UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

Fakultät für Elektrotechnik, Informatik und Mathematik
Heinz Nixdorf Institut & Institut für Informatik
Fachgruppe Algorithmen und Komplexität

Dissertation

Multi-Algorithmen-Rendering

Darstellung heterogener 3-D-Szenen in Echtzeit

Ralf Petring

Paderborn, 31. Oktober 2013

Gutachter: Prof. Dr. Friedhelm Meyer auf der Heide
Prof. Dr. Gitta Domik-Kienegger

Kontakt: Ralf Petring <ralf@petring.net>

Diese Arbeit wurde teilweise finanziert durch das DFG Schwerpunktprogram 1307 Algorithm Engineering <http://www.algorithm-engineering.de>

Zusammenfassung

Viele große virtuelle 3-D-Szenen sind nicht gleichmäßig strukturiert, z. B. weil sie eine stark schwankende Dichteverteilung der Geometrie aufweisen. Für solche ungleichmäßig strukturierten Szenen gibt es keinen Algorithmus, der an allen Betrachterpositionen gut funktioniert, also für beliebige Positionen ein Bild mit guter Qualität in annehmbarer Zeit darstellt. Für eine kleine Teilmenge dieser Szenen kann die Situation verbessert werden, indem ein erfahrener Benutzer per Hand für einzelne Bereiche einer Szene entscheidet, mit welchem Algorithmus diese dargestellt werden. In dieser Arbeit wird ein Verfahren vorgestellt, welches automatisch unterschiedliche Algorithmen gleichmäßig strukturierten Bereichen einer Szene zuordnet. Die Methode teilt dafür die Szene in brauchbare Bereiche auf und misst das Verhalten unterschiedlicher Algorithmen auf diesen Bereichen in einem Vorberechnungsschritt. Zur Laufzeit werden diese Daten dann genutzt, um eine Vorhersage für die Laufzeit und den entstehenden Bildfehler der einzelnen Algorithmen auf den Bereichen der Szene abhängig von der Position des Betrachters zu erhalten. Mittels Lösens eines in einen Regelkreis eingebetteten Optimierungsproblems kann dann die Bildqualität durch eine geschickte Zuordnung von Algorithmen zu Bereichen bei nahezu konstanter Bildrate optimiert werden. In einer experimentellen Evaluierung werden sowohl Laufzeit als auch Bildqualität unserer Methode mit denen von Standardverfahren verglichen.

Abstract

Many large virtual 3D scenes are not structured evenly, for example because they exhibit a highly varying density distribution of their polygons. For such heterogeneous data, there is no single algorithm that constantly performs well with any type of scene and that is able to render the scene at each position fast with the same high image quality. For a small set of scenes, this situation can be improved, if an experienced user is able to manually assign different rendering algorithms to particular parts of the scene. We introduce the Multi-Algorithm-Rendering method which automatically deploys different rendering algorithms simultaneously for a broad range of scene types. The method divides the scene into suitable subregions and measures the behavior of the different algorithms for each region in a preprocessing step. During runtime, this data is utilized to compute an estimate for the quality and running time of the available rendering algorithms from the observer's point of view. By solving an optimizing problem embedded in a control cycle, the image quality can be optimized by the appropriate assignment of algorithms to regions, while keeping the achieved frame rate almost constant. In an experimental evaluation, we compare our method's running time and image quality with that of standard rendering algorithms.

Inhaltsverzeichnis

1 Einleitung

Die dreidimensionale Darstellung virtueller Szenen wird heutzutage in nahezu allen Bereichen eingesetzt. Dies geht von Computerspielen und Filmen über Präsentation und Simulation von Fertigungsanlagen oder ganzen Städten bis hin zu Navigationsgeräten und anderen mobilen Geräten. Zwar wird die Hardware zur Darstellung solcher Szenen stetig besser, jedoch steigen gleichzeitig auch die Anforderungen an die Darstellungssysteme: Die Szenen bzw. Welten werden immer größer und detaillierter, die Darstellung soll stets noch realistischer sein, die Systeme sollen auch auf mobilen Geräten lauffähig sein etc. Ein Ende dieses Trends ist nicht abzusehen. In allen Bereichen, wo ein Benutzer durch die Szene navigiert, wird das Problem zusätzlich verschärft, da hier noch die Anforderung hinzukommt, dass die Szene in Echtzeit dargestellt werden muss. Ist dies aufgrund der Komplexität der Szene nicht möglich, kann man auf Approximationen zurückgreifen. Damit erkauft man eine schnellere Darstellung auf Kosten der Bildqualität. Ist hingegen ein Anwendungsfall gegeben, bei dem ein korrektes Bild benötigt wird, wie zum Beispiel bei Anwendungen im medizinischen Bereich, so muss darauf verzichtet werden.

1.1 Motivation und Handlungsbedarf

Bei vielen Anwendungsfällen müssen Szenen dargestellt werden, die heterogen – also in sich nicht gleichmäßig – strukturiert sind. Dies kann sich z. B. dadurch äußern, dass die Dichteverteilung der Polygone in der Szene nicht gleichmäßig ist. Abbildung 1.1 zeigt hierfür ein Beispiel. Nahezu alle Gebäude der Stadt sind innen leer und bestehen aus wenigen hundert Polygonen. Allerdings gibt es sehr viele dieser Gebäude. Die beiden großen Gebäude am Ende der Stadt sind hingegen innen detaillierter modelliert. Das Kraftwerk enthält ein komplexes Rohrsystem und das rechte Gebäude besitzt Mobiliar. Die Statue ist ein extrem detailliertes Einzelmodell, dessen alleinige (korrekte) Darstellung bereits eine Grafikkarte auslasten kann. Die Struktur dieser Szene ist folglich nicht gleichmäßig, sie enthält sowohl Bereiche mit vielen einfachen Objekten, als auch Bereiche mit wenigen komplexen Objekten.

Solche Szenen treten in der Praxis durchaus auf, da z. B. ein Architekt, der einem Kunden ein Haus präsentieren möchte, vermutlich das Haus detaillierter gestalten wird (Möbel etc.) als das umliegende Gelände. Ein Städteplaner hingegen wird auf Mobiliar vermutlich komplett verzichten und stattdessen die Umgebung genauer und weitläufiger modellieren. Wenn ein industrieller Maschinenbaubetrieb seine Fertigung optimieren oder durch zusätzliche Maschinen erweitern möchte, so könnte dazu z. B. im Vorfeld eine Materialflusssimulation durchgeführt werden. Die dabei entstehende Szene ist im Allgemeinen ebenfalls heterogen, da hierbei zum einen die Daten der Halle – seien sie vom Architekten zur Verfügung gestellt oder per Hand modelliert – und zum anderen die Daten der Maschinen aufeinandertreffen. Die Maschinendaten selbst können auch bereits heterogen sein, wenn die Daten der im Betrieb hergestellten Maschinen aus dem eigenen CAD-System erzeugt wurden und somit eventuell sehr genau sind, während die Daten der restlichen Maschinen weniger komplex

(a) Sichtposition mit Heterogenität in der Ferne.

(b) Sichtposition mit Heterogenität nahe der Kamera.

Abbildung 1.1: Beispiel für eine heterogene Szene: Eine weitläufige Stadt mit sehr vielen aber einfachen Objekten, darin jedoch zwei Gebäude mit komplexen Objekten sowie eine extrem detaillierte Statue.

sind, da zu diesen im Normalfall keine CAD-Daten verfügbar sind. Sobald es darum geht eine Szene interaktiv zu präsentieren, muss die Darstellung der Szenen in Echtzeit erfolgen, da sonst der Benutzer nicht flüssig durch die Szene navigieren kann. Es gibt also eine Mindestanzahl an Bildern, die pro Sekunde dargestellt werden muss, wodurch die Zeit, die zur Darstellung eines einzelnen Bildes zur Verfügung steht, begrenzt ist.

Die sich hieraus ergebende Problemstellung ist die Darstellung dieser heterogenen Szenen in Echtzeit mit möglichst guter Bildqualität. Die Laufzeit von Algorithmen und bei approximativen Darstellungsalgorithmen auch die Bildqualität hängt jedoch unter anderem von der Struktur der Szene und der Position des Betrachters ab. Das hat zur Folge, dass den Algorithmen die optimale Darstellung einer heterogenen Szene Probleme bereitet.

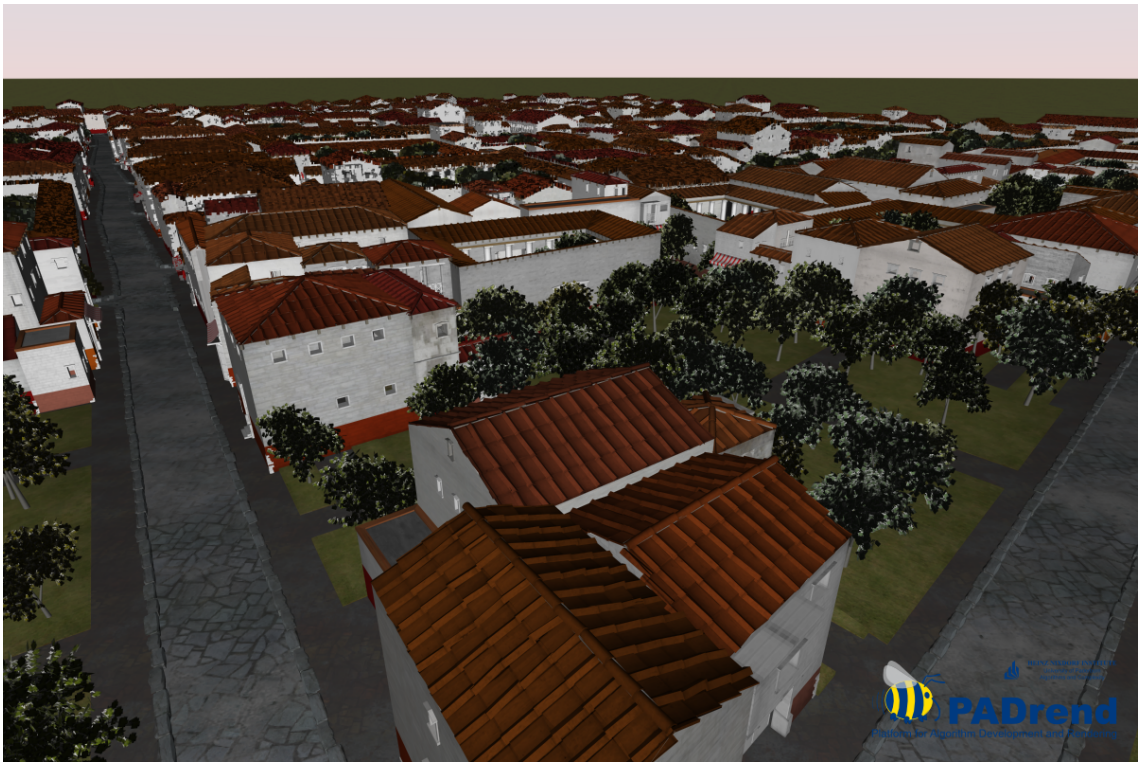
1.2 Ansatz des Multi-Algorithmen-Renderings

Approximative Darstellungsalgorithmen sind nur für Bereiche geeignet, in denen die Komplexität hoch und der durch die Approximation erzeugte Fehler im Bild klein ist, während Algorithmen, die verdeckte Objekte der Szene bei der Darstellung auslassen, nur für Positionen geeignet sind, an denen auch verdeckte Objekte vorhanden sind.

Des Weiteren kann es nicht nur notwendig sein, den Darstellungsalgorithmus abhängig von der Position des Benutzers zu wählen, sondern zusätzlich während der Erzeugung eines jeden Bildes für unterschiedlich strukturierte Bereiche der Szene den Algorithmus zu wechseln. Für das Beispiel in Abbildung 1.1 bedeutet dies, dass es besser ist, das Kraftwerk – obwohl es im Hintergrund ist – mit Hilfe von Verdeckungstests korrekt darzustellen, und einen Großteil der Häuser in mittlerer Entfernung nur approximiert darzustellen. Im ersten Bild ist es vielleicht noch möglich, die Szene mit einem einzigen Verfahren (z. B. Approximation auf Entfernung) effizient und mit hoher Qualität darzustellen. Wenn die Statue sich jedoch näher an der Kamera befindet, wie im rechten Bild, ist dies nicht mehr möglich, da die Statue auf jeden Fall approximiert werden muss. Hier müsste man also einen Spezialfall handhaben. Es ist natürlich möglich einen Algorithmus zu entwickeln, der mit dieser Problematik zurechtkommt, jedoch ist es dann auch wiederum möglich eine Szene zu konstruieren, in der ein anderer Spezialfall auftritt. Dies führt am Ende zu einem Algorithmus, der extrem kompliziert ist, fast nur aus Spezialfallbehandlungen besteht und ständig angepasst werden muss.

Selbst bei homogenen Szenen ist die Darstellung von Teilbereichen der Szene mit unterschiedlichen Algorithmen ab einer gewissen Komplexität sinnvoll. Ein Beispiel dafür ist die Stadt ohne die Statue und die beiden komplexen Gebäude. Betrachtet man die beiden Sichtpositionen in Abbildung 1.2, so treten bei der Darstellung des ersten Bildes keinerlei Probleme auf. Solange sich die Kamera in der Stadt befindet, kann die komplette Szene mit Hilfe von Verdeckungstests dargestellt werden. Sobald die Kamera jedoch über der Stadt schwebt, wie im rechten Bild, sollten mehrere Algorithmen eingesetzt werden. In diesem Beispiel: eine korrekte Darstellung im Nahbereich, eine gute Approximation (pro Objekt) im mittleren Entfernungsbereich und eine gröbere Approximation (Objektgruppen) für entfernte Bereiche.

(a) Sichtposition auf einer Straße.



(b) Sichtposition oberhalb der Stadt.

Abbildung 1.2: Beispiel für eine homogene Szene: Eine weitläufige Stadt mit sehr vielen aber einfachen Objekten.

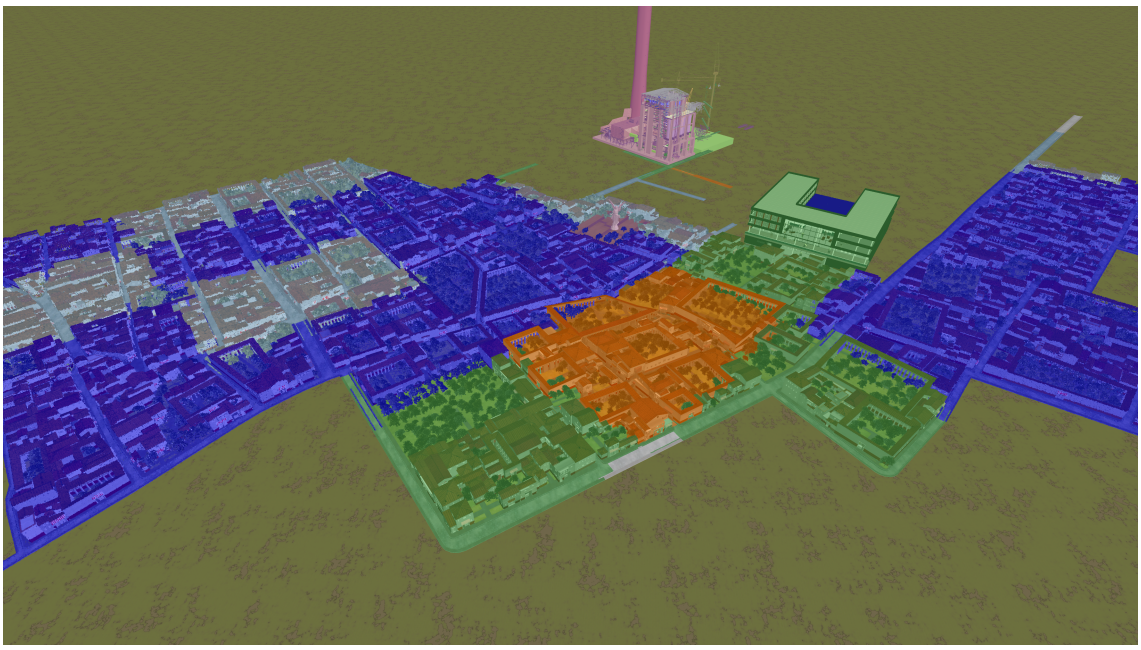


Abbildung 1.3: Darstellung einer Szene durch Multi-Algorithmen-Rendering. Unterschiedlich eingefärbte Regionen wurden mit unterschiedlichen Algorithmen dargestellt.

1.3 Ziel und eigener Beitrag

Ziel dieser Arbeit ist die Entwicklung und Evaluierung eines Meta-Renderingverfahrens (Multi-Algorithmen-Rendering) zur Echtzeitdarstellung dreidimensionaler Szenen, welches den im vorherigen Abschnitt beschriebenen Ansatz ausnutzt um die Bildqualität bei vorgegebener Laufzeit zu optimieren.

Dies wird erreicht, indem in einer Vorverarbeitung die Szene zunächst halbautomatisch in gleichmäßig strukturierte Regionen unterteilt wird, für die danach Messungen der Laufzeit und Bildqualität verschiedener Algorithmen an zufälligen Positionen durchgeführt werden. Mit Hilfe dieser Daten wird dann, während der Benutzer durch die Szene navigiert, die Bildqualität bei einstellbarer angestrebter Laufzeit vollautomatisch optimiert. Dazu werden für verschiedene Bereiche der Szene unterschiedliche – bereits existierende – Renderingalgorithmen benutzt. Das heißt, zur Darstellung jedes einzelnen Bildes wird jeder homogenen Region der Szene ein Algorithmus zugeordnet, mit dem die Region dargestellt wird. Abbildung 1.3 zeigt ein Beispiel hierfür, bei dem die Regionen entsprechend der benutzten Algorithmen eingefärbt wurden.

In dieser Arbeit ist es gelungen, den beschriebenen Ansatz auszunutzen, und ein praxistaugliches Verfahren zu entwickeln. Durch das Multi-Algorithmen-Rendering ist ein zusätzliches Spektrum an Szenen in Echtzeit darstellbar. Das Verfahren wurde implementiert und ausführlich evaluiert. Untersucht wurden unter anderem die Laufzeit und die Bildqualität bei der Darstellung, sowie die Laufzeit und der Speicherverbrauch der Vorverarbeitung. Dabei hat sich herausgestellt, dass das Verfahren in der Lage ist, die Bildrate mit annehmbaren Schwankungen um einen Vorgabewert zu regeln, so dass der Benutzer durch die Testszenen navigieren kann. Dabei wird die Qualität der Einzelbilder

auf Basis der in der Vorverarbeitung gemessenen Werte maximiert. Dies resultiert in Bilder, wie sie z. B. in Abbildung 1.1 und 1.2 zu sehen sind. Allerdings kommt es, bedingt durch das Wechseln der Algorithmen, zu störenden Flackereffekten zwischen aufeinanderfolgenden Bildern. Diese können jedoch durch zukünftige Erweiterungen des Verfahrens reduziert werden.

Das Hauptgerüst dieser Arbeit wurde bereits auf dem *11. Paderborner Workshop Augmented & Virtual Reality in der Produktentstehung* [Pet+13a] und dem *9th International Symposium on Visual Computing (ISVC 2013)* [Pet+13b] publiziert. Diese Arbeit stellt eine ausführlichere Beschreibung des Verfahrens und seiner Umsetzung sowie eine wesentlich umfangreichere Evaluierung dar. Des Weiteren wurden am Verfahren nach diesen Veröffentlichungen auch noch wesentliche Verbesserungen durchgeführt.

1.4 Aufbau der Arbeit

Im Folgenden werden zunächst in der Übersicht über den Stand der Technik (Kap. 2) die zu dieser Arbeit verwandten Verfahren erörtert, sowie die in dieser Arbeit benutzten Verfahren vorgestellt. Das Meta-Verfahren Multi-Algorithmen-Rendering (Kap. 3) wird daraufhin inklusive einer Übersicht über die Implementierung (Kap. 4) detailliert vorgestellt. Danach werden in der Evaluierung (Kap. 5) die Fragestellungen herausgearbeitet sowie die daraus resultierenden Messungen ausgewertet und zusammengefasst. Letztlich schließt ein Ausblick (Kap. 6) über Verbesserungsmöglichkeiten und mögliche zukünftige Erweiterungen die Arbeit ab.

2 Stand der Technik

Im Folgenden werden, nach einigen Begriffsdefinitionen (2.1), zunächst zum Multi-Algorithmen-Rendering verwandte Arbeiten vorgestellt (2.2). Danach werden gängige 3-D-Datenstrukturen erörtert (2.3). Darauf folgt ein Überblick über den Stand der Technik im Bereich Renderingalgorithmen (2.4, insbesondere über die in dieser Arbeit benutzten). Die Bereiche Bildqualität (2.5) und Sampling (2.6) werden ebenfalls erörtert, soweit dies für diese Arbeit notwendig ist.

2.1 Begriffsdefinitionen

Im Folgenden werden einige Begriffe definiert, welche im weiteren Verlauf dieser Arbeit häufig benutzt werden.

Primitive: Primitive sind Elemente, welche durch die Grafikkarte direkt dargestellt werden können. Dies sind Dreiecke, Linien und Punkte.

Objekt: Ein Objekt ist eine beliebig große unteilbare Menge an Primitiven. Gängige Größen für die Anzahl der Primitive eines Objekts liegen im ein- bis siebenstelligen Bereich. Der Begriff Objekt hat hierbei keine semantische Bedeutung wie Tisch, Auto, oder ähnliches. Ein Auto kann sowohl ein Objekt sein, als auch aus vielen Objekten bestehen, je nachdem, wie, von wem und wofür es modelliert wurde.

Szene: Die Szene ist eine dreidimensionale Anordnung von möglicherweise beweglichen Objekten. In dieser Arbeit werden jedoch lediglich unbewegliche Objekte betrachtet.

Szenengraph Der Szenengraph ist eine 3-D-Datenstruktur zur Speicherung der Szene. Generell ist dies ein gerichteter azyklischer Graph. Häufig wird jedoch, wie auch in dieser Arbeit, aus technischen Gründen ein Baum verwendet. Jeder Knoten im Baum kann dabei mit Zusatzinformationen wie Position, Farbe etc. annotiert sein, die sich dann auf die Darstellung des entsprechenden Teilbaums auswirken.

Bounding-Volume: Das Bounding-Volume eines Knotens im Szenengraph ist eine möglichst kleine und konvexe geometrische Form, welche alle Objekte des Knotens und seiner Kinder umschließt. Rechtwinklige Formen werden auch als Bounding-Box bezeichnet.

Renderingverfahren: Renderingverfahren im Sinne dieser Arbeit sind Algorithmen, die einen Szenengraphen traversieren und die darin enthaltene Szene mit Hilfe der Grafikkarte durch Rasterisierung [AHH08] darstellen. Die Begriffe **Algorithmus**, **Methode** und **Verfahren** werden in dieser Arbeit synonym verwendet. Andere Techniken, wie z. B. Raytracing, Image-Based Rendering oder Voxelrendering, werden in dieser Arbeit nicht betrachtet.

Meta-Renderingverfahren: Der Begriff Meta-Renderingverfahren bezeichnet in dieser Arbeit ein Verfahren, welches andere bereits existierende Algorithmen benutzt, um eine Szene darzustellen. Die Darstellung der Objekte innerhalb der Szene erfolgt dabei durch die benutzten Algorithmen und nicht durch das Meta-Verfahren selbst.

Bildrate Die Bildrate oder auch Framerate bezeichnet die Anzahl an Bildern, die von einem Renderingverfahren pro Sekunde angezeigt werden. Sie ist also das Inverse zur benötigten Zeit für die Berechnung eines Bildes.

Echtzeitdarstellung: Die Darstellung einer Szene erfolgt dann in Echtzeit, wenn die Bildrate so hoch ist, dass die Darstellung flüssig erscheint. Konkrete Ausprägungen reichen hier von einstelligen bis hin zu dreistelligen Bildraten, je nach Anforderungen durch die Art der Anwendung und den Benutzer.

Walkthrough: Bei einem Walkthrough bewegt ein Benutzer die Kamera frei durch eine Szene, welche währenddessen in Echtzeit dargestellt wird.

Frustum: Das Frustum ist der Sichtbereich des Benutzers innerhalb der Szene. Es hat die Form einer auf der Seite liegenden Pyramide mit rechteckiger Grundfläche, auf deren Spitze sich die Kamera befindet. Die Grundfläche bildet die sogenannte „far plane“, eine Ebene an der der Sichtbereich endet. Alle Primitive hinter dieser Ebene werden abgeschnitten.

Optimierung: Der Begriff Optimierung wird in verschiedenen wissenschaftlichen Bereichen unterschiedlich aufgefasst. In dieser Arbeit werden in der Vorverarbeitung Stichproben gezogen, die Messfehlern unterliegen und abhängig von einem zur Messung von Bildfehlern benutzten Abstandsmaß sind. Diese Werte werden zur Laufzeit für die aktuelle Betrachterposition interpoliert. Auf Grundlage dieser interpolierten Werte wird dann ein lineares Programm gelöst, dessen Parameter über einen Regelkreis nachjustiert werden. Die Lösung des LPs stellt also keinesfalls eine optimale Lösung für das Ausgangsproblem dar. Trotzdem wird der gesamte Vorgang in dieser Arbeit als Optimierung bezeichnet.

2.2 Verwandte Arbeiten

Es gibt einige Verfahren, die man als Vorgänger zu dieser Arbeit bezeichnen kann, da sie ebenfalls mehrere Algorithmen benutzen und die Entscheidung aufgrund einer Kosten-Nutzen-Funktion treffen.

Funkhouser und Séquin [FS93] benutzen eine Kosten-Heuristik, welche über die Anzahl Primitive im Objekt sowie dessen projizierte Größe die Zeit für dessen Darstellung schätzt. Zusätzlich wird mit Hilfe einer Nutzen-Heuristik über die projizierte Größe der Objekte noch deren Beitrag zum Bild geschätzt. Aufgrund dieser beiden Heuristiken wird dann ein Optimierungsproblem definiert, welches bei vorgegebenen Maximalkosten den Nutzen maximiert. Dieses Optimierungsproblem wird jedoch nicht gelöst, sondern lediglich mit einem Greedy-Ansatz approximiert, da es eine Variante des NP-vollständigen kontinuierlichen Multiple-Choice-Rucksackproblems [Iba+78] darstellt. Dieses ist, zumindest für Instanzen dieser Größe und auf der damaligen Hardware, nicht in Echtzeit lösbar.

Einen anderen Ansatz verfolgen Aliaga u. a. [Ali+99]. Sie benutzen auch mehrere Renderingverfahren, allerdings mit einem weniger flexiblen Konzept, welches die Szene in Nah- und Fernbereich unterteilt. Für den Nahbereich werden dann Culling und geometriebasierte approximative Darstellungsverfahren eingesetzt, während für den Fernbereich bildbasierte Verfahren eingesetzt werden, die Teile der Szene z. B. durch texturierte Quader ersetzen.

Bei Computerspielen ist die Darstellung einer Szene durch den Einsatz mehrerer Algorithmen eine durchaus gängige Praxis. Allerdings wird hier die Zuordnung von Algorithmen zu Regionen durch einen Experten manuell getroffen. Des Weiteren ist es hier durchaus möglich, die Szene selbst anzupassen, falls sie nicht von allen Positionen aus mit ausreichender Qualität und Geschwindigkeit dargestellt werden kann.

Verfahren, die das Problem mit Hilfe vorheriger Messung von Laufzeit und Bildqualität und anschließender linearer Optimierung lösen, sind aber nicht bekannt.

2.3 3-D-Datenstrukturen als Szenengraph

3-D-Datenstrukturen oder genauer gesagt Bounding-Volume-Hierarchien zum Speichern einer dreidimensionalen Szene sind hierarchische Datenstrukturen, bei denen gilt, dass das Bounding-Volume eines Knotens alle Bounding-Volumes seines Teilbaumes enthält. Ein Bounding-Volume ist dabei eine Struktur, die alle enthaltenen Objekte umschließt. Es sollte möglichst einfach aufgebaut sein, um Sichtbarkeitstests, wie z. B. eine Schnittberechnung mit dem Sichtbereich des Benutzers, effizient durchführen zu können. Andererseits sollte es die Objekte möglichst eng umschließen, um ein möglichst genaues Testergebnis zu erhalten. In *Foundations of Multidimensional and Metric Data Structures* [Sam05] wird ein genereller Überblick über 3-D-Datenstrukturen gegeben. Für Rendering gebräuchliche Strukturen sind z. B. (Loose-) Octrees [Hun78], k-d-trees [Ben75], BSP-trees [FKN80], AB-trees [Fle06] und Hull-trees [Süß+11]. All diese verfolgen einen Top-Down-Ansatz und teilen die Szene rekursiv in mehrere Teile auf. Der Unterschied liegt in der Wahl der Schnittebenen, welche zur Aufteilung benutzt werden, und der Form der Bounding-Volumes. Die Schnittebenen sind entweder beliebig (BSP-Trees) oder achsenparallel (Octrees, k-d-trees, AB-trees, Hull-trees). Es wird entweder an einer (BSP-trees, k-d-trees, AB-trees) oder gleichzeitig an mehreren (Octrees, Hull-trees) Schnittebenen geschnitten. Die Form eines Bounding-Volumes ist stets konvex. Gängige Formen von Bounding-Volumes sind konvexe Polyeder (Hull-tree, BSP-tree), Quader (k-d-trees, AB-trees), Würfel (Octrees) oder Kugeln. Welche dieser Strukturen am besten geeignet ist, hängt von der jeweiligen Eingabeszene ab. Jedoch tritt bei allen dasselbe Problem auf, wenn die Eingabe nicht aus Punkten, sondern aus Objekten mit Volumen besteht. Die Entscheidung, in welchem Kindknoten ein Objekt gespeichert wird, kann recht einfach über den Mittelpunkt des Objekts getroffen werden. Nicht so einfach zu beantworten ist jedoch die Frage, was mit Objekten passiert, die von einer Schnittebene geschnitten werden. Hierzu gibt es drei Lösungsansätze:

- Die Objekte werden zerschnitten und die entstehenden Teile im jeweils passenden Kindknoten gespeichert. Dies hat zur Folge, dass sich die Anzahl der Objekte möglicherweise stark erhöht.

- Die Objekte werden in allen Kindern redundant gespeichert. Dies hat zur Folge, dass sich zum einen der benötigte Speicherplatz erhöht und zum anderen bei der

Traversierung nicht mehr so einfach entschieden werden kann, ob ein Objekt bereits bearbeitet wurde oder nicht.

- Die Objekte werden in dem inneren Knoten gespeichert, in dessen Bounding-Volume sie noch hineinpassen. Dies hat zur Folge, dass auch kleine Objekte eventuell bereits in der Wurzel des Baums gespeichert werden. Dies kann umgangen werden, indem die Größe des Bounding-Volumes verdoppelt wird. Dann wird die Tiefe, in der das Objekt im Baum gespeichert wird, fast (± 1) nur noch durch dessen Größe bestimmt. Allerdings sind die einzelnen Bounding-Volumes einer Ebene im Baum dann nicht mehr disjunkt. Solch eine Loose-Datenstruktur wurde erstmals für Loose-Octrees [Ulr00] eingeführt, ist aber auch für andere Bounding-Volume Hierarchien umsetzbar.

Es gibt unzählige Varianten dieser Datenstrukturen, die jeweils unter bestimmten Umständen geringfügige Vorteile bieten. Bei Szenengraphen für beliebige (nicht in Voraus bekannte) Szenen ist aber keine generelle Aussage darüber zu treffen, welche Datenstruktur am geeignetsten ist. Allerdings gilt, je ungleichmäßiger eine Szene ist, desto wichtiger ist es eine der Loose-Varianten zu wählen.

Alternativ kann man eine Bounding-Volume-Hierarchie auch mit einem Bottom-Up-Anzatz, z. B. durch agglomeratives Clustering [Flo+51; Are08; Ack+12], berechnen. Hierbei startet man mit einer Menge von Gruppen (eine pro Objekt), die so lange zusammengefasst werden, bis nur noch eine Gruppe existiert. Dieser Ansatz wird jedoch aufgrund der höheren Berechnungszeit und der nur geringfügigen Vorteile in der Praxis selten eingesetzt. Außerdem muss hier die Datenstruktur bei dynamischen Updates jedes Mal komplett neu berechnet werden. Bei den Top-Down-Ansätzen ist es hingegen in der Regel möglich, die Datenstruktur zu aktualisieren.

2.4 Renderingverfahren

Renderingalgorithmen lassen sich grob aufteilen in konservative und approximative Verfahren. Konservative Renderingalgorithmen sind solche, die zumindest alle sichtbaren Objekte korrekt darstellen und keinen Fehler im resultierenden Bild erzeugen. Approximative Renderingalgorithmen sind solche, die zugunsten einer besseren Laufzeit Teile der Szene nicht korrekt oder gar nicht darstellen und dadurch eventuell einen Fehler im resultierenden Bild erzeugen. Für eine Übersicht über verschiedene Renderingtechniken sei hier auf die Artikel von Cohen-Or u. a. [Coh+03] sowie Gobbetti, Kasik und Yoon [GKY08] verwiesen. Im Folgenden werden nur die für diese Arbeit relevanten Aspekte erörtert.

2.4.1 Konservative Renderingverfahren

Die einfachste Variante eines konservativen Renderingalgorithmus ist der z-Buffer-Algorithmus [Str74; Cat74]. Hierbei wird aus Sicht der CPU jedes Objekt genauso dargestellt, wie es modelliert wurde; unabhängig davon ob es sichtbar ist oder nicht. Der z-Buffer-Algorithmus selbst ist auf der Grafikkarte in Hardware implementiert und testet jedes Mal, wenn ein Pixel beschrieben werden soll, ob der Tiefenwert des neuen Pixels vor dem des alten – also näher zum Betrachter – liegt. Nur wenn dies der Fall ist, wird der Pixel überschrieben und ansonsten verworfen. Beschleunigen kann man die Darstellung durch Auslassen der nicht sichtbaren Objekte. Dieser Vorgang nennt sich Culling und kann in drei Kategorien aufgeteilt werden:

- Frustum-Culling überspringt Objekte, die sich nicht im Sichtbereich des Benutzers befinden. Mit einer geeigneten Datenstruktur geht dies sehr schnell und sollte immer ausgeführt werden, da sich der Aufwand nicht bemerkbar macht, selbst wenn die komplette Szene im Sichtbereich liegt und somit kein Nutzen erzielt wird.
- Backface-Culling überspringt Dreiecke, deren Oberflächennormale dem Benutzer abgewandt ist. Es kann auf zwei Arten genutzt werden: Auf der Grafikkarte ausgeführt kann es lediglich ein- und ausgeschaltet werden. Ist die Darstellung eines Dreiecks sehr einfach, so bringt es keinen Nutzen, da die Darstellung eines Dreiecks dann nicht länger dauert als der Test, ob es sichtbar ist. Mit steigender Zeit für die Darstellung eines Dreiecks, z. B. durch aufwendige Effekte, steigt auch der Nutzen von Backface-Culling. Hierbei ist dann jedoch auch zu beachten, dass solche Dreiecke, die von beiden Seiten zu sehen sein sollen, doppelt vorhanden sein müssen. Auf der CPU ausgeführtes Backface-Culling teilt meist die Objekte in mehrere Teile auf, welche dann die Dreiecke enthalten, deren Oberflächennormale ungefähr in die gleiche Richtung zeigt, so dass zur Laufzeit nur noch ca. die Hälfte der Dreiecke dargestellt werden muss.
- Occlusion-Culling ist die Art des Cullings, bei der man durch geschickte Algorithmen am meisten erreichen kann. Hierbei werden Objekte bei der Darstellung übersprungen, die durch andere Objekte verdeckt werden. Hierzu gibt es verschiedene Techniken, die im Folgenden näher erläutert werden.

Die Occlusion-Culling-Verfahren kann man weiter unterteilen in Online- und Offlineverfahren. Offline-Occlusion-Culling-Verfahren berechnen die Sichtbarkeit der Objekte bereits vor dem eigentlichen Walkthrough in einem Vorverarbeitungsschritt. Die so gewonnenen Informationen werden zur Nutzung während des Walkthroughs gespeichert. Onlineverfahren hingegen berechnen die Sichtbarkeit der Objekte zur Laufzeit, also während die Szene dargestellt wird. Es existieren auch Mischformen dieser beiden Ansätze, bei denen eine im Vorfeld berechnete Menge potentiell sichtbarer Objekte zur Laufzeit weiter eingeschränkt wird.

2.4.1.1 Offline-Occlusion-Culling-Verfahren

Die exakte Berechnung der globalen Sichtbarkeitsinformationen – also die Aufteilung des 3-D-Raums in Bereiche, aus denen jeweils immer exakt die gleiche Menge an Dreiecken sichtbar ist – resultiert in einem Aspektgraphen [PD90], der eine Speicherplatz-Komplexität von $\mathcal{O}(n^9)$ hat. Dadurch ist er, selbst wenn n statt der Anzahl an Primitiven nur die Anzahl an Objekten in der Szene bezeichnet, in der Praxis nicht einsetzbar. Es gibt jedoch Verfahren, die eine Approximation dieses Graphen berechnen [ARBJ90; TS91]. Dazu wird die Szene z. B. in Zellen aufgeteilt und lediglich pro Zelle die Sichtbarkeit der anderen Zellen berechnet. Dies geschieht mit Hilfe von Portalen, welche die Zellen miteinander verbinden und funktioniert besonders gut bei architektonischen Szenen. Dabei sind dann die Zellen die Räume und die Portale die Türen bzw. Fenster. Die so entstehende Datenstruktur liefert zu jeder Zelle eine Menge von möglicherweise sichtbaren Zellen bzw. deren Objekte. Diese Menge ist ein Potentially-Visible-Set (PVS) und wird dargestellt, wenn sich der Benutzer in der Zelle aufhält. Sie umfasst zumindest die Objekte, welche von mindestens einem Standpunkt innerhalb der aktuellen Zelle aus sichtbar sind. Sie ist also eine Obermenge aller vom aktuellen Standpunkt aus sichtbaren Objekte.

2.4.1.2 Online Occlusion-Culling Verfahren

Von Onlineverfahren spricht man, wenn die Entscheidung, ob ein Objekt bei der Darstellung ausgelassen wird oder nicht, erst zur Laufzeit getroffen wird. Ein Beispiel hierfür ist ein Verfahren, das Hierarchical-Occlusion-Maps (HOM) [Zha+97] nutzt. Hierbei wird zunächst eine geschickt gewählte Menge von Objekten ausgewählt, welche große Teile der Szene verdeckt (Occluder). Diese werden dann zunächst in ein Schwarz-Weiß-Bild gerendert. Danach wird durch stufenweise Verkleinerung eine Bildpyramide erzeugt, mit Hilfe derer dann die Sichtbarkeitstests für die übrigen Objekte (Occludees) durchgeführt werden. Stand der Technik in diesem Bereich ist jedoch das Coherent-Hierarchical-Culling++-Verfahren (CHC++) [MBW08], welches eine Optimierung des CHC-Verfahrens [Bit+04] ist. Hierbei wird eine Fähigkeit neuerer Grafikkarten ausgenutzt, welche in der Lage sind zu bestimmen, wie viele Pixel eines Objektes bei dessen Darstellung beschrieben wurden. Dieser Wert kann abgefragt werden, sobald die Geometrie des Objekts die Grafikpipeline passiert hat. Stellt man nun nicht das Objekt selbst dar, sondern ein einfaches Bounding-Volume, so ergibt dies einen konservativen Sichtbarkeitstest für Objekte oder auch für ganze Teilbäume des Szenengraphen. Der CHC++-Algorithmus führt diese Verdeckungstests sehr geschickt für Teile der Szene durch, ohne dass dabei größere Wartezeiten auf die Ergebnisse der Tests entstehen.

2.4.2 Approximative Renderingverfahren

An dieser Stelle sind zunächst alle konservativen Renderingverfahren erneut zu nennen, da man jedes dieser Verfahren in ein approximatives Verfahren verwandeln kann. Hierbei lässt man zu, dass auch Objekte, die nur zu einem kleinen Teil sichtbar sind, bei der Darstellung ausgelassen werden. Zusätzlich gibt es noch einige approximative Culling-Verfahren, die keine konservative Variante haben. Hierzu gehört z. B. das Spherical-Visibility-Sampling (SVS) [Eik+13], das die Szene nicht (wie zur Berechnung von PVS häufig der Fall) in Zellen aufteilt, sondern stattdessen eine Kugel um einen Teilbaum des Szenengraphen legt und für beliebige Positionen außerhalb dieser Kugel abschätzt, welche der Objekte aus dem inneren der Kugel an dieser Position sichtbar sind. Dadurch ist die Komplexität der aufgebauten Struktur nicht von der Anzahl der Zellen abhängig, sondern nur von der Anzahl der Knoten des Szenengraphen.

Dazu kommen dann noch die Verfahren, welche die Darstellung beschleunigen, indem sie Objekte oder ganze Teilbäume des Szenengraphen durch einfachere, schneller darzustellende Ersatzrepräsentationen austauschen. Algorithmen, die einzelne Objekte austauschen, sind z. B. das Discrete-Level-of-Detail [Lue+03]. Hierbei wird im Vorfeld, mit Hilfe von Meshreduktionsverfahren [GH97; GH98] eine konstante Anzahl an reduzierten Varianten des Objekts berechnet, welche dann zur Laufzeit je nach projizierter Größe des Objekts benutzt werden. Eine Erweiterung davon stellen Progressive-Meshes [Hop96] dar, bei denen eine kontinuierliche, reproduzierbare Reduktion des Objekts bis auf konstante Größe vorberechnet wird. Dadurch steht zur Laufzeit jede benötigte Komplexitätsstufe des Objekts zur Verfügung. Das Verfahren ist sogar bedingt in der Lage die Komplexität eines Meshes nur in bestimmten Bereichen zu reduzieren. Des Weiteren gibt es Verfahren, welche ganze Teilbäume des Szenengraphen ersetzen können. Ein früher Vertreter dieser Klasse von Algorithmen ist das Color-Cubes-Verfahren [Cha+96]. Dieses ersetzt Teilbäume ab einer konstanten projizierten Größe durch eine teilweise transparente farbige Boundingbox. Dabei

entspricht die Farbe der Seitenflächen der Box der durchschnittlichen Pixelfarbe eines gerenderten Bildes vom Inhalt des zu ersetzenden Knotens. Bei Objekten, deren projizierte Größe über dem gegebenen Schwellwert liegt, wird die Originalgeometrie dargestellt. Ein Verfahren mit ähnlicher Vorgehensweise ist Progressive-Blue-Surfels [Jäh+13], hierbei werden jedoch die Teilbäume durch Punktwolken ersetzt. Der Begriff Surfel ist eine Kombination der Wörter *Surf*ace und Pix*el* bzw. *Surf*ace und *el*ement und bezeichnet Punkte auf der Oberfläche von Objekten. Die Surfels werden hierbei ähnlich zu der von Pfister u.a. [Pfi+00] beschriebenen Vorgehensweise erzeugt. Resultat ist in diesem Fall jedoch eine progressive Liste von Surfels, deren Präfix der Länge k jeweils eine gute Approximation des Teilbaums bei einer projizierten Größe von $c \cdot k$ bilden. Auch bei diesem Verfahren wird die Originalgeometrie dargestellt, wenn die Anzahl verfügbarer Surfels zu klein für die projizierte Größe eines Objektes ist.

2.5 Bewertung der Bildqualität approximativer Darstellungsverfahren

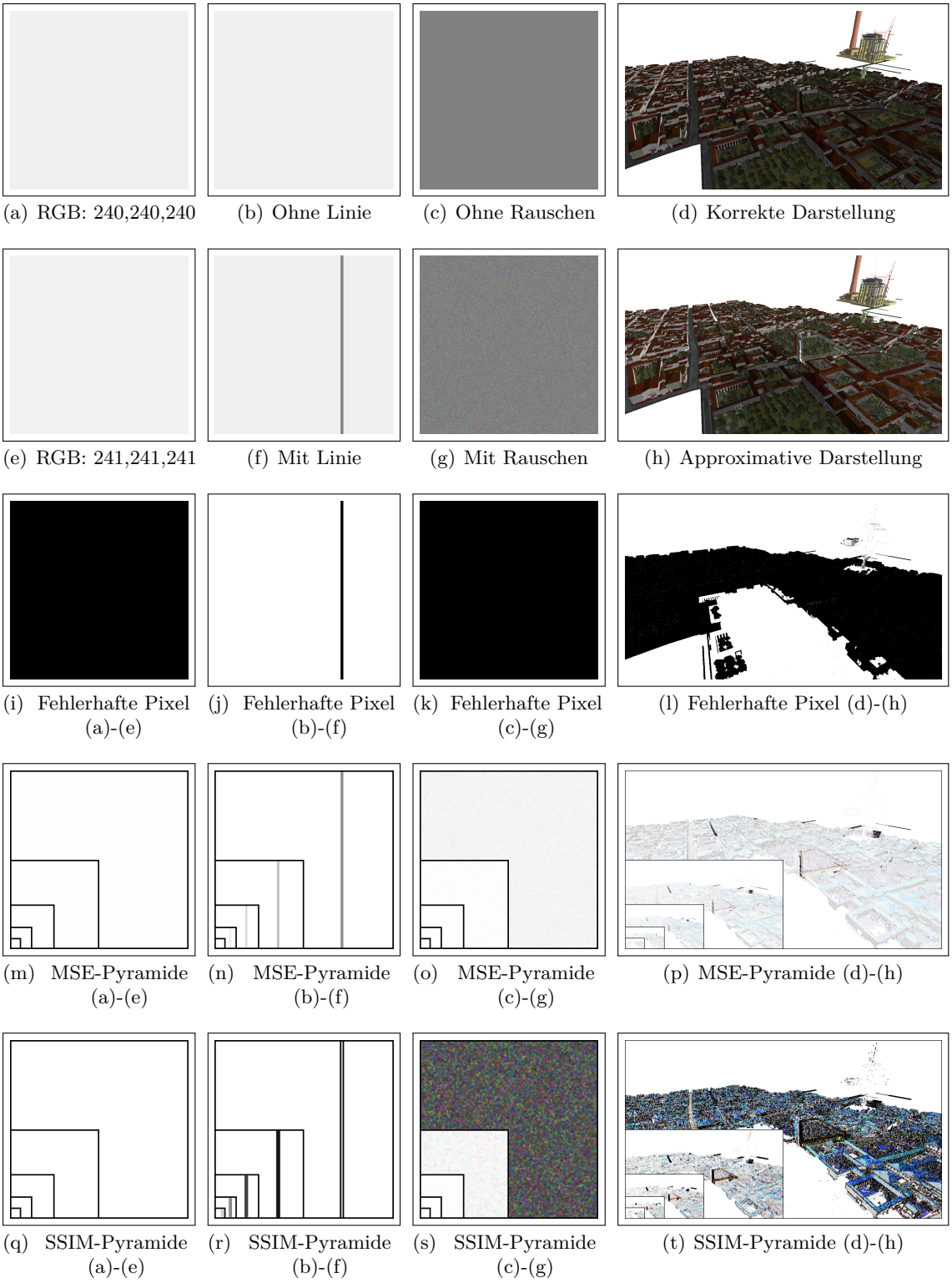
Die Bewertung der Qualität eines gerenderten Bildes, bzw. des Fehlers in einem solchen, geschieht, indem das approximativ dargestellte Bild mit einem konservativ dargestellten Bild verglichen wird. Einen generellen Überblick über den Stand der Technik im Bereich Bildqualität geben die Arbeiten von Pedersen [Ped11] bzw. Pedersen und Hardeberg [PH12]. Abbildung 2.1 zeigt einige Beispiele inklusive der Fehlerbilder unterschiedlicher Vergleichsverfahren. Ein dunkleres Bild bedeutet hier einen größeren Fehler, allerdings darf man nicht Fehlerbilder unterschiedlicher Verfahren miteinander vergleichen.

Die einfachste Variante einer Bildfehlerbewertung ist ein pixelweiser Vergleich der Bilder, bei dem die Anzahl fehlerhafter Pixel ermittelt wird. Diese Vorgehensweise unterscheidet nicht zwischen leicht falschen und völlig falschen Pixeln. Dies bedeutet für das erste (linke) Beispiel in der Abbildung, dass alle Pixel falsch sind (i), da im zweiten Bild (e) alle Pixel minimal heller sind als im ersten Bild (a). Daraus ist ersichtlich, dass diese Vorgehensweise nicht der menschlichen Wahrnehmung entspricht, da dieser Unterschied mit dem bloßen Auge nicht zu erkennen ist und ein Mensch eine solche Approximation als sehr gut bezeichnen würde.

Dieses Problem entsteht nicht, wenn man falsche Pixel mit dem (quadratischem) Abstand der Farbwerte (Mean-Squared-Error, MSE) [WB09] im RGB-Raum gewichtet. Dies führt in diesem Beispiel zu einem sehr kleinen Fehlerwert (m). Das zweite Beispiel zeigt ein weiteres Problem auf. Hier ist im zweiten Bild (f) ein Strich vorhanden, der im ersten Bild (b) fehlt. Der MSE-Fehlerwert (n) wäre in diesem Fall jedoch ähnlich dem aus dem vorherigen Beispiel, obwohl ein Mensch sagen würde, dass der Fehler hier wesentlich größer ist als zuvor.

Eine Methode, die diese und andere Problematiken beachtet, ist das Structural-Similarity-Verfahren (SSIM) [Wan+04]. Es ist in der Lage strukturelle Unterschiede in Bildern zu erkennen. Dazu wird, statt nur je zwei korrespondierende Pixel miteinander zu vergleichen, auch deren Nachbarschaft in die Berechnung einbezogen. Dadurch ist es möglich auch die Ableitung des Fehlerverlaufs innerhalb des Bildes zu bewerten. Dies hat zur Folge, dass im zweiten Beispiel der fehlerhafte Strich bei der Bewertung (r) hervorgehoben wird und somit stärker in die Fehlerberechnung einfließt.

Ein drittes Problem bei der Bewertung von Bildfehlern stellt Rauschen dar. Betrachtet



(a) RGB: 240,240,240 (b) Ohne Linie (c) Ohne Rauschen (d) Korrekte Darstellung

(e) RGB: 241,241,241 (f) Mit Linie (g) Mit Rauschen (h) Approximative Darstellung

(i) Fehlerhafte Pixel (a)-(e) (j) Fehlerhafte Pixel (b)-(f) (k) Fehlerhafte Pixel (c)-(g) (l) Fehlerhafte Pixel (d)-(h)

(m) MSE-Pyramide (a)-(e) (n) MSE-Pyramide (b)-(f) (o) MSE-Pyramide (c)-(g) (p) MSE-Pyramide (d)-(h)

(q) SSIM-Pyramide (a)-(e) (r) SSIM-Pyramide (b)-(f) (s) SSIM-Pyramide (c)-(g) (t) SSIM-Pyramide (d)-(h)

Abbildung 2.1: Beispiele für Bildfehlerbewertungen mit verschiedenen Verfahren. Zeile eins und zwei zeigen die verglichenen Bilder, drei bis fünf die Fehlerbilder.

man die Bilder im dritten Beispiel (c),(g), so sehen diese sehr ähnlich aus. Das zweite Bild ist lediglich ein wenig verrauscht. Dies hat allerdings zur Folge, dass das SSIM-Verfahren den Fehler im Bild (s) als sehr hoch bewertet, da er sich von Pixel zu Pixel ständig ändert. Diesem kann man entgegenwirken, indem man beim Vergleich der Bilder eine Bildpyramide [Bur81] aufbaut, indem man die zu vergleichenden Bilder sukzessiv verkleinert und dann paarweise mit Hilfe eines der vorherigen Verfahren vergleicht. Der Gesamtfehler ergibt sich hierbei dann durch eine gewichtete Aufsummierung der Einzelfehler. Diese Vorgehensweise sorgt dafür, dass Fehler, die nur räumlich begrenzte, verteilte Bereiche des Bildes betreffen oder stark fluktuieren, wie Rauschen, beim Verkleinern der Bilder in der Pyramide verschwinden. So ist im dritten Beispiel (s) das Rauschen bereits nach der ersten Verkleinerung fast verschwunden, während der Strich im zweiten Beispiel (r) jedoch erhalten bleibt. Dieses Verfahren kann auch mit dem MSE-Verfahren kombiniert werden, der beschriebene Effekt ist dort jedoch nur schwach zu erkennen, da dieses generell weniger anfällig für Rauschen ist.

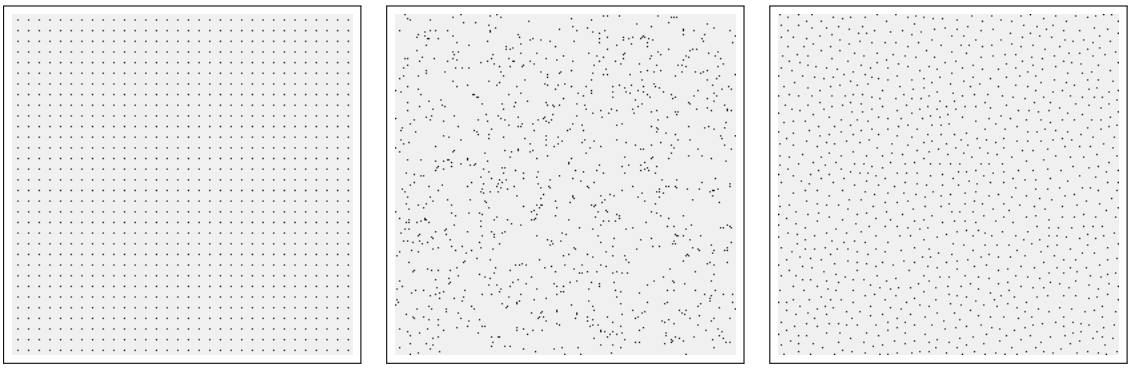
Das vierte Beispiel vergleicht die korrekte Darstellung einer gerenderten Szene (d) mit einer Approximation (h). Auch hier ist zu erkennen, dass das Rauschen, welches durch die Approximation entsteht, bereits in der zweiten Ebene der SSIM-Pyramide (t) verschwindet, während im Gegensatz zur MSE-Pyramide (p) die Bereiche, in denen der Fehler großflächig schwankt, länger erhalten bleiben.

Aussagekraft der Fehlerwerte

Während man beim Zählen der Pixel und beim MSE noch fragwürdige Aussagen treffen könnte, wie „Das Bild ist zu 75 % richtig.“, geht dies bei den anderen Verfahren nicht mehr. Hier lassen sich lediglich noch Aussagen treffen, wie „Bild A ist *bezgl. des benutzten Verfahrens* besser als Bild B.“, wenn man beide mit dem korrekten Bild verglichen hat. Selbst ob ein Bild gut oder schlecht bzw. besser oder schlechter als ein anderes ist, kann nicht entschieden werden, da dies stets eine subjektive Entscheidung des Betrachters ist. Eine Kombination aus Bildpyramide und SSIM liefert jedoch in der Praxis brauchbare Werte für das Multi-Algorithmen-Rendering. Eine ausführlichere Analyse der unterschiedlichen Verfahren wurde im Rahmen einer Bachelorarbeit von Fei Teng [Ten11] durchgeführt.

2.6 Sampling

Sampling bezeichnet die Abtastung einer Funktion an stichprobenartig gewählten Positionen – hier die Laufzeit und die Bildqualität verschiedener Algorithmen auf unterschiedlichen Regionen in einer 3-D-Szene. Die Art und Weise in der die Positionen gewählt werden (regelmäßig/zufällig) ist hierbei von entscheidender Bedeutung. Wählt man die Positionen regelmäßig, so ergibt sich ein dreidimensionales Gitter. Dies bietet den Vorteil, dass die Abdeckung des 3-D-Raums mit einer solchen Stichprobe gleichmäßig und die maximale Distanz eines beliebigen Punktes zum nächstgelegenen Samplepunkt minimal ist. Der Nachteil an dieser Vorgehensweise ist, dass die Abtastung einer periodischen Funktion – also einer regelmäßigen Szene – bei einem ungünstig gewählten Abstand der Positionen keine Aussagekraft mehr hat. Ein Beispiel hierfür ist ein Schachbrett, bei dem man bei einem ungünstig gewählten Abstand nur die weißen Felder trifft. Dies hat zur Folge, dass man sehr viele (je kleiner die Periode, desto mehr) Stichproben ziehen muss, um sicherzustellen, dass die Abtastung eine Aussagekraft hat.



(a) Regelmäßig (b) Zufällig, gleichverteilt (c) Modifizierter Dart-Throwing Algorithmus

Abbildung 2.2: Zweidimensionales Beispiel für unterschiedliche Verteilungen von jeweils ca. 1000 Sample-Positionen.

Alternativ kann man die Positionen für die Stichproben auch zufällig gleichverteilt wählen. Dies behebt die vorgenannten Probleme, hat jedoch den Nachteil, dass die Abtastung nicht mehr so gleichmäßig ist und man daher generell eine größere Menge an Stichproben braucht. Diese Vorgehensweise bietet jedoch noch einen weiteren Vorteil: Sie ist iterativ, das heißt, man kann zu einer beliebigen Menge an Stichproben stets noch eine weitere Probe hinzufügen, ohne die globale Verteilung der Stichproben zu beeinflussen. Bei einem Gitter muss man die Anzahl der Stichproben (im 3-D-Fall) hingegen verachtfachen, da die Verteilung sonst nicht mehr gleichmäßig ist.

Abbildung 2.2 zeigt ein Beispiel für beide Vorgehensweisen. Bei der zufälligen Verteilung ist deutlich sichtbar, dass einige Punkte sehr nah beieinander liegen und andererseits aber auch große Bereiche existieren, in denen gar keine Punkte liegen. Das dritte Bild zeigt eine Verteilung, welche die Vorteile der beiden vorgenannten Verfahren vereint. Diese Verteilung wurde mit Hilfe eines modifizierten Dart-Throwing-Algorithmus [Mit91] erzeugt. Hierbei werden statt einer zufälligen Position gleich mehrere (im Beispiel 100) erzeugt. Von diesen wird diejenige benutzt, die zu allen vorherigen Stichproben den größten Abstand hat. Die restlichen Positionen werden verworfen. Dadurch werden Positionen, die in einem bisher nur schwach abgetasteten Bereich liegen, bevorzugt und man erhält so insgesamt eine gleichmäßigere, jedoch nicht regelmäßige Abtastung.

3 Multi-Algorithmen-Rendering

Die generelle Idee des Multi-Algorithmen-Renderings besteht darin, eine Szene in Regionen geeigneter Größe aufzuteilen, welche gleichmäßig strukturiert (homogen) sind. Diese Regionen werden dann mit den jeweils am besten geeigneten Algorithmen dargestellt.

Um eine effiziente Auswahl der Algorithmen zur Laufzeit zu ermöglichen, werden vom Multi-Algorithmen-Rendering zunächst zwei Vorverarbeitungsschritte durchgeführt. Im ersten Schritt wird die Szene in homogene Bereiche geeigneter Größe aufgeteilt. Dazu selektiert ein *erfahrener* Benutzer wiederholt Kriterien, aufgrund derer dann die Szene unterteilt wird (Abschnitt 3.4.1). Dazu selektiert der Benutzer noch eine sinnvolle, möglichst kleine Teilmenge der Algorithmen die in der Szene benutzt werden sollen. Danach wird im zweiten Schritt stichprobenartig für alle Kombinationen von Algorithmen und Regionen an einigen Positionen die Laufzeit und der Bildfehler gemessen (Abschnitt 3.4.2) und das Ergebnis gespeichert.

Diese so ermittelten Werte werden dann zur Laufzeit genutzt um eine Zuordnung von Algorithmen zu Regionen zu finden, welche insgesamt ein gutes Bild liefert. Dazu werden die Stichproben aus der Vorverarbeitung für die aktuelle Position des Betrachters interpoliert (Abschnitt 3.5.1). Die Zuordnung von Algorithmen zu Regionen wird getroffen, indem auf Basis dieser Interpolation ein lineares Programm gelöst wird, welches den Gesamtfehler minimiert (Abschnitt 3.5.2). Eine der Nebenbedingungen des LPs sorgt hierbei dafür, dass die Summe der einzelnen Laufzeiten einen vorgegebenen Sollwert nicht überschreitet. Da die Laufzeit-Messwerte aus der Vorverarbeitung aus technischen Gründen fehlerbehaftet sind, wird das LP in einen Regelkreis eingebettet, welcher den vorgegebenen Sollwert für die Darstellungsdauer nachjustiert (Abschnitt 3.5.3). Dadurch wird eine leicht um den Sollwert schwankende Laufzeit erzielt.

Eine Beispielzuordnung von Algorithmen und Regionen ist in Abbildung 3.1 zu sehen. Hier werden vier verschiedene Algorithmen benutzt, um die einzelnen Regionen der Szene darzustellen.

In diesem Kapitel folgt zunächst die Definition von Homogen und Heterogen (3.1) sowie eine Erläuterung praktischer Kriterien hierfür (3.2). Danach werden die aktuell für das Multi-Algorithmen-Rendering einsetzbaren Verfahren erläutert (3.3). Nach der Beschreibung der Vorverarbeitung (3.4) und des Laufzeitalgorithmus (3.5) wird noch auf eine Umkehrung des Verfahrens – zur Optimierung der Laufzeit bei vorgegebenem Bildfehler (3.6) – sowie auf zusätzliche Einsatzbereiche des Verfahrens eingegangen (3.7).

3.1 Definition: heterogen & homogen

In dieser Arbeit ist die Definition des Begriffs heterogen variabel. Sie hängt von den vom Multi-Algorithmen-Rendering benutzten Algorithmen ab. Heterogen bedeutet, dass mindestens ein Kriterium, wie z. B. die Dichteverteilung der Polygone, welches die Laufzeit und die Bildqualität mindestens eines der benutzten Algorithmen beeinflusst, in einem

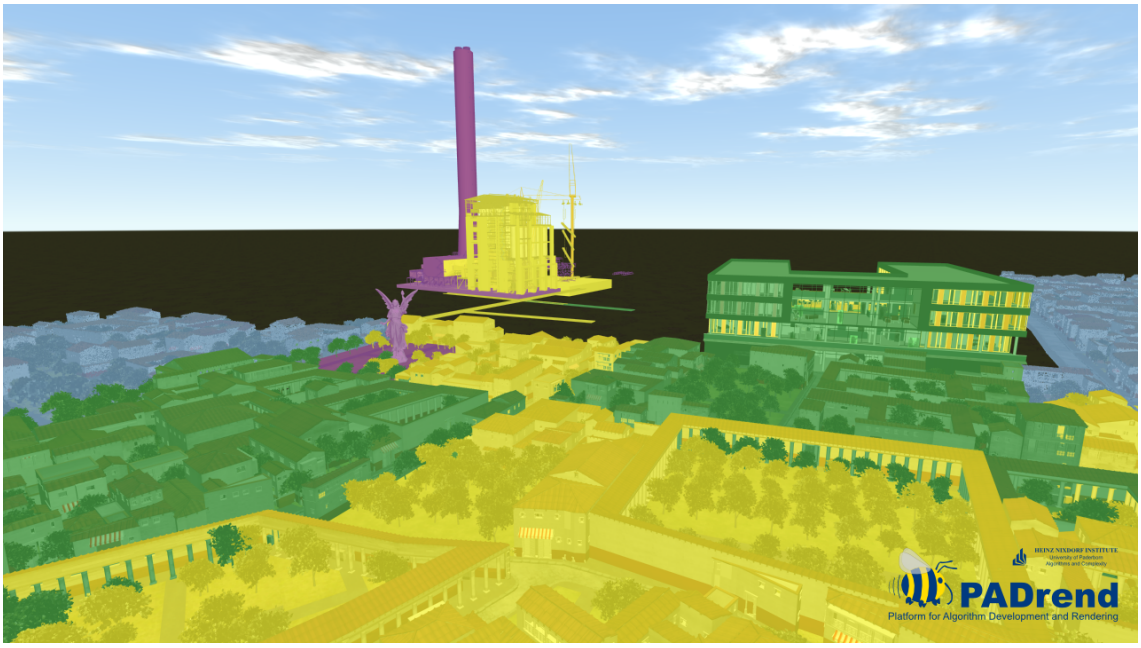


Abbildung 3.1: Beispiel für eine Zuordnung von Algorithmen zu Bereichen der Szene: gelb für CHC++, grün für SVS, blau für Blue-Surfels, magenta für LoD

Bereich stark schwankt. Homogen hingegen bedeutet, dass alle Kriterien nur geringfügig schwanken.

Das heißt, wenn man zur Menge der benutzten Algorithmen einen neuen hinzufügt, dann kann ein Bereich der Szene, der zuvor als homogen galt, nachher als heterogen gelten. Dies passiert genau dann, wenn innerhalb des Bereiches ein Kriterium schwankt, welches die Laufzeit oder Bildqualität der hinzugefügten Methode beeinflusst, jedoch für die bereits vorher benutzten Algorithmen irrelevant war.

3.2 Kriterien für Homogenität

Im Laufe dieser Arbeit hat sich herauskristallisiert, dass Laufzeit und Bildqualität der meisten – zumindest aller benutzten – Verfahren hauptsächlich von nur wenigen Kriterien abhängen:

- Die Dichteverteilung der Polygone in einem Bereich bestimmt maßgeblich, wie viel Verdeckung in einem Bereich besteht. Wenn man von realistischen Eingaben aus Oberflächenmodellen ausgeht und konstruierte Fälle, wie n Polygone die sich in einem Punkt schneiden und dabei alle sichtbar sind, ausschließt, dann müssen in einem beschränkten Bereich, in dem sich sehr viele oder sehr große Polygone bzw. Objekte befinden auch viele davon verdeckt sein. Ein Schwanken der Dichteverteilung ist somit ein guter Indikator für den Nutzen von Occlusion-Culling-Verfahren.

- Die Zahl an Objekten bzw. Polygonen pro Objekt ist maßgeblich für die Einsetzbarkeit von approximativen Darstellungsalgorithmen. Bei Objekten mit sehr vielen Polygonen benötigt man Verfahren die in der Lage sind diese zu simplifizieren, wie

z. B. Meshreduktionsverfahren. Ist hingegen die Anzahl Objekte der Flaschenhals, so werden Verfahren benötigt, die in der Lage sind viele Objekte zusammenzufassen und mit nur wenigen Funktionsaufrufen darzustellen.

Dazu kommen noch zwei Kriterien für einen geeigneten Bereich, welche nichts mit Heterogenität zu tun haben: Die Größe und Komplexität eines Bereiches. In Bereichen, die zu groß sind, kann das Multi-Algorithmen-Rendering den Algorithmus nicht wechseln. Folglich muss dieser Bereich, wenn der Bildfehler bei einer Approximation nahe dem Betrachter sehr groß wird, komplett korrekt dargestellt werden, obwohl dies für die entfernteren Bereiche eigentlich nicht notwendig ist. Bei Bereichen mit zu geringer Komplexität kann es hingegen vorkommen, dass die Kosten, die das Wechseln des Darstellungsverfahrens erzeugt, den Gewinn bei der Darstellung übersteigen.

3.3 Beim Multi-Algorithmen-Rendering eingesetzte Verfahren

Für das Multi-Algorithmen-Rendering können prinzipiell alle Verfahren benutzt werden, solange diese kein Budget verwenden, welches extern eingestellt werden muss. Dies wäre beispielsweise eine Zeitvorgabe für die Darstellung einer Szene/Region. Bei Verfahren die – wie das Multi-Algorithmen-Rendering selbst – ein solches Budget verwenden, müsste dieses kontinuierlich angepasst werden, indem das Budget des Multi-Algorithmen-Renderings auf die einzelnen Regionen verteilt wird, für die solche Algorithmen benutzt werden. Eine Heuristik zu diesem Zweck ist zwar generell denkbar, wurde jedoch im Rahmen dieser Arbeit nicht umgesetzt. Verfahren, die ein Budget benutzen, welches sich automatisch justiert, wie etwa eine fixe Anzahl an Polygonen pro Pixel, die die Szene auf dem Bildschirm einnimmt, stellen hingegen kein Problem dar.

Ebenfalls Probleme bereiten approximative Verfahren, welche bei der Darstellung Löcher im Bild hinterlassen. Falls solche Verfahren im Nahbereich eingesetzt werden und dahinter andere, die Occlusion-Culling benutzen, dann hat dies zur Folge, dass im hinteren Bereich zusätzliche Objekte als sichtbar klassifiziert werden. Dadurch kann sich die Laufzeit für den hinteren Bereich erhöhen. In der Regel sollte ein solcher Fall aber nicht auftreten, da Verfahren, welche (große) Löcher im Bild verursachen, aufgrund der schlechten Bildqualität nicht im Nahbereich eingesetzt werden.

Bei der Auswahl der Verfahren ist darauf zu achten, dass sowohl Verfahren verwendet werden, welche ein qualitativ gutes Bild erzeugen, als auch Verfahren die in der Lage sind jegliche Bereiche sehr schnell – mit eventuell großem Fehler – darzustellen. Solange jeweils mindestens ein Vertreter dieser Gruppen vorhanden ist, funktioniert das Multi-Algorithmen-Rendering. Zusätzliche Algorithmen können jedoch das Ergebnis deutlich positiv beeinflussen.

Die derzeitige Menge an Algorithmen, die für das Multi-Algorithmen-Rendering benutzt werden kann und deren Einsatzbereich, sieht wie folgt aus:

z-Buffer: Der Algorithmus stellt *alle* Objekte dar, die im Frustum liegen und erzeugt ein korrektes Bild. Er ist geeignet für Bereiche, in denen nahezu alle Objekte sichtbar sind.

Coherent-Hierarchical-Culling++: Der Algorithmus testet, ob Objekte bzw. Teilbäume sichtbar sind, und lässt unsichtbare Teile der Szene bei der Darstellung aus. Der

Algorithmus erzeugt ein fehlerfreies Bild und ist schnell, wenn nur wenig Geometrie sichtbar ist.

Approximate-Coherent-Hierarchical-Culling++: Eine zweite Variante des gleichen Algorithmus, der allerdings Objekte, die nur zu einem kleinen Teil sichtbar sind, nicht darstellt. Der Algorithmus wurde modifiziert, sodass er statt Objekte komplett auszulassen, zumindest deren Bounding-Box darstellt. Deshalb entstehen durch das Auslassen der Objekte keine Löcher, durch die wiederum andere Objekte sichtbar werden können.

Spherical-Visibility-Sampling: Der Algorithmus stellt alle *vermutlich* sichtbaren Objekte dar und erzeugt nur einen sehr kleinen Bildfehler. Die Laufzeit ist ähnlich der des CHC++, bei geeigneten Situation deutlich besser, bei ungeeigneten allerdings auch schlechter. SVS berechnet die Sichtbarkeit während einer Vorverarbeitung, weshalb zur Laufzeit kein zusätzlicher Aufwand durch Sichtbarkeitstests entsteht.

Discrete-Level-of-Detail: Der Algorithmus reduziert die Komplexität einzelner Objekte auf ein Maß, welches der projizierten Größe des Objekts angemessen ist. Der erzeugte Bildfehler ist mit den verwendeten Einstellungen sehr klein. Den größten Nutzen hat der Algorithmus bei Objekten, welche wesentlich detaillierter modelliert sind, als für ihre projizierte Größe notwendig ist.

Progressive-Blue-Surfels: Der Algorithmus ersetzt Teilbäume des Szenengraphen durch Surfels, sofern für die aktuelle projizierte Größe des Teilbaums genug Surfels vorhanden sind. Ansonsten wird der Baum weiter traversiert bis ein Blatt erreicht wird. Ist auch dieses noch zu groß, so wird die Geometrie gezeichnet. Der erzeugte Bildfehler ist für entfernte Bereiche akzeptabel. Der Algorithmus ist sehr schnell für entfernte oder komplexe Bereiche.

Forced-Surfels: Eine Variante des Progressive-Blue-Surfels-Algorithmus, welche bei Erreichen eines Blattknotens immer Surfels darstellt, auch dann, wenn die Anzahl nicht ausreichend ist. Der Algorithmus ist immer sehr schnell, allerdings ist die Bildqualität für nahe Bereiche mit großen, einfachen Objekten schlecht.

Color-Cubes: Der Algorithmus ersetzt Teilbäume des Szenengraphen durch farbige Würfel, sofern die projizierte Größe des Teilgraphen kleiner einer Konstante ist. Ansonsten wird der Baum weiter traversiert bis ein Blatt erreicht wird. Ist auch dessen projizierte Größe noch zu groß, so wird die Geometrie gezeichnet. Der erzeugte Bildfehler ist für entfernte Bereiche akzeptabel. Der Algorithmus ist dort schnell.

Der Algorithmus wurde an den heutigen Stand der Technik angepasst. Er arbeitet nicht wie im Original auf Dreiecken die in einem Octree organisiert sind, sondern auf Objekten in einer beliebigen Bounding-Volume Hierarchie. Dadurch sind die einzelnen Knoten der Hierarchie keine disjunkten Würfel mehr sondern Quader die sich eventuell auch schneiden können. Daher werden die Farbquader für die inneren Knoten ebenfalls durch Rendern der Farbquader der Kindknoten erzeugt.

Forced-Cubes: Eine Variante des Color-Cubes-Algorithmus, welche bei Erreichen eines Blattknotens immer einen Würfel darstellt, auch dann, wenn die projizierte Größe zu groß ist. Erzeugt im Nahbereich einen großen Bildfehler.

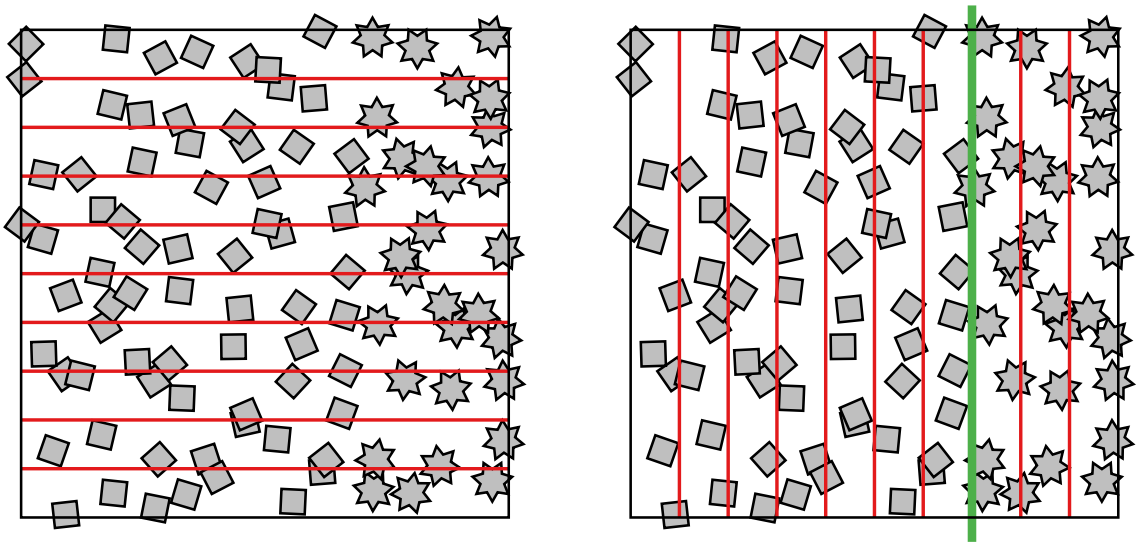


Abbildung 3.2: Die beiden Bilder zeigen eine Draufsicht auf eine 2-D-Szene bestehend aus einfachen (Quadrate) und komplexen (Sterne) Objekten. Dargestellt ist die Unterteilung von Regionen anhand der Dichteverteilung der Polygone. Verschiedene Schnitte (rot, bis zu 1024 pro Achse) werden berechnet. Der Schnitt (grün), der die Varianz in den entstehenden zwei Regionen minimiert, wird gewählt.

3.4 Analyse und Aufbereitung der Szene

Um eine effiziente Auswahl der Algorithmen zur Laufzeit zu ermöglichen, werden vom Multi-Algorithmen-Rendering zunächst zwei Vorverarbeitungsschritte durchgeführt. Im ersten Schritt wird die Szene in homogene Bereiche geeigneter Größe aufgeteilt. Dazu selektiert ein *erfahrener* Benutzer wiederholt Kriterien, aufgrund derer dann die Szene unterteilt wird (Abschnitt 3.4.1). Danach wird Stichprobenartig für alle Kombinationen von Algorithmen und Regionen an einigen Positionen die Laufzeit und die Bildqualität gemessen (Abschnitt 3.4.2). Dazu selektiert der erfahrene Benutzer noch eine sinnvolle, möglichst kleine Teilmenge der Algorithmen die in der Szene benutzt werden sollen. Diese so ermittelten Werte werden dann zur Laufzeit genutzt um eine Zuordnung von Algorithmen zu Regionen zu finden, welche insgesamt ein gutes Bild liefert.

3.4.1 Aufteilung der Szene in Regionen

Die Aufteilung der Szene in Regionen geschieht halbautomatisch. Ausgangspunkt hierfür ist eine Region, die die gesamte Szene umfasst. Diese wird dann durch eine Folge selektierter Kriterien immer weiter unterteilt, bis der Benutzer der Meinung ist, dass die Aufteilung nur noch homogene Regionen enthält. Die Aufteilung ist somit abhängig von den Algorithmen die im Weiteren benutzt werden sollen. Die Software reagiert während dieses Schrittes innerhalb weniger Sekunden. Dadurch ist die Aufteilung einer Szene innerhalb einer Minute möglich, vorausgesetzt der Benutzer weiß genau, wie er die Szene aufteilen möchte. Folgende, bereits genannte Kriterien stehen zur Verfügung:

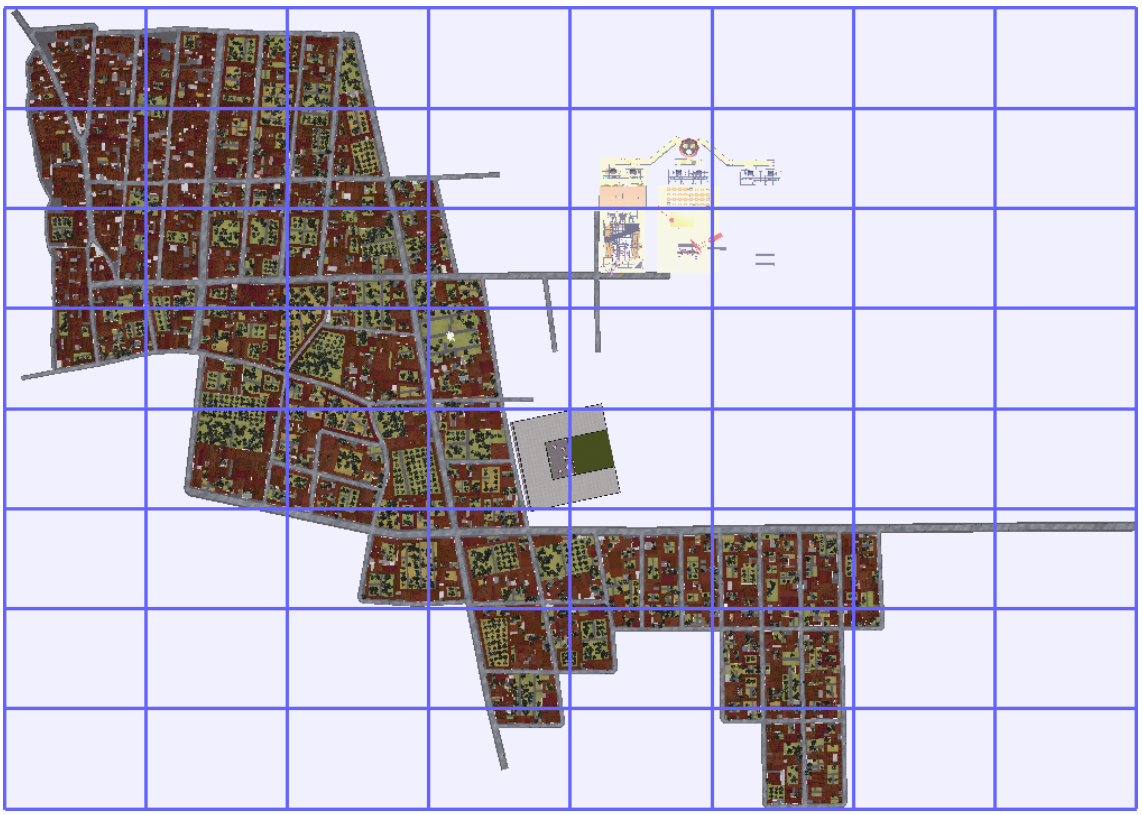
Dichteverteilung der Primitive in einer Region: Die Dichteverteilung der Polygone in einer Region. Wird dieses Kriterium ausgewählt, so wird für jede Region berechnet, wie sich die Dichteverteilung der Polygone innerhalb der Region verhält. Dazu wird jede Region entlang aller drei Koordinatenachsen in diskrete Scheiben geschnitten und für jede Scheibe die Polygondichte berechnet. Dann wird für jede Achse der optimale Schnittpunkt berechnet, so dass in den dabei entstehenden Subregionen die (diskrete) Varianz der Dichteverteilung möglichst gering ist. Abbildung 3.2 zeigt ein Beispiel hierfür. Der optimale Schnitt ergibt sich dann als Minimum der Werte aller drei Achsen. Dieser Wert wird für alle Regionen berechnet und gespeichert. Die Region mit dem besten Wert wird unterteilt. Der Benutzer muss also lediglich solange Regionen per Klick teilen, bis das gewünschte Ergebnis erreicht ist. Die Auswahl der zu unterteilenden Region geschieht automatisch.

Größe einer Region: Beim Teilen von Regionen anhand ihrer Größe werden diese zunächst nur entlang der Achsen unterteilt, für die gilt, dass die Region entlang dieser Achse um mehr als Faktor $\sqrt{2}$ länger ist als ihre kürzeste Seite. Sobald das Verhältnis von kürzester und längster Seite höchstens $\sqrt{2}$ ist, wird an allen Achsen unterteilt. Dadurch werden möglichst würfelförmig Regionen erzeugt. Der Benutzer hat bei dieser Vorgehensweise zwei Möglichkeiten: Er kann sowohl eine minimale Tiefe für den entstehenden Baum vorgeben als auch eine maximale Ausdehnung für die Regionen. Die Unterteilung selbst geschieht automatisch.

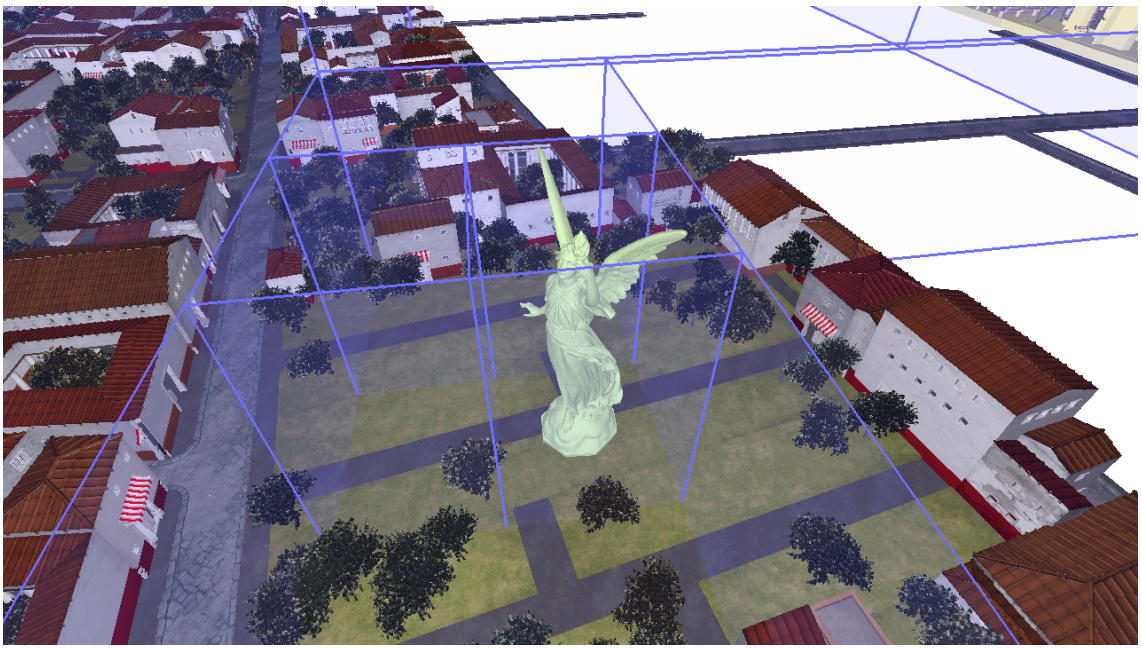
Komplexität einer Region: Hier kann der Benutzer eine maximale Anzahl an Primitiven pro Region vorgeben. Die Regionen werden solange unterteilt, bis alle Regionen höchstens so viele Primitive wie vorgegeben beinhalten. Auch hierbei werden möglichst würfelförmige Regionen erzeugt.

Diese Kriterien können in beliebiger Reihenfolge aufeinander aufbauend genutzt werden. Die Entscheidung, welche Kriterien in welcher Reihenfolge wie oft angewendet werden, trifft der Benutzer. Abbildung 3.3 zeigt eine Beispielaufteilung einer Szene. Hierbei wurde zunächst die Szene regelmäßig unterteilt, bis die Kantenlängen der Regionen kleiner als 250 m waren. Danach wurde die Szene noch einige Male anhand der Dichteverteilung der Polygone gesplittet, um das komplexe Modell der Statue vom Rest der Stadt, sowie das komplexe Rohrsystem des Kraftwerks von seiner Hülle zu separieren. Auf eine weitere Unterteilung der Szene wurde verzichtet, um die Anzahl der Regionen gering zu halten und somit die Größe des linearen Programms zur Laufzeit (siehe Kap. 3.5.2) zu reduzieren. Stattdessen könnte man auch zuerst anhand der Polygondichteverteilung unterteilen und erst danach anhand der Größe der Regionen. Dies führte allerdings auch zu mehr Regionen und wäre somit ungünstig.

Für jede der erstellten Regionen wird eine Bounding-Volume-Hierarchie aufgebaut, um die Teilszene zu speichern. Objekte, die über die Grenzen von Regionen hinausgehen, gehören zu der Region, in der ihr Mittelpunkt liegt. Hierfür wurde ein Loose-Octree gewählt. An dieser Stelle ist die Verwendung einer Loose-Datenstruktur wichtig, da ohne sie viele der Objekte in der Szene entweder sehr weit oben im Baum lägen oder mehrfach gespeichert werden müssten. Ob ein Octree, k-D Baum oder ähnliches gewählt wird, ist zweitrangig. Die optimale Wahl hat nur wenig Einfluss auf die Laufzeit von Algorithmen, hängt von der jeweiligen Szene ab und kann deshalb hier nicht getroffen werden.



(a) Regelmäßige Aufteilung der Szene.



(b) Aufteilung einzelner Bereiche nach Polygondichte.

Abbildung 3.3: Unterteilung einer Szene in Regionen: zunächst anhand der Größe der Regionen, danach weiter anhand der Dichteverteilung der Primitive in den Regionen. Die Teilungsebenen umschließen die Statue sehr eng.

3.4.2 Datenerhebung durch Sampling

Im zweiten Schritt der Vorverarbeitung werden alle Kombinationen aus Algorithmen und Regionen ausgemessen. Dabei werden zum einen die Laufzeit und zum anderen der Bildfehler gemessen. Da diese Werte abhängig von der Position des Betrachters sind, werden sie nicht nur einmalig, sondern stichprobenartig an vielen Positionen innerhalb eines durch den Benutzer definierten Bereichs getestet. Innerhalb dieses Bereiches kann dann später zur Laufzeit das Multi-Algorithmen-Rendering benutzt werden.

Abbildung 3.4 veranschaulicht den Ablauf beim Ziehen einer Stichprobe. Die Positionen für die Stichproben werden mit einem modifizierten Dart-Throwing-Algorithmus innerhalb dieses Bereiches generiert. Für jede Position wird dann ein Test gestartet, welcher zunächst die Regionen aufsteigend nach Abstand zu dieser Position sortiert. In dieser Reihenfolge werden dann die Regionen, bei denen ein Verdeckungstest ihrer Boundingbox das Resultat „sichtbar“ liefert, jeweils mit allen Algorithmen getestet.

Dazu wird die Kamera an die Position der Stichprobe gesetzt und auf den Mittelpunkt der aktuell zu testenden Region gerichtet. Sollte die Region nicht ganz ins Frustum passen, so wird dies, nahe der aktuellen Position, auch später zur Laufzeit der Fall sein. Durch die Ausrichtung auf den Mittelpunkt wird der schlimmste Fall angenommen, da hier ein möglichst großer Teil der Region im Frustum ist. Dadurch werden Laufzeit und Bildfehler der Messung nach oben abgeschätzt. Ein Fall wie in Abbildung 3.5 tritt nicht auf, da die Regionen homogen sind. Nur wenn die Position sehr nahe am Zentrum der Region liegt, kann es sein, dass bei flachen Regionen fehlerhafte Messergebnisse entstehen (siehe hierzu auch Kap. 3.5.1), weil die Kamera beispielsweise in einem Stadtteil in den Himmel gerichtet wird.

Als erstes wird nun ein Referenzbild mit dem CHC++ erzeugt. Mit diesem werden die Bildvergleiche für die approximativen Algorithmen durchgeführt.

Danach werden alle Algorithmen jeweils dreimal getestet. Als Laufzeit wird das Minimum der drei ermittelten Werte gespeichert. Der erste Durchlauf liefert häufig Werte, die deutlich zu groß sind, da hier oftmals ein Overhead (z. B. durch das Hochladen von Daten in den Grafikspeicher) entsteht. Der dritte Durchlauf sorgt dafür, dass auch Algorithmen, welche zeitliche Kohärenz ausnutzen, korrekt gemessen werden. Mit dem letzten Bild wird dann ein Bildvergleich (SSIM-Pyramide) mit dem Referenzbild durchgeführt, um den Bildfehler zu bestimmen. Nachdem alle Regionen abgearbeitet sind, werden die ermittelten Laufzeit- und Bildfehlerwerte in einem Punkt-Octree gespeichert. Für unsichtbare Regionen werden beide Werte auf Null gesetzt.

Für jeden Test eines Algorithmus müssen alle vorher getesteten Regionen dargestellt werden, da diese die aktuelle Region (teilweise) verdecken könnten. Dadurch können sowohl die Laufzeit eines Algorithmus als auch der Bildfehler beeinflusst werden. So müsste etwa die vorderste Region für alle nachfolgenden Regionen und jeweils alle Algorithmen erneut dargestellt werden, also insgesamt $\Theta(|Regionen|^2 \cdot |Algorithmen|^2)$ Mal. Diese Kosten wären um Größenordnungen höher als die des eigentlichen Tests. Deshalb wird jede Region, nachdem sie getestet wurde, mit dem CHC++ Algorithmus auf die Innenseiten eines Würfels gerendert, der die Kamera umschließt. Dieser Würfel wird dann anstatt der Regionen mit einem speziellen Shader inklusive der Tiefeninformationen in ein leeres Bild gerendert.

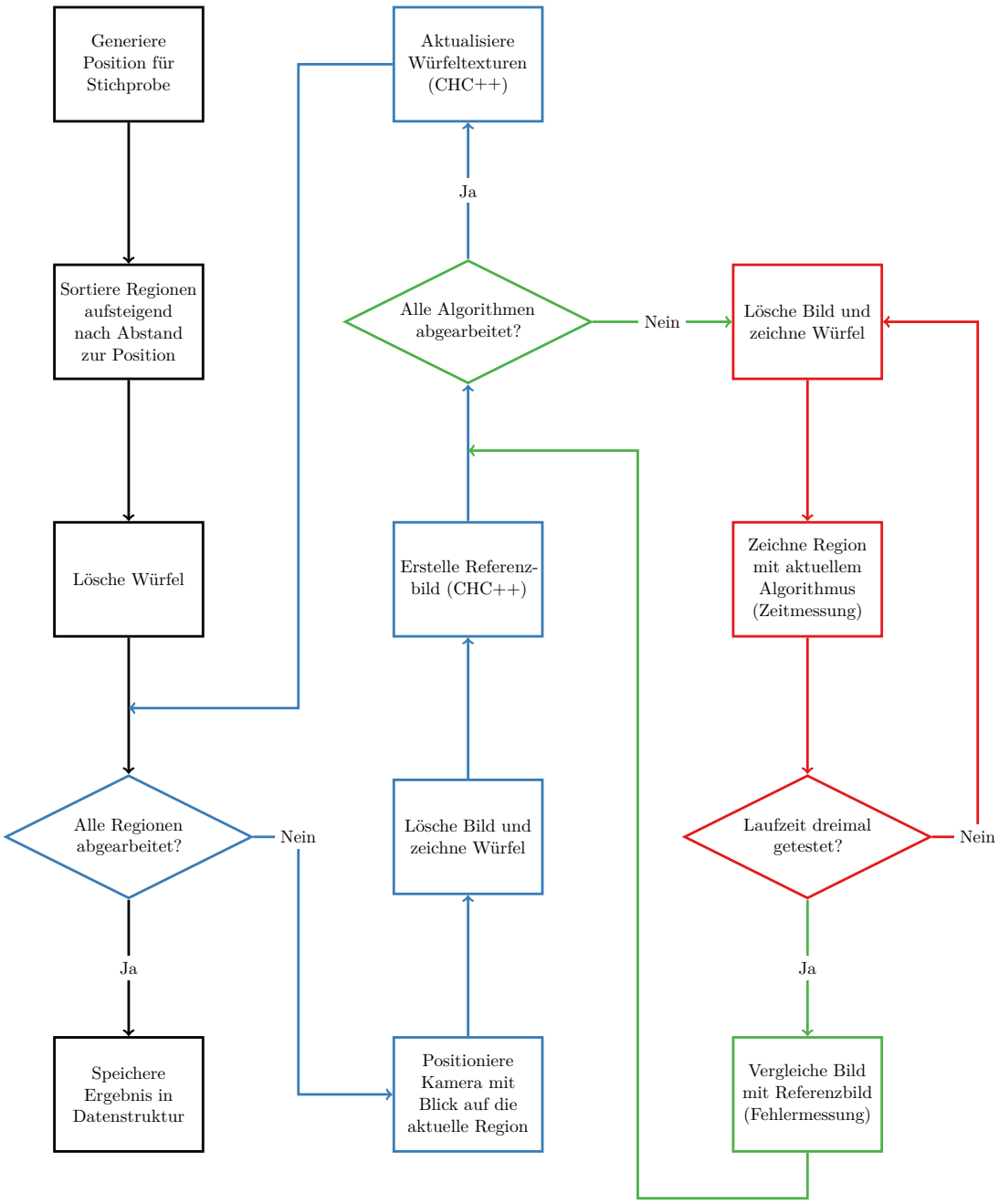


Abbildung 3.4: Ablauf des Ziehens einer Stichprobe: Ermittlung von Laufzeit und Bildfehler für alle Algorithmen für alle Regionen. Der „Würfel“ ist ein um die Kamera positionierter, von innen texturierter Einheitswürfel, welcher die korrekten Farb- und Tiefeninformationen aller vorhergehenden Regionen enthält.

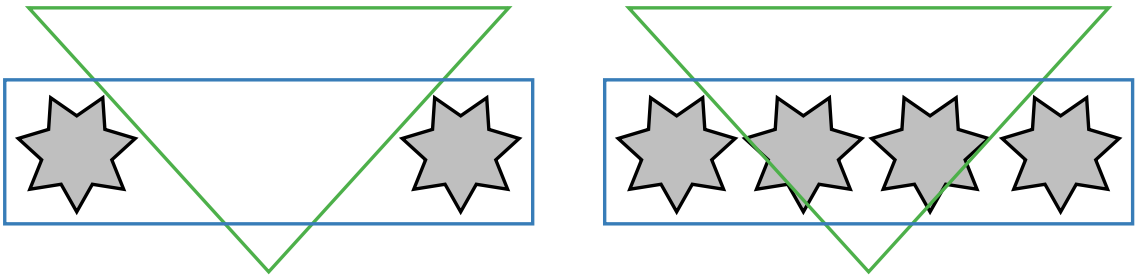


Abbildung 3.5: Ausrichtung der Kamera auf den Mittelpunkt einer Region (blau): Heterogene Regionen (links) würden zu fehlerhaften Messungen führen, wenn sie nicht ins Frustum (grün) passen und so beim Test eine nicht repräsentative, hier leere Teilmenge der Objekte dargestellt wird. Bei homogenen Regionen (rechts) kann dieser Fall nicht auftreten.

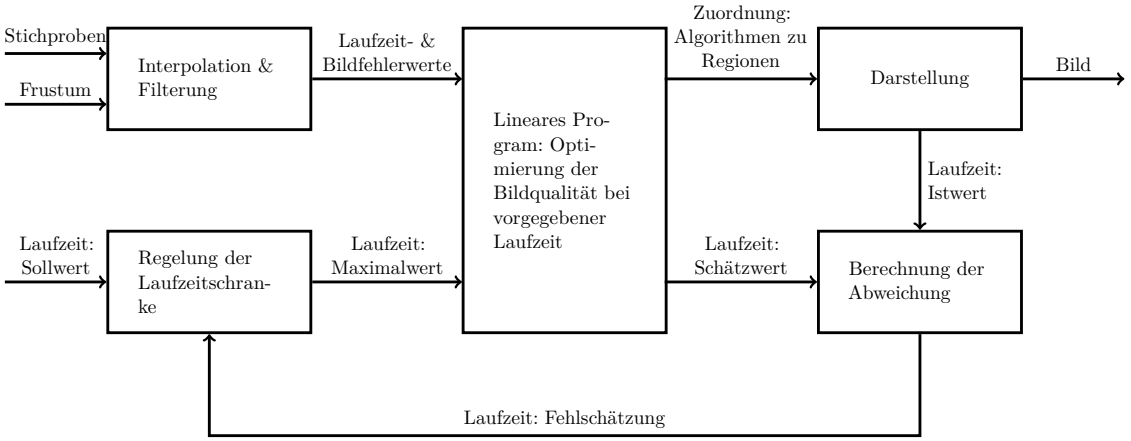


Abbildung 3.6: Schematische Darstellung der Optimierung der Bildqualität bei vorgegebener Bildrate durch Einbettung eines linearen Programms in einen Regelkreis.

3.5 Der Renderingalgorithmus: Darstellung der Szene im Walkthrough

Zur Laufzeit werden die in der Vorverarbeitung erhobenen Daten dazu genutzt, mit Hilfe eines linearen Programms eine optimale Zuordnung von Algorithmen zu Regionen zu finden. Dabei wird der Bildfehler bei einer vorgegebenen Laufzeit minimiert. Komplett unsichtbare Regionen werden hierbei, wie in der Vorverarbeitung, durch einen übergeordneten CHC++-Algorithmus ausgefiltert. Zur Stabilisierung des Verfahrens (z. B. Ausgleich von Messfehlern) wird das lineare Programm in einen Regelkreis eingebettet. Abbildung 3.6 stellt den kompletten Regelkreis grafisch dar. Die einzelnen Schritte werden in den folgenden Unterkapiteln näher erläutert. Dazu bezeichnet:

- R: die Menge der Regionen.
- A: die Menge der Algorithmen.

$e_{0,0}$	$e_{0,1}$	$e_{0,2}$	$e_{1,0}$	$e_{1,1}$	$e_{1,2}$	$e_{2,0}$	$e_{2,1}$	$e_{2,2}$	$e_{3,0}$	$e_{3,1}$	$e_{3,2}$	
1	1	1	0	0	0	0	0	0	0	0	0	$= 1$
0	0	0	1	1	1	0	0	0	0	0	0	$= 1$
0	0	0	0	0	0	1	1	1	0	0	0	$= 1$
0	0	0	0	0	0	0	0	0	1	1	1	$= 1$
$t_{0,0}$	$t_{0,1}$	$t_{0,2}$	$t_{1,0}$	$t_{1,1}$	$t_{1,2}$	$t_{2,0}$	$t_{2,1}$	$t_{2,2}$	$t_{3,0}$	$t_{3,1}$	$t_{3,2}$	$\leq T_{max}$

Abbildung 3.7: Beispielinstanz des linearen Programms zur Zuordnung von Algorithmen zu Regionen mit vier Regionen und drei Algorithmen.

- $e_{r,a}$: Der bei der Darstellung von Region r mit Algorithmus a entstehende Bildfehler.
- $t_{r,a}$: Die von Algorithmus a benötigte Zeit um Region r darzustellen.
- T_{max}: Die vorgegebene Zeit zur Darstellung aller Regionen.
- $x_{r,a}$: Entscheidungsvariablen die angeben ob Algorithmus a für Region r benutzt wird.

Die Eingabe des LPs setzt sich aus den gemessenen Fehlerwerten ($e_{r,a}$), den gemessenen Laufzeiten ($t_{r,a}$) und der Vorgabe für die Gesamtlaufzeit (T_{max}) zusammen. Die Ausgabe des LPs sind die Entscheidungsvariablen ($x_{r,a}$), deren Wert genau dann wahr ist, wenn Algorithmus a für Region r benutzt werden soll. Das lineare Programm sieht wie folgt aus:

$$\text{Minimiere} \sum_{r \in R} \sum_{a \in A} e_{r,a} \cdot x_{r,a} \tag{3.1}$$

$$\text{u.d.N.} \sum_{a \in A} x_{r,a} = 1 \qquad \forall r \in R \tag{3.2}$$

$$\sum_{r \in R} \sum_{a \in A} t_{r,a} \cdot x_{r,a} \leq T_{\max} \tag{3.3}$$

$$x_{r,a} \in \{0, 1\} \tag{3.4}$$

Zeile 3.1 des LPs ist die Zielfunktion: Minimierung des Gesamtfehlers. Zeile 3.2 und 3.4 sorgen dafür, dass für jede Region genau ein Algorithmus ausgewählt wird. Zeile 3.3 sorgt für die Einhaltung Schranke für die Gesamtlaufzeit. Die Beispielinstanz des LPs in Abbildung 3.5 lässt eine sehr einfache Struktur des LPs erkennen. Bis auf die erste Zeile (Zielfunktion) und die letzte Zeile (Laufzeitschranke) enthält das LP nur binäre Werte.

3.5.1 Interpolation der Messwerte aus der Vorverarbeitung

Vermutlich wird sich der Benutzer zur Laufzeit nie exakt an einer Position aufhalten, an welcher in der Vorverarbeitung eine Stichprobe gezogen wurde. Daher stellt sich die Frage, welche Stichprobe als Eingabe für das lineare Programm benutzt wird.

Die einfachste Möglichkeit ist die Benutzung der Stichprobe, welche den kleinsten Abstand zur aktuellen Position des Betrachters hat. Probleme entstehen, wenn die Stichprobe direkt hinter einem Objekt liegt, der Benutzer aktuell jedoch daneben steht. Für das Beispiel in Abbildung 3.8 bedeutet dies, dass Stichprobe s_2 benutzt wird. Wenn der Benutzer nun nach rechts schaut, dann sind die Sichtbarkeitsverhältnisse genau entgegengesetzt zu denen

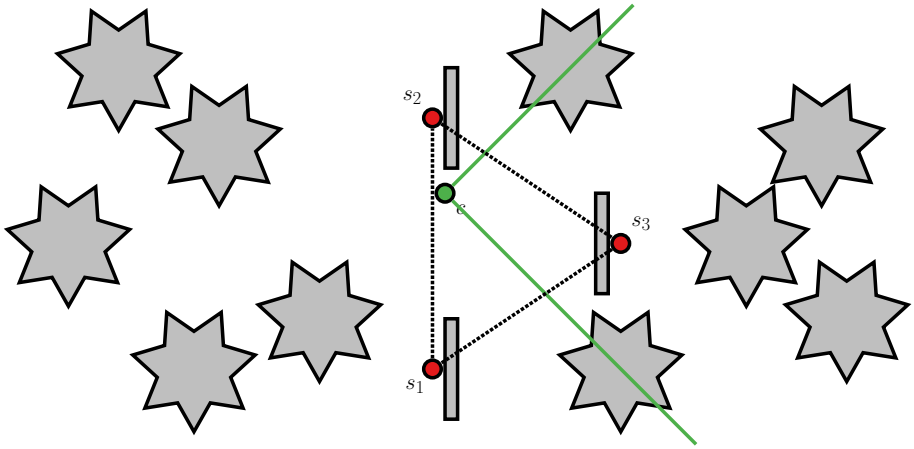


Abbildung 3.8: Konstruiertes schlechtes Beispiel für die Lage der zur Laufzeit benutzten Stichproben s_1, s_2, s_3 (rot). Für die aktuelle Position des Betrachters c (grün, mit Frustum) sind die Stichproben ungeeignet, da an diesen Positionen, im Gegensatz zur aktuellen Position, große Teile der Szene (grau) verdeckt sind.

aus der Vorverarbeitung. Somit ist an dieser Stelle keine sinnvolle Ausgabe des LPs zu erwarten, da dessen Eingabe bereits falsch ist.

Eine weitere Möglichkeit besteht darin, eine Interpolation der umliegenden Stichproben als Eingabe für das LP zu benutzen. Interpoliert man die Stichproben gewichtete mit ihrem Abstand zur aktuellen Position (z. B. mit Hilfe baryzentrischer Koordinaten [Möb27]), so ändert dies nichts am Problem. Dies liegt daran, dass Sichtbarkeit keine stetige Funktion ist, sondern insbesondere nahe an Objekten große Sprünge aufweist. Das bedeutet, dass die nächste Stichprobe nicht zwingend besser sein muss als die zweit- oder drittnächste (in Abbildung 3.8 wäre s_3 die beste Wahl). Daher sollte eine Interpolationsmethode gewählt werden, welche den Abstand der Stichproben zur aktuellen Position ignoriert. Im Laufe der Arbeit hat sich herausgestellt, dass in der Praxis das Maximum der vier nächstgelegenen Stichproben zu guten Ergebnissen führt. Diese Worst-Case-Annahme hat zur Folge, dass durch das LP schnellere Algorithmen bevorzugt werden. Außerdem werden Bereiche, welche in der Vorverarbeitung nur knapp verdeckt waren, so behandelt, als ob sie sichtbar wären. Dadurch werden bereits Bereiche mit in die Berechnung einbezogen, welche durch sehr geringe Bewegungen des Benutzers sichtbar werden.

Die benutzten, umliegenden Stichproben sollten hierbei möglichst gleichmäßig um den Betrachter herum verteilt sein. Dies könnte man etwa erreichen indem man eine Delaunay-Triangulierung [Del34; Sam05] über die Positionen der Stichproben berechnet. Die benutzten Stichproben wären dann die Eckpunkte des Tetraeders, in dem sich der Betrachter aktuell aufhält. Aufgrund der sehr gleichmäßigen Verteilung der Stichproben reicht es aber auch aus, die nächstliegenden vier Stichproben zu benutzen. Diese sind mit großer Wahrscheinlichkeit identisch mit denen, welche die Delaunay-Triangulierung liefern würde.

3.5.2 Auswahl der Algorithmen

Theoretisch funktioniert das Verfahren, so wie es bis hier beschrieben wurde. Praktisch ergeben sich jedoch noch einige Probleme bedingt dadurch, dass Grafikkarte und CPU parallel arbeiten. Daher kann bei der Darstellung die CPU bereits Berechnungen für folgende Regionen durchführen, während die Grafikkarte noch mit der Darstellung der vorherigen Regionen beschäftigt ist. Des Weiteren arbeitet die Grafikkarte wie eine Pipeline (Fließband), das heißt, es befinden sich immer mehrere Primitive gleichzeitig in Bearbeitung. Während der Vorverarbeitung werden jedoch für die Laufzeitmessungen stets einzelne Regionen dargestellt. Dazu wird eine Stoppuhr gestartet, die Region gerendert und dann die Stoppuhr abgefragt. Die Darstellung der Region beinhaltet dabei explizit auch das Warten auf das fertige Bild, also ein Leerlaufen der Pipeline. Wenn man nicht auf das fertige Bild warten würde, so wären die Messwerte absolut unbrauchbar und lägen überwiegend bei weniger als einer Millisekunde. Dieser Effekt entsteht durch die Parallelität von CPU und Grafikkarte. Das Anstoßen der Darstellung auf der CPU geht sehr schnell, während die Grafikkarte weitaus länger arbeiten muss. Wenn man bei der Laufzeitmessung allerdings auf das fertige Bild wartet, so ist das Leerlaufen der Pipeline in dieser Messung enthalten. Zur Laufzeit wird jedoch die gesamte Szene dargestellt, ohne nach jeder Region die Pipeline leerlaufen zu lassen. Bei Regionen, deren Darstellung lange dauert, fällt dieser konstante Overhead nicht ins Gewicht. Bei Regionen, deren Darstellung sehr schnell geht hingegen, wie z. B. bei projiziert kleinen Regionen die stark approximiert werden, kann dieser Mehraufwand größer sein als die eigentliche Darstellung. Dies hat zur Folge, dass die Summe der in der Vorverarbeitung gemessenen Laufzeiten für die einzelnen Regionen bei der Darstellung der gesamten Szene um ein Vielfaches von der wirklich benötigten Zeit abweichen kann.

3.5.3 Regelung der Gesamtdauer der Darstellung

Zum Ausgleich dieses (leider schwankenden) Fehlers wurde das LP in einen Regelkreis eingebettet. Dieser misst die wirklich benötigte Laufzeit T_{real} zur Darstellung der Szene und passt die Laufzeitschranke T_{max} (Eingabe für das LP) kontinuierlich in einem beschränkten Bereich an. Als Schranken dienen hierbei die Summen der Laufzeiten der jeweils schnellsten bzw. langsamsten Algorithmen pro Region. Dadurch ist sichergestellt, dass das LP zum einen immer lösbar ist und zum anderen der Wert von T_{max} nicht unnötig groß wird. Im Folgenden ist

$$T_{estimate} = \sum_{r \in R} \sum_{a \in A} x_{r,a} \cdot t_{r,a}$$

die Summe der Laufzeitschätzungen aus der Vorverarbeitung, welche in der Lösung des LPs benutzt werden, also eine Schätzung für die Gesamtlaufzeit zur Darstellung der Szene. T_{user} ist der vom Benutzer eingestellte Zielwert für die Gesamtlaufzeit. Es wurden drei unterschiedliche Regelkreise umgesetzt:

- Die erste Variante benutzt einen Korrekturfaktor T_{adjust}, welcher in jedem Durchlauf angepasst wird.

$$T_{adjust} = \begin{cases} T_{adjust} \cdot 2/3 & \text{falls } T_{real} > T_{user} \cdot 3/2 \\ T_{adjust} \cdot 11/10 & \text{falls } T_{real} < T_{user} \cdot 1/2 \\ T_{adjust} & \text{sonst} \end{cases}$$

Der Regelkreis versucht also T_{real} im Bereich $T_{user} \pm 50\,\%$ zu stabilisieren. Als Eingabe für das LP wird dann

$$T_{max} = T_{user} \cdot T_{adjust}$$

benutzt. Dass der Korrekturfaktor deutlich stärker erhöht als erniedrigt wird ist hier wichtig, da bei einer zu geringen Bildrate dieser Schritt wesentlich seltener ausgeführt wird als bei einer zu hohen Bildrate.

Dieser Regelkreis hat ein Gedächtnis: Die Information über die Entscheidung im letzten Durchlauf ist im aktuellen Durchlauf verfügbar. Dadurch ändert sich der Wert von T_{max} nur langsam und springt nicht (um mehr als den Faktor $2/3$). Dies ist von Vorteil, solange sich die Sichtbarkeit (und somit die Laufzeiten) innerhalb der Szene nur langsam ändert. Ändert sich die Sichtbarkeit jedoch schlagartig, so dauert es einige Durchläufe bis der Regelkreis die entsprechende Anpassung durchgeführt hat. Außerdem hat er vier Parameter, die bestimmen, wann der Faktor um wie viel erhöht bzw. erniedrigt wird und deren Einstellung schwierig ist. Des Weiteren neigt der Regelkreis zum Schwingen an Positionen, an denen die Laufzeiten der einzelnen Algorithmen einer Region sehr stark unterschiedlich sind, so dass T_{real} nie im angestrebten Bereich liegt, sondern abwechselnd mehrmals darunter und einmal darüber.

- Die zweite Variante berechnet das Verhältnis zwischen $T_{estimate}$ und T_{real}, also den relativen Fehler, um den die Schätzung von der Realität abweicht. Die Eingabe für das LP ergibt sich dann als:

  $$T_{max} = T_{user} \cdot \frac{T_{estimate}}{T_{real}}$$

  Dieser Regelkreis hat kein Gedächtnis, er beachtet ausschließlich den aktuell gemessenen Wert und reagiert somit direkt auf Veränderungen. Bei sehr geringen Laufzeitwerten treten Probleme auf, da hier Messungenauigkeiten bereits zu sehr starken Ausschlägen führen, was zu einer Oszillation des Wertes von T_{max} führen kann.

- Die dritte Variante berechnet die Differenz zwischen $T_{estimate}$ und T_{real}, also den absoluten Fehler, um den die Schätzung von der Realität abweicht. Die Eingabe für das LP ergibt sich dann als:

  $$T_{max} = T_{user} + (T_{estimate} - T_{real})$$

  Dieser Regelkreis hat ebenfalls kein Gedächtnis, er beachtet ausschließlich den aktuell gemessenen Wert und reagiert somit direkt auf Veränderungen. Probleme treten auf, falls die Laufzeitwerte aus der Vorverarbeitung eine Unterschätzung sind. Dann könnte der Wert von T_{max} negativ werden. Dies hätte zur Folge, dass aufgrund der Beschränkung das LP zwar lösbar bliebe, jedoch immer die schnellsten Algorithmen pro Region benutzt würden. Da die Werte aus der Vorverarbeitung jedoch Überschätzungen sind, tritt dieser Fall nicht ein.

Spätestens jetzt ist klar, dass das LP für jedes Bild erneut gelöst werden muss. Das bedeutet, dass die Berechnung des LPs genau wie die Darstellung der Szene in Echtzeit, also auch innerhalb der vom Benutzer eingestellte Zeit, erfolgen muss.

Nimmt man in Kauf, dass die Lösung des LPs mit einem Bild Versatz berechnet wird, so kann das LP parallel gelöst werden. Dadurch steht zur Darstellung und zur Lösung des LPs *jeweils* die eingestellte Zeit zur Verfügung.

Zusätzlich wird das LP, unabhängig von der benutzten Regelvariante, stets auf die Regionen beschränkt, welche sich im Frustum befinden und somit potenziell sichtbar sind. Dies beschleunigt zum einen die Lösung des LPs und führt zum anderen auch zu qualitativ besseren Ergebnissen.

Ein Nachteil der Umsetzung als Regelkreis besteht darin, das eventuell bei jedem Bild eine andere Zuordnung von Algorithmen zu Regionen getroffen wird. Dies wird vom Benutzer als Flackern im Bild wahrgenommen.

3.6 Optimierung der Laufzeit bei vorgegebener Bildqualität

Prinzipiell kann man, statt bei gegebener Laufzeit die Bildqualität zu optimieren, den Vorgang auch umkehren und bei gegebener Bildqualität die Laufzeit optimieren. Insbesondere bei der Erstellung von Videos und Filmen könnte dies eventuell Anwendung finden, um Zeit und damit Kosten zu sparen. Hierbei entstehen in der Praxis jedoch neue Probleme:

- Eine sinnvolle Laufzeit (z. B. 50 ms) vorzugeben ist einfach, einen sinnvollen maximalen Bildfehler vorzugeben ist hingegen sehr schwer. Dies liegt daran, dass die verwendeten Abstandsmaße zur Berechnung der Bildqualität nur *meistens* und *ungefähr* der menschlichen Wahrnehmung entsprechen. Verschärft wird das Problem dadurch, dass die Abstandsmaße nicht linear sind. Alle Fehlerwerte liegen zwar im Intervall von null bis eins, aber wenn man versuchen würde, Begriffe wie „schlecht“ und „gut“ in dieses Intervall einzuordnen, dann sähe das ähnlich aus wie: sehr gut(0,01); gut(0,05); mittel(0,1); akzeptabel(0,2); inverses Bild(0,5); kein Zusammenhang erkennbar(1,0). Dadurch ist es für einen Menschen schwierig, einen sinnvollen Maximalwert vorzugeben.
- Die Einbettung dieses LPs in einen Regelkreis ist nicht möglich, da man dazu die Bildqualität des dargestellten Bildes berechnen muss. Da man dafür jedoch ein korrektes Bild bräuchte, könnte man dieses direkt zur Darstellung nutzen.
- Für die Darstellung von Szenen für Filme sind vermutlich völlig andere Algorithmen nötig, als in dieser Arbeit betrachtet wurden. Hier ist vermutlich, wie bei Spielen, die Anzahl der darzustellenden Primitive wesentlich geringer, währen die Darstellung jedes einzelnen Primitives aufgrund von aufwendigen Effekten deutlich aufwendiger ist.

3.7 Einsatzbereiche des Verfahrens

Neben der reinen Darstellung von Szenen ist das Multi-Algorithmen-Rendering auch noch für weitere Zwecke einsetzbar:

Evaluierung neuer Renderingalgorithmen Bei der Evaluierung von neu entwickelten Darstellungsalgorithmen muss stets untersucht werden, unter welchen Umständen diese besonders gut oder schlecht funktionieren. Dies kann z. B. mit Hilfe des Multi-Algorithmen-Renderings geschehen. Dazu muss lediglich das neue Verfahren als einer

der benutzten Algorithmen einbezogen werden. Dann kann über die Einfärbung der einzelnen Algorithmen sehr leicht ermittelt werden, für welche Bereiche und bei welchem Abstand das Multi-Algorithmen-Rendering den Algorithmus einsetzt. Somit kann sehr leicht bestimmt werden, unter welchen Umständen das neue Verfahren den Vergleichsalgorithmen überlegen ist.

Design von Szenen Bei der Erstellung von Szenen kann das Multi-Algorithmen-Rendering ebenfalls als Hilfsmittel eingesetzt werden. Hier ist die Benutzung von stark approximierenden Methoden durch das Multi-Algorithmen-Rendering ein Indiz dafür, dass Bereiche der Szene ungünstig gestaltet sind. Dies wäre auch bei der Gestaltung von Welten für Computerspiele hilfreich. Hier allerdings vermutlich mit einer völlig anderen Menge von Algorithmen.

Feste Auswahl von Algorithmen Das Multi-Algorithmen-Rendering löst das Problem, welche Region mit welchem Algorithmus dargestellt wird, dynamisch zur Laufzeit. Es könnte jedoch auch als Tool benutzt werden, um eine pro Position feste Zuordnung von Algorithmen manuell zu treffen. Sofern der dazu erforderliche manuelle Aufwand zu rechtfertigen ist, könnte man so die Vorteile des Multi-Algorithmen-Renderings ohne den störenden Flackereffekt erhalten.

4 Implementierung

Dieses Kapitel beschreibt die Details bezüglich der Umsetzung und Implementierung des zuvor beschriebenen Multi-Algorithmen-Rendering. Es gliedert sich in eine Beschreibung des verwendeten Frameworks (4.1) und die verfahrensspezifischen Aspekte(4.2).

4.1 PADrend

Als Framework sowohl für die Implementierung als auch für die Evaluierung des Verfahrens wurde PADrend [EJP11] eingesetzt. PADrend steht für (P)latform for (A)lgorithm (D)evelopment and (rend)ering. PADrend wurde parallel zu dieser Arbeit in Zusammenarbeit mit Claudius Jähn und später auch Benjamin Eikel entwickelt. Des Weiteren waren mehrere Studentische Hilfskräfte daran beteiligt. Auch Ergebnisse mehrerer Bachelor- Diplom- und Masterarbeiten sind in die Software eingeflossen. Bedingt dadurch, dass die Software von mehreren Benutzern benutzt und implementiert wird, läuft sie deutlich robuster, weist einen deutlich besseren Code auf und ist auch durch andere Benutzer als die Entwickler selbst bedienbar.

PADrend besteht aus einem in C++ implementierten Kern von Basisfunktionalitäten und laufzeitkritischen Programmteilen, auf dem ein Script-System aufsetzt. Dies erlaubt es Entwicklern neue Ideen sehr schnell auszuprobieren, da die meisten benötigten Datenstrukturen und Algorithmen bereits in PADrend verfügbar sind. Lediglich die neuen Aspekte müssen programmiert werden. Insbesondere das Script-System für die nicht laufzeitkritischen High-Level-Funktionalitäten ist dabei sehr hilfreich. Dieses ermöglicht es dem Entwickler, Programmteile, ohne Neustart der Software, zur Laufzeit auszutauschen.

Bei der Evaluierung von Datenstrukturen und Algorithmen ist PADrend der sonst üblichen Spezialimplementierung ebenfalls deutlich überlegen. Zum einen sind Vergleichsverfahren häufig bereits vorhanden und müssen nicht extra implementiert werden. Zum anderen basieren alle Verfahren auf der gleichen Implementierung der zugrunde liegenden Funktionalitäten. Dadurch ist sichergestellt, dass bei einem Vergleich von Algorithmen nicht der Unterschied in der gleichen, jedoch unterschiedlich effizient implementierten Datenstruktur gemessen wird.

Die Kernstruktur von PADrend ist der Szenengraph. Dieser wurde als Baum realisiert und besteht aus Knoten, welche eine Bounding-Volume-Hierarchie bilden. Knoten können mit sogenannten States annotiert werden. Jeder State wird vor der Darstellung seines Knotens aktiviert und danach wieder deaktiviert. Die Aktivierung eines States kann dabei ganz einfache Dinge bewirken, wie beispielsweise eine Änderung der zur Darstellung benutzten Farbe oder dem Einschalten einer Lichtquelle. Es können aber auch komplexe Operationen durchgeführt werden, wie beispielsweise ein komplettes Austauschen des zur Darstellung benutzten Algorithmus für den aktuellen Teilbaum. Des Weiteren können Knoten auch noch mit beliebigen Attributen versehen werden. Dies sind Daten, die von den Algorithmen benötigt werden. Dazu gehören unter anderem Informationen darüber, in welchem Frame der Knoten zuletzt auf Sichtbarkeit getestet wurde, oder Ersatzrepräsentationen wir Surfels.

4.2 Multi-Algorithmen-Rendering

Die Implementierung des Multi-Algorithmen-Renderings gliedert sich in zwei Teile: zum einen die Vorverarbeitung und zum anderen der Algorithmus, der während des Walkthroughs ausgeführt wird.

4.2.1 Vorverarbeitung des Multi-Algorithmen-Renderings

Die Aufteilung der Szene in Regionen wurde teilweise in C++ implementiert, um Antwortzeiten auf Benutzerinteraktionen im Sekundenbereich zu erzielen. Das Verfahren zur Erhebung der Stichproben hingegen wurde nahezu komplett gescriptet, da die hier benötigte Zeit im Bereich von Zehntelsekunden liegt. Dies ist im Vergleich zu der Zeit, die für die Darstellung der Szene und die Bildvergleiche benötigt wird, vernachlässigbar. Die Verfahren zur Aufteilung der Szene in Regionen und zur Berechnung des Bildfehlers wurden so umgesetzt, dass die Methoden leicht austauschbar sind. Dies heißt beispielsweise für den Bildfehler, dass jedes Verfahren einsetzbar ist, welches zu zwei gegebenen Bildern deren Abstand liefert. Dadurch kann das Multi-Algorithmen-Rendering leicht an die Anforderungen der jeweiligen Anwendung angepasst werden.

4.2.2 Multi-Algorithmen-Rendering während des Walkthroughs

Zur Erklärung des Laufzeitalgorithmus ist in Abbildung 4.1 ein Beispiel-Szenengraph mit nur zwei Regionen abgebildet. Unterhalb der Wurzel gibt es zunächst für jede Region einen Multi-Algorithmen-Rendering-Knoten. An diesen Knoten hängt jeweils der Darstellungsalgorithmus (als State), der zur Darstellung dieser Region benutzt werden soll. Der Algorithmus wird somit automatisch vor der Darstellung der Region aktiviert und hinterher wieder deaktiviert.

Am Wurzelknoten der Szene hängt ein weiterer State. Dieser enthält den eigentlichen Laufzeitalgorithmus: Hier wird beim Aktivieren des States berechnet, welche Multi-Algorithmen-Rendering-Knoten im Frustum liegen, und die zur Position passenden Stichproben gewählt/interpoliert. Die Berechnungen für den Regelkreis werden durchgeführt und die Berechnung der Lösung des LPs gestartet. Aufgrund des Ergebnisses des LPs aus dem vorherigen Frame werden an den Multi-Algorithmen-Rendering-Knoten die Algorithmen ausgetauscht. Zuletzt wird eine Stoppuhr gestartet. Diese wird dann beim Deaktivieren des States – nach der Darstellung der Szene – abgelesen. Danach wird noch auf die Lösung des LPs gewartet.

Zusätzlich hängt am Wurzel-Knoten noch ein CHC++-Algorithmus, der dafür sorgt, dass Regionen, welche komplett verdeckt sind, nicht dargestellt werden. Dieser ist für das Multi-Algorithmen-Rendering nicht zwingend erforderlich, sorgt jedoch dafür, dass in Bereichen mit hoher Verdeckung und bei korrekter Darstellung keine unnötige Zeit verschwendet wird. Dies hat zur Folge, dass dort die Laufzeit weit unter dem Sollwert bleibt, auch wenn das LP den verdeckten Regionen einen möglicherweise nicht optimalen Algorithmus zuordnet, weil die Laufzeitschranke auf jeden Fall unterschritten wird.

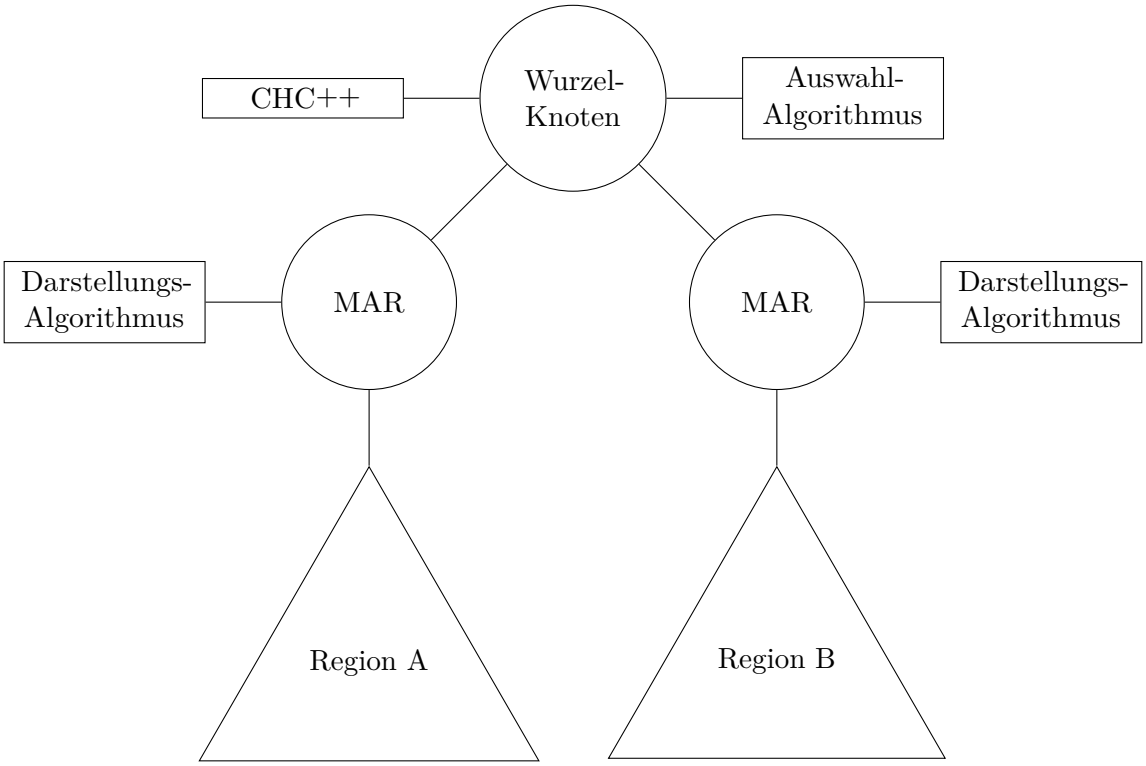


Abbildung 4.1: Darstellung eines Beispiel-Szenengraphen für das Multi-Algorithmen-Rendering mit nur zwei Regionen. Knoten sind als Kreise dargestellt, States als Rechtecke. Dreiecke repräsentieren die Teilgraphen, in denen die Regionen gespeichert sind.

5 Evaluierung

Die generelle Idee des Multi-Algorithmen-Renderings beruht auf den folgenden zwei Annahmen:

1. Es ist möglich, für jeden Renderingalgorithmus eine heterogene, realistische Szene zu konstruieren, die diesem Algorithmus Probleme bereitet. Das heißt, dass zumindest bei einigen Betrachterpositionen entweder die Darstellung zu lange dauert oder der Bildfehler zu groß wird.

2. Bei jeder homogenen, realistischen Szene gibt es für jede Betrachterposition einen Algorithmus, der die Szene ohne diese Probleme darstellen kann. Das heißt, dass sich sowohl Laufzeit als auch Bildfehler in einem akzeptablen Bereich befinden.

Die folgende Evaluierung soll nun zeigen, dass eine Kombination von Algorithmen, welche jeweils nur für die für sie geeigneten homogenen Regionen einer Szene eingesetzt werden, in der Lage ist, heterogene Szenen darzustellen, die ansonsten nicht mit ausreichender Bildqualität in Echtzeit darstellbar wären.

Außerdem wird gezeigt, dass es mit Hilfe des Multi-Algorithmen-Renderings möglich ist, diese Zuordnung von Algorithmen zu geeigneten Regionen der Szene vollautomatisch zu treffen. Dazu werden verschiedene Aspekte des Multi-Algorithmen-Renderings untersucht, deren Kombination das gewünschte Ergebnis zeigt.

Darüber hinaus werden die Aspekte des Multi-Algorithmen-Renderings untersucht (Dauer der Vorverarbeitung, Anzahl benötigter Stichproben etc.), die für eine praktische Verwendung des Verfahrens entscheidend sind.

Es folgt zunächst eine Übersicht über die untersuchten Fragestellungen (5.1). Danach wird die Messumgebung (Hardware, Software, Szene, Kamerapfad etc.) beschrieben (5.2). Darauf folgt die Beschreibung der einzelnen Messungen, inklusive Interpretation (5.3 und 5.4). Abschließend werden die Ergebnisse der Messungen zusammengefasst (5.6).

5.1 Übersicht über die untersuchten Fragestellungen

Die Evaluierung in dieser Arbeit teilt sich in zwei Bereiche auf: Im ersten Bereich (5.3) wird die Vorverarbeitung des Multi-Algorithmen-Renderings bezüglich Speicherverbrauch (5.3.1) und Laufzeit (5.3.2) untersucht. Letztere wurde auch noch aufgeschlüsselt, um zu sehen für welche Teilschritte der Vorverarbeitung die Zeit benötigt wird. Die konkret untersuchten Fragestellungen sind:

- Wie lange dauert das Ziehen einer Stichprobe?
- Wie verteilt sich die Laufzeit beim Ziehen einer Stichprobe?
  - Wie viel Zeit benötigt die Darstellung mit den einzelnen Algorithmen?
  - Wie viel Zeit benötigen die Bildvergleiche?

  - Wie lange dauert das Erzeugen der Referenzbilder?
- Wie viel Speicher verbrauchen die benutzten Verfahren?
- Wie viel Speicher verbrauchen die erhobenen Stichproben?

Im zweiten Bereich der Evaluierung (5.4) wird das Laufzeitverhalten des Multi-Algorithmen-Renderings untersucht. Bei der Untersuchung des Verhaltens des Multi-Algorithmen-Renderings während eines Walkthroughs gibt es zwei interessante Fragestellungen: zum einen die Laufzeit des Verfahrens und zum anderen die Qualität der erzeugten Bilder. Diese beiden Aspekte werden stets parallel untersucht, da eine Laufzeitmessung bei einem approximativen Verfahren keine Aussagekraft hat, wenn nicht auch gleichzeitig der Fehler des Verfahrens betrachtet wird. Die genauen Fragestellungen hierzu lauten:

- Wie lange dauert die Berechnung eines Bildes (5.4.1)?
  - Wie lange benötigen die verwendeten Algorithmen, wenn mit ihnen die komplette Szene dargestellt wird?
  - Wie stark weicht die Laufzeit des Multi-Algorithmen-Renderings vom vorgegebenen Sollwert ab?
  - Wie stark weicht die real gemessene Laufzeit von den Werten aus der Vorverarbeitung ab (5.4.5)?
- Wie groß ist der Bildfehler (5.4.2)?
  - Wie groß ist der Bildfehler, den die benutzten Verfahren erzeugen, wenn mit ihnen die gesamte Szene dargestellt wird?
  - Wie groß ist der Bildfehler, der vom Multi-Algorithmen-Rendering erzeugt wird?
  - Wie stark weicht der real gemessene Bildfehler von den Werten aus der Vorverarbeitung ab (5.4.6)?

Zusätzlich hierzu wurden noch Messungen zu den folgenden Fragestellungen durchgeführt:

- Welchen Einfluss haben die unterschiedlichen Interpolations- und Regelungsmethoden die für das Multi-Algorithmen-Rendering umgesetzt wurden (5.4.3)?
- Wie häufig werden die einzelnen Algorithmen eingesetzt (5.4.7)?
  - Wie viele Pixel werden mit den Algorithmen dargestellt?
  - Wie viele Regionen werden mit den Algorithmen dargestellt?
  - Wie groß ist der Nutzen der Algorithmen, wie ändert sich das Ergebnis wenn bestimmte Algorithmen deaktiviert werden?
- Wie verhält sich das Multi-Algorithmen-Rendering abhängig von der Anzahl der in der Vorverarbeitung gezogenen Stichproben und wie viele Stichproben werden benötigt (Abschnitt 5.4.4)?
- Wie groß bzw. störend sind die bereits in Abschnitt 3.5.3 erwähnten Flacker-Effekte beim Multi-Algorithmen-Rendering (5.4.8)?
- Wie viel Zeit wird benötigt um das Optimierungsproblem zu lösen (5.4.9)?

Die Evaluierung des Verhaltens des Multi-Algorithmen-Renderings wird zunächst wie üblich anhand eines Kamerapfades durchgeführt. Zusätzlich wird danach noch eine Analyse des Verfahrens über die von Jähn u. a. [Jäh+13] beschriebenen Szenen-Eigenschafts-Funktionen durchgeführt. Dies dient dazu, die Ergebnisse der ersten Analyse zu stützen und zu zeigen, dass die Ergebnisse nicht nur für den speziell gewählten Kamerapfad gelten, sondern auch für die anderen Teile der Szene, welche durch den Pfad nicht abgedeckt sind. Bei dieser Analyse traten einige zusätzliche Effekte auf, die bei der Analyse mit Hilfe des Kamerapfades übersehen wurden.

5.2 Messumgebung

Im Folgenden wird das generelle Setup für die Messungen beschrieben. Abweichungen hiervon sind bei den einzelnen Messungen angegeben.

5.2.1 Verwendete Hard- und Software

Als Software kam PADrend (siehe Abschnitt 4.1) zum Einsatz. Zum Lösen des linearen Programms wurde die Bibliothek lp_solve[1] benutzt. Sowohl die Vorverarbeitung als auch die Laufzeittests wurden auf folgender Hardware durchgeführt:

CPU: Intel Core i7 960; 4 × 3,2 GHz

Hauptspeicher: 24 GiB

Grafikkarte: NVIDIA GeForce GTX 660

Betriebssystem: Linux

Hervorzuheben ist, dass der Hauptspeicher bei allen Tests ausreichend groß war. Es wurden keine Out-Of-Core Mechanismen benutzt.

5.2.2 Aufbau der Testszene

Die Testszene (Abbildung 5.1) ist eine Kombination von Szenen aus unterschiedlichen Quellen:

- Eine prozedural erzeugte Stadt[2] [Mül+06] bestehend aus Gebäuden und Vegetation (Pompeji, 153.848 Objekte, ca. 63 Mio. Primitive, mit Texturen).
- Ein Modell eines Kohlekraftwerks[3] (Power-Plant, 1181 Objekte, ca. 12,7 Mio. Primitive).
- Ein Gebäude aus einem Architekturprogramm[4] (Universität Paderborn, Gebäude ZM 1, 2417 Objekte, ca. 1,5 Mio. Primitive, mit Texturen).
- Ein durch Laserscans erzeugtes Modell einer Statue[5] (Lucy, ein Objekt, ca. 28 Mio. Primitive).

[1] lp_solve 5.5.2.0: `http://lpsolve.sourceforge.net/`
[2] City Engine: `http://www.esri.com/software/cityengine/resources/casestudies/`
[3] The Walkthru Project: `http://gamma.cs.unc.edu/POWERPLANT/`
[4] Design: Matern und Wäschle GbR, Architekten BDA: `http://www.maternwaeschle.de/`
[5] The Stanford 3D Scanning Repository: `http://graphics.stanford.edu/data/3Dscanrep/`

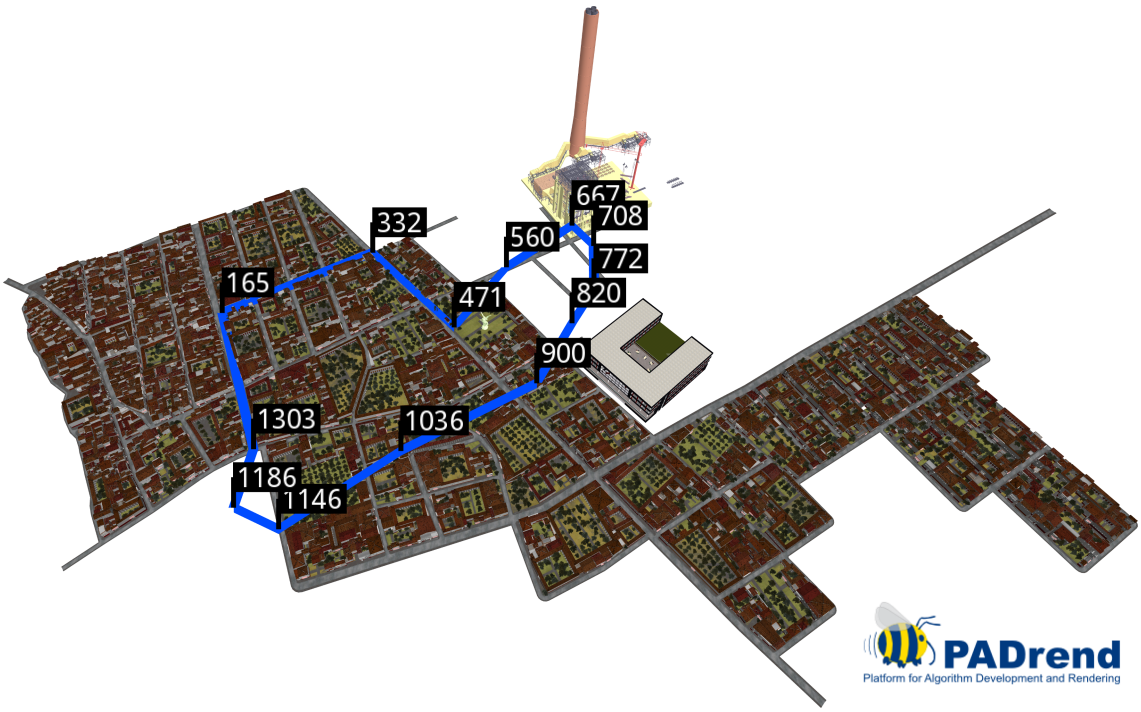


Abbildung 5.1: Darstellung der Szene und des zur Evaluierung benutzten Kamerapfades.

Die Szene besteht also sowohl aus hochkomplexen Einzelmodellen als auch aus einer großen Menge sehr einfacher Objekte. Die Objekte sind teilweise mit Texturen versehen.

Alle Komponenten der Szene wurden so belassen, wie sie sind; es wurden keine manuellen Schritte zur Optimierung der Eingabe durchgeführt. Dies bedeutet unter anderem, dass die Lucy mit ihren ca. 28 Mio. Polygonen nicht in mehrere Objekte unterteilt wurde, sowie dass die einzeln modellierten Bordsteine von Pompeji nicht zusammengefasst wurden. Unter diesen Voraussetzungen sind somit weder die Lucy noch Pompeji mit der verwendeten Hardware korrekt in Echtzeit darstellbar. Die anderen beiden Modelle sind jedes für sich problemlos darstellbar. Jedoch enthalten sie sehr große Objekte mit sehr wenig Primitiven, wie beispielsweise die Dachfläche des ZM 1 Modells und der Schornstein des Kraftwerks.

Man könnte die Eingabe auch manuell optimieren und/oder einen speziellen Darstellungsalgorithmus entwickeln. Damit wäre es durchaus denkbar diese Szene in Echtzeit darzustellen und dabei eine bessere Bildqualität zu erzielen als beim Multi-Algorithmen-Rendering. Dies würde allerdings einen sehr hohen Arbeitsaufwand erzeugen, wäre sehr unflexibel und würde Expertenwissen erfordern.

5.2.3 Wahl des Kamerapfades

Der für die folgenden Messungen benutzte Kamerapfad wird in Abbildung 5.1 gezeigt. Es ist ein geschlossener Pfad mit 1303 Messpunkten, bei dem Position 0 und 1303 identisch sind. Beginnend bei Punkt 0 führt er zunächst bis Punkt 471 durch die Straßen von Pompeji. Dabei ist ab Punkt 332 das Lucy-Modell im Sichtbereich. Zwischen Punkt 500 und 772 steigt der Pfad dann nach oben über die Szene. Danach verläuft der Pfad bis Punkt 1036 auf gleichbleibender Höhe, jedoch dreht sich die Blickrichtung der Kamera dabei auf die

Mitte der Szene. Bis Punkt 1186 steigt der Pfad dann wieder leicht an und die Kamera ist weiterhin mittig auf die Szene gerichtet. An Punkt 1186 ist dann der überwiegende Teil der Szene im Sichtbereich. Auf dem letzten Stück zurück zu Punkt 1303 sinkt der Pfad dann wieder ab und die Kamera dreht sich zurück nach vorn. Der Kamerapfad wurde so gewählt, dass alle Bereiche der Szene, die besonders gut oder schlecht zu handhaben sind, darauf vorkommen. Dies äußert sich wie folgt:

- Der Pfad führt durch die Straßen von Pompeji. Dort stellt die Darstellung der kompletten Szene mit dem CHC++ kein Problem dar. Somit kann hier eigentlich immer ein korrektes Bild erzeugt werden.
- Der Pfad führt direkt an der Lucy vorbei. Hier ist ein sehr komplexes Modell direkt vor der Kamera (projizierte Größe sehr groß). Lediglich der Discrete-Level-of-Detail- und der Forced-Surfels-Algorithmus sind in der Lage diese Situation in Echtzeit zu handhaben. Dabei erzeugt der Discrete-Level-of-Detail Algorithmus einen deutlich geringeren Bildfehler.
- Der Pfad führt durch die Straßen von Pompeji, während die Lucy im Frustum liegt und nicht komplett verdeckt ist. An dieser Stelle ist eine korrekte Darstellung aufgrund der sichtbaren Lucy nicht möglich. Approximative Darstellungsalgorithmen, die auf Basis von Objekten arbeiten, sind aufgrund der hohen Anzahl von Objekten im Frustum auch zu langsam. Stellt man jedoch, wie beim Multi-Algorithmen-Rendering, die Szene mit unterschiedlichen Algorithmen dar (Approximation für die Lucy und korrekte Darstellung für den Rest der Stadt), dann erhält man ein nahezu korrektes Bild in Echtzeit.
- Der Pfad führt über die Stadt, wobei große Teile der Stadt im Frustum liegen. Hier ist eine korrekte Darstellung der Szene nicht möglich, da die Anzahl sichtbarer Objekte zu groß ist.

5.2.4 Benutzte Algorithmen und deren Einstellung

Für das Multi-Algorithmen-Rendering selbst wurde zur Interpolation der Stichproben aus der Vorverarbeitung das Maximum der vier nächsten Stichproben gewählt. Als Regelkreis wurde die Variante benutzt, welche die Vorgabe für das lineare Programm mit Hilfe der absoluten Abweichung berechnet. Diese Kombination stellt sich im Laufe der Evaluierung als die beste heraus und wurde deshalb für alle übrigen Messungen benutzt. Der Bereich, der mit den Stichproben abgedeckt wurde, ist in Abbildung 5.2 dargestellt. Die bereits in den Abschnitten 2.4 und 3.3 verwendeten Algorithmen werden im Folgenden aufgelistet und ihre Parametrisierung (sofern von der Literatur abweichend) beschrieben:

- **Coherent-Hierarchical-Culling++**
- **z-Buffer**
- **Spherical-Visibility-Sampling**
- **Color-Cubes**: Bei einer projizierten Größe von weniger als 100 Pixeln wird approximiert.

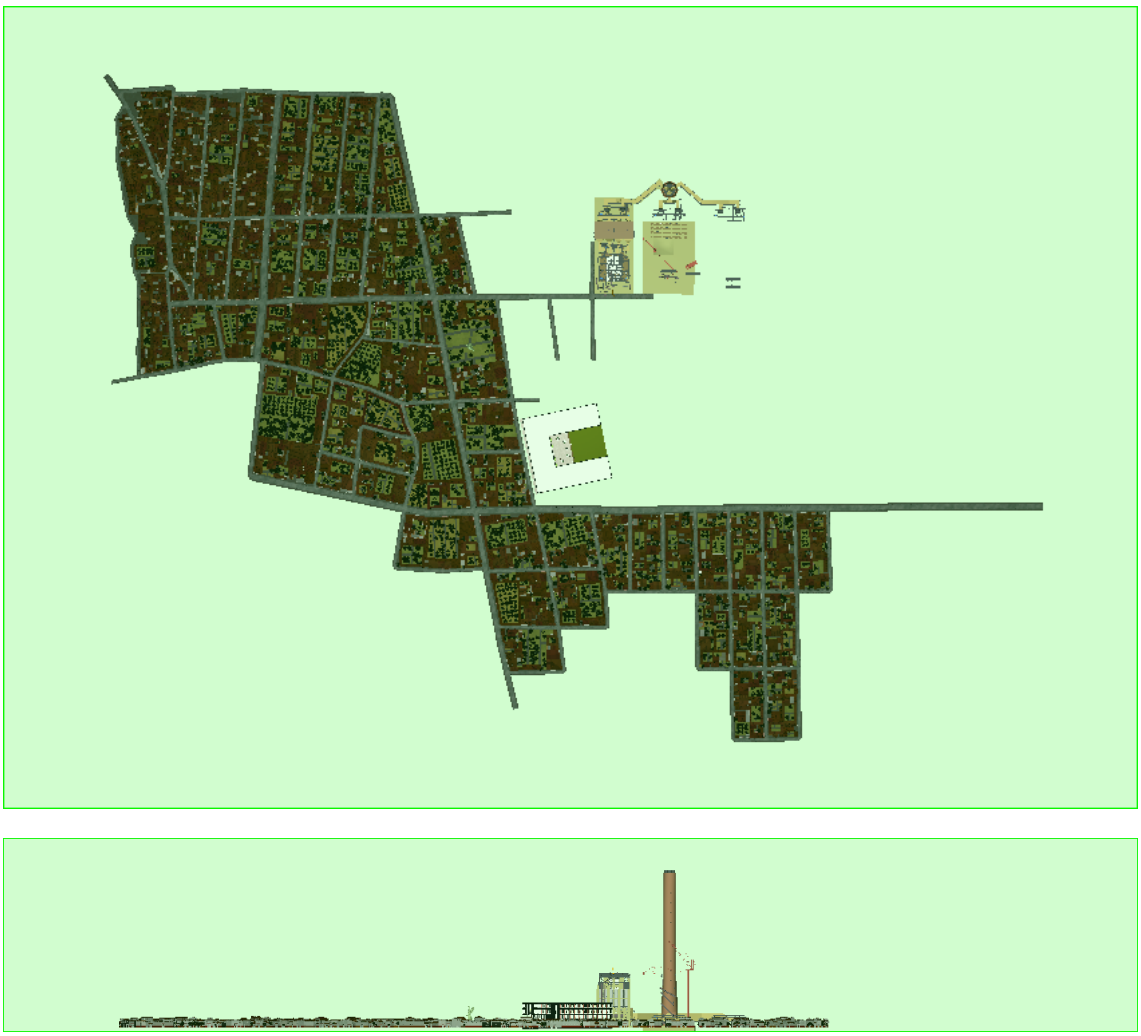


Abbildung 5.2: Darstellung des Bereiches (grün), innerhalb dessen in der Vorverarbeitung die Stichproben gezogen wurden.

- **Progressive-Blue-Surfels**: Falls das Achtfache der projizierten Größe kleiner ist als die Anzahl verfügbarer Samples, wird approximiert. Die Zahl an Surfels, die in der Vorverarbeitung erzeugt wird, ist ein Fünftel der Komplexität des zu approximierenden Teilbaums, maximal jedoch eine Million.

- **Forced-Surfels**: Es wird immer approximiert. Falls die Anzahl verfügbarer Surfels nicht ausreichend ist, wird die Größe der einzelnen Surfels auf bis zu 10×10 Pixel erhöht, um Löcher in der Darstellung zu vermeiden.

- **Discrete-Level-of-Detail**: Es wird die kleinste Detailstufe des Modells gewählt, die mehr Primitive hat als die projizierte Größe angibt. Maximal jedoch 2^{24} Primitive, da bei der benutzten Grafikkarte hier die Grenze liegt ab der die Darstellungsgeschwindigkeit für ein einzelnes Mesh einbricht. In der Vorverarbeitung wird die Komplexität eines jeden Modells schrittweise jeweils halbiert, bis es weniger als zwei tausend

Primitive hat.

Der Approximate-Coherent-Hierarchical-Culling++-Algorithmus und der Forced-Cubes-Algorithmus wurden nicht benutzt, da sich während der Entwicklung bereits herausgestellt hat, dass diese beiden keine weitere Verbesserung mehr mit sich bringen. Auf eine spezielle Messung hierfür wurde verzichtet.

5.2.5 Durchführung der Messungen

Die folgende Auflistung beschreibt die globalen Einstellungen, mit denen die Messungen durchgeführt wurden.

Messungen entlang des Kamerapfades

- Messungen, die ein deterministisches Ergebnis liefern, wie z. B. die Messung des Bildfehlers der benutzten Verfahren, wurden nur einmal durchgeführt.
- Bei Messungen, welche Schwankungen unterliegen, wurde der Kamerapfad 25-Mal nacheinander abgelaufen. Dies sind z. B. alle Laufzeitmessungen und die Bildfehlermessungen des Multi-Algorithmen-Renderings.
- Die während der Messung benutzte Bildauflösung betrug 1280×720 Pixel.
- Der benutzte Kameraöffnungswinkel war $60\,^\circ$ vertikal und dementsprechend ca. $90\,^\circ$ horizontal.
- Für alle Bildfehlermessungen wurde das Pyramidenverfahren mit SSIM als internem Vergleichsverfahren benutzt (siehe Abschnitt 2.5).

Messungen über Szenen-Eigenschafts-Funktionen

- Gemessen wurde der Durchschnitt aus acht Durchläufen.
- Da bei dieser Art der Messung die Kameraposition randomisiert springt, wurden vor jeder Messung zwei Aufwärmdurchläufe ausgeführt und deren Ergebnisse verworfen.
- Die während der Messung benutzte Bildauflösung betrug 1024×1024 Pixel.
- Der benutzte Kameraöffnungswinkel war $90\,^\circ$ sowohl vertikal als auch horizontal.
- Die Anzahl gezogener Stichproben für die Klassifizierungen betrug 5.000.
- Auch hier wurde für alle Bildfehlermessungen die Kombination aus Pyramiden- und SSIM-Verfahren benutzt.

5.2.6 Darstellung und Aufbereitung der Messergebnisse

Bei der Darstellung der Ergebnisse werden, mit einer Ausnahme, für die verwendeten Algorithmen die gleichen Farben verwendet, welche auch in den Bildschirmfotos zur Hervorhebung benutzt werden. Die entsprechenden Farben sind in Tabelle 5.1 angegeben.

Messungen entlang des Kamerapfades

- Die Kurven in den Diagrammen repräsentieren den Median der Messungen.
- Halbtransparente Flächen repräsentieren den Bereich zwischen oberem und unterem Quartil.

Tabelle 5.1: Zuordnung von Darstellungsfarben zu Algorithmen. Bis auf beim z-Buffer-Algorithmus stimmen die Farben bei Bildschirmfotos und Messkurven überein.

Algorithmus	Bildschirmfoto	Messkurven
z-Buffer	hellgrau	rot
CHC++	gelb	gelb
Color-Cubes	orange	orange
Spherical-Visibility-Sampling	grün	grün
Discrete-Level-of-Detail	violett	violett
Progressive-Blue-Surfels	hellblau	hellblau
Forced-Surfels	dunkelblau	dunkelblau

- Die vertikale Achse der Laufzeitmessungen ist logarithmisch skaliert.
- Die vertikale Achse der Bildfehlermessungen ist linear skaliert.
- Die Skalierung in der Horizontalen ist immer linear.

Messungen über Szenen-Eigenschafts-Funktionen

- Für alle Messungen dieser Art wurde aus den gezogenen Stichproben eine 2-D-Delaunay-Triangulierung berechnet und anhand der gemessenen Werte eingefärbt.
- Die absoluten Messwerte sind nicht mit denen der Messungen entlang des Kamerapfades vergleichbar. Dies liegt an den abweichenden Werten für die Bildauflösung, den Kameraöffnungswinkel etc. Dies lässt sich bei dieser Messtechnik leider nicht vermeiden. Der Vorteil dieser Messungen ist jedoch, dass damit das Multi-Algorithmen-Rendering flächendeckend evaluiert wird und nicht nur entlang eines Kamerapfades.

5.3 Laufzeit und Speicherverbrauch in der Vorverarbeitung

Für die Vorverarbeitung des Multi-Algorithmen-Renderings müssen zunächst die Vorverarbeitungen aller benutzten Verfahren durchgeführt werden. Das sind in diesem Fall die Algorithmen Color-Cubes, Forced-Cubes, Progressive-Blue-Surfels, Discrete-Level-of-Detail und Spherical-Visibility-Sampling. Dabei müssen die Surfels natürlich nur einmal berechnet werden. Erst danach kann die Vorverarbeitung des Multi-Algorithmen-Renderings selbst durchgeführt werden.

5.3.1 Persistenter Speicherverbrauch

Jedes Verfahren, welches eine Vorverarbeitung durchführt, um die Darstellung zur Laufzeit zu beschleunigen, muss die gesammelten Daten speichern. Dies ist unabhängig davon, ob es sich dabei um Sichtbarkeitsdaten, Surfels, Meshreduktionen oder, wie beim Multi-Algorithmen-Rendering, um Laufzeit- und Bildfehlerwerte handelt. Diese Daten müssen auch zur Laufzeit zur Verfügung stehen und belegen somit während des Walkthroughs zusätzlichen Speicherplatz.

Tabelle 5.2: Persistenter Speicherverbrauch der Daten aus der Vorverarbeitung. Angegeben sind die Werte der benutzten Algorithmen, der zusätzliche Speicherverbrauch des Multi-Algorithmen-Renderings bei ca. 13.000 Stichproben sowie zum Vergleich der Speicherverbrauch der Szene selbst.

Algorithmus	Speicherverbrauch
Discrete-Level-of-Detail	2448 MiB
Progressive-Blue-Surfels	908 MiB
Spherical-Visibility-Sampling	512 MiB
Color-Cubes	5 MiB
Multi-Algorithmen-Rendering	53 MiB
Σ	3926 MiB
Daten der Szene	4978 MiB

Tabelle 5.3: Dauer der Vorverarbeitung. Angegeben sind die Werte der benutzten Algorithmen und der zusätzliche Aufwand des Multi-Algorithmen-Renderings.

Algorithmus	Laufzeit (Std:Min:Sek)
Discrete-Level-of-Detail	Lucy + 00:19:41
Progressive-Blue-Surfels	00:30:21
Spherical-Visibility-Sampling	00:05:16
Color-Cubes	00:00:19
Σ	00:55:37
Multi-Algorithmen-Rendering (1 Stichprobe)	ca. 00:00:55
Multi-Algorithmen-Rendering (1.000 Stichproben)	15:14:28
Multi-Algorithmen-Rendering (13.000 Stichproben)	ca. 198:08:04

Fragestellung Wie viel Speicher belegen die Daten, die das Multi-Algorithmen-Rendering zusätzlich zu den benutzten Verfahren erzeugt?

Durchführung der Messung Gemessen wurde der persistente Speicherverbrauch, also die Größe der Daten, die gespeichert und zur Laufzeit verwendet werden. Der temporäre Speicherverbrauch liegt teilweise deutlich darüber. Insbesondere gilt dies für den Discrete-Level-of-Detail-Algorithmus: Die Reduktion der Lucy benötigte temporär über 32 GiB Speicher. Die für die Beispielszene und die benutzten Algorithmen ermittelten Werte sind in Tabelle 5.2 aufgeführt.

Auswertung Die Werte zeigen deutlich, dass der persistente Speicherverbrauch durch die benutzten Verfahren bestimmt wird. Der durch das Multi-Algorithmen-Rendering zusätzlich verursachte Speicherverbrauch zum Speichern der Laufzeit- und Bildfehlerwerte für ca. 13.000 Stichproben ist sehr gering.

Tabelle 5.4: Aufschlüsselung der Laufzeit des Multi-Algorithmen-Renderings nach Darstellung der Szene mit einzelnen Algorithmen, durchgeführten Bildvergleichen und Erzeugen von Referenzbildern.

Arbeitsschritt	Anteil an der Laufzeit
Darstellung mit z-Buffer	19,2 %
Darstellung mit Discrete-Level-of-Detail	18,5 %
Darstellung mit Coherent-Hierarchical-Culling++	13,1 %
Darstellung mit Spherical-Visibility-Sampling	10,0 %
Darstellung mit Color-Cubes	4,0 %
Darstellung mit Progressive-Blue-Surfels	1,3 %
Darstellung mit Forced-Surfels	0,9 %
Berechnung von Bildfehlern	5× 3,6 %
Erzeugen der Würfel-Texturen	7,6 %
Erzeugen der Referenzbilder	7,3 %

5.3.2 Dauer der Vorverarbeitung

Der zweite wichtige Aspekt bei einem Verfahren mit Vorverarbeitung ist die Laufzeit. Dauert die Vorverarbeitung zu lange, so reduziert dies die Einsatzmöglichkeiten des Verfahrens.

Fragestellung Wie lange dauert die Vorverarbeitung des Multi-Algorithmen-Renderings im Vergleich zu den benutzten Verfahren?

Durchführung der Messung Gemessen wurde für die benutzten Verfahren die komplette Laufzeit der Vorverarbeitung für alle Regionen. Die Ergebnisse sind in Tabelle 5.3 dargestellt. Eine Ausnahme bildet das Modell der Lucy. Dessen Reduktion musste aufgrund der hohen temporären Speicheranforderungen auf einem anderen Rechner durchgeführt werden und nahm mehrere Stunden in Anspruch.

Für das Multi-Algorithmen-Rendering wurde die Zeit gemessen, die benötigt wurde, um 1.000 Stichproben zu ziehen. Die beiden anderen angegebenen Werte wurden aus dieser Zeit berechnet.

Auswertung Im Gegensatz zum Speicherverbrauch wird in dieser Messung die Gesamtzeit deutlich vom Multi-Algorithmen-Rendering dominiert. Die Werte aller benutzten Verfahren liegen im Minutenbereich, während das Multi-Algorithmen-Rendering bereits ungefähr eine Minute pro Stichprobe benötigt. Die Gesamtlaufzeit der Vorverarbeitung des Multi-Algorithmen-Renderings ist abhängig von der Anzahl an Stichproben, die benötigt werden, damit das Multi-Algorithmen-Rendering zur Laufzeit zufriedenstellend funktioniert. Wie hoch diese benötigte Anzahl an Stichproben ist, wird in einer der nachfolgenden Messungen untersucht.

Zusätzlich hierzu stellt sich die Frage, wofür die Vorverarbeitung des Multi-Algorithmen-Renderings die Zeit verbraucht.

Fragestellung Wie verteilt sich die Laufzeit beim Ziehen der Stichproben auf die einzelnen Arbeitsschritte?

Durchführung der Messung Gemessen wurde wiederum über 1.000 Stichproben die Zeit, die für folgende Arbeitsschritte benötigt wurde:

- Darstellung von Regionen mit den einzelnen Algorithmen
- Berechnung von Bildfehlern
- Erzeugung von Referenzbildern für die Bildfehlermessungen
- Erzeugung der Würfeltexturen zur Wiederherstellung der Ausgangslage vor jedem Test

Die Ergebnisse der Messung sind in Tabelle 5.4 dargestellt.

Auswertung Wie zu erwarten, verbrauchen Algorithmen, die ein gutes Bild erzeugen, mehr Zeit bei der Darstellung, als solche, die stark approximieren. Allerdings muss pro benutztem approximativem Algorithmus ein Bildvergleich durchgeführt werden. Der Anteil der Laufzeit, welcher auf die Berechnung von Bildfehlern entfällt, steigt also mit der Anzahl benutzter approximativer Darstellungsalgorithmen. Der Anteil der Laufzeit, welcher auf die Erzeugung von Referenzbildern und Würfeltexturen entfällt, ist hingegen konstant und wird somit anteilsmäßig kleiner, wenn mehr Algorithmen benutzt werden. Des Weiteren sind die einzelnen Darstellungszeiten natürlich abhängig von der Szene, da beim Ziehen einer Stichprobe jede Region mit jedem Algorithmus dreimal dargestellt wird. Gleiches gilt auch für die Erzeugung der Würfeltexturen. Die Zeit zur Berechnung der Bildfehler ist abhängig von der Anzahl erzeugter Regionen und der Anzahl verwendeter approximativer Algorithmen. Auch das Erzeugen der Referenzbilder wird hierdurch beeinflusst. Die Zeit für diesen Schritt könnte man jedoch einsparen, solange mindestens ein Algorithmus benutzt wird, der ein korrektes Bild erzeugt, da man dieses für den Bildvergleich benutzen kann.

5.4 Verhalten des Multi-Algorithmen-Renderings beim Walkthrough

Zur Laufzeit sind die entscheidenden Kriterien für die Einsetzbarkeit eines approximativen Darstellungsalgorithmus zum einen die Zeit, die benötigt wird, um ein Bild darzustellen, und zum anderen der Bildfehler, der dabei entsteht. Ein weiterer Aspekt sind die Schwankungen dieser Werte.

Abbildung 5.3 zeigt zunächst einige Screenshots, die einen ersten Eindruck von den durch das Multi-Algorithmen-Rendering dargestellten Bildern liefern.

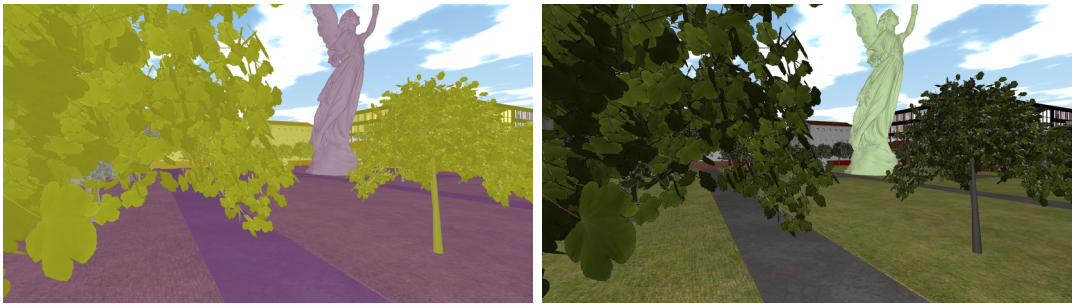
5.4.1 Dauer der Bildberechnung

Um Aussagen über die Laufzeit des Multi-Algorithmen-Renderings treffen zu können, ist es zunächst notwendig, das Verhalten der einzelnen Algorithmen zu untersuchen. Diese werden zwar vom Multi-Algorithmen-Rendering niemals für die Darstellung der kompletten Szene eingesetzt, jedoch liefern diese Werte gute Anhaltspunkte für dessen Grenzen.

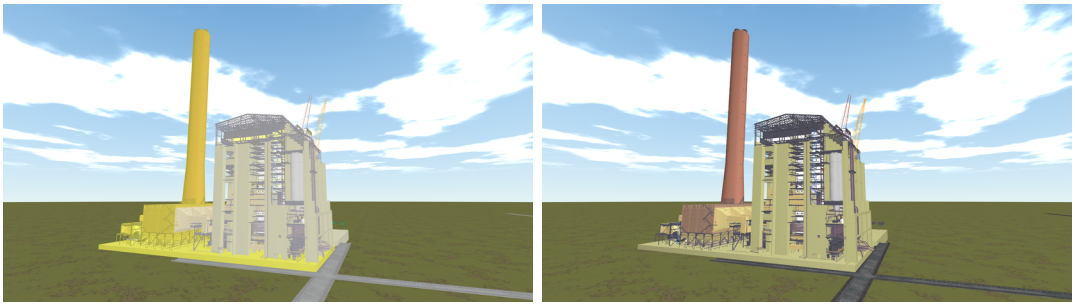
Fragestellung Wie verhält sich die Laufzeit der benutzten Algorithmen, wenn mit ihnen die gesamte Szene dargestellt wird?

(a) Position 0 bzw. 1303: An dieser Position wird der z-Buffer-Algorithmus eingesetzt, obwohl der CHC++-Algorithmus deutlich schneller wäre. Dies ist jedoch irrelevant, da die vorgegebene Laufzeit deutlich unterschritten wird.

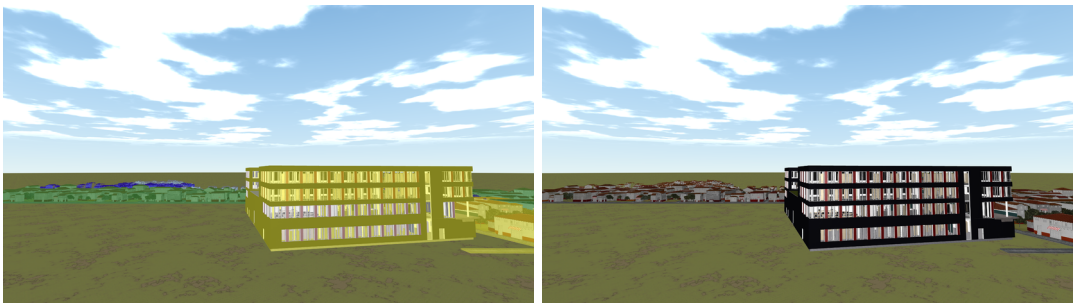
(b) Position 365: Trotz der teilweise sichtbaren Lucy (violett) wird nahezu das gesamte Bild korrekt dargestellt.



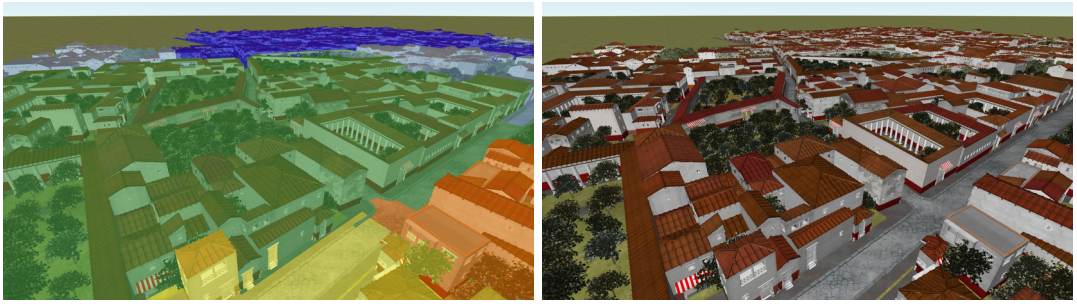
(c) Position 468: Im Bereich um diese Position bricht bei allen benutzten Verfahren entweder die Laufzeit oder der Bildfehler ein.



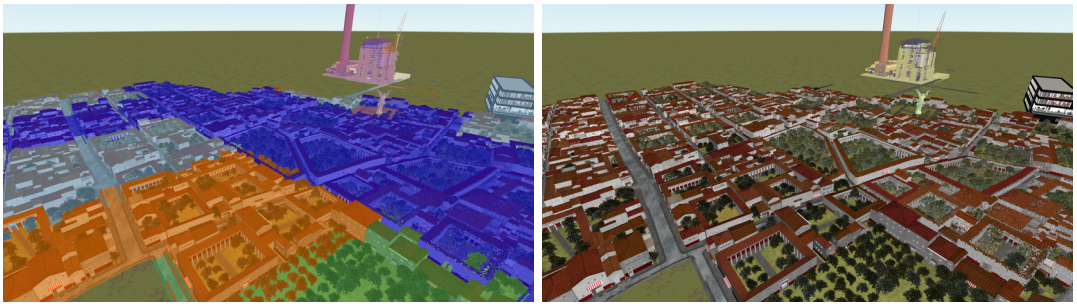
(d) Position 560: Das einzelne Powerplant stellt für die benutzte Grafikkarte keine Herausforderung dar. Hier spielt es keine Rolle, welche der korrekten Algorithmen eingesetzt werden.



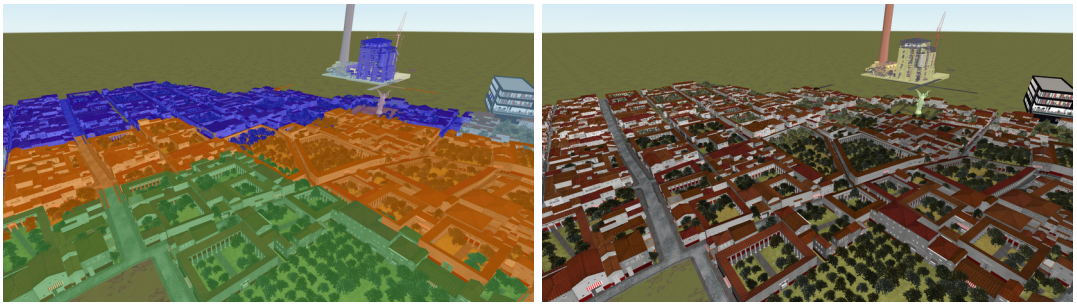
(e) Position 698: Hier beginnt der Teil des Kamerapfades, auf dem der Bildfehler für den Benutzer durchgehend bemerkbar wird. Es werden erstmalig Surfels eingesetzt.



(f) Position 900: Ein homogener Ausschnitt der Szene. Je weiter die Regionen von der Kamera entfernt sind, desto stärker wird approximiert.

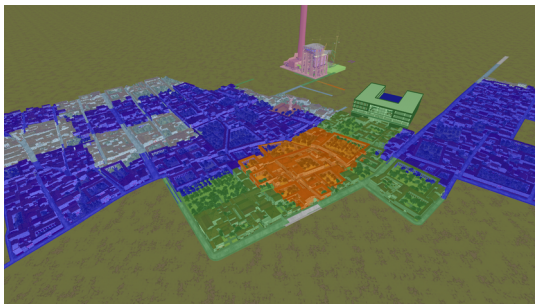


(g) Position 1186: Ein heterogener Ausschnitt der Szene. Entfernte Objekte werden weniger stark approximiert als nahe.

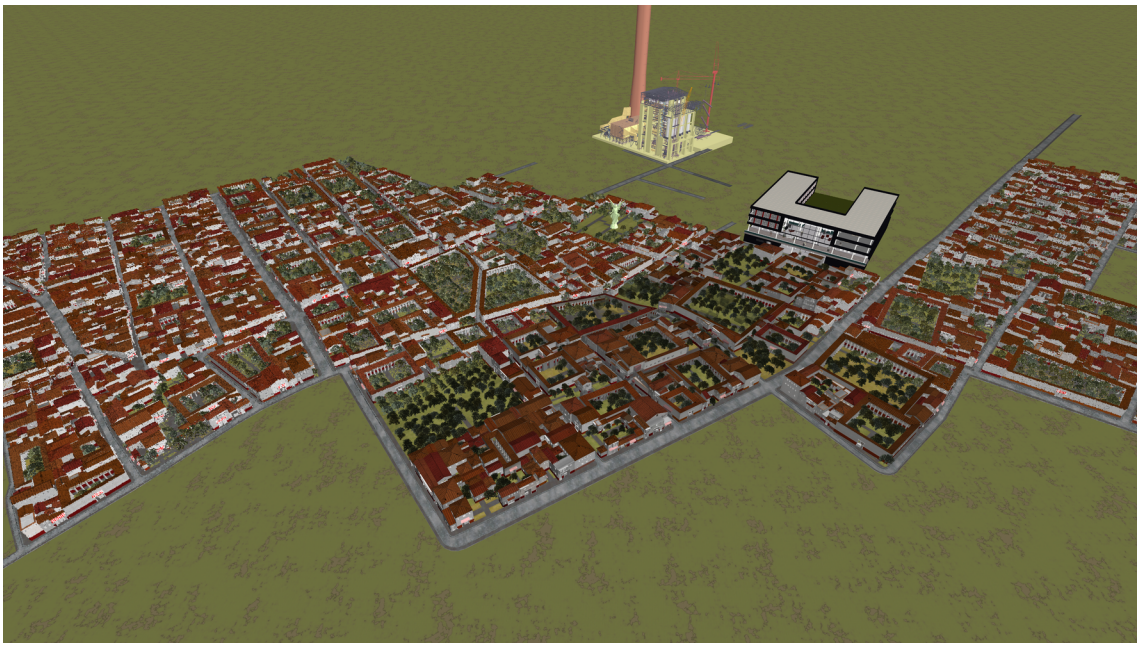


(h) Position 1186: Gleiche Position wie zuvor. Es wird eine andere Zuordnung von Algorithmen zu Regionen benutzt. Dies ist für den Benutzer als Flackern wahrnehmbar.

(i) Nahaufnahme der Lucy. Trotz extrem großer projizierter Größe wird das Modell approximiert dargestellt.



(j) Überblick über die Szene mit Hervorhebung der benutzten Algorithmen.



(k) Überblick über die Szene. Regionen, die durch Surfels dargestellt werden (blau in Abbildung (j)), sind generell zu hell. Dies ist deutlich störender als der geometrische Fehler, der durch die Surfels erzeugt wird.

Abbildung 5.3: Ausgewählte Bildschirmfotos entlang des Kamerapfades (mit Angabe der Position), sowie einige zusätzliche an speziellen Positionen. Bei allen Bildern wurde die vorgegebene Darstellungszeit von 100 ms eingehalten bzw. unterschritten. In Abbildung (a) bis (h) sind zu jeder Position jeweils Bilder mit und ohne Hervorhebung der Algorithmen dargestellt.

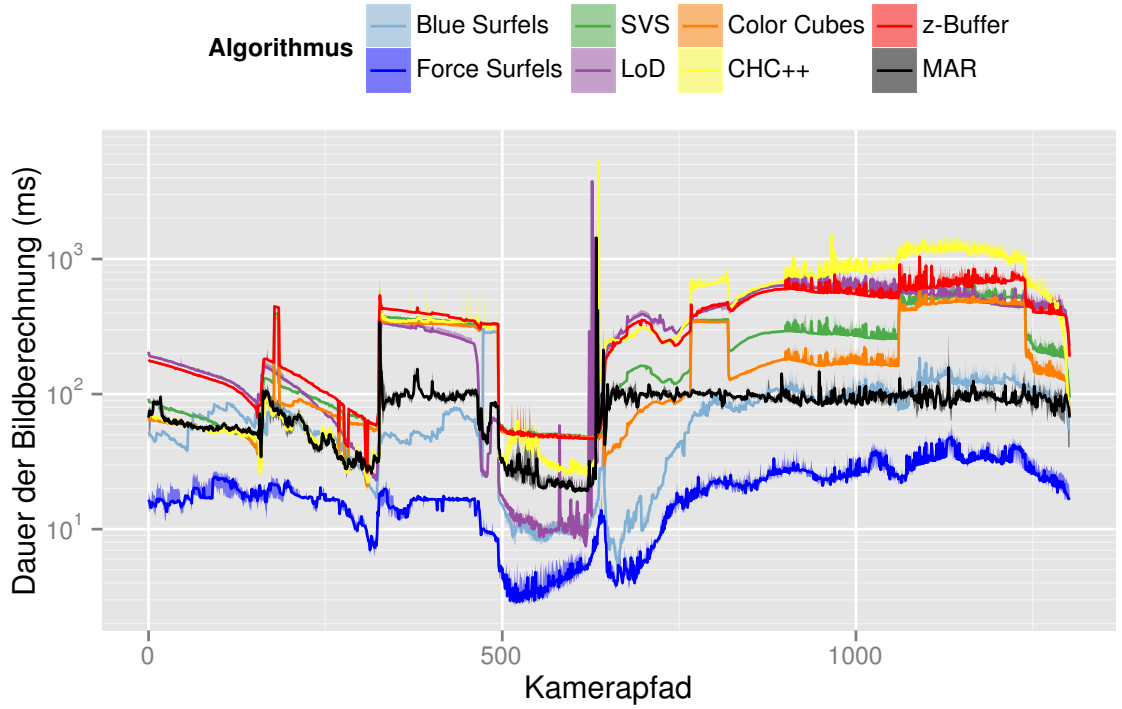


Abbildung 5.4: Die zur Darstellung der Testszene benötigten Zeit bei Einsatz des Multi-Algorithmen-Renderings und bei ausschließlichem Einsatz der einzelnen Algorithmen.

Durchführung der Messung Gemessen wurde die Laufzeit der benutzten Verfahren zur Darstellung der gesamten Szene entlang des Kamerapfades. Zum Vergleich wurde auch die Laufzeit des Multi-Algorithmen-Renderings bei optimalen Einstellungen gemessen. Die Ergebnisse der Messung sind in Abbildung 5.4 dargestellt.

Auswertung Die Messung zeigt zunächst, dass nur der Forced-Surfels-Algorithmus – der unter allen am stärksten approximiert – in der Lage ist die Szene durchgehend mit 100 ms oder weniger darzustellen. Dies war jedoch zu erwarten, da die Szene so aufgebaut ist, dass keiner der Algorithmen dazu geeignet ist, sie in ihrer Gänze darzustellen.

Weiterhin erhält man durch die dargestellten Kurven einen Überblick darüber, wie viel Zeit man mindestens benötigt, um die Szene darzustellen (Forced-Surfels) und wie viel Zeit man höchstens dafür benötigt (z-Buffer und CHC++), um die Szene korrekt darzustellen. So zeigt die Messung auch, dass es möglich ist die Szene mit Hilfe des Forced-Surfels-Algorithmus in nur 50 ms darzustellen. Daraus folgt, dass dies auch mit dem Multi-Algorithmen-Rendering möglich sein sollte, da dieses die Möglichkeit hat, genau diesen Algorithmus für die ganze Szene einzusetzen. Dies wird in Abschnitt 5.5.4 näher untersucht.

Außerdem kann man gut erkennen, dass die Kurven mancher Algorithmen sich schneiden. Dies gilt insbesondere für den z-Buffer-Algorithmus und den CHC++.

Das heißt, selbst wenn man jegliche Approximationen ausschließt und ein korrektes Bild darstellen möchte, so ist es sinnvoll, den Algorithmus abhängig von der Position zu wechseln.

Des Weiteren schwankt die Kurve des Multi-Algorithmen-Renderings um die eingestellte Ziellaufzeit von 100 ms oder liegt darunter. Nur in Ausnahmefällen wird dieser Wert signifikant überschritten. Betrachtet man hingegen z. B. die Kurve des CHC++ und beachtet die logarithmische Skalierung der y-Achse des Diagramms, so schwankt dessen Laufzeit deutlich mehr. Da das Multi-Algorithmen-Rendering alle Algorithmen zur Darstellung einsetzt, muss dessen Laufzeit ebenfalls schwanken, zumindest solange die Darstellung der bestimmende Faktor für die Laufzeit ist. Wie stark diese Schwankungen sind, wird im Zuge des Vergleichs der unterschiedlichen Interpolations- und Regelungsverfahren, in Abschnitt 5.4.3 näher untersucht.

5.4.2 Größe des Fehlers im Bild

Zusätzlich zur Laufzeit der Algorithmen muss man bei approximativen Algorithmen stets auch die Qualität des Ergebnisses betrachten. Bei Renderingalgorithmen ist dies der Fehler im erzeugten Bild.

Fragestellung Wie groß ist der Bildfehler, den die benutzten Algorithmen erzeugen, wenn mit ihnen die gesamte Szene dargestellt wird?

Durchführung der Messung Gemessen wurde der Bildfehler, den die benutzten Verfahren bei der Darstellung der gesamten Szene erzeugen, sowie zum Vergleich der Bildfehler des Multi-Algorithmen-Renderings bei optimalen Einstellungen. Die Ergebnisse der Messung sind in Abbildung 5.5 dargestellt.

Auswertung Die Grafik zeigt deutlich, dass der Forced-Surfels-Algorithmus, wie zu erwarten, auf dem ersten Stück des Kamerapfades einen sehr großen Bildfehler erzeugt, da hier der Abstand zwischen Kamera und Szene sehr gering ist. Des Weiteren schwankt der Bildfehler des Discrete-Level-of-Detail auf diesem Stück ebenfalls sehr stark. Dies liegt daran, dass der Pfad dicht an Bäumen vorbeiführt, welche nicht zusammenhängend modelliert sind, so dass bei einer Reduktion des Modells umgehend Blätter verschwinden.

Weiterhin ist zu erkennen, dass der Bildfehler des Progressive-Blue-Surfels-Algorithmus ähnlich dem des Multi-Algorithmen-Renderings ist. Da der Progressive-Blue-Surfels-Algorithmus auch sehr schnell ist, wäre dies der Algorithmus, den man bei dieser speziellen Szene zur Darstellung nutzen sollte, wenn man nur einen nehmen kann. Allerdings erzeugt er genau in dem Bereich den größten Fehler, in dem er auch die höchste Laufzeit hat. Bei der eingestellten Laufzeit von 100 ms ist dies vielleicht noch vertretbar, stellt man die Laufzeit jedoch auf z. B. 50 ms ein, so wird das Multi-Algorithmen-Rendering diesen Wert erreichen, der Progressive-Blue-Surfels-Algorithmus jedoch nicht.

Des Weiteren erzeugt der Forced-Surfels, welcher in der vorherigen Laufzeitmessung als einziger durchgehend schneller als 100 ms war, auch durchgehend den mit Abstand größten Bildfehler aller Algorithmen.

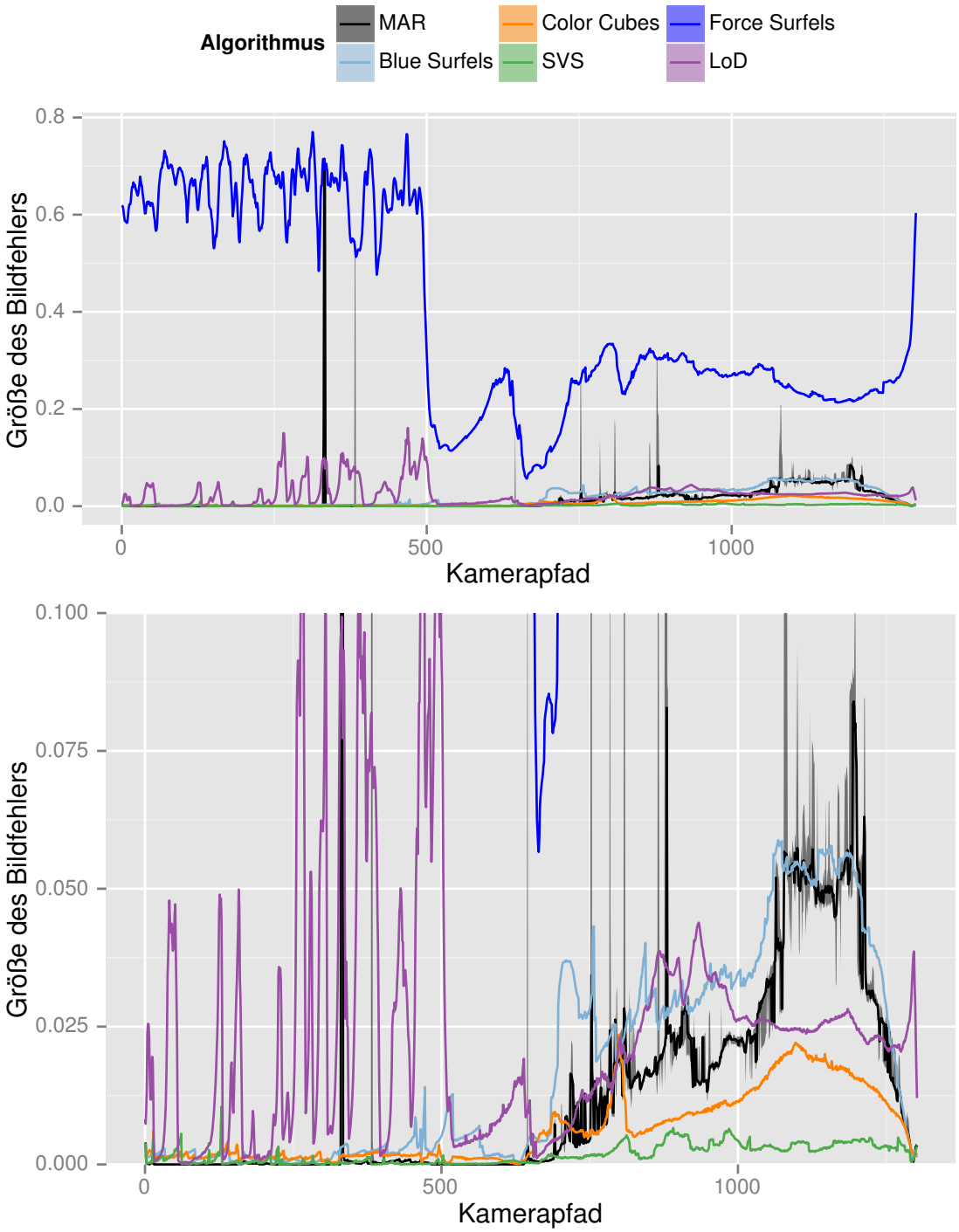


Abbildung 5.5: Der bei der Darstellung der Testszene auftretender Bildfehler bei Einsatz des Multi-Algorithmen-Renderings und bei ausschließlichem Einsatz der einzelnen Algorithmen. Oben: Gesamte Messung. Unten: Vergrößerte Darstellung der Fehlerwerte im Bereich kleiner 0.1.

Bereits an dieser Stelle ist klar, dass das Multi-Algorithmen-Rendering funktioniert. Im Bereich ungefähr zwischen Messpunkt 1.100 und 1.250 erreicht bis auf den Forced-Surfels-Algorithmus kein anderer eine Laufzeit von 100 ms. Das Multi-Algorithmen-Rendering hingegen schon. Beim Multi-Algorithmen-Rendering ist der Bildfehler jedoch wesentlich kleiner ist als beim Forced-Surfels-Algorithmus. Daher folgt, dass die Kombination von unterschiedlichen Algorithmen für geeignete Regionen, der ausschließlichen Benutzung einzelner Algorithmen überlegen ist.

Der ähnliche Verlauf der Kurven des Multi-Algorithmen-Renderings und des Progressive-Blue-Surfels-Algorithmus in diesem Bereich lässt weiter vermuten, dass letzterer in diesem Bereich überwiegend vom Multi-Algorithmen-Rendering eingesetzt wird. Dem ist jedoch nicht so (siehe dazu Abschnitt 5.4.7)

5.4.3 Die verschiedenen Interpolations- und Regelungsverfahren

Für das Multi-Algorithmen-Rendering wurden verschiedene Methoden zur Interpolation der Stichproben als auch zur Regelung der Laufzeit entwickelt. Um zu evaluieren, welche Kombination dieser Methoden die beste ist, wurden Laufzeit- und Bildfehlermessungen für alle Kombinationen aus Interpolations- und Regelungsverfahren durchgeführt.

5.4.3.1 Einfluss auf die Laufzeit

Bei der Laufzeit unterteilt sich die Fragestellung in zwei Aspekte: Wird die Bildrate im richtigen Bereich geregelt und wie stark schwankt die Bildrate.

Fragestellung Wie verhält sich die Laufzeit des Multi-Algorithmen-Renderings auf dem Kamerapfad bei Verwendung unterschiedlicher Kombinationen aus Regelungs- und Interpolationsmethode?

Durchführung der Messung Gemessen wurde die Laufzeit aller Kombinationen von Regelungs- und Interpolationsverfahren. Die Ergebnisse sind in Abbildung 5.6 dargestellt.

Auswertung Es ist deutlich zu erkennen, dass die relative Regelung sehr stark oszilliert. Dabei überschreitet die Laufzeit häufig das Vierfache des eingestellten Wertes. Praktisch ist diese Regelung damit unbrauchbar. Eine konstant langsame Bildrate wäre für den Benutzer weit besser zu handhaben als eine so stark schwankende.

Des Weiteren ist ersichtlich, dass die Maximums-Interpolationsmethode weniger Spitzen in der Messung aufweist. Da es sich hierbei um den Median der Messwerte handelt und nicht um Ausreißer, ist dies nicht zu vernachlässigen. Diese Messspitzen treten reproduzierbar bei fast jeder Iteration der Messung auf, allerdings nur bei Ablaufen dieses Kamerapfades. Kommt der Benutzer aus einer anderen Richtung oder bleibt an der entsprechenden Position stehen, so funktioniert das Multi-Algorithmen-Rendering umgehend wieder flüssig.

Eine Ausnahme hierzu stellt die Spitze kurz vor Messpunkt 500 bei der Kombination Maximum-Gedächtnis dar. Hierbei handelt es sich um mehrere aufeinanderfolgende Messpunkte, deren Position unmittelbar vor dem Modell der Lucy liegen. Würde der Benutzer hier stehenbleiben, würde sich das Verhalten des Multi-Algorithmen-Renderings bei dieser Kombination auch nicht einpendeln, sondern abwechselnd wenige sehr schnelle Bilder und ein sehr langsames Bild liefern. Dies liegt daran, dass der

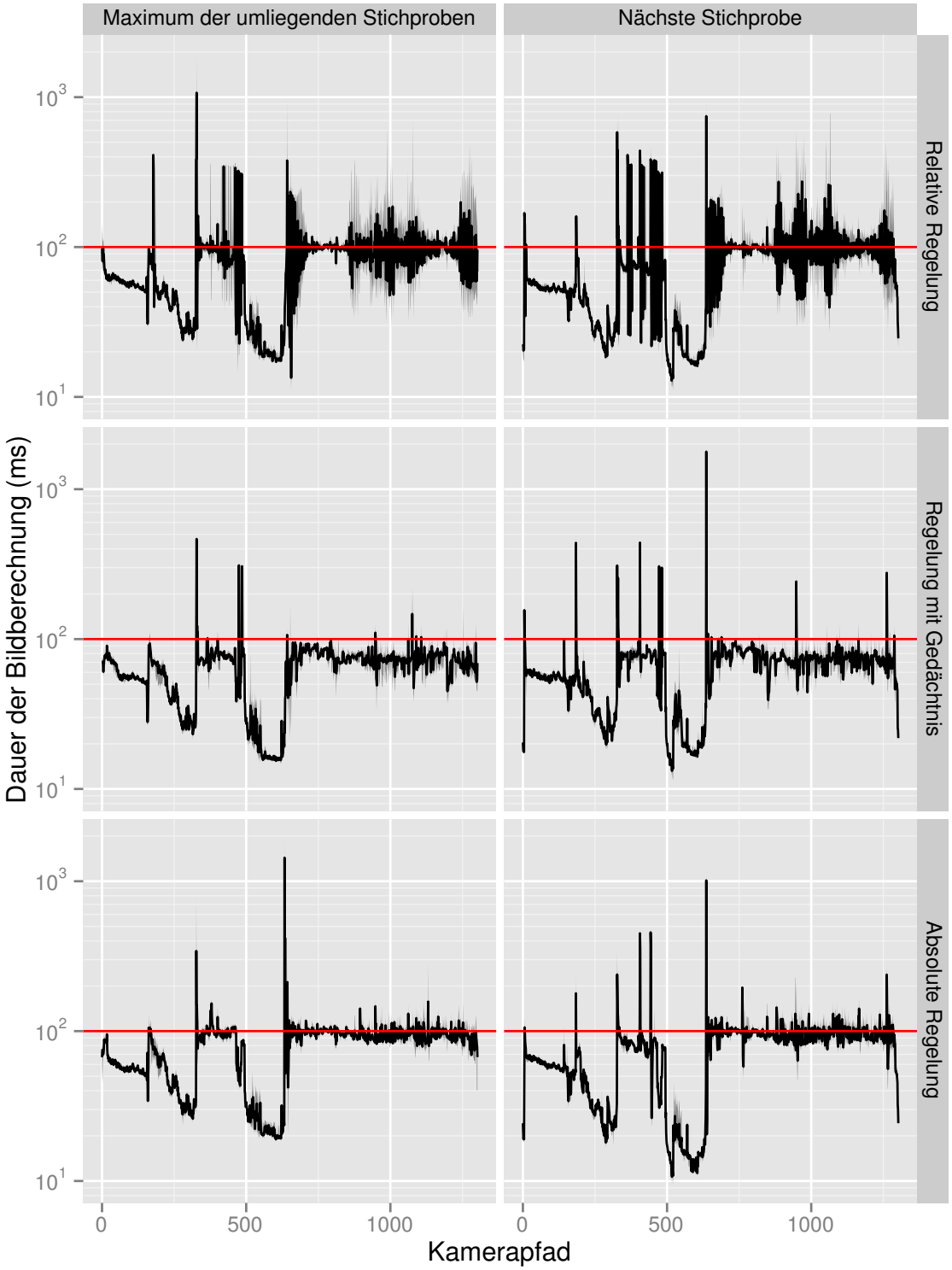


Abbildung 5.6: Darstellung der Laufzeit des Multi-Algorithmen-Renderings mit den verschiedenen Kombinationen von Interpolations- und Regelungsverfahren. Die rote Linie stellt den eingestellten Sollwert dar.

Regelkreis den Vorgabewert für das LP langsam hoch regelt, weil die Darstellung des Bildes schnell geht, bis das LP entscheidet, das Modell der Lucy korrekt darzustellen. Dies dauert dann einmalig sehr lange, weshalb der Regelkreis den Vorgabewert für das LP wieder stark reduziert.

Außerdem kann man noch sehen, dass bei den Varianten mit Gedächtnis die Regelung der Laufzeit nicht um den Sollwert herum, sondern darunter erfolgt. Dies kann jedoch durch eine einfache Skalierung des Sollwerts ausgeglichen werden. Daher ist dies kein Kriterium für die Güte der Regelung.

5.4.3.2 Einfluss auf die Bildqualität

Auch bei der Bildqualität ist nicht nur die Höhe der Messkurve sondern auch deren Schwankung interessant, da ein fluktuierender Bildfehler ebenfalls sehr störend auf den Benutzer wirkt. Dies ist eine der Haupt schwächen des Multi-Algorithmen-Renderings und wird in einer späteren Messung (5.4.8) gesondert untersucht.

Fragestellung Wie verhält sich der vom Multi-Algorithmen-Rendering erzeugte Bildfehler auf dem Kamerapfad bei Verwendung unterschiedlicher Kombinationen aus Regelungs- und Interpolationsmethode?

Durchführung der Messung Gemessen wurde der Bildfehler des durch das Multi-Algorithmen-Rendering erzeugten Bildes im Vergleich zu einem durch den CHC++-Algorithmus erzeugten Bild. Die Messkurven sind in Abbildung 5.7 dargestellt.

Auswertung Die Varianten mit Gedächtnis weisen generell einen größeren Bildfehler auf. Dies ist allerdings aufgrund der geringeren Laufzeit allerdings auch zu erwarten und daher wiederum kein Kriterium für die Güte der Regelung. Allerdings fluktuiert der Bildfehler bei diesen Varianten auch stark. Dies liegt daran, dass das Verfahren mit dieser Regelungsvariante den Sollwert kontinuierlich langsam hoch und schnell wieder herunter regelt.

Dies ist insbesondere bei der Messung mit absoluter Regelung nicht der Fall. Diese Weist zwar wiederum einige Messspitzen auf, allerdings liegen diese bis auf eine in Fehlerbereichen, in denen die Variante mit Gedächtnis in großen Bereichen des Kamerapfades schwankt. Auch hier ist es wieder so, dass bei stehender Kamera die Spitzen nur einmalig auftreten, wohingegen die Variante mit Gedächtnis durchgehend fluktuiert.

Insgesamt zeigt die Kombination aus absoluter Regelung und Interpolation der umliegenden Stichproben das beste Verhalten. Dies stimmt auch mit dem subjektiven Empfinden während eines manuellen Walkthroughs überein. Daher wurde diese Variante für alle folgenden Messungen benutzt, bei denen nicht explizit etwas anderes angegeben ist.

5.4.4 Wie viele Stichproben werden benötigt?

Um diese Frage beantworten zu können, ist es zunächst notwendig, das bestmögliche Ergebnis zu kennen. Dazu wurde eine zweite Vorverarbeitung durchgeführt, bei der exakt an den Positionen die bei der Messung auf dem Kamerapfad vorkommen jeweils eine Stichprobe gezogen wurde.

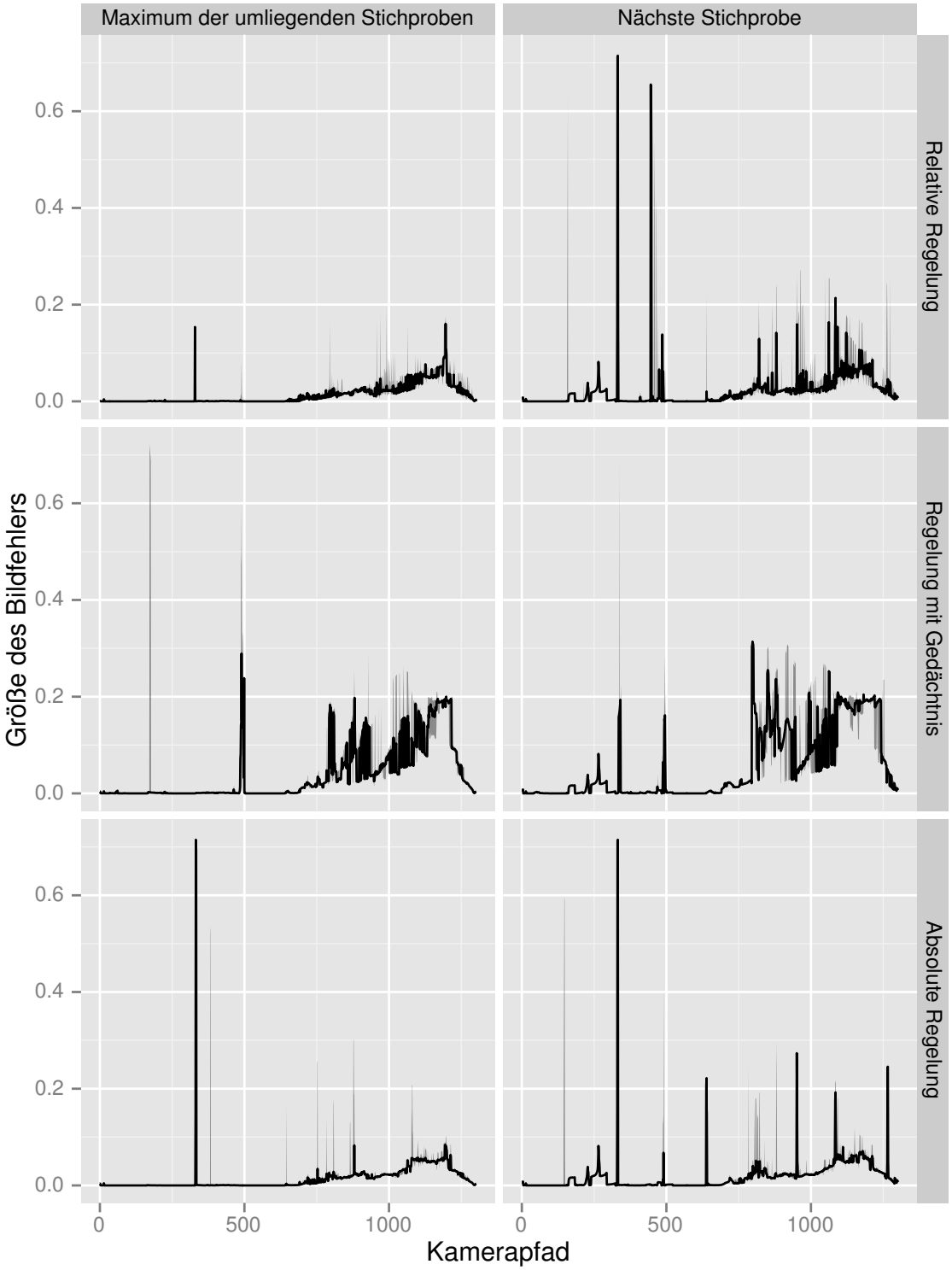


Abbildung 5.7: Darstellung des erzeugten Bildfehlers vom Multi-Algorithmen-Rendering mit den verschiedenen Kombinationen von Interpolations- und Regelungsverfahren.

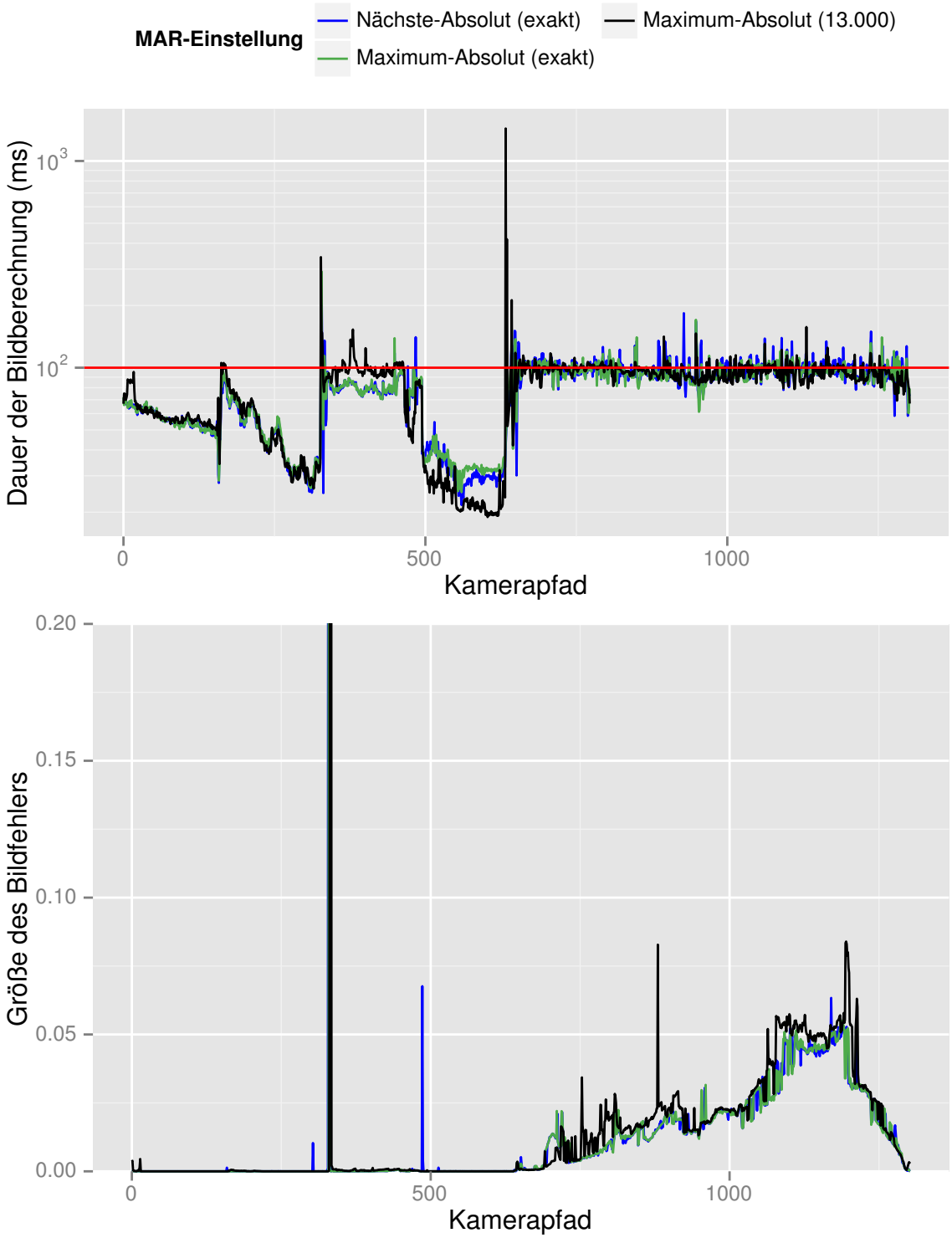


Abbildung 5.8: Darstellung von Laufzeit und Bildfehler bei exakten Positionen für die Stichproben sowie zum Vergleich die gleiche Messung mit 13.000 verteilten Stichproben.

Da diese Stichproben quasi perfekt sind, wurde bei den Messungen, zusätzlich zu der zuvor ermittelten besten Kombination aus absoluter Regelung und Interpolation der umliegenden Stichproben, auch wieder die ausschließliche Benutzung der nächsten Stichprobe untersucht.

Fragestellung Wie gut wird das Verfahren, wenn für die Positionen auf dem Kamerapfad Stichproben an exakt der gleichen Position zur Verfügung stehen?

Durchführung der Messung Gemessen wurden die Laufzeit und der Bildfehler des Multi-Algorithmen-Renderings bei absoluter Regelung mit beiden Interpolationsverfahren. Die Messkurven sind in Abbildung 5.8 dargestellt. Zusätzlich wurde die Messkurve, der im vorherigen Abschnitt ermittelten besten Kombination aus Interpolations- und Regelungsverfahren, bei 13.000 verteilten Stichproben eingeblendet. Da die Kurven sehr dicht beieinander liegen wurde auf die Darstellung der Quartile verzichtet.

Auswertung Zwischen den drei Kurven ist kein Signifikanter Unterschied zu erkennen. Sie unterscheiden sich hauptsächlich durch die Positionen der Messspitzen auf dem Kamerapfad und Unterschiede in der Laufzeit, die allerdings unterhalb des Sollwerts und in Bereichen mit korrekter Bilddarstellung liegen. Somit kann man sagen, dass 13.000 gleichmäßig verteilte Stichproben bei dieser Szene ausreichend sind.

Zur Ermittlung der benötigten Anzahl Stichproben wurde nun eine zweite Messung durchgeführt, bei der die Menge der Stichproben auf die 1.000 zuerst gezogenen Stichproben reduziert wurde. Der Wert 1.000 wurde deshalb gewählt, weil bei dieser Anzahl an Stichproben die Vorverarbeitung eine akzeptable Laufzeit hat. Akzeptabel deshalb, weil man die Vorverarbeitung hier zu Feierabend starten könnte und sie wäre am nächsten Tag zu Arbeitsbeginn fertig.

Fragestellung Sind 1.000 Stichproben ausreichend?

Durchführung der Messung Gemessen wurden für beide Interpolationsvarianten sowohl die Laufzeit als auch der Bildfehler. Die Stichproben wurden für diese Messung nicht neu gezogen, sondern sind identisch mit den 1.000 Stichproben, die in der bestehenden Vorverarbeitung zuerst gezogen wurden. Abbildung 5.9 zeigt die Ergebnisse.

Auswertung Es ist deutlich zu erkennen, dass bei Benutzung der nächstliegenden Stichprobe in zwei Bereichen der Messung sowohl die Laufzeit als auch die Bildqualität signifikant schlechter werden. Dies liegt daran, dass an diesen Stellen je eine Stichprobe am nächsten liegt, welche so positioniert ist, dass ein Großteil der Szene verdeckt ist. Dies hat zur Folge, dass in der Vorverarbeitung für den CHC++ ein sehr kleiner Laufzeitwert gemessen wurde. Dadurch wird zur Laufzeit ein Bereich der Szene, der nun jedoch nicht verdeckt ist und dessen Darstellung mit dem CHC++ sehr lange dauert, korrekt dargestellt, während im Nahbereich der Kamera stark approximiert wird.

Die Maximum-Interpolationsmethode hingegen zeigt sich von der geringen Anzahl an Stichproben völlig unbeeindruckt, da hier die geringe Laufzeit des CHC++ bei einer Stichprobe aufgrund der Bildung des Maximums nicht beachtet wird. Insgesamt sieht das Ergebnis hier sogar besser aus als bei der Verwendung von 13.000 Stichproben. Dies könnte daran liegen, dass sich bei jedem Wechseln einer Stichprobe das Verhältnis

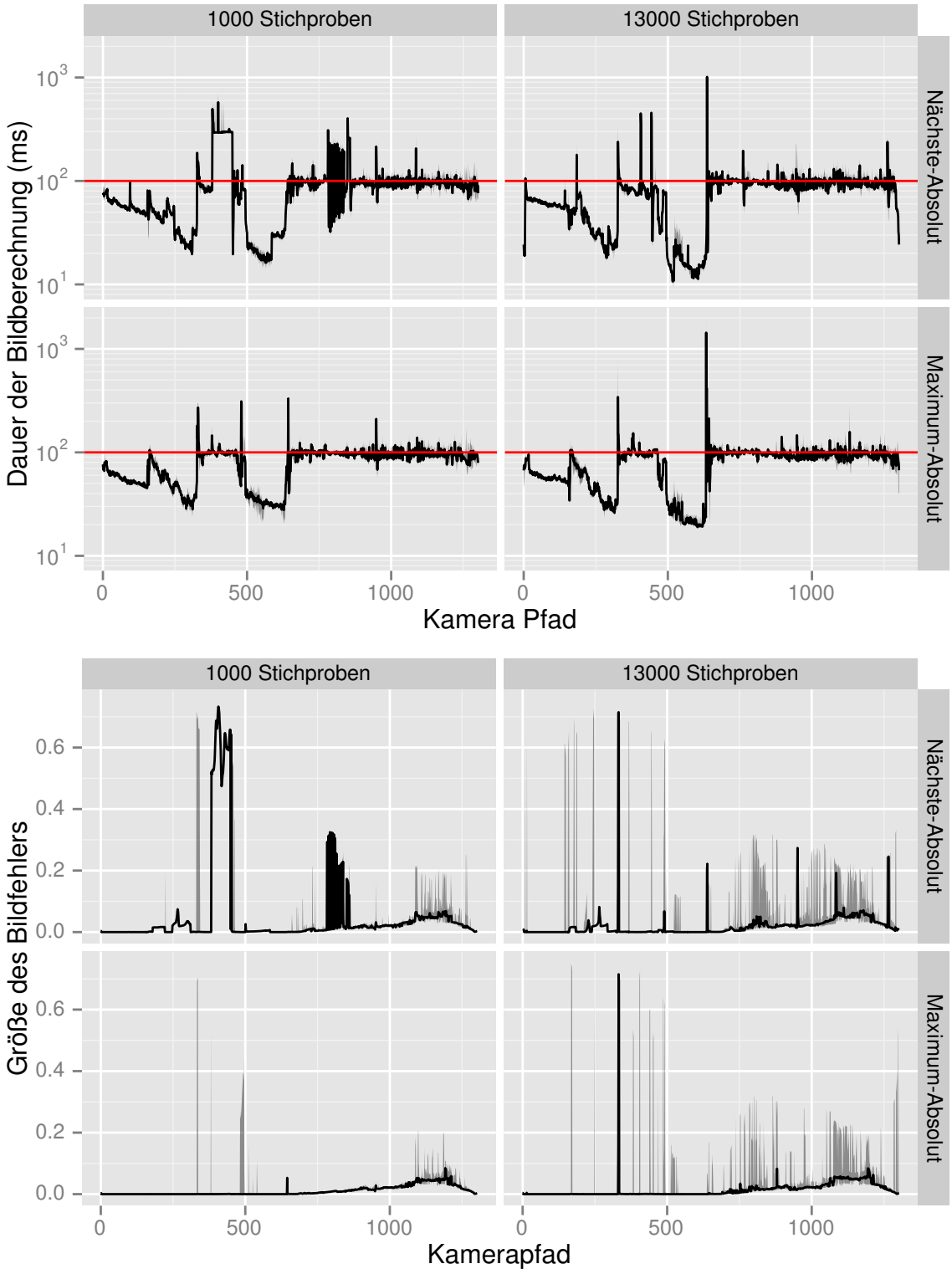


Abbildung 5.9: Darstellung von Laufzeit und Bildfehler mit unterschiedlicher Anzahl an Stichproben und unterschiedlichen Interpolationsmethoden.

Abbildung 5.10: Seitenansicht des Bereiches (grün), innerhalb dessen in der Vorverarbeitung die Stichproben gezogen wurden.

zwischen geschätzter und gemessener Laufzeit verändert. Dann muss das Multi-Algorithmen-Rendering jedoch nachregeln um dies wieder auszugleichen (siehe Abschnitt 5.4.5).

Zunächst scheint es also so zu sein, dass 1.000 Stichproben ausreichend sind, allerdings zeigt die Messung in Abschnitt 5.5.4.1, dass diese Aussage nicht für alle Positionen in der Szene gilt.

Diese Messung wirft allerdings zunächst eine weitere Fragestellung auf: Bei nur 1.000 Stichproben, welche in drei Dimensionen verteilt sind, ergibt sich eine Anzahl von nur zehn Stichproben pro Dimension. Wenn man den Bereich, in dem die Stichproben gezogen wurden, betrachtet (Abbildung 5.10), so ist ersichtlich, dass in der Regel nicht mehr als eine dieser zehn Stichproben innerhalb der Geometrie von Pompeji liegt. Die anderen neun liegen oberhalb der Stadt. Dieses wird noch dadurch verschärft, dass die Ausdehnung der Szene in der Höhe wesentlich geringer ist als in der Breite und der Länge. Dadurch werden in der Höhe auch deutlich weniger Stichproben gezogen.

Dadurch benutzt die Maximum-Interpolationsmethode jedoch stets mindestens eine Stichprobe, die oberhalb der Stadt liegt, welche hier vermutlich die Laufzeit- und auch die Bildfehlerwerte der einzelnen Regionen bestimmt, und somit auch alleinig das Verhalten des Multi-Algorithmen-Renderings. Somit wären jegliche Stichproben innerhalb der Szene überflüssig. Weiterhin könnte die Anzahl an Stichproben oberhalb der Szene eventuell auch noch weiter reduziert werden, da sich hier das Verhalten der einzelnen Algorithmen nicht sprunghaft verändern kann. Bei kontinuierlichem Verhalten von Laufzeit und Bildfehler könnte dann auch statt des Maximums eine gewichtete Interpolation der Stichproben (z. B. mit baryzentrischen Koordinaten) sinnvoll sein. Dies gilt jedoch nur für flache Szenen, die sich nur in der Ebene ausdehnen und nicht für Szenen, die sich auch in der Höhe ausdehnen.

5.4.5 Abweichung zwischen geschätzten und gemessenen Laufzeitwerten

Die Laufzeitwerte aus der Vorverarbeitung bilden einen Schätzwert für die zu erwartenden realen Werte während des Walkthroughs. Technisch bedingt sind die Laufzeit-Schätzwerte dabei stets größer als die realen Werte. Um dieses auszugleichen wurde das Multi-Algorithmen-Rendering als Regelkreis umgesetzt. Die Regelvariante mit Gedächtnis beachtet diese Schätzwerte nicht und regelt ausschließlich aufgrund der gemessenen Laufzeit.

Die beiden anderen Regelvarianten basieren jedoch auf diesen Schätzwerten. Sie regeln den Sollwert für das LP aufgrund der absoluten bzw. relativen Abweichung zwischen Schätzwert und gemessener Dauer der Bildberechnung. Daher wurde untersucht, wie sich diese Abweichungen über den Kamerapfad verhalten, da Sprünge der Abweichung eine

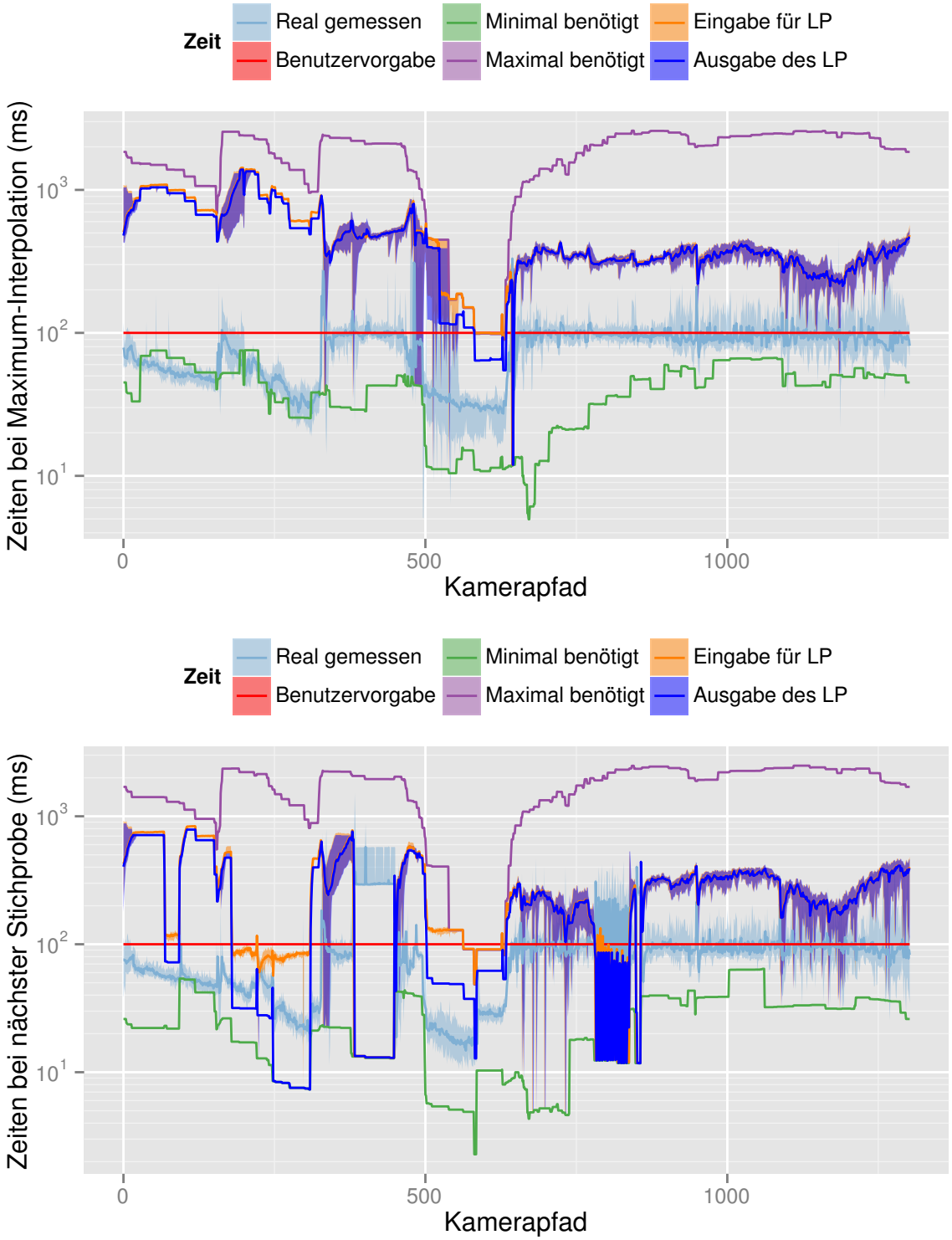


Abbildung 5.11: Darstellung der Zeiten mit denen das Multi-Algorithmen-Rendering intern die Gesamtlaufzeit regelt. Abgetragen ist zum einen die gemessene Laufzeit zur Darstellung des Bildes, sowie verschiedene Summen der Werte aus der Vorverarbeitung.

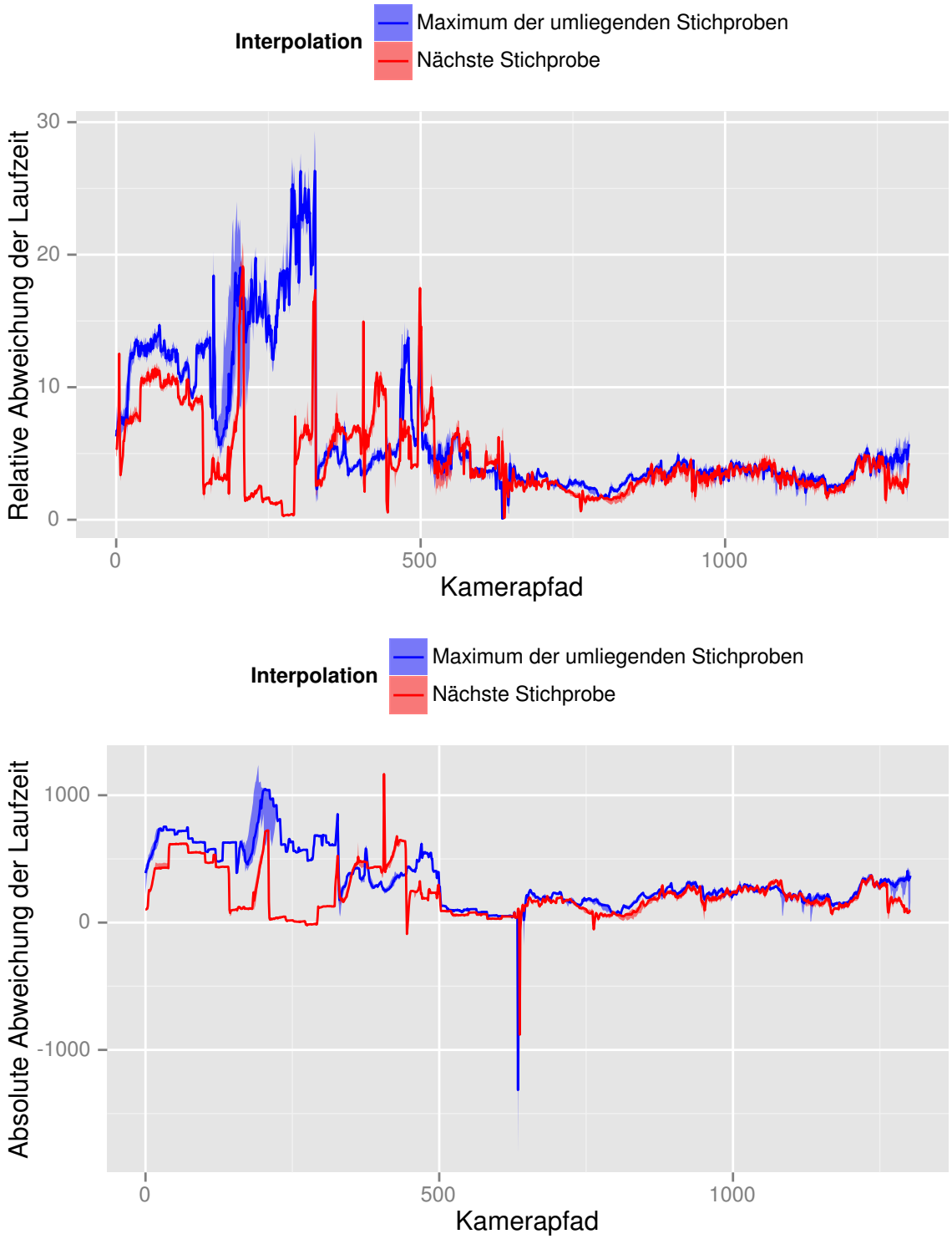


Abbildung 5.12: Darstellung der relativen und absoluten Abweichung zwischen gemessener Laufzeit und der auf Basis der Vorverarbeitungswerte geschätzten Laufzeit bei Verwendung unterschiedlicher Interpolationsmethoden.

Erklärung für die in den Laufzeit- und Bildfehlermessungen auftretenden Messspitzen sein könnten.

Fragestellung Wie verhalten sich die vom Multi-Algorithmen-Rendering intern benutzten Schätzwerte und insbesondere deren relative bzw. absolute Abweichung zu den real gemessenen Werten?

Durchführung der Messung Gemessen wurden folgende Werte jeweils für beide Interpolationsmethoden bei 1000 Stichproben:

- Als Minimum die Summe der jeweils schnellsten Algorithmen pro Region.
- Als Maximum die Summe der jeweils langsamsten Algorithmen pro Region.
- Der Wert, der als Eingabe für das LP dient.
- Die Summe der Werte der Algorithmen, die vom LP pro Region ausgewählt wurden (Ausgabe des LP).
- Die real gemessene Laufzeit des Multi-Algorithmen-Renderings.

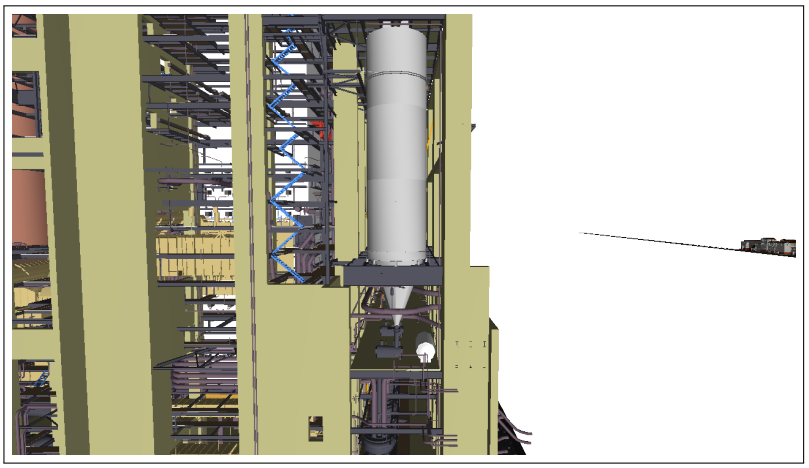
Die Ergebnisse dieser Messung sind in Abbildung 5.11 dargestellt. Abbildung 5.12 zeigt sowohl die relative Abweichung zwischen real gemessener Zeit und Ausgabe des LPs als auch die absolute Abweichung der Werte.

Auswertung Bei der Maximum-Interpolation ist die Ausgabe des LP lediglich in Bereichen, in denen ein korrektes Bild erzeugt wird, deutlich von der Eingabe zu unterscheiden. In den Bereichen, in denen approximiert werden muss, um die Benutzervorgabe für die Laufzeit einzuhalten, liegen die beiden Werte hingegen sehr dicht beieinander. Daher wurde auf eine weitergehende Analyse des LPs selbst verzichtet.

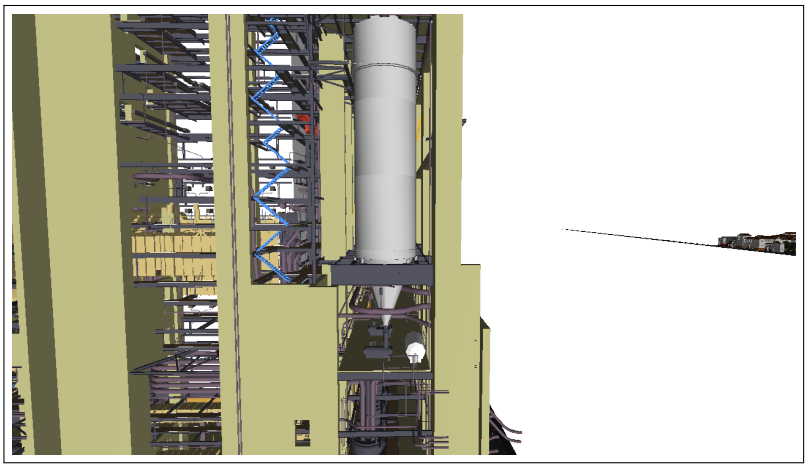
Bei der Interpolation mit Hilfe der nächstgelegenen Stichprobe sind die beiden Anomalien der Messung aus Abbildung 5.9 wiederzufinden. Bei der ersten, an der die gemessene Laufzeit kontinuierlich zu hoch ist, sind offensichtlich die Werte der Stichprobe für die aktuelle Kameraposition falsch. Der Regelkreis regelt (basierend auf der Stichprobe) die Zeit auf das Minimum zurück. Dies ist jedoch an der Position der Stichprobe eine Kombination von Algorithmen gewesen, welche an der aktuellen Position sehr langsam ist. Im zweiten Bereich, in dem die Laufzeit sehr stark fluktuiert, tun dies auch die Zeiten des LPs, welche wiederum gegen das Minimum schlagen. Erkennen kann man auch, dass in diesen Bereichen die berechnete Laufzeit unter der gemessenen liegt. Dies stellt ein generelles Problem für die Regelungsvariante mit absolutem Fehler dar, da dadurch der Wert für die Gesamtlaufzeit, der als Eingabe für das LP dient, negativ werden könnte, wenn er nicht durch das Minimum beschränkt wäre.

Vergleicht man die relative bzw. absolute Abweichung der Werte in Abbildung 5.12 mit den entsprechenden Laufzeitdiagrammen in Abbildung 5.9, so stellt man fest, dass die Positionen, an denen die Abweichung springt, überwiegend mit den Positionen übereinstimmen, an denen in der Laufzeitmessung Spitzen auftreten.

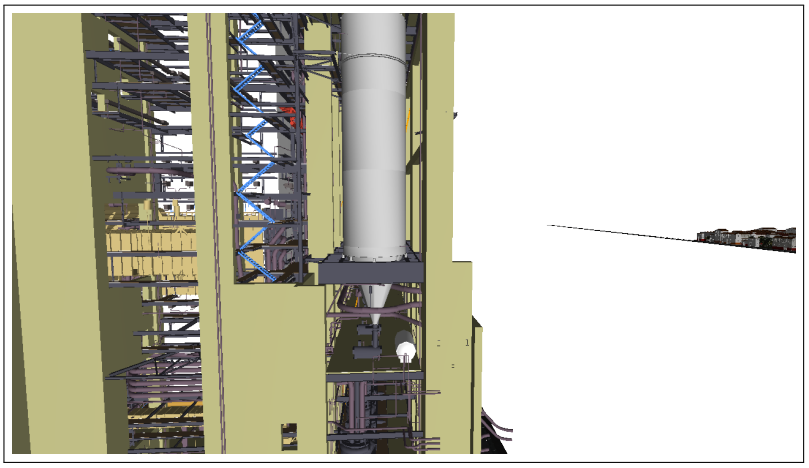
In dieser Messung fällt vor allem der negative Peak bei Messpunkt 636 auf. Hier wird die Laufzeit um mehr als eine Sekunde unterschätzt. Dies sollte jedoch eigentlich nicht möglich sein. Schaut man sich die bisherigen (und folgenden) Messungen an, so kann man diese Messspitze in sehr vielen Diagrammen wiederfinden, dort allerdings als positive Messspitze.



(a) Bild bei Messpunkt 635



(b) Bild bei Messpunkt 636



(c) Bild bei Messpunkt 637

Abbildung 5.13: Darstellung der gerenderten Bilder an den Messpunkten 635 – 637. Bei Bild 636 tritt eine Messspitze in nahezu allen Messungen auf, für die leider keine Erklärung gefunden wurde.

Diese extrem hohe Laufzeit tritt jedoch nur dann auf, wenn exakt dieser Kamerapfad mit exakt der in den Messungen benutzten Schrittweite abgelaufen wird. Insbesondere tritt die Spitze bereits in allen Messungen auf, in denen nur ein Algorithmus für die gesamte Szene benutzt wird. Daher ist es also kein Effekt des Multi-Algorithmen-Renderings, sondern ein generelles Problem. Dieser Effekt konnte nicht durch ein manuelles Navigieren in der Szene reproduziert werden. Selbst bei einem Ablaufen des Kamerapfades zur Laufzeit – mit hier nicht konstanter, sondern durch die Laufzeit bestimmter Schrittweite – trat die Spitze nicht auf. Es konnte leider keine Erklärung dafür gefunden werden. Die Bilder der Messpunkte 635 – 637 sind in Abbildung 5.13 dargestellt. Die Lucy befindet sich weit hinter der Kamera und wird somit aufgrund des Frustum-Cullings nicht dargestellt. Die Darstellung des Powerplants stellt kein Problem dar, unabhängig vom gewählten Algorithmus. Der Bereich von Pompeji, der in Bild 636 zusätzlich ins Bild kommt, ist minimal, und auch in Bild 637 noch sichtbar. Es ist nichts zu erkennen, das eine solch hohe Laufzeit erklären könnte.

5.4.6 Abweichung zwischen geschätzten und gemessenen Bildfehlerwerten

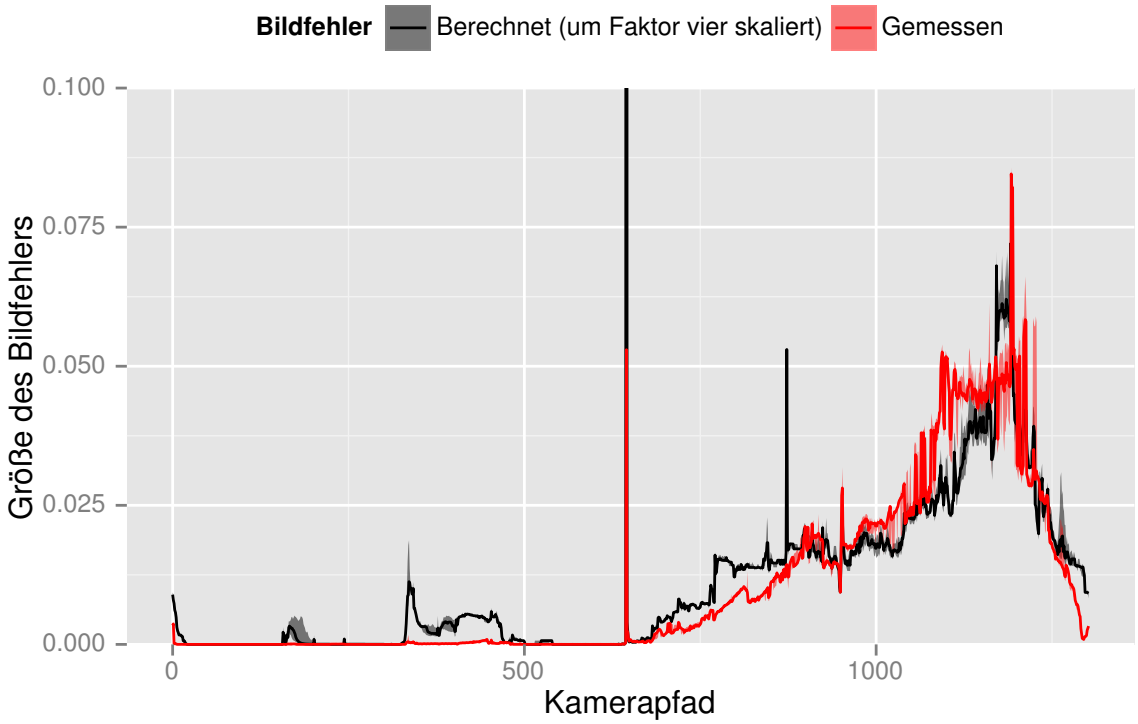


Abbildung 5.14: Darstellung der absoluten Abweichung zwischen gemessenem Bildfehler und dem auf Basis der Vorverarbeitungswerte geschätzten Bildfehler. Zur besseren Vergleichbarkeit wurden die Werte des geschätzten Fehlers mit vier multipliziert.

Der durch das Multi-Algorithmen-Rendering erzeugte Bildfehler hat im Gegensatz zur Laufzeit keinen Einfluss auf den Regelkreis. Dies ist auch nicht möglich, da zur Berechnung dieses Fehlers ein korrektes Bild benötigt wird. Nichts desto trotz ist dieser Fehler ein

wichtiger Punkt bei der Evaluierung, der bis hier noch nicht untersucht wurde. Es entsteht zwar nur dann ein Bildfehler, wenn eine korrekte Darstellung nicht möglich ist, allerdings stellt sich die Frage, ob der resultierende Bildfehler auch zu den in der Vorverarbeitung gemessenen Werten passt, da nur dann die Aussage, dass das Multi-Algorithmen-Rendering die Bildqualität optimiert, richtig ist.

Fragestellung Wie stark weicht der gemessene Bildfehler des gerenderten Bildes von der Summe der Schätzwerte aus der Vorverarbeitung ab?

Durchführung der Messung Die Ergebnisse der Messung sind in Abbildung 5.14 dargestellt. Bei der Darstellung wurden die Schätzwerte mit vier multipliziert, um die Ähnlichkeit der beiden Kurven besser beurteilen zu können.

Auswertung Da die dargestellten Kurven einen ähnlichen Verlauf zeigen, sind die Schätzwerte aus der Vorverarbeitung eine Unterschätzung. Die Werte sind durchschnittlich um Faktor vier kleiner als die gemessenen Werte im finalen Bild.

Dies liegt daran, dass das verwendete Abstandsmaß zur Bestimmung des Bildfehlers großflächige Fehler stärker bewertet als flächenmäßig kleine Fehler. Daher ist hier der Fehler aller Regionen insgesamt also größer als die Summe der Fehler der einzelnen Regionen. Das verwendete Abstandsmaß ist also keine Metrik, da die Dreiecksungleichung nicht gilt.

Dies sollte allerdings keinen Einfluss auf die Funktionsweise des Multi-Algorithmen-Renderings haben, da der Verlauf der Messkurven ähnlich ist und eine reine Skalierung der Fehlerwerte die Lösung des LPs nicht beeinflusst. Diese Änderung beeinflusst zwar die Zielfunktion und somit auch ihren Wert, allerdings keine der Bedingungen des LPs. Eine Skalierung der Zielfunktion um einen gleichmäßigen Faktor hat jedoch keine Auswirkung auf die Lösung eines LPs.

5.4.7 Welche Algorithmen werden genutzt?

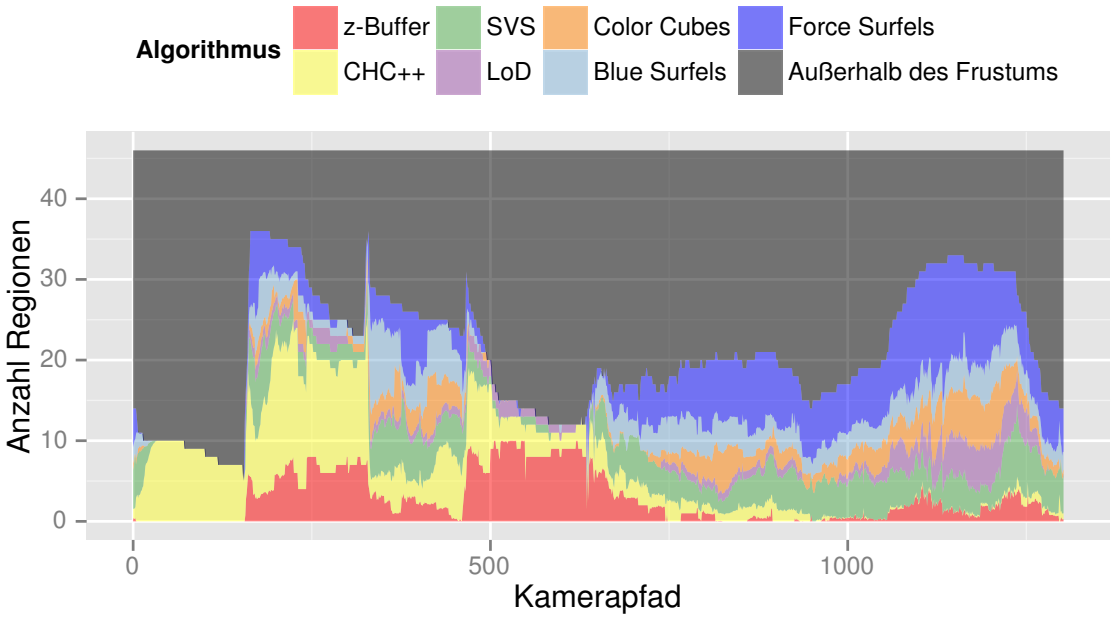
Hier wurde die Fragestellung betrachtet, wie häufig und für welche Regionen die einzelnen Algorithmen eingesetzt werden. Dazu wurden zwei Messungen durchgeführt: Zum einen wurde untersucht, für wie viele Regionen die Algorithmen pro Bild eingesetzt werden, und zum anderen, wie viele Pixel die Algorithmen zum Bild beitragen.

Die beiden Messungen wurden aus technischen Gründen nacheinander ausgeführt. Da das Multi-Algorithmen-Rendering eventuell im zweiten Durchlauf leicht andere Entscheidungen getroffen hat als im ersten, kann es Abweichungen geben. Dies gilt insbesondere für das Verhältnis zwischen dem SVS Algorithmus und dem Color-Cubes Algorithmus. Diese Abweichungen sollten jedoch nur gering sein und die Schlussfolgerungen nicht beeinflussen.

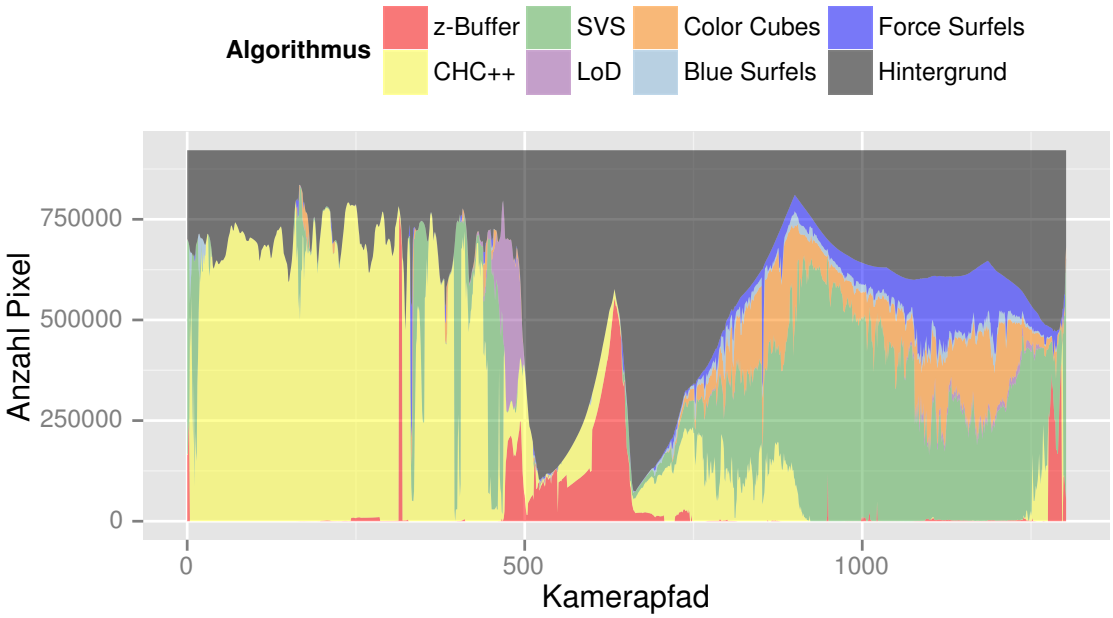
Fragestellung Für wie viele Regionen werden die benutzten Algorithmen eingesetzt?

Durchführung der Messung Gemessen wurde die Anzahl der Regionen, die mit den Algorithmen dargestellt werden. Regionen, die nicht im Frustum liegen, wurden dabei separat gezählt. Die Ergebnisse der Messung sind in Abbildung 5.15(a) dargestellt.

Auswertung Es ist deutlich zu erkennen, dass alle Algorithmen eingesetzt werden. Des Weiteren werden die approximativen Algorithmen, wie zu erwarten, überwiegend im



(a) Anzahl an Regionen, die mit den Algorithmen dargestellt wurde.



(b) Anzahl an Pixeln, die durch die Algorithmen erzeugt wurde.

Abbildung 5.15: Die Grafiken visualisieren, wie häufig das Multi-Algorithmen-Rendering die einzelnen Algorithmen einsetzt. Die Diagramme sind Stapel-Diagramme, der Messwert entspricht also der Höhe eines farbigen Bereiches. Die Position des Bereiches entlang der y-Achse hat keine Aussagekraft.

zweiten Teil des Pfades eingesetzt, auf dem sich die Kamera über der Szene befindet. Im ersten Teil des Kamerapfades überwiegen hingegen die Algorithmen, die keinen oder nur einen sehr kleinen Fehler erzeugen.

Fragestellung Wie viele Pixel werden mit den benutzten Algorithmen dargestellt?

Durchführung der Messung Gemessen wurde diesmal die Anzahl der Pixel, mit denen die Algorithmen zum Bild beitragen. Hintergrundpixel wurden dabei separat gezählt. Die Ergebnisse der Messung sind in Abbildung 5.15(b) dargestellt.

Auswertung Es ist wiederum zu erkennen, dass alle Algorithmen eingesetzt werden. Diese Messung zeigt deutlich, dass auf dem ersten Teil des Kamerapfades fast ausschließlich Algorithmen zum Bild beitragen, die keinen oder nur einen sehr kleinen Fehler erzeugen. Selbst auf dem zweiten Teil des Pfades ist der Anteil der approximativen Darstellungsalgorithmen deutlich geringer als bei der vorherigen Messung.

Vergleicht man die beiden Messungen miteinander, so ist auffällig, dass auf dem ersten Teil des Pfades bei den Regionen alle Algorithmen auftauchen, während bei den Pixeln des Bildes fast ausschließlich der CHC++ auftritt. Dies liegt daran, dass hier sehr viele Regionen im Frustum verdeckt sind. Für diese Regionen legt das LP willkürlich Algorithmen fest, da alle Laufzeit- und Bildfehlerwerte null sind.

Das zweite was auffällt, ist der großflächige Einsatz des z-Buffer-Algorithmus ab ca. Messpunkt 500. Dieser Algorithmus wird hier überwiegend eingesetzt, um die Hülle des Powerplants darzustellen, während die Geometrie innerhalb des Kraftwerks mit dem CHC++-Verfahren dargestellt wird. Hier ist allerdings die Dauer der Darstellung unterhalb der eingestellten Grenze. Außerdem würde die Darstellung des kompletten Powerplants mit dem CHC++-Algorithmus nicht merklich länger dauern. Deshalb hat die Verwendung des z-Buffer-Algorithmus vermutlich keinen Einfluss auf das Verhalten des Multi-Algorithmen-Renderings.

Des Weiteren fällt auf, dass der Progressive-Blue-Surfels-Algorithmus nur für sehr geringe Bereiche des Bildes benutzt wird. Schaut man sich genauer an, an welchen Stellen dies geschieht, so erkennt man, dass der Algorithmus häufig an Stellen eingesetzt wird, an denen er ausschließlich Surfels darstellt. Dort verhält er sich also genauso wie der Forced-Surfels-Algorithmus.

Dies ist auf den ersten Blick verwunderlich, da der Progressive-Blue-Surfels-Algorithmus unter allen den zum Multi-Algorithmen-Rendering ähnlichsten Bildfehlerverlauf aufweist (siehe Abbildung 5.5) und überwiegend auch sehr schnell ist. Allerdings bildet das Multi-Algorithmen-Rendering an den meisten Positionen auch genau das Verhalten dieses Algorithmus nach. Auch hier werden projiziert große Objekte besser dargestellt als kleine. Der Vorteil des Multi-Algorithmen-Renderings ist hier jedoch der, dass es die Algorithmen nicht an *ungeeigneten* Stellen einsetzt.

Ungeeignete Stellen sind für den Progressive-Blue-Surfels nur Positionen in der Nähe der Lucy, sowie solche sehr knapp über der Stadt. Das heißt jedoch nicht, dass er an allen anderen Stellen die beste Wahl ist. Für Regionen mit einer sehr großen projizierten Größe, die aber nicht zu komplex sind (also ohne Lucy), zeigt beispielsweise das Spherical-Visibility-Sampling ein besseres Kosten-Nutzen-Verhältnis. Daher wird dieser Algorithmus im Nahbereich bevorzugt. Erst wenn die projizierte Größe klein wird, rechnet sich der Aufwand des Spherical-Visibility-Sampling-Verfahrens nicht mehr. Diese Grenze wird

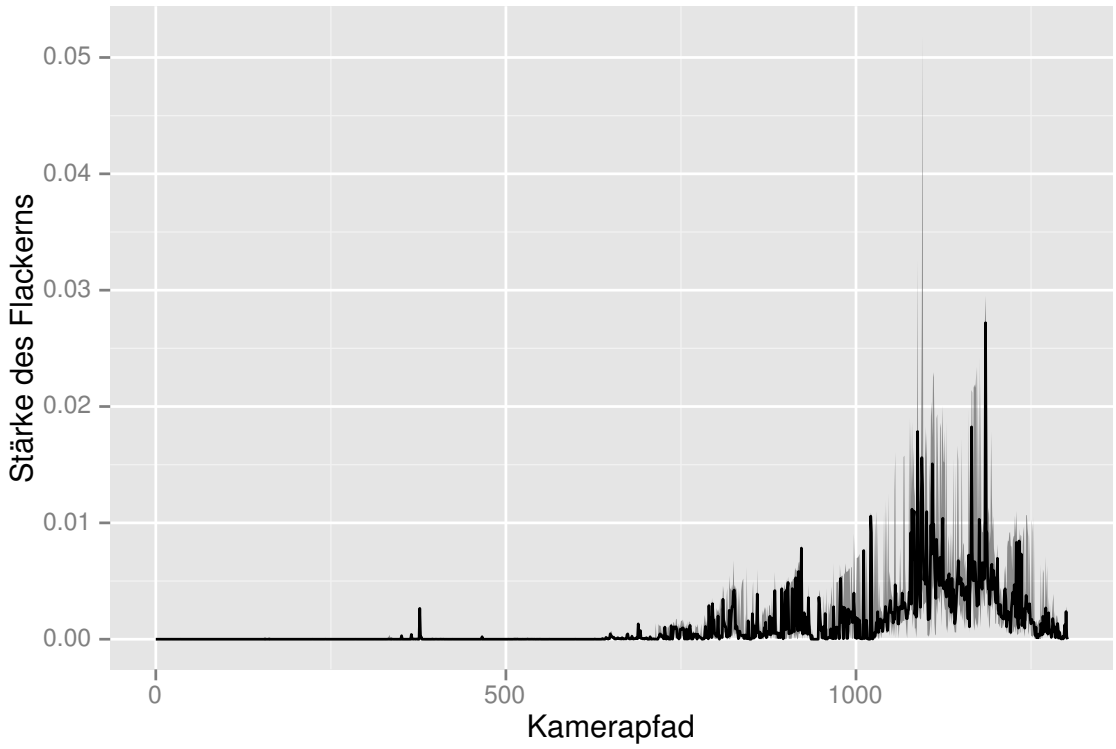


Abbildung 5.16: Darstellung des Unterschiedes zwischen je zwei aufeinanderfolgenden Bildern bei gleichem Frustum

anscheinend jedoch später erreicht als die Grenze, ab der die Progressive-Blue-Surfels-Methode nur noch Surfels darstellt. Daher würde ein Weglassen des Progressive-Blue-Surfels-Algorithmus vermutlich auch nicht viel am Verhalten des Multi-Algorithmen-Renderings ändern. In diesem Fall wäre es jedoch eventuell angebracht stattdessen über eine andere Parametrisierung des Algorithmus nachzudenken.

5.4.8 Flackern des Bildes beim Multi-Algorithmen-Rendering

Der größte Nachteil des Multi-Algorithmen-Renderings ist das Flackern des Bildes, welches durch das ständige Wechseln der Algorithmen entsteht. Daher wurde versucht dieses Flackern zu messen.

Fragestellung Wie groß ist der Unterschied zwischen aufeinanderfolgenden Bildern?

Durchführung der Messung Gemessen wurde der Unterschied zwischen je zwei aufeinanderfolgenden Bildern. Damit sich hierbei nicht der Sichtbereich des Benutzers und somit auch zwangsweise das Bild ändert, wurden hierzu an jeder Position des Kamerapfades zwei Bilder gerendert. Mit diesen wurde dann ein Bildvergleich durchgeführt. Das Ergebnis der Messung ist in Abbildung 5.16 dargestellt.

Auswertung Der Verlauf der Kurve sieht zunächst sehr gut aus, der Unterschied zwischen den Bildern ist durchgehend deutlich kleiner als der Unterschied zwischen den Bildern und einem korrekten Bild (siehe Abbildung 5.8).

Dies spiegelt jedoch absolut nicht das Empfinden eines Benutzers wieder. Der Effekt wird deutlich störender wahrgenommen als der eigentliche Fehler im Bild, da er dynamisch ist.

Zur Veranschaulichung des Effekts kann man sich eine Uhr vorstellen. Dazu seien noch zwei approximative Darstellungsalgorithmen gegeben. Der erste stellt die Ziffern und Zeiger korrekt dar und ersetzt das hübsche Gehäuse der Uhr durch eine einfarbige Box. Dadurch entsteht ein hoher, großflächiger Bildfehler. Der zweite Algorithmus stellt das Gehäuse korrekt dar, lässt jedoch reihum jeweils entweder die Zeiger oder je drei aufeinanderfolgende Ziffern aus. Dadurch entstehen nur kleine lokal beschränkte Bildfehler, die jedoch dynamisch sind. Selbst wenn man die Uhr nicht ablesen, sondern nur anschauen möchte, ist für einen Betrachter der kleine, dynamische Fehler des zweiten Algorithmus deutlich störender als der große, statische Fehler des ersten Algorithmus. Dies gilt insbesondere dann, wenn der Benutzer sich eigentlich etwas anderes ansehen möchte und kein Interesse an der Uhr selbst hat.

Der Effekt tritt beim Multi-Algorithmen-Rendering hauptsächlich dann störend in Erscheinung, wenn zwischen den Surfel-Algorithmen und den restlichen gewechselt wird. Dies liegt hauptsächlich daran, dass bei den zurzeit noch in der Entwicklung befindlichen Surfel-Algorithmen die Darstellung generell ein wenig zu hell ist. Dieser Mangel kann jedoch vermutlich in zukünftigen Versionen dieses Algorithmus behoben werden.

Ein Verzicht auf den Forced-Surfels-Algorithmus ist hingegen nicht denkbar, solange nicht ein anderer Algorithmus entwickelt wird, welcher ein ebenso gutes Verhältnis von Bildfehler und Laufzeit für entfernte oder komplexe Regionen hat.

5.4.9 Zeit zum Lösen des Optimierungsproblems

Ein beschränkender Faktor für die Laufzeit des Multi-Algorithmen-Renderings könnte die Zeit sein, die benötigt wird, um das LP zu lösen. Daher wurde untersucht, wie viel Zeit hierfür benötigt wird und an welchen Stellen diese Zeit zu lang ist. Die Lösung des Optimierungsproblems erfolgt zwar parallel, könnte allerdings zum beschränkenden Faktor werden, wenn sie höher als die vom Benutzer vorgegebene Zeit zur Darstellung des Bildes ist.

Fragestellung Wie lange dauert das Lösen des Optimierungsproblems?

Durchführung der Messung Gemessen wurde die Zeit, die benötigt wird, um eine optimale Lösung für das LP zu berechnen. Die Ergebnisse der Messung sind in Abbildung 5.17 dargestellt.

Auswertung Die Messung zeigt deutlich, dass in diesem Fall die zur Berechnung der Lösung benötigte Zeit durchweg unter der eingestellten Zeit von 100 ms zur Darstellung der Szene liegt. Somit ist hier das Optimierungsproblem nicht der beschränkende Faktor. Die Messung zeigt allerdings auch, dass die benötigte Zeit an Stellen, wo beinahe die gesamte Szene sichtbar ist und somit das LP für alle Regionen gelöst werden muss, stark steigt. Dies lässt vermuten, dass es Bereiche innerhalb der Szene geben könnte, an denen die Zeit noch weiter ansteigt als in dieser Messung. Insbesondere ist zu erwarten, dass bei geringeren Vorgabewerten für die Darstellungszeit, wie

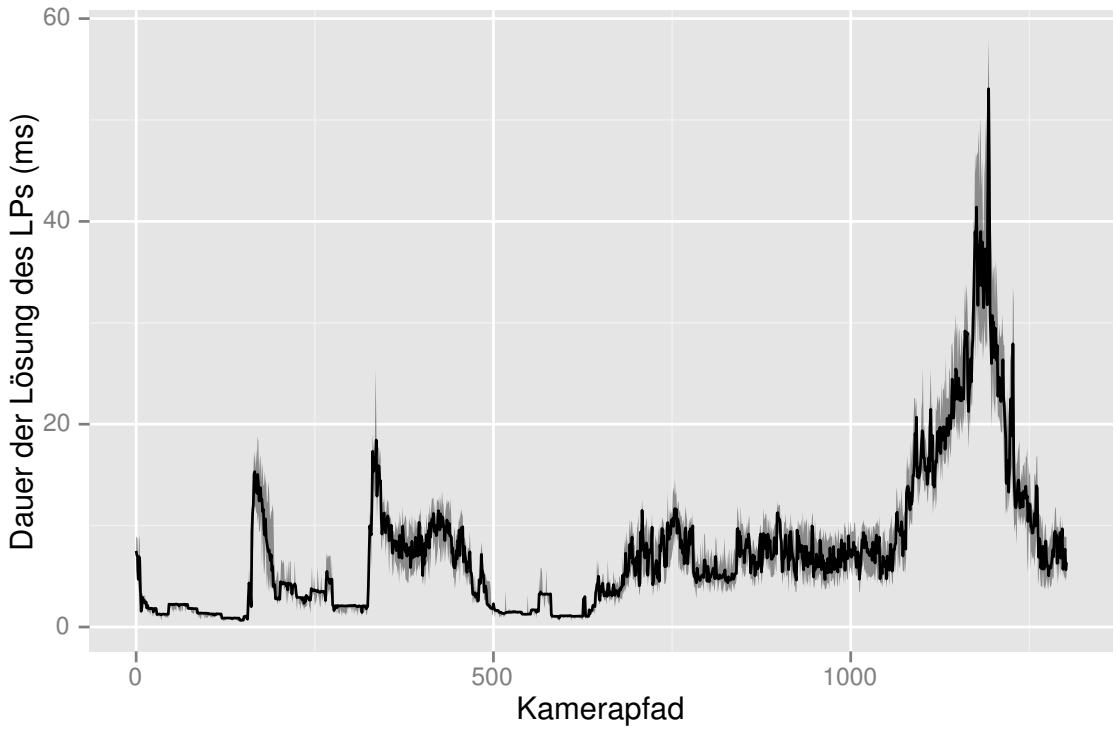


Abbildung 5.17: Messung der Zeit, die benötigt wird um das Optimierungsproblem zu lösen

beispielsweise nur 50 ms, die Berechnung der Lösung des Optimierungsproblems zum beschränkenden Faktor wird. Dies wird in nachfolgenden Messungen (Abschnitt 5.5.3 und 5.5.4.3) näher untersucht.

5.5 Evaluierung mit Hilfe von Szenen-Eigenschafts-Funktionen

In der Arbeit von Jähn u. a. [Jäh+13] wird vorgeschlagen, zusätzlich zu einer Evaluierung anhand eines Kamerapfades auch eine Evaluierung anhand von Szenen-Eigenschafts-Funktionen durchzuführen. Eine Szenen-Eigenschafts-Funktion ist dabei beispielsweise die Laufzeit eines Renderingalgorithmus. Hierbei wird ein Bereich der Szene stichprobenartig abgetastet, um eine flächen- bzw. volumendeckende Approximation einer solchen Funktion zu erhalten.

Im Unterschied zu einem Kamerapfad wird hierbei jedoch kein sich kontinuierlich bewegender Benutzer simuliert, sondern stattdessen, je nach Art der Durchführung, ein stillstehender bzw. in der Szene umherspringender Benutzer.

Im Folgenden wird diese Technik angewandt, um einen Verlauf des Bildfehlers, der Darstellungszeit sowie der zum Lösen des LPs benötigten Zeit zu erhalten. Außerdem werden Schlussfolgerungen bezüglich der Anzahl benötigter Stichproben in der Vorverarbeitung des Multi-Algorithmen-Renderings gezogen.

Da ein stillstehender Benutzer in der Praxis wesentlich häufiger vorkommt, wurde ein solcher für die nachfolgenden Messungen genutzt. Das Multi-Algorithmen-Rendering

verkraftet allerdings auch ein Umherspringen des Benutzers innerhalb der Szene sehr gut. Dies hat lediglich ein langsames und/oder ein fehlerhaftes Bild zur Folge, da der Regelkreis sehr schnell reagiert.

Zur Simulation eines stillstehenden Betrachters wird für jede Messung das Bild zehnmal dargestellt. Die ersten beiden Messwerte werden verworfen (Aufheizphase) und über die restlichen acht Messwerte wird der Durchschnitt gebildet. Das Sampling-Verfahren selbst dreht die Kamera pro Stichprobe in sechs Richtungen und bildet das Maximum. Der Messwert ist also der durchschnittliche Wert der ungünstigsten Richtung.

In der Arbeit über Szenen-Eigenschafts-Funktionen wird weiterhin Vorgeschlagen die Messungen mit einem vergrößerten Kameraöffnungswinkel durchzuführen. Dadurch überlappen sich die Frusta und Artefakte in den Messergebnissen wie in Abbildung 5.18 als schräge Linien zu erkennen werden vermieden. Daher wurden die folgenden Messungen zweimal durchgeführt: einmal mit einem Kameraöffnungswinkel von 90 ° und ein weiteres Mal mit einem Kameraöffnungswinkel von 120 °.

Die vorherigen Messungen haben gezeigt, dass Bereiche in denen eine hohe Verdeckung besteht und in denen eine korrekte Darstellung der Szene möglich ist, für eine Analyse eher uninteressant sind. In diesen Bereichen wird fast ausnahmslos ein korrektes Bild angezeigt, und auch das Lösen des Optimierungsproblems stellt kein Problem dar. Daher wurde für die folgenden Messungen eine Ebene knapp über die Dächer von Pompeji gelegt, innerhalb derer die Stichproben gezogen werden. Dort sind zum einen stets sehr viele Regionen sichtbar und zum anderen können nahe Regionen nicht stark approximiert werden, ohne einen großen Fehler zu erzeugen. Somit stellt diese Ebene einen Worst-Case sowohl für Laufzeit und Bildfehler als auch für die Lösung des LPs dar.

5.5.1 Bildfehler

Zunächst wurde der Bildfehler in der untersuchten Ebene betrachtet:

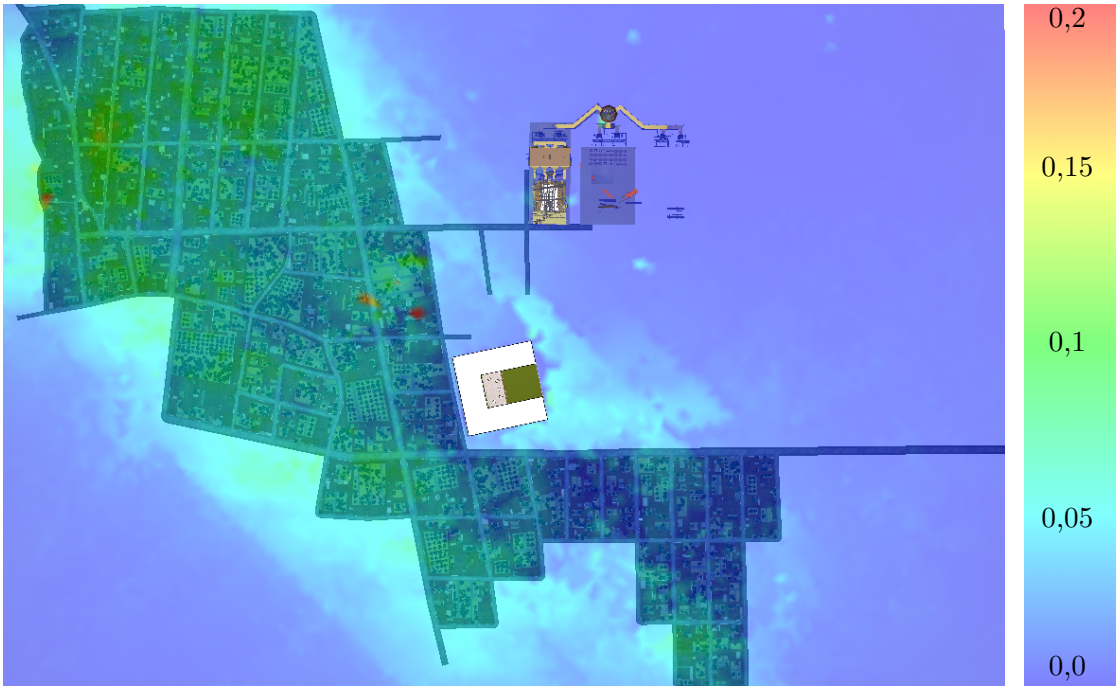
Fragestellung Wie groß ist der durchschnittliche Bildfehler in der gewählten Ebene?

Durchführung der Messung Gemessen wurde der durchschnittliche Bildfehler. Die Messung wurde zweimal durchgeführt, dabei wurde der Öffnungswinkel der Kamera variiert. Die Ergebnisse sind in Abbildung 5.18 dargestellt.

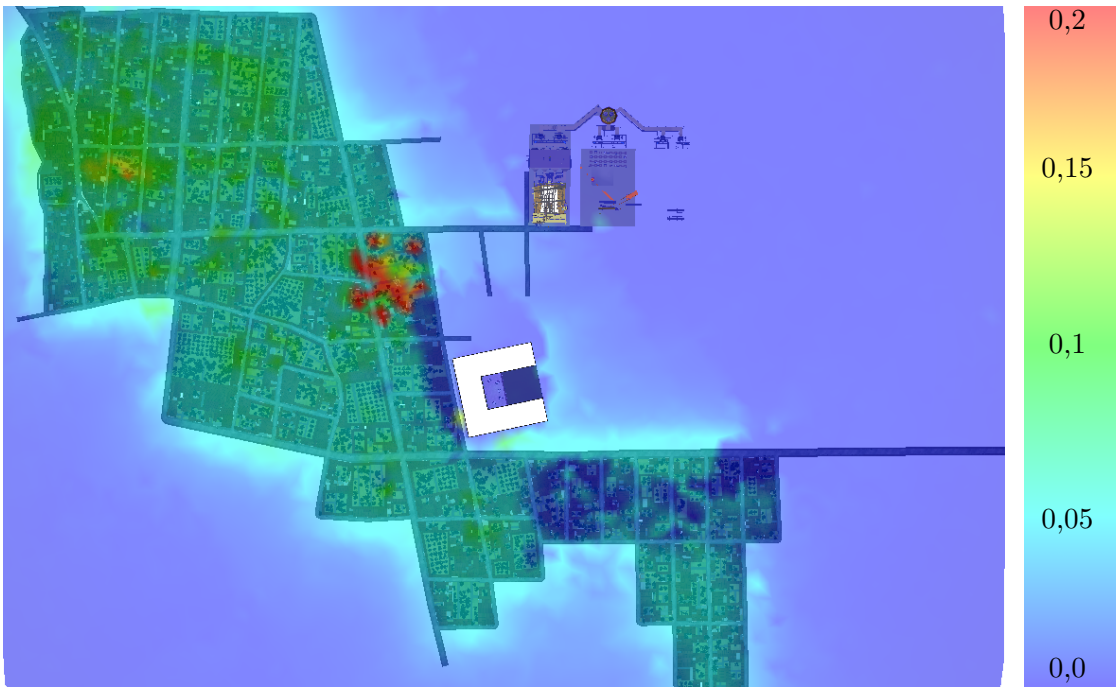
Auswertung Bei der Messung mit einem Kameraöffnungswinkel von 90 ° sind deutlich Artefakte (diagonale Linien) zu erkennen, welche in der Messung mit 120 ° nicht auftreten. Dies liegt daran, dass bei einem Kameraöffnungswinkel von 90 ° die Frusta disjunkt sind, während sie sich bei 120 ° überlappen.

In der unteren Messung mit 120 ° Öffnungswinkel ist jedoch ein roter Bereich, um die Lucy zu erkennen. Dort ist der Bildfehler signifikant größer als im Rest der Ebene. Hier sind die Fehlerwerte auch deutlich größer als auf dem zuvor betrachteten dem Kamerapfad. (Erklärung folgt bei der nächsten Messung)

Der restliche Bereich der Messung zeigt jedoch, dass der Bildfehler sich im gleichen Bereich bewegt, wie bei den Messungen auf dem Kamerapfad. Wie zu erwarten ist er über der Stadt größer als außerhalb.

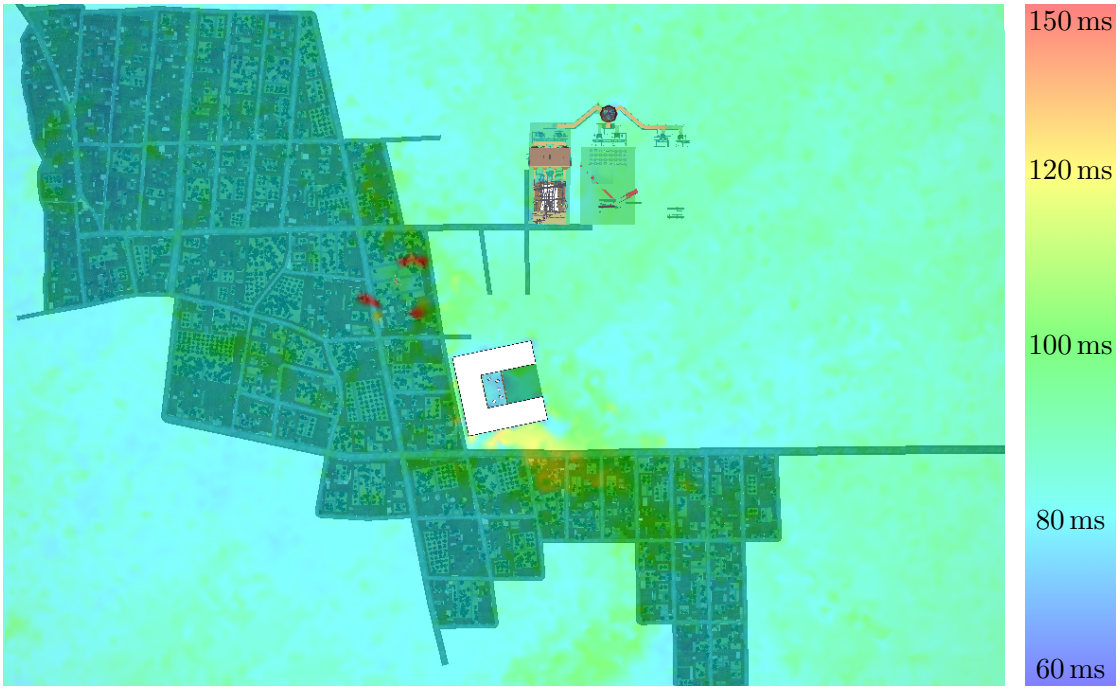


(a) Bildfehler bei 90° Kameraöffnungswinkel

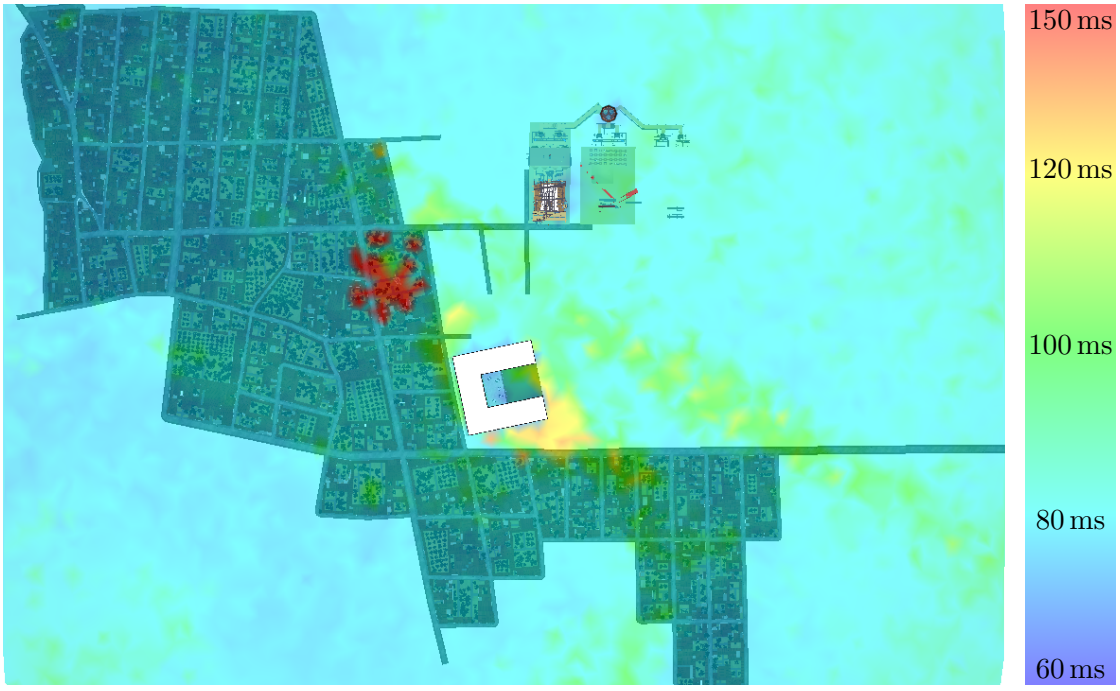


(b) Bildfehler bei 120° Kameraöffnungswinkel

Abbildung 5.18: Verlauf des Bildfehlers innerhalb einer Ebene etwas oberhalb von Pompeji bei verschiedenen Kameraöffnungswinkeln.

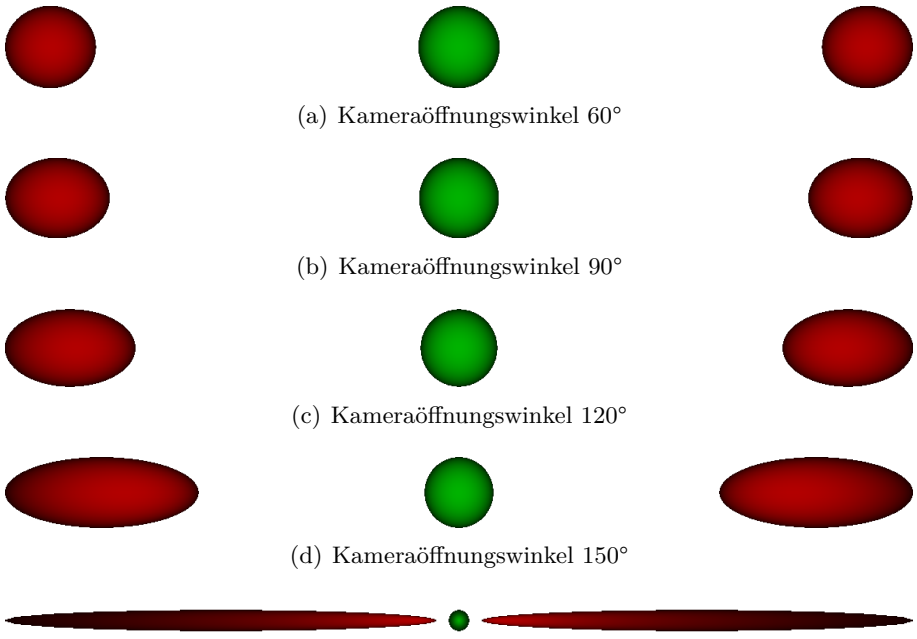


(a) Darstellungsdauer bei 90° Kameraöffnungswinkel



(b) Darstellungsdauer bei 120° Kameraöffnungswinkel

Abbildung 5.19: Verlauf der Darstellungsdauer innerhalb einer Ebene etwas oberhalb von Pompeji bei verschiedenen Kameraöffnungswinkeln.



(a) Kameraöffnungswinkel 60°

(b) Kameraöffnungswinkel 90°

(c) Kameraöffnungswinkel 120°

(d) Kameraöffnungswinkel 150°

(e) Kameraöffnungswinkel 179°

Abbildung 5.20: Mit steigendem Kameraöffnungswinkel werden Objekte am Rand des Bildes verzerrt und überproportional groß dargestellt. Alle dargestellten Objekte sind Kugeln.

5.5.2 Dauer der Darstellung

Als zweites wurde die benötigte Zeit für die Darstellung des Bildes untersucht.

Fragestellung Wie lange dauert die Darstellung eines Bildes innerhalb der gewählten Ebene?

Durchführung der Messung Gemessen wurde die durchschnittliche Laufzeit. Die Messung wurde wiederum zweimal durchgeführt, dabei wurde der Öffnungswinkel der Kamera variiert. Abbildung 5.19 stellt das Ergebnis dieser Messung dar.

Auswertung Auch diese Messung zeigt bei einem Kameraöffnungswinkel von 120 ° deutlich einen roten Bereich um die Lucy, in dem die Laufzeit sehr hoch ist. Hier wurde die vorgegebene Laufzeit um ein Vielfaches überschritten und liegt deutlich außerhalb der angegebenen Skala. Das Multi-Algorithmen-Rendering war also nicht in der Lage, die vorgegeben Zeit zur Darstellung der Szene einzuhalten, während die restlichen Messwerte überwiegend erwartungskonform sind. Lediglich das Modell des ZM 1 Gebäudes wirft einen gelben Schatten, der allerdings noch im akzeptablen Bereich liegt. Dies liegt vermutlich daran, dass dieses Gebäude Fenster hat. Die Darstellung transparenter Flächen wurde jedoch bei der Entwicklung des Multi-Algorithmen-Renderings vollständig ignoriert, da auch die benutzten Algorithmen Probleme mit solchen Objekten aufweisen.

Die beiden vorangegangenen Messungen haben deutliche Probleme des Verfahrens bei einem Kameraöffnungswinkel von 120 ° aufgezeigt. Diese konnten zunächst nicht nachvoll-

zogen werden, da der Effekt nur dann auftritt, wenn sich die Lucy am Rand des Bildes befindet. Solange sich die Lucy in der Mitte des Bildes befindet, funktioniert das Verfahren einwandfrei. Dies liegt daran, dass bei einem großen Kameraöffnungswinkel Objekte am Rand des Bildes verzerrt und überproportional groß dargestellt werden. Abbildung 5.20 stellt diesen Effekt dar. Dieser hat zur Folge, dass in diesem Fall die Laufzeitwerte aus der Vorverarbeitung für den Discrete-Level-of-Detail-Algorithmus um einen zusätzlichen Faktor zwei von der wirklich benötigten Zeit zur Darstellung der Lucy abweichen, da dieser Algorithmus aufgrund der übergroßen Darstellung des Objekts entscheidet, die nächstbessere Reduktionsstufe des Modells zu verwenden. Da andere Algorithmen davon jedoch nicht betroffen sind, weil ihre Laufzeit nicht (so stark) von der projizierten Größe eines Objekts abhängig ist, tritt hierdurch ein Ungleichgewicht zwischen den Algorithmen auf. Dies hat zur Folge, das beim Wechseln des Algorithmus aufgrund der hohen Laufzeit, die Abweichung zwischen geschätzter und gemessener Laufzeit entgegengesetzt springt, was wiederum bewirkt, dass der Algorithmus sofort wieder zurück gewechselt wird. Dadurch entsteht im Durchschnitt sowohl eine hohe Laufzeit als auch ein hoher Bildfehler, da das Multi-Algorithmen-Rendering aufgrund der fehlerhaften Werte unsinnige Entscheidungen trifft.

5.5.3 Lösung des Optimierungsproblems

Als drittes wurde mit dieser Technik die zur Lösung des Optimierungsproblems benötigte Zeit untersucht, bei der die Messungen anhand des Kamerapfades bereits nahegelegt haben, dass die ungünstigsten Positionen nicht im Pfad enthalten sind.

Fragestellung Wie lange dauert das Lösen des Optimierungsproblems innerhalb der gewählten Ebene?

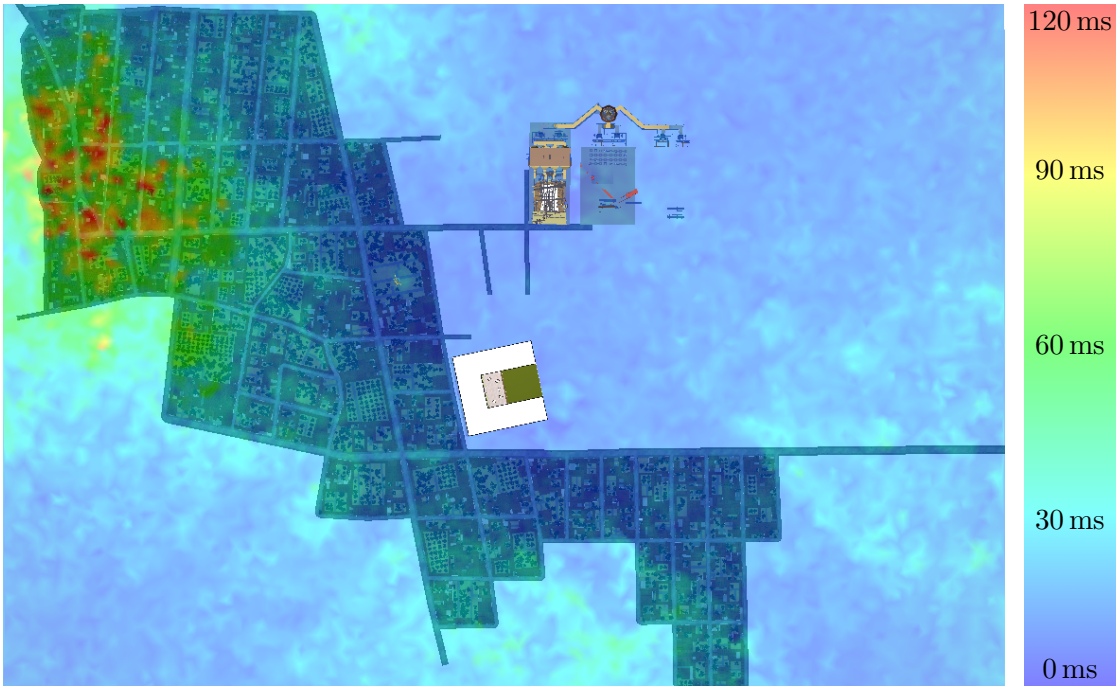
Durchführung der Messung Gemessen wurde die Berechnungsdauer für eine optimale Lösung des LPs. Die Ergebnisse sind in Abbildung 5.21 dargestellt.

Auswertung Es ist ganz klar ersichtlich, dass in dem Bereich, in dem die Lösung des LPs am meisten Zeit beanspruchte, der Kamerapfad nicht oberhalb, sondern innerhalb der Stadt verläuft. Somit ist der Pfad zumindest für diese Messung nicht repräsentativ gewählt worden.

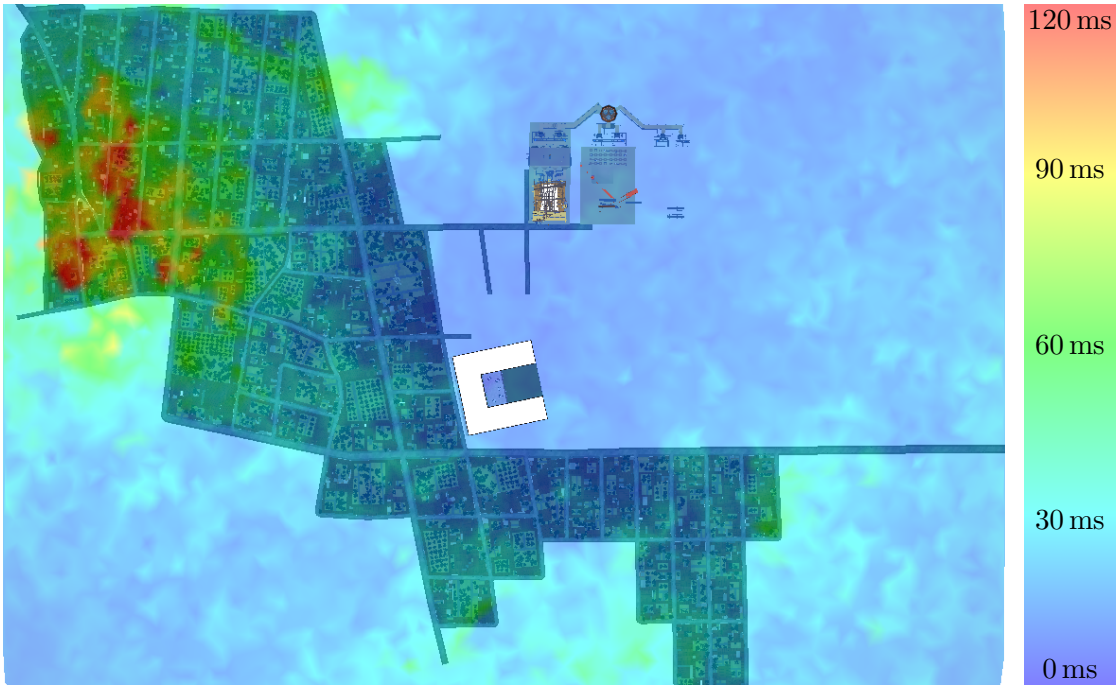
In den roten Bereichen liegen die Messwerte über 100 ms und somit über der Benutzervorgabe für die Darstellungszeit eines Bildes. Damit ist in diesem Bereich das Optimierungsproblem der beschränkende Faktor für die Laufzeit des Multi-Algorithmen-Renderings. Allerdings liegen die Werte auch hier noch in dem Bereich, in dem die Zeit zur Darstellung der Szene schwankt. Daher liegt die Laufzeit hier noch im akzeptablen Bereich.

5.5.4 Verschiedene Sollwerte für die Laufzeit

Bisher wurden alle Messungen mit einer Laufzeitvorgabe von 100 ms für die Darstellung des Bildes durchgeführt. Diese Zeit wird für die nachfolgenden Messungen auf 50 ms reduziert. Es werden wiederum der Bildfehler sowie die Laufzeiten zur Darstellung des Bildes und zum Lösen des Optimierungsproblems untersucht. Allerdings wird lediglich ein Kameraöffnungswinkel von 90 ° betrachtet.



(a) Lösungsdauer bei 90° Kameraöffnungswinkel.



(b) Lösungsdauer bei 120° Kameraöffnungswinkel.

Abbildung 5.21: Verlauf der Zeit zur Lösung des LPs innerhalb einer Ebene etwas oberhalb von Pompeji bei verschiedenen Kameraöffnungswinkeln.

5.5.4.1 Darstellungsdauer

Fragestellung Wie verhält sich die Laufzeit des Multi-Algorithmen-Renderings, wenn die Vorgabezeit zur Darstellung auf 50 ms reduziert wird?

Durchführung der Messung Gemessen wurde die durchschnittliche Zeit zur Darstellung der Szene. Die Ergebnisse sind in Abbildung 5.22(a) dargestellt.

Auswertung Erneut erkennt man einen Bereich um die Lucy, in dem die vorgegebene Laufzeit deutlich überschritten wird. Auch hier liegen die Werte ein Vielfaches über dem Sollwert.

Die Ursache ist in diesem Fall jedoch eine andere: Aufgrund des niedrigeren Sollwerts wird in dieser Messung nicht der Discrete-Level-of-Detail-Algorithmus für die Lucy benutzt, sondern der Progressive-Blue-Surfels-Algorithmus. Für die Messung wurden die Daten der Vorverarbeitung mit nur 1.000 Stichproben benutzt. Hier liegt jedoch keine der wenigen Stichproben dicht an der Lucy, so dass dieses Modell in der Vorverarbeitung stets durch Surfels ersetzt wurde. Der rote Bereich in der Messung ist allerdings der, in dem der Algorithmus aufgrund der projizierten Größe die Originalgeometrie des Modells darstellt. Dies geschieht durchgehend, da an den Stichprobenpositionen in der Vorverarbeitung der Algorithmus stets der schnellste war.

In den restlichen Bereichen der Messung sind die Werte wie erwartet: Die relative Abweichung der Laufzeit vom Sollwert unterscheidet sich nicht wesentlich von der Messung mit 100 ms.

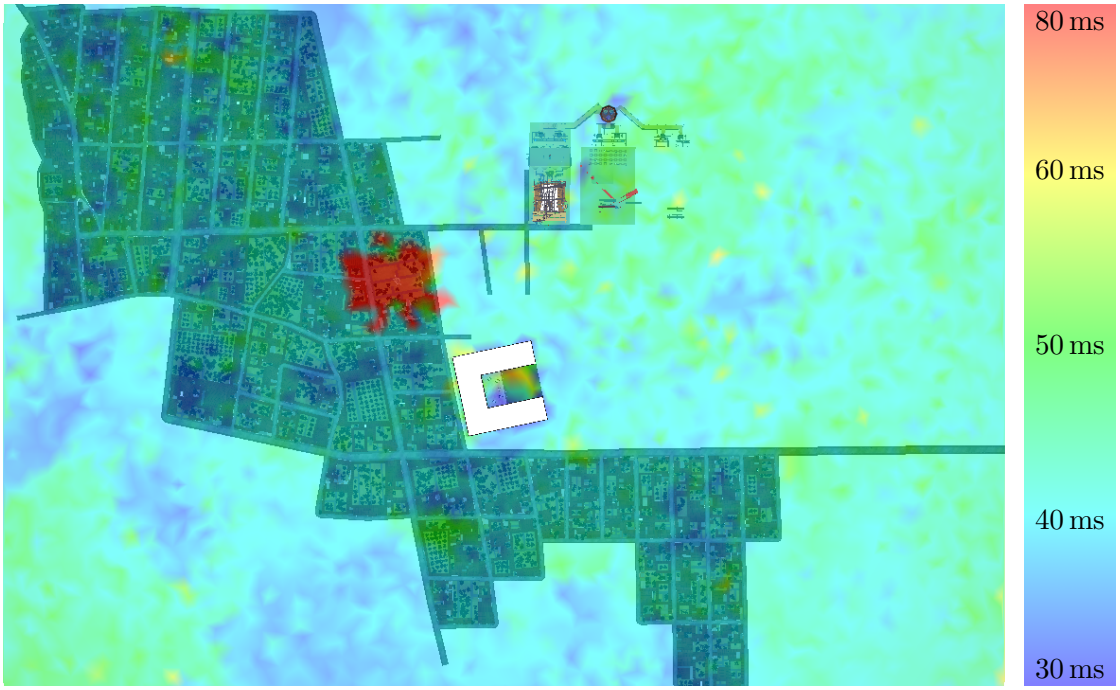
Zur Verifizierung der angegebenen Ursache für die stark erhöhte Laufzeit, wurde die Messung für diesen Bereich auf Grundlage der Vorverarbeitung mit 13.000 Stichproben wiederholt.

Fragestellung Wie verhält sich die Laufzeit des Multi-Algorithmen-Renderings bei 13.000 in der Vorverarbeitung gezogenen Stichproben, wenn die Vorgabezeit zur Darstellung auf 50 ms reduziert wird?

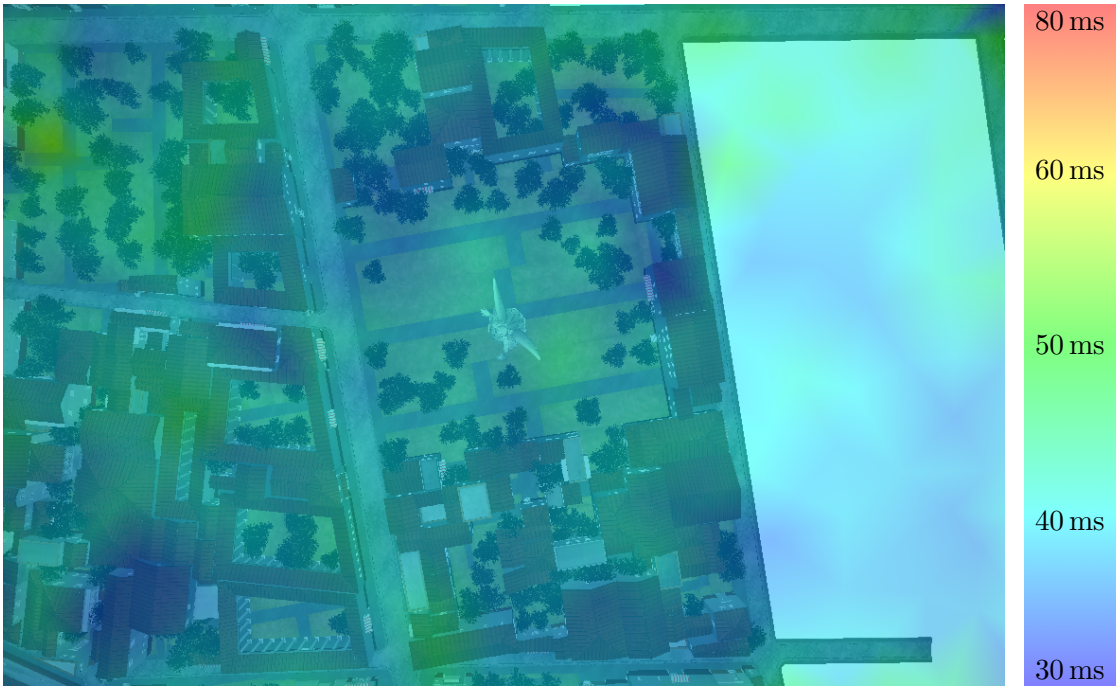
Durchführung der Messung Gemessen wurde die durchschnittliche Zeit zur Darstellung der Szene. Die Ergebnisse sind in Abbildung 5.22(b) dargestellt. Es wurde ausschließlich der Bereich der Szene untersucht, in dem in der vorherigen Messung der rote Bereich lag.

Auswertung Der Bereich mit der hohen Darstellungszeit ist in dieser Messung nicht mehr vorhanden. Dies liegt daran, dass nun einige Stichproben existieren, an deren Position für den Progressive-Blue-Surfels-Algorithmus in der Vorverarbeitung eine Laufzeit gemessen wurde, die der Darstellung des Originalmodells der Lucy entspricht. Die verwendete Maximum-Interpolation sorgt dafür, dass das Modell in dem besagten Bereich nie mit diesem Algorithmus dargestellt wird. Bei ausschließlicher Benutzung der nächstliegenden Stichprobe würde jetzt vermutlich ein verformter roter Kreis um die Lucy entstehen.

Die beiden vorherigen Messungen haben deutlich gezeigt, dass im überwiegenden Bereich der Szene 1.000 Stichproben ausreichend sind. Es gibt jedoch auch Bereiche, in denen



(a) Darstellungsdauer bei einem Sollwert von 50 ms und 1.000 Stichproben.



(b) Darstellungsdauer bei einem Sollwert von 50 ms und 13.000 Stichproben.

Abbildung 5.22: Einhaltung von verschiedenen vorgegebenen Zeiten zur Darstellung der Szene. In Bild (b) ist lediglich der Bereich der Szene dargestellt, der in Bild (a) rot ist.

es notwendig ist, mehr Stichproben zu ziehen. Daher scheint ein adaptives Verfahren beim Ziehen der Stichproben angeraten. Dies wurde jedoch in dieser Arbeit nicht mehr untersucht.

5.5.4.2 Bildfehler

Fragestellung Wie verändert sich der Bildfehler, wenn die Zeit zur Darstellung des Bildes auf 50 ms reduziert wird?

Durchführung der Messung Gemessen wurde der Bildfehler bei einem vorgegebenen Sollwert für die Darstellungszeit von 50 ms. Die Ergebnisse sind in Abbildung 5.23(b) dargestellt.

Auswertung Vergleicht man die beiden Bilder miteinander, so ist ersichtlich, dass durch die reduzierte Darstellungsdauer der Bildfehler wie erwartet flächendeckend ansteigt. Auch in dieser Messung verschwindet der rote Bereich um die Lucy, wenn man mehr Stichproben benutzt.

5.5.4.3 Lösungsdauer des Optimierungsproblems

Fragestellung Wie lange dauert die Berechnung der optimalen Lösung des LPs, wenn der Sollwert für die Dauer der Darstellung auf 50 ms reduziert wird?

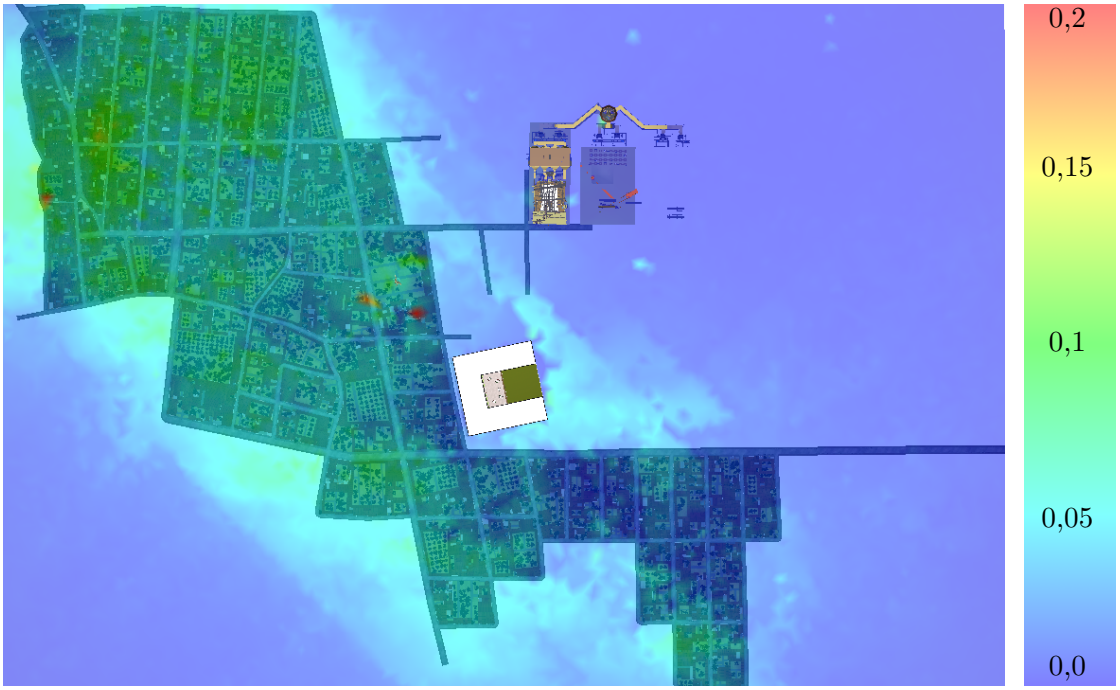
Durchführung der Messung Gemessen wurde die zur Lösung des Optimierungsproblems benötigte Zeit. Abbildung 5.24(a) zeigt die Ergebnisse der Messung mit einer zu der Messung mit 100 ms identischen Skala (vgl. Abb. 5.21). Abbildung 5.24(b) zeigt die gleichen Ergebnisse mit einer auf 50 ms angepassten Skala.

Auswertung Vergleich man das obere Bild mit dem entsprechenden aus Abbildung 5.21 so ist eine leichte Verschiebung der Werte in Richtung höherer Laufzeiten zu erkennen. Die Lösung des Problems scheint also etwas zeitaufwendiger zu sein als zuvor.

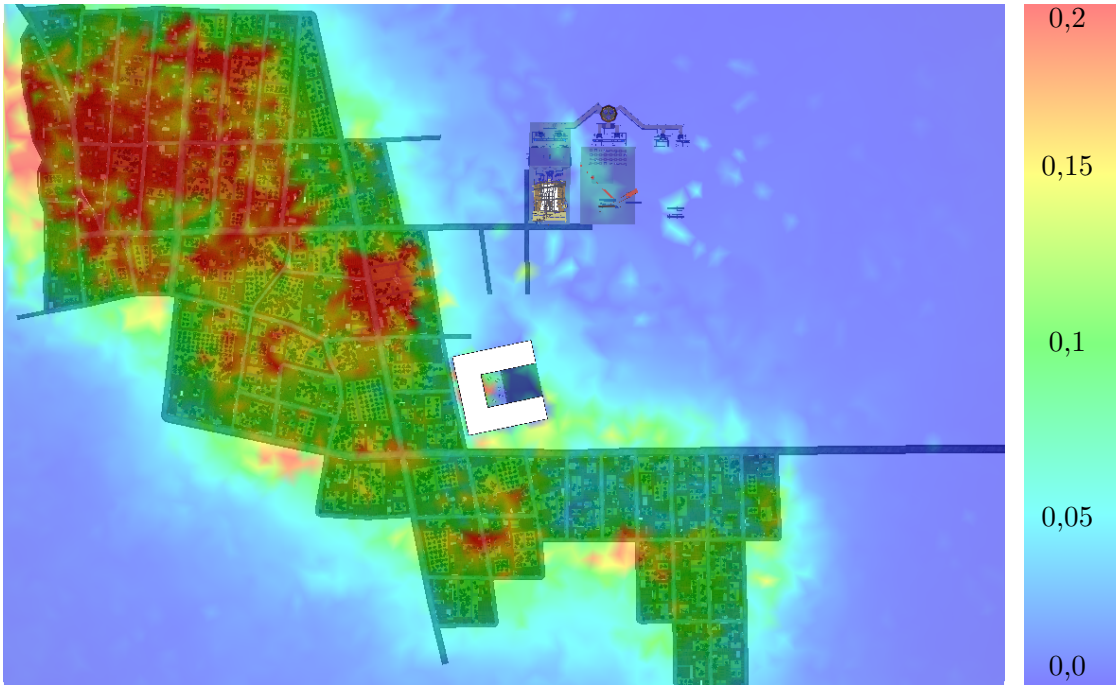
Das untere Bild mit einer entsprechend des Sollwerts zur Dauer der Darstellung angepassten Skala zeigt, dass jetzt in großen Bereichen der Szene die Lösung des LPs der beschränkende Faktor ist. Das heißt, obwohl die vorherige Messung gezeigt hat, dass das Multi-Algorithmen-Rendering in der Lage ist die Szene entsprechend schnell darzustellen, erfolgt diese Darstellung in großen Bereichen der Szene trotzdem nur mit zehn Bildern pro Sekunde, da zwischen den einzelnen Bildern auf die Lösung des LPs gewartet werden muss.

5.6 Zusammenfassung der Ergebnisse

Die Evaluierung der Vorverarbeitung hat ergeben, dass der durch das Multi-Algorithmen-Rendering zusätzlich benötigte Speicher im Verhältnis zu anderen Algorithmen sehr klein ist. Jedoch müssen beim Multi-Algorithmen-Rendering die Vorverarbeitungen sämtlicher benutzter Verfahren durchgeführt werden. Der gesamte Speicherverbrauch aller benutzten Verfahren liegt bei ca. 80 % des Speicherverbrauchs der verwendeten Szene. Er wird Maßgeblich bestimmt durch reduzierte Versionen der Modelle in der Szene. Diese belegen bereits ca. 50 % des zusätzlich benötigten Speichers.

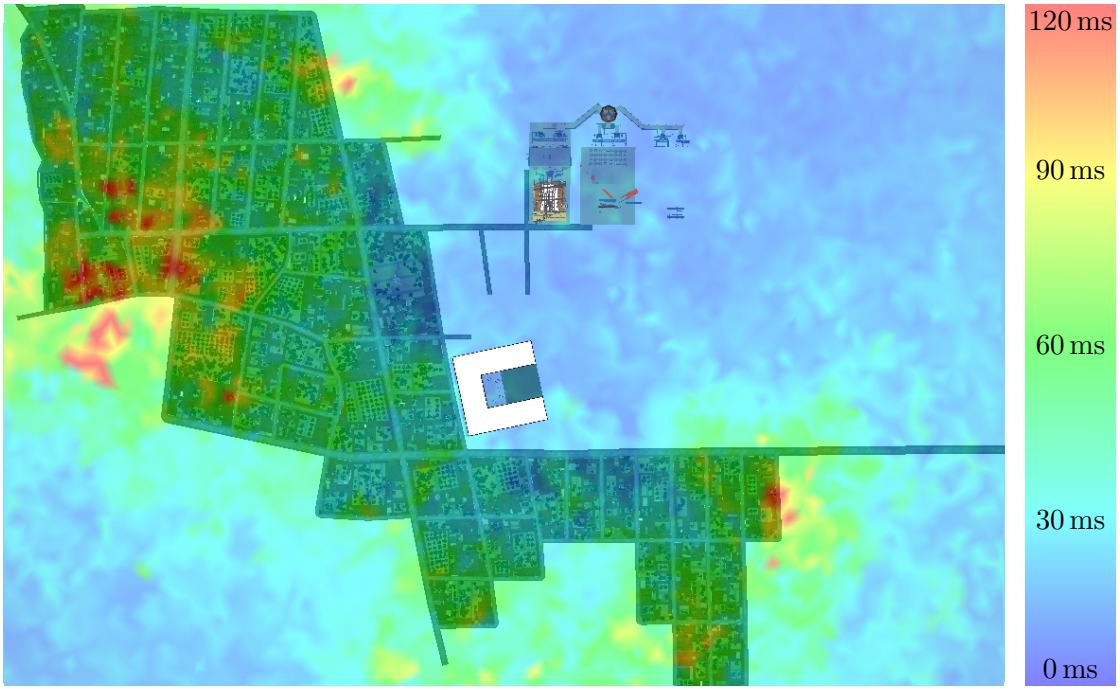


(a) Bildfehler bei einer Laufzeitvorgabe von 100 ms.

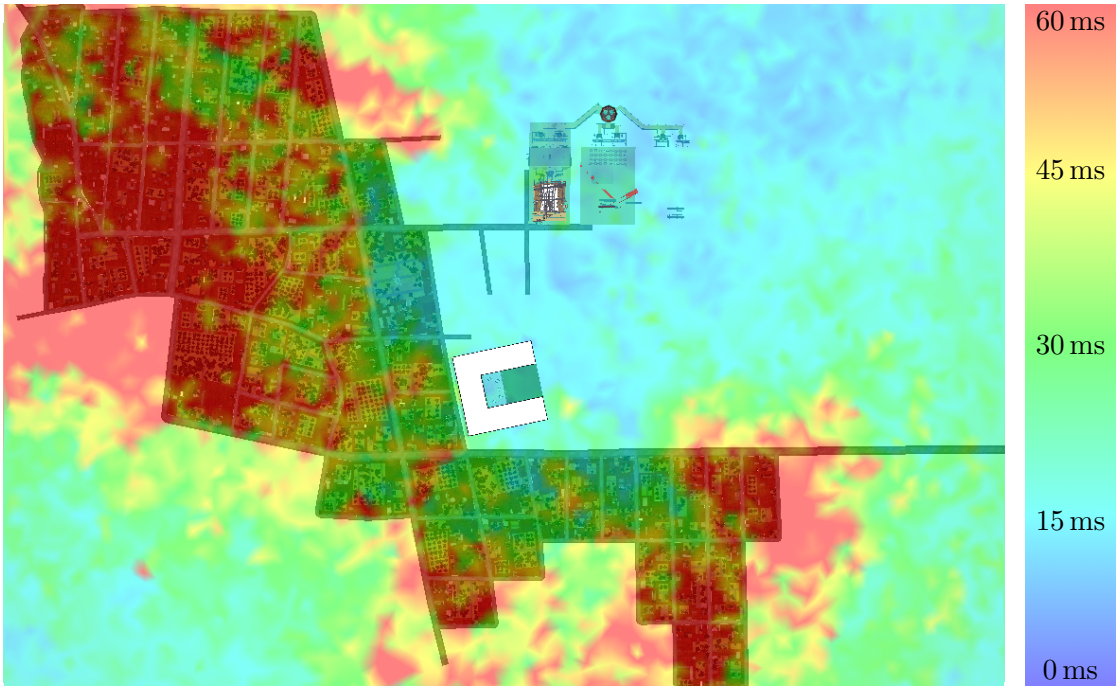


(b) Bildfehler bei einer Laufzeitvorgabe von 50 ms.

Abbildung 5.23: Bildfehler bei verschiedenen vorgegebenen Zeiten zur Darstellung der Szene. Die Farben repräsentieren die absoluten Fehlerwerte.



(a) Lösungsdauer des Optimierungsproblems bei Verwendung der gleichen Skala wie in Abbildung 5.21.



(b) Lösungsdauer des Optimierungsproblems bei einer an den Sollwert von 50 ms angepassten Skala.

Abbildung 5.24: Laufzeit zur Lösung des Optimierungsproblems bei einer vorgegebener Zeiten zur Darstellung der Szene von 50 ms. Der Unterschied zwischen den Bildern ist eine veränderte Farbskala.

Die Laufzeit der Vorverarbeitung ist abhängig von der Anzahl erzeugter Regionen, der Anzahl und Art der benutzten Verfahren sowie der Anzahl gezogener Stichproben. Im Gegensatz zum Speicherverbrauch wird die Laufzeit maßgeblich durch das Multi-Algorithmen-Rendering bestimmt. Im verwendeten Beispielszenario kann die Vorverarbeitung mit 1.000 Stichproben über Nacht durchgeführt werden. Daher ist das Verfahren praktisch einsetzbar.

Beim Verhalten des Multi-Algorithmen-Renderings während des Walkthroughs hat sich herausgestellt, dass das Multi-Algorithmen-Rendering den Anspruch der Minimierung des Bildfehlers bei vorgegebener Laufzeit generell erfüllt. Allerdings gibt es dabei eine untere Schranke für die vorgegebene Laufzeit. Bei geringeren Vorgaben erfolgt die Darstellung aufgrund der zur Lösung des Optimierungsproblems benötigten Zeit nicht mehr in der erwünschten Zeit. Die Laufzeit zur Darstellung des Bildes hält die gewählten Schranken allerdings sehr wohl ein. Dies wird jedoch auch nur bis zu der Laufzeit so sein, die benötigt wird um die Regionen mit dem jeweils schnellsten Algorithmus darzustellen. Die Analyse des Bildfehlers hat ergeben, dass dieser nur dann entsteht, wenn eine korrekte Darstellung der Szene nicht möglich ist.

Bei den verschiedenen Regelungs- und Interpolationsmethoden, die für das Multi-Algorithmen-Rendering umgesetzt wurden, hat sich herausgestellt, dass die Kombination aus Interpolation über das Maximum der umliegenden Stichproben mit absoluter Regelung den anderen Varianten durchgängig überlegen ist.

Des Weiteren wurde festgestellt, dass 1.000 gleichmäßig verteilte Stichproben im überwiegenden Teil der Szene ausreichend sind. Es gibt jedoch auch Bereiche, in denen mehr oder geschickter gewählte Stichproben benötigt werden. Daher ist eine adaptive Wahl der Stichprobenpositionen angeraten.

Der größte Nachteil des Verfahrens ist ein, durch das Wechseln der Darstellungsalgorithmen bedingtes, Flackern des Bildes. Dies konnte durch die Messungen nicht zufriedenstellend erfasst werden. Hier passen die absoluten Werte der Messung nicht zum subjektiven Empfinden eines Benutzers, da die Messung die Dynamik des Fehlers nicht ausreichend widerspiegelt.

6 Fazit und Ausblick

Insgesamt wurde mit dem Multi-Algorithmen-Rendering ein praxistaugliches Verfahren zur Darstellung von heterogenen (und auch homogenen) Szenen entwickelt.

Die Evaluierung der Vorverarbeitung hat ergeben, dass diese bei ca. 1.000 gezogenen Stichproben „über Nacht“ durchgeführt werden kann. Der Speicherverbrauch ist dabei im vergleich zu anderen Verfahren mit Vorverarbeitung sehr gering.

Das Laufzeitverhalten des Multi-Algorithmen-Renderings wurde umfassend evaluiert. Es hat sich herausgestellt, dass das Verfahren in der Lage ist, die Laufzeit in einem akzeptablen Bereich um den Vorgabewert zu regeln und dabei die Bildqualität optimiert. Wenn eine korrekte Darstellung der Szene innerhalb der vorgegebenen Zeit möglich ist, so erzeugt auch das Multi-Algorithmen-Rendering fast nie einen Bildfehler.

Das Verfahren ist nicht nur zur reinen Darstellung von Szenen, sondern auch als Hilfsmittel bei der Evaluierung neuer Algorithmen und beim Design von Szenen einsetzbar. Dies gilt insbesondere auch in Kombination mit den zur Evaluierung benutzten Szenen-Eigenschafts-Funktionen.

Folgende Aspekte des Multi-Algorithmen-Renderings können in zukünftigen Arbeiten optimiert bzw. erweitert werden:

Aufteilung der Szene in Regionen Da die Anzahl an Regionen ein maßgeblicher Faktor für die Komplexität des Optimierungsproblems ist, ist es wünschenswert die Anzahl an Regionen gering zu halten. Dies könnte beispielsweise erreicht werden, indem man dem Benutzer bessere Werkzeuge zur Aufteilung zur Verfügung stellt.

Alternativ könnte man auch versuchen ein Verfahren zu entwickeln, welches in der Lage ist, eine Szene automatisch in geeignete Regionen aufzuteilen. Dazu müsste man jedoch zunächst die Bedeutung von „homogen“ formal exakt spezifizieren und alle benutzen Algorithmen diesbezüglich analysieren.

Auf jeden Fall stellt eine Analyse des Einflusses unterschiedlicher Aufteilungen der Szene in Regionen, die in dieser Arbeit ausgespart wurde, ein interessantes Thema für nachfolgende Arbeiten dar.

Geeignete Stichproben Die Evaluierung hat gezeigt, dass in vielen Bereichen wenige Stichproben ausreichen, in einigen jedoch nicht. Daher erscheint eine ungleichmäßige Verteilung dieser Stichproben sinnvoll. Es sollte genauer untersucht werden, in welchen Bereichen viele Stichproben benötigt werden. Es ist auch denkbar, dass wenige aber geschickt platzierte Stichproben (z. B. je eine mittig über den Regionen) auch in diesen Bereichen ausreichen.

Verringerung des Flackereffekts Das durchgehende Flackern des Bildes in Bereichen, wo approximiert werden muss, ist der größte Mangel am Multi-Algorithmen-Rendering. Dieses kann, wie bereits erwähnt wurde, durch eine Verbesserung der Bildqualität der Surfel-Algorithmen verringert werden.

Des Weiteren könnte man noch versuchen das Optimierungsproblem nicht ständig, sondern nur dann neu zu lösen, wenn die Bildrate deutlich vom Sollwert abweicht. Die hohe Schwankung der Laufzeit bei einigen benutzten Algorithmen (z. B. CHC++) könnte hier jedoch zu Problemen führen.

Beschränkung der Größe des LPs Der Einsatz des Multi-Algorithmen-Renderings ist vorwiegend bei Szenen sinnvoll, welche nicht in Echtzeit korrekt darstellbar sind. Wenn man allerdings davon ausgeht, dass jedes Bild fehlerhaft ist, dann kann man argumentieren, dass es bei komplexen Regionen, welche nur wenige Pixel zum Bild beitragen, akzeptabel ist, diese immer approximiert darzustellen. Nimmt man dies in Kauf, so kann das LP auf einen Nahbereich um die Kamera beschränkt werden.

Da sinnvolle Regionen aus den in Abschnitt 3.4.1 beschriebenen Gründen eine sowohl nach oben als auch nach unten beschränkte Größe haben, ist die Anzahl der Regionen in diesem Bereich konstant. Dieser Nahbereich hätte eine Größe, die ungefähr der in dieser Arbeit verwendeten Szene entspricht, da hier das LP zumindest bei einem Sollwert für die Darstellungsdauer von 100 ms noch schnell genug lösbar ist. Alle Regionen, die außerhalb dieses Bereichs liegen, können Problemlos mit Hilfe des Forced-Surfels-Algorithmus dargestellt werden. Dadurch steigt also auch die Größe, der durch das Multi-Algorithmen-Rendering darstellbaren Szenen, um ein Vielfaches.

Beschleunigung der Vorverarbeitung Mit Hilfe der gleichen Vorgehensweise kann auch in der Vorverarbeitung die benötigte Zeit zum Ziehen einer Stichprobe beschränkt werden. Dies liegt daran, dass die Regionen, die nicht im Nahbereich liegen, auch nicht getestet werden müssen.

Update der Vorverarbeitungsdaten Für einen produktiven Einsatz des Verfahrens ist eine Implementierung der Vorverarbeitung wünschenswert, welche es erlaubt bei einer lokal beschränkten Änderung der Szene die in der Vorverarbeitung ermittelten Werte zu aktualisieren. Dies sollte konzeptuell keinerlei Probleme bereiten, ist jedoch vermutlich technisch aufwendig.

Wichtige Regionen Es sind Szenarien denkbar, bei denen manche Bereiche der Szene anwendungsbedingt wichtiger sind als andere. Durch eine entsprechende Skalierung der Fehlerwerte könnte man erreichen, dass bestimmte Bereiche der Szene in besserer Qualität als andere dargestellt werden.

Literatur

[Ack+12] Marcel R. Ackermann, Johannes Blömer, Daniel Kuntze und Christian Sohler. „Analysis of Agglomerative Clustering“. In: *Algorithmica* (Dez. 2012). DOI: 10.1007/s00453-012-9717-4.

[AHH08] Tomas Akenine-Möller, Eric Haines und Naty Hoffman. *Real-Time Rendering*. 3. Aufl. Wellesley, MA, USA: A K Peters, Ltd., 2008.

[Ali+99] Daniel Aliaga u. a. „MMR: an interactive massive model rendering system using geometric and image-based acceleration“. In: *Proceedings of the 1999 symposium on Interactive 3D graphics*. (Atlanta, Georgia, USA). I3D '99. New York, NY, USA: ACM, 1999, S. 199–206. DOI: 10.1145/300523.300554.

[ARBJ90] John M. Airey, John H. Rohlf und Frederick P. Brooks Jr. „Towards image realism with interactive update rates in complex virtual building environments“. In: *Proceedings of the 1990 symposium on Interactive 3D graphics*. (Snowbird, Utah, United States). I3D '90. New York, NY, USA: ACM, 1990, S. 41–50. DOI: 10.1145/91385.91416.

[Are08] Stephan Arens. „Culling unter Verwendung hierarchischer Cluster“. Diplomarbeit. Universität Paderborn, Okt. 2008.

[Ben75] Jon Louis Bentley. „Multidimensional binary search trees used for associative searching“. In: *Communications of the ACM* 18.9 (Sep. 1975), S. 509–517. DOI: 10.1145/361002.361007.

[Bit+04] Jiří Bittner, Michael Wimmer, Harald Piringer und Werner Purgathofer. „Coherent Hierarchical Culling: Hardware Occlusion Queries Made Useful“. In: *Computer Graphics Forum* 23.3 (Sep. 2004). Proceedings of Eurographics 2004, S. 615–624. DOI: 10.1111/j.1467-8659.2004.00793.x.

[Bur81] Peter J. Burt. „Fast filter transform for image processing“. In: *Computer Graphics and Image Processing* 16.1 (Mai 1981), S. 20–51. DOI: 10.1016/0146-664X(81)90092-7.

[Cat74] Edwin Earl Catmull. „A subdivision algorithm for computer display of curved surfaces“. Diss. Salt Lake City, UT, USA: Department of Computer Science, University of Utah, Dez. 1974.

[Cha+96] Bradford Chamberlain, Tony DeRose, Dani Lischinski, David Salesin und John Snyder. „Fast rendering of complex environments using a spatial hierarchy“. In: *Proceedings of Graphics Interface 1996*. GI '96. Toronto, Ont., Canada: Canadian Information Processing Society, Mai 1996, S. 132–141.

[Coh+03] Daniel Cohen-Or, Yiorgos Chrysanthou, Cláudio T. Silva und Frédo Durand. „A survey of visibility for walkthrough applications“. In: *IEEE Transactions on Visualization and Computer Graphics* 9.3 (2003), S. 412–431. DOI: 10.1109/TVCG.2003.1207447.

[Del34] Boris N. Delaunay. „Sur la sphère vide“. In: *Izvestia Akademii Nauk SSSR, Otdelenie Matematicheskikh i Estestvennykh Nauk* 6 (Okt. 1934), S. 793–800.

[Eik+13] Benjamin Eikel, Claudius Jähn, Matthias Fischer und Friedhelm Meyer auf der Heide. „Spherical Visibility Sampling“. In: *Computer Graphics Forum* 32.4 (Juli 2013). Proceedings of the 24th Eurographics Symposium on Rendering, S. 49–58. DOI: 10.1111/cgf.12150.

[EJP11] Benjamin Eikel, Claudius Jähn und Ralf Petring. „PADrend: Platform for Algorithm Development and Rendering“. In: *Augmented & Virtual Reality in der Produktentstehung*. Hrsg. von Jürgen Gausemeier, Michael Grafe und Friedhelm Meyer auf der Heide. Bd. 295. HNI-Verlagsschriftenreihe. Heinz Nixdorf Institut, Universität Paderborn, Mai 2011, S. 159–170.

[FKN80] Henry Fuchs, Zvi M. Kedem und Bruce F. Naylor. „On visible surface generation by a priori tree structures“. In: *Proceedings of the 7th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '80. New York, NY, USA: ACM, 1980, S. 124–133. DOI: 10.1145/800250.807481.

[Fle06] Martin Fleisz. „Spatial Partitioning Using an Adaptive Binary Tree“. In: Hrsg. von Michael Dickheiser. First Edition. Boston, MA, USA: Charles River Media, 2006. Kap. 5.2, S. 423–435.

[Flo+51] K. Florek, J. Lukaszewicz, J. Perkal, Hugo Steinhaus und S. Zubrzycki. „Sur la liaison et la division des points d'un ensemble fini“. fre. In: *Colloquium Mathematicae* 2.3-4 (1951), S. 282–285.

[FS93] Thomas A. Funkhouser und Carlo H. Séquin. „Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments“. In: *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*. (Anaheim, CA). SIGGRAPH '93. New York, NY, USA: ACM, 1993, S. 247–254. DOI: 10.1145/166117.166149.

[GH97] Michael Garland und Paul S. Heckbert. „Surface simplification using quadric error metrics“. In: *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '97. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1997, S. 209–216. DOI: 10.1145/258734.258849.

[GH98] Michael Garland und Paul S. Heckbert. „Simplifying surfaces with color and texture using quadric error metrics“. In: *Proceedings of the conference on Visualization '98*. VIS '98. Los Alamitos, CA, USA: IEEE Computer Society Press, Okt. 1998, S. 263–269. DOI: 10.1109/VISUAL.1998.745312.

[GKY08] Enrico Gobbetti, Dave Kasik und Sung-eui Yoon. „Technical strategies for massive model visualization“. In: *Proceedings of the 2008 ACM symposium on Solid and physical modeling*. New York, NY, USA: ACM, 2008, S. 405–415. DOI: 10.1145/1364901.1364960.

[Hop96] Hugues Hoppe. „Progressive meshes“. In: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. SIGGRAPH '96. New York, NY, USA: ACM, 1996, S. 99–108. DOI: 10.1145/237170.237216.

[Hun78] Gregory Michael Hunter. „Efficient computation and data structures for graphics“. Diss. Princeton, NJ, USA: Department of Electrical Engineering und Computer Science, Princeton University, 1978.

[Iba+78] Toshihide Ibaraki, Toshiharu Hasegawa, Katsumi Teranaka und Jiro Iwase. „The Multiple-Choice Knapsack Problem“. In: *Journal of the Operations Research Society of Japan* 21.1 (März 1978), S. 59–94.

[Jäh+13] Claudius Jähn, Benjamin Eikel, Matthias Fischer, Ralf Petring und Friedhelm Meyer auf der Heide. „Evaluation of Rendering Algorithms using Position-Dependent Scene Properties“. In: *Advances in Visual Computing*. Hrsg. von George Bebis u. a. Bd. 8033. Lecture Notes in Computer Science. Proceedings of the 9th International Symposium on Visual Computing (ISVC 2013). Springer Berlin Heidelberg, 2013, S. 108–118. DOI: 10.1007/978-3-642-41914-0_12.

[Lue+03] David Luebke, Martin Reddy, Jonathan d. Cohen, Amitabh Varshney, Benjamin Watson und Robert Huebner. *Level of Detail for 3D Graphics*. Hrsg. von Brian A. Barsky. The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling. San Francisco, USA: Morgan Kaufman Publishers, 2003.

[MBW08] Oliver Mattausch, Jiří Bittner und Michael Wimmer. „CHC++: Coherent Hierarchical Culling Revisited“. In: *Computer Graphics Forum* 27.2 (Apr. 2008). Proceedings of Eurographics 2008, S. 221–230. DOI: 10.1111/j.1467-8659.2008.01119.x.

[Mit91] Don P. Mitchell. „Spectrally optimal sampling for distribution ray tracing“. In: *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '91. New York, NY, USA: ACM, 1991, S. 157–164. DOI: 10.1145/122718.122736.

[Möb27] August Ferdinand Möbius. *Der barycentrische Calcul - ein neues Hülfsmittel zur analytischen Behandlung der Geometrie*. Leipzig, Germany: Verlag von Johann Ambrosius Barth, 1827.

[PD90] Harry Plantinga und Charles R. Dyer. „Visibility, occlusion, and the aspect graph“. In: *International Journal of Computer Vision* 5.2 (Nov. 1990), S. 137–160. DOI: 10.1007/BF00054919.

[Ped11] Marius. Pedersen. „Image quality metrics for the evaluation of printing workflows“. Diss. University of Oslo, Faculty of Mathematics und Natural Sciences, 2011.

[Pet+13a] Ralf Petring, Benjamin Eikel, Claudius Jähn, Matthias Fischer und Friedhelm Meyer auf der Heide. „Darstellung heterogener 3-D-Szenen in Echtzeit“. In: *11. Paderborner Workshop Augmented & Virtual Reality in der Produktentstehung*. Hrsg. von Jürgen Gausemeier, Michael Grafe und Friedhelm Meyer auf der Heide. Bd. 311. HNI-Verlagsschriftenreihe. Heinz Nixdorf Institut, Apr. 2013, S. 49–60.

[Pet+13b] Ralf Petring, Benjamin Eikel, Claudius Jähn, Matthias Fischer und Friedhelm Meyer auf der Heide. „Real-Time 3D Rendering of Heterogeneous Scenes“. In: *Advances in Visual Computing*. Hrsg. von George Bebis u. a. Bd. 8033. Lecture Notes in Computer Science. Proceedings of the 9th International Symposium on

Visual Computing (ISVC 2013). Springer Berlin Heidelberg, 2013, S. 448–458. DOI: 10.1007/978-3-642-41914-0_44.

[Pfi+00] Hanspeter Pfister, Matthias Zwicker, Jeroen van Baar und Markus Gross. „Surfels: surface elements as rendering primitives“. In: *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '00. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000, S. 335–342. DOI: 10.1145/344779.344936.

[PH12] Marius Pedersen und Jon Yngve Hardeberg. „Full-Reference Image Quality Metrics: Classification and Evaluation“. In: *Found. Trends. Comput. Graph. Vis.* 7.1 (Jan. 2012), S. 1–80. DOI: 10.1561/0600000037.

[Sam05] Hanan Samet. *Foundations of Multidimensional and Metric Data Structures (The Morgan Kaufmann Series in Computer Graphics and Geometric Modeling)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005.

[Str74] Wolfgang Straßer. „Schnelle Kurven- und Flächendarstellung auf graphischen Sichtgeräten“. Diss. Berlin, Germany: TU Berlin, 1974.

[Süß+11] Tim Süß, Clemens Koch, Claudius Jähn und Matthias Fischer. „Approximative occlusion culling using the hull tree“. In: *Proceedings of Graphics Interface 2011*. GI '11. School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada: Canadian Human-Computer Communications Society, Mai 2011, S. 79–86.

[Ten11] Fei Teng. „Image Error Evaluation“. Bachelor's thesis. Heinz Nixdorf Institute & Department of Computer Science, University of Paderborn, 2011.

[TS91] Seth J. Teller und Carlo H. Séquin. „Visibility preprocessing for interactive walkthroughs“. In: *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '91. New York, NY, USA: ACM, 1991, S. 61–70. DOI: 10.1145/122718.122725.

[Ulr00] Thatcher Ulrich. „Loose Octrees“. In: *Game Programming Gems*. Hrsg. von Mark DeLoura. Game Programming Gems. Boston, MA, USA: Charles River Media, 2000. Kap. 4.11, S. 444–453.

[Wan+04] Zhou Wang, Alan Conrad Bovik, Hamid Rahim Sheikh und Eero P. Simoncelli. „Image quality assessment: from error visibility to structural similarity“. In: *Image Processing, IEEE Transactions on* 13.4 (Apr. 2004), S. 600–612. DOI: 10.1109/TIP.2003.819861.

[WB09] Zhou Wang und Alan Conrad Bovik. „Mean squared error: Love it or leave it? A new look at Signal Fidelity Measures“. In: *IEEE Signal Processing Magazine* 26.1 (Jan. 2009), S. 98–117. DOI: 10.1109/MSP.2008.930649.

[Zha+97] Hansong Zhang, Dinesh Manocha, Tom Hudson und Kenneth E. Hoff III. „Visibility culling using hierarchical occlusion maps“. In: *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '97. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1997, S. 77–88. DOI: 10.1145/258734.258781.