzieren, werden nicht valide Dreiecke nochmals unterteilt oder verworfen. Die Validität eines Dreiecks wird anhand seiner Kanten bestimmt. Eine Kante wird als nicht valide Kante bezeichnet, wenn die Tiefenwerte einiger von ihr überspannten Pixel abweicht, oder die 3D Richtung der Kante sich dem Lot der Bildebene nährt.

Die Eckpunkte der zu prüfenden Kante, werden im folgenden mit p_1 und p_2 bezeichnet. Der Punkt p_m bezeichnet die Seitenhalbierende, den Median der Kante p_1p_2. Ein entscheidender Punkt, ist der das alle Streckenlängen mit Texturkoodinaten berechnet werden, damit die Gleichungen unabhängig von der eigentlichen Auflösung sind.

In der Gleichung 4.5 approximiert der erste Term das Verhältnis, zwischen dem Betrag des Tiefenunterschieds von p_1 und p_2 zu der Länge der Kante projiziert auf die xy-Ebene.

$$\frac{\mid d(p_1) - d(p_2) \mid}{\parallel p_1 - p_2 \parallel (d(p_1) + d(p_2)} < T_{angle} \tag{4.5}$$

Ist der Wert des ersten Terms der Gleichung 4.5 einer Kannte größer als der Schwellwert T_{angle}, dann steht diese Kante nahezu senkrecht auf der Bildebene und sie liegt mit hoher Wahrscheinlich über einen Tiefensprung. Eine Kante auf die das zutrifft wird als nicht valide bezeichnet. Enthält ein Dreieck mindestens zwei Kanten die valide sind, wird es direkt zum endgültigen Dreiecksnetz hinzugefügt. Wenn ein Dreieck dagegen mehr als zwei nicht valide Kanten enthält, wird es weiter unterteilt.

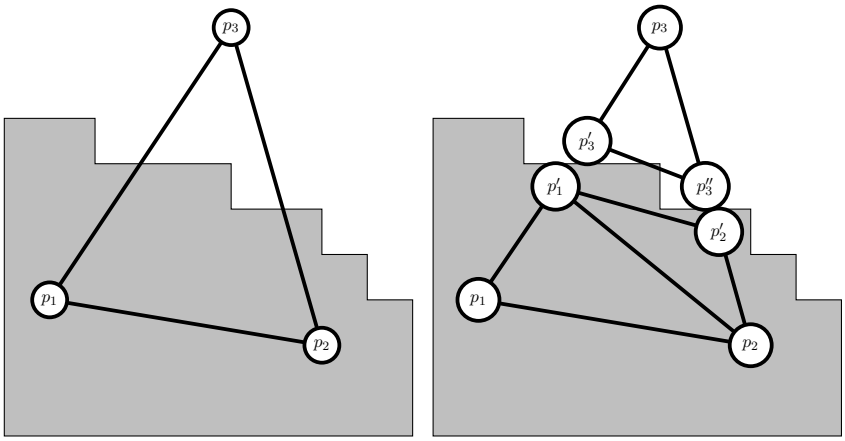
Zwei Bildpunkte p_1 und p_2 werden als verbindbar bezeichnet, wenn alle Bildpunkte zwischen p_1 und p_2 valide Tiefenwerte besitzen und ihre Tiefe sich zum größten Teil linear ändert. Um Rechenzeit zu sparen wird empfohlen nur den Median p_m zwischen p_1 und p_2 zu testen. Eine Strecke wird als Verbindbar bezeichnet, wenn sie die folgende Gleichung erfüllt:

$$\mid (d(p_2) - d(p_m)) - (d(p_m) - d(p_1)) \mid < T_{join} \tag{4.6}$$

.

Um ein Dreieck zu unterteilen muss eine Fallunterscheidung durchgeführt werden. Hat das Drei-

eck nur eine valide Kante p_1p_2, dann müssen die anderen beiden Kanten wie in der Abbildung 4.2 geteilt werden und es entstehen aus dem ursprünglichen Dreieck drei neue Dreiecke.



(a) (b)

Abbildung 4.2: Die Bilder a und b zeigen die Unterteilung des Dreiecks $p_1p_2p_3$, wenn die zwei Kanten p_1p_3, p_2p_3 nicht valide sind.

Dazu werden vier neue Vertices iterativ entlang der alten Strecken eingefügt:

Der Vertex p'_1 ist der von Punkt auf der Strecke p_3p_1, der am weitesten von p_1 entfernt liegt und mit dem Punkten p_1 und p'_2 verbindbar ist.

Der Vertex p'_2 ist der von Punkt auf der Strecke p_3p_2, der am weitesten von p_2 entfernt liegt und mit dem Punkten p_2 und p'_1 verbindbar ist.

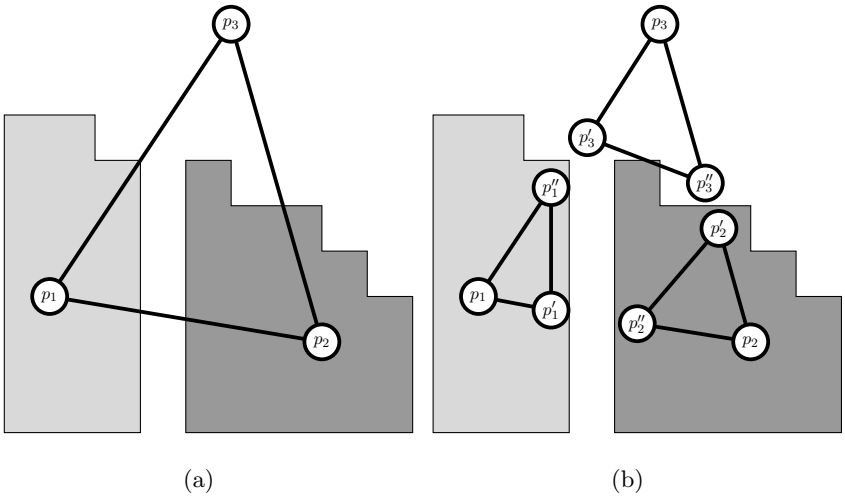
Der Vertex p'_3 ist der von Punkt auf der Strecke p_3p_1, der am weitesten von p_3 entfernt liegt und mit dem Punkt p_3 verbindbar ist.

Der Vertex p''_3 ist der von Punkt auf der Strecke p_3p_2, der am weitesten von p_3 entfernt liegt und mit dem Punkt p_3 verbindbar ist.

Anschließend wird das Dreieck $p_3p'_3p''_3$ zum endgültigen Dreiecksnetz hinzugefügt. Um die anderen beiden Dreiecke einzufügen muss eine weitere Fallunterscheidung durchgeführt werden. Wenn die Strecke $p_1p'_2$ kleiner ist als $p_2p'_1$, dann werden die Dreiecke $p_1p_2p'_2$, $p_1p'_2p'_1$ zu dem finalen Netz hinzugefügt, andernfalls die beiden Dreiecke $p_1p_2p'_1$ und $p_2p'_2p'_1$.

In dem Fall, das alle drei Kanten nicht valide sind, werden sechs neue Vertices eingefügt, die Abbildung 4.3 verdeutlicht diesen Fall. Die Berechnung aller Vertices geschieht analog zu dem Vertex p'_3 aus dem vorhergehenden Betrachtung mit einer validen Kante.

Zu beachten ist, dass jedes neu erzeugte Dreieck auf Kollinearität zu überprüfen und gegebe-



(a) (b)

Abbildung 4.3: Die Bilder a und b zeigen die Unterteilung des Dreiecks $p_1p_2p_3$, wenn keine Kante valide ist.

nenfalls zu verwerfen.

4.2 Implementierung

Als Grundlage dient das Konzept von Wessels *et al.* [WPJR11]. Die Klient-Anwendung ist eine Browser basierte Web-Anwendung, die mit dem Server über das WebSocket-Protokoll kommuniziert. Dabei handelt es sich um ein auf TCP basierendes Netzwerkprotokoll, das eine Bidirektionale Verbindung erlaubt. Die Arbeit von Wessels *et al.* wird dahingehend erweitert, das die Manipulation der Bilddaten mit Hilfe von WebGl durchgeführt wird. Bei WebGl handelt es sich um eine Bibliothek die von modernen Browsern zur Verfügung gestellt wird, um eine Hardware beschleunigte Bildsynthese zu ermöglichen. Der Vorteil dieser Technologie ist die Unabhängigkeit der Anwendung im Bezug, zur Plattform und dem Gerät, auf dem sie laufen soll.

Beide Komponenten tauschen Informationen mit Hilfe des kompakten vom Menschen lesbaren Java

4.2.1 16 Bit

Farb- und Tiefenbild werden mit base64 kodiert übertragen und lassen sich vom Browser nativ dekodieren. Die Farbtiefe pro Kanal ist Browserseitig auf 8 Bit beschränkt. Um 16 Bit Tiefeninformationen im Vertex-Shader nutzen zu können, müssen diese 16 Bit auf zwei 8 Bit Kanäle aufgeteilt werden. Die Abbildung 4.4 verdeutlicht dieses Verfahren. In den roten-Farbkanal wer-

den die ersten 8 Bit und in den grünen-Farbkanal die restlichen 8 Bit aufgeteilt. Die Aufteilung der 16 Bit auf die Farbkanäle erfolgt Server-seitig.

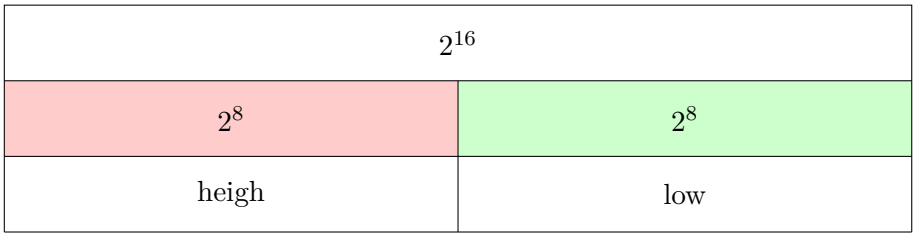


Abbildung 4.4: Hier wird die Aufteilung einer 16 Bit Zahl auf die Farbkanäle, links rot und rechts grün, visuell verdeutlicht. Das Schlüsselwort *heigh* bezeichnet die ersten 8 Bit und *low* die zweiten 8 Bit.

Um aus *heigh* und *low* die 16 Bit v zu berechnen, genügt es den Wert von *heigh* zuerst mit 255 zu multiplizieren und dann den Wert von *low* zu addieren. Die Gleichung 4.7 zeigt genau diesen Zusammenhang:

$$v = low + heigh \times 255. \tag{4.7}$$

Beim Laden einer Textur auf die Grafikkarte, werden alle Farbkanäle normiert, im Fall eines 8 Bit Bildes, wird jeder Wert durch den Maximalwert 255 geteilt, so dass der Wert im Intervall von $[0, 1]$ liegt.

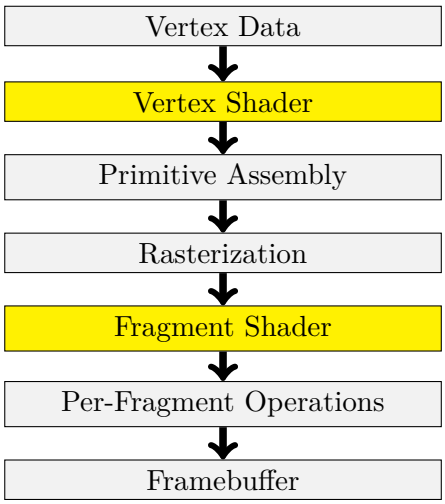


Abbildung 4.5: OpengGl ES 2.0 Darstellungspipeline. Grau unterlegt sind statischen und gelb die programmierbaren Elemente.

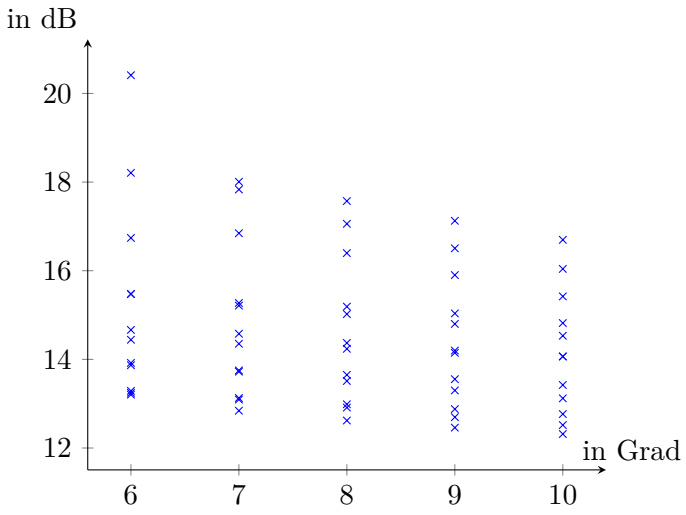
5 Ergebnisse

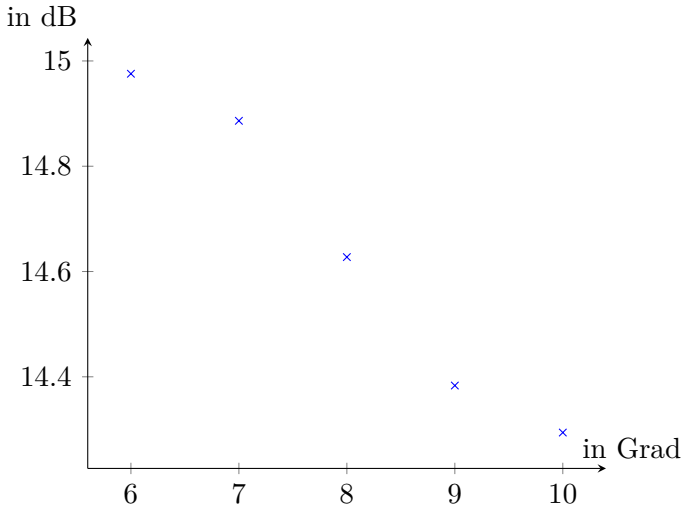
6 Diskussion

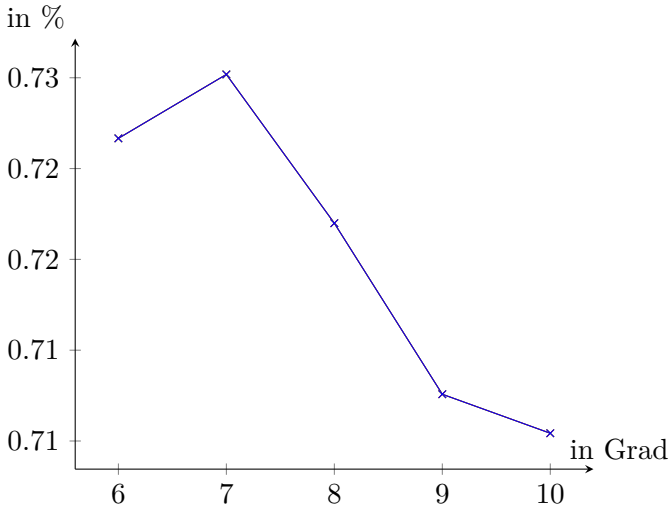
7 Zusammenfassung

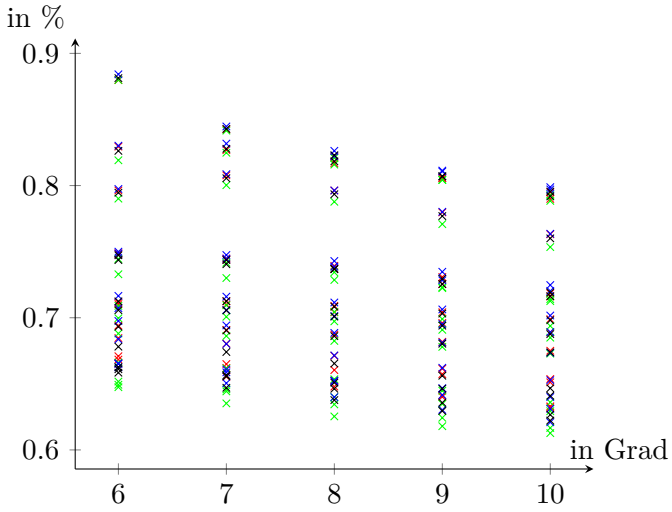
8 Ausblick

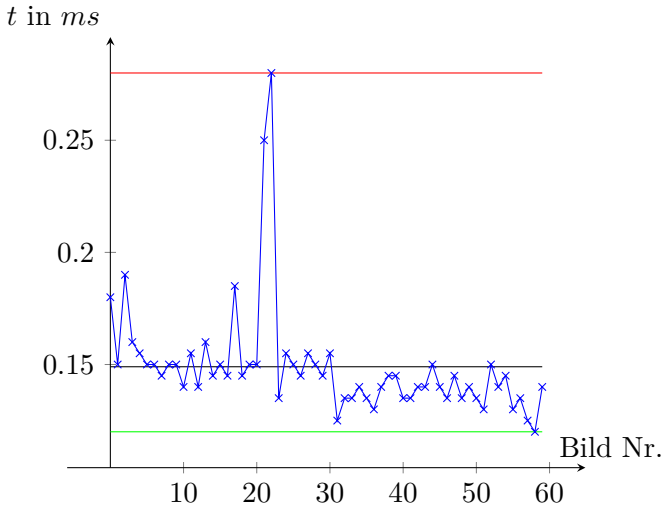
9 Noch mehr Ergebnisse



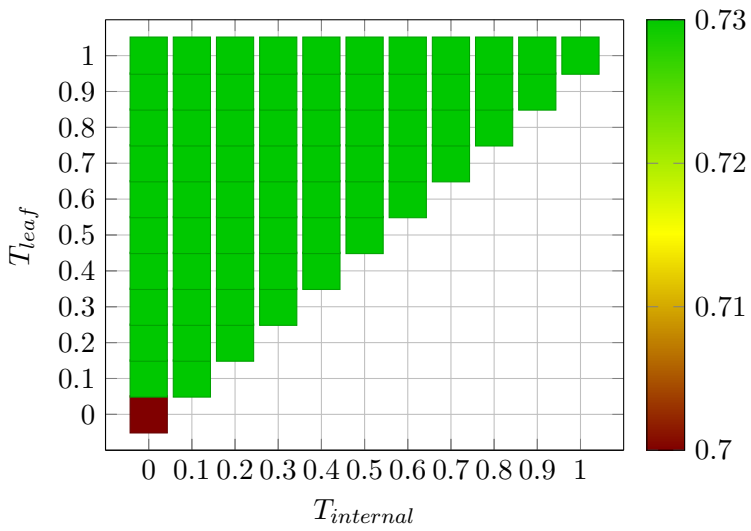



in %
0.73
0.72
0.72
0.71
0.71
in Grad
6
7
8
9
10


in %
0.9
0.8
0.7
0.6
in Grad
6
7
8
9
10


t in ms
0.25
0.2
0.15
Bild Nr.
10
20
30
40
50
60

[]


0.73
0.72
0.71
0.7
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
T_{leaf}
0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1
$T_{internal}$

Literaturverzeichnis

[Azu97] Azuma, Ronald. *A Survey of Augmented Reality.* 1997

[BG04] Bao, P. ; Gourlay, D.: Remote walkthrough over mobile networks using 3-D image warping and streaming. In: *IEE Proceedings - Vision, Image and Signal Processing* 151 (2004), Aug, Nr. 4, S. 329–336. – ISSN 1350–245X

[BGTB12] Bannò, Filippo ; Gasparello, Paolo S. ; Tecchia, Franco ; Bergamasco, Massimo: Real-time Compression of Depth Streams Through Meshification and Valence-based Encoding. In: *Proceedings of the 11th ACM SIGGRAPH International Conference on Virtual-Reality Continuum and Its Applications in Industry.* New York, NY, USA : ACM, 2012 (VRCAI '12). – ISBN 978–1–4503–1825–9, S. 263–270

[BKN05] Baratto, Ricardo A. ; Kim, Leonard N. ; Nieh, Jason: THINC: A Virtual Display Architecture for Thin-client Computing. In: *SIGOPS Oper. Syst. Rev.* 39 (2005), Oktober, Nr. 5, S. 277–290. – ISSN 0163–5980

[CSS02] Chai, Bing-Bing ; Sethuraman, S. ; Sawhney, H. S.: A depth map representation for real-time transmission and view-based rendering of a dynamic 3D scene. In: *3D Data Processing Visualization and Transmission, 2002. Proceedings. First International Symposium on*, 2002, S. 107–114

[Dee95] Deering, Michael: Geometry Compression. In: *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques.* New York, NY, USA : ACM, 1995 (SIGGRAPH '95). – ISBN 0–89791–701–4, S. 13–20

[EAB14] Evans, Alun ; Agenjo, Javi ; Blat, Josep: Web-based Visualisation of On-set Point Cloud Data. In: *Proceedings of the 11th European Conference on Visual Media Production.* New York, NY, USA : ACM, 2014 (CVMP '14). – ISBN 978–1–4503–3185–2, S. 10:1–10:8

[Eis07] Eisert, Peter: Remote rendering of computer games. In: *in Proc. Internation Conference on Signal Processing and Multimedia Applications (SIGMAP*, 2007

[GS98] Gumhold, Stefan ; Strasser, Wolfgang: Real Time Compression of Triangle Mesh Connectivity. In: *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA : ACM, 1998 (SIGGRAPH '98). – ISBN 0–89791–999–8, S. 133–140

[Jin06] Jin, Zhefan: Research on Rendering Instruction Stream Compression in Distributed VR Continuum. In: *Proceedings of the 2006 ACM International Conference on Virtual Reality Continuum and Its Applications*. New York, NY, USA : ACM, 2006 (VRCIA '06). – ISBN 1–59593–324–7, S. 13–18

[kho] khonos. *WebGL - OpenGL ES 2.0 for the Web*

[Lev95] Levoy, Marc: Polygon-assisted JPEG and MPEG Compression of Synthetic Images. In: *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA : ACM, 1995 (SIGGRAPH '95). – ISBN 0–89791–701–4, S. 21–28

[MKB+15] Mwalongo, Finian ; Krone, Michael ; Becher, Michael ; Reina, Guido ; Ertl, Thomas: Remote Visualization of Dynamic Molecular Data Using WebGL. In: *Proceedings of the 20th International Conference on 3D Web Technology*. New York, NY, USA : ACM, 2015 (Web3D '15). – ISBN 978–1–4503–3647–5, S. 115–122

[MWB+13] Ma, L. ; Whelan, T. ; Bondarev, E. ; de With, P.H.N ; McDonald, J.: Planar Simplification and Texturing of Dense Point Cloud Maps. In: *European Conference on Mobile Robotics (ECMR)*. Barcelona, Spain, September 2013. – Accepted. To appear.

[PD15] Ponchio, Federico ; Dellepiane, Matteo: Fast Decompression for Web-based View-dependent 3D Rendering. In: *Proceedings of the 20th International Conference on 3D Web Technology*. New York, NY, USA : ACM, 2015 (Web3D '15). – ISBN 978–1–4503–3647–5, S. 199–207

[PG10] Palomo, Cesar ; Gattass, Marcelo: An Efficient Algorithm for Depth Image Rendering. In: *Proceedings of the 9th ACM SIGGRAPH Conference on Virtual-Reality Continuum and Its Applications in Industry*. New York, NY, USA : ACM, 2010 (VRCAI '10). – ISBN 978–1–4503–0459–7, S. 271–276

[PGK02] Pauly, Mark ; Gross, Markus ; Kobbelt, Leif P.: Efficient Simplification of Point-sampled Surfaces. In: *Proceedings of the Conference on Visualization '02*. Washington, DC, USA : IEEE Computer Society, 2002 (VIS '02). – ISBN 0–7803–

7498–3, S. 163–170

[SH15] Shi, Shu ; Hsu, Cheng-Hsin: A Survey of Interactive Remote Rendering Systems. In: *ACM Comput. Surv.* 47 (2015), Mai, Nr. 4, S. 57:1–57:29. – ISSN 0360–0300

[SLBF09] Smit, F. ; van Liere, R. ; Beck, S. ; Froehlich, B.: An Image-Warping Architecture for VR: Low Latency versus Image Quality. In: *2009 IEEE Virtual Reality Conference*, 2009. – ISSN 1087–8270, S. 27–34

[SLN99] Schmidt, Brian K. ; Lam, Monica S. ; Northcutt, J. D.: The Interactive Performance of SLIM: A Stateless, Thin-client Architecture. In: *SIGOPS Oper. Syst. Rev.* 33 (1999), Dezember, Nr. 5, S. 32–47. – ISSN 0163–5980

[SME02] Stegmaier, Simon ; Magallón, Marcelo ; Ertl, Thomas: A generic solution for hardware-accelerated remote visualization. In: *In VISSYM '02: Proc. Symposium on Data Visualisation 2002*, 2002, S. 87

[SNC12] Shi, Shu ; Nahrstedt, Klara ; Campbell, Roy: A Real-time Remote Rendering System for Interactive Mobile Graphics. In: *ACM Trans. Multimedia Comput. Commun. Appl.* 8 (2012), Oktober, Nr. 3s, S. 46:1–46:20. – ISSN 1551–6857

[TR98] Taubin, Gabriel ; Rossignac, Jarek: Geometric Compression Through Topological Surgery. In: *ACM Trans. Graph.* 17 (1998), April, Nr. 2, S. 84–115. – ISSN 0730–0301

[WBSS04] Wang, Zhou ; Bovik, Alan C. ; Sheikh, Hamid R. ; Simoncelli, Eero P.: Image Quality Assessment: From Error Visibility to Structural Similarity. In: *IEEE TRANSACTIONS ON IMAGE PROCESSING* 13 (2004), Nr. 4, S. 600–612

[WPJR11] Wessels, Andrew ; Purvis, Mike ; Jackson, Jahrain ; Rahman, Syed (.: Remote Data Visualization through WebSockets. In: *Eighth International Conference on Information Technology: New Generations, ITNG 2011, Las Vegas, Nevada, USA, 11-13 April 2011*, 2011, S. 1050–1051

Abbildungsverzeichnis

Tabellenverzeichnis

Danksagung

Die Danksagung...

Erklärungen zum Urheberrecht

Hier soll jeder Autor die von ihm eingeholten Zustimmungen der Copyright-Besitzer angeben bzw. die in Web Press Rooms angegebenen generellen Konditionen seiner Text- und Bildübernahmen zitieren.