

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. rer. nat. Sebastian Grottel

Dresden, 11. Dezember 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 werden. 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 Client-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 Clienten an. Diese Aufbereitung ist vor allem die Generierung einer optimierten Tiefennetzgeometrie aus den Tiefenbilddaten. Der Client 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 Client 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 Client-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 Clienten 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 11. Dezember 2016

Josef Schulz

Kurzfassung

Remote-Visualisierung ist eine Technik, welche die Bildsynthese und die Darstellung der Bilder auf zwei Komponenten aufteilt. Um die Latenzzeit der Kommunikation beider Komponenten vor dem Nutzer verbergen zu können, kann mit Hilfe zusätzlicher geometrischer Informationen eine Bildextrapolation durchgeführt werden. Für diesen Zweck kann das bei Bildsynthese entstehende Tiefenbild genutzt werden. Dessen Tiefeninformationen lassen sich durch ein Dreiecksnetz approximieren. In dieser Arbeit werden mehrere Verfahren, hinsichtlich der Güte und der Anzahl der dabei entstehenden Dreiecke, zur adaptiven Konstruktion solcher Netze untersucht.

Abstract

Remote-visualization is a technique, which separates the rendering and displaying of images into two components. In order to hide the latency of the communication of both components of the user, it is possible to use additional information about geometry to extrapolate the image. The depth image, which will be produced in the rendering step can be used as the necessary geometric information. Triangle-meshes can be used to approximate the depth image. In this work, some methods to produce such meshes are tested with respect to image quality and mesh complexity

Inhaltsverzeichnis

1 Einleitung	3
2 Verwandte Arbeiten	5
3 Grundlagen	7
3.1 Datensätze	7
3.2 Extrapolation	8
3.3 Delaunay-Triangulierung	10
3.4 Bildvergleich	10
3.4.1 PSNR	10
3.5 SSIM	11
4 Methodik	15
4.1 Vollvernetzung	15
4.2 Delaunay-Triangulierung	16
4.2.1 Quadtree	17
4.2.2 Floyd-Steinberg	18
4.2.3 Erzeugung des Netzes	21
4.2.4 Netzoptimierung	22
5 Implementierung	27
5.1 Details zur Kommunikation	27
5.2 Details zur Implementierung der Algorithmen	29
5.2.1 Vollvernetzung	30
5.2.2 Delaunay-Triangulierung	31
6 Ergebnisse	33
7 Diskussion	53
8 Zusammenfassung und Ausblick	61
Literaturverzeichnis	63
Abbildungsverzeichnis	67
Tabellenverzeichnis	69

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 Client-Prozess gesendet wird. Der Client empfängt und dekodiert das Datenpaket und gibt das Bild 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 Verzögerungszeit und damit eine interaktive Visualisierung zu gewährleisten. Ein möglicher Ausweg besteht darin, dass der Client-Prozess ein bereits empfangenes Bild extrapoliert. Das bedeutet, dass das noch nicht empfangene Bild aus den vorhanden Informationen approximiert wird, wodurch die Bildwiederholfrequenz akzeptabel bleibt.

Bei der Bildsynthese des Server-Prozesses lässt sich aus dem Tiefenpuffer ein Tiefenbild erzeugen, zusätzlich zum Farbbild. Geometrisch entspricht das Tiefenbild einer 2.5D-Ansicht der Szene. Wird es zum Client gesendet, kann dieser das Tiefenbild als Punktfolge interpretieren. Das approximierte Bild entsteht, indem die Punktfolge aus einer neuen Kameraperspektive gezeichnet wird. Mit zunehmendem Winkelunterschied zwischen der alten und der neuen Kameraperspektive werden die Lücken zwischen den Punkten der Punktfolge, immer größer. Durch die Erzeugung eines Dreiecksnetzes aus dem Tiefenbild lassen sich die Lücken schließen. Das Farbbild wird dabei als Textur über das Netz gelegt.

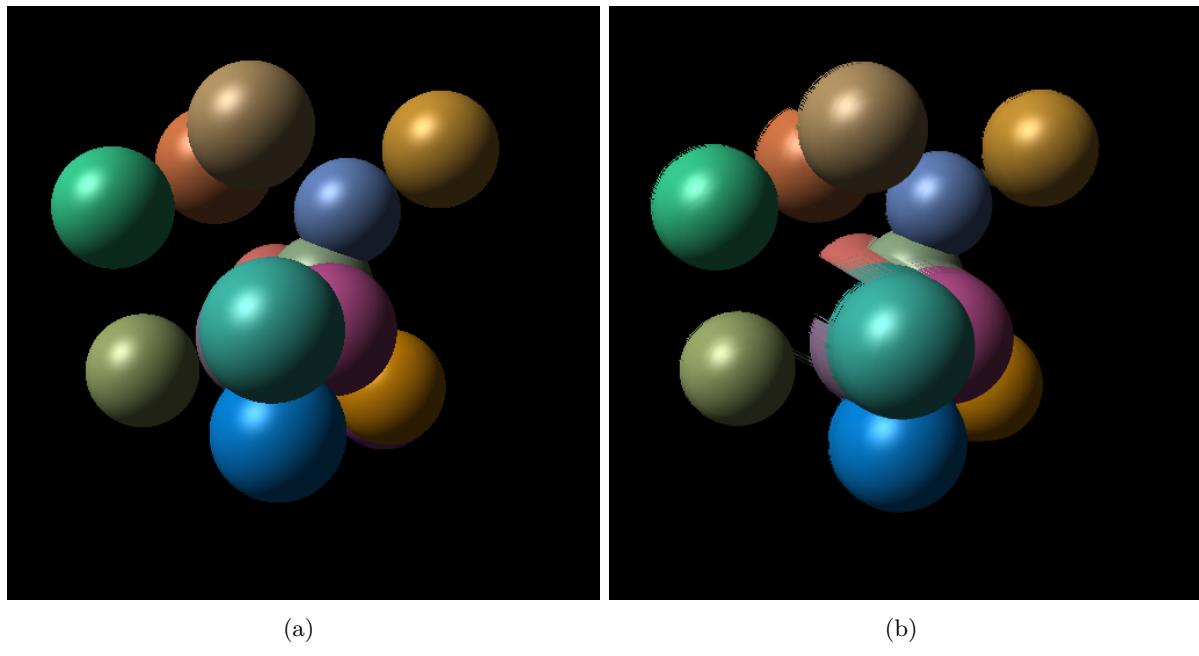


Abbildung 1.1: Die Abbildungen zeigen die Extrapolation des ersten Bildes a aus dem Datensatz *TestSpheres*. Der Winkelunterschied zum Bild b beträgt 8 Grad.

Die zentrale Aufgabe dieser Arbeit ist die Optimierung und die Übertragung der Tiefeninformationen vom Server zum Client. Zu diesem Zweck erzeugt der Server aus dem Tiefenbild ein adaptives Dreiecksnetz. Dieses wird an den Client übertragen und extrapoliert. Mit Ground-Truth-Datensätzen werden in Abhängigkeit des Kamerawinkels verschiedene Ansätze hinsichtlich ihrer Güte evaluiert. Um die Algorithmen zu testen, wurde eine Server- und eine Client-Komponente entwickelt. Beide Komponenten tauschen Informationen über das auf TCP-basierende WebSocket-Protokoll aus. Der Client ist ein Browser-basierter Web-Client, damit die implementierten Algorithmen mit den Einschränkungen durch JavaScript und WebGL evaluiert werden können.

Im Folgenden werden Arbeiten vorgestellt, die sich mit dieser Thematik bereits auseinander gesetzt haben. Anschließend werden die Grundlagen dieser Arbeit erläutert. In diesen Zusammenhang werden die Datensätze vorgestellt. Es wird gezeigt, wie die Rückprojektion und die Gütemetriken funktionieren. Danach werden die Algorithmen konzeptionell vorgestellt, gefolgt von Implementierungsdetails. Als Nächstes folgt die Präsentation der Ergebnisse, die im darauf folgenden Kapitel ausgewertet werden. Zum Schluss wird die gesamte Arbeit zusammengefasst und ein Ausblick gegeben, wie die Ergebnisse weiter verbessert werden könnten.

2 Verwandte Arbeiten

Einen Überblick über Architekturen und Methoden der interaktiven Remote-Visualisierung geben Shu Shi *et al.* [SH15]. Das zentrale Problem ihrer Arbeit ist die Latenz und die effiziente Übertragung der Daten vom Server zum Client. Lösungen hängen vom Anwendungsfall ab. Sie definieren *THIN*-Systeme als eine Klasse von Remote-Visualisierungssystemen, die auf die Übertragung von 2D Informationen beschränkt sind und die Bildwiederholfrequenz weitaus geringer sein kann, als es in interaktiven 3D-Anwendungen erforderlich ist. Das Ziel von *THIN*-Systemen besteht darin, Desktop-Anwendungen fernsteuern zu können. Beispiele für existierende Architekturen sind *SLIM* [SLN99] und *THiNC* [BKN05]. Sie sind für die Übertragung von 2D-Daten optimiert und profitieren davon, dass die meisten Änderungen nur Teilbereiche der eigentlichen Oberfläche betreffen. Eine allgemeinere Form der Remote-Visualisierung ist die Aufteilung der Bildsynthese auf verschiedene Host-Systeme wie bei dem Cluster-basierten System WireGL [HEB⁺01]. Jin Zhefan stellt in diesem Kontext Kompressionsmethoden für Zeicheninstruktionen, Vektoren, Normalen und Texturinformationen vor [Jin06]. Peter Eisert und Philipp Fechteler haben ein Remote-Visualisierungssystem für Computerspiele entwickelt [Eis07]. Ihr System überträgt bei kleinen Auflösungen für Endgeräte ohne GPU die Bilder kodiert mit H.264 oder wahlweise MPEG-4. Bei größeren Auflösungen werden Zeicheninstruktionen gesendet, die Bilder werden vom Client synthetisiert. Auch wenn ihr System ausschließlich für den Einsatz in lokalen Netzwerken konzipiert wurde, ist die Latenz zu hoch, um vom Benutzer nicht registriert zu werden. Mit Hilfe einer Bildextrapolation durch den Client kann die Latenz reduziert werden. Zu diesen Zweck schätzen Shu *et al.* in ihrer Arbeit [BG04] zusätzliche Referenz-Kamerapositionen und erzeugen zusätzlich zum Farbbild der originalen Kameraposition weitere Tiefenbilder. Durch Warping werden die Farbinformationen unter Zuhilfenahme der Tiefenbilder in die neue Kamera-perspektive überführt. Dabei lassen sich durch Verdeckung bedingte Lücken durch Pixel aus den Referenztiefenbildern schließen. Das Farbbild wird mit JPEG komprimiert und die Tiefenbilder mit ZLIB. Ein ähnlicher Ansatz wurde im VR-Bereich von Smit *et al.* erprobt [SLBF09]. Auch in diesem System werden Lücken mit Hilfe von Tiefenbildern aus Referenz-Kamerapositionen geschlossen. Palomo *et al.* nutzen ebenfalls einen Warping-Algorithmus, um mehrere Tiefenbilder und ein Farbbild miteinander zu vereinen. Zu diesem Zweck konstruieren sie mit Hilfe des Alpha-Farbkanals der Bilder einen speziellen z-Buffer [PG10], um ihren Algorithmus komplett auf die GPU auslagern zu können. Die Wahl des Warping-Algorithmus ist entscheidend für die Performance der Client-Komponente. Neben der im VR-System zum Einsatz kommenden Variante [MMB97] bieten Mark *et al.* eine Übersicht über verschiedene Warping-Verfahren [Mar99]. Die Erzeugung von mehreren Referenztiefenbildern ist eine rechenintensive Operation, da von diesen nur wenige Pixel tatsächlich benötigt werden, um die verdeckungsbedingten Lücken zu füllen. Popescu und Aliaga haben zu diesen Zweck ein spezielles Kameramodell entworfen, dass verdeckte Bildbereiche mit Hilfe gekrümmter Sichtstrahlen mitberechnen kann [PA06]. Die bisher vorgestellten Ansätze werden in dieser Arbeit nicht aufgegriffen; sie sollen lediglich verdeutlichen, welche alternativen Möglichkeiten existieren.

Die effiziente Übertragung und Visualisierung von Tiefenbildern ist auch für die Arbeit mit Tiefensensorsystemen von großem Interesse. Banno *et al.* erzeugen aus Tiefenbildern, die durch Tiefensensoren erhoben wurden, Dreicksnetze mit Hilfe der Delaunay-Triangulierung [BGTB12]. Ausgangspunkt für die Vernetzung ist eine Menge von Punkten, die Kanten und Krümmungen im Tiefenbild repräsentieren. Zur Erzeugung dieser Punkte kommt ein Quadtree zum Einsatz. Zusätzlich haben sie ein Verfahren entwickelt, um das erzeugte Netz in einem weiteren Schritt

hinsichtlich der optischen Qualität zu verbessern. Einen alternativen Ansatz für die Erzeugung der zur Vernetzung benötigten Punktmenge wird von Lee *et al.* vorgestellt [LYW00]. Sie nutzen ein Fehlerdiffusionsverfahren, um in der Nähe von Kanten Punkte zu platzieren. In beiden Arbeiten wird die Extrapolation der Bilder durch die Rückprojektion der Dreiecksnetze durchgeführt. Diese beiden Ansätze bilden den Kern dieser Arbeit.

Um die Latenz bei der Übertragung zu verbessern, können mehrere Qualitätsstufen von einem Dreiecksnetz erzeugt werden, wie zum Beispiel in der Arbeit von Chai *et al.* [CSS02], die auf dem Verfahren von Lindstrom *et al.* basiert [LKR⁺96]. Mwalongo *et al.* haben einen hybriden Ansatz entwickelt, der zusätzlich zu einzelnen Vertices des Netzes Parameter für Kugeln übermittelt, die anschließend vom Client rekonstruiert werden [MKB⁺15]. Evans *et al.* haben ein Dateiformat für Punktfolgen entwickelt, um diese progressiv an einen auf WebGL basierten Client zu senden [EAB14].

Wessels *et al.* stellen eine Konzeption für den Programmaufbau eines interaktiven Remote-Visualisierungssystems basierend auf dem WebSocket-Protokoll vor [WPJR11]. In ihrem System besteht der Server-Prozess aus zwei Hauptkomponenten, der Visualisierungs-Engine und dem Daemon. Während die Visualisierungs-Engine für die Bildsynthese zuständig ist, übernimmt der Daemon die Kommunikation mit dem Client-Prozess. Der Client schickt dabei seine Eingabeinformationen von Maus und Tastatur direkt an den Server. Dieser wertet die Daten aus und erzeugt daraufhin ein mit JPEG komprimiertes Bild, das mit Base64 kodiert wird und schließlich an den Client-Prozess geschickt wird. Dieser kann das Bild nativ mit Hilfe eines HTML5 Canvas dekodieren und darstellen. Ihr System wird zur Grundlage für die in dieser Arbeit verwendete Softwarearchitektur.

Auch wenn die Kompression und kompakte Kodierung der Dreiecksnetze nicht Teil dieser Arbeit sind, sollen die folgenden Arbeiten einen Überblick über mögliche Techniken geben. Gabriel Taubin und Jarek Rossignac haben einen 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 von Stefan Gumhold und Wolfgang Straßer geschrieben [GS98]. Kompression und Dekompression sind echtzeitfähig. Weitere geometrische Kompressionsverfahren für Vertices, Indices und Normalen, sind in der Arbeit von Michael Deering zu finden [Dee95] sowie in der Arbeit [TG98] von den Autoren Touma und Gotsman.

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. Anschließend werden die zwei Metriken PSNR und SSIM zum Vergleich von Bildern vorgestellt, die für die Evaluation verwendet werden.

3.1 Datensätze

Zur Analyse der verwendeten Methoden stehen zwei Datensätze zur Verfügung. Jeder Datensatz besteht aus einer Sequenz von Tripeln. Dabei setzt sich jedes Tripel aus einem Farbbild, einem Tiefenbild D sowie einem Satz von Kameraparametern zusammen. Die Auflösung der Farb- und Tiefenbilder beträgt für alle Datensätze $w \times h = 512 \times 512$. Die Tiefenbilder wurden mit 16 Bit und die Farbbilder mit 24 Bit quantisiert. Dabei entfallen jeweils 8 Bit auf die einzelnen Farbkanäle. In der Abbildung 3.1 werden Farb- und Tiefenbild des ersten Frames aus den beiden Datensätzen, *TestSpheres* und *CoolRandom* dargestellt.

Die Kameraparameter setzen sich aus einem Vektor für die Kameraposition, einem *lookAt*-Vektor, der Punkt, auf den die Kamera schaut und dem *up*-Vektor zusammen. Für die Projektion werden die Positionen der Clippingebenen durch die Werte z_{near} und z_{far} beschrieben. Des weiteren wird der Öffnungswinkel der Kamera benötigt.

Von beiden Szenen stehen 493 Bilder zur Verfügung. Beide Datensätze lassen sich in Abhängigkeit der Winkelschrittweite in 3 Sequenzen unterteilen. Der Winkel gibt dabei den Unterschied zwischen der Kameraposition aus dem ersten und dem gerade betrachteten Frame an. Eine Auflistung der Sequenzen zeigt die Tabelle 3.1.

#	Anzahl Bilder	min Winkel	max Winkel	Winkel Schritt
1	241	0	5	0.25
2	60	6	10	1
3	192	15	90	5

Tabelle 3.1: Die Tabelle zeigt eine Übersicht über die Winkel der aufgenommenen Sequenzen.

Die Datensätze *CoolRandom* und *TestSpheres* wurden aus Partikeldatensätzen erzeugt. *TestSpheres* ist ein synthetisch erzeugter Datensatz, der aus wenigen Partikeln generiert wurde. Bei dem Datensatz *CoolRandom* handelt es sich dagegen um einen komplexen Datensatz, der mit Molekül-Datensätzen vergleichbar ist, welche in existierenden Anwendungen visualisiert werden.

Weiterhin zu beachten ist, dass das Tiefenbild des ersten Frames, aus dem *TestSpheres*-Datensatz einen Fehler aufweist. Die vorderste Kugel, in der Abbildung 3.1 als die dunkelste Kugel zu sehen, ist im Tiefenbild nicht vollständig repräsentiert. Sie weist nur an den Seiten Krümmungen auf, ist im Inneren jedoch vollkommen flach. Was dazu führt, dass diese Kugel bei der Extrapolation nicht richtig rekonstruiert werden kann.

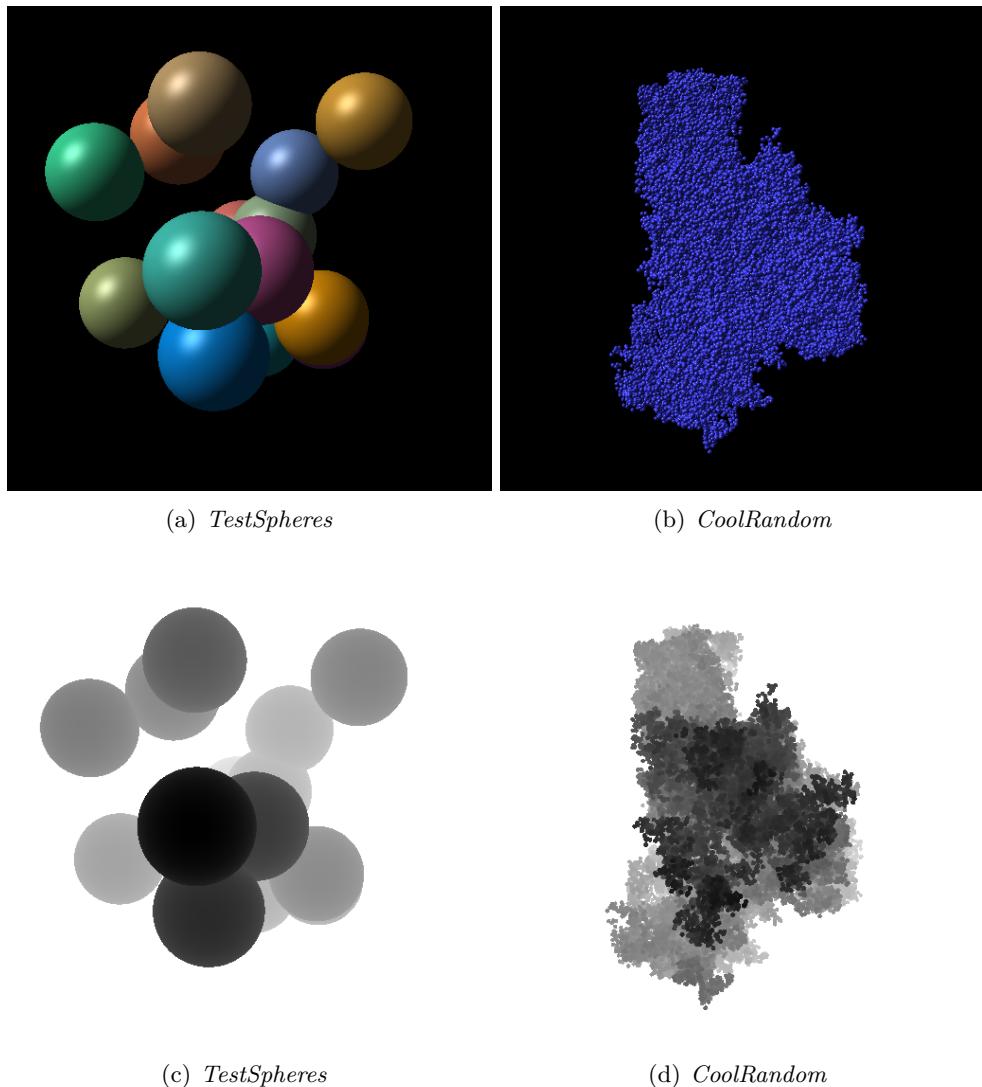


Abbildung 3.1: Farb- und Tiefenbilder der ersten Frames von beiden Datensätzen.

3.2 Extrapolation

Eine einfache Möglichkeit, um die Extrapolation durchzuführen, besteht darin, das Tiefenbild als Punktfolge zu interpretieren und diese aus einer neuen Kameraperspektive zu zeichnen. Dazu wird aus jedem Pixel $d \in D$ ein Vertex erzeugt. Mit Hilfe der Rückprojektion lassen sich die Vertices in die gewünschten Positionen transformieren.

Um das Prinzip der Rücktransformation zu erläutern, soll zunächst betrachtet werden, wie ein Vertex v aus dem Modellkoordinatensystem in normalisierte Gerätekordinaten transformiert wird. Die Abbildung 3.2 verdeutlicht diesen Vorgang.

Der Vertex $v = (x, y, z, 1)^T$ liegt zunächst in homogenen Modellkoordinaten vor. Mit Hilfe der Modellmatrix $M = RT$, die aus einer Rotations- und einer Translationsmatrix besteht, lässt sich der Vektor in das Weltkoordinatensystem überführen. Die Transformation in das Kamerakoordinatensystem wird durch die Multiplikation der Kameramatrix V mit dem Vertex v berechnet. Schließlich kann die Projektion aus dem Kamerakoordinatensystem in normalisierte Gerätekordinaten mit Hilfe der Projektionsmatrix P berechnet werden. In der Gleichung 3.1 wird die Transformation von dem Vertex v aus den Modellkoordinaten in die normalisierten

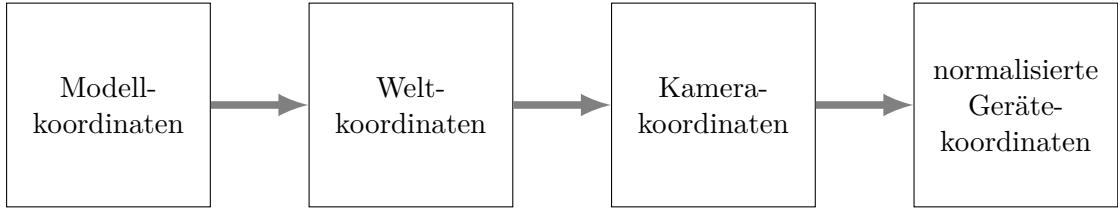


Abbildung 3.2: Die Abbildung zeigt die Teilschritte der Transformation eines Vertices aus den Modellkoordinaten in Bildschirmkoordinaten.

Gerätekordinaten v' zusammengefasst:

$$v' = M \cdot V \cdot P \cdot v. \quad (3.1)$$

Normalisierte Gerätekordinaten haben für die x-, y- und z-Komponente den Wertebereich von -1 bis 1 . Diese lassen sich in Bildschirmkoordinaten umrechnen, indem die drei Komponenten in das Intervall $[0, 1]$ übersetzt werden. Anschließend werden die x- und die y-Komponenten auf den Bildbereich gestreckt, indem sie mit $w - 1$ und $h - 1$ der gewünschten Auflösung $w \times h$ multipliziert werden.

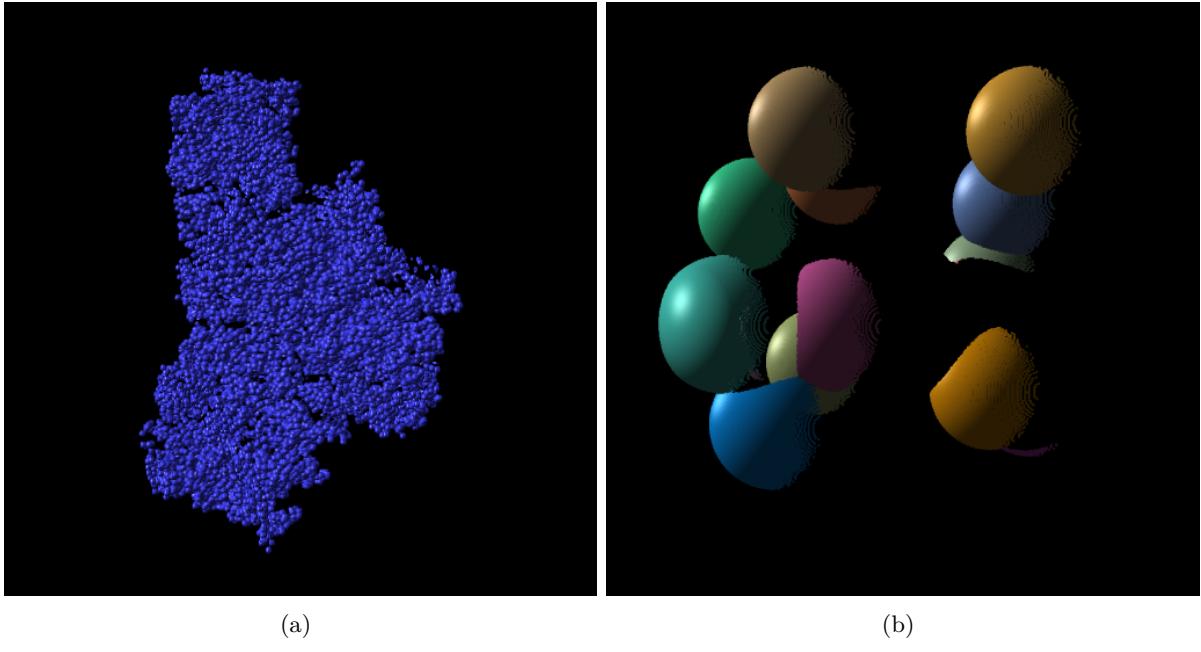
Im Folgenden wird die Menge der Tiefenbilder D_i und den dazugehörigen Transformationsmatrizen T_i , mit $i \in \{0, 1, \dots, N\}$ betrachtet. Jede Transformationsmatrix T_i setzt sich dabei aus einer Modellmatrix M_i , einer Kameramatrix V_i und einer Projektionsmatrix P_i zusammen:

$$T_i = M_i \cdot V_i \cdot P_i. \quad (3.2)$$

Um die Punktwolke des Tiefenbildes D_i mit der Transformationsmatrix T_j auf das Tiefenbild D_j abzubilden, muss zunächst für jeden Pixel $d_i \in D_i$ ein Vertex v_i erzeugt werden mit $i, j \in \{0, 1, \dots, N\}$. Zunächst liegt v_i in Bildschirmkoordinaten des Pixels d_i vor und muss in normalisierte Gerätekordinaten übersetzt werden. Zu diesem Zweck werden zuerst die x- und y-Komponenten durch $w - 1$ beziehungsweise $h - 1$ geteilt. Anschließend müssen alle drei Komponenten des Vektors v_i aus dem Wertebereich $[0, 1]$ in das Intervall $[-1, 1]$ transformiert werden. Jetzt liegt v_i in normalisierten Gerätekordinaten vor. Durch die Multiplikation der inversen Transformationsmatrix T_i^{-1} mit v_i , kann der Vertex in seine Modellkoordinaten überführt und im Anschluss mit Hilfe von T_j aus einer neuen Kameraperspektive gezeichnet werden:

$$v'_i = T_j \cdot T_i^{-1} v_i. \quad (3.3)$$

Die Interpretation der Tiefenbilder als Punktwolke ist keine optimale Lösung, da bei der Bildsynthese Lücken im Bild entstehen. Eine leistungsfähigere Lösung ist es, aus dem Tiefenbild ein Dreiecksnetz zu erzeugen und dieses für die Bildsynthese zu verwenden. Mit Hilfe der Delaunay-Triangulierung ist es möglich, aus Tiefenbildern adaptive Dreiecksnetze zu erzeugen, welche die Lücken füllen.



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 schnellsten erreichen eine Laufzeit von $O(n \log n)$ und sind damit für den Einsatz in Echtzeitanwendungen tauglich. Beispiele hierfür sind der Sweep-Algoritmus und die inkrementelle Konstruktion.

3.4 Bildvergleich

Im Folgenden werden zwei Metriken zum Vergleich von Bildern vorgestellt. Sie werden genutzt, um die Ähnlichkeit zwischen den extrapolierten und den Ground-Truth Bildern zu bestimmen. Beide Metriken werden für jeden Farbkanal separat berechnet.

3.4.1 PSNR

Die Abkürzung PSNR, im Englischen *Peak signal-to-noise ratio*, gibt das Verhältnis zwischen dem Maximalwert des Signals, auch als Nutzleistung bezeichnet, und dem Einfluss des Rauschens, also der Störleistung, an. Der PSNR wird auf eine logarithmische Skala abgebildet, weil die Nutzleistung in der Regel wesentlich größer als die Störleistung ist.

Um die Stärke des Fehlers zu bestimmen, wird der *mean squared error*, kurz *MSE* verwendet. Dieser lässt sich bestimmen, indem innerhalb eines Fensters der Größe $m \times n$ die quadratischen Abstände zwischen dem Originalbild I und dem rekonstruierten Bild K aufsummiert werden:

$$MSE = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i, j) - K(i, j)]^2. \quad (3.4)$$

Der PSNR ist der Quotient, gebildet aus dem maximal möglichen Farbwert MAX_I , des untersuchten Farbkanals und dem MSE :

$$PSNR = 10 \times \log_{10} \left(\frac{MAX_I^2}{MSE} \right) \quad (3.5)$$

$$= 20 \times \log_{10} \left(\frac{MAX_I}{\sqrt{MSE}} \right) \quad (3.6)$$

$$= 20 \times \log_{10}(MAX_I) - 10 \times \log_{10}(MSE) \quad (3.7)$$

Bei einer Farbtiefe von 8 Bit pro Kanal stehen Werte von 30 - 40 dB für ein geringes Störsignal und gleichzeitig für eine höhere Übereinstimmung der Bilder. Der PSNR ist ein häufig genutztes Maß, um die Ähnlichkeit zweier Bilder zu bestimmen.

3.5 SSIM

Im Gegensatz zum PSNR ist die von Wang *et al.* entwickelte Metrik *structural similarity index* ein auf die menschliche Wahrnehmung angepasstes Modell[WBSS04]. Die Metrik basiert auf der Idee, dass die Struktur der abgebildeten Objekte unabhängig von Beleuchtung und Kontrast gemessen werden kann.

Die mittlere Intensität des diskreten Signals x lässt sich in einem lokalen Bereich der Größe N folgendermaßen bestimmen:

$$\mu_x = \frac{1}{N} \sum_{i=1}^N x_i. \quad (3.8)$$

Die Vergleichsfunktion, welche die Beleuchtung zwischen zwei Signalen x und y misst, wird durch den Term $l(x, y)$ repräsentiert. Dieser setzt die mittleren Intensitäten μ_x und μ_y wie folgt ins Verhältnis:

$$l(x, y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1}. \quad (3.9)$$

Die Konstante C_1 wird benötigt, um die numerische Stabilität zu gewährleisten. Analog zur Abbildung 3.3 wird im Anschluss an die Beleuchtungsmessung die mittlere Intensität von beiden Signalen $x - \mu_x$ und $y - \mu_y$ abgezogen.

Der Kontrast wird unter Verwendung der Standardabweichung σ_x approximiert. In diskreter Form lässt sich diese mit der folgenden Formel berechnen:

$$\sigma_x = \left(\frac{1}{N-1} \sum_{i=1}^N (x_i - \mu_x)^2 \right)^{\frac{1}{2}}. \quad (3.10)$$

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

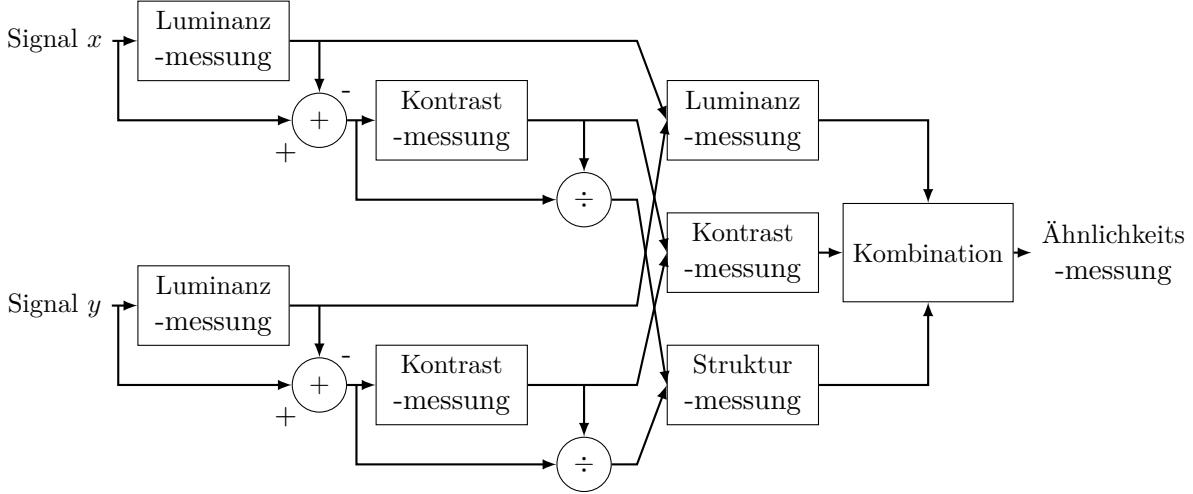


Abbildung 3.3: Das Modell zeigt das Vorgehen von dem SSIM-Algorithmus [WBSS04].

$$c(x, y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2}. \quad (3.11)$$

Wie bei der Berechnung der mittleren Intensität wird eine Konstante C_2 eingeführt. An dieser Stelle wird das Signal normalisiert, indem es durch seine eigene Standardabweichung geteilt wird und die beiden Signale $(x - \mu_x)/\sigma_x$, $(y - \mu_y)/\sigma_y$ entstehen. Der Vergleich der strukturellen Eigenschaften $s(x, y)$ wird mit den normalisierten Signalen 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 $s(x, y) = 1$ geben, genau dann und nur dann, wenn gilt, dass $x = y$.

Die folgende Definition erfüllt alle drei Forderungen:

$$s(x, y) = \frac{\sigma_{xy} + C_3}{\sigma_x + \sigma_y + C_3}. \quad (3.12)$$

Die Kovarianz σ_{xy} berechnet sich folgendermaßen:

$$\sigma_{xy} = \frac{1}{N-1} \sum_{i=1}^N (x_i - \mu_x)(y_i - \mu_y). \quad (3.13)$$

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, dass 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:

$$SSIM(x, y) = [l(x, y)]^\alpha \times [c(x, y)]^\beta \times [s(x, y)]^\gamma. \quad (3.14)$$

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. Dabei werden 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 umformen, wobei w_i die Gewichtung an dem Punkt i bezeichnet:

$$\mu_x = \sum_{i=1}^N w_i x_i \quad (3.15)$$

$$\sigma_x = \left(\sum_{i=1}^N w_i (x_i - \mu_x)^2 \right)^{\frac{1}{2}} \quad (3.16)$$

$$\sigma_{xy} = \sum_{i=1}^N w_i (x_i - \mu_x)(y_i - \mu_y). \quad (3.17)$$

Damit lässt sich der $SSIM(x, y)$ durch die folgende Gleichung bestimmen:

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}. \quad (3.18)$$

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

$$MSSIM(X, Y) = \frac{1}{M} \sum_{j=1}^M SSIM(x_j, y_j). \quad (3.19)$$

Die Evaluation wird mit beiden Metriken durchgeführt, um eine bessere Vergleichbarkeit mit anderen Arbeiten zu gewährleisten.

4 Methodik

In dieser Arbeit werden zwei grundsätzliche Verfahren untersucht, um die Tiefeninformationen zu übertragen und als Dreiecksnetz darzustellen. Der erste Ansatz besteht darin, das Tiefenbild als solches verlustfrei zu komprimieren und zu übertragen. Der Client empfängt und dekodiert das Tiefenbild und erzeugt daraus ein voll vernetztes Dreiecksnetz. Im zweiten Ansatz wird aus dem Tiefenbild ein adaptives Dreiecksnetz mit Hilfe der Delaunay-Triangulierung vom Server konstruiert und an den Client übertragen.

Die Erzeugung von adaptiven Dreiecksnetzen lässt sich als eine Form der verlustbehafteten geometrischen Kompression bezeichnen. Die zu übertragende Informationsmenge kann durch Valenz-basierte Kodierung weiter verringern werden.

Weil die Tiefenwerte des Tiefenbildes komplett und ohne Informationsverlust im voll vernetzen Dreiecksnetz enthalten sind, liefert dieses Verfahren Ergebnisse, die als *Ground-Truth*-Information zum Vergleich der adaptiven Vernetzung dienen.

4.1 Vollvernetzung

Bei der Vollvernetzung wird für jeden Pixel aus dem Tiefenbild D ein Vertex erzeugt. Die entstandenen Vertices werden über Kanten zu Dreiecken miteinander verbunden. Die Abbildung 4.1 zeigt drei unterschiedliche Varianten, wie sich die Vertices zu voll vernetzten Dreiecksnetzen verbinden lassen. Alle drei Varianten wurden im Rahmen dieser Arbeit implementiert, weisen bei entsprechend großen Auflösungen jedoch keine Unterschiede hinsichtlich der Qualität der Darstellung auf.

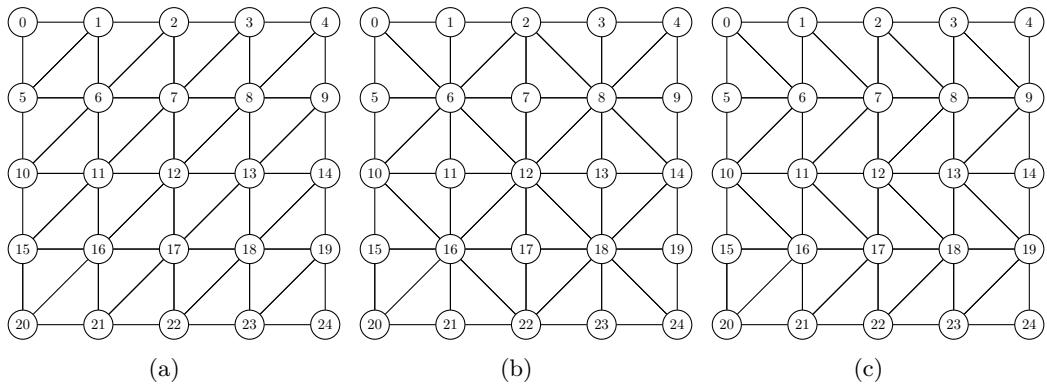


Abbildung 4.1: Darstellung von drei verschiedenen Varianten der Vollvernetzung.

Die Anzahl der Vertices entspricht der Anzahl der insgesamt gegebenen Bildpunkte. Die x- und y-Komponenten lassen sich aus der bekannten Auflösung vorausberechnen und müssen im Falle einer Auflösungsänderung neu bestimmt werden. Für jeden Vertex wird die z-Komponente mit Hilfe der Pixelwerte aus dem Tiefenbild während der Bildsynthese gesetzt. Die Anzahl der Dreiecke lässt sich dabei wie folgt berechnen: $2(w - 1)(h - 1)$.

Problematisch bei diesem Verfahren ist, dass die Anzahl der Dreiecke von der Auflösung abhängt

und der Rechenaufwand der clientseitigen Bildsynthese linear mit der Auflösung steigt.

4.2 Delaunay-Triangulierung

Die zweite in dieser Arbeit untersuchte Variante zur Erzeugung eines Dreiecksnetzes aus einem Tiefenbild D , besteht in der serverseitigen Erzeugung einer adaptiven Vernetzung. Prinzipiell lässt sich die Vorgehensweise in drei Schritte unterteilen: Im ersten Schritt wird eine Menge von Punkten $P \subset D$ definiert. Dabei muss die Menge P so gewählt werden, dass die Werteverläufe des Tiefenbildes möglichst optimal wiedergegeben werden. Der zweite Schritt besteht in der Vernetzung der Punkte $p \in P$ zu Dreiecken. Das dabei eingesetzte Verfahren ist die Delaunay-Triangulierung. Abschließend kann das erzeugte Netz in einem dritten Schritt weiter verfeinert werden.

Entscheidend für das Ergebnisnetz und für die Geschwindigkeit des gesamten Verfahrens ist die Wahl der Punktmenge P . Das Ziel bei der Verteilung der Punktemenge P ist es, eine hohe Punktdichte im Bereich von Kanten und gekrümmten Flächen zu erzielen. Im Gegensatz dazu sollten planare Regionen möglichst keine Punkte enthalten, weil diese mit wenigen Dreiecken approximiert werden können. Ausgangspunkt für die Erzeugung der Punktmenge P sind die Gradienten des Tiefenbildes ∇_x und ∇_y . Die Gradienten werden mit Hilfe des Sobel-Operators berechnet. Dazu wird die erste Ableitung berechnet und gleichzeitig wird die dazu orthogonale Richtung geglättet. Mit Hilfe einer Faltungsmatrix der Größe 3×3 lassen sich die Gradienten berechnen:

$$\nabla_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} * D, \quad \nabla_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * D.$$

Auf diese Weise entstehen für ein Tiefenbild zwei Gradientenbilder wie in Abbildung 4.2 dargestellt. Zur Erzeugung der Punktmenge P wurden zwei Verfahren untersucht, die im Folgenden näher betrachtet werden.

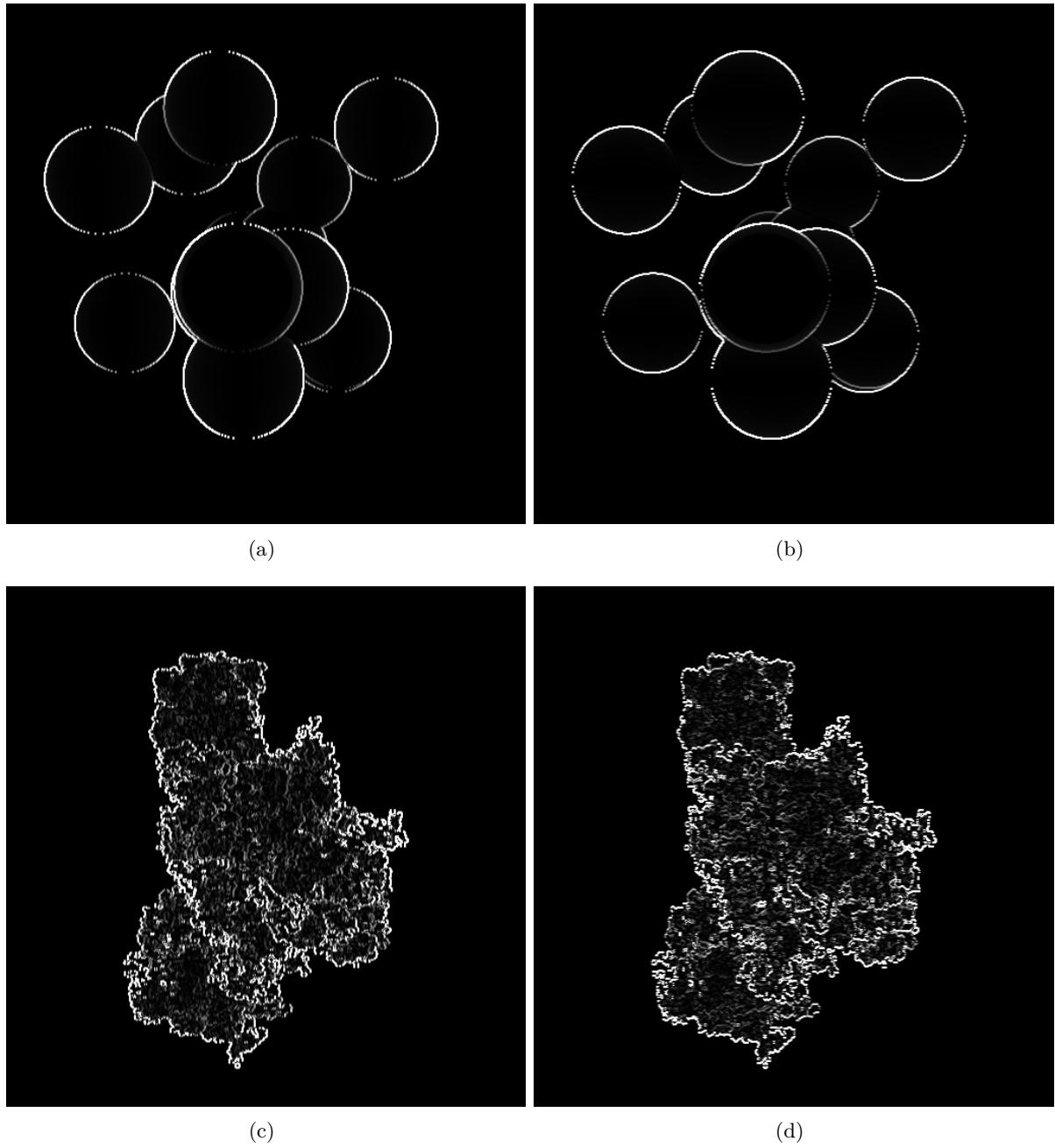


Abbildung 4.2: Gradientenbilder beider Datensätze vom jeweils ersten Tiefenbild.

4.2.1 Quadtree

Die erste in dieser Arbeit vorgestellte Methode zur Erzeugung der Punktmenge P basiert auf einer Quadtree-Datenstruktur. Ein Quadtree ist eine Baumdatenstruktur, in der jeder Knoten maximal vier Kindknoten hat. Im Rahmen dieser Arbeit repräsentiert jeder Knoten eine rechteckige Region R des Tiefenbildes D . Eine Region R wird durch seine Eckpunkte definiert. Um die Baumdatenstruktur zu erzeugen, wird ein Wurzelknoten definiert, dessen Region $R = (0, 0, w - 1, h - 1)$ den Ausmaßen des Tiefenbildes D entspricht. Im Anschluss wird die Region des Wurzelknotens in vier gleichgroße Teilregionen zerlegt. Diese Teilregionen sind die Kindknoten des Wurzelknotens. Alle Kindknoten werden auf diese Weise sukzessive weiter unterteilt, bis die Tiefe des erzeugten Baumes eine maximale Baumtiefe t_{max} erreicht.

Die Kindknoten der Baumtiefe t_{max} enthalten selbst keine weiteren Kindknoten und werden als Blattknoten bezeichnet, die Regionen, die sie repräsentieren, als Blattregionen. Die maximale Tiefe des Quadtrees ist dabei invers proportional zur Breite der Blattregionen. Mit dem Parameter t_{max} kann die Qualität und die Kompression des erzeugten Netzes sowie der Berechnungsaufwand skaliert werden. Die erzeugte Baumdatenstruktur hängt von der Auflösung der Tiefenbilder ab und muss nur neu erzeugt werden, wenn sich die Auflösung oder der Parameter t_{max} ändert.

Im nächsten Schritt wird mit Hilfe der Baumdatenstruktur die Punktmenge P erzeugt. Zu diesem Zweck wird der Baum, beginnend mit den Blattknoten, zum Wurzelknoten traversiert. Mit Hilfe der Gradienten kann die Region R eines Knoten auf Planarität getestet werden. Ist eine Region R nicht planar, dann werden die Eckpunkte von R zur Menge P hinzugefügt.

Um zu überprüfen, ob eine Region R als planar bezeichnet werden kann, lässt sich mit Hilfe der Differenz zwischen dem maximalen und dem minimalen Wert der Gradienten innerhalb von R bestimmen:

$$c_x = \max_R \nabla_x - \min_R \nabla_x \quad (4.1)$$

$$c_y = \max_R \nabla_y - \min_R \nabla_y. \quad (4.2)$$

Wenn entweder c_x oder c_y größer als ein zuvor definierter Schwellwert ist, dann ist die Region R nicht planar, und die Eckpunkte werden der Menge P hinzugefügt. Die Werte für c_x und c_y der inneren Konten lassen sich effizient aus den bereits berechneten maximalen und minimalen Gradienten seiner Kindknoten berechnen.

Die Wahl des Schwellwerts ist entscheidend für die Anzahl der eingefügten Punkte in P . Dieser kann adaptiv gewählt werden, um die Komplexität des Dreiecksnetzes anzupassen.

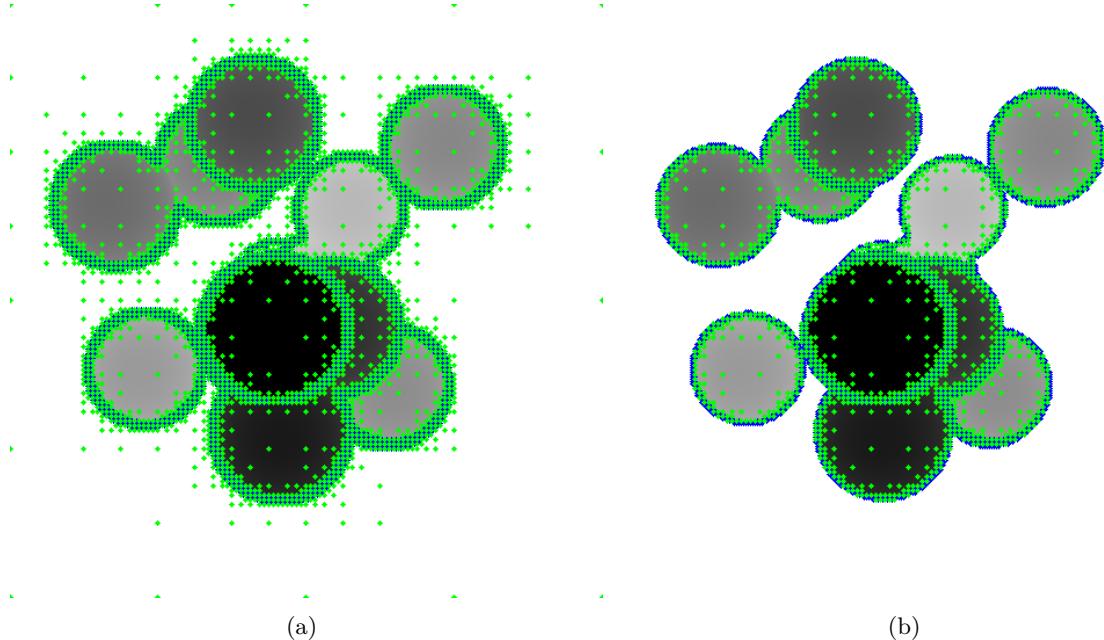
Für den eigentlichen Planaritätstest unterscheidet sich dieser Schwellwert in Abhängigkeit, ob es sich um einen inneren Knoten oder einen Blattknoten handelt. Der Schwellwert für die Blattknoten wird mit l bezeichnet und der für die inneren Knoten mit i . Blattknoten sollten vorrangig Tiefenuntersprünge, also Kanten detektieren und somit die Kontur der dargestellten Objekte kennzeichnen. Innere Knoten dagegen repräsentieren die Oberfläche der Objekte und werden gesetzt, um Verläufe von nicht planaren Flächen abzubilden zu können. Weil Tiefensprünge größere Gradienten zur Folge haben als gekrümmte Oberflächen, sollte der Schwellwert i für die inneren Knoten kleiner sein als der Schwellwert l für die Blattknoten, $i < l$. Die Werte von i und l liegen im Intervall $[0, 1]$.

Das Ergebnis der Traversierung des Baums ist eine Menge von Punkten P , die sich aus den Eckpunkten der getesteten Regionen zusammensetzt, in denen die Gradientendifferenzen den entsprechenden Schwellwert überschreiten.

Diese Punktmenge P kann weiter reduziert werden, indem getestet wird, ob ein Punkt $p \in P$ zum Hintergrund gehört. Das ist der Fall, wenn der Tiefenwert des Tiefenbildes an Stelle p dem maximalen Tiefenwert entspricht. Zur Erinnerung: Die Tiefenwerte liegen linear zwischen den beiden Clippingebenen, wobei ein großer Wert bedeutet, dass dieser nahe der Bildschirmebene liegt.

4.2.2 Floyd-Steinberg

Der zweite in dieser Arbeit vorgestellte Ansatz zur Erzeugung der Punktmenge P basiert auf einem Fehler-Diffusionsverfahren. Im ersten Schritt wird aus den beiden Gradientenbildern ∇_x, ∇_y ein Merkmalsbild σ erzeugt:

Abbildung 4.3: Szene *TestSpheres*, Baumtiefe 9, l=5, i=4

$$\sigma_d = \left(\frac{\|\nabla d\|}{A} \right)^\gamma. \quad (4.3)$$

Dabei entspricht der Nenner von 4.3 der Länge des Gradientenvektors ∇d , des Pixels $d \in D$. Dieser wird mit A , dem größtmöglichen Wert von $\|\nabla d\|$, normiert. Der Parameter $\gamma \in [0, 1]$ wird genutzt, um die Ausprägung von schwachen Kanten im Merkmalsbild σ zu erhöhen.

Um aus dem Merkmalsbild σ die Punktmenge P zu erzeugen, kommt der Floyd-Steinberg-Algorithmus zum Einsatz. Bei dem Floyd-Steinberg-Algorithmus handelt es sich um einen einfachen und effizienten Dithering-Algorithmus, der auf einem, Fehler-Diffusionsverfahren fußt. Das Merkmalsbild wird mit Hilfe des Schwellwerts δ binarisiert. Zu diesem Zweck wird das Merkmalsbild σ Pixel für Pixel zeilenweise durchlaufen. Der betrachtete Pixel wird mit einem Schwellwert δ verglichen. Wenn der Wert des Pixels größer als der Schwellwert ist, dann wird der Wert des Pixels mit δ , andernfalls mit 0 ersetzt. Zudem wird durch die Binarisierung resultierende Quantisierungsfehler des Pixels $d \in D$ entsprechend der Abbildung 4.4 auf die benachbarten Pixel verteilt.

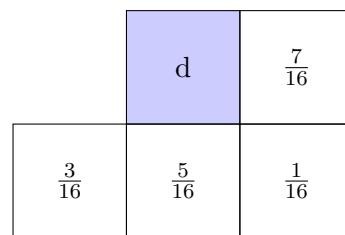


Abbildung 4.4: Darstellung der gewichteten Verteilung des Quantisierungsfehlers [Wik14].

Dadurch lässt sich die Verteilung des Quantisierungsfehlers ohne einen zusätzlichen Puffer in einem einzigen Durchlauf berechnen. Das folgende Pseudocodebeispiel 4.1 verdeutlicht das Verfahren:

```

1  for each y
2      for each x
3          oldpixel      := pixel[x][y]
4          newpixel      := (pixel[x][y] > δ) ? δ : 0
5          pixel[x][y]    := newpixel
6          quant_error   := oldpixel - newpixel
7          pixel[x+1][y]  := pixel[x+1][y] + quant_error * 7 / 16
8          pixel[x-1][y+1] := pixel[x-1][y+1] + quant_error * 3 / 16
9          pixel[x][y+1]  := pixel[x][y+1] + quant_error * 5 / 16
10         pixel[x+1][y+1] := pixel[x+1][y+1] + quant_error * 1 / 16

```

Listing 4.1: Floyd-Steinberg-Algorithmus [Wik14]

Immer wenn die Bindung erfüllt wird und der Wert eines Pixels auf δ gesetzt wird, dann wird die Menge P mit der Position des gerade bearbeiteten Pixels erweitert. Die Abbildung 4.5 zeigt verschiedene bereits auf δ und 0 reduzierte Merkmalsbilder σ .

Ein Vorteil dieses Verfahrens ist die effiziente Erzeugung der Menge P . Mit Hilfe der beiden Parameter γ, δ lässt sich die Anzahl der Punktmenge P variieren. Dabei spielt insbesondere δ eine entscheidende Rolle, da durch diese Größe die Punktdichte angepasst werden kann. Wenn δ kleine Werte annimmt, dann wird die Punktdichte erhöht und im Fall großer Werten reduziert.

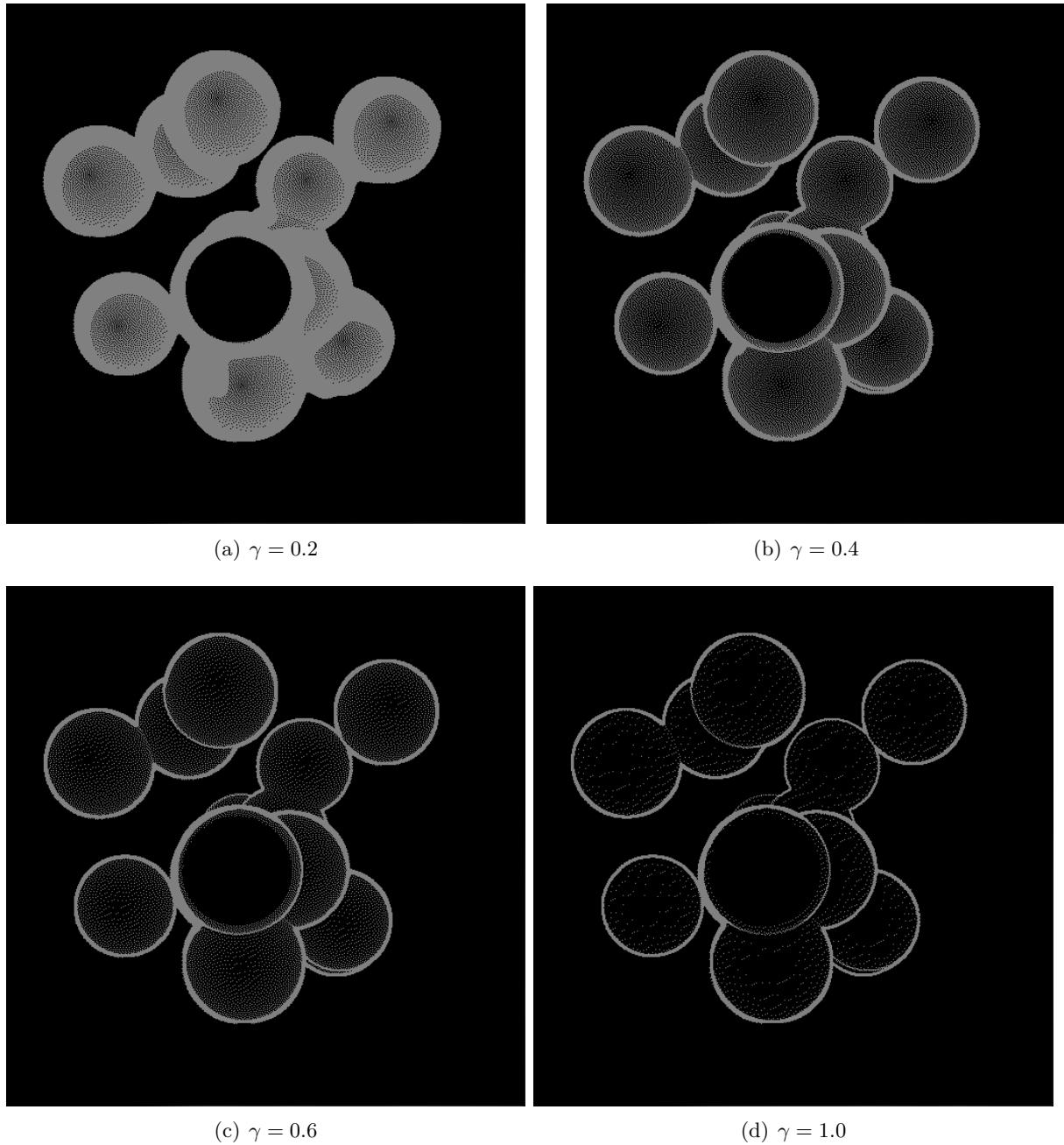


Abbildung 4.5: Die Bilder a bis d zeigen das Merkmalsbild in Abhängigkeit des Parameters γ bei einem festgesetzten Schwellwert von 0.5.

4.2.3 Erzeugung des Netzes

Die aus den beiden Methoden erzeugten Punkte $p \in P$ können jetzt mit Hilfe der Delaunay-Triangulierung zu einem Dreiecksnetz miteinander verbunden werden. In dieser Arbeit wird zu diesem Zweck die Delaunay-Triangulierung aus dem opencv-Framework verwendet.

4.2.4 Netzoptimierung

Die auf diese Weise entstandene Netze können in Abhängigkeit der Eingabedaten zwei Arten von Artefakten enthalten. 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, wodurch Kanten nicht korrekt vom Netz abgebildet werden. Diese Art von Artefakte kommen zustande, wenn die Blattregionen zu große Bildregionen repräsentieren. 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 nähert, während gleichzeitig eine bestimmte Länge in der xy -Ebene überschritten wird.

Die Eckpunkte der zu prüfenden Kante werden im folgenden mit p_1 und p_2 bezeichnet. Der Punkt p_m bezeichnet dabei den Mittelpunkt der Kante p_1p_2 .

$$\frac{d_{p_1} + d_{p_2}}{2} - d_{p_m} < \varepsilon \quad (4.4)$$

Eine Kante ist nicht valide, wenn der eigentliche Tiefenwert $D(p_m)$. 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.

$$\frac{|d_{p_1} - d_{p_2}|}{\|p_1 - p_2\| (d_{p_1} + d_{p_2})} < \alpha \quad (4.5)$$

Ist der Wert des ersten Terms der Gleichung 4.5 einer Kante größer als der Schwellwert T_{angle} , dann steht diese Kante nahezu senkrecht auf der Bildebene, und sie liegt mit hoher Wahrscheinlichkeit über einem 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:

$$|(d(p_2) - d(p_m)) - (d(p_m) - d(p_1))| < \beta \quad (4.6)$$

Um ein Dreieck zu unterteilen, muss eine Fallunterscheidung durchgeführt werden. Hat das Dreieck nur eine valide Kante p_1p_2 , dann müssen die anderen beiden Kanten wie in der Abbildung 4.6 geteilt werden, und es entstehen aus dem ursprünglichen Dreieck drei neue Dreiecke.

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.

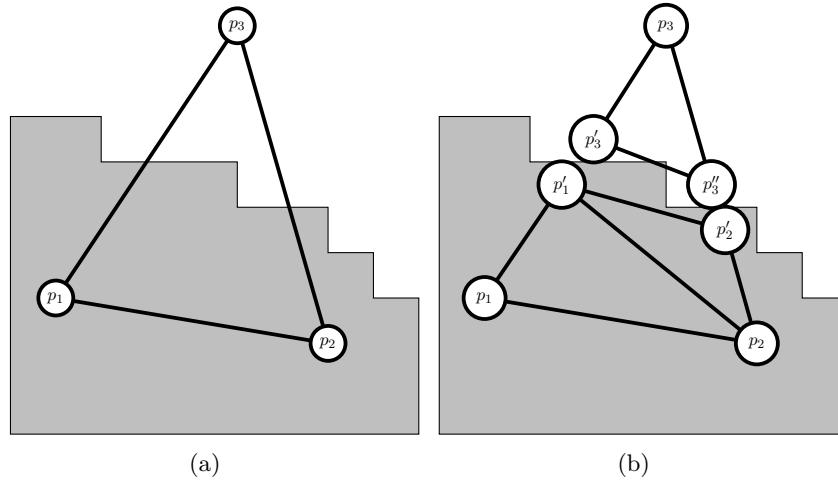


Abbildung 4.6: Die Bilder a und b zeigen die Unterteilung des Dreiecks $p_1p_2p_3$, wenn die zwei Kanten p_1p_3 , p_2p_3 nicht valide sind.

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, dass alle drei Kanten nicht valide sind, werden sechs neue Vertices eingefügt. Die Abbildung 4.7 verdeutlicht diesen Fall. Die Berechnung aller Vertices geschieht analog zu dem Vertex p'_3 aus dem vorhergehenden Betrachtung mit einer validen Kante.

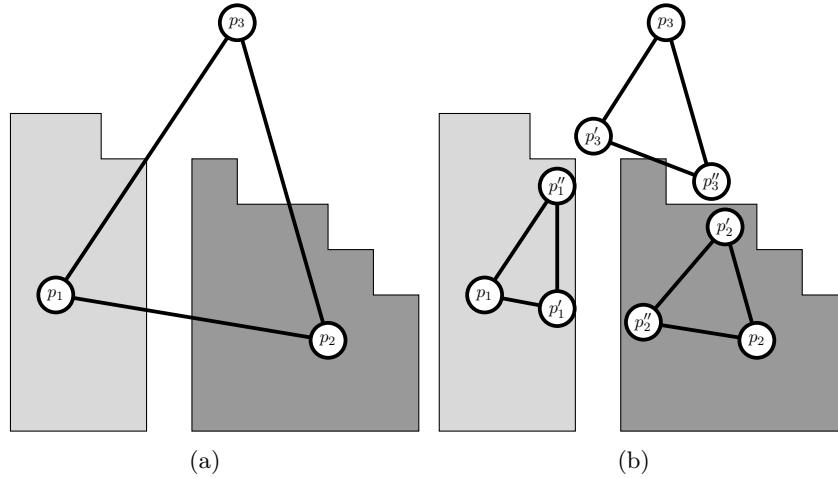


Abbildung 4.7: Die Bilder a und b zeigen die Unterteilung des Dreiecks $p_1p_2p_3$, wenn keine Kante valide ist.

Es muss beachtet werden, dass jedes neu erzeugte Dreieck auf Kollinearität zu überprüfen und gegebenenfalls zu verwerfen.

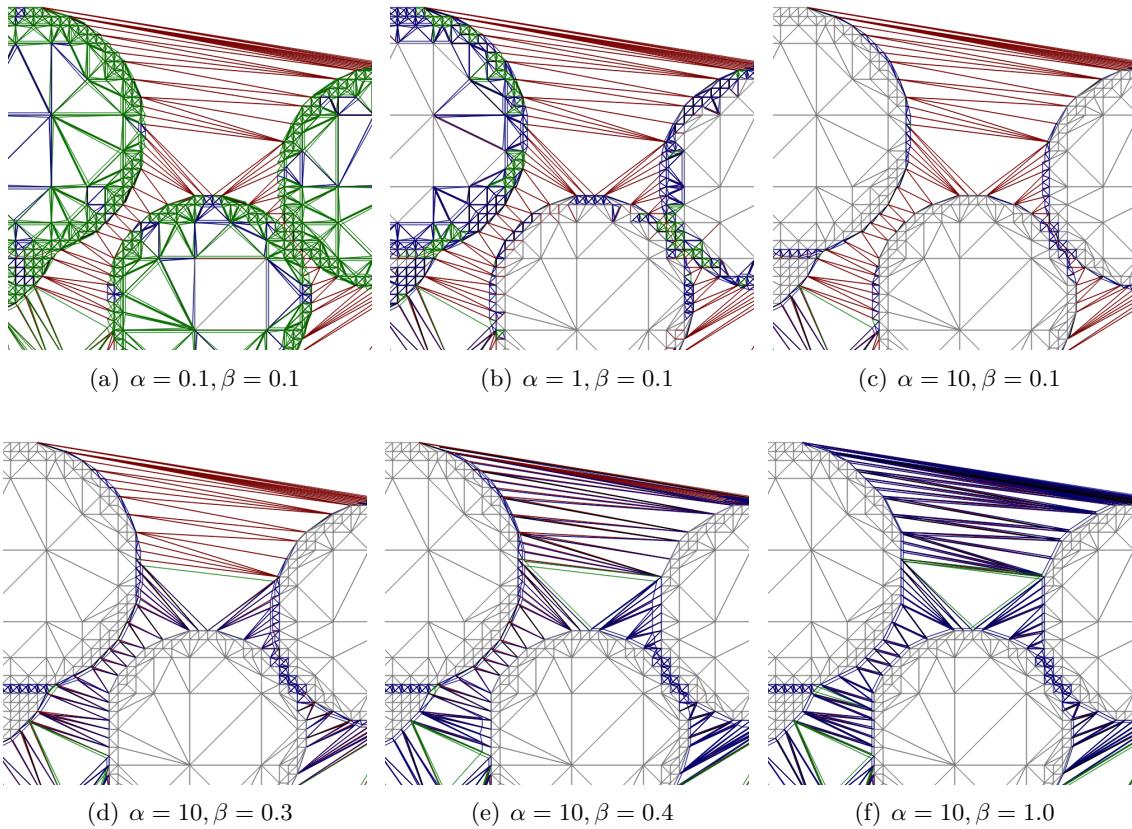


Abbildung 4.8: Auswirkung der Parameter auf die Nachbearbeitung. Das Netz wurde mit der *Quadtree*-Methode erzeugt. Dazu wurde $t_{max} = 8, l = 0.7, i = 0.6$ gewählt.

Die Pixelpositionen in den Gleichungen 4.5 und 4.6 werden in Texturkoordinaten angegeben und die Farbwerte der Pixel auf das Intervall $[0, 1]$ transformiert. Zu beachten ist, dass der Parameter α von der Baumtiefe abhängt.

Dabei zeigt die Abbildung 4.8 den Einfluss von α und β bei der Nachbearbeitung. Rot markierte Kanten sind nicht valide und werden nicht zum endgültigen Netz hinzugefügt. Alle grünen Kanten gehören zu Dreiecken, die aus einem Dreieck ohne valide Kante erzeugt wurden. Analog dazu wurden Dreiecke, hervorgehoben durch blaue Kanten, aus Dreiecken erzeugt mit nur einer validen Kante. Kollineare Dreiecke fallen zu schwarzen Kanten oder Punkten zusammen.

Wenn die Parameter $\alpha = 0$ und $\beta = 0$ gewählt werden, degeneriert das Netz. Es ist zu erkennen, dass mit größer werden α der Einfluss auf gekrümmte Oberflächen abnimmt. Mit dem Parameter β lässt sich die Verbindungsneigung beeinflussen.

Die Abbildung 4.9 zeigt ein komplettes nachbearbeitetes Dreiecksnetz.

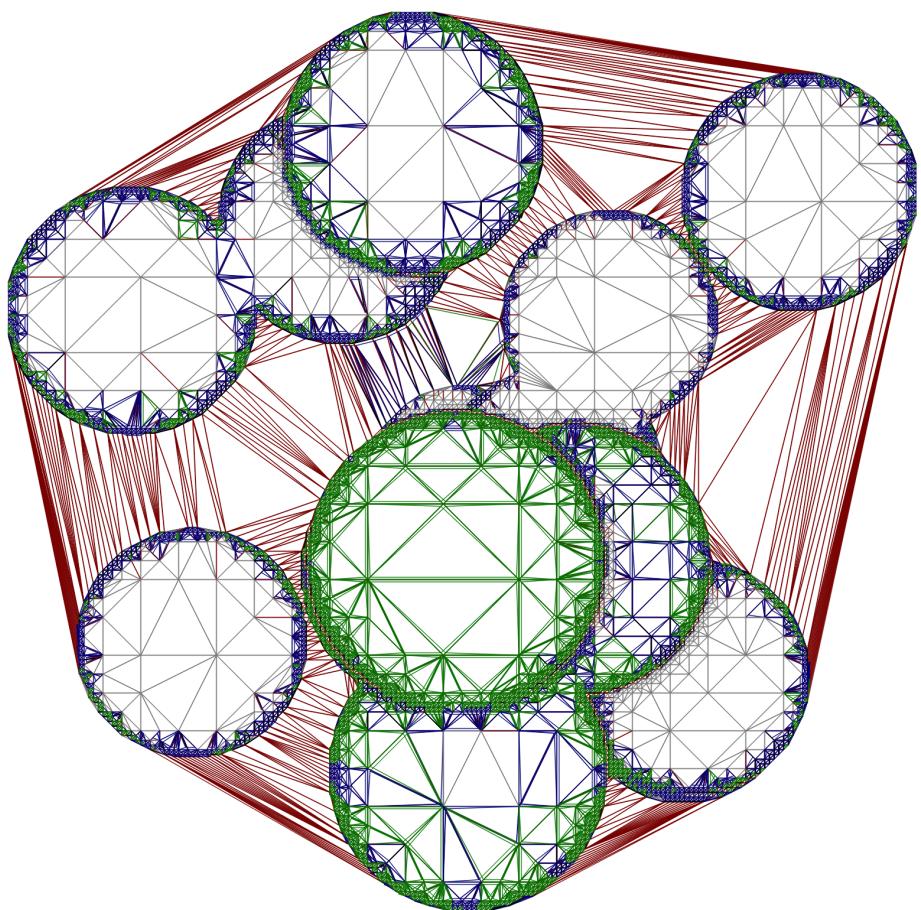


Abbildung 4.9: Nachbearbeitetes Netz des ersten Frames aus dem *TestSpheres*-Datensatz. Das Netz wurde mit der *Quadtree*-Methode erzeugt. Dazu wurde $t_{max} = 9, l = 0.5, i = 0.4, \alpha = 1.0, \beta = 0.1$ gewählt.

5 Implementierung

Die Client-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 zwischen den Verbindungsteilnehmern erlaubt, ohne auf Methoden wie Long Polling zurückgreifen zu müssen. Die clientseitige Extrapolation der Bilddaten wird mit Hilfe von WebGL durchgeführt. 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 in Kombination mit einer hohen Rechengeschwindigkeit.

Dieses Kapitel wird dabei in zwei Hauptbereiche unterteilt. Im ersten werden Details über die Kommunikation zwischen Client und Server behandelt. Der zweite Abschnitt beschäftigt sich mit implementierungsspezifischen Details der Algorithmen.

5.1 Details zur Kommunikation

Die Server-Anwendung wurde mit der Sprache c++ entwickelt. Dabei wird auf die Implementierung der WebSocket-Serverkomponente aus dem QT-Framework zurückgegriffen. Der Server-Prozess besteht aus einem Thread. Zur Verwaltung der Anfragen wird das Event-Managementsystem von QT verwendet. Beim Start der Serveranwendung werden die Szenen geladen. Aus diesen wird eine *Playlist* erstellt, diese vereint die Informationen aller Szenen. Im Anschluss daran wird die Standardparameterkonfiguration geladen, und der Server wartet darauf, dass sich ein Client verbindet.

Die Kommunikation zwischen Server- und die Client-Komponente basiert auf dem JSON-RPC 2.0 Protokoll. Die Abkürzung JSON-RPC steht für *JavaScript Object Notation Remote Procedure Call*. Es handelt sich dabei um ein Protokoll, das den Aufbau und die Nominatur von Nachrichten beschreibt. Im Speziellen geht es dabei um Nachrichten, die dazu dienen, dass der Kommunikationsteilnehmer bestimmte Methoden mit bestimmten Parameterkonfigurationen startet. Es basiert auf dem textbasierten Datenformat JSON. Nachrichten werden in Anfragen und Antworten unterteilt. Mit Hilfe einer *id* lassen sich Nachrichten bei asynchroner Kommunikation zuordnen. Es werden drei verschiedene Typen von Anfragen unterschieden, *Request*, *Notification* und *Batch Request*. Ein *Request* ist eine Anfrage, welche die Antwort eines Kommunikationsteilnehmers verlangt. Diese Form der Anfragen kann zur Synchronisierung der Nachrichtenübertragung genutzt werden. Im Gegensatz dazu entfällt bei einer *Notification* die *id*, weil keine Antwort verlangt wird. Der *Batch Request* fasst eine Menge von *Requests* und *Notifications* in einer einzigen Anfrage zusammen.

Die in dieser Arbeit implementierte Variante von JSON-RPC 2.0 wird auf Anfragen vom Typ *Request* beschränkt, weil die gesamte Kommunikation synchronisiert stattfindet.

Eine JSON-RPC-Anfrage ist ein JSON-Objekt, das sich aus vier Teilen zusammensetzt. Der erste ist ein String mit der JSON-RPC Version gefolgt vom Namen der aufzurufenden Methode. Im Anschluss folgt ein JSON-Objekt, das die Parameter der aufzurufenden Methode enthält, und zum Schluss kommt die *id* der Anfrage. Als Beispielanfrage soll der Aufruf der Methode

resize dienen. Möchte der Client die Auflösung ändern, schickt er die Anfrage dem Beispiel 5.1 entsprechend zum Server.

```

1  {
2      "jsonrpc"      : "2.0",
3      "method"       : "resize",
4      "params"       : [512, 512],
5      "id",          : 0
6  }

```

Listing 5.1: RPC-Request

Hat der Server die Anfrage empfangen, ruft dieser darauf hin die Methode *resize* mit den entsprechenden Parametern aus der Anfrage auf. Im Anschluss daran wird eine Antwort zurückgesendet. Eine JSON-RPC-Antwort ist ebenfalls ein JSON-Objekt, das aus drei Teilen besteht. Der erste Teil ist ebenfalls die JSON-RPC-Version als String kodiert. Der zweite Teil enthält den Rückgabewert in Form eines JSON-Objektes, und zum Schluss folgt die *id* der Anfrage, wie im Beispiel 5.2 demonstriert.

```

1  {
2      "jsonrpc"      : "2.0",
3      "result"       : [512, 512],
4      "id",          : 0
5  }

```

Listing 5.2: RPC-Response

Der Server stellt eine Menge von Methoden zur Verfügung, die durch den Client initiiert und von dem Serverprozess ausführt werden.

Eine Beispielkommunikation, dargestellt im Sequenzdiagramm 5.1, veranschaulicht den Remote-Aufruf verschiedener Methoden.

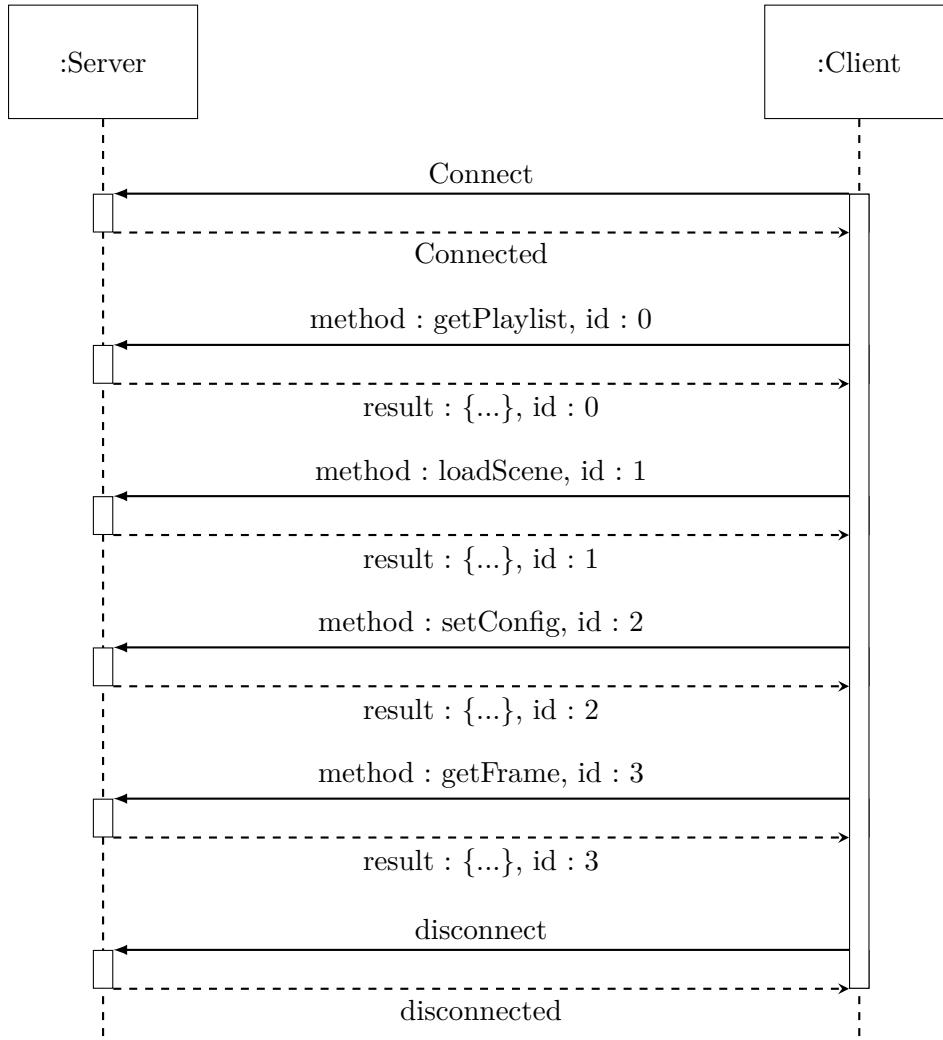


Abbildung 5.1: Das Sequenzdiagramm zeigt eine Beispielkommunikation, um ein Bild vom Server anzufordern.

Kann sich der Client erfolgreich mit dem Server verbinden, fragt er die *Playlist* ab. Diese liefert dem Client alle Informationen über die zur Verfügung stehenden Szenen. Sie enthält für jeden Frame einer Szene dessen Kamerainformationen. Nachdem der Client alle Szenen kennt, kann er den Server mitteilen, welche dieser laden soll. Anschließend wird mit der Methode *setConfig* eine gewählte Parameterkonfiguration gesetzt. Mit Hilfe der *Playlist* kann ein beliebiger Frame der geladenen Szene angefordert werden. Wird der Frame empfangen, kann dieser vom Client anschließend extrapoliert werden. Neben den vorgestellten stehen zusätzliche Methoden zum Starten einer Messung beziehungsweise zum Speichern eines Bildes durch den Server zur Verfügung.

5.2 Details zur Implementierung der Algorithmen

In diesem Abschnitt werden Details zur Implementierung der vorgestellten Algorithmen genauer betrachtet. Begonnen wird hier mit den Farbbildern. Um eine verlustbehaftete Komprimierung ermöglichen zu können, kommt das JPEG-Format zum Einsatz. Anschließend müssen die Bildinformationen mit Base64 kodiert werden, damit diese als String in ein JSON-Objekt integriert werden können.

5.2.1 Vollvernetzung

Im Fall der Vollvernetzung wird neben dem Farbbild auch das Tiefenbild mit Base64 kodiert und als String in ein JSON-Objekt eingebettet. Im Gegensatz zu den Farbbildern werden die Tiefenbilder verlustfrei als PNG komprimiert.

Die Dekodierung von Farb- und Tiefenbildern kann mit Hilfe des Bildcontainers nativ durch vom Browser durchgeführt werden. Im Bezug auf das Tiefenbild ergibt sich ein Problem. Dieser Bildcontainer erlaubt für jeden Farbkanal ausschließlich eine Farbtiefe von 8 Bit. Damit ein Shaderprogramm trotzdem auf 16 Bit-Informationen aus einer Textur zurückgreifen kann, muss der 16 Bit-Wert auf zwei Farbkanäle aufgeteilt werden, und die Rekonstruktion erfolgt im Shader. Diese Aufteilung findet auf dem Server statt, bevor das Bild mit Base64 kodiert wird. Auf den roten Farbkanal entfallen die ersten 8 Bit und die zweiten 8 Bit, der 16 Bit-Zahl werden auf den grünen Farbkanal ausgelagert. Die Abbildung 5.2 verdeutlicht die Aufteilung.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
d_{low}								d_{up}							

Abbildung 5.2: Hier wird die Aufteilung einer 16 Bit-Zahl auf die Farbkanäle, links rot und rechts grün, visuell verdeutlicht. Das Schlüsselwort d_{up} bezeichnet die ersten 8 Bit und d_{low} die zweiten 8 Bit.

Um aus d_{up} und d_{low} die 16 Bit-Zahl d zu erhalten, genügt es, den Wert von d_{up} zuerst mit 255 zu multiplizieren und den Wert von d_{low} zu addieren, wie in der Gleichung 5.1 beschrieben.

$$d = d_{low} + d_{up} \cdot 256. \quad (5.1)$$

Die rekonstruierte Zahl d aus der Gleichung 5.1 liegt im ganzzahligen Intervall $[0, 65535]$. Beim Zugriff auf die Farbwerte einer Textur im Shader werden diese jedoch normiert. Das bedeutet im Fall eines 8 Bit-Farbkanals wird jeder Wert durch den Maximalwert 255 geteilt. Auf diese Weise werden die Werte eines Pixels in das Intervall $[0, 1]$ überführt. Um eine 16 Bit-Zahl in dieses Intervall zu überführen, muss sie durch den Wert 65535 geteilt werden. Aus diesem Zusammenhang ergibt sich die folgende Gleichung zur Rekonstruktion der Zahl d :

$$d = \frac{255 \cdot \frac{d_{low}}{255} + \frac{d_{up}}{255} \cdot 255 \cdot 265}{2^{16} - 1}. \quad (5.2)$$

Die Zahl 65535 lässt sich in das Produkt $65535 = 255 \cdot 257$ umschreiben und dadurch kann die Gleichung 5.2 weiter vereinfacht werden:

$$d = \frac{255 \cdot \left(\frac{d_{low}}{255} + \frac{d_{up}}{255} \cdot 265 \right)}{255 \cdot 257} \quad (5.3)$$

$$= \frac{d_{low}}{255} \cdot \frac{1}{257} + \frac{d_{up}}{255} \cdot \frac{256}{257} \quad (5.4)$$

Im Shader kann die 16 Bit-Zahl deshalb einfach entsprechend der Gleichung 5.4 rekonstruieren. Der Farbwert des roten Farbkanals wird mit $(1/257)$ multipliziert und der des grünen mit

(256/257). Im Anschluss daran werden die beiden Werte aufsummiert, und der Wert der 16 Bit Zahl steht im Shader zur Verfügung.

Im Fall der Vollvernetzung werden alle Dreiecke, die zum Hintergrund gehören, transparent gezeichnet. Zusätzlich wird im Vertex-Shader der Gradient des bearbeiteten Vertices berechnet. Der x- und y-Anteil des Gradienten wird mit dem Schwellwert g verglichen. Ist einer dieser beiden Gradientkomponenten größer als der Schwellwert g , wird der Alphakanal an dieser Stelle ebenfalls auf 0 gesetzt.

Damit verdeckte Fragmente durch die transparenten sichtbar werden, kommt ein 3- und ein 5-Pass *depth peeling*-Verfahren zum Einsatz.

5.2.2 Delaunay-Triangulierung

Beim *Quadtree* wird der Bildbereich sukzessiv in vier gleichgroße Regionen aufgeteilt, deren Eckpunkte in der Punktmenge P auftauchen können. Die Abbildung 5.3 veranschaulicht die Aufteilung.

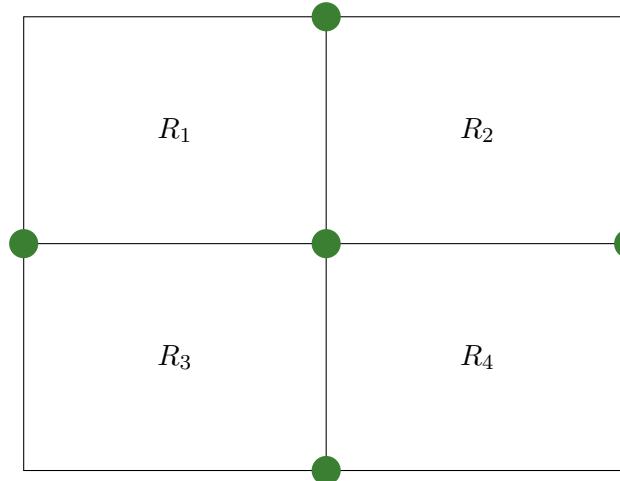


Abbildung 5.3: Unterteilung eines Rechtecks in vier Teilflächen. An den grünen markierten Positionen gibt es mehrere Möglichkeiten, wie die Eckpunkte der betreffenden Rechtecke gesetzt werden können.

Dabei gibt es mehrere Möglichkeiten, wie die Eckpunkte der Teilfenster gewählt werden können. Einerseits können die Flächen exakt gleich groß gewählt werden. Mit dem Effekt, dass dadurch viele kleine Dreiecke entstehen, dass die grün markierten Punkte sich aus mindestens zwei Eckpunkten zusammensetzen, die sich jeweils nur um eine Pixeleinheit voneinander unterscheiden.

Die zweite Möglichkeit besteht darin, für derartige Eckpunkte dieselbe Koordinate zu wählen. Auf diese Weise lässt sich die Anzahl der in der Punktmenge P enthaltenen Punkte bei gleichbleibender Bildqualität senken. In dieser Arbeit wird die zweite Variante verwendet.

Bei der Erzeugung der Gradientenbilder mit dem Sobel-operator entsteht ein Rand mit der Breite von einem Pixel. Dieser wird auch bei der Binärisierung von dem Merkmalsbild σ ausgelassen.

Um die Netzoptimierung durchführen zu können, müssen die Punkte $p'_1, p''_1, p'_{2,2}, p''_{2,2}$, sowie p'_3 und p''_3 bestimmt werden. Diese Aufgabe wird mit Hilfe einer Linienrasterisierung gelöst. Dabei wird nicht zwischen den Pixeln interpoliert.

6 Ergebnisse

Zu Beginn dieses Kapitels wird beschrieben, wie die Ergebnisse erhoben werden. Anschließend werden die Parameterkonfigurationen und deren Resultate vorgestellt. Dabei auftretende Besonderheiten werden kenntlich gemacht.

Die Vollvernetzung und die beiden Varianten der Delaunay-Triangulierung werden mit den beiden *Ground-Truth*-Datensätzen, *TestSpheres* und *CoolRandom*, evaluiert. Der erste Frame einer Szene dient als Referenz für die Extrapolation. Aus dessen Tiefenbild wird mit dem gewählten Algorithmus ein Dreiecksnetz erzeugt. Zusammen mit dem dazugehörigen Farbbild wird es zum Client gesendet. Anschließend wird das Dreiecksnetz, texturiert mit dem Farbbild, aus allen Kameraperspektiven der Szene gezeichnet, und die dabei entstandenen Bilder werden zurück zum Server gesendet. Die Qualität aller extrapolierten Bilder wird durch den PSNR und den SSIM bestimmt. Zusätzlich wird gemessen, wie lange der Server für die Kodierung benötigt und wie lange der Client für die Extrapolation eines Bildes braucht.

Für die Vollvernetzung wurde das Tiefenbild mit 8 und 16 Bit kodiert. Es wurde ein 3-Pass und ein 5-Pass Verfahren verwendet, um die Transparenz mit *depth peeling* zu realisieren. Und schließlich wurden die Schwellwerte $g \in \{1, 0.9, 0.8\}$ für den Vergleich mit dem Gradienten getestet. Die besten Ergebnisse werden mit 16 Bit, $g = 1$, und dem 5-Pass Verfahren erzielt. Diese Konfiguration dient als Basis für den Vergleich mit den Delaunay-Verfahren.

Im Fall der Delaunay-Triangulation wird zwischen den beiden Punktgenerierungsverfahren *Quadtree* und *Floyd-Steinberg* unterschieden. Für den *Quadtree*-Ansatz werden generell alle Hintergrundpixel verworfen. Die Bäume werden mit den maximalen Baumtiefen $t_{max} \in \{10, 9, 8\}$ erzeugt. Für jede Baumtiefe t_{max} wurden die Schwellwertkonfigurationen $l \in \{0.1, 0.2, \dots, 1.0\}$ und $i \in \{0.1, \dots, l\}$ gemessen. Für alle $l > 0.6$ und $i = 0.6$ wurde zusätzlich die Netzverfeinerung durchgeführt. Mit den Schwellwerten $\alpha \in \{0.1, 1, 10, 100, 1000\}$ und $\beta \in \{0.1, 0.2, \dots, 1.0\}$. Die Erzeugung der Punktemenge, erhoben mit dem *Floyd-Steinberg*-Fehlerdiffusionsverfahren, wird mit den Parametern $\delta, \gamma \in \{0.1, 0.2, \dots, 1.0\}$ evaluiert. Die folgenden Diagramme zeigen ausgewählte Beispiele von den erhobenen Daten. Dabei wird mit den besten Ergebnissen begonnen:

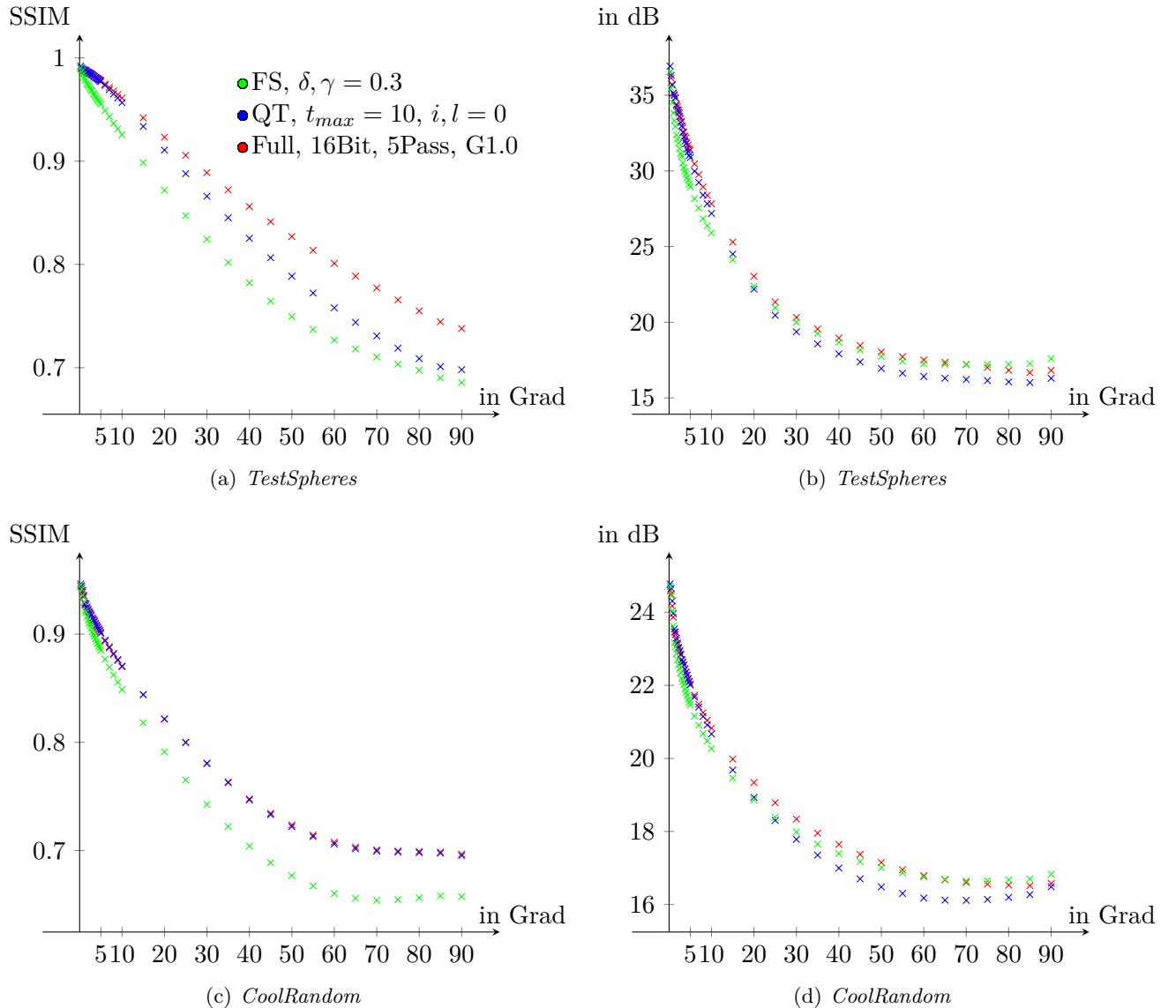


Abbildung 6.1: Die Diagramme a und b zeigen die Ergebnisse des Datensatzes *TestSpheres*, während c und d die Resultate des Datensatzes *CoolRandom* abtragen. Dabei zeigen a und c die Ergebnisse des SSIM, während in b und d die Ergebnisse des PSNR gezeigt werden.

Die Abbildung 6.1 zeigt die besten Resultate der Vollvernetzung vom *Quadtree* sowie von der *FloydSteinberg*-Variante der Delaunay-Triangulierung. Für beide Szenen, gemessen mit dem SSIM, liefert die Vollvernetzung insgesamt die besten Ergebnisse dicht gefolgt vom *Quadtree*-Ansatz. Die schlechtesten Ergebnisse liefert der *FloydSteinberg*-Ansatz. Interessant ist, dass die Güte der Ergebnisse von dem *CoolRandom*-Datensatz, gemessen mit dem PSNR, schon bei den ersten extrapolierten Frames unter 25 dB liegt. Außerdem fällt beim PSNR auf, dass die Werte ab einem Winkel von 70 Grad wieder leicht ansteigen.

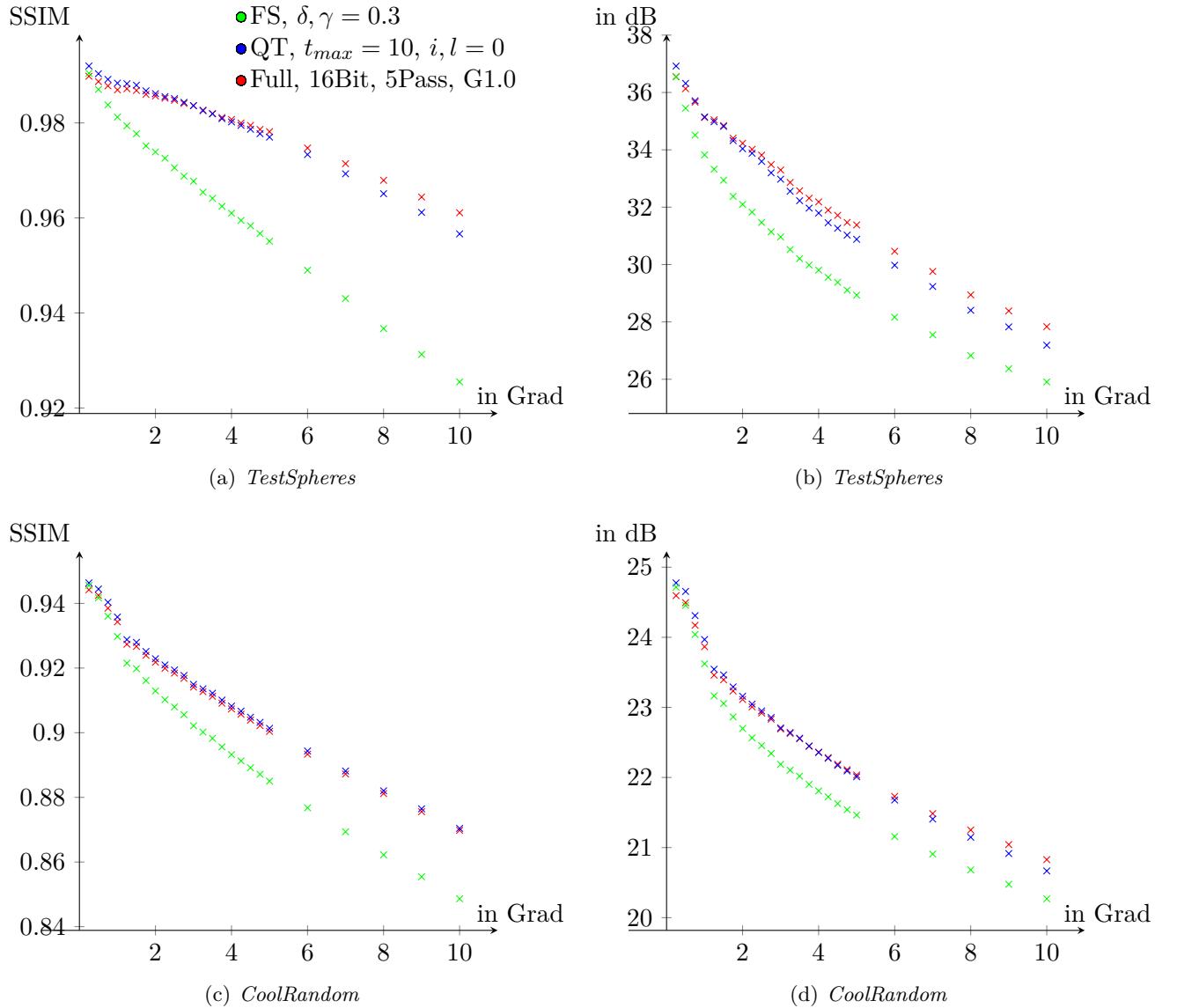


Abbildung 6.2: Darstellung der Graphen aus der Abbildung 6.1 im Winkelintervall von 0 bis 10 Grad.

Die Abbildung 6.2 zeigt die gleichen Parameterkonfigurationen wie die Abbildung 6.1, dieses Mal jedoch beschränkt auf die Kamerawinkelunterschiede von 0 bis 10 Grad. Hier zeigt sich sowohl mit dem SSIM als auch beim PSNR ein signifikanter Unterschied zwischen dem FloydSteinberg-Ansatz und den anderen beiden Methoden. Der *Quadtree*-Ansatz ist in den ersten Frames sogar besser als die Vollvernetzung.

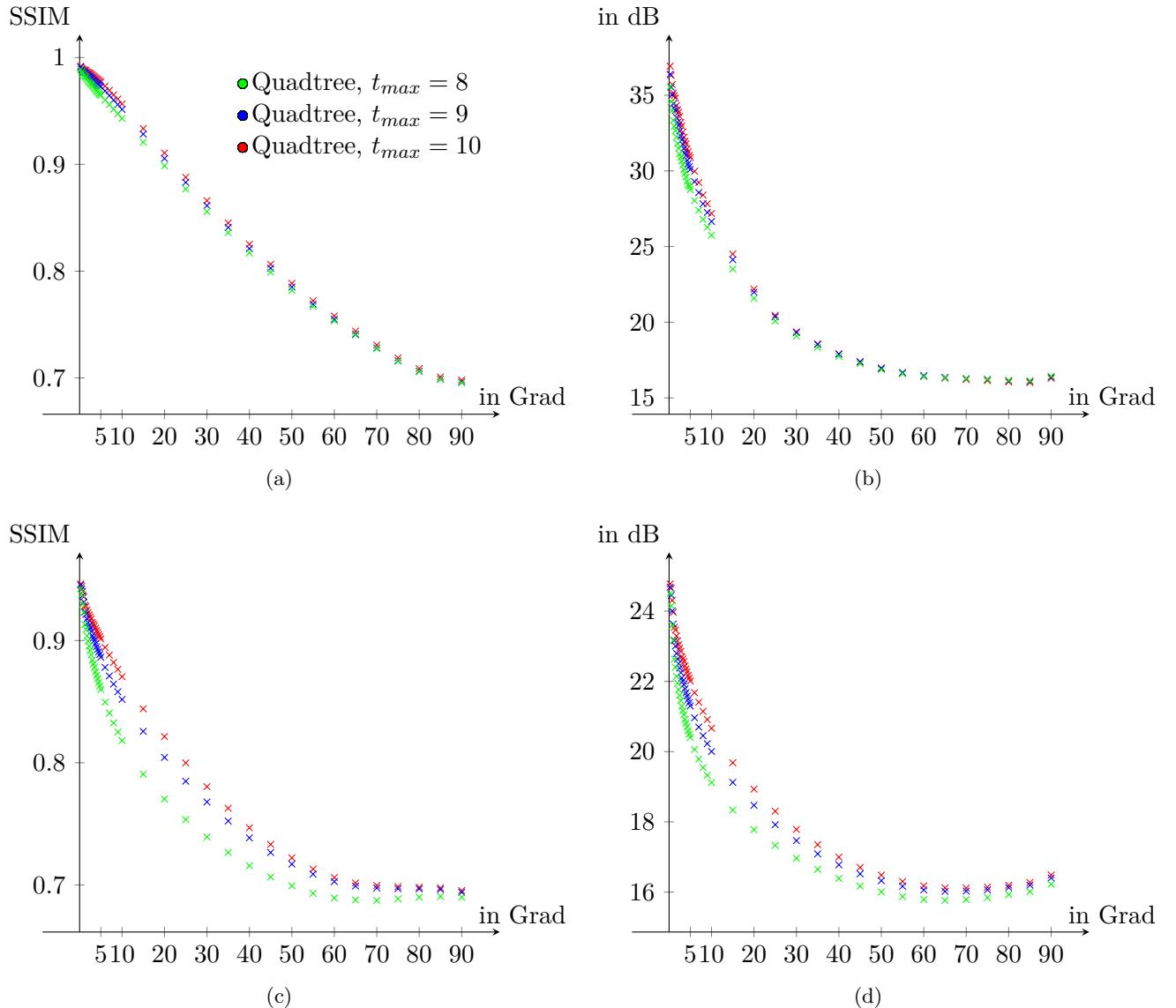


Abbildung 6.3: Die Diagramme a und b zeigen die Ergebnisse des Datensatzes *TestSpheres*, während c und d die Resultate des Datensatzes *CoolRandom* abtragen. Dabei zeigen a und c die Ergebnisse des SSIM, während in b und d die Ergebnisse des PSNR gezeigt werden.

Die Abbildung 6.3 zeigt die Ergebnisse *Quadtrees*-Ansatzes mit unterschiedlichen maximalen Baumtiefen. Dabei zeigt sich, dass der Unterschied in der Szene *TestSpheres* nur gering ist. Anders fallen die Ergebnisse für die *CoolRandom* Szene aus. Hier zeigt sich ein deutlicher Unterschied zwischen drei Varianten, insbesondere im Intervall von 5 bis 50 Grad. Die Gütwerte für die Baumtiefe $t_{max} = 8$ liegen dabei deutlich unterhalb der anderen. Im Fall der *CoolRandom* Szene zeigt sich bei der Messung mit dem PSNR eine erneute Verbesserung bei Winkelunterschieden, die größer als 70 Grad sind.

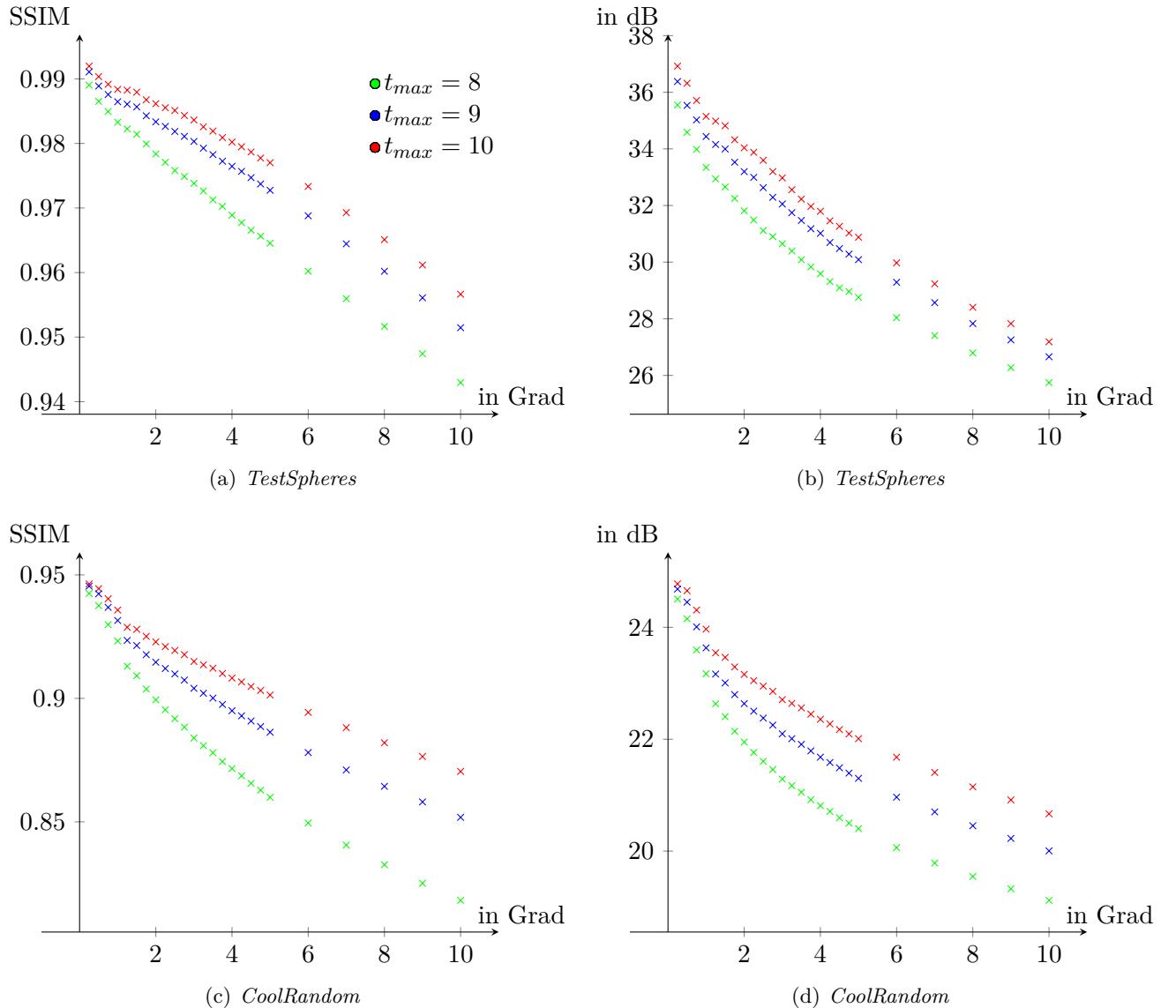


Abbildung 6.4: Die Diagramme a und b zeigen die Ergebnisse des Datensatzes *TestSpheres*, während c und d die Resultate des Datensatzes *CoolRandom* abtragen. Dabei zeigen a und c die Ergebnisse des SSIM, während in b und d die Ergebnisse des PSNR gezeigt werden.

Die Diagramme aus der Abbildung 6.4 zeigen die gleichen Graphen wie die Abbildung 6.3, allerdings im Intervall von 0 bis 10 Grad. Der Unterschied der Güte zwischen den Parameterkonfigurationen wird hierbei noch deutlicher.

Um die Schwellwerte l und i zu untersuchen, wird der SSIM von einem konstanten Kamerawinkelunterschied betrachtet. Zu diesem Zweck wird der 20 Grad-Winkel gewählt, weil wie in der Abbildung 6.3 zu erkennen ist, eine höhere Streuung der Werte zu erwarten ist. Dadurch werden die Qualitätsunterschiede zwischen den Parameterkonfigurationen besonders deutlich.

In der Abbildung 6.5 sind die Güte und die Netzkonstruktionszeit des *TestSpheres*-Datensatzes abgebildet. Es ist zu erkennen, dass die Güte mit der maximalen Baumtiefe t_{max} abnimmt. Der Güteunterschied zwischen den einzelnen Konfigurationen von l und i ist bei konstanter Baumtiefe allerdings gering. Anders sieht es mit der Konstruktionszeit aus; hier zeigen sich

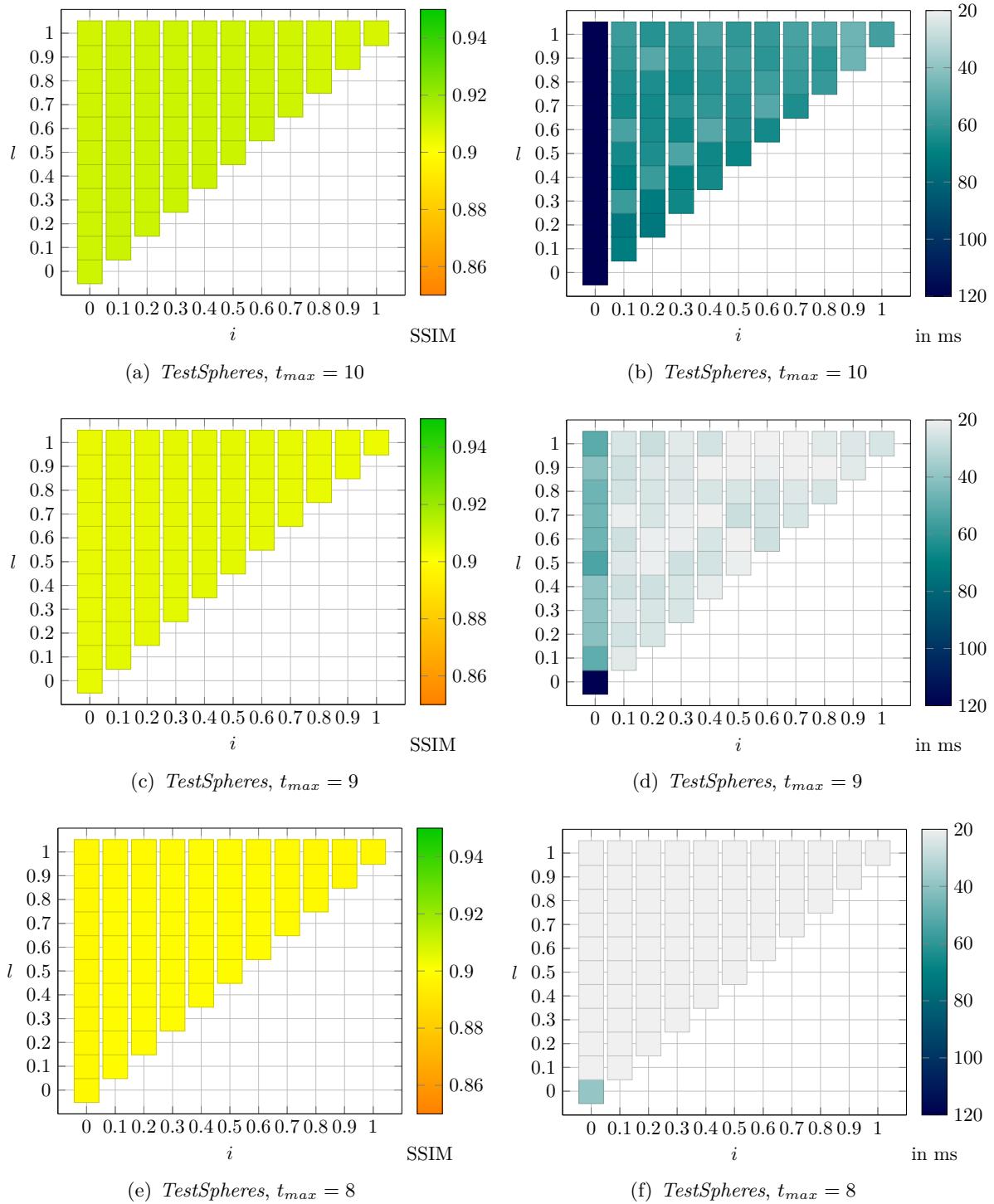


Abbildung 6.5: Ergebnisse des *Quadtree*-Ansatz mit *TestSpheres*Datensatz, bei einem konstanten Kamerawinkelunterschied von 20 Grad. Dabei liegt die maximale Baumtiefe in a,b bei 10, in c,d bei 9 und in e,f bei 8. Die Diagramme auf der linken Seite zeigen die Güte mit Hilfe des SSIM Wertes. Auf der rechten Seite wird die Konstruktionszeit des Netzes abgetragen, die der Server benötigt.

deutliche Unterschiede. Bei $t_{max} = 10$ und für $i = 0$ ist die Konstruktionszeit für alle l größer als 120 ms. Für die maximale Baumtiefe $t_{max} = 9$ überschreitet die Konfiguration $l, i = 0$ die Marke von 120 ms.

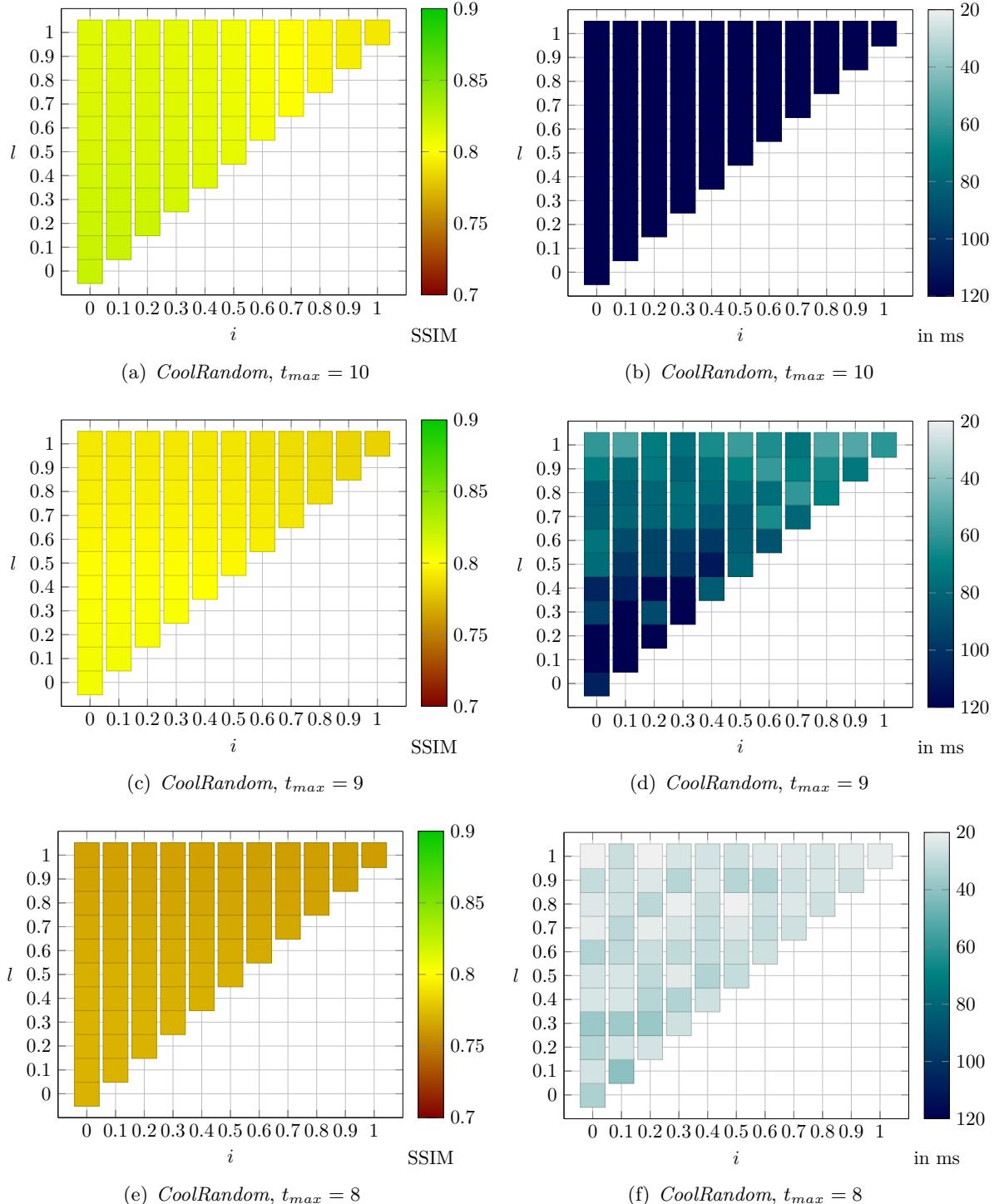


Abbildung 6.6: Ergebnisse des *Quadtree*-Ansatz mit *CoolRandom*, bei einem konstanten Kamera-winkelunterschied von 20 Grad. Dabei liegt die maximale Baumtiefe in a,b bei 10, in c,d bei 9 und in e,f bei 8. Die Diagramme auf der linken Seite zeigen die Güte mit Hilfe des SSIM-Wertes. Auf der rechten Seite wird die Gesamtkonstruktionszeit des Netzes abgetragen.

Vergleichbar zur Abbildung 6.5 werden Güte und Konstruktionszeit in der Abbildung 6.6 für den *CoolRandom*-Datensatz dargestellt. Die Güte variiert wesentlich stärker als im *TestSpheres*-Datensatz. Zu erkennen ist eine Abnahme der Güte mit steigenden l und i , wenn die maximale Baumtiefe $t_{max} = 10$ beträgt. Für die anderen beiden maximalen Baumtiefen 8 und 9 ist die Güte für alle i und l relativ konstant. Auch bei dieser Szene zeigt sich das bei $t_{max} = 10$ die Konstruktionszeiten am größten sind. Unabhängig von l und i überschreiten alle die Marke von 120 ms. Bei der maximalen Baumtiefe $t_{max} = 9$ zeigt sich eine deutliche Veränderung der Konstruktionszeiten bezüglich l und i . Je kleiner l und i ist, umso größer ist die Berechnungszeit. Anders sieht es bei $t_{max} = 8$ aus. In diesem Fall sind alle Berechnungszeiten unterhalb von 40 ms.

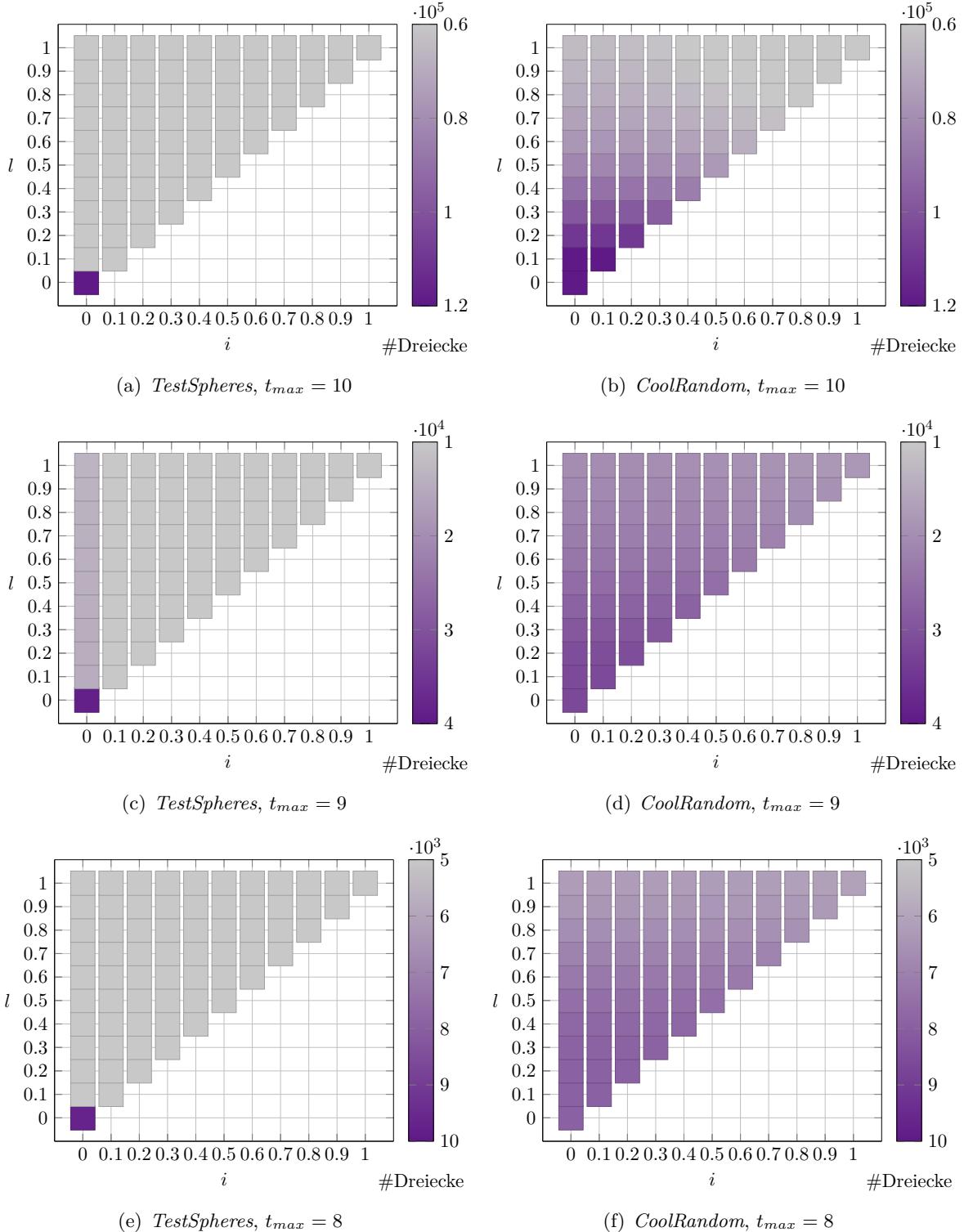


Abbildung 6.7: Die Diagramme zeigen für den *Quadtree*-Ansatz bei einem konstanten Kamerawinkelunterschied von 20 Grad die Anzahl der Dreiecke in Abhängigkeit der Parameter l, i und t_{max} . Dabei liegt die maximale Baumtiefe in a,b bei 10, in c,d bei 9 und in e,f bei 8. Die Diagramme auf der linken Seite zeigen die Anzahl der Dreiecke von der *TestSpheres* Szene. Auf der rechten Seite wird die Anzahl der Dreiecke von *CoolRandom* dargestellt.

In der Abbildung 6.7 ist die Anzahl der Dreiecke in Abhangigkeit von l und i sowie der maximalen Baumtiefe t_{max} abgebildet. Wird der Datensatz *TestSpheres* betrachtet, zeigt sich fur alle t_{max} , dass bei der Parameterkonfiguration $l = 0, i = 0$ wesentlich mehr Dreiecke erzeugt werden als bei allen anderen Konfigurationen von l und i . Fur den Datensatz *CoolRandom* sieht es etwas anders aus. Hier zeigt sich, dass die Anzahl der Dreiecke sinkt, wenn l und i groer werden, vor allem, wenn $t_{max} = 10$ ist.

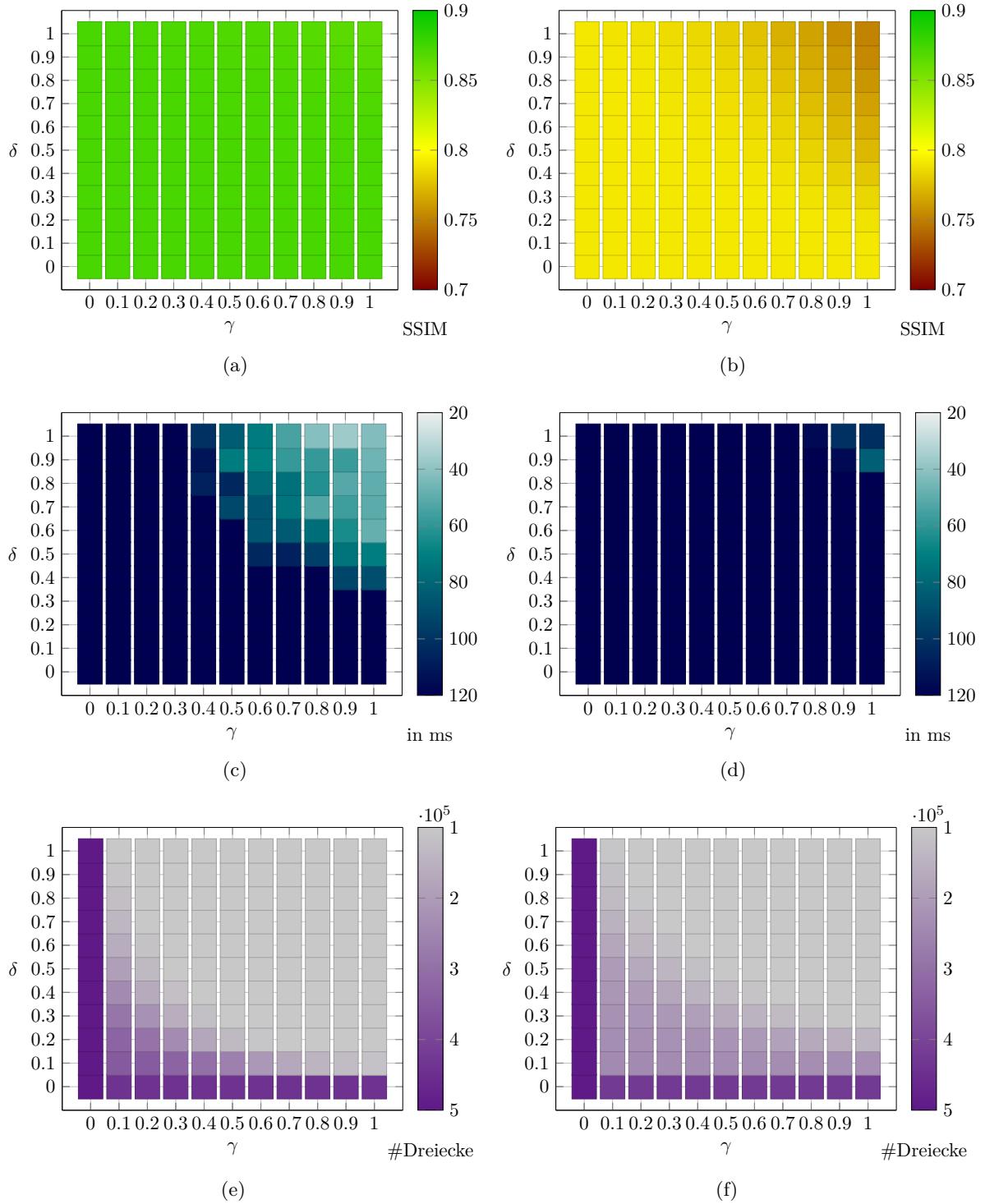


Abbildung 6.8: Die Diagramme a, b stellen die Ergebnisse von dem *CoolRandom* Datensatz und c, d die des *TestSpheres*-Datensatzes, erhoben mit *FloydSteinberg*-Ansatz dar. Die Diagramme auf der linken Seite zeigen die Güte mit Hilfe des SSIM, und die Diagramme auf der rechten Seite bilden die Gesamtkompressionszeit des Netzes ab.

Analog zu den Schwellwerten der *Quadtree*-Methode werden die Parameter δ und γ vom *Floyd-Steinberg*-Verfahren in der Abbildung 6.8 dargestellt. Dabei sind die Ergebnisse für die Szene *TestSpheres* hinsichtlich der Güte für alle Konfigurationen größer als 0.9. Bei der Berechnungszeit

zeigt sich, dass erst ab den Konfigurationen $\delta > 0.5$ und $\gamma > 0.5$ die Konstruktionszeiten kleiner als 100 ms werden. Die Anzahl der Dreiecke ist bei den Konfigurationen $\gamma = 0$ oder für $\delta = 0$ am Höchsten. Anders sieht es bei dem Datensatz *CoolRandom* aus. Die Gütwerte liegen alle unterhalb von 0.8. Dabei zeigt sich eine zunehmende Verschlechterung für Konfigurationen mit $\delta > 0.6$ und $\gamma > 0.6$. Die Konstruktionszeiten des Netzes sind für alle Parameterkonfigurationen größer als 120 ms bis auf die Konfiguration $\gamma = 1$ und $\delta = 0.9$. Mit der Anzahl der Dreiecke verhält es sich ähnlich wie bei der Szene *TestSpheres*.

Der Netzoptimierungsschritt wird ausschließlich mit dem *Quadtree*-Ansatz untersucht, da dessen Ergebnisse weitaus besser sind als die der FloydSteinberg-Methode. Um die Parameter α und β auszuwerten, werden l und i konstant auf einen Wert fixiert. Zu diesem Zweck wurde $l = 0.7$ und $i = 0.6$ gewählt.

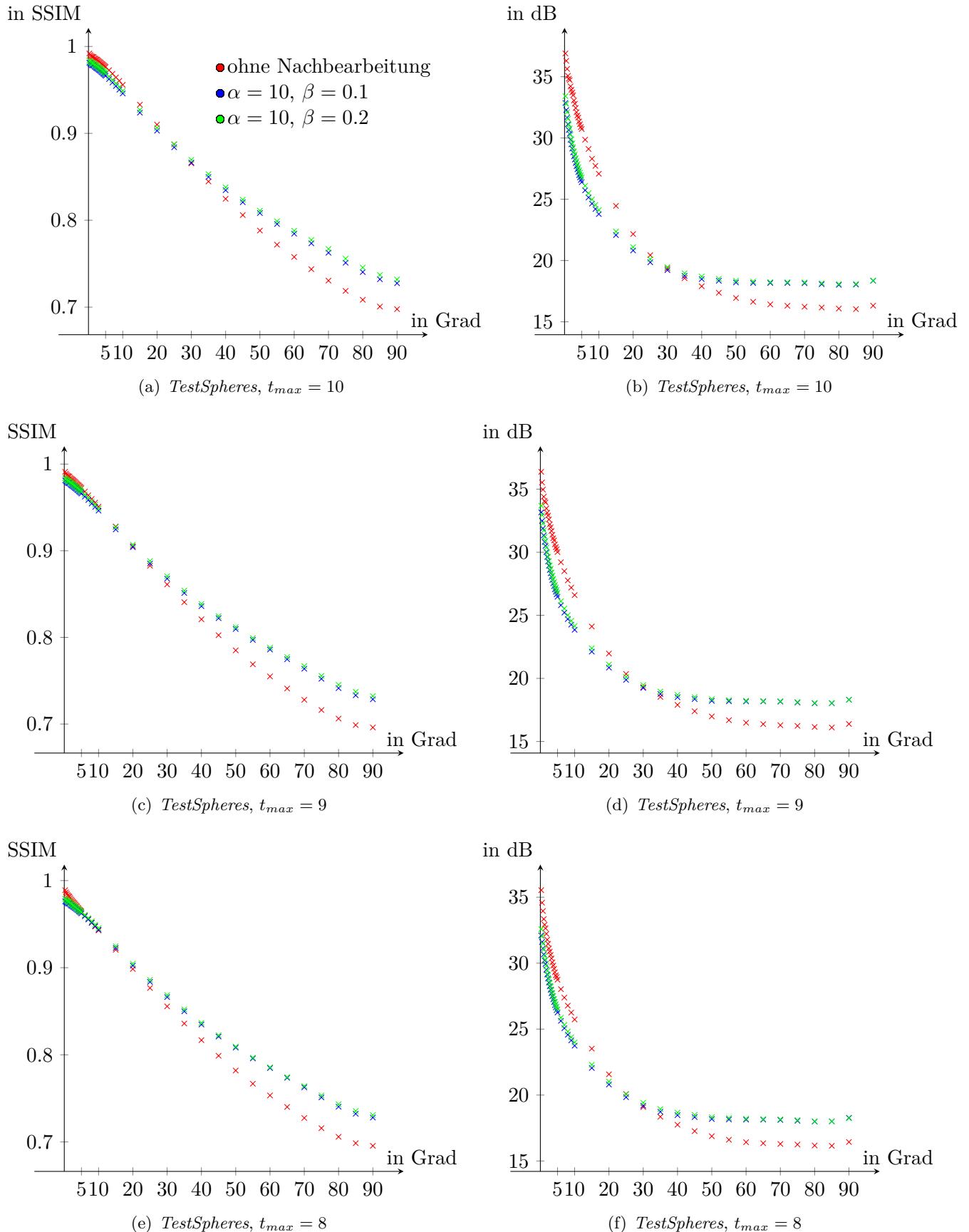


Abbildung 6.9: Es wird der Datensatz *TestSpheres* betrachtet. Die Diagramme zeigen die Güteentwicklung in Abhängigkeit des Parameters j . Dabei liegt die maximale Baumtiefe in a,b bei 10, in c,d bei 9 und in e,f bei 8.

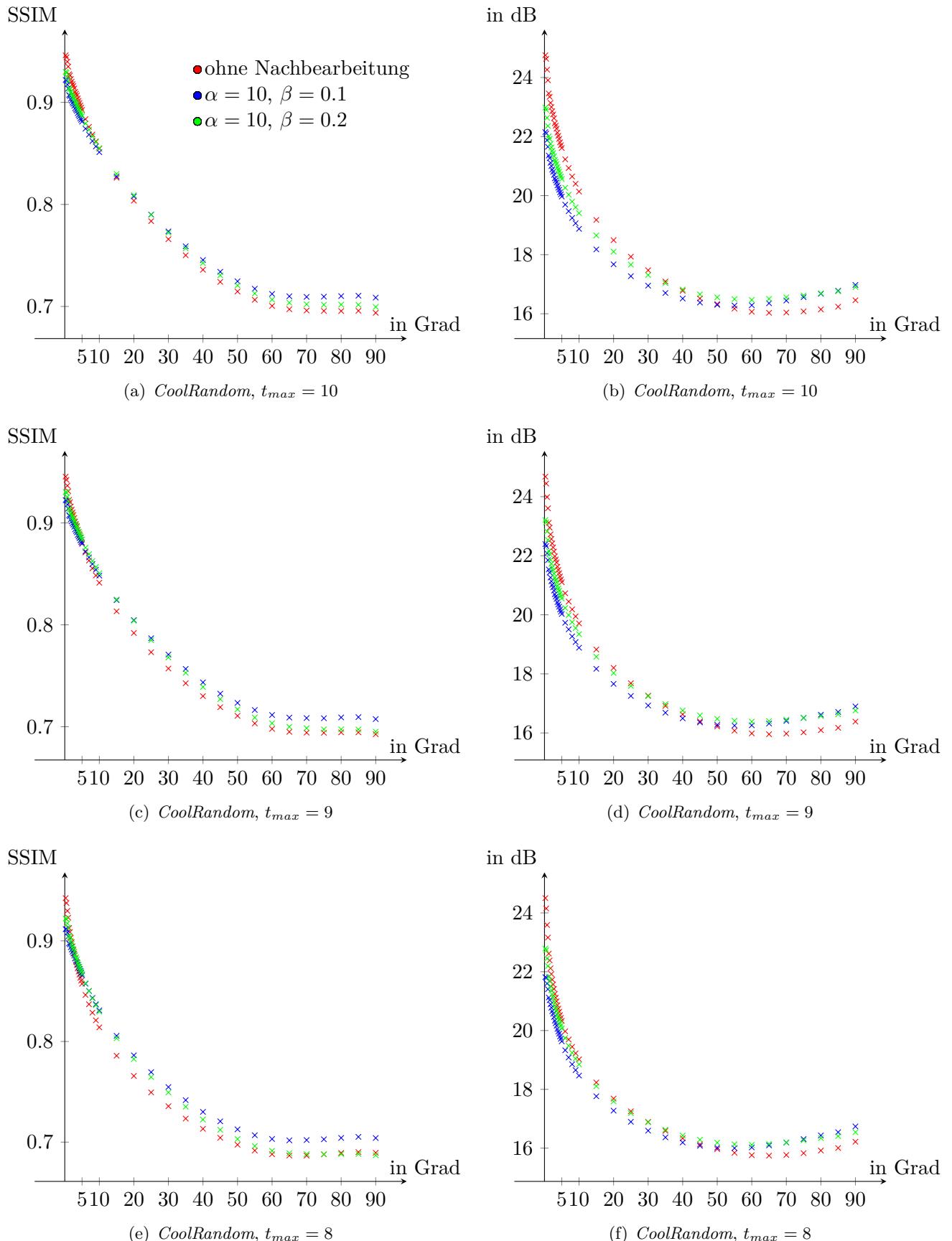


Abbildung 6.10: Es wird der Datensatz *CoolRandom* betrachtet. Die Diagramme zeigen die Güteentwicklung in Abhängigkeit des Parameters j . Dabei liegt die maximale Baumtiefe in a,b bei 10, in c,d bei 9 und in e,f bei 8.

Die Diagramme aus der Abbildungen 6.9 und 6.10 stellen dar, wie sich der Netzoptimierungsschritt auf die Dreiecksnetze hinsichtlich der Güte auswirkt. Zum Vergleich dient das nicht nachbearbeitete Netz. Dabei wird $\alpha = 10$ gesetzt und $\beta \in \{0.1, 0.2\}$ gewählt.

Deutlich zu sehen ist, dass bei kleinen Winkeln die nachbearbeiteten Netze schlechter abschneiden als die nicht optimierten. In allen Fällen existiert ein Kippunkt, ab dem die optimierten Netze hinsichtlich der Güte bessere Ergebnisse liefern. Bei kleineren t_{max} wird dieser Kippunkt schneller erreicht. Beim PSNR zeigt sich dieser Kippunkt erst bei größeren Winkeln. Außerdem steigt beim PSNR die Güte oberhalb von 60 Grad wieder an.

Die Parameter des Netzoptimierungsschrittes werden ähnlich ausgewertet wie zuvor l und i . Zu diesen Zweck werden $l = 0.7$ und $i = 0.6$ gesetzt. Getestet werden die Parameter $\alpha \in \{0.1, 1, 10, 100, 1000\}$ und $\beta \in \{0.1, 0.2, \dots, 1.0\}$.

In der Abbildung 6.11 wird der Datensatz *TestSpheres* betrachtet. Es zeigt sich für $t_{max} = 10$, dass die Güte ab $\alpha > 1$ größer als 0.9 ist. Im Vergleich dazu ist eine Verbesserung der Güte bei $t_{max} < 10$ auch bei $\alpha < 10$ zu erkennen. Es lässt sich feststellen, dass für alle t_{max} die besten Konstruktionszeiten für $\beta < 0.3$ erreicht werden.

In der Abbildung 6.12 sind die Parameter für den Datensatz *CoolRandom* abgebildet. Hinsichtlich der Güte lässt sich kein Unterschied im Bezug auf α erkennen. Die besten Ergebnisse werden für $\beta < 0.3$ erreicht. Mit steigenden t_{max} steigt die Güte. Die Konstruktionszeiten sind bei $t_{max} > 0.8$ für alle Parameterkonfigurationen größer als 120 ms. Für $t_{max} = 8$ liegen alle unterhalb von 70 ms.

In der Abbildung 6.13 wird die Anzahl der Dreiecke in Abhängigkeit von α, β und t_{max} dargestellt. Zu erkennen ist für die Szene *TestSpheres*, dass die Anzahl der Dreiecke für die Konfigurationen $\alpha > 1$ am kleinsten ist. In der Szene *CoolRandom* zeigt sich diese Tendenz nicht ganz so stark. Weiterhin fällt auf, dass bei kleinen Werten für *beta* weniger Dreiecke entstehen als bei großen Werten. Dieser Zusammenhang zeigt sich für beide Szenen.

Die Abbildung 6.14 zeigt die Güte von nicht nachbearbeiteten Netzen im Vergleich zu einem optimierten Netz. Dabei wurden die Parameter, wie zuvor $l = 0.7$ und $i = 0.6$, für alle Netze gewählt. Es ist zu sehen, dass in der Szene *TestSpheres* ein Kippunkt existiert, bei dem das optimierte Netz besser als die anderen Netze abschneidet. Ein solcher Kippunkt existiert für *CoolRandom* nicht. Am Beispiel der *TestSpheres*-Szene wird die Anzahl der erzeugten Dreiecke verglichen. Für $i_{max} = 9$ besteht das nicht optimierte Netz aus 7320 Dreiecken. Wenn $i_{max} = 8$ ist, besteht das nicht nachbearbeitete Netz aus 3003 Dreiecken und das optimierte Netz aus 4279 Dreiecken.

Die Abbildung 6.15 zeigt repräsentativ die Zeit, die der Client für die Extrapolation benötigt. Diese liegen sowohl für die Vollvernetzung als auch für die Delaunay-Triangulierung im Schnitt unter einer Millisekunde. Es gibt ein paar Ausreißer, die eine Zeit von 1.5 ms nicht überschreiten.

Im Folgenden werden die vorgestellten Ergebnisse diskutiert.

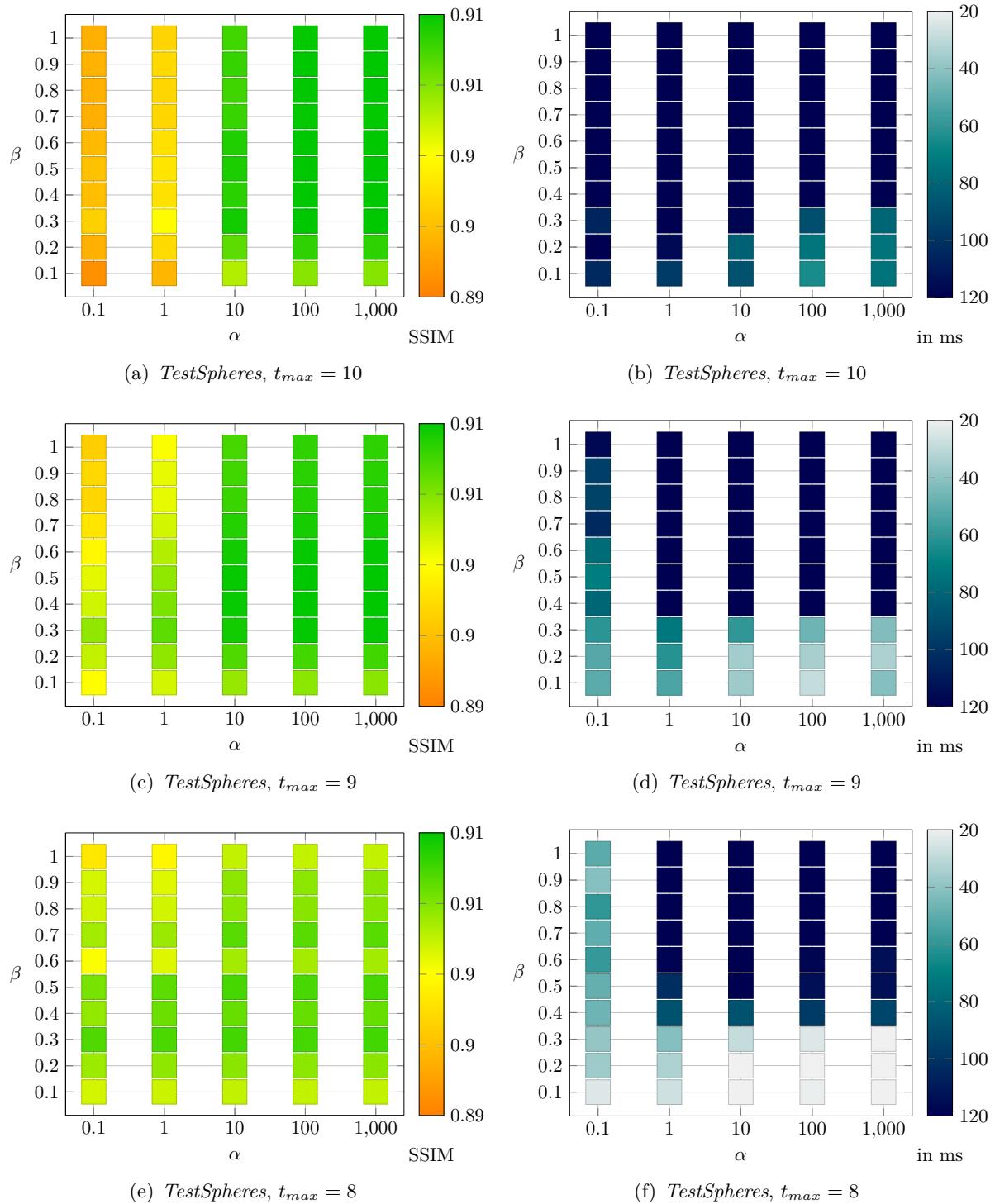


Abbildung 6.11:

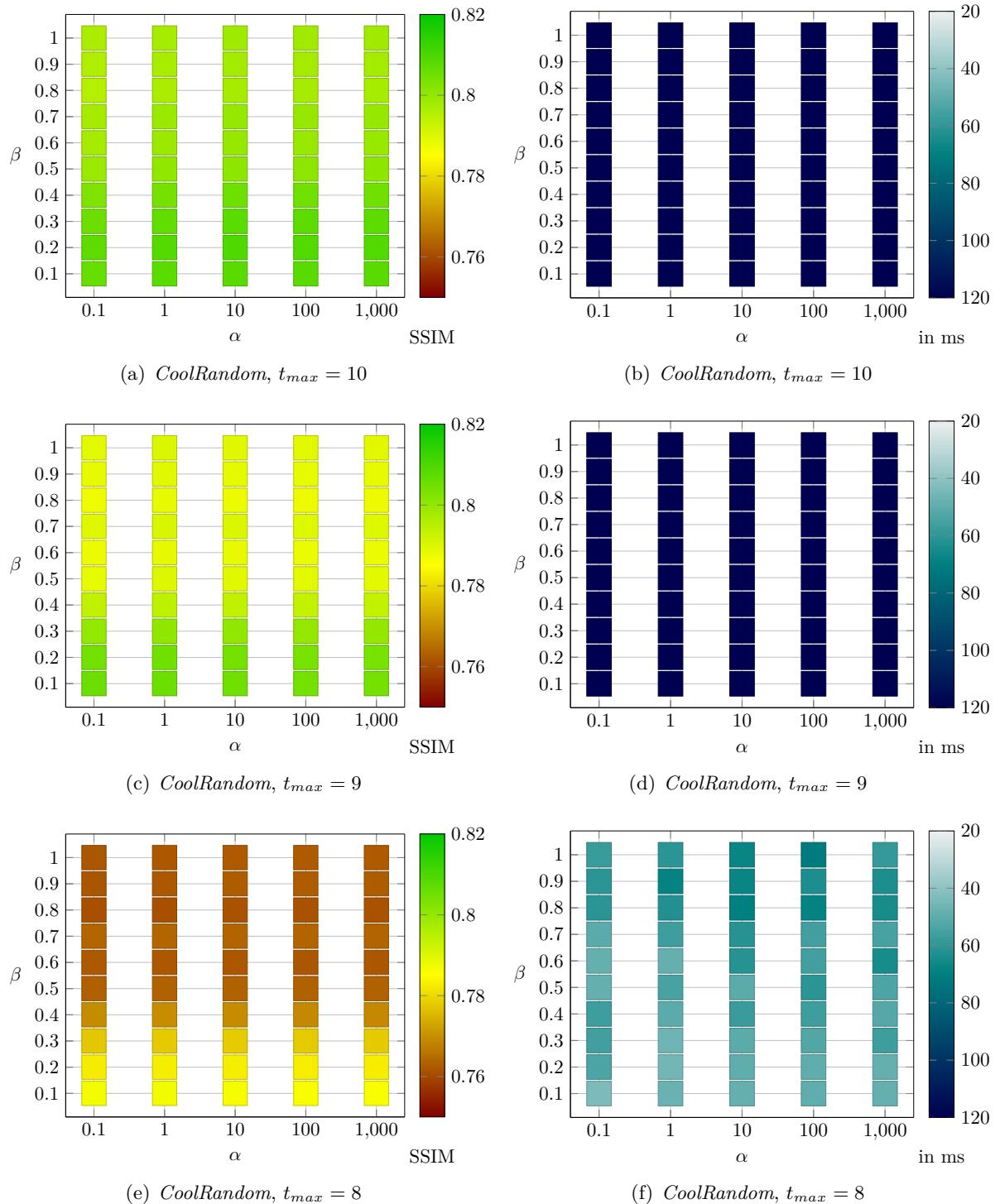


Abbildung 6.12: SSIM und Netzkonstruktionszeit bei einem konstanten Winkelunterschied von 20 Grad.

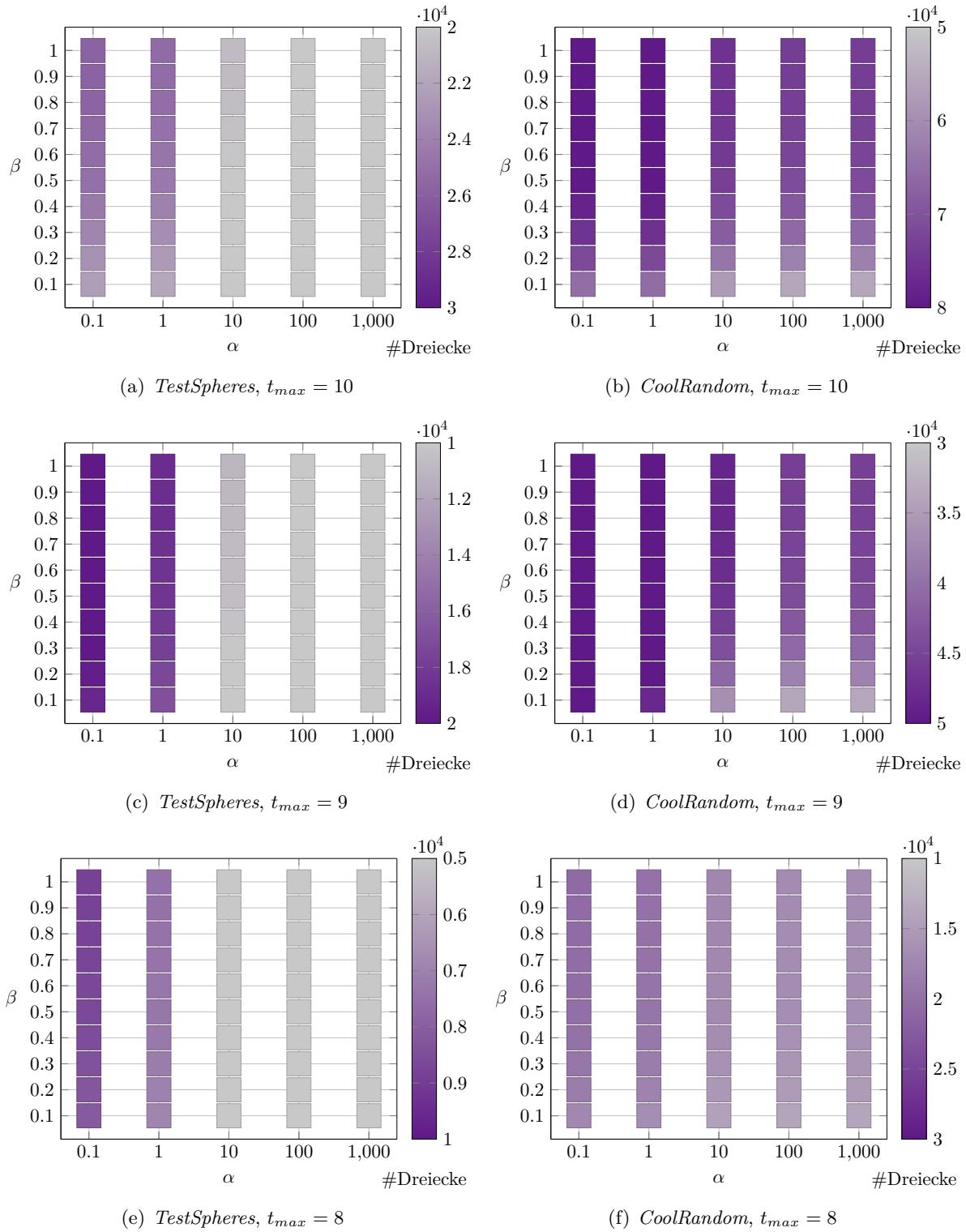


Abbildung 6.13:

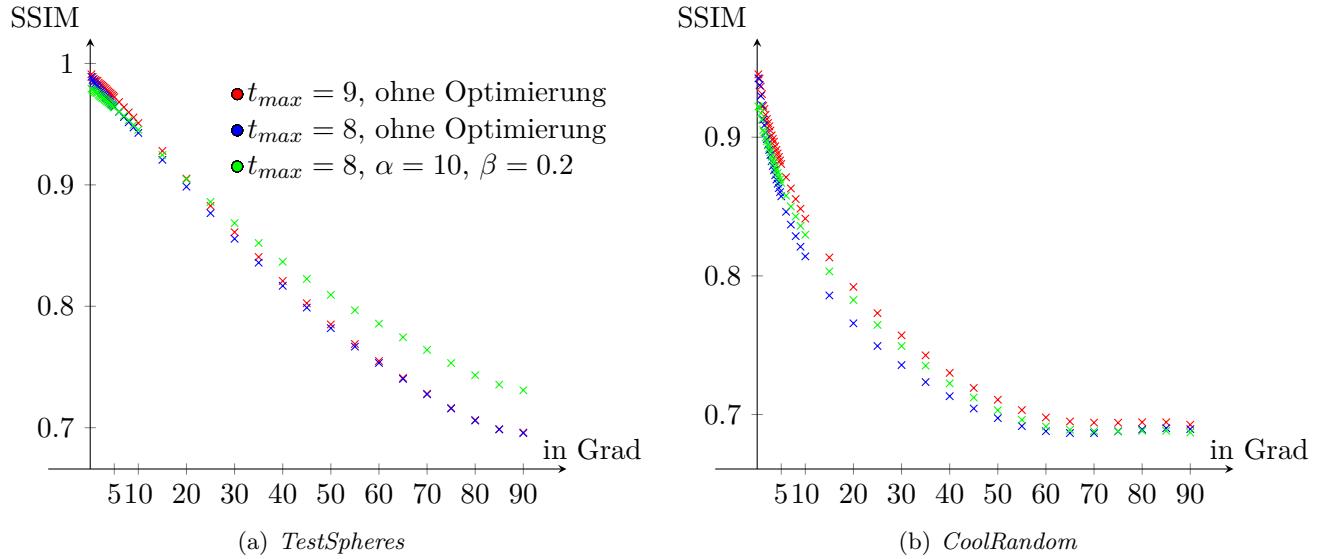


Abbildung 6.14: Vergleich zwischen nicht optimierten und einem optimiertem Netz.

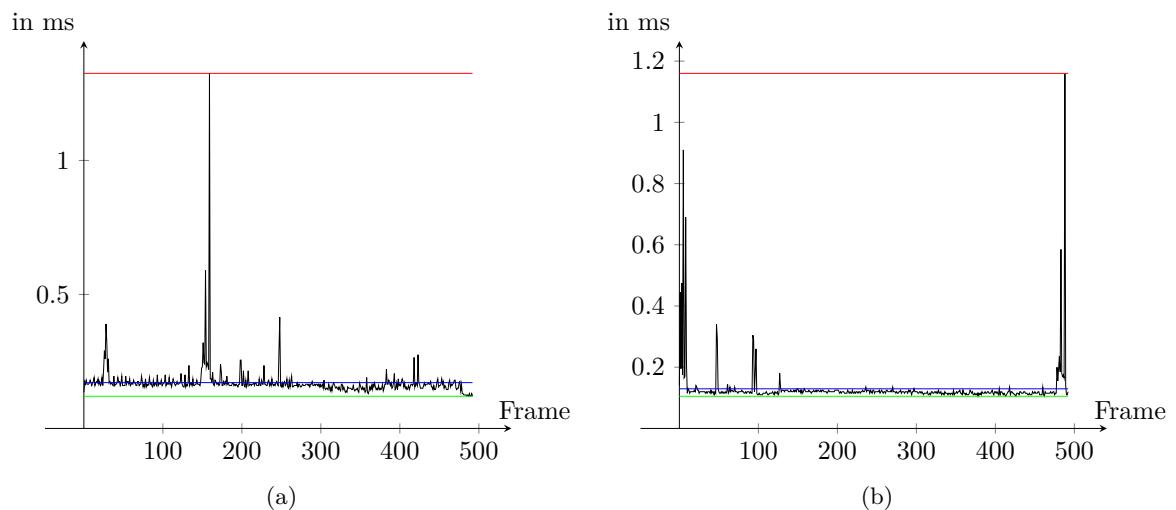


Abbildung 6.15: Die Diagramme stellen die Zeit, die der Client, zur Extrapolation der Frames benötigt grafisch dar. Dabei ist im Diagramm a die Vollvernetzung zu sehen und im Diagramm b die Delaunay-Triangulation.

7 Diskussion

In diesem Kapitel werden die zuvor präsentierten Ergebnisse diskutiert. Dabei werden in dem Vollvernetzungsansatz die Tiefeninformationen aus dem Tiefenbild unverändert und unkompri-miert für die Bildextrapolation genutzt. Sie liefert, wie es zu erwarten war, die besten Ergebnisse hinsichtlich der Qualität für beide Szenen. Zu erwähnen ist, dass die Ergebnisse in Abhängigkeit der gewählten Szene stark variieren. Dies hängt damit zusammen, dass die Szenen unterschiedlich beschaffen sind: Die Szene *TestSpheres* besteht aus wenigen großen Kugeln mit großen Oberflächen und wenigen Kanten, wohingegen die Szene *CoolRandom* aus vielen kleinen Kugeln mit kleinen Oberflächen und vielen Kanten besteht. Das führt dazu, dass die Dreiecksnetze der Szene *CoolRandom* aus vielmehr Dreiecken bestehen als in die der *TestSpheres*-Szene. Aus diesem Grund ist die Erzeugung der Dreiecksnetze für die *CoolRandom* Szene mit weit aus mehr Rechenaufwand verbunden. Zusätzlich steigt der Fehler hinsichtlich der Qualität deutlich schneller als in der Szene *TestSpheres*. Das resultiert daraus, dass in dem *CoolRandom*-Datensatz weitaus mehr Kugeln existieren als in der *TestSpheres*-Szene, die im Tiefenbild des ersten Frames nicht abgebildet werden, aus dem die Netze erzeugt werden und deshalb in den extrapolierten Bildern nicht existieren.

Insgesamt liefert der *Quadtree* weitaus bessere Resultate als die FloydSteinberg-Methode. Das Problem der FloydSteinberg-Methode besteht darin, das extrem viele Punkte vorwiegend in der Nähe von Kanten platziert werden. Dabei gibt es einen Unterschied zwischen äußeren und inneren Kanten. Während die äußeren Kanten die Kontur des Objektes abbilden und sehr dicht besetzt sind, werden innere Kanten nicht ausreichend repräsentiert. Planare Flächen bestehen ebenso aus zu vielen Dreiecken. Das führt dazu, dass auch bei einer sehr hohen Zahl von Dreiecken die Qualität der extrapolierten Bilder schon bei geringen Kamerawinkelunterschieden um einiges schlechter ausfällt als bei den anderen Methoden. Weil insgesamt zu viele Punkte erzeugt werden, wird auch die Vernetzung langsam. Die Abbildungen 7.3 und 7.6 verdeutlichen diesen Zusammenhang.

Im Vergleich dazu schneidet der *Quadtree*-Ansatz im Bezug auf Qualität, Rechenaufwand weitaus besser ab. Weiterhin lässt sich die Qualität des resultierenden Dreiecknetzes mit dem Parameter t_{max} , adaptiv anpassen. Große Werte für t_{max} bedeuten in erster Linie, dass die Bereiche stärker unterteilt werden, so dass an markanten Stellen, wie Kanten viele Punkte für die Vernetzung zur Verfügung stehen. Das hat aber einen direkten Einfluss auf die Anzahl der Dreiecke und die Konstruktionsgeschwindigkeit. Bei kleinen Werten für t_{max} , werden aufgrund der großen *Quadtree*-Regionen, wie in der Abbildung 7.1 zu sehen ist, ganze Bereiche nicht vom Dreiecksnetsnetz repräsentiert. Außerdem werden bei der Extrapolation einzelne Regionen deutlich sichtbar. Die Variablen l und i beeinflussen die Güte kaum, haben jedoch einen starken Einfluss auf die Anzahl der Dreiecke. Nehmen l und i große Werte an, dann wird die Anzahl der Dreiecke reduziert. Die Güte wird durch l und i deshalb kaum beeinflusst, weil sowohl äußere als auch innere Kanten auch für große Werte von l ausreichend abgebildet werden. Der Parameter i bestimmt vorwiegend die Unterteilung gekrümmter Flächen; bei kleinen Winkeln wird die Güte selbst bei einer größeren Unterteilung kaum verschlechtert. Es sind große Werte für l zu empfehlen, beispielsweise 0.7. Der Parameter i sollte kleiner oder gleich l sein.

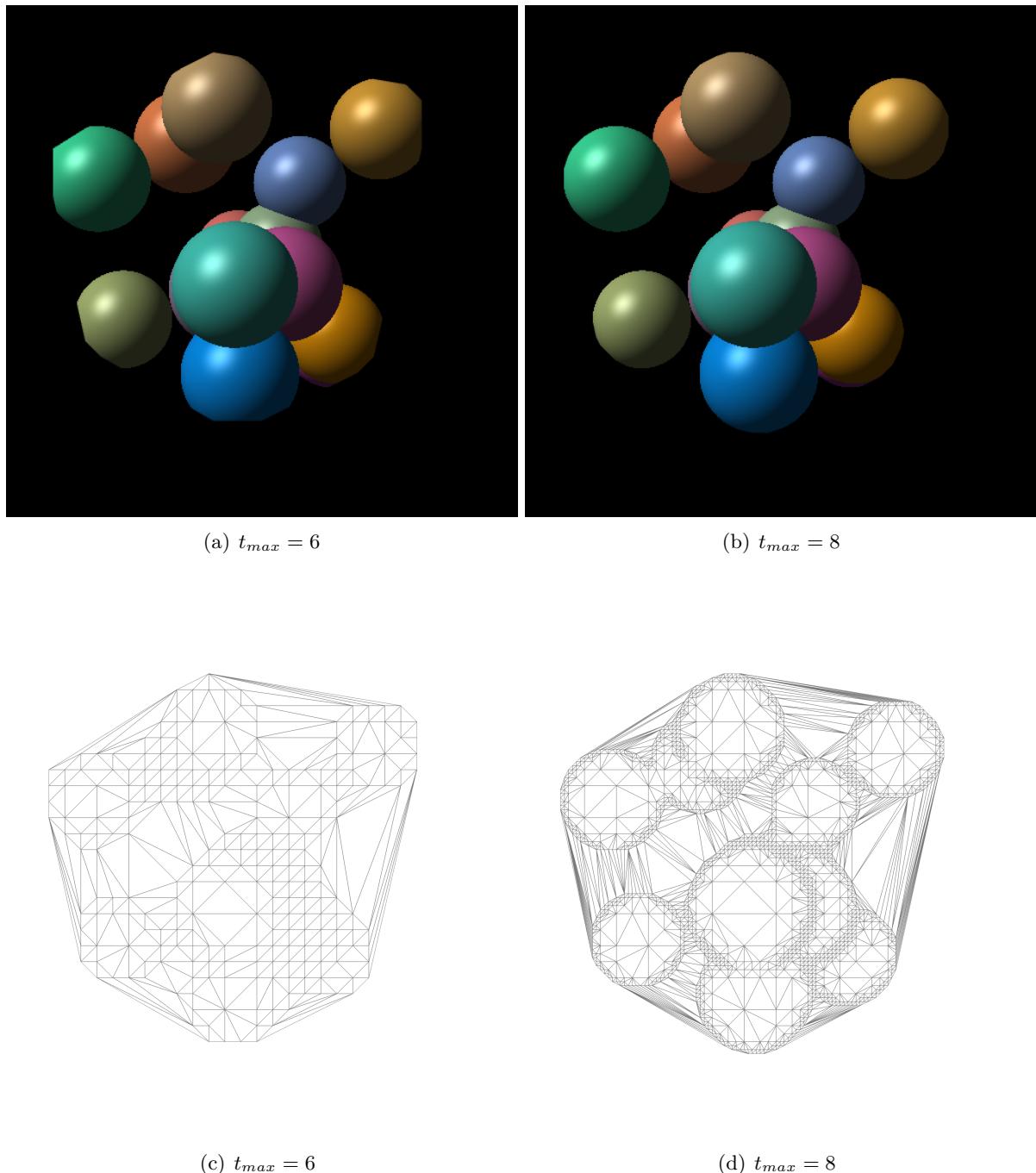


Abbildung 7.1:

Die Nachbearbeitung der Netze bewirkt grundsätzlich einen Gütezuwachs zulasten einer erhöhten Dreiecksanzahl. Gleichzeitig entstehen Artefakte an den Rändern der Kugeln, wie in der Abbildung 7.4 beispielhaft zu erkennen ist. Diese Artefakte in Form von Schnittkanten sorgen dafür, dass die Güte bei kleinen Kamerawinkelunterschieden im Vergleich zu nicht nachbearbeiteten Netzen geringer ausfällt. Wenn α Werte annimmt, die größer gleich als 10 sind, dann hat die Formel 4.5 keinen Einfluss auf das Ergebnisnetz. Nimmt α Werte kleiner 10 an, dann werden Dreiecke innerhalb der Kugeln zusätzlich unterteilt, was die Anzahl der Dreiecke insgesamt erhöht. Allerdings gibt es in diesen Bereich bereits ausreichend viele Dreiecke, um die Krümmung der Flächen zu repräsentieren, sodass deren Unterteilung keinen messbaren Güteunterschied bewirkt. Dieser Effekt wird in der Szene *CoolRandom* besonders deutlich.

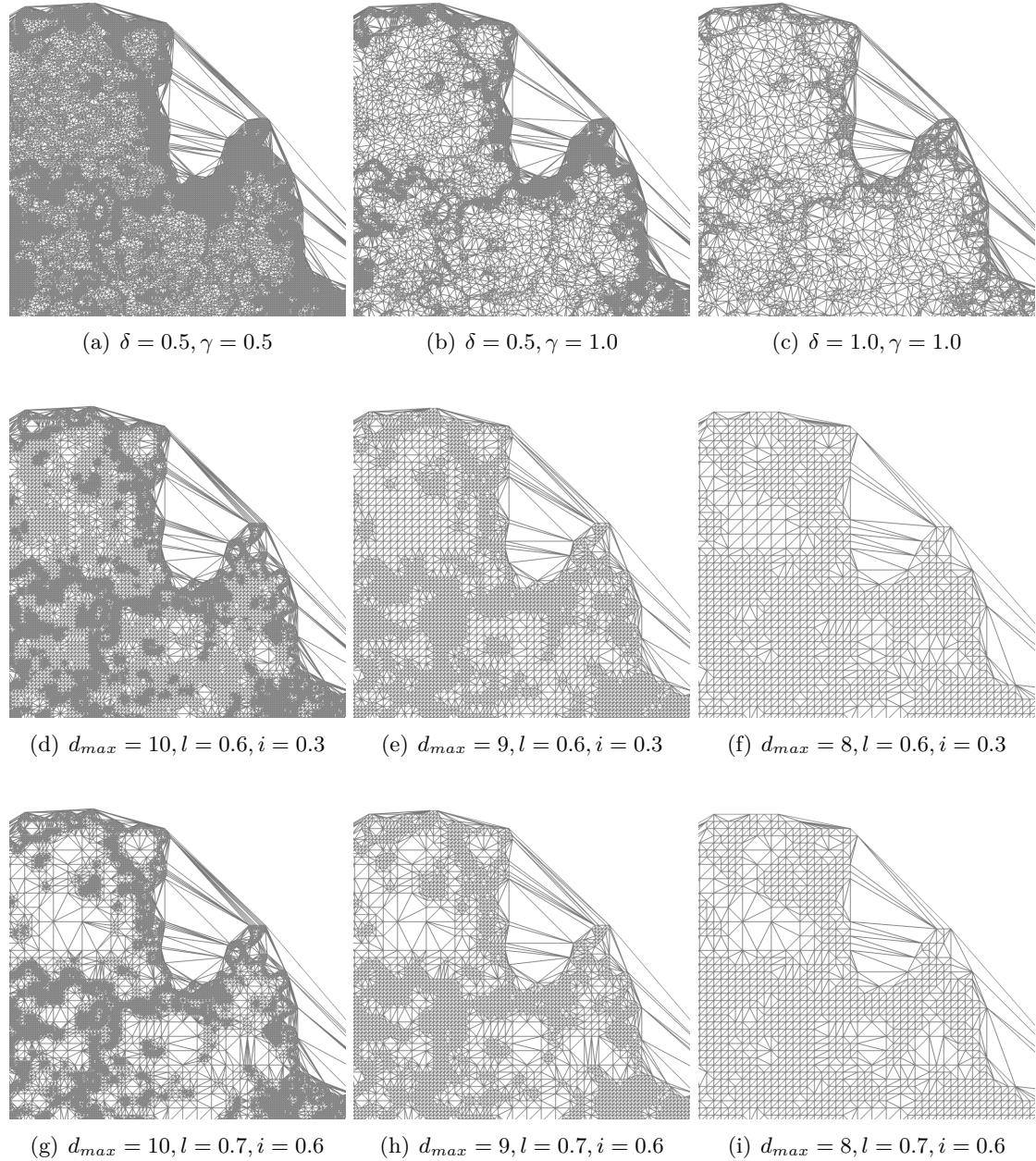


Abbildung 7.2: Die Dreiecksnetze a-c wurden mit dem FloydSteinberg-Ansatz erzeugt und die Dreicksnetze d-i mit dem *Quadtree*-Ansatz.

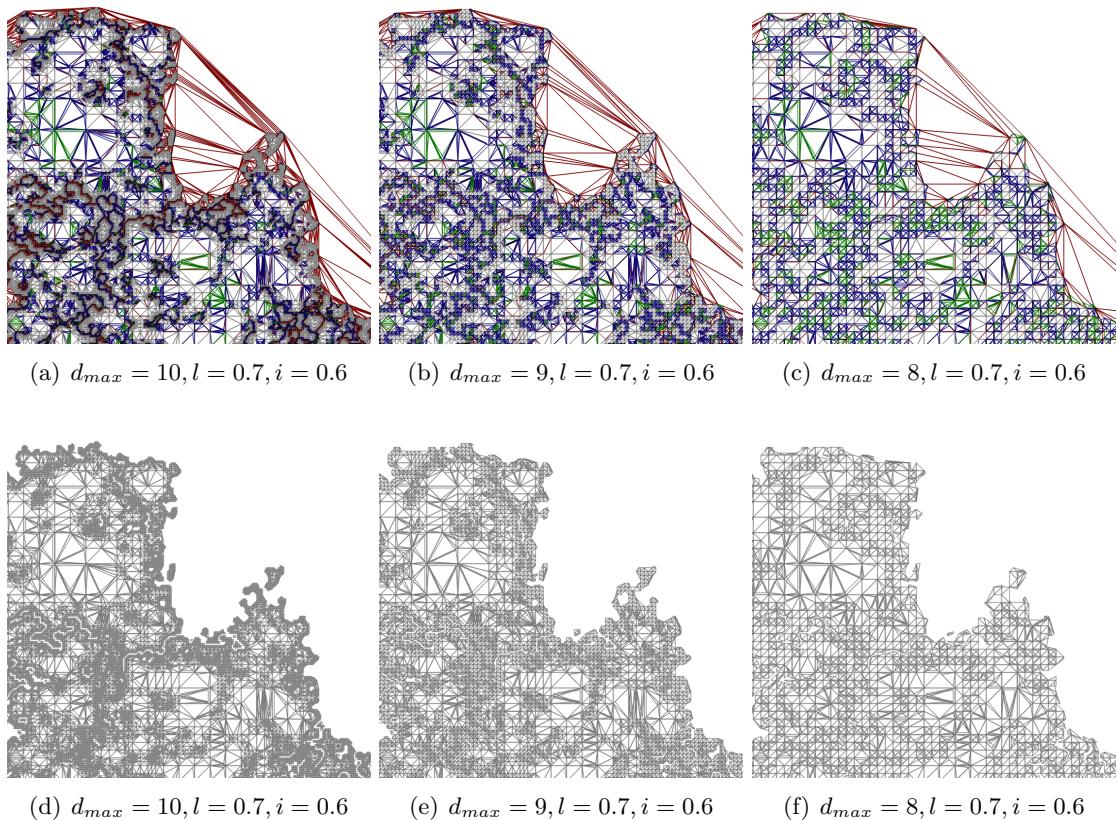


Abbildung 7.3: Vergleich von nachbearbeiteten Dreiecksnetzen, die mit dem *Quadtree*-Ansatz erzeugt wurden. Dabei wurde $\alpha = 10$ und $\beta = 0.2$ gewählt.

Da die Nachbearbeitung für alle Dreiecke unabhängig voneinander berechnet werden kann, lässt sich dieser Schritt gut parallelisieren. Auf diese Weise kann der Nachbearbeitungsschritt um ein Vielfaches beschleunigt werden. Diese Parallelisierung wurde in dieser Arbeit jedoch nicht implementiert.

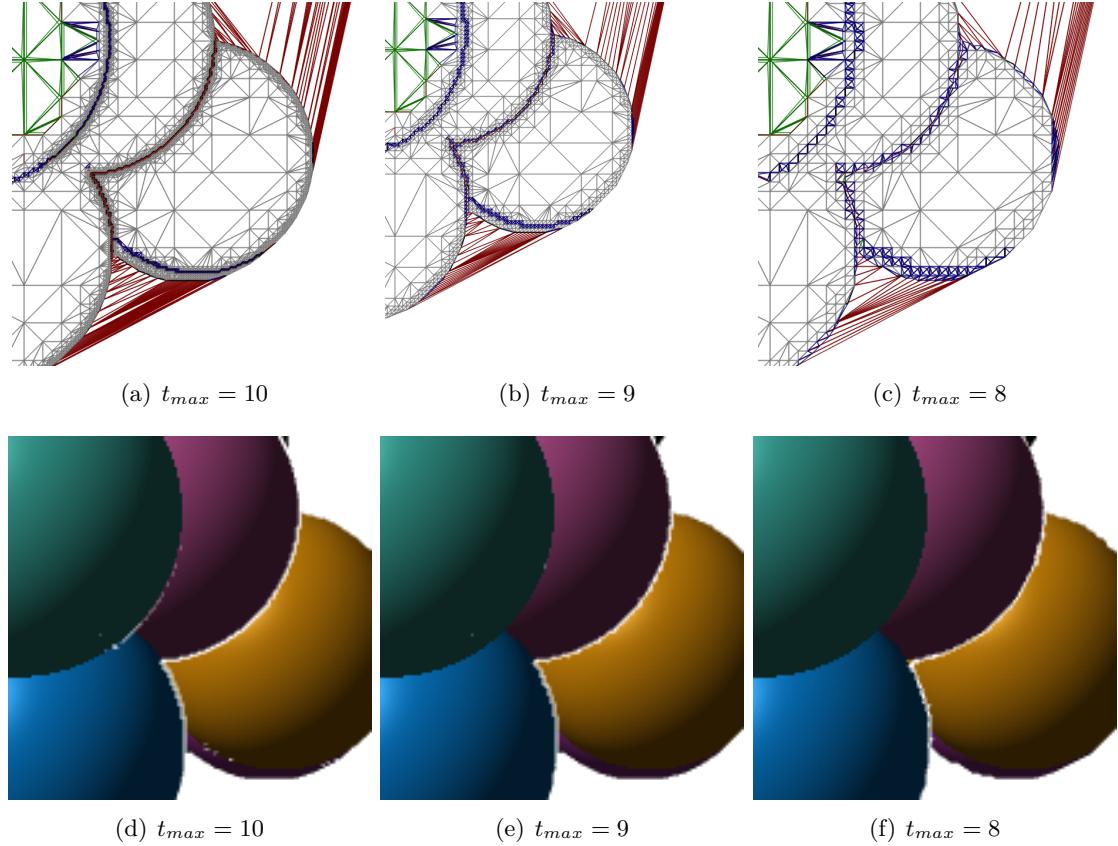


Abbildung 7.4: Beispiele für Schnittkantenartefakte in der Szene *TestSpheres*, des ersten Frames.
Das Netz wurde mit $l = 0.7, i = 0.6, \alpha = 100, \beta = 0.2$ erzeugt.

Aufgrund der Messergebnisse vom SSIM und dem PSNR lässt sich sagen, dass das Verfahren der Extrapolation durch die Rückprojektion durchaus geeignet ist, um Bilder für kleine Kamerawinkelunterschiede von 5 bis 10 Grad ausreichend gut zu approximieren. Dabei sind die Ergebnisse gemessen durch SSIM weitaus besser als die des PSNR, so dass hier eine gute Approximation bis zu 20 Grad Winkelunterschied möglich ist. Dies ist positiv zu bewerten, da der SSIM im Gegensatz zum PSNR die menschliche Wahrnehmung mit einbezieht.

Ein grundsätzliches Problem der Extrapolation mit Hilfe der Rückprojektion ist die Beleuchtung. Insbesondere spekulare Lichtanteile sind ein Problem, weil diese von der Position der Kamera abhängen. Da das Farbbild als Textur dient und keine zusätzliche Beleuchtungsberechnung stattfindet, entsteht bei größeren Kamerawinkelunterschieden zwangsläufig ein Fehler, wie in Abbildung 7.5. Dieser steigt mit der Größe des Kamerawinkelunterschiedes und taucht in den getesteten Datensätzen nur bei sehr großen Winkeln auf und ist im Verhältnis zu den anderen entstehenden Artefakten jedoch zu vernachlässigen. Dieser könnte dadurch reduziert werden, dass Informationen zu Lichtquellen mit übertragen werden und eine clientseitige Nachbeleuchtung durchgeführt wird. Das führt auf Seiten des Clients zu erhöhten Berechnungskosten. Ein anderer Ausweg könnte eine von der Kameraperspektive abhängigen Texturierung des Dreiecksnetzes sein. Zu diesem Zweck müssen aber zusätzliche Texturinformationen vom Server zum Client übermittelt werden.

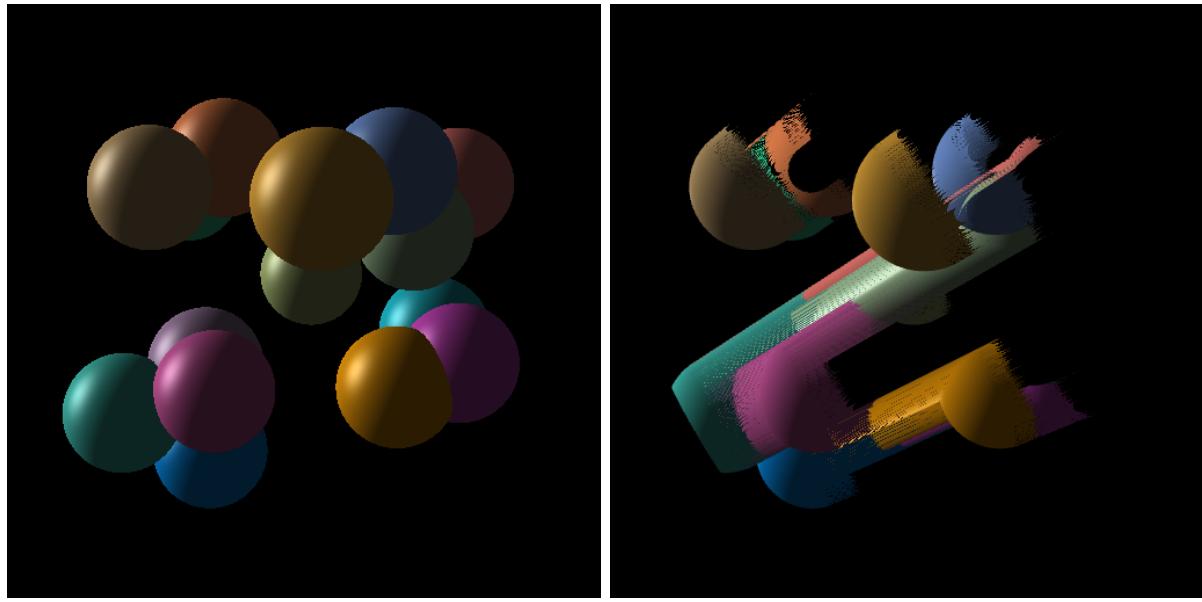
(a) $t_{max} = 6$ (b) $t_{max} = 8$

Abbildung 7.5: title

Desweiteren sind transparente Oberflächen problematisch. Objekte, die hinter diesen Oberflächen liegen, werden im Tiefenbild nicht repräsentiert. Es entsteht auch hier ein Fehler bei der Extrapolation.

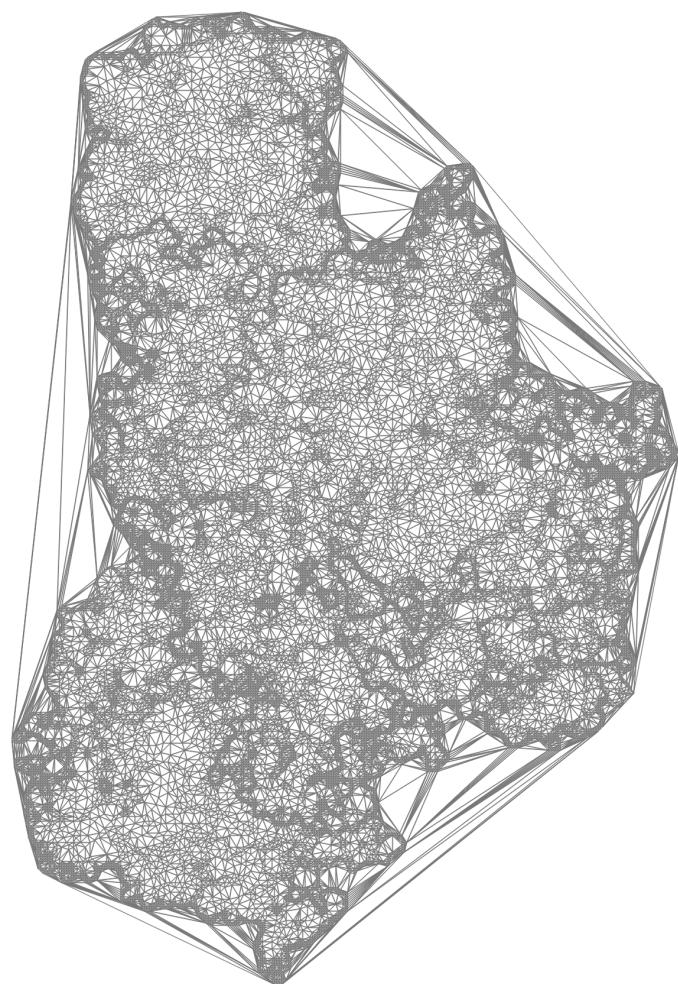


Abbildung 7.6: FloydSteinberg-Ansatz mit den Parametern $\delta = 0.8, \gamma = 0.8$.

8 Zusammenfassung und Ausblick

In Rahmen dieser Arbeit wurde ein Remote-Visualisierungssystem konstruiert, in dem Server zusätzlich zum Farbbild ein adaptives Dreiecksnetz aus dem dazugehörigen Tiefenbild erzeugt. Das Farbbild und das Dreiecksnetz wird anschließend zum Client gesendet, der eine Extrapolation auf Basis der Rückprojektion durchführt. Mit Hilfe von zwei *Ground-Truth*-Datensätzen wurde untersucht, wie sich die Güte in Abhängigkeit des Kamerawinkelunterschiedes verhält. Dabei haben die Dreiecksnetze, die Aufgabe die Grauwertverläufe der Tiefenbilder zu approximieren. Konstruiert werden die Dreiecksnetze mit Hilfe der Delaunay-Triangulierung. Die für die Vernetzung notwendige Menge von ausgewählten Punkten, wird anhand von Gradientenbildern bestimmt. Zu diesem Zweck wurde ein Fehlerdiffusionsverfahren, das auf dem FloydSteinberg-Algorithmus basiert und ein *Quadtree*-Ansatz implementiert. Beide wurden hinsichtlich der Konstruktionszeiten, der Anzahl von Dreiecken und der erzielten Güte untersucht. Dabei hat sich gezeigt, dass der *Quadtree*-Ansatz hinsichtlich aller Kriterien bessere Ergebnisse erzielt. Weiterhin wurde ein Nachbearbeitungsschritt implementiert, mit dem die Qualität der Netze hinsichtlich der Güte insbesondere bei großen Winkeln deutlich verbessert wird, allerdings zu lasten der Dreiecksanzahl und der Konstruktionszeit. Dabei treten jedoch bei sehr kleinen Kamerawinkelunterschieden Artefakte in der Nähe von großen Tiefensprüngen im Tiefenbild auf.

Die Extrapolation, auf Basis der Rückprojektion von Dreiecksnetzen, ist durchaus ein geeignetes Verfahren um neue Ansichten einer Szene durch den Client zu approximieren, vorausgesetzt, der Kamerawinkelunterschied wird nicht zu groß. Die Dreiecksnetze, erzeugt mit dem *Quadtree*, lassen sich mit beliebiger Genauigkeit erzeugen und sind eine gute Grundlage um diese Verlustbehaftet komprimieren zu können.

Der Nachbearbeitungsschritt führt zu einer deutlichen Verbesserung hinsichtlich der Güte, lässt sich sicherlich noch Verbessern. Insbesondere, im Bezug auf die Anzahl der erzeugten Dreiecke. Im nächsten Schritt, könnten die Dreiecksnetze durch Valenz-basierte Kodierung weiter komprimiert werden. Weiterhin sollte eine Binäre-Datenbeschreibungssprache gewählt werden um die Größe der Nachrichtenpakete zu verkleinern. Durch Verdeckung bedingte Fehler könnten mit Hilfe des Kameramodells von Popescu und Aliaga [PA06] weiter reduziert werden. Desweiteren sollte untersucht werden, inwieweit die extrapolierten Bilder genutzt werden können, um Informationen bei der Übertragung des Farbbildes mit dem Differenzbildansatz zu reduzieren. Progressive Übertragung der Dreiecksnetze könnte ebenso interessanter Ansatz für die Extrapolation sein. Außerdem kann der Client in Abhängigkeit seiner Ressourcen ein eigenes Modell der Szene, aus den übertragenen Dreiecksnetzen konstruieren.

Literaturverzeichnis

- [Aur91] AURENHAMMER, Franz: Voronoi diagrams – a survey of a fundamental geometric data structure. In: *ACM COMPUTING SURVEYS* 23 (1991), Nr. 3, S. 345–405
- [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 ; GASPERELLO, 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
- [DYB98] DEBEVEC, Paul ; YU, Yizhou ; BORSHUKOV, George: *Efficient View-Dependent Image-Based Rendering with Projective Texture-Mapping*. Vienna : Springer Vienna, 1998, S. 105–116. – ISBN 978-3-7091-6453-2
- [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
- [Gir93] GIROD, Bernd: Digital Images and Human Vision. Cambridge, MA, USA : MIT Press, 1993. – ISBN 0-262-23171-9, Kapitel What's Wrong with Mean-squared Error?, S. 207–220
- [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
- [HEB⁺01] HUMPHREYS, Greg ; ELDRIDGE, Matthew ; BUCK, Ian ; STOLL, Gordan ; EVERETT, Matthew ; HANRAHAN, Pat: WireGL: A Scalable Graphics System for Clusters. In:

- Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques.* New York, NY, USA : ACM, 2001 (SIGGRAPH '01). – ISBN 1–58113–374–X, S. 129–140
- [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
- [LKR⁺96] LINDSTROM, Peter ; KOLLER, David ; RIBARSKY, William ; HODGES, Larry F. ; FAUST, Nick ; TURNER, Gregory A.: Real-time, Continuous Level of Detail Rendering of Height Fields. In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA : ACM, 1996 (SIGGRAPH '96). – ISBN 0–89791–746–4, S. 109–118
- [LYW00] LEE, J. ; YANG, Y. ; WERNICK, M. N.: A new approach for image-content adaptive mesh generation. In: *Image Processing, 2000. Proceedings. 2000 International Conference on* Bd. 1, 2000. – ISSN 1522–4880, S. 256–259 vol.1
- [Mar99] MARK, William R. *Post-Rendering 3D Image Warping: Visibility, Reconstruction, and Performance for Depth-Image Warping*. 1999
- [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
- [MMB97] MARK, William R. ; McMILLAN, Leonard ; BISHOP, Gary: Post-rendering 3D Warping. In: *Proceedings of the 1997 Symposium on Interactive 3D Graphics*. New York, NY, USA : ACM, 1997 (I3D '97). – ISBN 0–89791–884–3, S. 7–ff.
- [MPS05] MEI, Chunhui ; POPESCU, Voicu ; SACKS, Elisha: The Occlusion Camera. In: *Computer Graphics Forum* (2005). – ISSN 1467–8659
- [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.
- [PA06] POPESCU, Voicu ; ALIAGA, Daniel: The Depth Discontinuity Occlusion Camera. In: *Proceedings of the 2006 Symposium on Interactive 3D Graphics and Games*. New York, NY, USA : ACM, 2006 (I3D '06). – ISBN 1–59593–295–X, S. 139–143
- [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
- [TG98] TOUMA, Costa ; GOTSMAN, Craig: Triangle Mesh Compression. In: *Proceedings of the Graphics Interface 1998 Conference, June 18-20, 1998, Vancouver, BC, Canada*, 1998, S. 26–34
- [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
- [TR09] THUNG, K. H. ; RAVEENDRAN, P.: A survey of image quality measures. In: *2009 International Conference for Technical Postgraduates (TECHPOS)*, 2009, S. 1–4
- [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
- [Wik14] WIKIPEDIA: *Floyd-Steinberg-Algorithmus* — Wikipedia, Die freie Enzyklopädie. 2014. – [Online; Stand 23. November 2016]
- [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

1.1	Die Abbildungen zeigen die Extrapolation des ersten Bildes a aus dem Datensatz <i>TestSpheres</i> . Der Winkelunterschied zum Bild b beträgt 8 Grad.	4
3.1	Farb- und Tiefenbilder der ersten Frames von beiden Datensätzen.	8
3.2	Die Abbildung zeigt die Teilschritte der Transformation eines Vertices aus den Modellkoordinaten in Bildschirmkoordinaten.	9
3.3	Das Modell zeigt das Vorgehen von dem SSIM-Algorithmus [WBSS04].	12
4.1	Darstellung von drei verschiedenen Varianten der Vollvernetzung.	15
4.2	Gradientenbilder beider Datensätze vom jeweils ersten Tiefenbild.	17
4.3	Szene <i>TestSpheres</i> , Baumtiefe 9, l=5, i=4	19
4.4	Darstellung der gewichteten Verteilung des Quantisierungsfehlers [Wik14].	19
4.5	Die Bilder a bis d zeigen das Merkmalsbild in Abhängigkeit des Parameters γ bei einem festgesetzten Schwellwert von 0.5.	21
4.6	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.	23
4.7	Die Bilder a und b zeigen die Unterteilung des Dreiecks $p_1p_2p_3$, wenn keine Kante valide ist.	23
4.8	Auswirkung der Parameter auf die Nachbearbeitung. Das Netz wurde mit der <i>Quadtree</i> -Methode erzeugt. Dazu wurde $t_{max} = 8, l = 0.7, i = 0.6$ gewählt.	24
4.9	Nachbearbeitetes Netz des ersten Frames aus dem <i>TestSpheres</i> -Datensatz. Das Netz wurde mit der <i>Quadtree</i> -Methode erzeugt. Dazu wurde $t_{max} = 9, l = 0.5, i = 0.4, \alpha = 1.0, \beta = 0.1$ gewählt.	25
5.1	Das Sequenzdiagramm zeigt eine Beispielkommunikation, um ein Bild vom Server anzufordern.	29
5.2	Hier wird die Aufteilung einer 16 Bit-Zahl auf die Farbkanäle, links rot und rechts grün, visuell verdeutlicht. Das Schlüsselwort d_{up} bezeichnet die ersten 8 Bit und d_{low} die zweiten 8 Bit.	30
5.3	Unterteilung eines Rechtecks in vier Teilflächen. An den grünen markierten Positionen gibt es mehrere Möglichkeiten, wie die Eckpunkte der betreffenden Rechtecke gesetzt werden können.	31
6.1	Die Diagramme a und b zeigen die Ergebnisse des Datensatzes <i>TestSpheres</i> , während c und d die Resultate des Datensatzes <i>CoolRandom</i> abtragen. Dabei zeigen a und c die Ergebnisse des SSIM, während in b und d die Ergebnisse des PSNR gezeigt werden.	34
6.2	Darstellung der Graphen aus der Abbildung 6.1 im Winkelintervall von 0 bis 10 Grad.	35
6.3	Die Diagramme a und b zeigen die Ergebnisse des Datensatzes <i>TestSpheres</i> , während c und d die Resultate des Datensatzes <i>CoolRandom</i> abtragen. Dabei zeigen a und c die Ergebnisse des SSIM, während in b und d die Ergebnisse des PSNR gezeigt werden.	36

6.4	Die Diagramme a und b zeigen die Ergebnisse des Datensatzes <i>TestSpheres</i> , während c und d die Resultate des Datensatzes <i>CoolRandom</i> abtragen. Dabei zeigen a und c die Ergebnisse des SSIM, während in b und d die Ergebnisse des PSNR gezeigt werden.	37
6.5	Ergebnisse des <i>Quadtree</i> -Ansatz mit <i>TestSpheres</i> -Datensatz, bei einem konstanten Kamerawinkelunterschied von 20 Grad. Dabei liegt die maximale Baumtiefe in a,b bei 10, in c,d bei 9 und in e,f bei 8. Die Diagramme auf der linken Seite zeigen die Güte mit Hilfe des SSIM Wertes. Auf der rechten Seite wird die Konstruktionszeit des Netzes abgetragen, die der Server benötigt.	38
6.6	Ergebnisse des <i>Quadtree</i> -Ansatz mit <i>CoolRandom</i> , bei einem konstanten Kamerawinkelunterschied von 20 Grad. Dabei liegt die maximale Baumtiefe in a,b bei 10, in c,d bei 9 und in e,f bei 8. Die Diagramme auf der linken Seite zeigen die Güte mit Hilfe des SSIM-Wertes. Auf der rechten Seite wird die Gesamtkonstruktionszeit des Netzes abgetragen.	39
6.7	Die Diagramme zeigen für den <i>Quadtree</i> -Ansatz bei einem konstanten Kamerawinkelunterschied von 20 Grad die Anzahl der Dreiecke in Abhängigkeit der Parameter l , i und t_{max} . Dabei liegt die maximale Baumtiefe in a,b bei 10, in c,d bei 9 und in e,f bei 8. Die Diagramme auf der linken Seite zeigen die Anzahl der Dreiecke von der <i>TestSpheres</i> Szene. Auf der rechten Seite wird die Anzahl der Dreiecke von <i>CoolRandom</i> dargestellt.	41
6.8	Die Diagramme a, b stellen die Ergebnisse von dem <i>CoolRandom</i> Datensatz und c, d die des <i>TestSpheres</i> -Datensatzes, erhoben mit <i>FloydSteinberg</i> -Ansatz dar. Die Diagramme auf der linken Seite zeigen die Güte mit Hilfe des SSIM, und die Diagramme auf der rechte Seite bilden die Gesamtkompressionszeit des Netzes ab.	43
6.9	Es wird der Datensatz <i>TestSpheres</i> betrachtet. Die Diagramme zeigen die Güteentwicklung in Abhängigkeit des Parameters j. Dabei liegt die maximale Baumtiefe in a,b bei 10, in c,d bei 9 und in e,f bei 8.	45
6.10	Es wird der Datensatz <i>CoolRandom</i> betrachtet. Die Diagramme zeigen die Güteentwicklung in Abhängigkeit des Parameters j. Dabei liegt die maximale Baumtiefe in a,b bei 10, in c,d bei 9 und in e,f bei 8.	46
6.11	48
6.12	SSIM und Netzkonstruktionszeit bei einem konstanten Winkelunterschied von 20 Grad.	49
6.13	50
6.14	Vergleich zwischen nicht optimierten und einem optimiertem Netz.	51
6.15	Die Diagramme stellen die Zeit, die der Client, zur Extrapolation der Frames benötigt grafisch dar. Dabei ist im Diagramm a die Vollvernetzung zu sehen und im Diagramm b die Delaunay-Triangulation.	51
7.1	54
7.2	Die Dreiecksnetze a-c wurden mit dem <i>FloydSteinberg</i> -Ansatz erzeugt und die Dreicksnetze d-i mit dem <i>Quadtree</i> -Ansatz.	55
7.3	Vergleich von nachbearbeiteten Dreiecksnetzen, die mit dem <i>Quadtree</i> -Ansatz erzeugt wurden. Dabei wurde $\alpha = 10$ und $\beta = 0.2$ gewählt.	56
7.4	Beispiele für Schnittkantenartefakte in der Szene <i>TestSpheres</i> , des ersten Frames. Das Netz wurde mit $l = 0.7$, $i = 0.6$, $\alpha = 100$, $\beta = 0.2$ erzeugt.	57
7.5	title	58
7.6	<i>FloydSteinberg</i> -Ansatz mit den Parametern $\delta = 0.8$, $\gamma = 0.8$	59

Tabellenverzeichnis

3.1 Die Tabelle zeigt eine Übersicht über die Winkel der aufgenommenen Sequenzen. 7

Danksagung

An dieser Stelle möchte ich meinen Eltern, meinen Freunden und meinen Betreuer für ihre Unterstützung bedanken.

