UNIVERSITÄT PADERBORN
Die Universität der Informationsgesellschaft

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

Dissertation

Bewertung von Renderingalgorithmen für komplexe 3-D-Szenen

Claudius Jähn

Paderborn 2015

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

Kontakt: Claudius Jähn <claudius@uni-paderborn.de>
Dokumentenversion vom 21. Dezember 2015

Zusammenfassung

Die Effizienz von Renderingalgorithmen für komplexe virtuelle 3-D-Szenen hängt oft stark von der Position des Betrachters in der Szene ab. Die Bewertung der Algorithmeneffizienz in einer gegebenen Szene erfolgt daher typischerweise durch die Messungen entlang eines charakteristischen Kamerapfades durch die Szene. Objektive Aussagen über das generelle Verhalten des Algorithmus sind dabei in ihrer Aussagekraft deutlich beschränkt. Ich stelle ein Verfahren vor, das die evaluierte Kenngröße von Renderingalgorithmen, wie die Renderingzeit oder Anzahl durchgeführter Operationen, als positionsabhängige Szeneneigenschaft betrachtet, deren Verteilung durch adaptives Sampling für alle Positionen der Szene angenähert wird. Die statistische Auswertung erlaubt einen direkten, objektiven Vergleich verschiedener Renderingalgorithmen oder Parameterwerte; ihre Visualisierung kann zum Verständnis des Verhaltens von Algorithmen beitragen. Die vorgestellte Technik lässt sich sowohl während der Entwicklung von Renderingalgorithmen als auch zur Vorbereitung von konkreten Anwendungsfällen verwenden. Des Weiteren wird das punktbasierte Progressive-Blue-Surfels-Renderingverfahren für die Darstellung hochkomplexer virtueller Szenen vorgestellt. Das Verfahren generiert sortierte Punktfolgen auf der sichtbaren Oberfläche der Geometrie der Szene, so dass jeder Präfix der Folge eine gute Näherung der Geometrie darstellt und die Qualität und Laufzeit durch die Anzahl der dargestellten Punkte feinschrittig eingestellt werden kann. Die Techniken sind in PADrend implementiert, einem Renderingsystem, das speziell für den Entwurf von Renderingalgorithmen entwickelt wurde.

Abstract

The efficiency of rendering algorithms for complex virtual 3D scenes does not only dependent on the scene's overall properties, but also on the observer's position inside the scene. To experimentally evaluate an algorithm, measurements are typically performed along a characteristic camera path. This allows only for a weak assessment of the algorithm's general performance even for a fixed scene. I present an approach to represent aspects of an algorithm's behavior, like its running time or the number of performed operations, as position dependent scene properties. The properties' distribution can be approximated for all positions in the scene using an adaptive sampling technique. A distribution's statistical evaluation allows for a direct and objective comparison of different algorithms and parameter values. Its visualization yields intuitive insight into the algorithms behavior. Additionally, I present the point-based Progressive Blue Surfels rendering algorithm for visualizing highly complex virtual scenes. The algorithm places a sorted sequence of points on the visible surface of the scene's geometry, so that every prefix of the points represents a complete approximation of the geometry. By choosing the rendered sequence's length, image quality and running time can be adjusted at runtime. The presented techniques are implemented in PADrend, a rendering system specially designed for supporting the process of developing rendering algorithms.

Inhaltsverzeichnis

1 Einleitung

Im Bereich der 3-D-Computergrafik spielt, neben der optischen Qualität der erzeugten Bilder, die Geschwindigkeit, mit der die Bilder berechnet werden können, eine entscheidende Rolle für die Anwendung. Sobald ein Benutzer sich frei in einer virtuellen Umgebung bewegen möchte, darf eine bestimmte Bildrate nicht unterschritten werden. Bei Renderingsystemen zur Visualisierung von 3-D-CAD-Daten können bereits weniger als 10 fps (frames per second) ausreichend sein, um sich in der Umgebung zurechtzufinden und für das Anwendungsszenario notwendige Aktionen auszuführen. Für Computerspiele sind hingegen oft Bildraten von bis zu 60 fps wünschenswert, um dem Benutzer eine schnelle Reaktion auf Ereignisse in der virtuellen Welt zu ermöglichen. Unabhängig von der tatsächlich notwendigen Bildrate ist die Komplexität der darzustellenden Daten oft so groß, dass trotz mittlerweile sehr leistungsfähiger Spezialhardware die Bilder nicht auf einfache Art in entsprechender Zeit berechnet werden können. Dies gilt insbesondere für die Darstellung von 3-D-Daten im industriellen Umfeld, wo die Daten durch die Anwendung vorgegeben sind (z. B. automatisch generiert aus CAD-Daten) und nicht zum Zwecke der Darstellung erzeugt werden, wie es bei Computerspielen der Fall ist.

Um die Komplexität der in Echtzeit darstellbaren Szenen immer weiter zu erhöhen, wird zum einen die Grafikhardware ständig in ihrer Leistungsfähigkeit und in ihren Möglichkeiten erweitert. Zum anderen wurden und werden eine Vielzahl unterschiedlicher Renderingalgorithmen und Datenstrukturen entwickelt, um die 3-D-Daten zu filtern und so aufzubereiten, dass die Grenze der handhabbaren Szenenkomplexität bisweilen um Größenordungen angehoben wird. In diesem Kontext werden in dieser Arbeit drei Ziele verfolgt:

1. Der Schwerpunkt dieser Arbeit bezieht sich auf die Entwicklung einer Technik zur Evaluierung von Renderingalgorithmen, welche speziell das Verhalten eines Algorithmus in Abhängigkeit der Betrachterposition innerhalb der virtuellen Szene untersucht. Im Vergleich zu gängigen, kamerapfadbasierte Evaluierungstechniken, erlaubt die vorgestellte Technik eine objektivere Bewertung von Renderingalgorithmen. Durch die Visualisierung der Messdaten liefert sie zusätzlich einen neuen und intuitiven Zugang zum Verständnis des Verhaltens von Renderingalgorithmen. Vorgestellt wurde die Technik in der Arbeit *„Evaluation of Rendering Algorithms using Position-Dependent Scene Properties“* [JEF+13].

2. Das *Progressive-Blue-Surfels*-Näherungsverfahren ist ein Renderingverfahren für die interaktive Darstellung von hochkomplexen Szenen. Das Verfahren beschleunigt den Renderingprozess indem komplexe, aber weit entfernte Bereiche von Szenen durch Folgen von Punktprimitiven dargestellt werden. Im Vergleich zu anderen, punktbasierten Renderingverfahren, ist die zugrundeliegende geometrische Struktur und die Oberflächeneigenschaften der Szene nicht durch das Verfahren eingeschränkt. Erstmalig vorgestellt

wurde das Verfahren in dem technischen Bericht *„Progressive Blue Surfels“* [Jäh13]; eine weitere Publikation ist in Vorbereitung.

3. Um den Entwicklungsprozess von Renderingalgorithmen technisch zu unterstützen, wurde das *PADrend-System* entwickelt. Das System stellt Evaluierungswerkzeuge zur Verfügung, um neu entwickelte Renderingalgorithmen zu bewerten und mit zahlreichen existierenden Verfahren zu vergleichen. Vorgestellt wurde das System in der Arbeit *„PADrend: Platform for Algorithm Development and Rendering“* [EJP11].

Im Folgenden gebe ich einen kurzen Überblick über die behandelten Aspekte.

Evaluierung mit positionsabhängigen Szeneneigenschaften

Renderingalgorithmen für die Darstellung komplexer Szenen setzen unterschiedliche Techniken ein, um den Berechnungsaufwand für ein einzelnes Bild zu reduzieren. Bei *Occlusion-Culling-Algorithmen* geschieht dies beispielsweise durch Erkennen und Verwerfen von verdeckten Teilen der virtuellen Szene möglichst früh im Renderingprozess. Das Ziel dabei ist, zur Laufzeit möglichst wenige Ressourcen auf die Berechnung von Daten aufzuwenden, die nicht zu dem endgültigen Bild beitragen. Wenn in einer Szene generell eine hohe Verdeckung herrscht, wie beispielsweise in vielen architektonischen Szenen, kann sie dadurch flüssig dargestellt werden, auch wenn ihre Gesamtkomplexität die Fähigkeiten der Hardware bei einer einfachen Darstellung ohne spezielles Verfahren überschreitet. Ist die Szene aber aufgrund ihrer Struktur nicht für ein Verfahren geeignet oder werden die Parameter des Renderingalgorithmus bzw. der verwendeten Datenstruktur schlecht gewählt, so kann der Einsatz eines Occlusion-Culling-Algorithmus auch den Renderingprozess verlangsamen. Dabei ist das Verhalten des Algorithmus jedoch nicht nur von der Szene als Ganzes abhängig, sondern auch von der aktuellen Position und der Blickrichtung des Betrachters innerhalb der virtuellen Szene. Steht ein Betrachter direkt vor einer Wand, so dass große Bereiche der Szene hinter der Wand verdeckt sind, ist es wahrscheinlich, dass sich ein Occlusion-Culling-Algorithmus auszahlt. Wird dieselbe Szene jedoch von einer anderen Position aus betrachtet, können große Bereiche sichtbar sein und sich das Occlusion-Culling ggf. nicht mehr lohnen und sogar zu einer geringeren Bildrate führen. Abbildung 1.1 verdeutlicht diese Situation an einem einfachen Beispiel. Diese starke Abhängigkeit von Renderingalgorithmen gegenüber der Betrachterposition stellt in verschiedenen Situationen eine Herausforderung dar:

- Bei der **Entwicklung von neuen Renderingalgorithmen** muss der Entwickler das Verhalten von Algorithmen in möglichst vielen Situationen nachvollziehen können, um den Algorithmus für den gewünschten Einsatzbereich anzupassen. Bei der Evaluierung eines neuen Verfahrens erfordert die Positionsabhängigkeit daher aufwändige Testverfahren, um die Leistung des neuen Verfahrens objektiv gegen andere Verfahren abzugrenzen (als Teil des Algorithm-Engineering von Renderingalgorithmen).
- Bei der **Aufbereitung der Daten** einer komplexen virtuellen Szene muss für eine ausreichend effiziente Darstellung ein geeigneter Renderingalgorithmus sowie eine entsprechende Datenstruktur zur Speicherung der Szene gewählt werden. Hinzu kommen

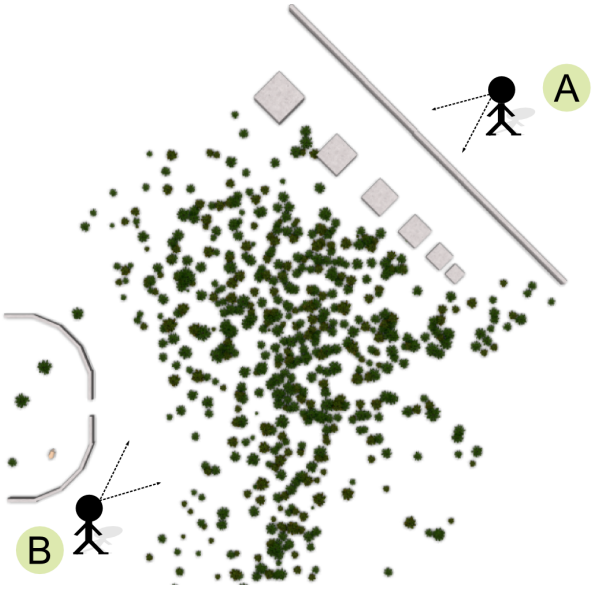


Abbildung 1.1: Draufsicht auf eine einfache Beispielszene (*Szene 1*) bestehend aus einigen Bäumen und Wänden. An Position A steht der Betrachter hinter einer Wand: Occlusion-Culling lohnt sich. An Position B herrscht wenig Verdeckung im Sichtbereich: Occlusion-Culling kann den Renderingprozess verlangsamen.

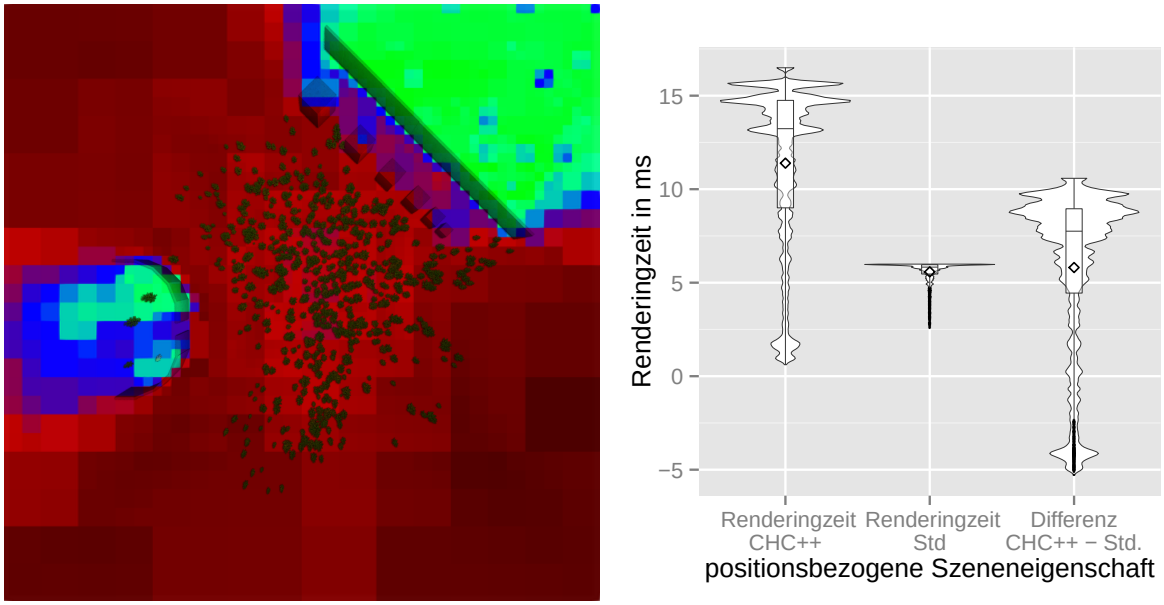


Abbildung 1.2: Beispiel für die Auswertung der globalen Näherung der Szeneneigenschaft *Renderingzeit* für Szene 1 (gespeichert in einem Octree). (links) Visualisierung der Differenz aus der Renderingzeit mit Occlusion-Culling und Renderingzeit mit Standardrendering; grün: Occlusion-Culling ist schneller; rot: Standardrendering ist schneller. (rechts) Verteilung der Szeneneigenschaft über den untersuchten Bereich der Szene als Box- und Violinen-Plot.

ggf. noch zahlreiche Parameter, für die entsprechende Werte gewählt werden müssen. Hierfür benötigt man entweder entsprechendes Expertenwissen, um das Verhalten des Algorithmus abschätzen zu können, oder aber wiederum aufwändige Testverfahren.

- **Während des Walkthroughs**, also während sich der Betrachter durch die Szene bewegt, kann der ausgewählte Renderingalgorithmus mit sehr unterschiedlichen Situationen konfrontiert werden. Ist im Algorithmus kein Mechanismus integriert, der eine situationsbedingte Anpassung ermöglicht, kann dies seine Einsatzmöglichkeiten deutlich einschränken.

Die gebräuchliche Herangehensweise zur Untersuchung des positionsabhängigen Verhaltens eines Algorithmus in einer gegebenen Szene ist die Messung der zu untersuchenden Eigenschaft, wie etwa Laufzeit des Algorithmus, durchgeführte Operationen oder Menge der sichtbaren Geometrie, entlang eines „typischen“ Kamerapfades. Bei einem gut gewählten Kamerapfad kann dies wichtige Einblicke in das Verhalten des Algorithmus liefern. Die Gültigkeit einer statistischen Auswertung der Ergebnisse bezieht sich jedoch nur auf den gewählten Pfad. In dieser Arbeit stelle ich eine Technik vor, die eine weitreichendere systematische Bestimmung und Auswertung von positionsabhängigem Verhalten von Renderingalgorithmen erlaubt. Dazu werden die untersuchten positionsabhängigen Eigenschaften als Funktionen des Raumes der Szene aufgefasst. Dabei wird die Sichtrichtung abstrahiert, so dass nur ein repräsentativer Wert für eine Position verwendet wird. Diese Funktionen lassen sich durch einen adaptiven Sampling-Ansatz global, d. h. für alle Positionen in der Szene, annähern und diese Näherung in einer Datenstruktur speichern. Diese Datenstruktur bezeichne ich als *globale Näherung einer Szeneneigenschaft*. Ein Beispiel für die Visualisierung der Datenstruktur sowie der Möglichkeit der statistischen Auswertung zeigt Abbildung 1.2. Die Abbildung visualisiert die globale Näherung der Differenz der Renderingzeit aus einfachem Standardrendering und Rendering mit einem Occlusion-Culling-Algorithmus (CHC++ [MBW08]) für einen Bereich in der Szene. In der farblichen Darstellung werden die Positionen in der Szene in grün hervorgehoben, an denen die Verwendung des Occlusion-Culling-Algorithmus zu einer höheren Bildrate führt. In den roten Bereichen ist unter den Randbedingungen das Standardrendering schneller. Durch die Visualisierung lässt sich so das Verhalten des Algorithmus auf einen Blick wesentlich anschaulicher darstellen, als dies allein mit der Untersuchung von Kamerapfaden möglich ist. Die statistische Auswertung zeigt die Verteilung der Renderingzeit für alle Positionen im untersuchten Bereich. Auf dieser Basis kann man nun entscheiden, welchen Algorithmus man für das angestrebte Anwendungsszenario einsetzen möchte oder weitere Varianten und Parameterwerte gegeneinander abwägen. Für diese Entscheidung bietet die statistische Auswertung der Szeneneigenschaften eine deutlich objektivere Grundlage als eine ähnliche Auswertung eines frei gewählten Kamerapfades in der Szene.

Progressive-Blue-Surfels: Algorithmus zur Darstellung hochkomplexer Szenen

Ein weiterer Bestandteil dieser Arbeit ist die Vorstellung eines neuen Renderingverfahrens für die Darstellung hochkomplexer Szenen: das *Progressive-Blue-Surfels*-Verfahren. Die grundle-

gende Idee ist die Annäherung komplexer, aber weit entfernter Bereiche der Szene durch eine sortierte Folge von Punkten. Die Punkte liegen auf der von außen sichtbaren Oberfläche der angenäherten Geometrie. Die wesentlichen Merkmale des Verfahrens im Vergleich zu anderen punktbasierten Näherungsverfahren sind die Möglichkeit, durch die Wahl der Länge des dargestellten Präfixes der Punktfolge die Darstellungsqualität und Renderingzeit präzise einstellen zu können, und seine Robustheit in der Art der darzustellenden Szene. Durch die Möglichkeit, die Detaillierung kontinuierlich einstellen zu können, werden die bei vielen Näherungsverfahren auftretenden, störenden Artefakte beim Umschalten zwischen verschiedenen Qualitätsstufen deutlich reduziert. Die Robustheit gegenüber der Szene erlaubt die Darstellung unterschiedlich strukturierter Objekte, inklusive komplexer CAD-Daten, gescannter Oberflächenmodelle und prozedural generierter Landschaften. Oberflächeneigenschaften wie Texturen und dynamischer Beleuchtung werden ebenfalls unterstützt. Insgesamt erlaubt das Verfahren die Darstellung von Szenen aus mehreren Milliarden Polygonen mit interaktiven Bildraten.

PADrend: Softwaresystem für die Entwicklung von Renderingalgorithmen

PADrend ist ein Softwaresystem zur Darstellung virtueller Szenen, bei dem sich der Benutzer frei durch die Szene bewegen kann. Das System wurde gemeinsam von Benjamin Eikel, Ralf Petring und mir konzipiert und implementiert, unter Einbeziehung studentischer Arbeiten. Im Vergleich zu anderen Renderingsystemen, wie beispielsweise Game-Engines, liegt der Schwerpunkt nicht auf dem Rendering für einen speziellen Anwendungsfall, sondern in der Unterstützung bei der Entwicklung und der Evaluierung von Renderingalgorithmen. Dies spiegelt sich in mehreren Aspekten im Systemaufbau wider: Die enthaltene Szenengraphbibliothek *MinSG* (Minimalistic-Scene-Graph) erlaubt die Übernahme der vollständige Kontrolle über den Renderingprozess – kapselt aber bei Bedarf alle technischen Details in einem High-Level-Interface, um den Entwickler zu entlasten und die Einstiegshürden für neue Entwickler zu senken. Zum Vergleich neuer Algorithmen mit aktuellen Verfahren aus der Literatur sind zahlreiche Occlusion-Culling- und Näherungsverfahren im System implementiert. Das Benutzerinterface bietet verschiedene Messwerkzeuge an, um verschiedene Kenngrößen zur Laufzeit zu erheben; die Bestimmung von positionsabhängigen Szeneneigenschaften ist eines davon. Unter anderem wurde das System bereits für die Entwicklung mehrerer Renderingverfahren verwendet und diente als Ausgangsbasis zahlreicher studentischer Abschlussarbeiten. Abseits des Algorithmenentwurfs wird das System auch im Rahmen von Industrieprojekten für die Durchführung von Design-Reviews virtueller Prototypen von Maschinen und Industrieanlagen genutzt. Durch die Möglichkeit, das mechanische Verhalten der virtuellen Prototypen durch High-Level-Funktionen nachzubilden, lassen sich mit geringem Arbeitsaufwand komplexe Situationen virtuell nachstellen. Auf einer stereoskopischen Mehrkanalprojektion kann mit getrackten Eingabegeräten mit dem virtuellen Prototypen interagiert werden.

Aufbau der Arbeit

In dieser Arbeit beschäftige ich mich speziell mit einem Teilbereich des Themengebietes der Computergrafik (dem Echtzeitrendering) und baue dabei auf unterschiedlichen, existierenden Verfahren auf. Eine Abgrenzung des Themengebietes sowie einen Überblick über diese verwendeten Arbeiten gebe ich in Kapitel 2. Als Basis für die Evaluierung der in dieser Arbeit vorgestellten Methoden wurde das PADrend-Softwaresystem verwendet, welches in weiten Teilen speziell für diesen Einsatzzweck entworfen wurde. Die grundlegende Architektur und die Möglichkeiten des Systems stelle ich in Kapitel 3 vor.

Eine detailliertere Erläuterung des Konzeptes der positionsabhängigen Szeneneigenschaft sowie einiger konkreter Szeneneigenschaften gebe ich in Kapitel 4. Um das Konzept der Szeneneigenschaft praktisch einsetzten zu können, verwende ich eine Näherung des Verlaufs der Eigenschaft in einem Bereich der Szene – der globalen Näherung der Szeneneigenschaft. Zur Berechnung dieser Näherung habe ich ein adaptives Sampling-Verfahren entwickelt, dessen Details ich in Kapitel 5 beschreibe und durch experimentelle Untersuchungen bewerte. Das punktbasierte Progressive-Blue-Surfels-Renderingverfahren erlaubt die interaktive Darstellung hochkomplexer Szenen. Das Verfahren und eine experimentelle Bewertung der Punkteverteilung beschreibe ich in Kapitel 6. Für die Anwendung der globalen Näherung einer Szeneneigenschaft für eine Verbesserung des Renderings habe ich verschiedene Einsatzszenarien identifiziert. Anhand von Beispielen erläutere ich diese in Kapitel 7. Dazu gehört eine Evaluierung der Laufzeiteigenschaften des Progressive-Blue-Surfel-Verfahrens. Abschließend ziehe ich in Kapitel 8 ein Fazit und gebe Anregungen für mögliche Anschlussfragen.

2 Abgrenzung im Bereich der Computergrafik

Das mit dieser Arbeit angerissene Feld der Computergrafik erstreckt sich über einen großen und vielseitigen Bereich. Im Folgenden grenze ich den in dieser Arbeit betrachtete Teilbereich genauer ein. Dabei erläutere ich fachbezogene Begriffe und stelle relevante Verfahren vor.

Rendering

Mit *Rendering* bezeichne ich den Vorgang, in dem aus einer virtuellen Szene durch *Renderingverfahren* ein Bild erzeugt wird. Ich beschränke mich in dieser Arbeit dabei auf die hardwareunterstützte Rasterisierung von polygonalen Daten in Pixelbitmaps (im Gegensatz z. B. zum Raytracing oder Volumenrendering). Für das in dieser Arbeit angepeilte Einsatzgebiet des Echtzeitrenderings werden kontinuierlich neue Bilder gerendert, während sich der Betrachter in einem *Walkthrough* frei durch die virtuelle Szene bewegen kann. Ein einzelnes Bild aus diesem Prozess bezeichne ich als *Frame*. Dies entspricht im Wesentlichen dem Echtzeitrendering-Begriff, wie er im Buch „Real-Time Rendering“ [AMHH08] Verwendung findet. Ein *Renderingverfahren* oder auch *Renderingalgorithmus* ist eine algorithmische Beschreibung, wie aus einer virtuellen Szene und weiteren Parametern (wie des Betrachterstandpunktes und Blickrichtung) das Bild erzeugt wird.

Virtuelle Szene

Als *virtuelle Szene* bezeichne ich eine Sammlung konvexer Polygone im 3-D-Raum (o. B. d. A. kann hier von Dreiecken ausgegangen werden), die von einem Betrachter durchwandert werden kann, während sie durch ein Renderingverfahren dargestellt wird. Eine Szene beschreibt die geometrischen und optischen Eigenschaften von Oberflächen, anders als z. B. im Bereich des Volumenrenderings, wo die Szenenbeschreibung auch das Innere von Objekten umfassen kann. Es gibt mehrere Quellen für die polygonalen Daten einer Szene:

- Parametrisch beschriebene CAD-Modelle können durch einen automatisierten Triangulierungsprozess in Polygonmodelle umgewandelt werden. Das in dieser Arbeit verwendete Power-Plant-Modell[1] wurde mutmaßlich ursprünglich aus einem CAD-Datensatz erstellt.
- Auf Basis von 3-D-Scans können hochauflösende Oberflächenmodelle realer Objekte erstellt werden. Das in Kapitel 6 verwendete Modell eines Drachen[2] wurde auf diese Weise erzeugt.

[1] http://gamma.cs.unc.edu/Powerplant/ (University of North Carolina at Chapel Hill)
[2] http://graphics.stanford.edu/data/3Dscanrep/ (The Stanford 3D Scanning Repository)

- Polygonale Modelle können manuell mit entsprechender 3-D-Modellierungssoftware erstellt werden. Die in dieser Arbeit verwendeten Baum-Modelle wurden auf diese Weise erzeugt.

- Über prozedurale Regeln lassen sich auch komplexe und weitläufige Objekte bzw. Oberflächen erzeugen. Die in Kapitel 6 und 7 verwendete Landschaft wurde durch ein Regelsystem beschrieben und mit Hilfe der Grafikhardware erzeugt. Das dabei eingesetzte Verfahren ist angelehnt an das Verfahren zur Erzeugung prozeduraler Landschaften von Geiss [Gei07].

Räumliche Datenstrukturen

Im Rahmen dieser Arbeit wird zusätzlich zu den Polygonen und Oberflächenbeschreibungen auch die Strukturierung der Szene in eine *räumliche Datenstruktur* als Teil der Szene gesehen und gehört damit, wie auch die Szene, zur Eingabe der vorgestellten Methoden. Eine gebräuchliche räumliche Datenstruktur ist die *Octree*-Datenstruktur. Die Datenstruktur ist ein Baum, wobei jeder Knoten einen quaderförmigen Bereich der Szene repräsentiert. Der Wurzelknoten (bzw. sein Bereich) umschließt die Geometrie der gesamten Szene. Innere Knoten besitzen acht Kinder, deren Regionen die des Elternknotens in gleiche Teile vollständig aufteilen. Für die Zuordnung der Objekte, aus denen die Szene besteht, zu den Knoten des Baums, gibt es mehrere Möglichkeiten, je nach genauer Ausprägung des Octrees. Ein Objekt kann beispielsweise an dem Knoten mit dem kleinsten Quader gespeichert werden, der das Objekt vollständig umschließt. Dadurch können auch sehr kleine Objekte auf der Unterteilung zwischen zwei Kindern auf einer hohen Ebene „hängenbleiben“. Da jedoch oft erwünscht ist, dass Objekte ähnlicher Größe gemeinsam in einem möglichst kleinen Knoten gespeichert werden, lässt sich diesem Problem mit dem *Loose-Octree* [Ulr00] entgegenwirken. Hierbei wird der Raum des Elternknotens nicht disjunkt zwischen den Kindern aufgeteilt, sondern die Knoten der Kinder werden vergrößert (z. B. um den Faktor zwei). Dadurch wird der Level im Baum, in dem die Objekte gespeichert werden, durch ihre Größe bestimmt. Um den entstehenden Baum auf die spezifischen Anforderungen eines verwendeten Renderingalgorithmus anpassen zu können, können Parameter, wie die maximale Tiefe des Baums, eingestellt werden.

Eine *Bounding-Volume-Hierarchie* beschreibt eine räumliche Baumdatenstruktur, bei der die Geometrie der Szene den Knoten zugeordnet und jedem Knoten ein Hüllvolumen (Bounding-Volume) zugeordnet ist, welches die in dem entsprechenden Teilbaum liegende Geometrie eng umschließt. Betrachtet man bei den Knoten eines Octree nicht die gesamten abgedeckten Bereiche, sondern bildet jeweils nur das Hüllvolumen, das alle in dem entsprechenden Teilbaum liegenden Geometrie umschließt, dann ergibt sich aus dem Octree eine Bounding-Volume-Hierarchie. Neben Bounding-Volume-Hierarchien, die durch andere algorithmische Regeln definiert werden, gibt es noch Hierarchien, die sich anhand des logischen (oder semantischen) Aufbaus der Szene ergeben. Dies ist beispielsweise der Fall, wenn die Struktur der Szene aus einem CAD-Modell übernommen wird, in dem die Bauteile hierarchisch nach Baugruppen abgelegt sind. Für das Rendering wird die Bounding-Volume-Hierarchie der Szene in Form eines *Szenengraphen* gespeichert. Dieser entspricht der technischen Umsetzung der Bounding-Volume-Hierarchie, es können dabei jedoch zusätzliche Daten an den Knoten hinterlegt werden.

Die in den Knoten des Szenengraphen verwendeten Hüllvolumen sind oft axial ausgerichtete Quader, die *Bounding-Boxen* der Knoten.

Objektbegriff

Die Polygone der Szene sind in mehrere in *Objekten* aufgeteilt, wobei im Extremfall ein Objekt auch nur aus einem einzelnen Polygon bestehen kann. Im Rahmen dieser Arbeit gehe ich davon aus, dass diese Objektdefinition als Teil der Szenen ein Teil der Eingabedaten ist, die jedoch, abhängig von verwendeten Renderingverfahren, einen erheblichen Einfluss auf die Effizienz bei der Darstellung der Szene haben kann. Häufig werden alle Polygone, die zur Darstellung eines realen Objektes benötigt werden, in einem virtuellen Objekt zusammengefasst; beispielsweise können alle Polygone, die zu einem virtuellen Tisch gehören, zu einem einzelnen Objekt zusammengefasst werden. Dies hat eine Reihe von Vorteilen: Bei einer Interaktion mit der Szene in einer Anwendungssituation kann der virtuelle Tisch als Einheit identifiziert werden und in der Szene bewegt werden. Des Weiteren besteht die Abbildung ein reales Objekt häufig aus einem einzelnen Material, welches sich dann einfach durch eine einzelne, an das Objekt gebundene, Materialbeschreibung abbilden lässt. Besteht der Tisch aus dem Beispiel vollständig aus Holz, kann dies einfach durch die Zuordnung einer Holztextur zu dem Objekt abgebildet werden, anstatt die Zuordnung für jedes Polygon einzeln vorzunehmen. Besteht ein reales Objekt aus Teilen mit unterschiedlich darzustellenden Materialien, bietet es sich daher an, die entsprechenden Teile als virtuelle Objekte zu definieren (z. B. Tischplatte und Tischbeine). Letztlich führt die Definition von Objekten zu größeren zusammenhängenden Gruppen von Polygonen, den *Meshes*. Diese lassen sich, im Gegensatz zu einer entsprechende Menge unzusammenhängender Polygone, mit der hier betrachteten hardwareunterstützten Rasterisierung wesentlich schneller darstellen.

2.1 Sichtbarkeit

Die in dieser Arbeit untersuchten Verfahren werden in vielerlei Hinsicht durch Sichtbarkeit beeinflusst. Im Folgenden erläutere ich die grundlegenden Konzepte und einige verwendete Algorithmen.

Die Bestimmung derjenigen Teile einer Szene, die, abhängig von der Betrachterposition und Blickrichtung, tatsächlich im endgültigen Bild sichtbar sind, ist seit jeher eines der grundlegenden Probleme der 3-D-Computergrafik. Das Problem, welches Polygon die Farbe eines Pixels bestimmt, wird in der hier betrachteten Form des Renderings praktisch durch den von Catmull 1974 vorgestellten Z-Puffer-Algorithmus [Cat74] gelöst. Dank der hochentwickelten dedizierten Grafikhardware lassen sich so mittlerweile Szenen mit mehreren Millionen Polygonen in Echtzeit darstellen. Werden alle Daten einer Szene jedoch einfach nur zur Darstellung an die Grafikkarte gesendet, erzeugen auch alle unsichtbare Objekte und Polygone Last auf der Grafikkarte und ggf. auf der CPU. Um die zu rendernde Datenmenge zu reduzieren, existieren mehrere grundlegende *Culling-Verfahren*, um unsichtbare Teile der Szene zu erkennen und vom Rendering auszuschließen (siehe Abbildung 2.1). Der Teil einer Szene, der außerhalb des Sichtbereichs des Betrachters (*Frustum*) liegt, lässt sich relativ einfach durch *Frustum-Culling-*

Verfahren identifizieren und vom Rendering aussparen, beispielsweise durch hierarchisches Frustum-Culling [Cla76]. Bei einer komplexen Szene kann die Menge der Geometrie innerhalb des Sichtbereichs dennoch die Fähigkeiten der Hardware deutlich übersteigen, auch wenn aufgrund von Verdeckung letztlich nur ein kleiner Teil dieser Geometrie zum Bild beiträgt. Das Ziel von *Occlusion-Culling-Verfahren* ist es, die nicht verdeckte Geometrie zu identifizieren und möglichst nur diese zu rendern. Die Relevanz dieses Problems wird unter anderem in den zahlreichen Publikationen zu verschiedenen Sichtbarkeits- bzw. Culling-Verfahren und entsprechender Übersichtsarbeiten deutlich [COCSD03, PT02]. Eine Möglichkeit der Klassifizierung dieser Verfahren (nach [COCSD03]) ist die Unterteilung in Verfahren, die die globale oder die lokale Sichtbarkeit bestimmen oder annähern. Verfahren zur Bestimmung der *lokalen Sichtbarkeit* ermitteln nur für die aktuelle Position des Betrachters die Sichtbarkeit der Teile der Szene. Sobald sich der Benutzer in der Szene bewegt, werden die Sichtbarkeitsinformationen zur Laufzeit neu berechnet. Verfahren zur Bestimmung der *globalen Sichtbarkeit* berechnen und speichern hingegen Sichtbarkeitsinformationen für Bereiche der Szene in der Vorverarbeitung (im *Preprocessing*). Während sich der Benutzer zur Laufzeit durch die Szene bewegt, können diese vorher gesammelten Informationen abgefragt werden und für die Darstellung der sichtbaren Teile verwendet werden.

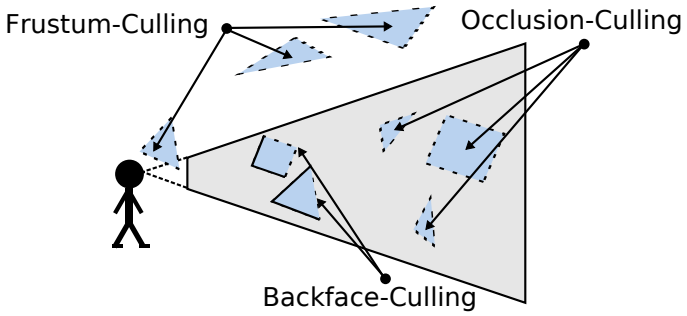


Abbildung 2.1: Unterschiedliche Arten von Culling (nach Cohen-Or et al. [COCSD03])

2.1.1 Globale Sichtbarkeit

Die globale Bestimmung der Sichtbarkeit ist eng verknüpft mit der in dieser Arbeit betrachteten globalen Näherung von Szeneneigenschaften: Sichtbarkeit lässt sich als Szeneneigenschaft beschreiben und kann damit durch die vorgestellten Verfahren genähert, statistisch untersucht und zum Rendering eingesetzt werden (siehe Kapitel 7.6). Die allgemeine Untersuchung globaler Szeneneigenschaften kann damit als eine Verallgemeinerung der Untersuchung von globaler Sichtbarkeit gesehen werden. Des Weiteren sind einige der untersuchten Szeneneigenschaften in ihrer Struktur eng mit der globalen Sichtbarkeit verknüpft, wie beispielsweise die Anzahl sichtbarer Objekte oder die Renderingzeit von Occlusion-Culling-Algorithmen.

Exakte geometrische Sichtbarkeit und die Visibility-Space-Partition (VSP)

Die *geometrische Sichtbarkeit* eines Polygons zeigt an, ob eine Strecke vom Betrachterstandpunkt zu einem Punkt auf dem Polygon existiert, die keine anderen Polygone schneidet. Nach

dieser Definition können alle Polygone einer Szene von einem Standpunkt aus sichtbar sein. Die von Platinga und Dyer vorgestellte *VSP* (Visibility-Space-Partition) [PD90] ist eine Datenstruktur, bei der die Szene in Bereiche aufgeteilt wird in der die gleiche geometrische Sichtbarkeit herrscht (oder genauer, die gleiche topologische Struktur der sichtbaren Geometrie). Die Übergänge zwischen Bereichen werden durch Sichtbarkeitsereignisse bestimmt, beispielsweise wenn sich der Betrachterstandpunkt auf einer Geraden mit zwei Eckpunkten von Polygonen der Szene befindet. Eine VSP lässt sich damit prinzipiell zur Kodierung der Sichtbarkeit einer Szene verwenden, um ausschließlich sichtbare Polygone darzustellen. Die Worst-Case-Komplexität einer VSP für den perspektivischen 3-D-Fall liegt bei $\Theta(n^9)$ in der Anzahl der Polygone der Szene. Auch wenn diese Worst-Case-Komplexität für eine realistische Szene nicht erreicht wird, ist sie für relevante Szenen immer noch deutlich zu hoch.

In der 3-D-Computergrafik wird unter Sichtbarkeit oft die *Pixelsichtbarkeit* verstanden; d.h. ein Polygon wird als sichtbar bezeichnet, wenn es mindestens die Farbe eines Pixels des fertigen Bildes bestimmt oder dazu beiträgt. Lässt man Multisampling und Blending außer Acht, ist damit die Anzahl der maximal gleichzeitig sichtbaren Polygone durch die Bildschirmauflösung beschränkt.

Portalbasierte Verfahren

Eine Möglichkeit, globale Sichtbarkeit praktisch für Rendering nutzbar zu machen, ist, nicht die Sichtbarkeit für jedes Polygon oder Objekt einzeln zu bestimmen, sondern für Teilbereiche der Szene. Ziel von portalbasierten Verfahren ist es, die Szene so in Regionen aufzuteilen, dass die Regionen untereinander nur durch räumlich begrenzte Flächen, die Portale, zu sehen sind. Als Aufteilung bei einem architektonischen Modell, bietet es sich beispielsweise an, dass die Räume die Regionen darstellen und Türen und Fenster die Portale bilden. Für jede Region kann dann eine Menge potentiell sichtbarer Objekte (*PVS – Potentially-Visible-Set*) bzw. anderer sichtbarer Regionen berechnet und gespeichert werden. Die Aufteilung in Regionen kann beispielsweise anhand der Geometrie der Szene erfolgen. Ein frühes und bekanntes Verfahren ist das *„Visibility Preprocessing For Interactive Walkthroughs“*-Verfahren von Teller und Séquin [TS91]. Dieses Verfahren eignet sich jedoch fast ausschließlich für architektonische Szenen mit axial ausgerichteten, rechteckigen Räumen.

Sampling-basierte Sichtbarkeitsverfahren

Sobald die Szenen komplexer werden oder keine klare, geometrische Struktur mehr zu erkennen ist, lassen sich die Regionen mit ähnlicher Sichtbarkeit auch durch Sampling bestimmen; also durch Näherung der globalen Sichtbarkeit durch eine Reihe an stichprobenartig bestimmten Sichtbarkeitstests. Die Samples können beispielsweise aus Strahlen bestehen, die im Raum der Szene verteilt werden. Die Strahlen enden auf der Oberfläche der Objekte der Szene und zeigen an, dass die getroffenen Objekte von jeder Position entlang des Strahls sichtbar sind. In der Arbeit *„Adaptive global visibility sampling“* beschreiben Bittner und andere [BMW$^+$09] ein Verfahren, bei dem solche Strahlen adaptiv in der Szene verteilt werden. Eine Besonderheit des Verfahrens ist, dass die Sampling-Strategie, wie die Samples in der Szene platziert werden, im Verlauf des Prozesses automatisch angepasst wird.

Globale Sichtbarkeit mit Spherical-Visibility-Sampling (SVS)

Eine weitere Möglichkeit, globale Sichtbarkeit effizient beim Rendering einzusetzen, ist, die *äußere Sichtbarkeit* zu betrachten – für komplexe Bereiche einer Szene wird hierarchisch gespeichert, welche Teile von außen sichtbar sind. Zur Laufzeit werden dann die Teile der Szene gerendert, die der Betrachter von außen sieht. Der große Vorteil dieser Betrachtungsweise ist, dass der Aufwand in der Vorverarbeitung und der Speicherplatzbedarf nur in der Komplexität der Szene steigt – und nicht zusätzlich mit der Ausdehnung der Szene, wie bei bei vielen anderen globalen Sichtbarkeitsverfahren. Dies eignet sich vor allem für die Darstellung von Szenen mit vielen, in sich geschachtelten Objekten, wie beispielsweise CAD-Modellen von Maschinen oder ausmodellierten Gebäuden. Ein Nachteil derartiger Verfahren ist, dass lokale Verdeckungen, die nur in einem begrenzten Bereich der Szene auftreten, nicht erkannt und ausgenutzt werden.

Ein auf äußerer Sichtbarkeit basierendes Verfahren ist das *Spherical-Visibility-Sampling-Verfahren* (SVS) von Eikel et al. [EJFMadH13], mit dem sich auch sehr weitläufige Szenen darstellen lassen. Für die inneren Knoten der räumlichen Datenstruktur, in der die Szene gespeichert ist, wird eine sogenannte Sichtbarkeitskugel berechnet. Diese enthält für mehrere Richtungen, welche Objekte der Szene jeweils von außerhalb der Kugel aus der Richtung sichtbar sind. Zur Laufzeit werden für die Teilbäume der Szene, deren Sichtbarkeitskugeln die Position des Betrachters nicht enthalten, nur die Objekte für die am nächsten liegenden Richtungen gerendert (von der Position des Betrachters in Richtung des Zentrums der Kugel). Die praktische Effizienz des Verfahrens wird in Abschnitt 7.4 als Beispiel für die Evaluierung mittels Szeneneigenschaften betrachtet.

Das in dieser Arbeit vorgestellte Progressive-Blue-Surfels-Verfahren (siehe Kapitel 6) nutzt ebenfalls die äußere Sichtbarkeit zur Beschleunigung des Renderings, da die Surfel nur auf der von außen sichtbaren Oberflächen der Geometrie platziert werden.

2.1.2 Lokale Sichtbarkeit (Online-Occlusion-Culling)

Bei lokalen Sichtbarkeitsverfahren wird die Sichtbarkeit von Objekten in der Szene nicht vorberechnet, sondern zur Laufzeit für die aktuelle Betrachterposition und Sichtrichtung bestimmt. Seitdem Grafikkarten hardwareseitige Unterstützung für Verdeckungstests anbieten, haben sich Verfahren, die dies ausnutzen, weit verbreitet. Die grundlegende Idee für hardwarebasierte Verdeckungstests wurde von Greene, Kass und Miller mit dem *hierarchischen Z-Puffer* [GKM93] vorgestellt. Beim Rendern einer hierarchisch organisierten Szene wird vor dem Rendern eines Teilbaums zunächst geprüft, ob mindestens ein Pixel eines den Teilbaum umschließenden Hüllvolumens (in diesem Fall einer Bounding-Box) sichtbar wäre, wenn sie gerendert würde. Beim hierarchischen Z-Puffer-Verfahren wird der Test, ob ein Pixel den Tiefentest besteht, auf der CPU durchgeführt. Durch eine Erweiterung der Hardware kann ein Verfahren von der Grafikkarte die Rückmeldung über diesen Test erhalten. Der zusätzliche Berechnungsaufwand auf der Grafikkarte entspricht nur in etwa der Zeit, die zum Rendern der Ersatzgeometrie benötigt wird. Jedoch erfolgt die Rückmeldung des Ergebnisses erst, nachdem die Pipeline der Grafikkarte durchlaufen wurde. Wartet ein Culling-Verfahren wiederholt auf ein Ergebnis, kann das Rendering dadurch deutlich verzögert werden. Für diese Problematik versuchen verschie-

dene Verfahren Lösungen anzubieten: Das *CHC-Verfahren* (Coherent-Hierarchical-Culling) von Bittner et al. [BWPP04] verwendet unter anderem die Informationen über die Sichtbarkeit von Knoten der Datenstruktur aus vorangegangenen Bildern, um nicht auf alle Verdeckungsanfragen zu warten. Das *Near-Optimal-Hierarchical-Culling-Verfahren* von Guthe, Balázs und Klein [GABK06] versucht mit statistischen Vorhersagen die Anzahl der durchgeführten und vor allem die Anzahl der Situationen, in denen auf eine Verdeckungsanfrage gewartet wird, zu minimieren. Das in dieser Arbeit als Referenzverfahren verwendete *CHC++-Verfahren* von Mattausch, Bittner und Wimmer [MBW08] setzt auf dem CHC-Verfahren auf, erweitert es aber um zahlreiche Heuristiken, um die Zahl der Verdeckungstests zu minimieren. Es werden beispielsweise mehrere Bounding-Boxen gemeinsam in Gruppen getestet, wobei die Zuordnungen zu diesen Gruppen dynamisch wechselt.

Für die Effizienz all dieser Verfahren ist es notwendig, dass die Szene in einer geeigneten räumlichen Datenstruktur mit geeigneten Parametern gespeichert ist. Als Beispiel für die Optimierung eines Parameterwertes mittels genäherter Szeneneigenschaften wird in Abschnitt 7.3 der Parameter der maximalen Tiefe eines Loose-Octrees für eine gegebene Szene und dem CHC++-Algorithmus ermittelt.

2.2 Genähertes Rendering

Die zweite Strategie zur Darstellung komplexer Szenen neben dem Occlusion-Culling ist die Darstellung von Ersatzrepräsentationen für komplexe Teile der Szene, die jedoch effizienter darstellbar sind. Der Nachteil ist, dass die Geometrie der Szene nicht mehr exakt dargestellt wird, sondern nur angenähert wird und dabei die Bildqualität sinken kann. Das in Kapitel 6 vorgestellte Progressive-Blue-Surfels-Verfahren nutzt für die Darstellung der Szene eine Folge von generierten Punktprimitiven, die die Geometrie annähern. Im Folgenden stelle ich einige Verfahren vor, die ebenfalls Punktprimitive für das Rendering einsetzen und ein Verfahren, das (wie das Progressive-Blue-Surfels-Verfahren) eine feingranulare Steuerung der Näherungsqualität ermöglicht.

QSplats: a multiresolution point rendering system for large meshes

Das *QSplat-Verfahren* von Rusinkiewicz und Levoy [RL00] wurde für die Visualisierung einzelner, hochkomplexer Oberflächenmodelle entwickelt. Es baut eine hierarchische Baumdatenstruktur aus Kugeln auf, die die Geometrie in unterschiedlichen Qualitätsstufen repräsentiert. Um den Baum aus einer Menge an Polygonen zu erzeugen, wird auf unterster Ebene für jeden Vertex eine Kugel erzeugt. Die Kugel kodiert die Normale und Farbe des Vertex. Die inneren Knoten der Datenstruktur werden dann rekursiv durch größere Kugeln zusammengefasst, die alle Kugeln in ihrem Teilbaum enthalten. Die Normale und Farbe für eine innere Kugel werden aus den Werten ihrer Kinder gemittelt. Beim Rendering wird der Baum traversiert und es werden nur die Kugeln, die auf einem Schnitt durch den Baum liegen, durch Punkte dargestellt. Der Verlauf des Schnittes durch den Baum bestimmt die Anzahl der dargestellten Punkte und damit Renderingzeit und Bildqualität.

Ein Vorteil der QSplats im Vergleich zu den Progressive-Blue-Surfels ist, dass durch das

Zusammenfassen der Oberflächeneigenschaften auf höheren Ebenen ein automatisches Filtern durchgeführt wird, wodurch Aliasingeffekte bei der Darstellung reduziert werden können. Ein weiterer Vorteil ist, dass durch die Art, die Kugeln zu erzeugen, keine Löcher beim Rendering entstehen. Nachteile der QSplats sind zum einen, dass die Ausgangsbasis für die Kugeln die Vertices des Modells sind – wenn es sich bei dem Modell nicht um ein hochtesseliertes Oberflächenmodell handelt, sondern um ein trianguliertes CAD-Modell, können nur wenige, grobe Kugeln auf unterster Ebene erzeugt werden. Ein anderer Nachteil ist, dass zur Darstellung einer Näherung der Baum traversiert werden muss – d. h. die Punktprimitive werden nacheinander, einzeln zur Grafikkarte übertragen. Aktuellen Grafikkarten können so nur einen Bruchteil ihrer Leistungsfähigkeit umsetzten.

Surfels: Surface Elements as Rendering Primitives

Das *Surfels-Verfahren* von Pfister et al. [PZvBG00] wurde, wie das QSplat-Verfahren, für die Visualisierung komplexer Einzelmodelle durch Punkte entwickelt. In der Vorverarbeitung wird das Modell zunächst durch Raycasting entlang der drei Hauptachsen in geschichtete Tiefenbilder (Layered depth images [SLS+96]) abgebildet. Auf Basis eines Octrees werden daraus hierarchisch die Surfels als Oberflächenpunkte mit Farbe und Normale gebildet. Zur Laufzeit wird der Octree traversiert und ein Schnitt des Baumes als Punkte dargestellt.

Das Surfels-Verfahren ist gegenüber dem QSplat-Verfahren robust gegenüber unterschiedlich triangulierte Oberflächen und unterstützt Texturen. Neben der ebenfalls nicht mehr effizienten Traversierung beim Rendering, ist ein Nachteil des Verfahrens, dass auch auf von außen nicht sichtbaren Flächen Punkte erzeugt werden.

The Randomized z-Buffer Algorithm: Interactive Rendering of Highly Complex Scenes

Der *randomisierte Z-Puffer* von Wand et al. [WFP+01] zielt, wie das Progressive-Blue-Surfels-Verfahren, darauf ab, nicht nur Einzelmodelle, sondern hochkomplexe Szenen darzustellen. Dazu werden die Dreiecke der Szene in einer hierarchischen Datenstruktur abgelegt. Zur Laufzeit werden dann für die aktuelle Betrachterposition Sampling-Punkte auf den Dreiecken im Frustum gezogen und gerendert. Die erwartete Anzahl der für ein Dreieck gezogene Samples ist dabei proportional zu seiner projizierten Größe.

Ein großer Vorteil des Verfahrens liegt darin, dass keine aufwändigen und speicherintensiven Vorverarbeitungen durchgeführt werden, da das Sampling zur Laufzeit durchgeführt wird. Nachteile des Verfahrens sind, dass die Punkte nicht effizient an die Grafikkarte geschickt werden und dass Punkte auch von unsichtbaren Oberflächen gezogen werden.

Progressive Meshes

Anstatt Teile der Szene durch eine Ersatzrepräsentation auszutauschen, lassen sich polygonale Modelle auch direkt vereinfachen und somit schneller darstellen. Mit dem *Progressive-Meshes-Verfahren* von Hoppe [Hop96] werden die Vertices eines Meshes in eine Datenstruktur

überführt, die es erlaubt, eine Näherung mit einer beliebigen Anzahl von Dreiecken darzustellen. Nacheinander werden bei der Vereinfachung jeweils zwei Vertices zusammengefasst, bis die gewünschte Anzahl an Dreiecken erreicht ist. Die Reihenfolge der Zusammenfassungen wird dabei so gewählt, dass nach verschiedenen Kriterien die Ähnlichkeit zum Original-Mesh möglichst lange erhalten bleibt. Durch das Verfahren lässt sich die Qualität der Meshes zur Laufzeit sehr feingranular anpassen (z. B. entsprechend der projizierten Größe des Meshes), wodurch störende Popping-Artefakte beim Umschalten der Qualitätsstufe vermieden werden. Das Progressive-Blue-Surfels-Verfahren folgt einem ähnlichen Ansatz für die Vermeidung von Popping-Artefakten, indem die Anzahl der dargestellten Punkte progressiv in kleinen Schritten gesteuert werden kann.

2.3 Renderingsysteme

Ein *Renderingsystem* ist eine Software, die das Rendering virtueller Szenen ermöglicht. Die vermutlich leistungsfähigsten, verfügbaren Renderingsysteme stellen Game-Engines dar, die als Grundlage für 3-D-Computerspiele dienen. Diese Systeme sind jedoch hochspezialisiert auf die Anwendung und erlauben nur begrenzte Möglichkeiten, um die enthaltenen Renderingalgorithmen anzupassen oder neue zu entwickeln. Szenengraph-Bibliotheken wie OpenSceneGraph[3] oder OGRE[4] geben dem Entwickler deutlich mehr Freiheiten. Renderingsysteme, die ihren Fokus speziell auf die Entwicklung und Evaluation von Renderingalgorithmen legen, sind jedoch selten. Ein Renderingsystem mit dieser Zielsetzung wurde von Yuan, Green und Lau im Jahr 1997 entwickelt [YGL97]. Eine Besonderheit in dieser Arbeit ist, dass die Autoren für die Generierung realistischer Kamerapfade für die Evaluierung aufgezeichnete Benutzerbewegungen verwenden. Für eine Gruppe von Benutzern wird aufgezeichnet, wie sie bestimmte vorgegebene Interaktionen mit der virtuellen Szene durchführt. Ein System für die Entwicklung und Evaluierung von Algorithmen zur Aufbereitung einzelner Meshes ist *Meshlab* [CCC+08]. Die frei verfügbare Software enthält bereits eine große Menge an unterschiedlichen Algorithmen, z. B. zum Bereinigen oder Optimieren von Gittermodellen. Durch zahlreiche Metriken zur Bewertung von unterschiedlichen Eigenschaften von Meshes lassen sich die Ergebnisse unterschiedlicher Verfahren direkt vergleichen. Das in Kapitel 3 vorgestellte PADrend-System folgt dieser Idee, ein System mit der Zielsetzung des Algorithmenentwurfs zur Verfügung zu stellen, jedoch für Szenen anstatt für einzelne Meshes.

[3] http://www.openscenegraph.org
[4] http://www.ogre3d.org

3 PADrend – Plattform for Algorithm Development and Rendering

Die in dieser Arbeit vorgestellten Arbeitsweisen und Methoden sind nicht auf die Verwendung mit einer spezifische Renderingsystem beschränkt, sondern sie sollten sich mit überschaubarem Aufwand in die meisten der aktuell verbreiteten Systeme (wie Game-Engines oder Szenengraph-Bibliotheken) integrieren lassen. Für die Entwicklung der Techniken und für die experimentelle Evaluation im Rahmen dieser Arbeit wurde das PADrend-System (**P**latform for **A**lgorithm **D**evelopment and **ren**dering) [EJP11] eingesetzt, welches speziell für die Entwicklung und Erprobung von Renderingalgorithmen entwickelt wird. Diese Zielsetzung unterscheidet PADrend von anderen Renderingsystemen, bei denen eher andere Anwendungen im Fokus steht, z. B. Computerspiele oder Simulationen. Der Kern von PADrend wurde zu gleichen Teilen von Ralf Petring, Benjamin Eikel und mir entwickelt. Viele Erweiterungen des Systems wurden im Rahmen von studentischen Arbeiten erstellt und verbessert.

Das PADrend-System besteht aus einer Reihe modular aufgebauter Bibliotheken (entwickelt in C++) und einer darauf aufbauenden, ebenfalls modularen Anwendung (entwickelt in EScript). Abbildung 3.1 zeigt die Struktur des Systems als Diagramm. Im Folgenden gebe ich einen kurzen Überblick über die wesentlichen Eigenschaften einiger der Komponenten.

3.1 Systembibliotheken

Um eine weitreichende Unabhängigkeit zu Systemeigenschaften wie Betriebssystem (Windows, Linux, MacOS oder Android) oder OpenGL-Version zu erreichen, beinhaltet PADrend zwei Bibliotheken zum Zugriff auf systemspezifische Funktionen. Die *Util*-Bibliothek abstrahiert unter anderem Dateioperationen, Netzwerkzugriffe und Threads. Jeder Zugriff auf die Grafikhardware erfolgt über die *Rendering*-Bibliothek. Diese enthält High-Level-Wrapper für unterschiedliche Grafikkomponenten wie Dreiecksmeshes, Frame-Buffer-Objekte oder Shader. Um eine breite Palette von OpenGL-Systemen zu unterstützen (von OpenGL-ES auf Mobilsystemen über OpenGL 2.0 auf älteren Rechnern bis zu OpenGL 4.0 auf aktuellen Workstations) bietet die Bibliothek eine eigene Statusverwaltung an. Diese verwaltet Eigenschaften wie Projektionsmatrizen oder Beleuchtungs- und Materialparameter und gibt sie transparent über die verfügbaren Schnittstellen an das Grafiksystem weiter. Ein Entwickler von Renderingalgorithmen braucht sich daher im Idealfall nicht mit systemspezifischen Low-Level-Aspekten befassen.

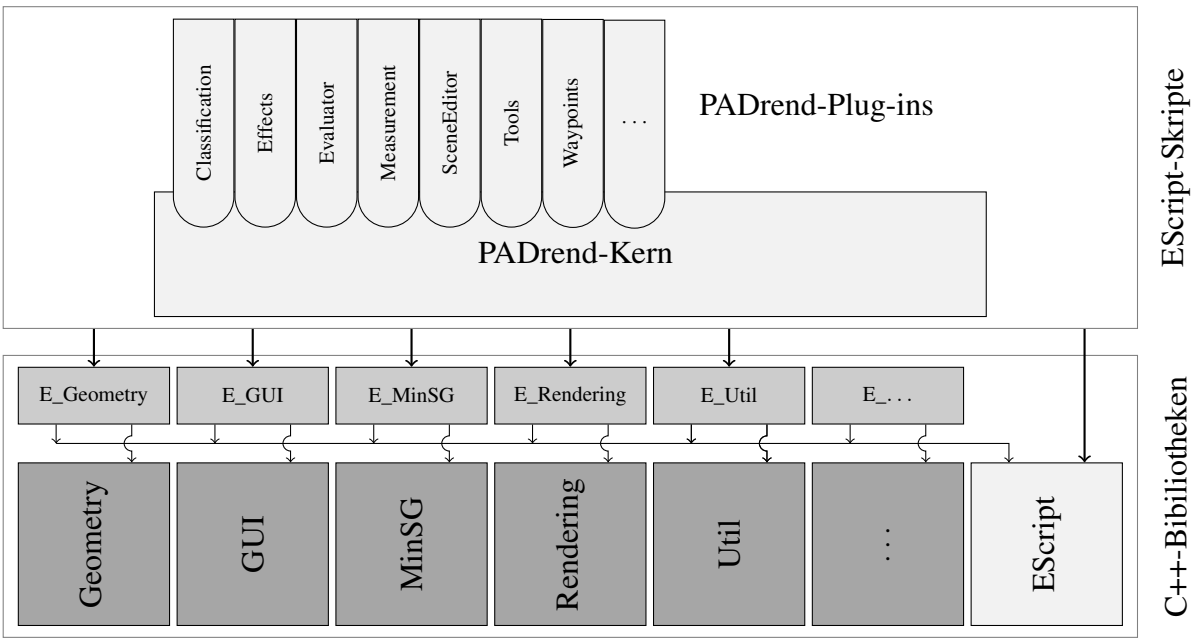


Abbildung 3.1: Systemarchitektur des PADrend-Systems (aus Eikel, Jähn, Petring 2011 [EJP11])

3.2 Szenengraph: MinSG

Die Basis des Renderingsystems bildet die Szenengraphimplementierung *MinSG* (Minimalistic-Scene-Graph). Die Objekte einer virtuellen Szene werden als Knoten in einer Baumstruktur repräsentiert. Was MinSG bei der Entwicklung und Erprobung von Renderingalgorithmen auszeichnet, ist, dass Renderingalgorithmen als Eigenschaften von Knoten definiert werden, und nicht als externer Prozess. So wie es möglich ist, einem Knoten eine bestimmte Materialeigenschaft zuzuweisen, ist es möglich, einem Teilbaum einen Renderingalgorithmus, wie etwa ein Occlusion-Culling-Verfahren, zuzuweisen. Dieses Renderingverfahren wird dann zur Darstellung aller Knoten des entsprechenden Teilbaums verwendet. Durch dieses einfache aber mächtige Konzept wird es möglich, verschiedene Teile einer Szene in einem Bild mit unterschiedlichen Algorithmen zu rendern, ohne dafür Anpassungen am Quelltext des Renderingprozesses vornehmen zu müssen.

Viele Verfahren benötigen zusätzliche Daten für jeden Knoten im Szenengraph, wie beispielsweise den Sichtbarkeitsstatus des Knotens in den letzten Frames. Diese Daten lassen sich in MinSG als *generische Attribute* zur Laufzeit an den Knoten speichern. Dadurch lassen sich auch komplexere Renderingalgorithmen als abgeschlossene Einheit implementieren, ohne dass die Knotenimplementierung angepasst oder erweitert werden muss. Zusammenfassend enthält ein Knoten im MinSG-Szenengraph im Wesentlichen folgende Komponenten:

- Eine Beschreibung der Transformation (Translation, Rotation und Skalierung) des Knotens relativ zu seinem Elternknoten.
- Referenzen auf die dem Knoten zugeordneten Kindkonten (innerer Knoten) oder alterna-

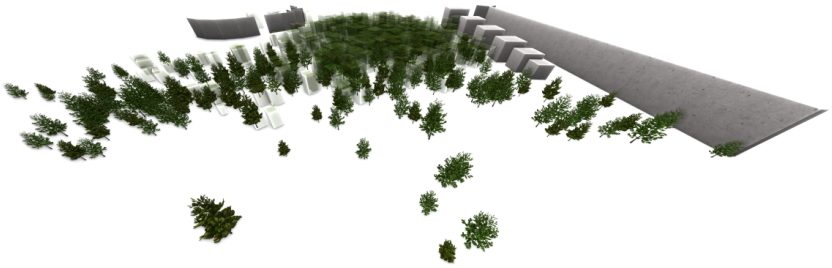


Abbildung 3.2: Darstellung von Szene 1 mit dem Renderingverfahren *Fast rendering of complex environments using a spatial hierarchy* nach Chamberlain et al. [CDL+96]

tiv eine Referenz auf ein Geometrieobjekt (Blattknoten). Jeder Knoten ist maximal einem Elternknoten zugeordnet, wobei ein Geometrieobjekt auch von mehreren Blattknoten referenziert werden kann.

- Eine 3-D-Boundingbox, die alle Geometrieobjekte im Teilbaum des Knotens umschließt.
- Eine Liste an Eigenschaften (*States*), die den Renderingprozess beeinflussen – wie z. B. Materialeigenschaften, Shader oder auch Renderingverfahren.
- Eine Menge an Attributen, in denen Metainformationen zu dem Knoten gespeichert werden können – z. B. Daten, die von den States beim Rendering gebraucht werden (Sichtbarkeitsinformationen, Farbwerte etc.).

Durch das sogenannte *Rendering-Channel-Konzept* ist es möglich, die Eigenschaften verschiedener Verfahren miteinander zu kombinieren. Ein Rendering-Channel in MinSG besteht technisch aus einer Zuordnung von einem Channel-Bezeichner zu einem *Renderer* – einer Softwarekomponente zur Darstellung eines Knotens oder Teilbaums des Szenengraphens. Bei der Traversierung des Szenengraphen kann ein Renderer entscheiden, Teilbäumen nicht selbst darzustellen, sondern sie an einen anderen Renderer weiterzugeben, der für einen anderen Channel registriert ist. Als Beispiel lässt sich so ein modulares Näherungsverfahren umsetzten: Auf dem Standard-Channel wird ein Renderer registriert, der Knoten mit einer großen projizierten Größe weiter traversiert, bzw. Blattknoten direkt darstellt. Kleinere Knoten werden an einen anderen Channel weitergereicht. Für diesen ist ein Renderer registriert, der anstatt des Teilbaums eine farbige Bounding-Box darstellt, deren Farbe und Transparenz die Erscheinung des Teilbaums nachbildet, aber wesentlich schneller dargestellt werden kann als die Originalgeometrie (siehe auch Abbildung 3.2). Die Kombination dieser zwei Renderer-Komponenten entspricht damit dem *„Fast rendering of complex environments using a spatial hierarchy“*-Renderingverfahren nach Chamberlain et al. [CDL+96]. Um diese Renderingverfahren zu einem approximativen Occlusion-Culling-Verfahren zu erweitern, kann man nun die erste Renderer-Komponente gegen einen Occlusion-Culling-Renderer austauschen, der verdeckte Knoten verwirft und Knoten mit einer kleinen *sichtbaren* Größe an den anderen Channel gibt. In diesem Beispiel lässt sich natürlich auch die Komponente, die die farbigen Boxen darstellt, durch andere Näherungsverfahren, wie Reliefboards, Textured-Depth-Meshes oder Punktrendering ersetzen. In

dieser Art lassen sich durch die Rendering-Channel aus einzelnen Komponenten auch komplexe Renderingverfahren komponieren.

3.3 Anwendungsinterface

Die eigentliche PADrend-Anwendung baut auf den als Softwarebibliotheken konzipierten Teilen des PADrend-Systems auf. Die Anwendung stellt dem Benutzer ein Interface bereit, um Szenen interaktiv zu begehen, zu bearbeiten und mit den implementierten Renderingverfahren zu experimentieren. Eines der wesentlichen Designziele bei der Entwicklung des Systems ist es, möglichst verschiedene Anwendungsszenarien zu unterstützen, die durch mehrere Entwickler gleichzeitig entwickelt werden. Wird ein neues Renderingverfahren entwickelt und in das System integriert, soll dieses möglichst einfach allen interessierten Nutzern zur Verfügung stehen. Sorgt jedoch ein neues, experimentelles Teilsystem für Probleme, sollen andere Benutzer und Entwickler hiervon nicht beeinträchtigt werden. Um dieses Ziel zu unterstützen, wurde die PADrend-Anwendung als modulares Plugin-System entwickelt. Einzelne Komponenten und Funktionen werden dabei als eigenständige Plugins entwickelt, die über fest definierte Schnittstellen miteinander interagieren. Die zu ladenden Plugins kann der Benutzer unter Beachtung der definierten Abhängigkeiten zwischen den Plugins frei auswählen und so in der Entwicklung befindliche Funktionen ausblenden.

Ein weiteres Ziel bei der Entwicklung des PADrend-Systems ist es, die Entwicklung neuer Werkzeuge und Verfahren zu unterstützen und zeitlich effizient zu gestalten. Auf technischer Ebene stellt sich dabei die für die Bibliotheken verwendete Programmiersprache C++ als problematisch dar. Sie erlaubt zwar das Erstellen von sehr effizienten Programmen, diese müssen bei Änderungen jedoch zumindest neu kompiliert und neu gestartet werden. Die in PADrend verwendeten Plugins sind daher in der Skriptsprache *EScript*[1] entwickelt. Bei EScript handelt es sich um eine objektorientierte Sprache zur Steuerung von C++-Anwendungen, mit JavaScript ähnlicher Syntax. Änderungen an der PADrend-Anwendung können damit ohne ein Neukompilieren durchgeführt werden. Viele Änderungen können auch zur Laufzeit des Programms angewendet werden, ohne das eine betrachtete Szene neu geladen werden muss. Die Herausforderung bei dem Einsatz einer Skriptsprache liegt vor allem in dem deutlichen Geschwindigkeitsunterschied im Vergleich zu nativ ausgeführten C++-Programmteilen. Daher werden geschwindigkeitskritische Funktionen in den C++-Bibliotheken implementiert und aus dem geskripteten Hauptprogramm heraus aufgerufen. Auch wenn für die Aktualisierung der Benutzeroberfläche und für die Reaktion auf Benutzereingaben einige Tausend EScript-Befehle pro Bild bearbeitet werden, wirkt sich dieses schon bei kleinen Szenen nicht auf die durchschnittliche Bildraten des Systems aus, da die Laufzeit durch den eigentlichen Renderingprozess dominiert wird.

In Abbildung 3.3 werden einige der Funktionen der PADrend-Anwendung anhand eines Screenshots gezeigt. Die Funktionen werden im Folgenden kurz erläutert (die Nummern beziehen sich auf die Abbildung).

- Der *Szenengraph-Explorer* erlaubt die Anzeige und Bearbeitung der Eigenschaften von

[1] https://github.com/EScript

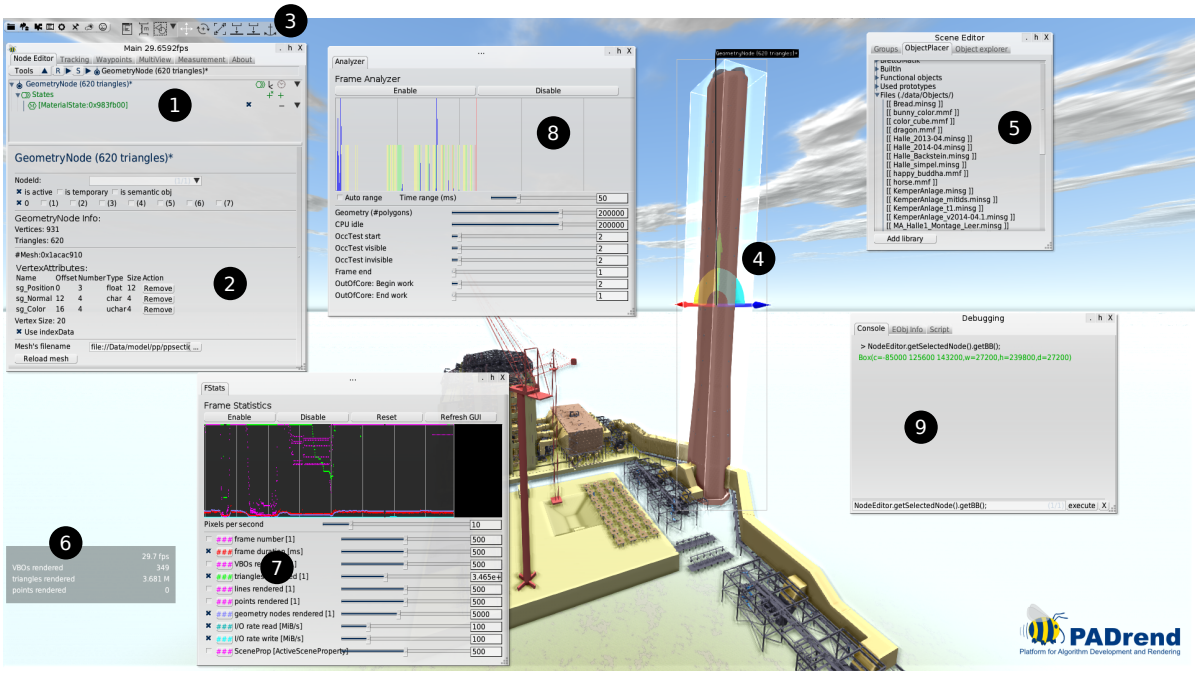


Abbildung 3.3: Bildschirmfoto der Bedienoberfläche der PADrend-Anwendung (Erläuterung zu den mit Ziffern hervorgehobenen Funktionen finden sich im Text)

Knoten im Szenengraph. Dazu gehören unter anderem mit dem Knoten assoziierten *States*: In der Abbildung erkennt man, dass dem selektierten Knoten ein Material-State zugewiesen ist(1). In der Detailansicht (2) werden die Eigenschaften des im Knoten gespeicherten Geometrieobjektes angezeigt und können bearbeitet werden. Mit Hilfe des Szenengraph-Explorers werden die Renderingverfahren im Szenengraph eingefügt und konfiguriert.

- Der *Szenen-Editor* erlaubt eine interaktive Modifikation der geladenen Szene. Eine Reihe von Transformationswerkzeugen (3) steht zur Auswahl, mit denen der selektierte Knoten des Szenengraphen in der Szene bewegt und skaliert werden kann (4). Geometrieobjekte rasten dabei optional an Oberflächen ein, um genaues platzieren möglichst zeiteffizient zu erlauben. Neben dem Modifizieren von existierenden Objekten der Szene können auch neue Objekte einfach aus einer Objektbibliothek in die Szene hineingezogen werden (5). Durch diese Möglichkeiten können einfach Testszenen für experimentelle Evaluierungen erzeugt und bearbeitet werden.

- Für die Evaluierung des aktuellen Renderingprozesses stehen mehrere Funktionen direkt in der Benutzeroberfläche bereit. Einfache statistische Daten über das vorangegangene Bild (wie Renderingzeit, Anzahl der gerenderten Polygone etc.) können in frei konfigurierbaren *Statistikfenstern* angezeigt werden (6). Diese Daten können über den Verlauf mehrerer Bilder als Kurve dargestellt werden (7). Einen genaueren Einblick in das Rendering eines einzelnen Bildes erlaubt das *Frame-Analysetool* (8). Dieses zeigt die einzelnen, während des Renderings durchgeführten Aktionen in Form eines Zeitstrahls an – z. B.

Senden eines Geometrieobjektes an die Grafikkarte (blaue Balken), Starten eines Verdeckungstests (gelbe Balken), Erhalt eines Verdeckungstestergebnisses (grüne Balken) oder das Ende des Frames (roter Balken). Hierdurch lassen sich wichtige Hinweise auf Engpässe bei der Ausführung eines Renderingverfahrens erkennen; beispielsweise, ob die Verdeckungstests eines Occlusion-Culling-Verfahrens den Renderingprozess wesentlich beeinflussen.

- Die Skript-Konsole (9) erlaubt es, während der Laufzeit einzelne EScript-Anweisungen auszuführen. Dies kann beispielsweise zur Fehlersuche verwendet werden. Für komplexere und wiederkehrende Aufgaben stehen weitere Möglichkeiten bereit, um Skript-Dateien zu laden – ohne dafür ein gesondertes Plugin schreiben zu müssen. Dazu gehören prototypische Implementierung von neuen Werkzeugen, Renderingverfahren oder automatisierte Testläufe für experimentelle Evaluierungen.

3.4 Eckdaten des Testsystems

Die in dieser Arbeit verwendete System für die experimentelle Evaluierung besteht aus folgenden Komponenten:

CPU: Intel Core i7-3770 mit 3,4 GHZ (8 virtuelle Kerne), Hauptspeicher: 32 GB, Betriebssystem: Windows 8.1 64 Bit, Grafikkarte: AMD Radeon 7870 (2 GB DDR5), Treiberversion: 13.251, OpenGL Version 4.3, Compiler: MinGW 4.7.1 (tdm64-1), PADrend 1.1 beta

4 Szeneneigenschaften

Als Szeneneigenschaft im Allgemeinen bezeichne ich alle messbaren Eigenschaften von virtuellen Szenen. Eine in der Literatur über Renderingalgorithmen häufig betrachtete Szeneneigenschaft ist die Szenenkomplexität – meist gemessen in der Anzahl der Polygone, aus denen die Szene besteht. Da sich diese Eigenschaft für eine gegebene Szene als Konstante beschreiben lässt, bezeichne ich diese Art von Szeneneigenschaft im Folgenden als *konstante Szeneneigneschaft*. Andere Beispiele sind die räumliche Ausdehnung der Szene oder auch die durchschnittliche Polygondichte. Wird die räumliche Datenstruktur, in der die Szene gespeichert wird, als Teil der Szene gesehen, lassen sich auch verschiedene Eigenschaften der Datenstruktur als konstante Szeneneigenschaft in diesem Sinne auffassen; wie etwa die maximale Tiefe des Baumes, in dem sie gespeichert ist.

Die konstanten Szeneneigenschaften sind in der Literatur ein verbreitetes Mittel um verschiedene Eigenschaften von Renderingalgorithmen abzuschätzen. So wird die maximale Laufzeit von Renderingalgorithmen oft mit der Szenenkomplexität abgeschätzt. Bei einfachem Rendering ohne spezielle Renderingverfahren wächst die maximale Laufzeit beispielsweise linear in der Szenenkomplexität (falls die Szene noch im Hauptspeicher abgelegt werden kann). Bei anderen Verfahren ist es gelungen, einen logarithmischen Zusammenhang zwischen der Szenenkomplexität und der maximalen Laufzeit herzustellen, wenn verschiedene Annahmen an die Struktur der Szene gelten (z. B. [CDL+96, WFP+01]).

Die Möglichkeiten, differenzierte Rückschlüsse aus konstanten Szeneneigenschaften zu ziehen, sind jedoch begrenzt, da das konkrete Verhalten von Renderingalgorithmen auch von der aktuellen Position und Blickrichtung des Betrachters innerhalb der Szene abhängt. Hängen die ermittelten Werte nur von der Szene, konstanten Rahmenbedingungen und der aktuellen Position des Betrachters ab, dann bezeichne ich diese Werte als *lokale Szeneneigenschaften*[1]. Hängen die Werte zusätzlich noch vom zeitlichen Verlauf der Benutzerbewegung in der Szene ab, dann fallen diese nicht in diese Definition. Dies kann bei Eigenschaften von Renderingalgorithmen der Fall sein, die temporale Kohärenz ausnutzen.

Definition 1 (Positionsabhängige Szeneneigenschaft) *Eine positionsabhängige Szeneneigenschaft ist im Allgemeinen definiert als eine Funktion der Positionen einer Szene im dreidimensionalen Raum auf einen Wert eines prinzipiell beliebigen Wertebereichs.*

$$F_{Szene}\colon pos \mapsto \mathit{Wert}, pos \in \mathbb{R}^3$$

Obwohl auch die Blickrichtung und der Öffnungswinkel des Sichtbereichs des Betrachters einen starken Einfluss auf die betrachteten Eigenschaften haben kann, gehe ich im Folgenden

[1] Anmerkung: Im Rahmen dieser Arbeit ist, soweit nichts anderes angebenden, eine lokale Szeneneigenschaft gemeint, wenn der Begriff *Szeneneigenschaft* verwendet wird.

von einer vereinfachten Betrachtung des omnidirektionalen Betrachters mit einer kompletten Rundumsicht aus. Diese Betrachtungsweise vereinfacht den Umgang mit positionsabhängigen Eigenschaften und ist für viele der angestrebten Anwendungen ausreichend.

4.1 Anforderungen an praktisch auswertbare Szeneneigenschaften

Aus der sehr offenen Definition von positionsabhängigen Szeneneigenschaften ergibt sich eine große Menge an denkbaren Ausprägungen von Funktionen. Von diesen lassen sich jedoch nur wenige praktisch für die in dieser Arbeit verfolgten Ziele der Bewertung des Verhaltens von Renderingverfahren und deren Verbesserung einsetzen. Im Folgenden werden daher einige Anforderungen an die hier untersuchten Eigenschaften bezüglich ihrer praktischen Relevanz aufgelistet.

4.1.1 Effiziente Bestimmbarkeit

Um eine Szeneneigenschaft praktisch einsetzen zu können, ist es notwendig, dass sie für eine gegebene Position effizient bestimmt werden kann. Effizient in diesem Zusammenhang bedeutet, dass sowohl die Laufzeit, als auch der benötigte Speicherplatz subquadratisch in der Komplexität der Szene wachsen sollte; und konkret, dass die Auswertung eines Samples nur einige Millisekunden bis hin zu wenigen Sekunden dauern sollte. Die Laufzeiten aller später in dieser Arbeit vorgestellten Methoden zur globalen Näherung der positionsabhängigen Szeneneigenschaften werden maßgeblich vom Aufwand zur Bestimmung der Szeneneigenschaften bestimmt. Ist dieser zu hoch, lassen sich die Methoden praktisch nicht für größere Szenen einsetzen. Um dieses Kriterium zu erfüllen, bieten sich vor allem Szeneneigenschaften an, die durch Rendern der Szene an der gegebenen Position ermittelt werden können. Dies ist für Szenen mit bis zu mehreren Millionen Polygonen oft in wenigen Millisekunden möglich.

4.1.2 Begrenzung des Wertebereichs

Um die Werte der Funktion praktisch speichern und handhaben zu können, darf der Wertebereich der Funktion nicht zu groß sein. Ein einzelner Wert sollte aus wenigen Ganzzahlen oder Fließkommazahlen mit festem Speicherbedarf (z. B. 32 Bit) bestehen. Für praktisch relevante Szenen ist dies z. B. die Anzahl der sichtbaren Objekte (oder Polygone) kodiert als einzelner Wert. Als praktisch gerade noch handhabbare Grenze haben sich Vektoren als Wert herausgestellt, die für jedes Objekt der Szene einen Wert besitzen. Die Eigenschaft beispielsweise, die die Position auf ein von dort aus gerendertes und entsprechend kodiertes Bild der Szene abbildet (z. B. mit einer Millionen Pixel), ist nicht mehr praktisch handhabbar.

4.1.3 Praktische Gutmütigkeit des Wertebereichs

Ein wesentliches Kriterium für eine praktikabel einsetzbare Szeneneigenschaft ist ihre praktische Gutmütigkeit. Damit bezeichne ich das Verhalten, dass sich die Werte in realistischen

Szenen, bei minimaler Veränderung der Position meistens nicht, oder nur wenig, ändern. Dies ist essentiell, da die später vorgestellten Verfahren zur Näherung der globalen Verteilung darauf basieren, dass der Wert einer Szeneneigenschaft an einer Position auch mit einiger Sicherheit als Näherung der Werte seiner direkten Umgebung dienen kann.
Als Beispiel bietet sich wieder die Eigenschaft der Anzahl der sichtbaren Objekte an einer Position an: Für einen Großteil der praktisch relevanten Szenen gilt hier: Wenn an einer Stelle eine Anzahl von Objekten sichtbar ist, dann sind in der näheren Umgebung an den meisten Positionen eine ähnliche Zahl von Objekten sichtbar. Ein Beispiel für eine Szene, in der sich die Sichtbarkeit in einigen Bereichen nicht gutmütig verhält, ist die in Abbildung 4.1 dargestellte „Schlüsselloch"-Situation. Diese enthält Bereiche mit kleinem Volumen (im 2-D-Beispiel mit kleiner Fläche), die eine große Differenz bezüglich der Anzahl sichtbarer Objekte zu ihren benachbarten Bereichen aufweisen. Auch in realistischen Szenen können Bereiche mit ähnlicher Charakteristik auftreten; die problematischen Bereiche sind dann jedoch oft räumlich begrenzt.

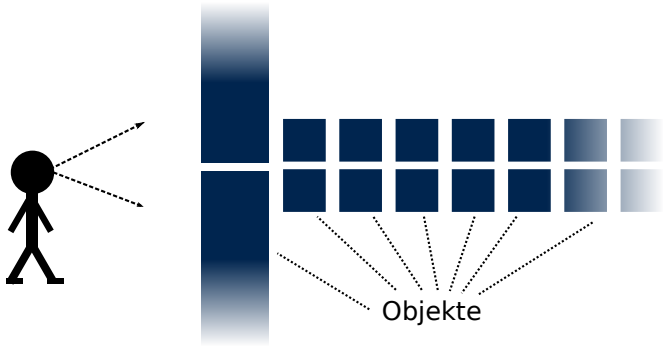


Abbildung 4.1: „Schlüsselloch"-Situation: Bei geringer Bewegung des Betrachters ändert sich die Anzahl sichtbarer Objekte deutlich.

4.1.4 Determinismus

Unter dem deterministischen Verhalten einer Szeneneigenschaft verstehe ich, dass ihr Wert nur von der Szene, den Rahmenbedingungen (wie der eingesetzten Hardware und Software, Bildschirmauflösung etc.) abhängt und nicht von zusätzlichen Variablen. Gerade bei Eigenschaften, die auf Laufzeitmessungen beruhen, ist dies praktisch kaum strikt einzuhalten. Hier reichen jedoch einfache Maßnahmen, wie das wiederholte Durchführen der Messung und Wahl des Medians, um durch externe Störungen verursachte Ausreißer auszugleichen. Größere Probleme bereiten hier Eigenschaften, die sich auf Algorithmen beziehen, welche die temporale Kohärenz durch die kontinuierliche Bewegung des Betrachters mit einbeziehen – d. h. Algorithmen, die den Zustand der vorangegangenen Frames für die Berechnung des nächsten ausnutzen (wie z. B. der CHC oder der CHC++; siehe Abschnitt 2.1.2). Beträgt die Länge der Zeitspanne, aus der zurückliegende Daten noch weiterverwendet werden, nur wenige Frames, dann lassen sich diese Eigenschaften dennoch in modifizierter Form verwenden. Darauf gehe ich in Abschnitt 4.2.3 genauer ein.

4.2 Betrachtete Szeneneigenschaften

In diesem Abschnitt werden einige Szeneneigenschaften definiert und ihre Auswertungsmethoden erläutert. Beginnend wird zunächst die Eigenschaft der *exakten Sichtbarkeit* beschrieben. Diese wird zwar im Rahmen dieser Arbeit nicht praktisch bestimmt, bietet jedoch die Grundlage vieler weiterer Überlegungen.

4.2.1 Exakte Sichtbarkeit

Die exakte Sichtbarkeit beschreibt die Anzahl der Polygone, von denen es eine direkte Sichtlinie (ohne Verdeckung) zur Betrachterposition gibt (siehe Abschnitt 2.1.1).

Um die exakte Sichtbarkeit gemäß der Definition der positionsabhängigen Szeneneigenschaft zu kodieren, lässt sie sich wie folgt definieren:

Definition 2 (Szeneneigenschaft: Anzahl geometrisch sichtbarer Polygone)

$$\begin{aligned} &ExakteSichtbarkeit_{Szene}\colon pos \mapsto visPoly, \\ &\quad pos \in \mathbb{R}^3, \\ &\quad visPoly = |\textit{Polygone aus } Szene \textit{ mit direkter Sichtlinie zu } pos| \in \mathbb{N}. \end{aligned}$$

Ebenso, wie für die Anzahl der sichtbaren Polygone, lässt sich auch die Menge der Polygone, die an einer Position sichtbar sind, als positionsabhängige Szeneneigenschaft kodieren:

Definition 3 (Szeneneigenschaft: Menge geometrisch sichtbarer Polygone)

$$ExakteSichtbarkeitsMenge_{Szene}\colon pos \mapsto \begin{pmatrix} p_0 \\ p_1 \\ \dots \\ p_n \end{pmatrix}, pos \in \mathbb{R}^3,$$

$$n = \textit{Anzahl der Polygone in } Szene,$$

$$p_i = \begin{cases} 0 & \textit{keine Sichtlinie von } pos \textit{ zum } i\textit{-ten Polygon von } Szene \\ 1 & \textit{sonst.} \end{cases}$$

Die Relevanz der exakten Sichtbarkeit ergibt sich primär daraus, dass sie die Grundlage für alle Betrachtungen von Sichtbarkeit bildet. Die in dieser Arbeit untersuchten Renderingalgorithmen werden praktisch jedoch eher von der Pixelsichtbarkeit beeinflusst.

4.2.2 Pixelsichtbarkeit

Die Definition der Pixelsichtbarkeit als Szeneneigenschaft entspricht im Wesentlichen der exakten Sichtbarkeit – bezieht sich jedoch auf die, für das Rendering eher relevante, Pixelsichtbarkeit: Ein Teil der Szene ist sichtbar, wenn er zu der Farbe (oder genauer: zu einem Tiefenwert) eines Pixels beiträgt. Aus den technischen Rahmenbedingungen des Renderingprozesses und der Verwendung eines omnidirektionalen Betrachters ergeben sich mehrere Besonderheiten:

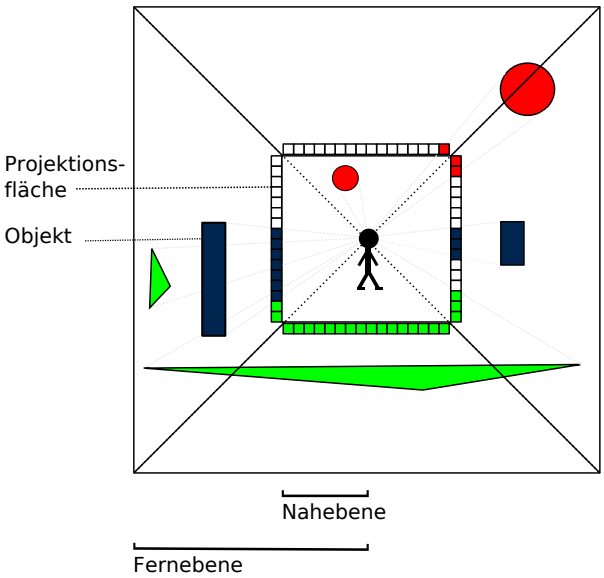


Abbildung 4.2: Schematische Darstellung der Bestimmung der Pixelsichtbarkeit bei einem omnidirektionalen Betrachter mit mehreren Frusta. Im Bild sind vier der sechs Frusta und der entsprechenden Projektionsflächen abgebildet, mit je 16 Pixeln Auflösung. Das kleinere runde Objekt liegt vollständig vor den Nahebenen der Frusta und wird daher nicht als sichtbar klassifiziert.

- Anstatt von Polygonen werden bei vielen Renderingalgorithmen Objekte als kleinste Einheit betrachtet; daher wird auch bei der Szeneneigenschaft die Sichtbarkeit von Objekten anstatt von Polygonen betrachtet. Da die Zuordnung von Polygonen zu Objekten jedoch variabel ist, kann auch in der Szene jedes Polygon als Objekt definiert werden, um so die Sichtbarkeit von einzelnen Polygonen zu erhalten.

- Anders als bei der geometrischen Sichtbarkeit, ist die Anzahl der sichtbaren Objekte durch die Auflösung der Projektionsfläche beschränkt. Die Auflösung der verwendeten Projektionsfläche ist daher ein Teil der *Renderingparameter*, die zur Definition dieser Szeneneigenschaft gehört.

- Ebenso verhält es sich mit der Nah- und Fernebene des Frustums, durch den der zu rendernde Bereich um den Betrachter herum eingeschränkt wird. Die Entfernung zur Nahebene ist daher Teil der Rahmenbedingungen; die Entfernung zur Fernebene wird zur Vereinfachung so gewählt, dass die komplette Szene eingeschlossen ist.

- Da für die Szeneneigenschaft ein omnidirektionaler Betrachter angenommen wird, ist eine naheliegende Idee, die Szene nicht auf die Pixel einer üblichen geraden Projektionsfläche zu projizieren, sondern auf die Oberfläche einer Kugel um den Betrachter. Um jedoch für die Bestimmung der Sichtbarkeit die Standardrenderingverfahren nutzen zu können, verwende ich als grobe Annäherung für diese Kugel die Oberfläche eines Würfels. Jede der sechs Seiten ist definiert durch ein Frustum und eine eigene Projektionsfläche. Trägt ein Objekt mindestens einen Pixel zu einer der Projektionsflächen bei, gilt das Objekt

als sichtbar für die gegebene Position. Abbildung 4.2 zeigt zur Veranschaulichung eine zweidimensionale Darstellung der Projektion der Szene unter Verwendung von vier Frusta.

Unter Einbeziehung dieser Aspekte ist die Szeneneigenschaft der *Anzahl sichtbarer Objekte* (unter Pixelsichtbarkeit) definiert als:

Definition 4 (Szeneneigenschaft: Anzahl sichtbarer Objekte)

$$PixelSichtbarkeit_{Szene,Renderingparameter} : pos \mapsto visObj,$$
$$pos \in \mathbb{R}^3,$$
$$visObj = \left| \begin{array}{c} \textit{Objekte aus } Szene\textit{, die einen Tiefenwert zu einer} \\ \textit{der sechs Projetionsflächen um } pos \textit{ beitragen} \end{array} \right| \in \mathbb{N}.$$

Und analog die Menge der sichtbaren Objekte:

Definition 5 (Szeneneigenschaft: Menge sichtbarer Objekte)

$$PixelSichtbarkeitsMenge_{Szene,Renderingparameter} : pos \mapsto \begin{pmatrix} p_0 \\ p_1 \\ \dots \\ p_n \end{pmatrix}, pos \in \mathbb{R}^3,$$
$$n = \textit{Anzahl der Objekte in } Szene,$$
$$p_i = \begin{cases} 0 & \textit{das } i\textit{-te Objekt in } Szene \textit{ trägt zu keinem} \\ & \textit{der Tiefenwerte der sechs Projektionsflächen um } pos \textit{ bei,} \\ 1 & \textit{sonst.} \end{cases}$$

Bestimmung der Pixelsichtbarkeit Technisch erfolgt die Bestimmung der Pixelsichtbarkeit mittels hardwareunterstützter Verdeckungsanfragen. Um die sichtbaren Objekte zu identifizieren, wird zunächst für jede der Seiten eines den Betrachter umschließenden Würfels ein Frustum konstruiert, das vom Betrachter aus die Seite vollständig abdeckt. Nun wird mit diesen Frusta die Szene in den Tiefenpuffer gerendert. Im Anschluss wird der Tiefentest so konfiguriert, dass nur Pixel akzeptiert werden, die den Tiefenwert aufweisen, der bereits an der Position im Puffer steht. Jetzt wird für jedes Objekt eine hardwareunterstützte Verdeckungsanfrage gestartet, die ermittelt, wie viele Pixel den Tiefentest bestehen, und damit das Objekt erneut gerendert. Das Ergebnis der Verdeckungsanfrage sagt daraufhin aus, ob mindestens einer der Pixel des projizierten Objektes nicht durch ein anderes Objekt verdeckt wurde. Die Menge aller Objekte, die diesen Test bestehen, wird gesammelt und zurückgegeben.

4.2.3 Renderingzeit

Die Zeit, die ein Renderingalgorithmus für die Darstellung eines Bildes benötigt, gehört bei der Untersuchung von Renderingalgorithmen zu den wichtigsten Merkmalen. Möchte man

diese Zeit, anders als das üblicherweise verwendete Messen entlang eines Kamerapfades, als positionsabhängige Szeneneigenschaft betrachten, sind wiederum einige Besonderheiten zu beachten:

- Wie bei der Pixelsichtbarkeit stellt sich die Frage, wie die Renderingzeit von einem Algorithmus für Rendering mit einem normalen Frustum für einen omnidirektionalen Betrachter ermittelt werden kann. Mein Ziel bei der Definition der Eigenschaft ist es, möglichst einfach eine Näherung des höchsten Wertes zu erhalten, den ein Betrachter mit beliebigem Blickwinkel an der Position erreichen kann. Auch wenn andere Zielvorgaben denkbar sind (z. B. der Durchschnitt oder das Minimum), ist das Maximum bei vielen Untersuchungen der interessanteste Aspekt, da es die maximale Bildrate beschränkt.
  Zur Ermittlung verwende ich hier wieder mehrere Renderingvorgänge in unterschiedliche Richtungen, wobei jeder Renderingvorgang eine eigene Zeit liefert. Im Normalfall werden wieder die sechs Seiten eines den Betrachter umgebenden Würfels verwendet.

- Durch die Verwendung sich nicht oder nur kaum überlappender Frusta ergibt sich das Problem, dass komplexe Teile der Szene auf mehrere Frusta aufgeteilt werden können. Dadurch könnte sich eine wesentlich geringere Renderingzeit ergeben, als sie durch einen freien Betrachter erreicht wird, der den kompletten Bereich von der gleichen Position aus komplett im Frustum hat. Um diesem Effekt entgegenzuwirken, verwende ich für die Renderingzeit daher überlappende Frusta mit einem größeren Öffnungswinkel, so dass die Bereiche an den Rändern auf mehr als einer Seite in Betracht gezogen werden. Wenn sechs Richtungen verwendet werden, zeigen sich bereits ab 120 Grad Öffnungswinkel kaum mehr merkliche Auswirkungen dieses Effektes. Größere Öffnungswinkel zeigen keine wesentliche Verbesserung, führen aber zu einer immer stärkeren Verzerrung der gerenderten Bilder.

- Ein Problem ergibt sich für die Messung der Renderingzeit von Algorithmen, die die temporale Kohärenz zwischen aufeinanderfolgende Frames ausnutzen (siehe auch: Anforderung Determinismus in Abschnitt 4.1.4). Beispielsweise nutzen viele Occlusion-Culling-Algorithmen die Sichtbarkeitsinformationen der vorangegangenen Frames als Basis für die aktuellen Berechnungen aus, was sich in der Renderingzeit niederschlägt. Um für die Szeneneigenschaft der Renderingzeit reproduzierbare Werte zu erhalten, die unabhängig von den vorher untersuchten Messungen sind, führe ich zwei Varianten der Eigenschaft ein: *unkonditionierte Renderingzeit* und *konditionierte Renderingzeit*. Bei der unkonditionierten Renderingzeit werden explizit alle Informationen aus vorangegangenen Frames vor jeder Messung entfernt, was im Normalfall die schlechteste Renderingzeit an dieser Position provoziert. Bei der konditionierten Renderingzeit werden jeweils so viele Frames mit den gleichen Einstellungen gerendert, bis keine weitere Verbesserung mehr durch zusätzliche Daten zu erwarten ist. Dies entspricht im Normalfall der besten zu erreichenden Renderingzeit an der Position. Je nach Anwendungsfall muss dann die passende Version ausgewählt werden. Die Grenzen dieser Methode sind erreicht, wenn ohne Konditionierung nur schwerlich überhaupt ein Bild erstellt werden kann oder wenn im konditionierten Fall die Zeit für eine einzelne Messung nicht mehr

praktikabel ist. Bei einem Out-Of-Core-Verfahren könnte es sein, dass für den unkonditionierten Fall alle Daten aus dem Speicher gelöscht werden und ein leeres Bild gerendert wird. Im konditionierten Fall müsste man ggf. mehrere Sekunden warten, bis alle Daten tatsächlich in den Speicher geladen wurden.

- Um äußere Störeinflüsse auszugleichen werden die Laufzeitmessungen immer mehrfach durchgeführt und der Median verwendet.

Definition 6 (Szeneneigenschaft: (Un-)Konditionierte Renderingzeit)

$$(Un-)KondRenderingzeit_{Szene,Algorithmus,Renderingparameter} : pos \mapsto t,$$
$$pos \in \mathbb{R}^3, t \in \mathbb{R}.$$

Wobei t die maximale Zeit in Sekunden ist, die beim Rendern der Szene mit dem gegebenen (un-)konditionierten Algorithmus und Renderingparametern für die sechs Hauptrichtungen und mit einem Kameraöffnungswinkel von $120°$ bei mehrfacher Wiederholung im Median benötigt wird.

Bei Algorithmen ohne die Ausnutzung von temporaler Kohärenz wird die Szeneneigenschaft *Renderingzeit* genannt, da hier die Konditionierung keine Relevanz hat.

4.2.4 Anzahl von Operationen

Eine weitere Möglichkeit, das Verhalten von Renderingalgorithmen mithilfe von positionsabhängigen Szeneneigenschaften zu beschreiben, ist die Verwendung der Anzahl von verschiedenen Operationen als Eigenschaft, die der Algorithmus zum Rendern durchführt. Dabei definiert jede spezifische Operation eine eigene Szeneneigenschaft, was dadurch eine ganze Klasse von verschiedenen algorithmenspezifischen Eigenschaften eröffnet. Einige für die Anwendung relevante Beispiele sind die Anzahl der gerenderten Polygone, die Anzahl durchgeführter Verdeckungsanfragen (für Verdeckungstests) oder die Anzahl der Traversierungsschritte durch die Datenstruktur der Szene.

Generell gelten für alle Eigenschaften dieser Klasse die gleichen Rahmenbedingungen wie für die Renderingzeit: jeweils Messungen für (mindestens) sechs Richtungen, Verwendung von überlappenden Frusta und Unterscheidung zwischen konditionierter oder unkonditionierter Anwendung der Algorithmen. Im Einzelfall muss man entschieden, ob als Wert für die Funktion, wie bei der Renderingzeit, das Maximum der Einzelwerte die größte Relevanz hat, oder ob man hier einen anderen Wert verwenden sollte.

4.2.5 Bildqualität

Viele Renderingalgorithmen erzeugen im Vergleich zur Darstellung der Originalgeometrie ein fehlerhaftes Bild. Beispielsweise können approximative Occlusion-Culling-Verfahren auch einen Teil der sichtbaren Geometrie verwerfen oder Näherungsverfahren ersetzen beim Rendering Teile der Szene durch effizienter renderbare Ersatzrepräsentationen. Für die Evaluierung

solcher Verfahren ist die Bildqualität als Szeneneigenschaft von großer Bedeutung. Zur automatischen Bestimmung benötigt man zum einen ein Referenzbild und zum anderen eine Funktion, die den Unterschied in zwei Bildern als einen Qualitätswert ausdrücken kann. Das Referenzbild erhält man, indem die Szene mit einem konservativen Renderingalgorithmus (dessen Ausgabe also der Originalgeometrie entspricht) gerendert wird. Als Qualitätsbewertungsfunktion kann man z. B. das einfache Zählen der andersfarbigen Pixel verwenden. Deutlich aussagekräftigere Ergebnisse in Hinblick auf die von einem Betrachter wahrgenommene Qualität erzielt man jedoch mit komplexeren Funktionen, wie dem Maß der *Strukturellen Ähnlichkeit*, wie es von Wang et al. [WBSS04] für die automatische Qualitätsbewertung von Bildkompressionsverfahren vorgeschlagen wurde. Um die Auswirkungen großflächiger Fehler hervorzuheben, kann das Verfahren so ergänzt werden, dass die intern ausgewerten Fehlerbilder zusätzlich hierarchisch gefiltert werden (nach [Bur81]). Ein einzelner absoluter Wert der Bewertungsfunktion für zwei Bilder hat dabei wenig Aussagekraft, er erlaubt aber im Vergleich mit anderen Werten eine Aussage darüber, welches Bild eine höhere Bildqualität aufweist.

Die Messung zur Bestimmung der Szeneneigenschaft läuft wiederum äquivalent mit mehreren Richtungen, wobei die Szene jeweils zwei mal gerendert wird – mit dem betrachteten Algorithmus und als Referenz. Der verwendete Wert ist das Minimum der Einzelwerte der Richtungen, wenn es sich um ein Qualitätswert handelt; das Maximum, wenn es sich um einen Fehlerwert handelt.

Definition 7 (Szeneneigenschaft: Bildqualität)

$$Bildqualität_{Szene,Algorithmus,Renderingparameter,Qualitätsbewertungsfuntion}:$$
$$pos \mapsto q, pos \in \mathbb{R}^3, q \in \mathbb{R}.$$

Wobei q der minimale Wert der Qualitätsbewertungsfunktion für sechs Eingaben ist. Die Eingaben bestehen aus je zwei Bildern, die beim Rendern der Szene mit dem gegebenen Algorithmus und Renderingparametern und einem konservativen Referenzalgorithmus, in die sechs Hauptrichtungen und mit einem Kameraöffnungswinkel von $120°$ an Position pos erzeugt werden.

4.2.6 Kombinierte Szeneneigenschaften

Eine weitere Klasse von Szeneneigenschaften ergibt sich aus der Möglichkeit, die bereits beschriebenen Eigenschaften z. B. durch Subtraktion oder andere Operationen zu kombinieren. Zum Effizienzvergleich zweier Algorithmen lässt sich beispielsweise die Eigenschaft konstruieren, die die Differenz der Renderingzeit der zwei Algorithmen für eine gegebene Position beschreibt. Auch wenn eine bestimmte Kombination inhaltlich Sinn ergibt, müssen dennoch weitere Punkte beachtet werden:

- Durch die Kombination kann für die entstehende Verteilung der Werte deren praktische Gutmütigkeit leiden (siehe Abschnitt 4.1.3). Dies kann zu Problemen bei der Auswertung der Eigenschaften führen. Da dieser Effekt nur schwer im Vorfeld abzusehen ist, ist besonders für komplizierte Kombinationen eine kritische experimentelle Prüfung angeraten.

- Mehrere Werte einer kombinierten Eigenschaft können ursprünglich aus Messungen in verschiedene der sechs Richtungen entstanden sein. Dies muss bei der Konzeption der kombinierten Eigenschaft mit bedacht werden.

5 Globale Näherung von Szeneneigenschaften

Im vorherigen Kapitel wurden verschiedene positionsabhängige Szeneneigenschaften beschrieben, die für eine gegebene Betrachterposition in der Szene den entsprechenden Wert liefern. Für die meisten Eigenschaften lässt sich die Berechnung eines einzelnen Wertes dabei innerhalb der Zeit durchführen, die für die Darstellung einiger weniger Frames benötigt wird. Für die Bewertung von Renderingalgorithmen oder die Verbesserung des Renderings während des Walkthroughs reicht die alleinige Möglichkeit, die Szeneneigenschaften an diskreten Positionen auswerten zu können, jedoch nicht aus: Sollen die Werte einer Szeneneigenschaft zur Laufzeit zur Verbesserung des Renderings benutzt werden, dauert die Ermittlung in den meisten Fällen länger als der zu erwartende Laufzeitgewinn. Für die quantitative Bewertung von Renderingalgorithmen möchte man nicht nur Aussagen nicht über einzelne Werte der Szeneneigenschaft treffen, sondern auch über die Verteilung der Werte im gesamten Raum einer gegebenen Szene zu treffen.

In diesem Kapitel stelle ich Methoden vor, wie eine *globale Näherung* einer Szeneneigenschaft praktisch ermittelt werden kann: eine Abbildung eines Teils des Raumes auf den Wertebereich der betrachteten Szeneneigenschaft. Diese Abbildung wird in Form einer räumlichen Datenstruktur gespeichert, bei der diskreten Bereichen des Raumes ein Wert zugewiesen wird (oder alternativ auch ein Wertegradient). Als Ausgangspunkt für die zugewiesenen Werte dient eine Menge an Stichproben der verwendeten Szeneneigenschaft. Diese globale Näherung bietet dann zum einen effiziente Punktanfragen, zum anderen aber auch Bereichsanfragen und die Möglichkeit zum Bestimmen von Werteverteilungen.

Die Näherung der Szeneneigenschaft kann dabei entweder für einen dreidimensionalen Bereich der Szene (z. B. einen die komplette Szene umschließenden Quader) erfolgen, oder aber man beschränkt sich auf die Näherung für einen zweidimensionalen Schnitt durch die Szene. Der 3-D-Fall hat den Vorteil, dass man für alle zulässigen Betrachterpositionen zur Laufzeit eine Näherung der untersuchten Szeneneigenschaft bestimmen kann. Auch eine statistische Auswertung kann so über alle möglichen Betrachterpositionen erfolgen. Der Vorteil der Betrachtung eines 2-D-Bereichs liegt darin, dass mit einer geringeren Anzahl von Sampling-Punkten eine Näherung guter Qualität ermittelt werden kann, die sich darüber hinaus noch einfach und anschaulich visualisieren lässt. Dazu muss jedoch ein charakteristischer Schnitt durch die Szene gewählt werden, was jedoch einfacher möglich ist als z. B. einen charakteristischen Kamerapfad bei der klassischen Evaluierung von Renderingalgorithmen zu wählen.

Im Folgenden erläutere ich zunächst einige Anforderungen, die ich an ein spezifisches Verfahren zu Erzeugung einer globalen Näherung identifiziert habe. In Abschnitt 5.2 stelle ich den allgemeinen Ansatz von Sampling-basierten Verfahren zur Bestimmung der globalen

Näherung vor, bevor ich konkrete Verfahren zur Ermittlung der Näherung genauer betrachte (Abschnitte 5.3, 5.4 und 5.5). Abschließend zeige ich in Abschnitt 5.6 mehrere Möglichkeiten auf, die Daten einer globalen Näherung grafisch und statistisch auszuwerten.

5.1 Anforderungen

Das generelle Ziel bei der globalen Näherung einer Szeneneigenschaft ist der Aufbau einer Datenstruktur, die zu Positionen innerhalb der Szene eine Näherung der entsprechenden Szeneneigenschaft liefert. Für die praktische Anwendbarkeit einer solchen Datenstruktur habe ich mehrere Anforderungen identifiziert: sowohl an die Datenstruktur selbst, als auch an den Prozess, um diese aufzubauen. Diese umfassen Anforderung an die Laufzeitkomplexität von Operationen sowie an die Speicherplatzkomplexität der Datenstrukturen. Obwohl sich auch das Worst-Case-Verhalten beschränken lässt, beziehe ich mich im Folgenden primär auf eine anwendungsbezogene Interpretation von Effizienz.

5.1.1 Akzeptabler Zeitaufwand im Preprocessing

Ein wesentlicher Punkt, der über die praktische Anwendbarkeit von Verfahren zur globalen Näherung von Szeneneigenschaften entscheidet, ist die Zeit, die benötigt wird, um die Näherung für realistische Eingaben zu ermitteln. Konkret bedeutet dies, dass eine Näherung einer Funktion für eine Szene bestehend aus mehreren Millionen Polygonen auf aktueller Hardware in wenigen Minuten oder Stunden erstellt werden können soll – und das mit einer für den jeweiligen Anwendungsfall passenden Qualität.

5.1.2 Kompakter Speicherplatz

Ähnlich wie bei der Laufzeit, ist auch der Speicherplatzbedarf der Datenstruktur zur Speicherung der globalen Näherung durch die Anwendung beschränkt. Der benötigte Speicherplatz für die Näherung sollte dabei maximal in einer ähnlichen Größenordnung wie die untersuchte Szene liegen. Die exakte Ermittlung der geometrischen Sichtbarkeit auf Basis der VSP (siehe Abschnitt 2.1.1) eignet sich daher Aufgrund der hohen Komplexität nicht als Grundlage für eine hier untersuchte Szeneneigenschaft.

5.1.3 Effiziente Punktabfragen

Wenn der genäherte Wert der Szeneneigenschaft zur Laufzeit für die Verbesserung des Renderingprozesses genutzt werden soll, ist es wichtig, den zusätzlichen Laufzeitaufwand für eine Punktanfrage (Lesen eines Wertes zu einer gegebenen Position) so gering wie möglich zu halten. Nur so ist es möglich, einen Nutzen aus der Kenntnis dieser zusätzlichen Informationen zu erlangen. Idealerweise sind hier sublineare Laufzeiten in der Größe der Datenstruktur, mit kleinen Konstanten, wünschenswert.

5.1.4 Gute Qualität der Näherung

Die Qualität der Näherung ergibt sich aus dem akkumulierten Fehler des Wertes der Näherung gegenüber dem tatsächlichen Wert der untersuchten Funktion für alle Positionen. Auch wenn sich für die Qualität einer Näherung, die sich aus einem bestimmten Verfahren ergibt, nur schwer eine konkrete Anforderung aufstellen lässt, so lassen sich mehrere Verfahren jedoch gut miteinander vergleichen. Dazu werden die erzielten Qualitäten mehrerer Verfahren im Verhältnis zu den eingesetzten Ressourcen (Laufzeit und Speicher) miteinander verglichen. Ein wesentliches Ziel beim Entwurf der im Weiteren vorgestellten Verfahren ist es, eine möglichst hohe Qualität für die eingesetzte Zeit zu erreichen; eine entsprechende experimentelle Untersuchung folgt in Kapitel 5.7.

5.1.5 Parametrierbar, aber robust

Eine Anforderung an praktikable Näherungsverfahren ist, dass sie so parametrierbar sind, dass die eingesetzte Laufzeit möglichst direkt (z. B. über einen einzelnen Wert) gesteuert werden kann, um so Laufzeit gegen Qualität tauschen zu können. Darüber hinaus sollten sich die Verfahren weitestgehend robust in Bezug auf die genauen Werte von Parametern verhalten; d.h. nicht optimal gewählte Parameterwerte dürfen nicht zu einschneidenden Qualitätsverlusten in der Näherung führen. Nur so lassen sich die Verfahren einfach ohne große Anpassungsschritte und ohne gesondertes Expertenwissen seitens des Anwenders praktisch einsetzen.

5.1.6 Einfachheit

Um sich tatsächlich im Bereich der experimentellen Algorithmenevaluation und Anpassung durchzusetzen, sollen die Verfahren vom Aufbau möglichst einfach sein. Dadurch soll die Hürde gesenkt werden, dass sie in existierende Softwaresysteme integriert werden.

5.2 Allgemeine Form des Sampling-Ansatzes

Im Folgenden Abschnitt beschreibe ich die allgemeine Idee, die der Sampling-basierten, globalen Näherung von positionsabhängigen Szeneneigenschaften zugrunde liegt. Konkrete Ausprägungen dieser Technik stelle ich später in den Abschnitten 5.3 und 5.4 vor.

5.2.1 Aufbau der Datenstruktur

Das Ziel bei der Sampling-basierten globalen Näherung von positionsabhängigen Szeneneigenschaften ist eine vollständige, disjunkte Aufteilung des untersuchten Raumes (ein Teilbereich der Szene), in einfache, konvexe Regionen. Diesen Regionen ist jeweils ein konstanter Wert aus dem Wertebereich der untersuchten Szeneneigenschaften zugeordnet, welcher als Näherung für den Funktionswert aller Positionen in der Region verwendet wird. Die Form der Regionen richtet sich nach dem untersuchten Bereich der Szene. Wird nur ein zweidimensionaler Schnitt durch die Szene untersucht, sind die Regionen recht- oder dreieckig. Bei der Untersuchung

dreidimensionaler Bereiche sind die Regionen entweder quader- oder tetraederförmig. Um die Anforderung der effizienten Punktanfragen zu erfüllen (siehe Abschnitt 5.1.3), können die Regionen in einer geeigneten räumlichen Datenstruktur gespeichert werden. Im zweidimensionalen Fall kann beispielsweise ein Quadtree oder im dreidimensionalem Fall ein Octree verwendet werden. Bei einer gleichmäßigen Größenverteilung der Regionen lassen sich damit Punktanfragen in erwartet logarithmischer Zeit in der Anzahl der Regionen durchführen. Als Erweiterung können den Regionen auch, anstatt eines einzelnen Wertes, Werte die die Ecken der Region zugewiesen werden. Die Werte der Positionen im Inneren werden dann entsprechend ihres Abstandes zu den Ecken interpoliert.

Der generelle Ablauf wird in Algorithmus 5.1 beschreiben:

Algorithmus 5.1: Allgemeine Formulierung des Sampling-Algorithmus für die globale Näherung positionsabhängiger Szeneneigenschaften

Input: *Szene* s , *Region* r , *positionsabhängige Szeneneigenschaft* f
Output: *Datenstruktur mit Näherung von* f *im Bereich* r

```
1 var positionen ← erzeugeEineMengeVonPositionen( r )
2 var werte ← { }
3 foreach( position ∈ positionen )
4     werte ← werte ∪ {(position, f_s(position))}
5 var bereiche ← erzeugeDisjunkteAufteilungVon ( r )
6 foreach( bereich ∈ bereiche )
7       var bereichsWert ← fasseWerteZusammen( w | (pos, w) ∈ werte ∧ pos ∈ bereich )
8       assoziiere bereich mit bereichsWert
9 return bildeHiereachischeDatenstruktur( bereiche )
```

Diese sehr allgemeine Beschreibung des Sampling-Algorithmus bietet mehrere Freiheitsgrade:

1. Welche räumliche Datenstruktur wird letztlich aufgebaut?
2. Wie werden die Werte der Samples einer Region zum Wert der Region zusammengefasst?
3. Wie werden die Sampling-Positionen innerhalb der gegebenen Region gewählt?
4. Wie wird die Gesamtregion in Teilregionen aufgeteilt?

Der erste Punkt ist praktisch relativ unkritisch für das Ergebnis des Sampling-Prozesses, wenn eine der üblichen hierarchischen Datenstrukturen gewählt wird. Lediglich die Anfragezeit für Punkt- und Bereichsanfragen wird marginal durch diese Entscheidung beeinflusst. Im Folgenden gehe ich daher immer von einem Octree im dreidimensionalen und einem Quadtree im zweidimensionalen Fall aus.

Für die Zusammenfassung der Samples zur Bestimmung des Wertes der entsprechenden Region verwende ich, wenn nichts weiteres angegeben ist, das arithmetische Mittel der Werte. Je nach untersuchter Szeneneigenschaft und Anwendungsfall können aber auch andere Kombinationen sinnvoll sein: beispielsweise sollte bei der Eigenschaft der Menge der sichtbaren

Objekte (siehe Definition 5) eher das komponentenweise Maximum genutzt werden, um möglichst alle von einer beliebigen Position in der Region sichtbaren Objekte einzuschließen; oder das komponentenweise Minimum, um nur von überall aus sichtbare Objekte zu erhalten.

Punkte drei und vier werden maßgeblich durch die konkrete Ausprägung des Sampling-Verfahrens bestimmt. Das Ziel für die Konstruktion eines guten Sampling-Verfahrens ist dabei: *Für eine gegebene Anzahl an Samplen, wähle diejenigen Sampling-Positionen und eine Aufteilung, so dass die resultierende Qualität der Näherung maximiert wird.*

5.3 Regelmäßiges Sampling

Das regelmäßige Sampling stellt eine sehr einfache Form der Näherung einer Szeneneigenschaft dar. Der untersuchte Bereich der Szene wird in ein regelmäßiges zwei- oder dreidimensionales Gitter unterteilt. Die Auflösung dieses Gitters ist ein Parameter dieses Sampling-Verfahrens. Für jeden der entstehenden, quaderförmigen Teilbereiche wird am Mittelpunkt die untersuchte Szeneneigenschaft bestimmt und der Wert dem Teilbereich zugewiesen (siehe Algorithmus 5.2).

Algorithmus 5.2: Regelmäßiges Sampling mit dem zusätzlichem Parameter *Auflösung*

Input: *Szene* s , *Region* r , *positionsabhängige Szeneneigenschaft* f , *Auflösung* $res(x, y, z)$
Output: *Datenstruktur mit Näherung von* f *im Bereich* r

```
1 var bereiche ← teile r in res.x · res.y · res.z viele , gleichgroße Quader auf
2 // res.x nebeneinander , res.y übereinander und res.z hintereinander
3 foreach( bereich ∈ bereiche )
4     var pos ← wähleMittelpunktVon( bereich )
5     var bereichsWert ← f_s( position )
6     assoziiere bereich mit bereichsWert
7 return bildeHiereachischeDatenstruktur( bereiche )
```

Dieses Verfahren bietet einige Vorteile: Es ist sehr einfach, benötigt nur die Auflösung als zusätzlichen Parameter, durch den die Laufzeit und die Genauigkeit bestimmt wird, und es entstehen keine potentiellen Ungenauigkeiten durch die Zusammenfassung von Werten, da immer nur Einzelwerte betrachtet werden.

Dem gegenüber stehen zwei Nachteile: Zunächst kann es durch die regelmäßige Struktur der Sampling-Punkte zu Aliasing-Effekten kommen; insbesondere bei niedrigen Auflösungen und Szenen mit ebenfalls regelmäßig angeordneten Objekten. Der entscheidende Nachteil des Verfahrens liegt jedoch in der Laufzeit, die für die Erstellung einer globalen Näherung einer inhomogenen Szeneneigenschaft mit akzeptabler Qualität benötigt wird. Da das Verfahren die Verteilung der Eigenschaft nicht mit einbezieht, werden in Bereichen mit einem homogenen Eigenschaftswert ebenso viele Samples gezogen, wie in gleichgroßen, inhomogenen Bereichen, in denen zusätzliche Samples jedoch insgesamt einen wesentlich größeren Qualitätszuwachs erzeugt hätten. Für viele praktische Szenen, bei denen viele Szeneneigenschaften in großen Bereichen nur wenig variieren, in kleinen Bereichen jedoch sehr stark, ist das regelmäßige Sampling-Verfahren daher kaum geeignet.

5.4 Adaptives Sampling

Eine Möglichkeit, Sampling-Punkte besser zu verteilen, so dass für eine gegebene Anzahl von Punkten eine möglichst gute Näherung der Szeneneigenschaft entsteht, ist das adaptive Sampling. Die generelle Idee ist, während des Sampling-Vorgangs die bisher gesammelten Informationen als Basis für die Wahl des jeweils nächsten Sampling-Punktes zu verwenden. Durch verschiedene Heuristiken wird versucht, einen Sampling-Punkt jeweils dort zu wählen, wo ein möglichst großer Qualitätsgewinn durch den zusätzlichen Messwert erwartet wird. Zunächst erläutere ich den groben Ablauf des Algorithmus und gehe dann im Anschluss genauer auf verschiedene Aspekte des Verfahrens sowie auf die zusätzlichen Parameter ein.

5.4.1 Beschreibung des Algorithmus

Der von mir entworfene adaptive Sampling-Algorithmus (siehe Algorithmus 5.3) arbeitet durch eine hierarchische Aufteilung des untersuchten Raumes der Szene. Beginnend mit dem gesamten zu untersuchenden Bereich, wählt der Algorithmus iterativ die Region aus, die, nach einer Qualitätsheuristik, den niedrigsten Qualitätswert aufweist. Eine niedriger Qualitätswert gibt an, dass innerhalb eines großen Raumes eine hohe Fluktuation der bisher gemessenen Werte vorliegt – es sich also um einen Bereich handelt, bei dem große Unterschiede in der untersuchten Szeneneigenschaft vorliegen. Eine solche Region lässt sich als Teil der globalen Näherung der Szeneneigenschaft nur unzureichend durch einen einzelnen Wert repräsentieren. Daher wird davon ausgegangen, dass weitere Samples in dieser Region zu einer insgesamt höheren Qualität der Näherung führen. Zunächst wird dafür die Region in mehrere kleinere Regionen unterteilt (meist gleichmäßig in acht Teilregionen im 3-D-Fall oder in vier beim 2-D-Fall). Aus jeder dieser neuen Regionen werden dann zusätzliche Sampling-Punkte gewählt, bevor für jede der neuen Regionen wieder ein Qualitätswert berechnet wird. Daraufhin beginnt das Verfahren wieder von Neuem mit der nächsten Region, die jetzt den niedrigsten Qualitätswert besitzt. Der Sampling-Prozess wird beendet, sobald eine konfigurierbare Abbruchbedingung erfüllt ist, z. B. wenn eine bestimmte Anzahl von Sampling-Punkten gezogen wurde. Zum Abschluss können dann die beim Sampling-Prozess gebildeten Regionen direkt als Basis für die globale Näherung verwendet werden. Die globale Näherung besteht dann im 3-D-Fall entsprechend aus quaderförmigen Regionen; bei bei der Untersuchung einer 2-D-Region entsprechend aus rechteckigen Regionen. Alternativ kann basierend auf den einzelnen Sampling-Punkten eine *Delaunay-Triangulierung* [Del34] des untersuchten Bereichs der Szene erzeugt und deren Teilregionen als Basis für die globale Näherung verwendet werden. Die resultierende Näherung besteht dann aus einer entsprechenden Menge an Tetraedern oder Dreiecken.

Algorithmus 5.3: Adaptiver Samplingalgorithmus

Input: *Szene* s , *Region* r , *positionsabhängige Szeneneigenschaft* f ,
Abbruchbedingung abbruch (...) $\rightarrow \{true, false\}$,
Regionsunterteilungsfunktion teileAuf(region)$\rightarrow \{r \mid r \subseteq region\}$
Qualitätsbewertungsfunktion qualität(region, werte) $\rightarrow \mathbb{R}$,
Positionsauswahlfunktion wählePositionen(region,
positionen) $\rightarrow \{pos \mid pos \in region\}$
Output: *Datenstruktur mit Näherung von* f *im Bereich* r

```
1  var priorityQueue ← erzeuge Prioritätenwarteschlange ( ) // Qualität → Region
2  var werte ← { } // globale Sammlung aller (Position , Wert)−Paare
3  var positionen ← { } // globale Sammlung aller Positionen
4
5  // füge initial die gesamte untersuchte Region hinzu
6  füge ein ( priorityQueue ⇐ ( r, 0.0 ) )
7
8  while( !abbruch() )
9      var region ← extrahiere Region mit geringster Qualität( priorityQueue )
10     var neueRegionen ← teileAuf( region )
11     foreach( neueRegion ∈ neueRegionen )
12         var neuePositionen ← wählePositionen( neueRegion , positionen )
13         positionen ← positionen ∪ neuePositionen
14         foreach( position ∈ neuePositionen )
15             werte ← werte ∪ {(position, f_s(position))}
16         var bereichsWerte ← sammle alle Werte aus werte , die in neueRegion liegen
17         füge ein ( priorityQueue ⇐ ( region , qualität( region , bereichsWerte ) ) )
18
19 | var regionen ← extrahiere alle Regionen aus priorityQueue
20 alternativ
21 | var regionen ← bilde Delaunay−Triangulierung aus Punkten in positionen
22
23 foreach( region ∈ regionen )
24     var regionsWert ← kombiniereWerte( w | (pos, w) ∈ werte ∧ pos ∈ region )
25         assoziiere region mit regionsWert
26
27 return bildeHiereachischeDatenstruktur( regionen )
```

5.4.2 Beschreibung der weiteren Parameter

Der vorgestellte Algorithmus zum adaptiven Sampeln von positionsabhängigen Szeneneigenschaften gewinnt seine Flexibilität durch die Menge an zusätzlichen Parametern, die das konkrete Verhalten des Algorithmus maßgeblich bestimmen. Eine größere Menge von Parameter birgt jedoch auch die Herausforderung, die in Bezug auf die für ein Sampling-Verfahren definierten Anforderungen (siehe Abschnitt 5.1) noch zu erfüllen. Unter Berücksichtigung der

Anforderungen stelle ich im Folgenden die einzelnen Parameter vor.

Parameter: Abbruchbedingung

Ein großer Vorteil des vorgestellten adaptiven Sampling-Verfahrens ist, dass nach jedem Iterationsschritt eine gültige globale Näherung der untersuchten Eigenschaft erzeugt werden kann. Zusätzliche Iterationen führen nur zu einer insgesamt höheren Qualität der Näherung. Diese Verbesserung ist dabei zwar nicht zwingend strikt – in einzelnen Schritten kann es je nach Wahl der Samples auch zu einer kurzfristigen Verschlechterung führen – praktisch konvergiert die Näherung jedoch mit zunehmender Iterationszahl gegen die tatsächliche Verteilung der Szeneneigenschaft. In Bezug auf die Anforderungen zeigt sich, dass sich das Verfahren daher robust in Bezug auf die Laufzeit verhält. Die Möglichkeit, den Prozess regelmäßig unterbrechen zu können, erlaubt eine freie Wahl der Abbruchbedingung, auch in Bezug auf die angepeilte Anwendung. Mögliche Abbruchbedingungen sind:

- *Anzahl untersuchter Sampling-Punkte (Standardmethode):* Die Angabe der Anzahl der zu ziehenden Sampling-Punkte, nach der abgebrochen werden soll, erlaubt einen direkten Qualitätsvergleich unterschiedlicher Sampling-Verfahren und Parameterwerte für die Gegenüberstellung von Anzahl der Punkte zu erzielter Qualität. Damit eignet sich diese Bedingung besonders für die experimentelle Untersuchung in Kapitel 5.7. Des Weiteren kann durch diese Bedingung die Laufzeit des Sampling-Prozesses relativ genau gesteuert werden, da sich diese praktisch linear zur Anzahl der untersuchten Samples verhält. Die übrigen Schritte haben einen vernachlässigbaren Einfluss auf die Laufzeit.

- *Zeit:* Lässt sich für eine Anwendung sagen, wie viel Zeit genau für das Sampling verwendet werden soll, kann nach einer bestimmten Zeit das Sampling einfach beendet werden.

- *Benutzerinteraktion* Ist das Ziel eine visuelle Analyse des Verhaltens eines Algorithmus (siehe Abschnitt 5.6) und werden Zwischenschritte des Samplings kontinuierlich visualisiert, dann kann ein Benutzer beim Erreichen des gewünschten Ergebnisses den Sampling-Prozess beenden.

- *Erreichen eines bestimmten Qualitätswertes:* Prinzipiell lassen sich auch die während des Samplings anfallenden Kennzahlen für eine Abbruchbedingung verwenden. Beispielsweise wird der Prozess abgebrochen, wenn keine Region mehr mit einem Qualitätswert unterhalb eines bestimmten Grenzwertes mehr vorhanden ist. Praktisch hat sich jedoch gezeigt, dass sich daraus ein nicht robustes Verhalten ergibt. Bereits geringfügige Änderungen an anderen Parametern können zu deutlichen Laufzeitunterschieden und damit auch Qualitätsunterschieden führen.

Parameter: Regionsunterteilungsfunktion

Der Parameter *Regionsunterteilungsfunktion* beschreibt, wie die Region mit dem jeweils aktuell niedrigsten Qualitätswert in weitere Regionen unterteilt wird. Das Ziel dabei ist, dass durch die

getrennte Betrachtung der Teilregionen eine bessere Näherung erzeugt wird, als bei Betrachtung der Ursprungsregion. Daher ist es wünschenswert, eine vorher inhomogene Region möglichst in mehrere homogenere Regionen aufzuteilen, welche zwar konvex, aber nicht zwangsläufig rechteckig sein müssen. Eine weiteres Ziel bei der Erzeugung der Regionen ist es, möglichst große Volumen (oder Flächen bei der Untersuchung einer 2-D-Region) im Verhältnis zu den Oberflächen (bzw. Umfängen) der entstehenden Regionen zu erzeugen. Vor allem bei sehr langgezogenen Regionen kann keine lokal gleichmäßige Verteilung der Samples mehr erreicht werden.

Um möglichst dem Anspruch der Einfachheit zu genügen, beschränke ich mich bei der von mir gewählten Unterteilungsfunktion auf den zweiten Aspekt: möglichst großes Volumen. Die folgende Beschreibung bezieht sich auf den 3-D-Fall, funktioniert aber analog für den 2-D-Fall: Ausgehend von einer quaderförmigen Ursprungsregion wird zunächst geprüft, ob das Verhältnis der längsten Seite zur kürzesten Seite größer als $\sqrt{2}$ ist. Ist dies der Fall, wird die Region entlang der längsten Seite gleichmäßig in zwei Regionen mit nun besserem Seitenverhältnis aufgeteilt. Andernfalls wird dir Region entlang aller Achsen gleichmäßig in insgesamt acht (bzw. vier im 2-D-Fall) Teilregionen aufgespalten. Um die Robustheit des Verfahrens zu erhöhen, gibt es die zusätzliche Rahmenbedingung, dass bei Erreichen einer minimalen Seitenlänge keine weiteren Unterteilungen mehr durchgeführt werden. Diese minimale Länge sollte so gewählt werden, dass Regionen dieser Größe für mögliche Anwendungen keine wesentliche Rolle mehr spielen. Geht man beispielsweise von einem normalgroßen, humanoiden Betrachter aus, der sich in Schrittgeschwindigkeit durch die Szene bewegt, ist eine minimale Regionsgröße von einem Kubikmeter angemessen. Diese zusätzliche Beschränkung sorgt dafür, dass ein einzelner, kleiner Bereich trotz sehr großer Sprünge in den Werten der untersuchten Eigenschaft nicht zu viele Samples auf sich konzentrieren kann.

Parameter: Qualitätsbewertungsfunktion

Die Qualitätsbewertungsfunktion definiert für eine gegebene Region aus den bisher bestimmten Werten einen Qualitätswert, der die Reihenfolge bei der Abarbeitung der Regionen bestimmt. Ziel bei der Berechnung ist es, eine Sortierung der Regionen zu erreichen, bei der die Regionen vorne stehen, die insgesamt zur größten Verbesserung der globalen Näherung beitragen, wenn sie weiter untersucht werden. Der absolute Wert, den die Funktion zurückgibt, spielt für den Sampling-Algorithmus keine Rolle. (Außer, dass, wenn der Wertebereich normiert ist, während des Sampling-Vorgangs die aktuellen Regionen gemäß des Qualitätswertes eingefärbt werden können, was dem Benutzer eine anschauliche visuelle Rückmeldung über den Status des Prozesses liefert.) Die von mir als Grundlage für mögliche Bewertungsfunktionen verwendeten Heuristiken enthalten folgende Komponenten:

- *Verteilung der bisher in der Region gemessen Werte:* Die Grundlage hierfür ist die Überlegung, dass eine große Fluktuation der untersuchten Werte auch auf einen inhomogenen Bereich der untersuchten Eigenschaft hindeutet, welcher sich schlecht durch eine einzelne homogene Region annähern lässt. Je größer also die Fluktuation der Werte einer Region, desto niedriger sollte ihr Qualitätswert sein. Eine wichtiger Teil der Heuristik ist das konkrete Maß der Werteverteilung, das für die Berechnung des Qualitätswertes

verwendet wird. Auch einzelne Ausreißer der Verteilung können einen wichtigen Hinweis darauf liefern, dass größere inhomogene Bereiche der Region existieren, die bisher noch nicht mi ausreichend vielen Sampling-Punkten untersucht wurden. Als Maß der Verteilung hat sich daher die Differenz aus Maximum und Minimum gegenüber anderen Maßen, wie z. B. der Varianz, als funktional herausgestellt.

- *Anzahl der bisher in der Region erhobenen Werte:* Je mehr Werte bereits aus einer Region untersucht wurden (wenn man von einer gleichmäßigen Verteilung der Werte ausgeht), desto höher ist die Sicherheit, dass inhomogene Bereiche auch tatsächlich entdeckt wurden. Regionen mit wenigen homogenen Werten sollten demnach ein niedrigeren Qualitätswert erhalten als gleichgroße Regionen mit ähnlicher Werteverteilung, aber mehr Samples.

- *Die Größe der Region:* Große Bereiche tragen insgesamt mehr zu der globalen Näherung bei als ein kleinerer Bereich mit ansonsten gleichen Eigenschaften. Ein entsprechender Fehler beim Sampling wirkt sich also stärker aus als bei einer kleinen Region. Daher sollten große Regionen kleinere Qualitätswerte erhalten als gleichartige Regionen geringerer Größe.

Nach experimenteller Untersuchung unterschiedlicher Ausprägungen von konkreten Heuristiken habe ich die folgende Qualitätsbewertungsfunktion für meine Anwendungen ausgewählt:

Anzahl der bereits in der Region gezogenen Werte,
geteilt durch das Produkt aus dem Durchmesser der Region und
der Differenz aus dem minimalen und maximalen gemessenen Eigenschaftswert.

Diese Funktion hat sich in der Praxis als sehr robust gegenüber der Wahl der untersuchten Szeneneigenschaft, sowie auch gegenüber der Größe des untersuchten Bereichs gezeigt. Funktionen mit mangelnder Robustheit machen sich insbesondere dadurch bemerkbar, dass je nach Wahl der Parameterwerte die Samples auf zu wenige Bereiche der Szene konzentriert werden, wodurch andere relevante Bereiche nicht identifiziert werden.

Parameter: Positionsauswahlfunktion

Die Positionsauswahlfunktion bestimmt für eine neue Region, an wie vielen und welchen Positionen zusätzliche Samples gezogen werden sollen – an welchen Positionen die positionsabhängige Szeneneigenschaft also untersucht werden sollen. Für die Wahl geeigneter Sampling-Punkte konnte ich im Wesentlichen zwei Anforderungen identifizieren:

1. *Flexible Anzahl von Sampling-Punkten:* Um den gesamten Sampling-Ansatz nicht in seiner Flexibilität einzuschränken, sollen sich prinzipiell beliebig viele Sampling-Punkte in einem Bereich untersuchen lassen.

2. *Gute räumliche Verteilung:* Da während des Sampling-Vorgangs eine Region zunächst als homogener Bereich aufgefasst wird, sollten die Sampling-Punkte innerhalb einer Region auch möglichst gleichmäßig verteilt werden. Insbesondere sollten keine größeren

Bereiche innerhalb der Region entstehen, in denen keine Samples liegen, während es andere gleichgroße Bereiche gibt, in denen mehrere Samples liegen.

Wie bei den anderen Parametern gibt es bei der Positionsauswahlfunktion wieder viele mögliche Ausprägungen, von denen ich im Folgenden einige näher betrachte:

Gleichmäßiges Gitter: Eine einfache Möglichkeit, eine Region gleichmäßig zu samplen, liegt darin, die Sampling-Punkte jeweils auf den Punkten eines regelmäßigen, rechteckigen Gitters zu wählen. Ein Nachteil ist, dass durch die regelmäßige Struktur Aliasing-Effekte auftreten können, wie auch schon bei dem regelmäßigen Sampling; siehe Abschnitt 5.3. Ein weiterer Nachteil besteht darin, dass die Zahl der Samples nur in festgelegten Schritten ausgewählt werden kann (quadratisch oder kubisch). Abbildung 5.1a zeigt ein Beispiel für ein 2-D-Sampling mit einem Gitter in einem Quadrat. Im unteren Teil der Abbildung wird die Verteilung der relativen Punktabstände, relativ zum Mittelpunkt des Bildes dargestellt.

Zufällig gleichverteilt: Eine weitere sehr einfache Möglichkeit, Punkte aus einer Region zu wählen, ist, diese zufällig gleichverteilt zu wählen. Der Vorteil dieser Methode gegenüber dem Gitter ist, dass hier eine beliebige Anzahl von Punkten gezogen werden kann. Der große Nachteil des Verfahrens ist, dass die Samples ungleichmäßig verteilt sind. In Abbildung 5.1b und Abbildung 5.2 erkennt man anhand des Beispiels, dass viele Punkte dicht beieinander liegen, dadurch aber größere freie Regionen entstehen.

Blue-Noise-Sampling: Durch ein etwas komplexeres Verfahren zur Bestimmung der Sampling-Positionen, lassen sich auch beliebig viele Sampling-Punkte weitestgehend gleichmäßig verteilt erzeugen. Das Ziel der gleichmäßigen Abdeckung lässt sich auch als das Ziel zum Erreichen der *Blue-Noise*-Eigenschaft beschreiben. Als Grundlage für das hier verwendete Sampling-Verfahren dient ein *Poisson-Disk-Sampling-Verfahren mittels Dart-Throwing* nach Cook [Coo86], jedoch mit Aufhebung der harten Restriktion eines vorgegebenen Radius. Diese Idee wurde von McCool und Fiume [MF92] aufgezeigt.

Die einzelnen Schritte des Sampling-Verfahrens sind: Für die Wahl des nächsten Samples in einer gegebenen Region wird zunächst eine feste Anzahl zufällig gleichverteilter Punkte aus der Region als Kandidaten gezogen. Für jeden dieser Kandidaten wird der kleinste Abstand zu allen bisher gezogenen Punkten berechnet; also auch zu denen, die außerhalb der untersuchten Region liegen. Als nächster Punkt wird derjenige mit dem größten Abstand gewählt. Die Bestimmung des nächsten Punktes lässt sich effizient durch die Verwendung eines Octrees (für den 3-D-Fall) erreichen. Die Anzahl der für einen Sample gezogenen Kandidaten bestimmt die Qualität der Verteilung. Aufgrund von Erfahrungen aus experimentellen Untersuchungen hat sich ein Wert von 200 für die Praxis als ausreichend erwiesen.

Im unteren Bild in Abbildung 5.1c kann man erkennen, dass durch das Verfahren um jeden Pixel eine kreisförmige Region frei bleibt. Dies wird auch in der Verteilung der minimalen Abstände zwischen den Sampling-Punkten deutlich (siehe Abbildung 5.2c). Im Unterschied zu Poisson-Disk-basierten Sampling-Verfahren kann die Einhaltung des Freiraums zwar nicht garantiert werden, dafür passt sich der Radius jedoch automatisch an, wenn weitere Sampling-Punkte gezogen werden.

Blue-Noise-Sampling mit Eckpunkten: Ein Problem des normalen Blue-Noise-Samplings tritt in Kombination mit dem verwendeten adaptiven Sampling-Algorithmus 5.3 auf. Die Sampling-Punkte, die auf den Randbereichen von mehreren Regionen liegen, werden für die Berechnung der Werte aller anliegenden Regionen herangezogen. Insbesondere die Eckpunkte der Regionen haben so einen Einfluss auf bis zu acht anliegende Regionen. Beim Blue-Noise-Sampling werden diese Punkte jedoch nicht betrachtet – d. h. jeder Sampling-Punkt liefert im Normalfall nur Informationen für eine einzelne Region. Durch die einfache Erweiterung lässt sich dieser Nachteil beheben: Die ersten neu gewählten Punkte einer Region sind – wenn nicht bereits untersucht – die Eckpunkte der Region. Bei weiteren Punkten wird nach dem normalen Blue-Noise-Sampling Verfahren. Ein dadurch entstehender Nachteil ist, dass die Gleichmäßigkeit der Verteilung abnimmt, da die Eckpunkte dicht an bereits untersuchten Punkten liegen können.

Dies kann man im unteren Bild in Abbildung 5.1d und an Abbildung 5.2d erkennen. Der freie Bereich um einen Sampling-Punkt wird häufiger durch andere (vorher gezogene) Sampling-Punkte verletzt. Insgesamt entstehen durch das Blue-Noise-Sampling mit Eckpunkten jedoch bei gleicher Sample-Anzahl Näherungen mit höherer Qualität als beim reinen Blue-Noise-Sampling. Es kombiniert dadurch die Vorteile des Gitter-Samplings mit denen des Blue-Noise-Samplings bei einer nur leicht geminderten Sampling-Qualität.

Die Anzahl der Samples, die für eine neue Region gezogen wird, wird durch folgende einfache Heuristik bestimmt: Ein konstanter Wert plus der mit einer weiteren Konstante gewichtete Durchmesser der Region. Werden die Konstanten zu klein gewählt, steigt das Risiko, dass wesentliche Fluktuationen in der Eigenschaftsfunktion nicht entdeckt werden. Bei zu großen Konstanten werden viele Samples für die großen Regionen aufgewendet und die kleineren Details der Funktion bleiben bei gleicher Sample-Anzahl eher unbeachtet. Für Walkthrough-Anwendungen haben sich folgende Werte als relativ robust herausgestellt: Durchmesser der Region in Metern gewichtet mit einem Faktor $0.05\frac{Samples}{Meter}$ plus 5 Samples.

5.5 Parallelisierung

Im Vergleich zum regelmäßigen Sampling erlaubt das adaptive Sampling die Erstellung globaler Näherungen ähnlicher Qualität mit deutlich weniger Sampling-Punkten und damit in wesentlich kürzerer Zeit. Um die Hürde für den Einsatz der in dieser Arbeit vorgestellten Methoden weiter zu senken, kann diese Zeit durch Parallelisierung noch einmal verkürzt werden. Wenn eine Szeneneigenschaft untersucht wird, die sich unabhängig von der verwendeten Hardware verhält, dann können die Samples auch unabhängig voneinander auf verschiedenen Rechnersystemen erhoben werden.

Das Gesamtsystem besteht aus einem Kontrollknoten und mehreren Sampling-Knoten, die durch ein Netzwerk verbunden sind. Die Szene ist auf allen Sampling-Knoten verfügbar. Der Kontrollknoten führt den Sampling-Algorithmus wie in Algorithmus 5.3 aus, jedoch mit einigen Änderungen bei der Wahl und Untersuchung der neuen Sampling-Punkte für neu erstellte Regionen (siehe Algorithmus 5.4). Nachdem die Region mit dem niedrigsten Qualitätswert in neue Regionen unterteilt wurde, werden zunächst alle zu untersuchenden Positionen der neuen Regionen gesammelt. Für die Auswertung der Szeneneigenschaft werden dann zu jedem freien

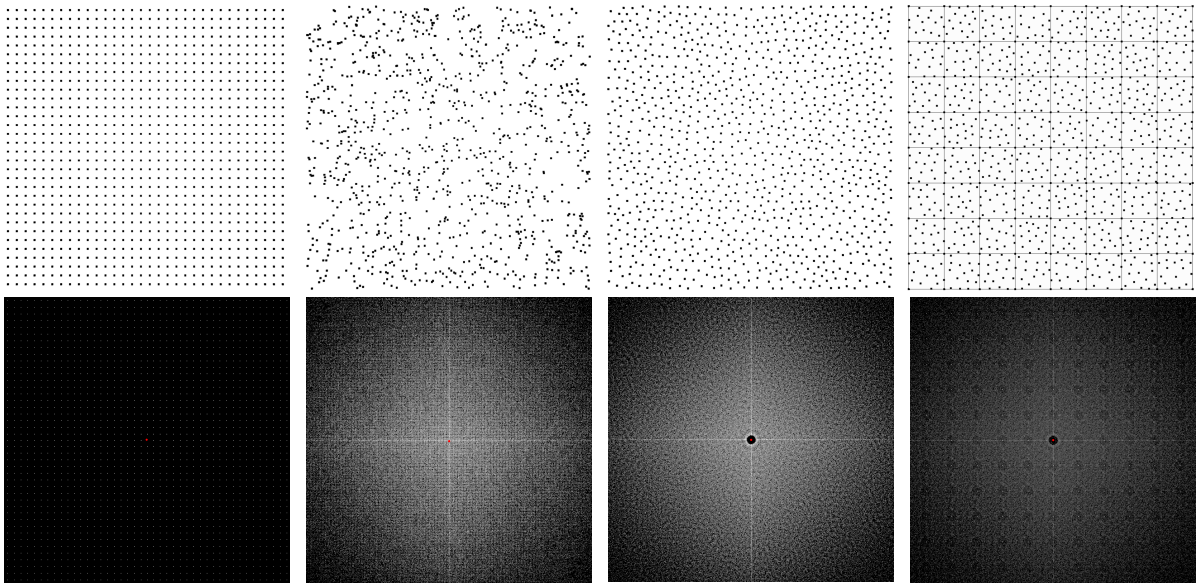


Abbildung 5.1: (oben) Durch unterschiedliche Sampling-Verfahren erzeugte Punktverteilungen in einem Quadrat mit jeweils 1024 Punkten. (unten) Die Verteilung der relativen Positionsdifferenzen für alle Punkte relativ zum Bildmittelpunkt; die Häufung entlang der Hauptachsen liegt technisch in der Diskretisierung der Sampling-Positionen begründet.

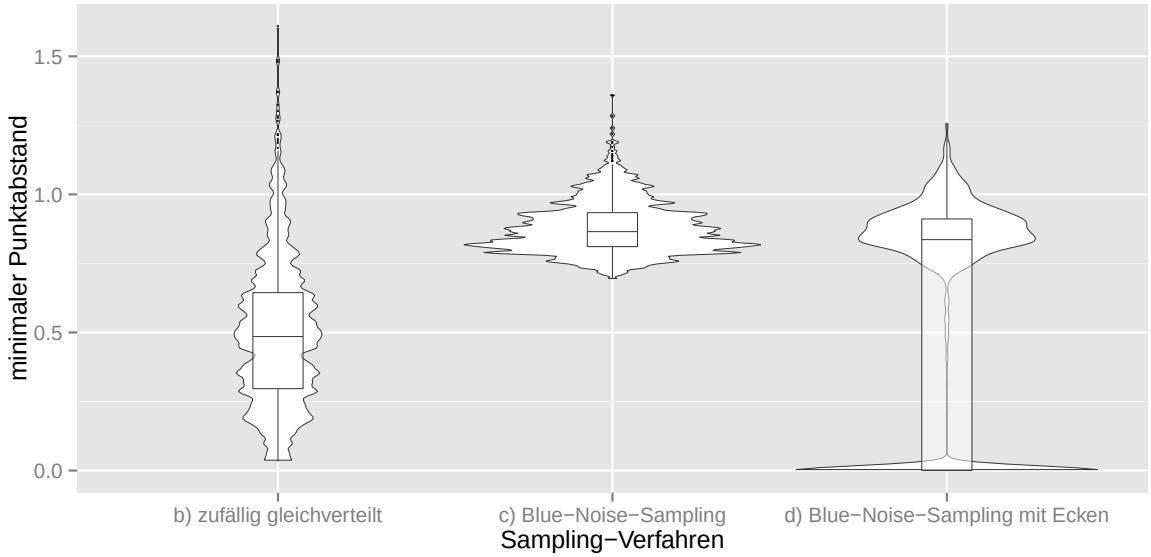


Abbildung 5.2: Verteilung minimal paarweiser Punktabstände durch unterschiedliche Sampling-Verfahren (2-D-Sampling in einem Quadrat). Alle minimalen Abstände beim Gitter-Sampling liegen bei 1.0.

Sampling-Knoten die Parameter geschickt, die zum Berechnen der Szeneneigenschaft an einer der Positionen nötig sind. Sobald ein Sampling-Knoten einen Wert ermittelt hat, wird dieser an den Kontrollknoten zurückgesandt, der den Sampling-Knoten daraufhin wieder eine neue Position zusendet, bis alle aktuell zu untersuchenden Positionen abgearbeitet sind.

Algorithmus 5.4: Paralleler adaptiver Samplingalgorithmus

```
1  ...
2  // Zeilen 1−6 aus Algorithmus 5.3
3  while( !abbruch() )
4      var region ← extrahiere Region mit geringster Qualität( priorityQueue )
5      var neueRegionen ← teileAuf( region )
6      var neuePositionen ← { }
7      foreach( neueRegion ∈ neueRegionen )
8          var p ← wählePositionen( neueRegion , positionen )
9          neuePositionen ← neuePositionen ∪ p
10         positionen ← positionen ∪ p
11     while( neuePositionen ≠ { } )
12         if( freier Samplingknoten verfügbar )
13             var position = extrahiere wert aus neuePositionen
14             schicke (f_s, position) an freien Samplingknoten
15         if( Wert von Samplinknoten verfügbar )
16             werte ← werte ∪ { Wert von Samplingknoten }
17     foreach( neueRegion ∈ neueRegionen )
18         var bereichsWerte ← sammle alle Werte aus werte die in neueRegion liegen
19         füge ein ( priorityQueue ⇐ ( region , qualität( region , bereichsWerte ) ) )
20 ...
21 // Zeilen 19−27 aus Algorithmus 5.3
```

Die Vorteile dieses Algorithmus liegen darin, dass das Originalverfahren für diese Erweiterung nur geringfügig geändert werden muss und dass durch die asynchrone Bearbeitung der Berechnungen auch heterogene Knoten mit unterschiedlich performanter Grafikhardware ohne größere Probleme verwendet werden können (bei der Untersuchung einer entsprechenden Eigenschaft). Ein Nachteil des Verfahrens besteht darin, dass es nicht beliebig mit der Anzahl der verfügbaren Knoten skaliert. Wenn mehr Rechner zur Verfügung stehen als in einem Schritt Sampels gezogen werden, können diese nicht verwendet werden. Auch können am Ende jeder Runde einige Sample-Knoten leerlaufen, bis die letzten Ergebnisse zur Verfügung stehen.

5.6 Auswertungsmöglichkeiten

Die globale Näherung von Szeneneigenschaften lässt sich auf mehrere Arten qualitativ und qualitativ auswerten. Im Folgenden gebe ich einen Überblick über verschiedene Möglichkeiten der Auswertung. Zur Demonstration verwende ich dabei eine einfache, virtuelle Szene (*Szene 1*) bestehend aus 626 Bäumen und Wandfragmenten, die jeweils ein Objekt darstellen (siehe Abbildung 5.3a). Die untersuchte Szeneneigenschaft ist die Anzahl sichtbarer Objekt (siehe Abschnitt 4.2.2).

5.6.1 Qualitative Auswertung durch Visualisierung

Das Ziel bei der Visualisierung von genäherten Szeneneigenschaften ist es, einen möglichst einfachen und intuitiven Einblick in das Verhalten der untersuchten Funktion zu erlangen. Insbesondere bei dem Entwurf von Renderingalgorithmen kann dies hilfreiche Hinweise für Verbesserungen liefern, wenn bestimmte Aspekte des Verhaltens von Algorithmen positionsabhängig visualisiert werden.

Für die Visualisierung wird zunächst eine Abbildung des Wertebereichs der Funktion auf Farbwerte (und optional auch Transparenzwerte) benötigt. Darauf basierend werden die Regionen der genährten Szeneneigenschaft eingefärbt und dargestellt (siehe Abbildung 5.3b). Wenn den Regionen nicht einzelne Werte, sondern Werteverläufe zugewiesen sind, lassen sie sich durch entsprechende Farbverläufe darstellen (siehe Abbildung 5.3d).

Die Darstellung lässt sich noch durch verschiedene Parameter konfigurieren: Bei zweidimensionalen Näherungen lassen sich zusätzlich zu den Farben auch noch Höhenwerte in die Abbildung hinzunehmen. Hierdurch können ausgewählte Bereiche des Wertebereichs zusätzlich zur Farbgebung noch deutlicher optisch hervorgehoben werden (siehe Abbildung 5.3c).

Ein Herausforderung stellt die Darstellung dreidimensionalen Näherungen als Volumendaten dar. Werden alle Regionen teiltransparent übereinander gezeichnet, lassen sich daraus Informationen nur noch schwer ablesen. Um relevante Bereiche zu erkennen, lassen sich daher zusätzlichen Filterregeln definieren, die Regionen mit einem bestimmten Wertebereich von der Darstellung ausschließen. So lässt sich das Volumen von Bereichen der Szene mit bestimmten Eigenschaften visualisieren. In Abbildung 5.3e wird beispielsweise der Bereich der Szene hervorgehoben, in dem weniger als einhundert Objekte sichtbar sind.

5.6.2 Statistische Auswertung der Verteilung

Aus der genäherten Szeneneigenschaft lassen sich direkt einzelne statistische Werte, wie Minimum, Maximum und Durchschnitt, ermitteln. Grafisch darstellen lässt sich die Verteilung der Eigenschaft beispielsweise durch ein Histogramm; oder platzsparender durch einen Violinen-Plot. Abbildung 5.3f zeigt beispielhaft die Verteilung der Anzahl der sichtbaren Objekte über die in Abbildung 5.3b gezeigten Regionen.

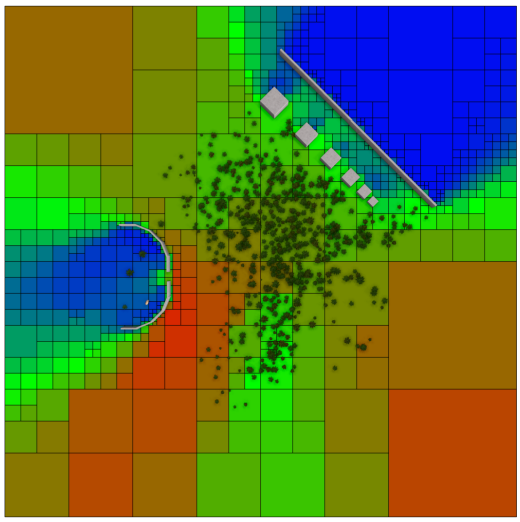
Einige Punkte sind bei der Aufbereitung der statistischen Daten zu beachten:

- Von einer alleinigen Abbildung der Verteilung durch einen Boxplot rate ich ab, da man bei der Verteilung nicht davon ausgehen kann, dass die Verteilung nur einen Häufungspunkt besitzt.

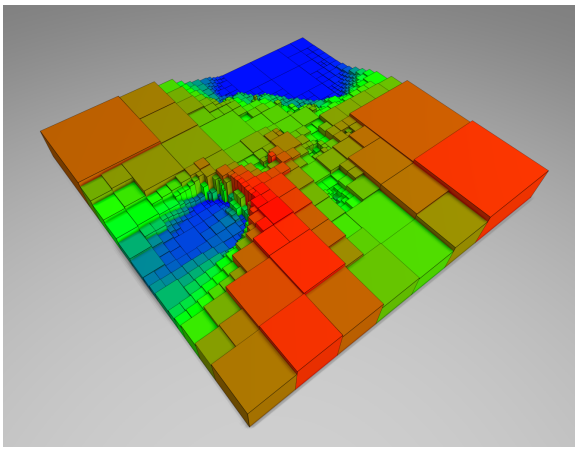
- Die Werte der Regionen müssen mit der Größe der jeweiligen Region gewichtet werden.

- Das Volumen bzw. die Fläche, in der die Szeneneigenschaft genähert wurde, muss mit beachtet und angegebenen werden, sonst kann der Informationsgehalt der Daten erheblich leiden – ähnlich wie die Verwendung von statistischen Daten, die auf einem nicht beschriebenen Kamerapfad aufgenommen wurden.

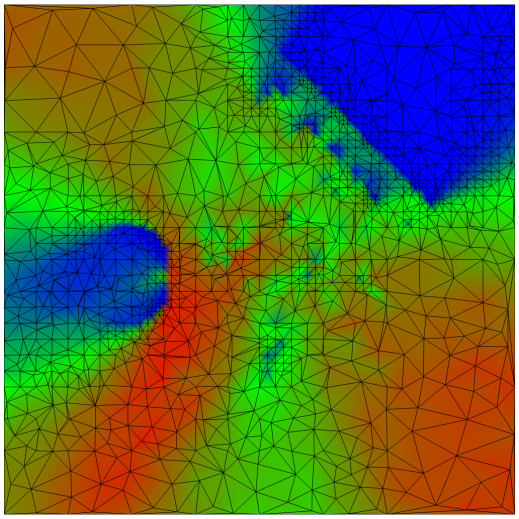
(a) Szene 1



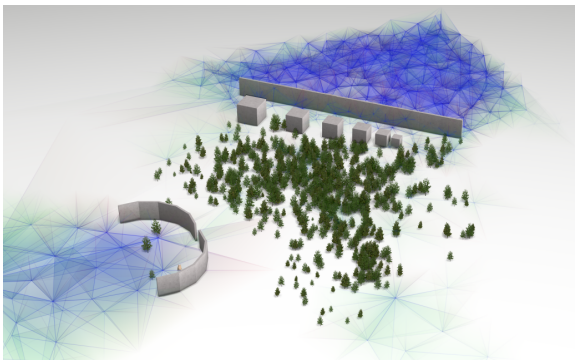
(b) Gefärbte 2-D-Regionen



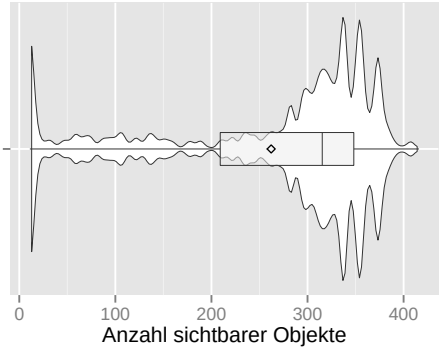
(c) 2-D-Regionen mit Höhen



(d) Gefärbte 2-D-Triangulierung der Sampling-Punkte



(e) Darstellung des Volumens mit geringer Sichtbarkeit



(f) Verteilung der Eigenschaft über die Fläche als Box- und Violinen-Plot

Abbildung 5.3: Unterschiedliche Auswertungsmöglichkeiten für die genäherte Szeneneigenschaften *Anzahl sichtbarere Objekte* in Szene *Szene 1*

5.7 Experimentelle Bewertung der Sampling-Verfahren

Das Ziel des adaptiven Sampling-Ansatzes ist die Erzeugung einer Näherung mit hoher Qualität im Verhältnis zur untersuchten Zahl an Sampling-Punkten. Im Folgenden vergleiche ich vier Sampling-Varianten miteinander: gleichmäßiges Sampling auf einem Gitter, adaptives Sampling mit zufällig gleichverteilten Positionen, adaptives Blue-Noise-Sampling und adaptives Blue-Noise-Sampling mit Ecken. Als Basis für den Vergleich habe ich die Szeneneigenschaft *Anzahl sichtbarer Objekte* (6 Richtungen, Auflösung 1024^2 Pixel) gewählt, da sich die Struktur vieler anderer Eigenschaften stark an der Objektsichtbarkeit orientiert (z. B. Renderingzeit von Culling-Algorithmen). Als Szene habe ich das Power-Plant-Modell gewählt, welches komplexe und sehr unterschiedliche Sichtbarkeitsverhältnisse bietet. Hinter dem Schornstein ist beispielsweise ein Großteil der Szene verdeckt, wobei es aber auch Positionen im Inneren und vor dem Hauptgebäude gibt, an denen bis zu 21 Prozent aller Objekte der Szene sichtbar sind (274 von 1171).

Für die Messungen werden globale Näherungen der Szeneneigenschaft für 2-D- und 3-D-Bereiche der Szene erstellt; jeweils einmal in einem Bereich, der die Szene relativ eng umschließt und in einem Bereich mit doppelter Seitenlänge (siehe Abbildung 5.4). Um die Qualität einer Näherung abzuschätzen, wird an 2000 gleichverteilt zufälligen Positionen innerhalb der untersuchten Region der tatsächliche Wert der Szeneneigenschaft gemessen und der Absolutwert der Differenz zu dem Wert der genäherten Szeneeingenschaft als Fehler gewertet. Die Laufzeit zur Erstellung der Näherungen verläuft in den untersuchten Fällen linear mit der Zeit zur Messung an einer Position. Diese beträgt mit dem für diese Arbeit eingesetzten Testsystems (siehe Abschnitt 3.4) jeweils ca. 22 ms. Abbildung 5.5 zeigt einige charakteristische Ergebnisse, bei denen die Verteilung des ermittelten Fehlers für die unterschiedlichen Messungen aufgetragen ist. Eine Häufung von kleineren Fehlerwerten entspricht einer höheren Qualität der zugrundeliegenden Näherung der Szeneneigenschaft.

Was generell an den Messungen auffällt, ist, dass es für alle Sampling-Methoden Ausreißer gibt, an denen die gemessenen Werte weit von den Werten der genäherten Szeneneigenschaft abweichen. Dies stellt weniger ein Problem für die globale Einschätzung des Verhaltens eines Algorithmus dar, sondern eher für die Verwendung der genäherten Szeneneigenschaft zur Laufzeit. Darauf gehe ich noch einmal in den Abschnitten 7.5 und 7.6 ein.

Sowohl im 2-D-Fall als auch im 3-D-Fall zeigt von den adaptiven Sampling-Varianten das Blue-Noise-Sampling mit Ecken die höchste Qualität, wobei das zufällig gleichverteilte Sampling die niedrigste Qualität aufweist. Im Folgenden wird daher nur noch das adaptive Blue-Noise-Sampling mit Ecken weiter betrachtet. Der Vergleich zwischen dem adaptiven Blue-Noise-Sampling mit Ecken und dem Referenzverfahren ist weniger eindeutig. Das gleichmäßige Sampling ist im Vorteil, wenn relativ wenige Samples in einer insgesamt stark fluktuierenden Region gezogen werden: Der durchschnittliche Fehler liegt im 3-D-Fall mit 1024 Samples in der kleinen Region beim Gitter bei 12.65 und beim adaptiven Sampling bei 19.22. In allen anderen Fällen ähneln sich die Verteilungen der Fehler so sehr, dass kein klarer Vorsprung ausgemacht werden kann. Das Argument, das letztlich für das adaptive Blue-Noise-Sampling mit Ecken im Vergleich zum regelmäßigen Sampling spricht, ist die Möglichkeit,

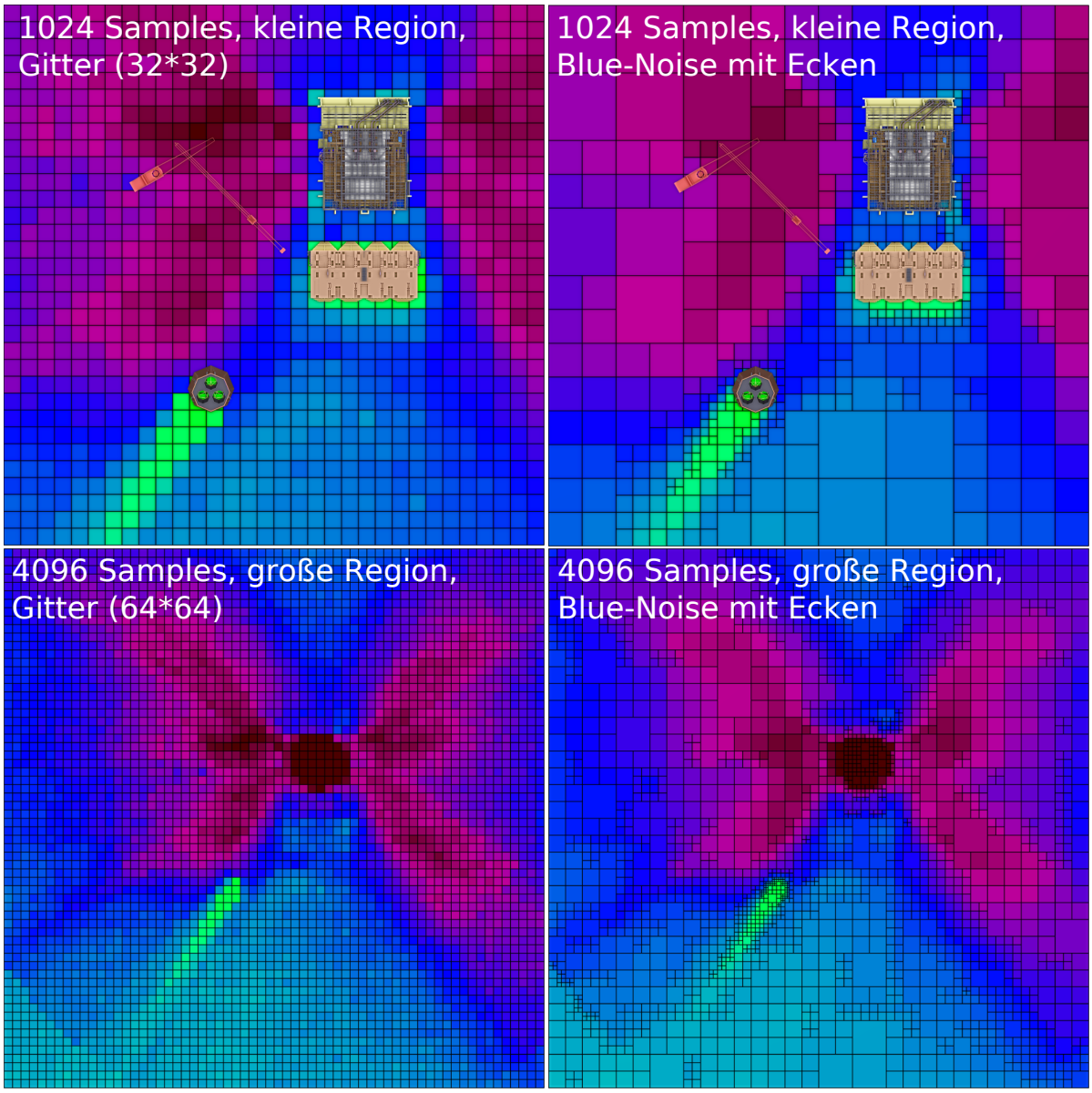


Abbildung 5.4: Visualisierung der genäherten Szeneneigenschaft *Anzahl sichtbarer Objekte* in der Power-Plant-Szene mit unterschiedlichen Sampling-Verfahren bzw. Parametern; Rot: 400 Objekte, Violett: 200 Objekte, Blau: 100 Objekte, Türkis: 50 Objekte, Grün: 20 Objekte

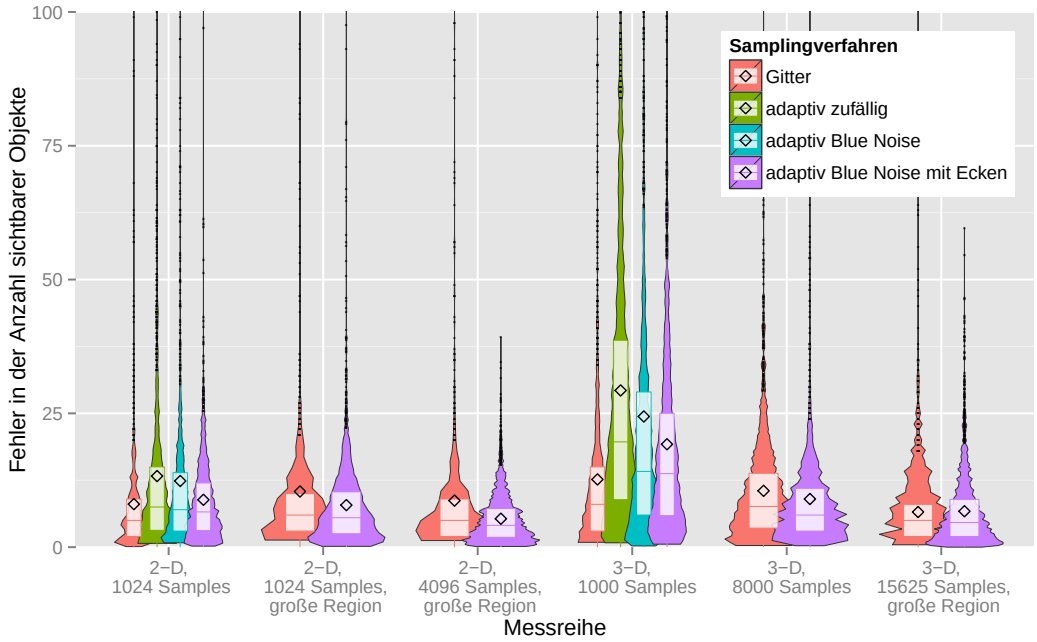


Abbildung 5.5: Abschätzung der Qualität unterschiedlicher Sampling-Verfahren anhand der Verteilung der Abweichung bei 2000 zufällig gewählten Positionen in der Power-Plant-Szene

die Sampling-Anzahl flexibel zu wählen – wobei nur bei geringer Sampling-Anzahl mit einem Qualitätsverlust zu rechnen ist. Um die Gesamtqualität besser einschätzen zu können, zeigt Abbildung 5.6 die Korrelation zwischen gemessenen und genäherten Werten für das adaptive Blue-Noise-Sampling mit Ecken in der Power-Plant-Szene.

5.7.1 Benötigte Anzahl an Samples

Wie aus den vorangegangenen Messungen deutlich wird, ist es für die Qualität von einzelnen, positionsbezogenen Anfragen an die genäherte Szeneneigenschaft notwendig, eine große Anzahl an Sampling-Punkten zu verwenden. Wenn jedoch keine einzelnen Werte benötigt werden, sondern eine statistische Auswertung der Verteilung der Szeneneigenschaft, lassen sich mit relativ wenigen Sampling-Punkten zuverlässige Aussagen treffen. Im Folgenden wird dazu wiederum die Objektsichtbarkeit in der Power-Plant-Szene in einer 3-D-Region untersucht (mit dem adaptiven Blue-Noise-Sampling mit Ecken) und die Verteilung der Szeneneigenschaft für eine unterschiedliche Anzahl von Sampling-Punkten miteinander verglichen. Bereits ab 800 Samples verändern sich trotz der komplexen Sichtbarkeitsverhältnisse die 0.25-, 0.5- und 0.75-Quantile sowie der Durchschnitt nur noch leicht (siehe Abbildung 5.7 und Tabelle 5.1). Wenn es bei einer Untersuchung um Extremwerte geht, wie beispielsweise der geschätzten höchsten Renderingzeit, gilt jedoch wie bei Positionsanfragen auch: je mehr Samples desto besser.

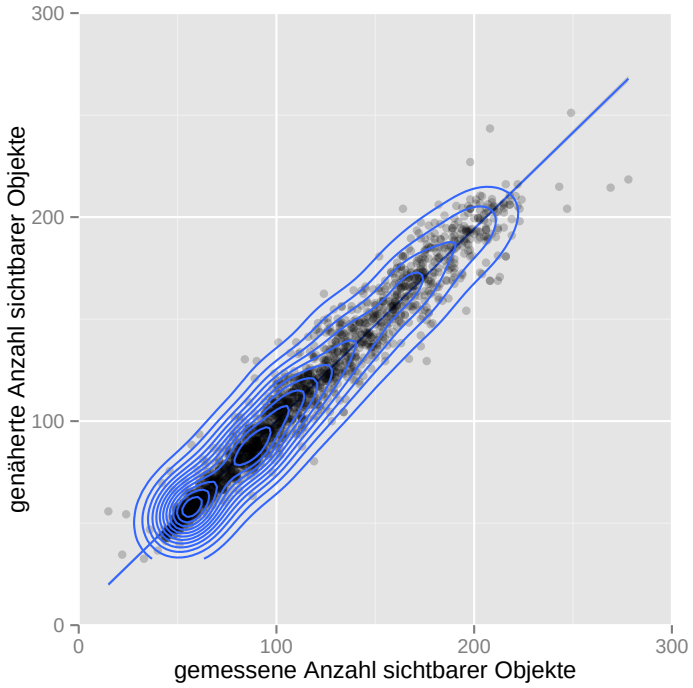


Abbildung 5.6: Korrelation zwischen 2000 gemessenen und den genäherten Werte für die *Anzahl der sichtbaren Objekte* mit dem adaptive Blue-Noise-Sampling mit Ecken, Power-Plant-Szene, 3-D-Region mit 15625 Samples

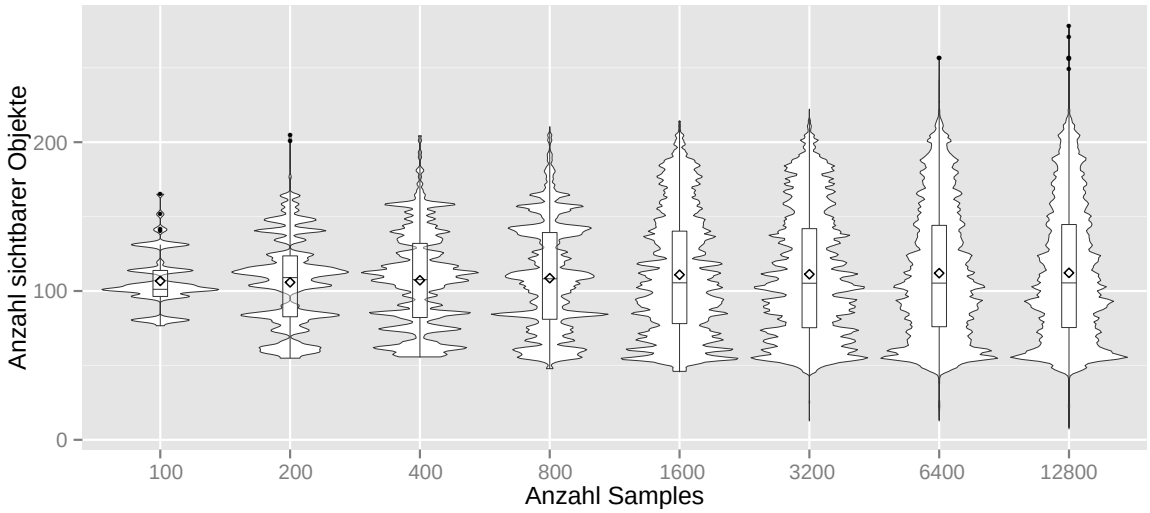


Abbildung 5.7: Verteilung der Werte in der genäherten Szeneneigenschaft *Anzahl sichtbarer Objekte* in Abhängigkeit der Anzahl der verwendeten Samples, Power-Plant-Szene, 3-D-Region

Samples	Minimum	0.25-Quantil	Median	0.75-Quantil	Maximum	Durchschnitt
100	76.3	96.0	101.1	113.8	165.4	106.8
200	54.8	83.0	109.0	123.6	204.8	105.8
400	55.2	82.2	107.8	132.0	204.8	107.2
800	47.7	81.0	108.2	139.3	210.4	108.6
1600	45.4	78.1	105.5	140.2	214.7	110.9
3200	12.4	75.3	105.2	141.9	222.4	111.3
6400	12.4	76.0	105.3	144.1	256.7	112.0
12800	7.6	75.4	105.4	144.7	278.2	112.2

Tabelle 5.1: Eckdaten der Verteilung der genäherten Szeneneigenschaft *Anzahl sichtbarer Objekte* in Abhängigkeit der Anzahl der verwendeten Samples, Power-Plant-Szene, 3-D-Region

6 Approximatives Rendering mit Progressive-Blue-Surfels

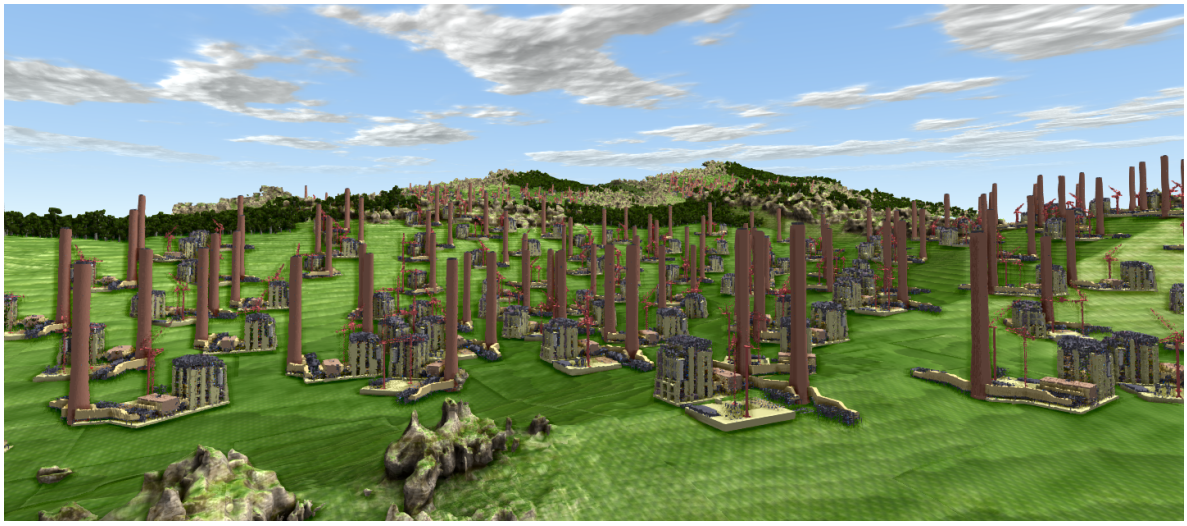


Abbildung 6.1: Beispiel für die Darstellung einer komplexen Szene durch das Progressive-Blue-Surfels-Verfahren. Im Frustum befinden sich ca. 5.8 Milliarden Dreiecke, verteilt auf ca. 480 Tausend Szenengraphknoten. Dargestellt werden ca. 14 Millionen Dreiecke und 8 Millionen Punkte mit 11.8 fps.

Das im Rahmen dieser Arbeit entwickelte *Progressive-Blue-Surfels*-Näherungsverfahren ist ein Renderingverfahren für die interaktive Darstellung hochkomplexer virtueller Szenen. Die grundlegende Idee des Verfahrens ist es, weiter entfernte Teile der Szene durch eine Punktmenge geringerer Komplexität anzunähern und damit die Menge der tatsächlich gerenderten Geometrie deutlich zu reduzieren. Was das Verfahren von anderen, punktbasierten Näherungsverfahren unterscheidet, ist im Wesentlichen die Art, in der die Punkte erzeugt und sortiert werden: Die Punkte werden auf der Oberfläche der von außen sichtbaren Geometrie verteilt und dabei in einer Folge so sortiert, dass jeder Präfix der verwendeten Punkte eine möglichst gute Näherung der Geometrie darstellt – je mehr Punkte verwendet werden, desto besser wird die Qualität der Näherung. Dadurch lassen sich zur Laufzeit die Anzahl der Punkte, die als Ersatz für einen Teil der Szene gerendert werden, sehr feinschrittig an die projizierte Größe des angenäherten Szenenteils angepasst werden. Dies erlaubt eine gute Ausnutzung der Renderingkapazität und reduziert gleichzeitig die visuellen Artefakte beim Umschalten zwischen verschiedenen Qualitätsstufen. Das Verfahren ist relativ robust in Bezug auf die geometrische Struktur der angenäherten Objekte und erlaubt so die unkomplizierte Darstellung

sehr großer Szenen mit Objekten unterschiedlicher Herkunft. Abbildung 6.1 zeigt als Beispiel eine Szene aus komplexen CAD-Modellen (Power-Plant; einfache Materialien, hohe Komplexität), modellierten Bäumen (organische Struktur, Materialien mit Texturen und Shader) und einer prozedural generierten Landschaft.

Beim Erstellen der *Surfels* (Punkte auf der Oberfläche) in der Vorverarbeitung, wird angestrebt, dass jeder Punkt einen möglichst großen Teil der Oberfläche abdeckt, wobei der durch einen Punkt abgedeckte Bereich mit der Position des Punktes in der sortierten Folge (und damit seine Relevanz) abnimmt. Anders ausgedrückt soll die kürzeste Distanz zwischen zwei Punkten maximiert werden (angestrebte Blue-Noise-Eigenschaft), wobei der erwartete minimale Abstand mit der Länge des betrachteten Präfix abnimmt. Abbildung 6.2 zeigt beispielhaft Präfixe unterschiedlicher Länge zum Power-Plant-Modell. Das zugrundeliegende Sampling-Verfahren orientiert sich dabei an dem Sampling-Verfahren, dass auch für die Verteilung der Messpunkte für die globale Approximation der Szeneneigenschaften verwendet wird (siehe Abschnitt 5.4).

In Abschnitt 6.1 beschreibe ich die Berechnung der Punkte und ihre Einbettung in den Szenengraph. In Abschnitt 6.2 beschreibe ich das Renderingverfahren unter Verwendung der Punkte. Eine Evaluierung des Verfahrens in Hinblick auf Laufzeit und Bildqualität erfolgt später im Rahmen der Anwendung von genäherten positionsabhängigen Szeneneigenschaften in Abschnitt 7.2.

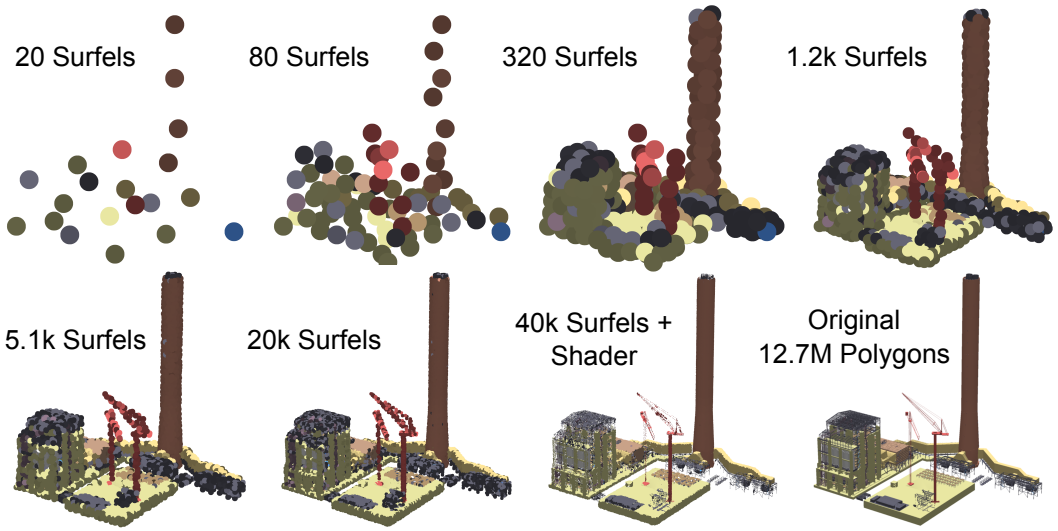


Abbildung 6.2: Unterschiedliche Präfixe aus einer Folge von Surfels für das Power-Plant-Modell. Im vorletzten Bild wird die Größe der Surfels individuell im Shader angepasst. Das letzte Bild zeigt das Original.

6.1 Vorverarbeitung: Berechnung der Surfels

Das Ziel der Vorverarbeitung des Verfahrens ist es, für komplexe Teilbäume eines Szenengraphen jeweils eine Surfel-Repräsentation zu berechnen und diese mit dem jeweiligen Teilbaumwurzelknoten zu assoziieren. Die Eingabe ist der Szenengraph in einer geeigneten,

räumlich lokalen Struktur (beispielsweise auf Basis eines Octrees) und ein Satz an Parametern (eine Zusammenfassung aller Parameter folgt in Abschnitt 6.3). Im Folgenden beschreibe ich zunächst, wie eine einzelne Surfel-Repräsentation für einen Teilbaum berechnet wird. Im Anschluss beschreibe ich dann, wie die Surfel-Repräsentationen für eine komplette Szene generiert werden.

6.1.1 Berechnung einer Surfel-Repräsentation

Das Ziel dieses Schrittes ist die Berechnung einer Surfel-Repräsentation für einen Teilbaum; d. h. eine Folge von 3-D-Punkten, von denen jeder Präfix eine Approximation der von außen sichtbaren Geometrie des Teilbaums darstellt. Ein einzelner Punkt besteht aus einer 3-D-Position, eines 3-D-Normalenvektors, einer oder mehrerer Farbwerte (unbeleuchtet) und einem Wert für die relative Größe des Punktes.

Erstellung einer initialen Menge möglicher Surfels

Der Algorithmus zur Erstellung der Surfel-Folge beginnt mit der Ermittlung einer *initialen Menge von möglichen Surfeln*. Hierzu wird der Teilbaum aus mehreren Richtungen in einer orthografischen Projektion gerendert. Dabei werden gleichzeitig in drei Texturen jeweils die 3-D-Positionen, die 3-D-Normalenvektoren, sowie die Farbe der Oberflächen der Geometrie geschrieben. Die *Auflösung* beim Rendering ist ein Parameter des Verfahrens. Außer der Beleuchtung werden für diesen Rendervorgang alle Oberflächeneigenschaften der Geometrie, wie etwa Materialeigenschaften, Texturemapping oder Normalmapping, verwendet. Die Bilder der unterschiedlichen Richtungen werden nebeneinander in dieselben Texturen gerendert (Abbildung 6.3 zeigt ein Beispiel hierfür). Als Richtungen haben sich die acht Richtungen von den Eckpunkten eines den Teilbaum umschließenden Würfels in Richtung des Würfelmittelpunktes bewährt. Weniger Richtungen, beispielsweise entsprechend der sechs Seiten eines Würfels, führen auch bei einfachen Objekten schon zu Löchern in der Surfel-Repräsentation; mehr Richtungen reduzieren zwar Fehler bei Objekten mit tiefen Einkerbungen abseits der untersuchten Richtungen, erzeugen jedoch in der Praxis meist unnötigen Mehraufwand.

Im nächsten Schritt wird für jeden belegten Pixel ein Datensatz mit entsprechender Position, Normale und Farbe erstellt. Diese Datensätze bilden die initiale Menge der möglichen Surfels. Zusätzlich wird für die Surfel-Repräsentation des Teilbaums noch das Verhältnis zwischen der Anzahl der belegten Pixel in den Texturen zur Gesamtauflösung der Texturen ermittelt (im Folgenden *relative Abdeckung* genannt). Dieser Wert wird beim Rendering als Heuristik dafür verwendet, wie sich die projizierte Fläche des Teilbaums relativ zur projizierten Fläche der Bounding-Box des Teilbaums verhält. Hierbei handelt es sich zwar nur um einen Wert für alle Richtungen, er erlaubt jedoch eine sehr einfache und automatische Einbeziehung unterschiedlicher Objektformen auch bei heterogenen Szenen; ein Würfel bedeckt einen großen Teil seiner projizierten Bounding-Box, ein Baum mit dünnen Ästen ggf. nur einen kleinen Bruchteil. Das hier als Beispiel verwendete Power-Plant-Modell hat beispielsweise eine relative Abdeckung von 0.196.

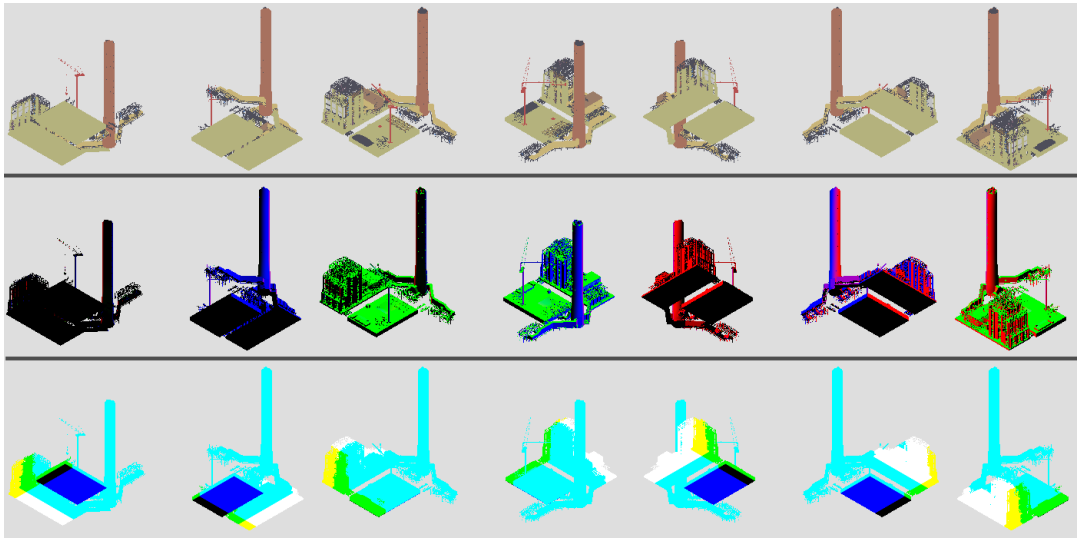


Abbildung 6.3: Beispiel für die Texturen, aus denen die Menge möglicher Surfels erzeugt wird. Oben: Farben; Mitte: Normalen in Weltkoordinaten; Unten: Positionen in Weltkoordinaten.

Progressives Sampling

Aus der im vorherigen Schritt erzeugten initialen Surfel-Menge wird im Anschluss durch einen Zufallsprozess eine Teilfolge ausgewählt. Das hierbei eingesetzte Sampling-Verfahren entspricht von der grundsätzlichen Idee dem Verfahren zur Bestimmung der Positionen bei der Approximation globaler Szeneneigenschaften (siehe Abschnitt 5.4.2) – mit dem wesentlichen Unterschied, dass die zu wählenden Positionen nicht frei aus einem 3-D-Volumen gewählt werden können, sondern aus einer gegebenen Menge von Punkten ausgewählt werden müssen.

Der Prozess beginnt mit der Wahl eines beliebigen zufälligen Punktes aus der initialen Menge der möglichen Surfels. Der Punkt wird in die Folge der Surfels eingefügt und aus der Menge der möglichen Surfels entfernt. Bis die Folge der Surfels die gewünschte *Anzahl an Surfeln* (ein Parameter des Algorithmus) erreicht hat oder die Menge der möglichen Surfels leer ist, wird iterativ ein neuer Surfel gewählt, dessen Position möglichst weit von allen vorher gewählten Surfeln entfernt liegt. Um eine für die Praxis anwendbare Laufzeit des Algorithmus zu erreichen, wird nicht der insgesamt am weitesten entfernte Surfel gesucht, sondern es wird eine zufällige Teilmenge aus der Menge der möglichen Surfel ausgewählt. Hieraus wird der beste Kandidat (mit dem größten Abstand zu allen bisher gewählten Surfels) in die Folge der Surfel übernommen und aus der Menge der möglichen Surfel entfernt. Die zufällige Stichprobe wird in den ersten Runden für die Auswahl jedes Surfels neu gezogen. Um die Laufzeit weiter zu senken wird eine Heuristik angewendet: Nach jeweils 500 Runden wird die Anzahl der Runden, für die eine Stichprobe verwendet wird, um eins erhöht – bis irgendwann die Hälfte der Surfels aus einer Stichprobe in die Folge übernommen wird. Dies folgt der Überlegung, dass die Rolle eines einzelnen Surfels für die Qualität der Näherung mit der Länge des Präfixes sinkt, so dass die Gesamtqualität der erzeugten Folge nicht zu stark beeinflusst wird. Als *Startgröße für die Stichprobe* haben sich in der Praxis Werte in der Größenordnung von 100 bis

200 Punkten bewährt; als Standardwert wurden 160 Punkte gewählt.

Bestimmung der relativen Punktgrößen

Während des Renderings können die dargestellten Punkte eine unterschiedliche Größe annehmen. Dies wird durch ihre Ausrichtung zur Kamera bestimmt, aber auch von der individuellen *relativen Punktgröße* der Punkte. Diese dient als Maß dafür, wie groß die dargestellte Fläche eines Punktes relativ zur projizierten Fläche des Teilbaums ist, um die Kontur des angenäherten Teilbaums möglichst gut zu erhalten. Liegt ein Surfel mittig auf einer größeren Fläche (im Power-Plant beispielsweise auf dem Hauptgebäude), sollte der Wert groß sein, um einen großen Bereich abzudecken. Liegt ein Surfel auf einer filigraneren Struktur (im Power-Plant beispielsweise auf dem Kran), sollte der Wert klein sein um die Struktur auch aus der Entfernung möglichst wenig zu ändern.

Um diesen Wert zu schätzen, wird abschließend folgende Heuristik für jeden Surfel angewendet: Die relative Punktgröße eines Surfels entspricht der durchschnittlichen Summe der Cosinus-Werte der Winkel zwischen dem Normalenvektor des Surfels und den Normalenvektoren seiner nächsten Nachbarn. Als Anzahl von betrachteten Nachbarn hat sich in der Praxis 20 als guter Wert herausgestellt.

Die Vorteile der Heuristik sind, dass sie sehr einfach und schnell zu berechnen ist, jedoch vor allem, dass sie ohne zusätzliche Annahmen über die geometrische Struktur des approximierten Objektes relativ robust funktioniert. Abbildung 6.4 zeigt das Ergebnis der Heuristik durch eine farbliche Kodierung der Punktgrößen.

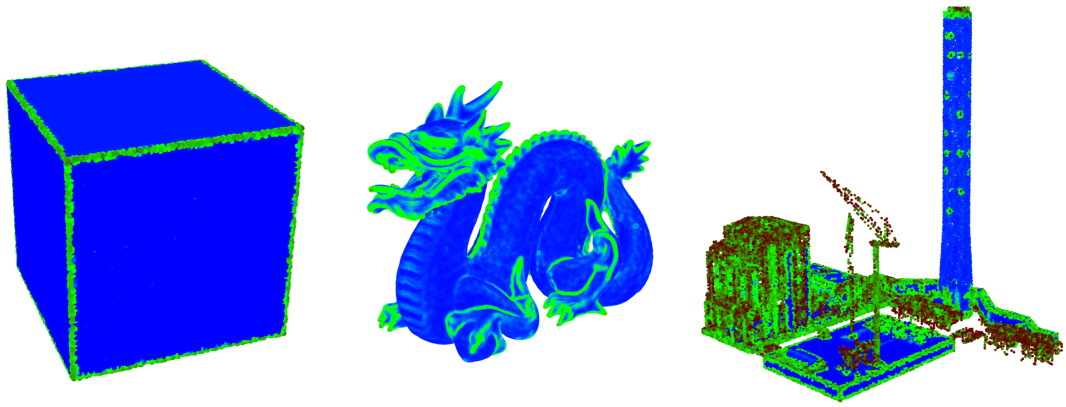


Abbildung 6.4: Visualisierung der relativen Punktgrößen für unterschiedliche Modelle. Verlauf: kleine relative Punktgröße (rot), mittlere relative Punktgröße (grün), große relative Punktgröße (blau)

6.1.2 Hierarchische Berechnung der Surfel-Repräsentationen

Das Ziel bei der Berechnung der Surfel-Repräsentationen für eine komplette Szene ist, dass möglichst nur für die Knoten im Szenengraph Surfels berechnet werden, für die durch die Verwendung der Surfels beim Rendering ein Geschwindigkeitsvorteil erreicht werden kann.

Dazu wird der Szenengraph traversiert und eine Approximation für diejenigen Knoten erzeugt, in dessen Teilbaum die Menge an Geometrie einen bestimmten Wert übersteigt. Der Wert ist ein Parameter des Verfahrens und sollte sich in der gleichen Größenordnung bewegen, wie die Anzahl der Surfels, die für einen Knoten berechnet werden.

Für instanziierte Teilbäume – d. h. für Teilbäume, die auch an anderer Position in der Szene, jedoch mit der gleichen Geometrie, vorhanden sind – brauchen Surfels nur einmal berechnet zu werden. Das eine Approximation an verschiedenen Positionen (und Ausrichtungen) benutzt werden kann, wird dadurch ermöglicht, dass die Lichtberechnung erst zur Laufzeit durchgeführt wird, und nicht bereits in die Surfels kodiert wird.

Dynamische Objekte müssen gesondert behandelt werden. Bewegt sich ein Objekt als Ganzes, ohne sich zu verformen, kann für die Darstellung auch eine Surfel-Repräsentation verwendet werden, falls für die Elternknoten bis zum Szenenwurzelknoten keine Surfels dargestellt werden. Verformbar dynamische Objekte lassen sich nicht ohne Weiteres mit Surfels darstellen.

6.1.3 Speicherplatzbedarf

Die Daten für einen einzelnen Surfel setzten sich im Standardfall aus einer 3-D-Position, einer 3-D-Normalen, eines RGB-Farbwertes und einem Wert für die relative Punktgröße zusammen. Die Position wird als drei Fließkommawerte mit je 4 Byte gespeichert, die Normale durch 4 Byte (es sind prinzipiell nur 3 Byte notwendig, aus technischen Gründen muss jedoch beim Rendering ein 4 Byte Alignment verwendet werden) und die Farbe mit der relativen Punktgröße zusammen mit weiteren 4 Byte; insgesamt also 20 Byte pro Surfel. Für eine Surfel-Näherung eines Teilbaums werden zusätzlich noch einige Metainformationen benötigt: für das Speichern in einer Datei werden in der hier verwendeten Implementierung 92 Byte Speicherplatz benötigt; während des Renderings im Hauptspeichers liegt der zusätzliche Speicherbedarf in einer ähnlichen Größenordnung. Für eine Näherung mit 40000 Surfels werden so ca. 800 kByte benötigt.

6.2 Rendering: Darstellung mit Hilfe von Surfels

Das Ziel bei der Darstellung der Szene mittels Surfels ist das Erreichen einer hohen Bildrate bei gleichzeitig guter Bildqualität. Ein weiteres Ziel ist, dass die bei vielen hierarchischen Näherungsverfahren auftretenden Artefakte beim Umschalten der Näherungen möglichst gering ausfallen (sogenannte Popping-Artefakte).

Für das Rendering eines Frames muss für jeden Knoten des Szenengraphen im Frustum entschieden werden, ob seine Originalgeometrie dargestellt wird, ob seine Surfel-Repräsentation dargestellt wird oder ob der Knoten ausgelassen wird. Für jede dargestellte Surfel-Repräsentation muss die Größe und die Anzahl der dargestellten Punkte festgelegt werden. Um dies zu erreichen werden beginnend mit dem Wurzelknoten folgende Schritte durchgeführt:

- Liegt der Knoten außerhalb des Frustums, wird er nicht weiter behandelt.

- Ist die projizierte Größe der Bounding-Box des Knotens größer als die *maximale projizierte Größe* oder enthält der Knoten keine Surfels, dann wird für einen inneren Knoten der Algorithmus für die Kinder ausgeführt oder für einen Blattknoten die enthaltene Geometrie gerendert.

- Ist die projizierte Größe der Bounding-Box des Knotens kleiner oder gleich der *minimalen projizierte Größe* und enthält der Knoten Surfels, dann wird ein Präfix der Surfels des Knoten gerendert und die Traversierung für den Teilbaum des Knotens abgebrochen. Die Länge des Präfixes wird bestimmt durch das Produkt aus der projizierten Größe der Bounding-Box des Knotens, der relativen Abdeckung des Knotens (siehe Abschnitt 6.1.1) und einem einstellbaren *Überzeichnungsfaktor*. Übersteigt der Wert die Anzahl der verfügbaren Surfel wird die gesamte Surfel-Folge dargestellt. In diesem Fall wird die *vorgegebene Punktgröße* der Sampling-Punkte zusätzlich im Verhältnis der projizierten Fläche zur Anzahl dargestellter Surfels vergrößert, um das Entstehen von Löchern in der Oberfläche zu reduzieren.

- Liegt die projizierte Größe der Bounding-Box des Knotens zwischen der *minimalen projizierten Größe* und *maximalen projizierten Größe* und enthält der Knoten Surfels, wird sowohl ein Präfix der Surfel gerendert, als auch die Kinder des Knotens weiter traversiert (bzw. die enthaltene Geometrie gerendert). Sowohl die Länge des Präfixes als auch die vorgegebene Punktgröße werden linear zwischen Null und den Werten im Falle der minimalen Größe interpoliert (minimale Werte bei maximaler projizierter Größe, größte Werte bei minimaler projizierter Größe). Durch diesen Schritt wird ein optisch weicher Übergang zwischen der Verwendung der Näherungen auf unterschiedlichen Ebenen im Baum erreicht und Popping-Artefakte bei der Navigation durch die Szene reduziert.

6.2.1 Rendern eines Surfel-Präfixes

Die Darstellung eines Surfel-Präfixes geschieht technisch durch das Rendering einer Folge von Punktprimitiven unter Verwendung eines Shaders. Die Punktprimitive sind in einem fortlaufenden Speicherbereich als Vertex-Daten abgelegt; vorzugsweise im Speicher der Grafikkarte. Der Shader zur Darstellung arbeitet wie ein Standard-Shader zur Darstellung von Punktwolken (z. B. mit Phong-Shading), wobei die Oberflächeneigenschaften in den Vertex-Daten der Punkte kodiert sind. Eine zusätzliche Berechnung besteht darin, dass für jeden Punkt individuell die Punktgröße eingestellt wird. Dazu wird die *vorgegebene Punktgröße* (vorgegeben durch das Traversierungsverfahren) gewichtet mit der relative Punktgröße (siehe Abschnitt 6.1.1) und dem Cosinus des Winkels zwischen der Oberflächennormale des Punktes und der Sichtrichtung. Dadurch werden der Kamera abgewandte Punkte ausgeblendet. Punkte, die auf einer zur Kamera steilen Oberfläche liegen oder eine kleine relative Punktgröße besitzen, werden als einzelne Pixel dargestellt. Punkte auf der Kamera im flachen Winkel zugewandten Oberflächen mit einer größeren relativen Punktgröße werden als größere Flächen dargestellt. Diese Flächen sind im Normalfall kreisförmig oder quadratisch, in Abhängigkeit von den verwendeten Renderingparametern. In Abbildung 6.5 wird die Auswirkung des Shaders auf die Bildqualität an

einem Beispiel verdeutlicht. Durch die individuelle Anpassung der Punktgrößen bleiben mehr Details erkennbar und auch die Kontur bleibt besser erhalten; insgesamt wirkt die Näherung jedoch noch immer etwas voluminöser als die Originalgeometrie.

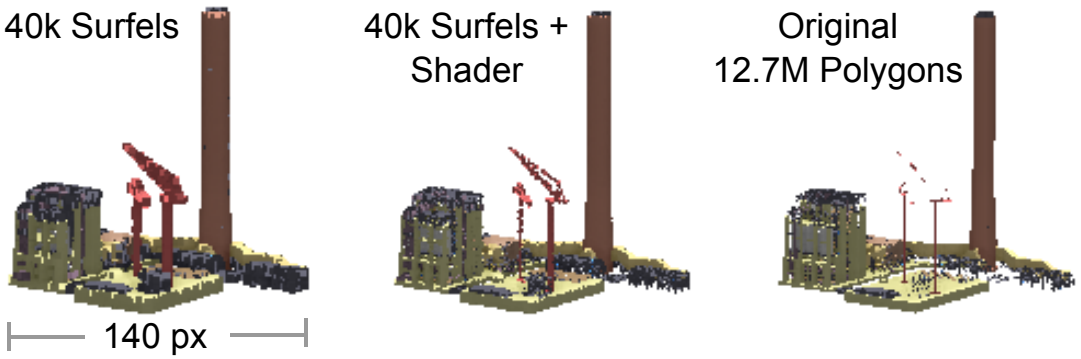


Abbildung 6.5: Auswirkungen der individuellen Punktgrößen auf die Bildqualität am Beispiel; Vergrößerung ohne Multisampling.

6.3 Überblick über die Parameter des Verfahrens

Die folgende Aufzählung fasst alle Parameter des Verfahrens zusammen und gibt Hinweise für sinnvolle Standardwerte:

Vorverarbeitung

- Die *Auflösung* beim Rendern der Texturen bestimmt die Größe der initialen Surfel-Menge. Dies hat Einfluss auf die Laufzeit der Vorverarbeitung und Qualität der gewählten Surfel-Folge. Die gewählte Auflösung sollte deutlich größer sein, als die *maximale projizierte Größe* während des Renderings.
  Standardwert: 1024^2 Pixel

- Die *Anzahl der Surfel* dient als Abbruchkriterium für den Sampling-Prozess. Der Wert beeinflusst die Laufzeit der Vorverarbeitung, den Speicherbedarf der Surfel-Repräsentationen und die maximale Größe, für die eine Surfel-Repräsentation mit guter Bildqualität verwendet werden kann.
  Standardwertebereich: 40000 bis 80000 Surfel

- Die *Startgröße der Stichprobe* bestimmt, wie groß die gezogene Stichprobe aus der initialen Surfel-Menge in der ersten Runde ist. Der Wert beeinflusst zum einen die Qualität der Sampling-Verteilung, zum anderen die Laufzeit des Samplings. Hohe Werte sorgen potentiell für eine höhere Qualität, aber auch für eine höhere Laufzeit.
  Standardwert: 160

Rendering

- Die *minimale projizierte Größe* bestimmt, ob ein Knoten entweder angenähert wird oder weiter traversiert (bzw. gerendert) wird. Bei zu kleinen Werten wird zu viel Geometrie dargestellt (hohe Renderingzeit), bei zu großen Werten sinkt die Bildqualität.
  Standardwert: 100^2 Pixel

- Die *maximale projizierte Größe* bestimmt relativ zur minimalen projizierten Größe die Länge der Übergangsphase, für die eine Surfel-Repräsentation zusätzlich dargestellt wird. Bei einem zu kleinen Wert wird der Wechsel der Darstellung zwischen zwei Ebenen im Szenengraph bei Bewegung deutlicher sichtbar, bei zu großen Werten werden überflüssige Punkte dargestellt; ggf. sinkt die Bildqualität etwas.
  Standardwert: 200^2 Pixel

- Der *Überzeichnungsfaktor* beeinflusst die Anzahl der gerenderten Punkte in Abhängigkeit der projizierten Größe und dient zum Ausgleich der Ungenauigkeit der geschätzten relativen Abdeckung. Ein zu großer Wert führt zu unnötig gerenderten Punkten (hohe Renderingzeit), ein zu kleiner Wert führt zu Löchern in der Darstellung.
  Standardwert: 4

- Die *vorgegebene Punktgröße* beeinflusst die Größe der gerenderten Punkte. Ein zu kleiner Wert führt zu Löchern in der Darstellung bei Surfel-Repräsentationen mit großer projizierter Größe. Ein zu großer Wert führt zum „Ausfransen“ der Silhouette der dargestellten Geometrie.
  Standardwert: 7

6.4 Experimentelle Bewertung der Sampling-Verteilung

Im Folgenden werden verschiedene Eigenschaften der Sampling-Verteilung untersucht. Ein wichtiges Maß ist dabei der *minimale relative Punktabstand*. Der minimale Punktabstand gibt für einen Punkt die minimale euklidische Distanz zu seinem nächsten Nachbarn an. Der minimale relative Punktabstand ist normiert mit der Länge der Diagonalen der Bounding-Box des untersuchten Objektes. Durch das Ziel der gleichmäßigen Verteilung der Punkte auf der Oberfläche ist die Verteilung der minimalen Punktabstände ein Maß für die Qualität der Verteilung.

6.4.1 Stabilität des Zufallsprozesses

Da es sich bei dem Sampling-Prozess um einen Zufallsprozess handelt, soll zunächst die Stabilität des Prozesses betrachtet werden. Dazu wurden 100-mal Surfels für das Power-Plant-Modell berechnet und die Verteilung der kleinsten, mittleren und größten, minimalen Abstände für unterschiedlich lange Präfixe berechnet (siehe Abbildung 6.6).

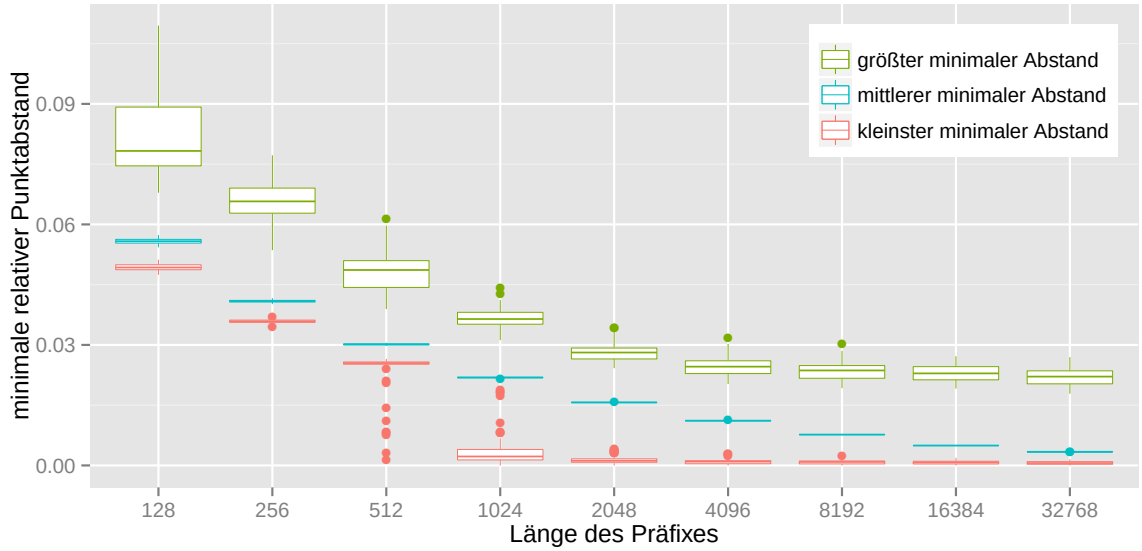


Abbildung 6.6: Verteilung der kleinsten, mittleren und größten minimalen Punktabstände in 100 Durchläufen im Verhältnis zur Präfixlänge; Power-Plant-Modell

Es zeigt sich, dass sich die mittleren minimalen Abstände sehr stabil verhalten. Bei kurzen Präfixen variieren die größten minimalen Abstände relativ stark; ab einer Länge von 1024 nähern sich die Werte jedoch deutlich an. Die kleinsten minimalen Abstände zeigen im Bereich von 512 und 1024 einige Ausreißer, stabilisieren sich jedoch bei längeren Präfixen bei Null. Aufgrund der insgesamt geringen Fluktuation gehe ich im Weiteren davon aus, dass auch einzelne Surfel-Verteilungen eines Modells als Repräsentanten des Prozesses in der Evaluierung verwendet werden können, insbesondere für längere Präfixe.

6.4.2 Einfluss des genäherten Objektes

Eine wesentliche Eigenschaft des Progressive-Blue-Surfel-Verfahrens ist seine Robustheit gegenüber der geometrischen Struktur der genäherten Objekte. Zur experimentellen Unterstützung dieser Behauptung wurden für unterschiedliche Modelle (Drache, Power-Plant und ein Würfel; siehe auch Abbildung 6.4) Surfels erstellt und die Verteilung der minimalen Abstände betrachtet (siehe Abbildung 6.7). Als Referenz wurde für das Würfelmodell eine weitere Messreihe erhoben, bei der die Punkte nicht über das Progressive-Blue-Surfels-Verfahren ermittelt wurden, sondern zufällig gleichverteilt aus der Menge der initialen Surfels.

Aus den Messungen ist zu erkennen, dass sich sich die Verteilung der minimalen Abstände für die drei untersuchten Modelle ähnlich verhält: Die minimalen Abstände der Punkte konzentrieren sich relativ stark um den Median, es gibt jedoch einige Ausreißer, an denen einzelne Surfels deutlich dichter aneinander liegen. Insgesamt nehmen die minimalen Abstände mit der Länge des Präfixes ab. Damit zeigen die Verteilungen bei allen Modellen insgesamt das für die Progressive-Blue-Surfels gewünschte Verhalten.

Es zeigt sich jedoch auch, dass für das Würfelmodell die Punkte einen größeren minimalen

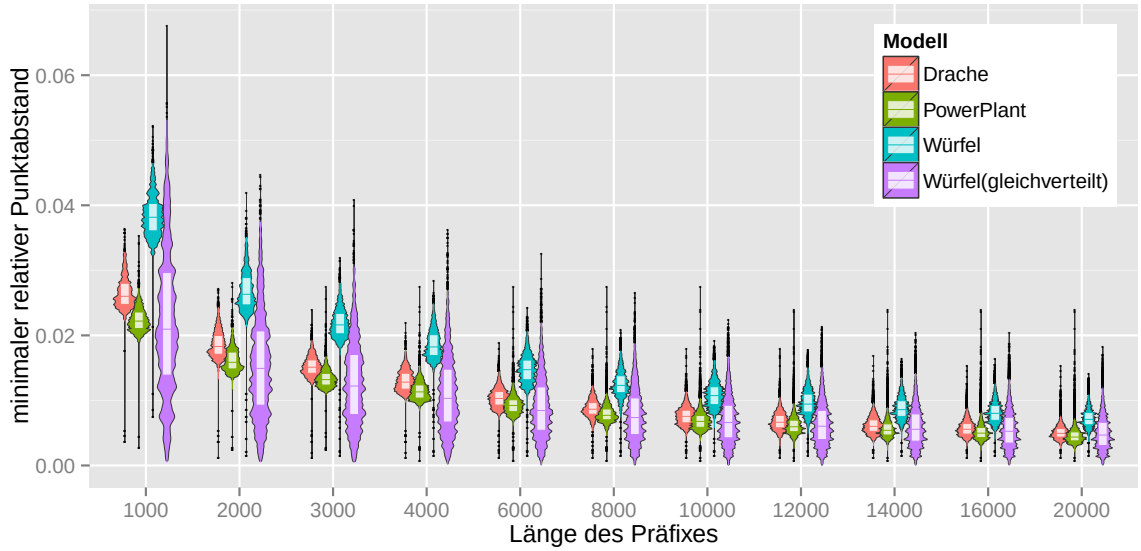


Abbildung 6.7: Verteilung der minimalen Punktabstände für unterschiedliche Modelle (Würfel, Drache und Power-Plant) im Verhältnis zur Präfixlänge; für *Würfel (gleichverteilt)* wurden die Punkte zufällig gleichverteilt gezogen

relativen Abstand aufweisen, als bei den anderen, deutlich komplexer geformten Modellen. Bei dem strukturell am komplexesten aufgebauten Modell, dem Power-Plant, sind die minimalen relativen Abstände am kleinsten. Im Vergleich zu der Verteilung der minimalen relativen Abstände mit zufällig gleichverteilt gezogenen Samples zeigt sich, dass die Qualität der Verteilung mit dem Progressive-Blue-Surfels-Sampling bei allen Modellen deutlich besser ist.

Auch wenn es leichte Unterschiede in der erzielten Qualität der Verteilung bei unterschiedlichen Modellen gibt, zeigt der Algorithmus bei allen untersuchten Modellen das gewünschte Verhalten.

6.4.3 Sampling-Qualität und Laufzeit

Um für das vorgestellten Sampling-Verfahrens eine möglichst schnelle Laufzeit bei der Vorverarbeitung zu erreichen, wird nur eine relativ kleine Stichprobengröße verwendet und die Genauigkeit des Sampling-Auswahlprozesses wird im Laufe der Runden durch die eingesetzte Heuristik weiter reduziert (siehe Abschnitt 6.1.1). Im Folgenden wird untersucht, wie sich dies auf die Qualität der Verteilung auswirkt.

In den Messungen wird das Sampling mit unterschiedlichen Startwerten für die Stichprobengröße durchgeführt und als Variante in der in jeder Runde eine neue Stichprobe ohne die Heuristik mit konstanter Größe gezogen wird. Als Referenz wird wieder das Sampling mit zufälliger gleichverteilter Wahl der Sampling-Punkte verwendet.

Die Messungen zeigen, dass alle untersuchten Varianten im Vergleich zum zufällig gleichverteilten Sampling ähnliche Verteilungen aufweisen (siehe Abbildung 6.8). Bei einer kurzen Präfixlänge bis 1024 zeigt das Sampling mit konstant 1000 Samples, gefolgt vom Sampling mit

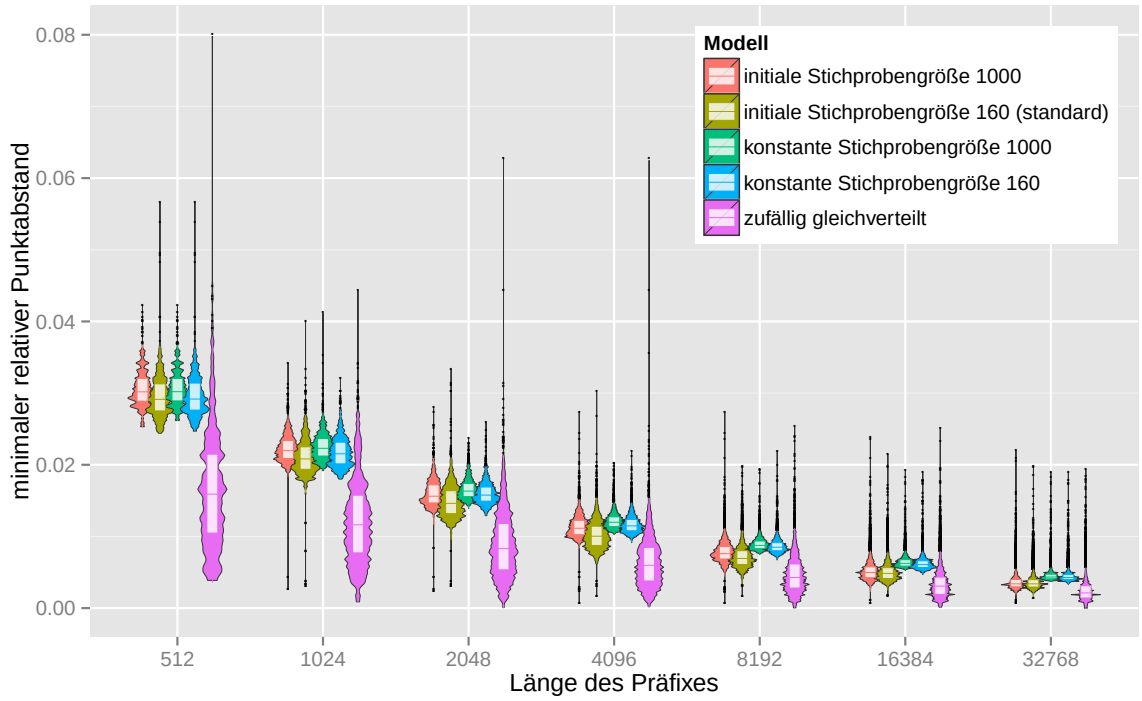


Abbildung 6.8: Verteilung der minimalen Punktabstände für unterschiedliche Sampling-Varianten im Verhältnis zur Präfixlänge; Power-Plant Modell

initial 1000 Samples, die höchste Qualität (größere minimale Abstände). Mit steigender Präfixlänge sorgt die Heuristik dafür, dass die Qualität der neu gezogenen Samples kontinuierlich leicht abnimmt, so dass ab der Messung für Präfixlänge 2048 die Qualität bei Verwendung der Heuristik unter der der konstanten Stichprobengröße liegt. Die Standardvariante mit initialer Stichprobengröße 160 liegt in der Qualität noch einmal knapp unter der Variante mit initialer Stichprobengröße von 1000. Abbildung 6.9 zeigt anhand eines Beispiels die Auswirkungen der unterschiedlichen Varianten auf die visuelle Qualität (mit fester Punktgröße in einer starken Vergrößerung).

Der Qualitätsunterschied ist vor allem zwischen der Variante mit zufällig gleichverteiltem Sampling und dem Progressive-Blue-Noise-Sampling deutlich: Es fehlen Details (z. B. am Kran), die Darstellung zeigt mehr Löcher und die Kontur ist stärker ausgefranst. Zwischen den übrigen Varianten lässt sich noch ein leichter Qualitätsunterschied in den Details zwischen der Standardvariante mit initialer Stichprobengröße von 160 und den anderen Varianten ausmachen. Der Unterschied zwischen den anderen Varianten liegt in einem ähnlichen Bereich, wie der Unterschied, der sich auch durch den Zufallsprozess bei mehreren Durchläufen mit den gleichen Parametern ergibt.

In Tabelle 6.1 sind die Laufzeiten der unterschiedlichen Schritte des Algorithmus zur Erstellung einer Surfel-Repräsentation mit 40000 Surfels in den untersuchten Varianten aufgeführt. (Die verwendete Hardware wird in Abschnitt 3.4 beschrieben.) Das zufällig gleichverteilte Sampling ist mit 25 ms mit Abstand das schnellste untersuchte Sampling-Verfahren, liefert

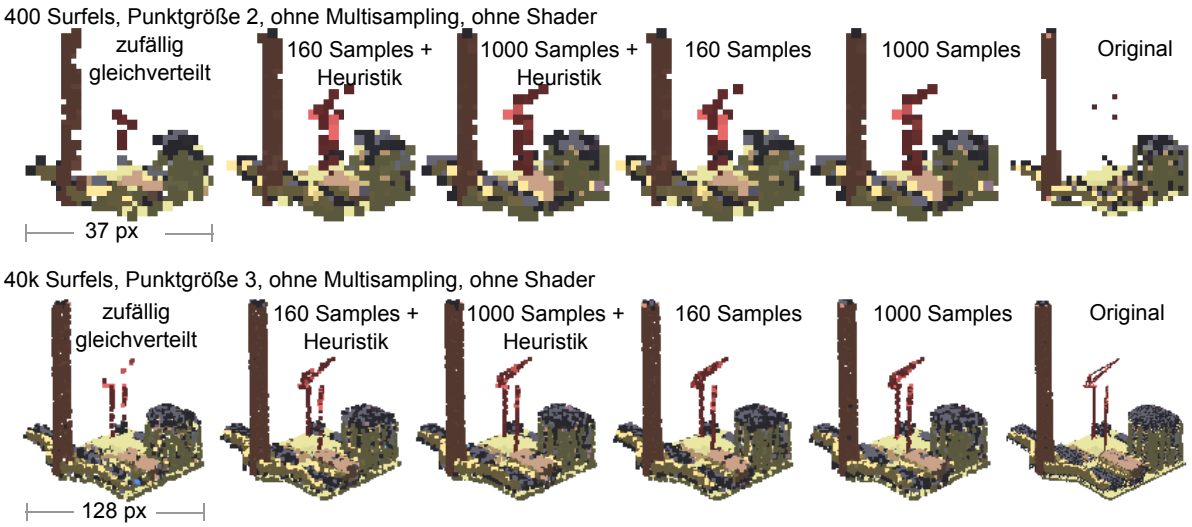


Abbildung 6.9: Vergrößerte Darstellung unterschiedlicher Sampling-Varianten

Erstellen der Texturen (8-mal Rendern der Szene)	270 ms
Transfer der Texturen in Hauptspeicher	111 ms
Erzeugung der initialen Surfel-Menge (1.3 M Einträge)	18 ms
Sampling-Variante: zufällig gleichverteilt	25 ms
Sampling-Variante: Initiale Stichprobengröße 160 (Standard)	520 ms
Sampling-Variante: Initiale Stichprobengröße 1000	1740 ms
Sampling-Variante: Konstante Stichprobengröße 160	13000 ms
Sampling-Variante: Konstante Stichprobengröße 1000	81000 ms
Bestimmen der relativen Punktgrößen	456 ms
Gesamtdauer mit Standardparametern	**1375 ms**

Tabelle 6.1: Laufzeiten zur Berechnung einer Surfel-Repräsentation des Power-Plant-Modells mit 40k Surfeln

jedoch auch eine erkennbar schlechtere Näherung. Die Standardparameter, mit einer initialen Stichprobengröße von 160, liefern sowohl in der statistischen Analyse der Verteilung, als auch beim optischen Vergleich nur minimal schlechtere Ergebnisse als die anderen Varianten, ist jedoch mit 520 ms deutlich schneller als die übrigen Varianten mit bis zu 81 Sekunden Laufzeit. Die Standardvariante eignet sich daher auch für große Szenen, für die mit überschaubaren Zeitaufwand eine große Anzahl von Näherungen mit gleichzeitig guter Qualität berechnet werden kann.

6.5 Anwendungen und Erweiterungen

Das Progressive-Blue-Surfel-Verfahren bot die Grundlage für einige weiterführende Arbeiten, die ich im Folgenden kurz zusammenfasse:

Projektgruppe: Algorithms for 3D Rendering using Cloud Computing (2012-2013) In der Projektgruppe wurde von den Teilnehmern das vorgestellte Surfel-Verfahren als Basis für ein Out-of-Core-Renderingsystem für mobile Endgeräte (Smartphones und Tablets mit Android-Betriebssystem) umgesetzt (siehe Abbildung 6.10). Bei dem System muss die virtuelle Szene nicht direkt auf den Endgeräten gespeichert werden, sondern sie kann kontinuierlich von einer Serverinstanz verteilt werden. Dabei wird ausgenutzt, dass auch schon ein kurzes Präfix das darzustellende Objekt vollständig repräsentiert. Auch wenn nur ein kleiner Teil seiner Gesamtdaten übertragen wurde, kann das Objekt bereits in reduzierter Qualität dargestellt werden. Die Länge der dargestellten Präfixe richtet sich daher nicht nur nach der projizierten Größe, sondern damit auch nach der Menge an verfügbaren Daten.

Die prinzipielle Funktion des Konzeptes konnte von der Projektgruppe gezeigt werden. Bei der Evaluierung stellten sich im Wesentlichen folgenden limitierenden Faktoren heraus:

- Die verwendete Hardware muss genügend Punktprimitive in Echtzeit darstellen können, um große Teile des Bildschirms mit Näherungen bedecken zu können. Praktisch ist die Darstellung mehrerer Millionen Punkte pro Sekunde notwendig, um eine ausreichende Bildrate und Bildqualität zu erreichen.

- Im Nahbereich wird für die Darstellung für Objekte mit einer großen projizierten Größe in akzeptabler Qualität eine große Menge an Punkten oder die Originalgeometrie benötigt. Dies stellt in dem Szenario eine große Herausforderung dar: Durch das freie Bewegen durch die Szene existieren fast immer dem Betrachter nahe Objekte, mit einer großen projizierten Größe, wobei gleichzeitig die Menge der darstellbaren Geometrie (Originale und Näherung) durch die relativ schwache Hardware und den limitierten Speicher begrenzt wird.

Auch wenn der Ansatz für aktuelle Standardgeräte noch nicht vollständig einsatzfähig scheint, bietet er bei steigender Rechenleistung und Speicherkapazität in Zukunft Potential, auch komplexere Szenen auf Mobilgeräten darzustellen, ohne dass die gesamte Szene zur Verfügung stehen muss.

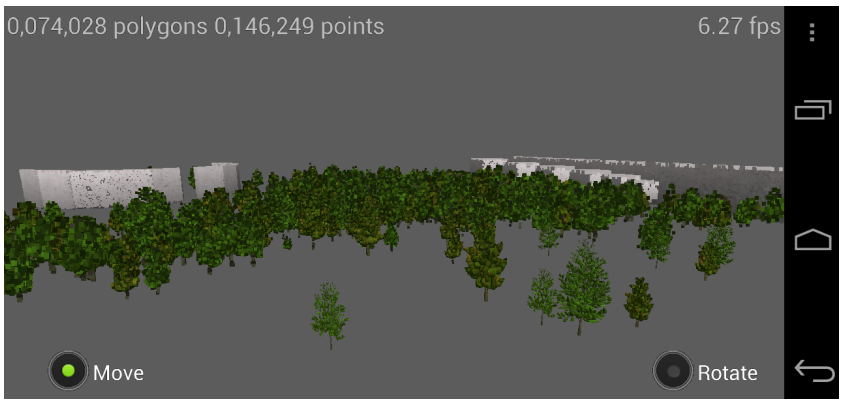


Abbildung 6.10: Punktnährungen auf einem Android Smartphone; Szene1 (Quelle: Abschlussdokumentation, Projektgruppe „Algorithms for 3D Rendering using Cloud Computing“, 2013)

Masterarbeit: Approximative Rendering using Sample-Based Occlusion Culling von Jonas Knoll (2013) In der Masterarbeit untersuchte Jonas Knoll, in wieweit das Konzept der Progressive-Blue-Surfels so abgewandelt werden kann, dass anstatt Punkten auf der Oberfläche sichtbare Polygone der Originalgeometrie in einer sortierten Folge verwendet werden können. Dazu werden für ein anzunäherndes Objekt zunächst die von außen sichtbaren Polygone bestimmt. Diese Polygone werden dann nach einem Qualitätskriterium sortiert. In das Qualitätskriterium fließen z. B. Größe und eine gleichmäßige Verteilung ein, wobei unterschiedliche Kriterien in der Arbeit untersucht werden. Wird beim Rendering ein Objekt von außen dargestellt, kann es durch einen Präfix der sortierten Polygone angenähert werden (in Abhängigkeit seiner projizierten Größe).

Der Vorteil bei der Verwendung von Polygonen anstatt von Punkten ist, dass bei geeigneten Modellen mit großen Polygonen weniger Löcher in der Darstellung auftreten, wenn die gleiche Zeit für die Darstellung einer Polygonnäherung anstatt einer Punktnäherung verwendet wird. Nachteile des Verfahrens im Vergleich zu den Progressive-Blue-Surfels sind, dass das Verfahren abhängig von der polygonalen Struktur des genäherten Modells ist (für feiner tesselierte Objekte funktioniert es weniger gut) und, dass Oberflächeneigenschaften, wie etwa unterschiedliche Texturen, nur schwer abgebildet werden können. Das in der Arbeit entwickelte Verfahren zeigt darin einige Parallelen zum randomisierten Z-Puffer [WFP+01], bei dem optional ebenfalls große Polygone direkt gezeichnet werden und nicht durch Punkte angenähert werden.

Masterarbeit: Ein Out-of-Core-Verfahren zum Rendern von großen dynamischen Szenen von Sascha Brandt (2014) In der Masterarbeit von Sascha Brandt wurde das Progressive-Blue-Surfels-Verfahren als Teil eines Out-of-Core-Renderingsystems für sehr große Szenen eingesetzt. Hierbei besteht die Möglichkeit, nicht nur die Geometrie der Szene nicht im Hauptspeicher vorzuhalten, sondern auch Teile des Szenengraphen nur bei Bedarf nachzuladen. Solange ein Teil der Szene weit entfernt ist und er durch eine Surfel-Repräsentation dargestellt werden kann, wird die Struktur des entsprechenden Teilbaums des Szenengraphen nicht benötigt. Des Weiteren wurde in der Arbeit untersucht, wie sich die notwendigen Neuberechnungen

von Surfel-Repräsentationen nach einer Änderung im Szenengraph reduzieren lassen: Wird ein Objekt in der Szene bewegt, werden im Anschluss nur für so viele Hierarchien im Szenengraph oberhalb des Objektes aktualisiert, bis im erwarteten Fall kein Surfel mehr zu dem modifizierten Objekt gehört.

Real-Time 3D Rendering of Heterogeneous Scenes In der Arbeit von Petring et al. [PEJ+13] wurde ein Verfahren vorgestellt, mit dem Bereichen einer Szene automatisch unterschiedliche Renderingalgorithmen zugewiesen werden können. Dabei wird für die aktuelle Betrachterposition die Bildqualität optimiert, wobei gleichzeitig immer eine minimale Bildrate erreicht wird. Das Verfahren bewertet automatisch die durch einen bestimmten Renderingalgorithmus erzielte Bildqualität und Laufzeit für Bereiche der Szene und kann so mit fast beliebigen Renderingalgorithmen verwendet werden. Das Progressive-Blue-Surfels-Verfahren wurde in einer leicht vereinfachten Variante als ein mögliches Näherungsverfahren in der Evaluierung verwendet.

7 Anwendungen für genäherte Szeneneigenschaften

In diesem Kapitel gebe ich einen Überblick über mögliche Anwendungen, die sich aus der Bestimmung von Szeneneigenschaften ergeben. Die Anwendungen erstrecken sich über die Phase des Algorithmenentwurfs, über die Offline-Anpassung von Parametern für ein spezifisches Szenario, bis hin zu der Verwendung der genäherten positionsabhängigen Szeneneigenschaften zur Verbesserung des Renderings zur Laufzeit. Für die Messungen wurde die in Abschnitt 3.4 beschriebenen Hardware verwendet.

7.1 Szeneneigenschaften im Bereich des Algorithm-Engineering

Unter *Algorithm-Engineering* versteht man die Entwicklung von Algorithmen, bei der neben der theoretischen Betrachtung des asymptotischen Verhaltens der entwickelten Algorithmen, auch das praktische Verhalten der Algorithmen eine wesentliche Rolle spielt. Zu diesem praktischen Verhalten gehören, neben realistischen Eingaben, auch die Randbedingungen, die durch die verwendete Hardware bestimmt werden, bis hin zu Aspekten der Implementierung des Algorithmus. Demetrescu, Finocchi und Italiano beschreiben dies in ihrem Artikel *„Algorithm Egineering“* folgendermaßen:

> „Algorithm Engineering is concerned with the design, analysis, implementation, tuning, debugging and experimental evaluation of computer programs for solving algorithmic problems. It provides methodologies and tools for developing and engineering efficient algorithmic codes and aims at integrating and reinforcing traditional theoretical approaches for the design and analysis of algorithms and data structures. “ (aus Demetrescu, Finocchi und Italiano 2003 [DFI03])

Bei der Entwicklung von Computergrafikalgorithmen spielt dieser praktisch orientierte Ansatz eine besondere Rolle. Auch wenn viele der gängigen Renderingalgorithmen beispielsweise asymptotisch lineare Laufzeit in der Komplexität der Szene aufweisen, zeigen sie für realistische Eingaben sehr unterschiedliche Bildraten. Um die Effizienz eines Algorithmus – oder zunächst einmal seiner Implementierung – während der Entwicklungsphase einzuschätzen und zu bewerten, wird dabei üblicherweise zunächst eine möglichst anwendungsnahe Testszene ausgewählt. Nun werden einige *möglichst charakteristische* Standpunkte in der Szene, oder auch ein *möglichst charakteristischer* Kamerapfad ausgewählt, der dann als Grundlage für die weitere experimentelle Evaluierung dient. Beispiele für diese Art der Untersuchung bei Culling-Algorithmen

finden sich in diversen Arbeiten ([GKM93, ZMHH97, BWPP04, MBW08, EJF10, SKJF11]). Da insbesondere bei Culling-Algorithmen die Laufzeit extrem von der Sichtbarkeit am Standpunkt des Betrachters in der Szene abhängt, sind alle Ergebnisse, die auf diesen subjektiv gewählten Positionen oder Pfaden basieren, in ihrer Aussagekraft deutlich beschränkt. Bei unzureichender Dokumentation der gewählten Positionen in einer wissenschaftlichen Publikation, können diese Messungen sogar lediglich zur beispielhaften Demonstration der Effizienz eines Algorithmus dienen; von einer systematischen Evaluierung im Sinne des Algorithm-Engineering kann dann keine Rede mehr sein. Um neben qualitativen Aussagen auch fundierte quantitative Aussagen treffen zu können (wie etwa „Der neue Algorithmus A ist bei der gegebenen Szene und Hardware meistens schneller als Algorithmus B.“), muss man die Kamerapfade sorgsam auswählen (beispielsweise anhand ausgewerteten Benutzerverhaltens; siehe [YGL97]). Zumindest muss der Pfad ausreichend gut beschrieben werden, damit er bei der Einschätzung der Aussagekraft der Ergebnisse mit einbezogen werden kann.

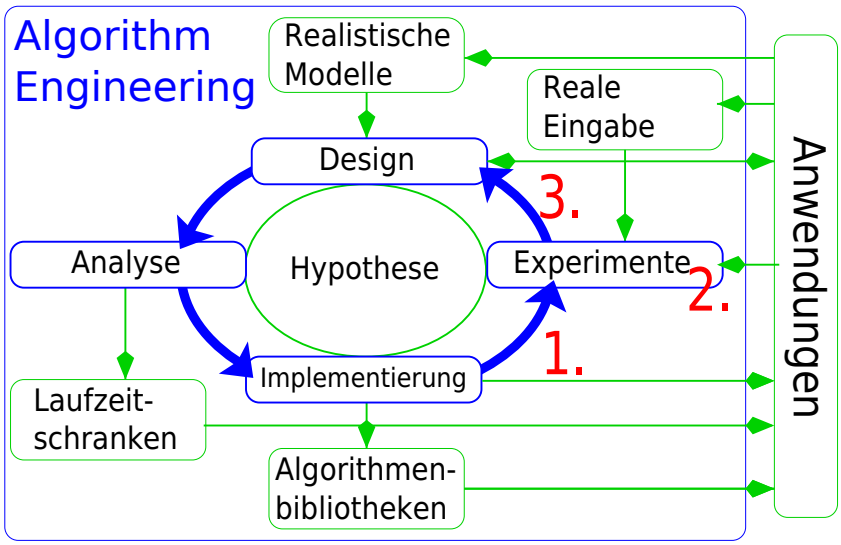


Abbildung 7.1: Überblick über den Prozess des Algorithm-Engineering mit Referenzen (in rot) auf die Einsatzgebiete der in dieser Arbeit vorgestellten Techniken (basierend auf der Übersetzung von Abbildung 1 aus „Algorithm Engineering – An Attempt at a Definition“ von Sanders [San09].)

Die in dieser Arbeit vorgestellte Näherung von positionsabhängigen Szeneneigenschaften bietet eine Möglichkeit, die gängigen Techniken mit überschaubarem Aufwand sinnvoll zu ergänzen. Abbildung 7.1 zeigt dabei verschiedene Punkte, an denen die hier vorgestellten Methoden den Prozess des Algorithm-Engineering bei der Entwicklung von Grafikalgorithmen unterstützen können:

1. Eine Hürde beim Einsatz komplexer Evaluierungsmethoden ist ihre **Implementierung**. Die hier vorgestellten Methoden wurden daher speziell mit dem Ziel entwickelt, sich ohne großen Aufwand in bestehende Evaluierungsumgebungen integrieren zu lassen. Sie stehen außerdem bereits als Teil von PADrend (siehe Kapitel 3) als Open-Source-Implementierung frei zur Verfügung.

2. Bei der **Durchführung** einer experimentellen Evaluierung bieten die Methoden zur Bestimmung einer globalen Näherung einer Szeneneigenschaft eine Möglichkeit, effizient umfassende Daten zu sammeln, welche deutlich weniger von subjektiven Entscheidungen abhängen als bei einem Kamerapfad. Dadurch erhalten die gewonnenen Daten eine wesentlich größere Aussagekraft. Der Algorithmenentwickler wird bei seiner Arbeit dadurch unterstützt, dass die vorgestellten Methoden bei einem breiten Spektrum an Szenen und Renderingalgorithmen weitestgehend automatisch angewendet werden können.

3. Bei der **Auswertung** bei den experimentellen Untersuchungen können die Messdaten anschaulich visualisiert und einfach statistisch ausgewertet werden. Dadurch lässt sich das Verhalten des Algorithmus sowie möglicherweise relevante strukturelle Eigenschaften der Eingabe einfacher nachvollziehen und die Bildung von Rückschlüssen für den weiteren Designprozess wird unterstützt.

7.2 Exploration: Auswirkungen der projizierten Größe beim Rendering mit Progressive-Blue-Surfels

Das in Kapitel 6 vorgestellte Progressive-Blue-Surfels-Verfahren benötigt während des Renderings mehrere Parameter (siehe Abschnitt 6.3). Im Folgenden soll der Effekt der projizierten Größe auf die Renderingzeit und die Bildqualität mit Hilfe der entsprechenden Szeneneigenschaften untersucht werden; d. h. im Rahmen des Algorithm-Engineering soll ein Aspekt des Verhaltens des Algorithmus anhand eines Beispiels für eine realistische Eingabe untersucht und beschrieben werden.

Als Szene für die Evaluierung wird ein Teil einer generierten Landschaft mit einer Reihe an Bäumen und einem mittig platzierten Power-Plant-Modell verwendet (siehe Abbildung 7.2). Die Szene besteht aus insgesamt 162 Millionen Polygonen in 10 264 Einzelobjekten, strukturiert in einem Loose-Octree. Für die Szene wurden für alle Teilbäume mit mehr als 2 Millionen Polygonen je 80 000 Surfels erzeugt. Alle sonstigen Parameter entsprechen den Standardwerten.

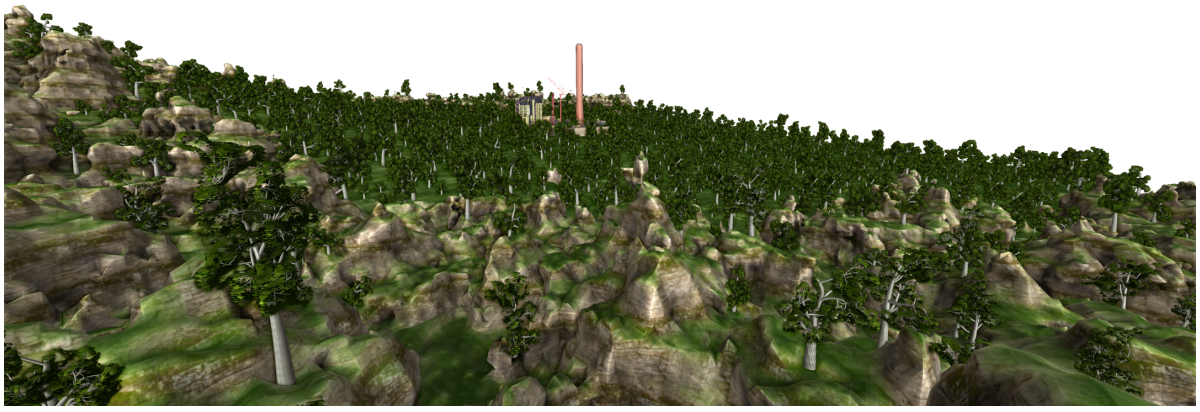


Abbildung 7.2: Testszene mit 162 Millionen Polygonen und 10 264 Objekten; dargestellt mit Progressive-Blue-Surfels; minimale und maximale projizierte Größe: [50,100]

Für die Messungen werden als Werte für die minimale und maximale projizierte Seitenlänge (entspricht der Wurzel der projizierten Größe) die Werte $[50, 100]$, $[100, 200]$ (Standardwert), $[200, 300]$ und $[300, 400]$ Pixel verwendet. Die untersuchten Szeneneigenschaften sind die *Renderingzeit mit dem Surfel-Algorithmus* und die *Bildqualität*, mit dem hierarchisch angewendeten SSIM-Verfahren [WBSS04, Bur81]. Die Szeneneigenschaften wurden je mit 1024 Sampeln für einen vertikalen Schnitt durch die Szene genähert, der sowohl den Bereich dicht über dem Boden, den Bereich direkt um das Power-Plant-Modell, als auch einen größeren freien Bereich über der Szene enthält.

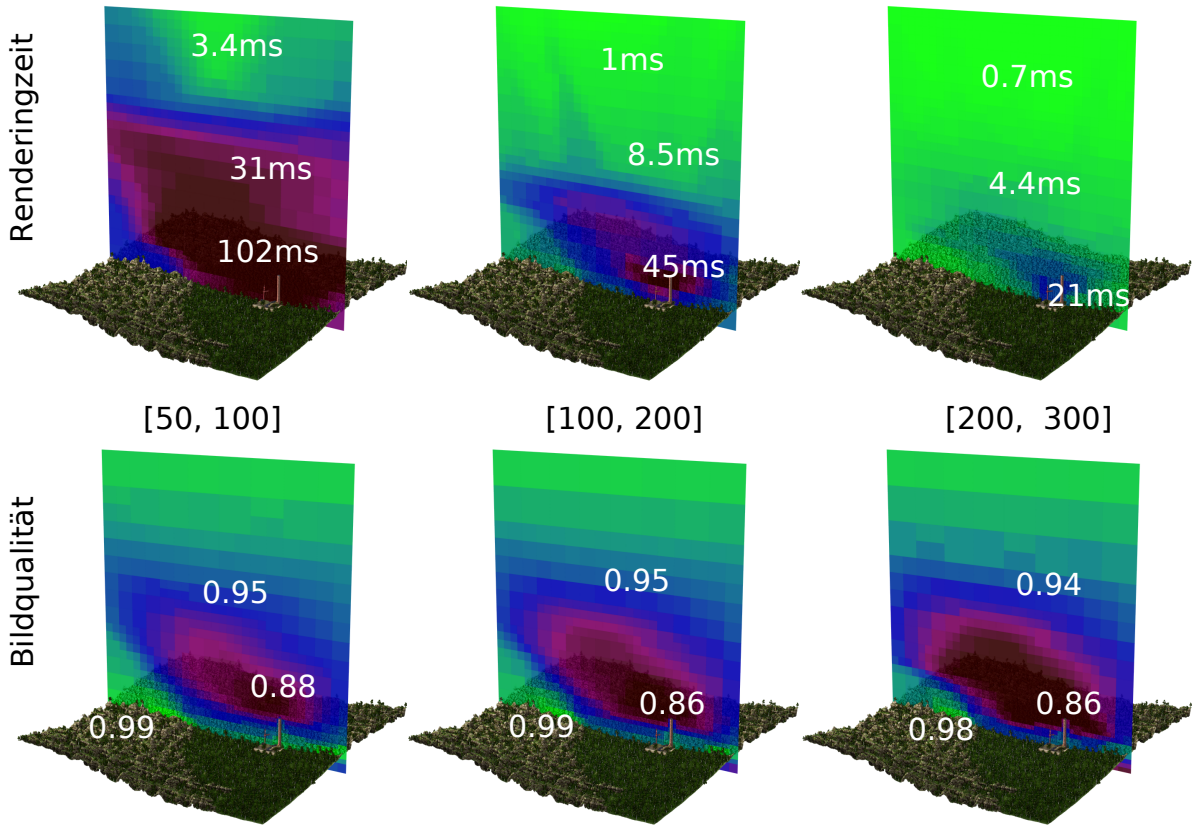


Abbildung 7.3: Visualisierung der Szeneneigenschaften Renderingzeit (oben) und Bildqualität (unten) mit unterschiedlichen Werten für minimale und maximale projizierte Größe (Mitte); Extrem- und Mittelwerte sind angegeben.

Die visuelle Auswertung in Abbildung 7.3 zeigt, dass die Laufzeit erwartungskonform insgesamt mit der Entfernung zur Szene abnimmt. Zum einen nimmt die Menge der nicht genäherten Geometrie mit steigender Entfernung ab, bis nur noch Punkte dargestellt werden. Zum anderen deckt die hier untersuchte Szene bei steigender Distanz einen immer kleiner werdenden Bereich des Bildschirms ab, so dass auch weniger Punkte dargestellt werden. Besonders hohe Renderingzeiten finden sich dicht an komplexen Bereichen der Szene (hier direkt am Power-Plant-Modell), wo viele Objekte mit großer projizierter Größe dargestellt werden müssen. Durch größere Werte beim untersuchten Parameter werden auch Objekte mit größerer projizierter Größe angenähert, wodurch die Renderingzeit abnimmt. Dies macht sich insbeson-

dere an den Positionen hoher Renderingzeit positiv bemerkbar. Schon bei Parameterwerten von $[100, 200]$ erreicht im gegebenen Rahmen die maximale Renderingzeit weniger als 45 ms – also immer mehr als 22 fps (siehe Abbildung 7.4).

Die räumliche Verteilung der Bildqualität zeigt eine etwas andere Verteilung als die Renderingzeit. Gute Bildqualität herrscht vor allem nah an der Oberfläche der Objekte der Szene und oberhalb der Szene mit größerem Abstand. Die gute Qualität nah an den Objekten liegt darin begründet, dass große Teile des Bildes durch nicht genäherte Objekte bedeckt sind und damit keinen Bildfehler erzeugen. Gleichzeitig verdecken sie jedoch auch mögliche Fehler von dahinter positionierten, genäherten Objekten. In der Visualisierung der Qualität in Abbildung 7.3 wird dies beispielsweise lokal um den Schornstein des Power-Plant-Modells deutlich. Mit größerer Distanz zur Szene steigt die gemessene Bildqualität dadurch an, dass die gesamte Szene projiziert einen immer kleineren Bereich des Bildes ausmacht, wodurch auch ein ggf. großer, jedoch räumlich begrenzter Bildunterschied insgesamt zu höheren Qualitätswerten für das gesamte Bild führt. Wie die Renderingzeit, wird auch die Bildqualität im Allgemeinen kleiner, wenn die Werte für die minimale und maximale projizierte Größe erhöht werden (siehe Abbildung 7.4). Nur an Positionen mit hoher Verdeckung (in den Schluchten am linken Rand der Szene) und weit über der Szene bleibt die Qualität annähernd gleich (siehe Abbildung 7.3). Abbildung 7.5 zeigt ein Beispiel für eine Position mit einem niedrigen Bildqualitätswert von 0.85, bei der die Szene komplett das Bild bedeckt und fast vollständig als Näherung dargestellt wird.

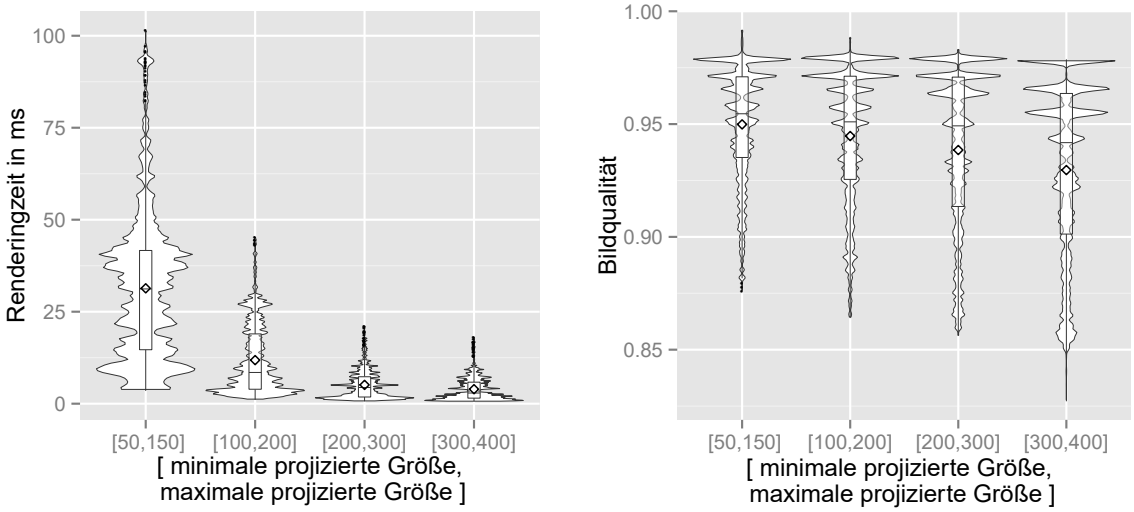


Abbildung 7.4: Verteilung der Eigenschaften Renderingzeit und Bildqualität in der Testszene für unterschiedliche Parameterwerte für die minimale und maximale projizierte Größe

Für die Wahl des Parameters muss man also zwischen Bildqualität und Renderingzeit abwägen, wobei es von der konkreten Anwendung abhängt, wie die entsprechende Anforderungen zu gewichten sind und welche Restriktionen die verwendete Hardware liefert.

Abbildung 7.5: Beispiel für eine Position mit Bildqualitätswert von 0.85 mit Ausgabe der Qualitätsbewertungsheuristik; minimale und maximale projizierte Größe: [300,400]; Der Fehler ergibt sich hauptsächlich aus der etwas voluminöseren Darstellung der Surfels, wodurch alle Silhouetten leicht gestört werden.

7.3 Parameteroptimierung: Beste Tiefe eines Octrees für minimale Renderingzeit mit CHC++

Als Beispiel für die Parameteroptimierung mit Hilfe von positionsabhängigen Szeneneigenschaften soll im Folgenden der Parameter der maximalen Tiefe eines Octrees (oder genauer eines Loose-Octrees mit Vergrößerungsfaktor 2.0) als räumliche Datenstruktur für eine Szene optimiert werden, so dass das CHC++-Occlusion-Culling-Verfahren (siehe Abschnitt 2.1.2) die kürzeste durchschnittliche Renderingzeit erreicht. Wird die Szene in einem Baum geringer Tiefe gespeichert, kann der Algorithmus weniger feingranular die Sichtbarkeit gesamter Teilbäume testen (da es nur wenige, große gibt), wodurch entweder die Sichtbarkeit der Objekte direkt getestet werden muss oder mehr Objekte durch die grobe Zusammenfassung irrtümlich als sichtbar klassifiziert werden. Ist der Baum sehr tief, können zwar feingranular auch ganze Teilbäume getestet werden, dies kann bei zu vielen inneren Knoten auch zu einem merklichen Zusatzaufwand führen. Der CHC++-Algorithmus verwendet intern einige Heuristiken, die die Anzahl der tatsächlich durchgeführten Tests schwer vorhersehbar macht. Es hängt letztlich von der Szene und der verwendeten Hardware ab, wie genau die räumliche Datenstruktur aufgebaut sein muss, um einen möglichst geringe durchschnittliche Renderingzeit zu erhalten.

Als Szene wird die Power-Plant-Szene verwendet. Die untersuchte positionsabhängige Szeneneigenschaft ist die konditionierte Renderingzeit für den CHC++-Algorithmus mit Standardparametern. Sie wird für jeden Parameterwert mit je 1024 Samples in einer quaderförmigen 3-D-Region genähert; die Region entspricht einer leicht vergrößerten Bounding-Box der Szene. Mit einer linearen Suche werden für verschiedene Werte für die maximale Tiefe des Octrees die Verteilung der Renderingzeit ermittelt, bei einem Wert von 0 liegen alle Objekte direkt unter dem Szenenwurzelknoten. Anhand der statistischen Auswertung (siehe Abbildung 7.6), ist zu erkennen, dass bei einem Octree der Tiefe 5 ein lokales Optimum der durchschnittlichen Renderingzeit mit 9.9 ms erreicht wird, wobei größere Werte keinen deutlichen Nachteil bringen. Die Messung für einen Parameterwert dauert unter diesen Rahmenbedingungen ca. 77 Sekunden. Diese Art der Bewertung von Parameterwerten kann automatisiert werden, wodurch sich auch größere Parameterräume z. B. durch eine binäre Suche nach lokalen Optima mit überschaubarem Zeitaufwand durchführen lassen.

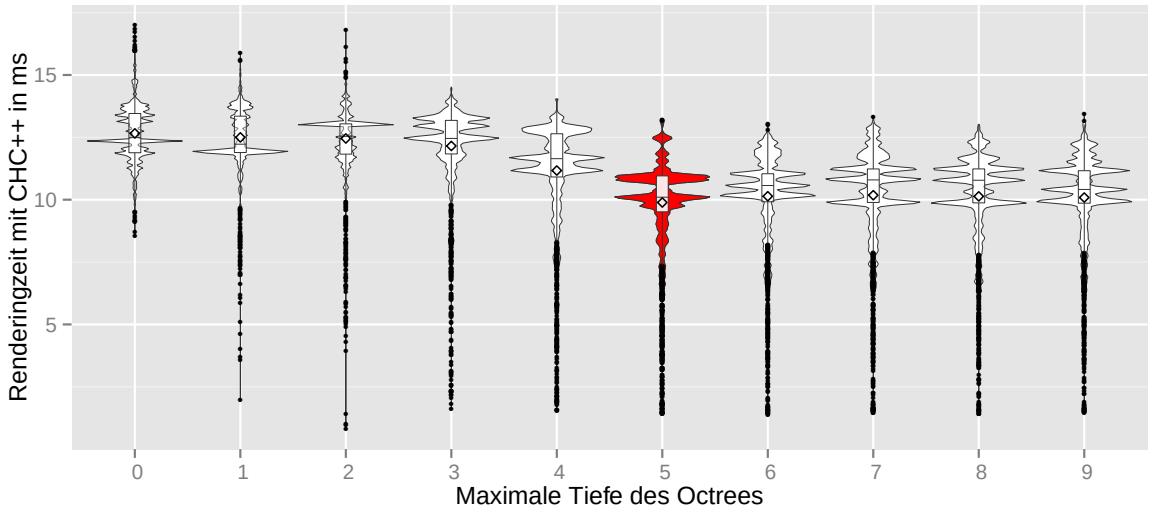


Abbildung 7.6: Verteilung der Renderingzeit mit dem CHC++-Algorithmus für unterschiedliche maximale Tiefen des Octrees; Power-Plant-Szene

7.4 Vergleich von Renderingalgorithmen: SVS gegen CHC++

Neben der Optimierung von Parameterwerten kann auch die Entscheidung, welches Renderingverfahren für ein bestimmtes Anwendungsszenario verwendet werden soll, durch genäherte Szeneneigenschaften unterstützt werden. Mehrere mögliche Verfahren können über die erzielte Renderingzeit verglichen werden, was das Ergebnis jedoch stark abhängig von der Hardware macht, die für die Messungen verwendet wird. Eine andere Möglichkeit besteht darin, weitere, hardwareunabhängige Eigenschaften zu betrachten. Als Beispiel für den Vergleich von Renderingalgorithmen anhand unterschiedlicher Kriterien wird im Folgenden ein Vergleich zwischen dem CHC++ und dem SVS-Algorithmus [EJFMadH13] durchgeführt (siehe 2.1.1): einmal bezüglich der (konditionierten) Renderingzeit und einmal bezüglich der Anzahl der gerenderten Polygone. Zusätzlich wird noch die Menge der Polygone in sichtbaren Objekten ermittelt – als untere Schranke für die zu rendernden Polygone, die jeder konservative Occlusion-Culling-Algorithmus auf Objektbasis einhalten muss.

Die verwendeten Szene besteht aus vier Power-Plant-Modellen mit insgesamt 50.8 Millionen Polygone in 4684 Objekten. Die Szeneneigenschaft wird mit 4069 Samples in einem 2-D-Schnitt horizontal durch die Szene, durch die komplexen Gebäude des Power-Plant-Modells, gemessen.

Die visuelle Darstellung der Ergebnisse in Abbildung 7.8 zeigt für den CHC++-Algorithmus, dass sich die Strukturen der Verteilung der Renderingzeit, wie auch die der gerenderten Polygone, erkennbar an der Struktur der sichtbaren Polygone orientieren. An Positionen mit hoher Sichtbarkeit werden viele Polygone dargestellt, es müssen jedoch auch gleichzeitig große Teile des Szenengraphen traversiert werden und dabei Verdeckungstests durchgeführt werden. An Positionen mit geringer Sichtbarkeit, hinter oder im Inneren der Gebäude, sinkt

die Renderingzeit deutlich, da hier mit wenigen Tests große Teile der Szene verworfen werden können. Insgesamt wird die Menge der sichtbaren Objekte vom CHC++ jedoch durchgängig überschätzt.

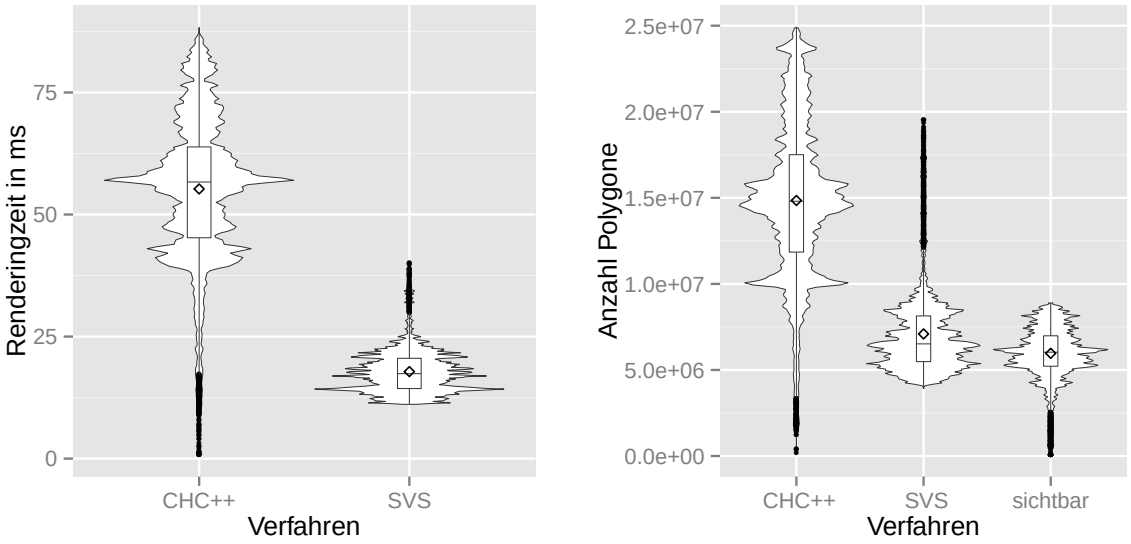


Abbildung 7.7: Verteilung der Szeneneigenschaften Renderingzeit von CHC++ und SVS (links); Verteilung der Szeneneigenschaften Anzahl gerenderter Polygone von CHC++ und SVS, Anzahl der Polygone in sichtbaren Objekten (rechts); Szene mit vier Power-Plant-Modellen

Die Visualisierung der Eigenschaften für den SVS-Algorithmus weisen die, für das Verfahren typischen, Kreise aus. Diese entsprechen Schnitten durch die Sichtbarkeitskugeln des Verfahrens. Außerhalb einer Sichtbarkeitskugel werden die sichtbaren Objekte dargestellt, die zu der Kugel aus der Richtung des Betrachters assoziiert sind. Innerhalb der Kugel wird der Szenengraph weiter traversiert. Die Verdeckung, die durch nahe Objekte erzeugt wird, wird beim SVS-Algorithmus nicht ausgenutzt. Dies wird auch dadurch deutlich, dass die Struktur der Anzahl an Polygonen in sichtbaren Objekten, wenig Ähnlichkeit mit der Menge der gerenderten Polygone aufweist. Die Abbildung lässt auch erkennen, dass die Renderingzeit direkt durch die Menge der gerenderten Geometrie bestimmt wird; das Verfahren benötigt zur Laufzeit neben dem Rendering nur einen sehr geringen Zusatzaufwand für die Abfrage der Menge der aktuell als sichtbar klassifizierten Objekte.

Die Auswertung der Verteilungen in Abbildung 7.7 zeigt, dass in dem untersuchten Szenario der SVS-Algorithmus dem CHC++-Algorithmus im Allgemeinen deutlich überlegen ist. Beispielsweise beträgt die durchschnittliche Renderingzeit des SVS-Algorithmus 18 ms; beim CHC++ sind es 55 ms. Auch die durchschnittliche Anzahl der gerenderten Polygone beträgt beim SVS mit 7.1 Millionen weniger als die Hälfte des CHC++ mit 15 Millionen. Des Weiteren beachtet der SVS-Algorithmus (in der hier untersuchten Standardvariante) nicht die lokale Verdeckung durch nahe Objekte und wird daher vom CHC++ in einigen kleinen Regionen in der Szene durch eine geringere Renderingzeit übertroffen. Insgesamt erreicht der CHC++ eine

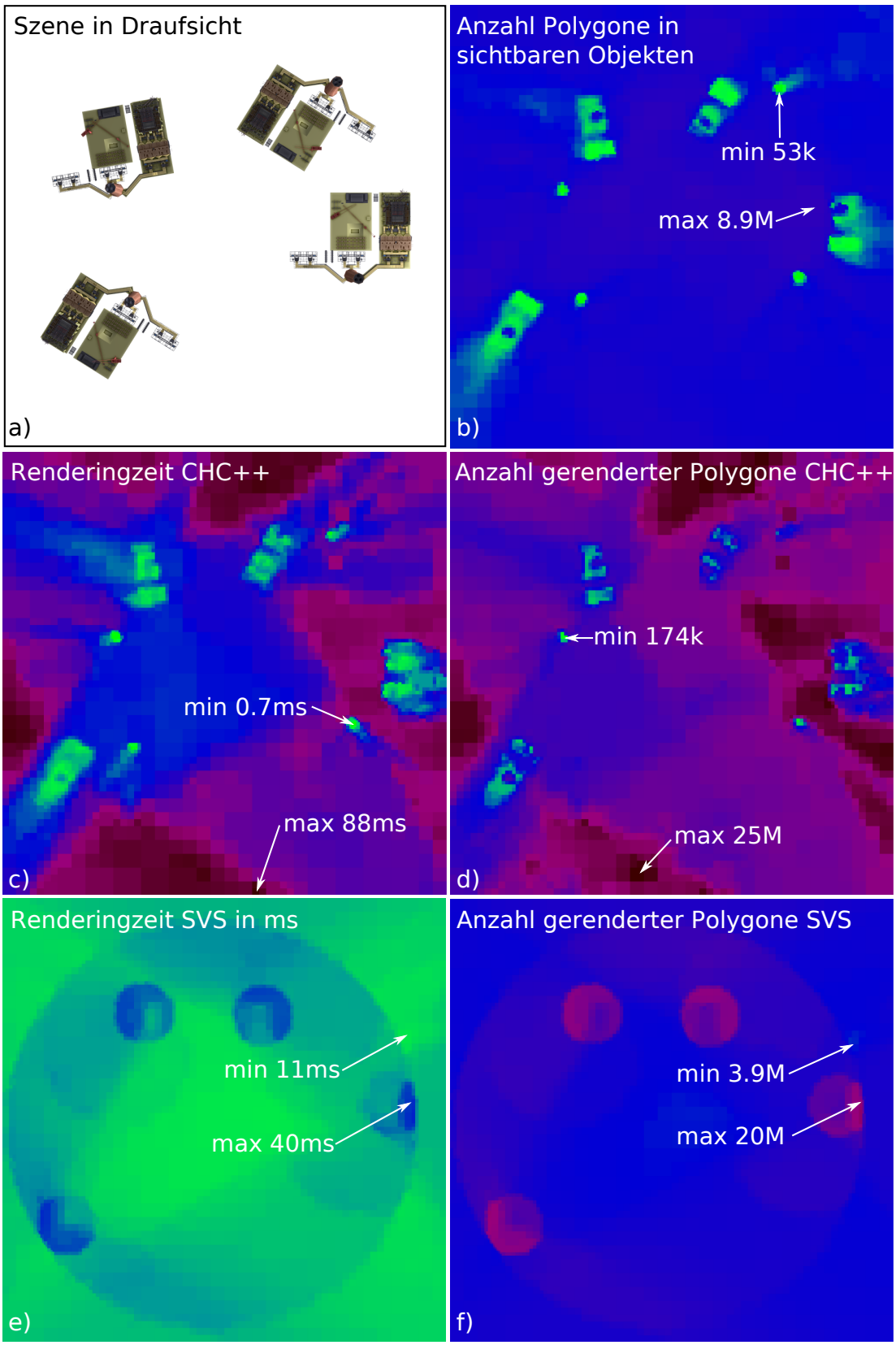


Abbildung 7.8: Szene mit vier Power-Plant-Modellen (51M Polygone): a) Szene in Draufsicht; b-f) Visualisierung unterschiedlicher Szeneneigenschaften mit minimalem und maximalen Wert

minimale Renderingzeit von unter einer Millisekunde, wobei diese beim SVS 11 ms beträgt. Für eine weiterführende Evaluierung des SVS-Algorithmus ergibt sich ein Ausgangspunkt für weitere Untersuchungen: Der SVS-Algorithmus ist ein approximatives Sichtbarkeitsverfahren. Das bedeutet, dass nicht ausgeschlossen ist, dass auch sichtbare Objekte als unsichtbar klassifiziert werden. In einem nächsten Schritt könnte durch eine Analyse der Bildqualität als Eigenschaft untersucht werden, wie groß der dadurch erzeugte Fehler ist.

7.5 Auswahl von Renderingmethoden zur Laufzeit

Neben der Entscheidung für ein bestimmtes Renderingverfahren in der Vorverarbeitung, kann diese Entscheidung auch abhängig von der aktuellen Kameraposition zur Laufzeit getroffen werden. Dafür können die in der Vorverarbeitung gemessenen Renderingzeiten mehrerer Verfahren verwendet werden, um dann zur Laufzeit das für die aktuelle Position als schnellstes bewertete Verfahren auszuwählen. Ein Nachteil bei der Wahl der Renderingzeit als Entscheidungskriterium ist, dass sich die Ergebnisse sehr stark an die bei der Messung verwendete Hardware gebunden sind. Robuster und portabler ist es daher, anhand einer allgemeineren Szeneneigenschaft (wie der Sichtbarkeit) auf die Situation an der Position des Betrachters zu schließen und daraufhin den Algorithmus auszuwählen. Generell bleibt das Problem, das bei allen betrachteten Szeneneigenschaften der Einfluss der Sichtrichtung des Betrachters abstrahiert wird, so dass es bei Entscheidungen für eine konkrete Richtung zu Ungenauigkeiten kommen kann.

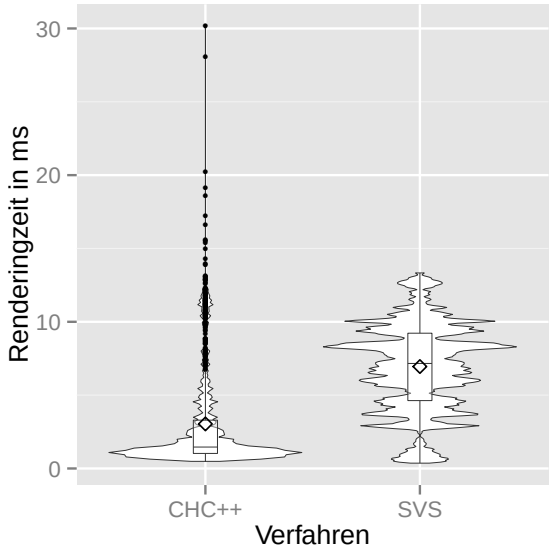


Abbildung 7.9: Vergleich der richtungsabhängigen Renderingzeit von CHC++ und SVS in Bereichen mit hoher Verdeckung; Messungen an 1000 zufälligen Positionen (max. 0.5M Polygone in sichtbaren Objekten) und zufälligen Richtungen; Szene mit vier Power-Plant-Modellen

Als Beispiel bietet sich direkt das im vorherigen Abschnitt 7.4 beschriebene unterschiedliche Verhalten der Algorithmen CHC++ und SVS an. In Bereichen mit einer hohen Verdeckung durch nahe Objekte funktioniert der CHC++-Algorithmus besonders gut (siehe Abbildung 7.8). Sobald der Betrachter jedoch hauptsächlich Objekte von außen in größerer Entfernung sieht, steigt die Renderingzeit deutlich. Der SVS-Algorithmus zeigt genau das gegenteilige Verhalten: Außenansichten funktionieren prinzipiell deutlich besser als Standpunkte innerhalb oder dicht neben Objekten. An Positionen mit erwartet hoher Verdeckung (weniger als 0.5M Polygone laut der Szeneneigenschaft *Anzahl Polygone in sichtbaren Objekten*) wird daher der CHC++-Algorithmus verwendet, an allen übrigen Positionen der SVS-Algorithmus. Für die Evaluierung werden 1000 zufälligen Positionen und Sichtrichtungen innerhalb der Regionen mit hoher Verdeckung betrachtet und die Renderingzeit mit beiden Algorithmen gemessen. Die Messung wird in diesem Falle nicht durch die Bestimmung einer positionsabhängigen Szeneneigenschaft durchgeführt, um explizit den Einfluss der Sichtrichtung mit einzubeziehen. Die Auswertung in Abbildung 7.9 zeigt, dass in den untersuchten Bereichen durch die Wahl des CHC++-Algorithmus anstatt des SVS die Renderingzeit im Mittel von 7 ms deutlich auf 3 ms reduziert werden kann. Da in der verwendeten Szene der Bereich mit hoher Verdeckung sehr klein ist, ist jedoch der Vorteil der automatischen Auswahl bezogen auf die ganze Szene sehr gering.

7.6 Rendering: Sichtbarkeit als positionsabhängige Eigenschaft

Neben der Bewertung, Anpassung und Auswahl existierender Renderingalgorithmen durch positionsabhängige Szeneneigenschaften, lassen auch eigenständige Renderingalgorithmen auf Basis von Szeneneigenschaften entwickeln. Ein einfaches Verfahren basiert auf der Szeneneigenschaft der *Menge sichtbarer Objekte* (siehe Abschnitt 4.2.2). Diese kodiert für eine gegebene Position, welche der Objekte der Szene als sichtbar klassifiziert sind – die Eigenschaft nähert damit die globale Sichtbarkeit der Szene an. Zur Laufzeit wird die Menge der potentiell sichtbaren Objekte aus der genäherten Szeneneigenschaft abgefragt und gerendert. Das Verfahren bietet eine kurze Renderingzeit, hat jedoch einige Nachteile: Zum einen benötigt die Kodierung der Menge der sichtbaren Objekte im Vergleich zu einer einfacheren Szeneneigenschaft, aber auch gegenüber anderen globalen Sichtbarkeitsverfahren, mehr Speicherplatz. Zum anderen kommt es durch die begrenzte Genauigkeit einer genäherten Szeneneigenschaft zu Bildfehlern, wenn eigentlich sichtbare Objekte an einer Position als unsichtbar klassifiziert wurden. Im direkten Vergleich zum SVS-Algorithmus bietet das Verfahren den Nachteil, dass die Sichtbarkeitsinformationen nur für den in der Vorverarbeitung untersuchten Bereich gültig sind; nähert man die Sichtbarkeitseigenschaft nur in einer 2-D-Region, kann sich der Betrachter auch nur in dieser Region bewegen. Die Größe bzw. das Volumen des untersuchten Bereichs wirken sich direkt auf die Anzahl der benötigten Samples und auf den benötigten Speicherplatzbedarf aus. Die Betrachtung der richtungsabhängigen Sichtbarkeit beim SVS ist hier klar im Vorteil.

Für die Evaluierung verwende ich die Szene mit vier Power-Plant-Modellen aus den vorangehenden Abschnitten. Die Eigenschaft der Menge der sichtbaren Objekte wurde mit

4096 Samples in einem 2-D-Schnitt durch die Szene ermittelt. Die Datengröße für die Sichtbarkeitsinformationen beträgt 31 MB (ein großer Teil davon ist jedoch nicht verfahrens-, sonder implementierungsbedingt). Um die Qualität des Algorithmus zu bewerten, wird die Renderingzeit und die Bildqualität durch genäherte Szeneneigenschaften untersucht.

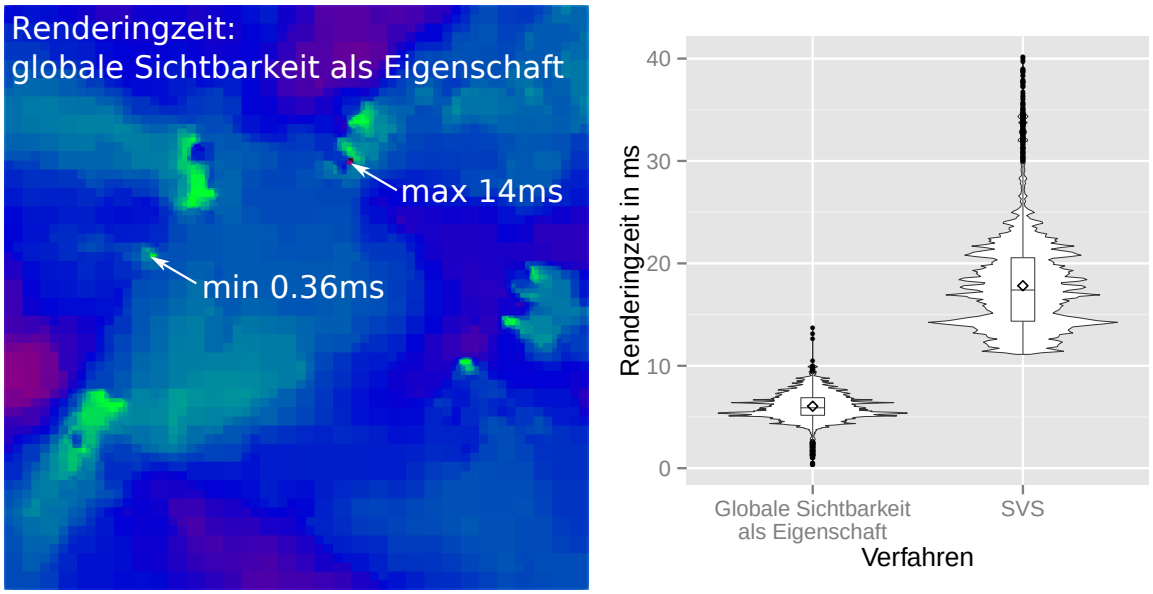


Abbildung 7.10: Renderingzeit des globalen Sichtbarkeitsverfahrens: (links) visuelle Darstellung mit minimalen und maximalen Wert; (rechts) Statistische Verteilung der Werte, inklusive Vergleichsdaten des SVS-Algorithmus aus Abschnitt 7.4

Die Renderingzeit des Verfahrens ist noch einmal kürzer als die des SVS-Verfahrens (siehe Abbildung 7.4). Der größte gemessene Wert beträgt nur 14 ms. Die Bildqualität erreicht an den meisten Positionen sehr gute Werte, an denen optisch kein Unterschied zum konservativen Verfahren feststellbar sind. An den wenigen Positionen jedoch, an denen es zu Fehlern im Sampling gekommen ist, können die Bildfehler sehr massiv ausfallen. In Abbildung 7.11 sieht man ein Bild aus dem Bereich mit minimaler Qualität: große Teile der Rohrleitungen aus dem Innenraum eines der Power-Plant-Modelle fehlen fast vollständig.

In der Arbeit *„Preprocessed Global Visibility for Real-Time Rendering on Low-End Hardware“* von Eikel, Jähn und Fischer [EJF10] wurden aufbauend auf diesem einfachen Sichtbarkeitsverfahren einige Erweiterungen entwickelt, um dadurch das Rendering von komplexen Szenen auf Endgeräten mit leistungsschwacher Hardware zu ermöglichen. Um die Größe des benötigten Speicherplatzes zu reduzieren, werden Regionen mit ähnlicher Sichtbarkeit zusammengefasst und so die Anzahl der zu speichernden Regionen reduziert. Äquivalent können auch Objekte zusammengefasst werden, die weitestgehend nur aus den gleichen Regionen sichtbar sind. Dadurch sinkt jedoch auch die Genauigkeit der genäherten Sichtbarkeitsdaten. Obwohl die Menge der von dem Verfahren dargestellten Geometrie schon sehr gering ist, kann sie die Leistungsfähigkeit einfacher Systeme noch überfordern, so dass die Menge noch einmal reduziert wird. Dies geschieht dadurch, dass in der zugrundeliegenden Szeneneigenschaft nicht nur die Menge der sichtbaren Objekte kodiert wird, sondern die Anzahl der sichtbaren

Abbildung 7.11: Bildqualität des globalen Sichtbarkeitsverfahrens (hierarchisches SSIM): (links) visuelle Darstellung mit minimalen und maximalen Werten; (rechts) Gerendertes Beispielbild und Original aus einer Position mit minimaler Bildqualität

Pixel jedes Objekts. Die Eigenschaft kodiert also den geschätzten Einfluss jedes Objektes zum gerenderten Bild. Während des Renderings können jetzt die Objekte mit dem größten geschätzten Einfluss ausgewählt und gerendert werden, bis ein maximaler Wert an gerenderten Polygonen erreicht ist. Dadurch vergrößert sich der Bildfehler zwar deutlich, es ist aber möglich auch mit sehr leistungsschwacher Hardware flüssig durch die Szene zu navigieren. Sobald der Betrachter seine Position nicht mehr verändert, können kontinuierlich die verbleibenden sichtbaren Objekte gerendert werden.

8 Fazit und Ausblick

Im Folgenden ziehe ich ein Fazit für die unterschiedlichen, in dieser Arbeit vorgestellten Aspekte und zeige Potentiale für zukünftige Arbeiten auf.

Positionsabhängige Szeneneigenschaften

Das Konzept der positionsabhängigen Szeneneigenschaften ist ein neues Werkzeug für die experimentelle Evaluierung von Renderingalgorithmen. Global genäherte Szeneneigenschaften können bei Messungen deutliche Vorteile gegenüber dem gebräuchlichen Einsatz von Kamerapfaden bieten. Statistische Aussagen über das generelle Verhalten von Algorithmen sind durch die Auswertung global genäherter Szeneneigenschaften deutlich aussagekräftiger, da sie nicht von der subjektiven Wahl eines Kamerapfades abhängen. Die visuelle Darstellung kann einen intuitiven und unmittelbaren Einblick in die Funktionsweise von Algorithmen geben; sowohl für den Entwickler beim Entwurf eines Algorithmus als auch zur Veranschaulichung des Algorithmus in einer wissenschaftlichen Publikation oder in der Lehre. Die vorgestellte Evaluierungsmethode kann klassische Methoden jedoch nur ergänzen und nicht ersetzen, da das spezifische Verhalten eines Algorithmus an einer bestimmten Position und Blickrichtung nur ungenau abgebildet wird. Auch das Verhalten von Algorithmen in Bezug auf längerfristige temporale Kohärenz (beispielsweise von Out-of-Core-Verfahren) lässt sich nur schwer als Szeneneigenschaft abbilden. Global genäherte Szeneneigenschaften können auch zur Laufzeit eingesetzt werden, um das Rendering zu verbessern. Es zahlt sich jedoch meistens eher aus, die gewonnenen Evaluierungsergebnisse eher für eine Verbesserung und Anpassung der verwendeten Renderingalgorithmen einzusetzen, da durch den Einfluss der Blickrichtung zur Laufzeit die erzielbaren Verbesserungen meist nur marginal sind.

Mögliches Potential zur Verbesserung der Methode sehe ich vor allem in Erweiterungen der Sampling-Methoden. Um sehr weitläufige Szenen effizienter zu untersuchen, könnten die Sampling-Positionen beispielsweise auf die Regionen konzentriert werden, in denen sich die Benutzer im geplanten Anwendungsszenario gehäuft aufhalten. Eine weitere Möglichkeit besteht darin, die Sampling-Positionen auf die Bereiche der Szene einzuschränken, die von einem Benutzer überhaupt erreichbar sind. Im Rahmen des Algorithm-Engineerings im Bereich der Computergrafik wäre es wünschenswert, wenn sich weitere Arbeiten mit der systematischen Evaluierung unterschiedlicher Aspekte beschäftigten, wie etwa der objektiven Bewertung der Bildqualität von Näherungsverfahren.

Progressive-Blue-Surfels

Mit dem Progressive-Blue-Surfels-Verfahren habe ich ein robustes Verfahren vorgestellt, dass es erlaubt, sehr große Szenen in Echtzeit darzustellen. Das Verfahren unterstützt unterschiedlich

strukturierte Objekte, inklusive komplexer CAD-Daten, gescannter Oberflächenmodelle, weitläufiger, prozedural generierter Landschaften, Texturen und anderer Oberflächeneigenschaften und dynamischer Beleuchtung. Der Laufzeit- und Speicherplatzbedarf in der Vorbereitung erlauben auch die Verarbeitung von sehr großen Szenen mit mehreren Milliarden Polygonen. Auf technischer Ebene verläuft das Rendering sehr effizient, da nur wenige, zusammenhängende Datenblöcke (Surfel-Präfixe) von der Grafikkarte verarbeitet werden müssen. Die Bildqualität des Verfahrens ist in den meisten Fällen gut, auch wenn die Näherung der Szene zu erkennen ist – es handelt sich nicht um ein konservatives Verfahren. Positiv in Bezug auf Bildqualität fällt auf, dass durch die flexible Wahl der Präfixlänge fast keine Popping-Artefakte beim Umschalten unterschiedlicher Qualitätsstufen zu erkennen sind, die sonst bei vielen hierarchisch arbeitenden Näherungsverfahren störend auffallen. Negativ fallen jedoch Löcher in Oberflächen oder zu grob aufgelöste Oberflächen auf, die bei unpassender Parameterwahl auftreten können. Bei der Wahl der Renderingparameter liegen die Grenzen in der Robustheit des Verfahrens. Eine weitere Grenze teilt sich das Verfahren mit zahlreichen anderen Näherungsverfahren, in denen Teile der Szene mit kleiner projizierter Größe durch Ersatzrepräsentationen angenähert werden: Befindet sich zu viel Geometrie mit großer projizierter Größe im Sichtbereich, wird das Verfahren für diese komplexen Bereiche nicht eingesetzt und die Renderingzeit steigt (oder die Qualität sinkt).

Obwohl das Verfahren in seiner jetzigen Form sehr gut funktioniert, sehe ich einige Anknüpfungspunkte für zukünftige Erweiterungen:

- Es können weitere Eigenschaften der Originalgeometrie in die Surfels kodiert werden: von weiteren Oberflächeneigenschaften bis hin zu Daten, die eine begrenzte Animation der Objekte im Shader ermöglichen.

- Das Rendering sollte angepasst werden, um die Bildqualität weniger abhängig von den Renderingparametern zu machen.

- Die Form eines Surfels bei der Darstellung könnte weiter angepasst werden, um den möglichen Beitrag eines Surfels zum Gesamtbild zu steigern und so mit weniger Surfels auszukommen.

- Das Verfahren könnte mit existierenden Occlusion-Culling-Algorithmen kombiniert werden, um einer zu hohen Komplexität im Nahbereich zu begegnen.

PADrend

Das PADrend-System hat sich mittlerweile zu einem umfangreichen Renderingsystem entwickelt, das bereits erfolgreich in der Entwicklung mehrerer Renderingalgorithmen eingesetzt wurde. In der Lehre wird das System als Ausgangsbasis für Abschluss- und Projektarbeiten im Bereich Computergrafik eingesetzt. Hier sorgt es für eine deutliche Reduzierung des Einarbeitungsaufwandes, so dass Studierende sich besser auf die eigentliche Fragestellung konzentrieren können und weniger Zeit mit der Klärung technischer Details verbringen. In Projekten wird das System für virtuelle Design-Reviews im industriellen Kontext eingesetzt.

Hier werden auf Basis von CAD-Daten Fragestellungen und Varianten des technischen Designs von komplexen Maschinen betrachtet und diskutiert.

Die grundlegende Softwarearchitektur von PADrend hat sich als sehr robust herausgestellt, so dass es auch in Zukunft in der Algorithmenentwicklung eingesetzt werden kann. Es bleibt jedoch notwendig, auch weiterhin neue technische Funkionen, die sich beispielsweise aus der Weiterentwicklung von OpenGL ergeben, in PADrend zu integrieren, um aktuelle Forschungsfragen adäquat untersuchen zu können. Die wesentliche Herausforderung für den langfristigen Einsatz von PADrend sehe ich in der weiteren Verbesserung der Benutzerschnittstellen und die Erstellung geeigneter Anleitungen, um langfristig genügend Nutzer und Entwickler für das System interessieren zu können.

Literaturverzeichnis

[AMHH08] AKENINE-MÖLLER, TOMAS, ERIC HAINES und NATY HOFFMAN: *Real-Time Rendering*. A K Peters, Ltd., Wellesley, MA, USA, 3. Auflage, 2008.

[BMW+09] BITTNER, JIŘÍ, OLIVER MATTAUSCH, PETER WONKA, VLASTIMIL HAVRAN und MICHAEL WIMMER: *Adaptive global visibility sampling*. ACM Transactions on Graphics, 28(3):1–10, Juli 2009.

[Bur81] BURT, PETER J.: *Fast filter transform for image processing*. Computer Graphics and Image Processing, 16(1):20–51, Mai 1981.

[BWPP04] BITTNER, JIŘÍ, MICHAEL WIMMER, HARALD PIRINGER und WERNER PURGATHOFER: *Coherent Hierarchical Culling: Hardware Occlusion Queries Made Useful*. Computer Graphics Forum, 23(3):615–624, September 2004. Proceedings of Eurographics 2004.

[Cat74] CATMULL, EDWIN EARL: *A subdivision algorithm for computer display of curved surfaces*. Doktorarbeit, Department of Computer Science, University of Utah, Salt Lake City, UT, USA, Dezember 1974.

[CCC+08] CIGNONI, PAOLO, MARCO CALLIERI, MASSIMILIANO CORSINI, MATTEO DELLEPIANE, FABIO GANOVELLI und GUIDO RANZUGLIA: *MeshLab: an Open-Source Mesh Processing Tool*. In: SCARANO, VITTORIO, ROSARIO DE CHIARA und UGO ERRA (Herausgeber): *Eurographics Italian Chapter Conference*, Seiten 129–136. Eurographics Association, 2008.

[CDL+96] CHAMBERLAIN, BRADFORD, 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, Seiten 132–141, Toronto, Ont., Canada, Mai 1996. Canadian Information Processing Society.

[Cla76] CLARK, JAMES H.: *Hierarchical geometric models for visible surface algorithms*. Communications of the ACM, 19(10):547–554, Oktober 1976.

[COCSD03] COHEN-OR, DANIEL, YIORGOS CHRYSANTHOU, CLÁUDIO T. SILVA und FRÉDO DURAND: *A survey of visibility for walkthrough applications*. IEEE Transactions on Visualization and Computer Graphics, 9(3):412–431, 2003.

[Coo86] COOK, ROBERT L.: *Stochastic sampling in computer graphics*. ACM Trans. Graph., 5(1):51–72, Januar 1986.

[Del34] DELAUNAY, BORIS N.: *Sur la sphère vide*. Izvestia Akademii Nauk SSSR, Otdelenie Matematicheskikh i Estestvennykh Nauk, 6:793–800, Oktober 1934.

[DFI03] DEMETRESCU, CAMIL, IRENE FINOCCHI und GIUSEPPE F. ITALIANO: *Algorithm engineering, Algorithmics Column*. Bulletin of the EATCS, 79:48–63, 2003.

[EJF10] EIKEL, BENJAMIN, CLAUDIUS JÄHN und MATTHIAS FISCHER: *Preprocessed Global Visibility for Real-Time Rendering on Low-End Hardware*. In: BEBIS, GEORGE, RICHARD BOYLE, BAHRAM PARVIN, DARKO KORACIN, RONALD CHUNG, RIAD HAMMOUD, MUHAMMAD HUSSAIN, TAN KAR-HAN, ROGER CRAWFIS, DANIEL THALMANN, DAVID KAO und LISA AVILA (Herausgeber): *Advances in Visual Computing*, Band 6453 der Reihe *Lecture Notes in Computer Science*, Seiten 622–633. Springer Berlin Heidelberg, 2010. Proceedings of the 6th International Symposium on Visual Computing (ISVC 2010).

[EJFMadH13] EIKEL, BENJAMIN, CLAUDIUS JÄHN, MATTHIAS FISCHER und FRIEDHELM MEYER AUF DER HEIDE: *Spherical Visibility Sampling*. Computer Graphics Forum, 32(4):49–58, Juli 2013. Proceedings of the 24th Eurographics Symposium on Rendering.

[EJP11] EIKEL, BENJAMIN, CLAUDIUS JÄHN und RALF PETRING: *PADrend: Platform for Algorithm Development and Rendering*. In: GAUSEMEIER, JÜRGEN, MICHAEL GRAFE und FRIEDHELM MEYER AUF DER HEIDE (Herausgeber): *Augmented & Virtual Reality in der Produktentstehung*, Band 295 der Reihe *HNI-Verlagsschriftenreihe*, Seiten 159–170. Heinz Nixdorf Institut, Universität Paderborn, Mai 2011.

[GABK06] GUTHE, MICHAEL, ÁKOS BALÁZS und REINHARD KLEIN: *Near Optimal Hierarchical Culling: Performance Driven Use of Hardware Occlusion Queries*. In: AKENINE-MÖLLER, TOMAS und WOLFGANG HEIDRICH (Herausgeber): *Proceedings of the 17th Eurographics Symposium on Rendering*, EGSR '06, Seiten 207–214. Eurographics Association, Juni 2006.

[Gei07] GEISS, RYAN: *Gpu Gems 3*, Kapitel Generating Complex Procedural Terrains Using the GPU, Seiten 7–37. Addison-Wesley Professional, 2007.

[GKM93] GREENE, NED, MICHAEL KASS und GAVIN MILLER: *Hierarchical Z-Buffer Visibility*. In: *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '93, Seiten 231–238, New York, NY, USA, 1993. ACM.

[Hop96] HOPPE, HUGUES: *Progressive meshes*. In: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, Seiten 99–108, New York, NY, USA, 1996. ACM.

[Jäh13] JÄHN, CLAUDIUS: *Progressive Blue Surfels*. Technischer Bericht, Heinz Nixdorf Institut, Universität Paderborn, 2013. http://arxiv.org/abs/1307.0247.

[JEF+13] JÄHN, CLAUDIUS, BENJAMIN EIKEL, MATTHIAS FISCHER, RALF PETRING und FRIEDHELM MEYER AUF DER HEIDE: *Evaluation of Rendering Algorithms using Position-Dependent Scene Properties*. In: BEBIS, GEORGE, RICHARD BOYLE, BAHRAM PARVIN, DARKO KORACIN, BAOXIN LI, FATIH PORIKLI, VICTOR ZORDAN, JAMES KLOSOWSKI, SABINE COQUILLART, XUN LUO, MIN CHEN und DAVID GOTZ (Herausgeber): *Advances in Visual Computing*, Band 8033 der Reihe *Lecture Notes in Computer Science*, Seiten 108–118. Springer Berlin Heidelberg, 2013. Proceedings of the 9th International Symposium on Visual Computing (ISVC 2013).

[MBW08] MATTAUSCH, OLIVER, JIŘÍ BITTNER und MICHAEL WIMMER: *CHC++: Coherent Hierarchical Culling Revisited*. Computer Graphics Forum, 27(2):221–230, April 2008. Proceedings of Eurographics 2008.

[MF92] MCCOOL, MICHAEL und EUGENE FIUME: *Hierarchical Poisson Disk Sampling Distributions*. In: *Proceedings of the Conference on Graphics Interface '92*, Seiten 94–105, San Francisco, CA, USA, 1992. Morgan Kaufmann Publishers Inc.

[PD90] PLANTINGA, HARRY und CHARLES R. DYER: *Visibility, occlusion, and the aspect graph*. International Journal of Computer Vision, 5(2):137–160, November 1990.

[PEJ+13] PETRING, RALF, BENJAMIN EIKEL, CLAUDIUS JÄHN, MATTHIAS FISCHER und FRIEDHELM MEYER AUF DER HEIDE: *Real-Time 3D Rendering of Heterogeneous Scenes*. In: BEBIS, GEORGE, RICHARD BOYLE, BAHRAM PARVIN, DARKO KORACIN, BAOXIN LI, FATIH PORIKLI, VICTOR ZORDAN, JAMES KLOSOWSKI, SABINE COQUILLART, XUN LUO, MIN CHEN und DAVID GOTZ (Herausgeber): *Advances in Visual Computing*, Band 8033 der Reihe *Lecture Notes in Computer Science*, Seiten 448–458. Springer Berlin Heidelberg, 2013. Proceedings of the 9th International Symposium on Visual Computing (ISVC 2013).

[PT02] PANTAZOPOULOS, IOANNIS und SPYROS TZAFESTAS: *Occlusion Culling Algorithms: A Comprehensive Survey*. J. Intell. Robotics Syst., 35(2):123–156, November 2002.

[PZvBG00] PFISTER, HANSPETER, 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, Seiten 335–342, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[RL00] RUSINKIEWICZ, SZYMON und MARC LEVOY: *QSplat: a multiresolution point rendering system for large meshes*. In: *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '00, Seiten 343–352, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[San09] SANDERS, PETER: *Algorithm Engineering – An Attempt at a Definition*. In: *Efficient Algorithms*, Band 5760 der Reihe *Lecture Notes in Computer Science*, Seiten 321–340. Springer, 2009.

[SKJF11] SÜSS, TIM, CLEMENS KOCH, CLAUDIUS JÄHN und MATTHIAS FISCHER: *Approximative occlusion culling using the hull tree*. In: *Proceedings of Graphics Interface 2011*, GI '11, Seiten 79–86, School of Computer Science, University of Waterloo, Waterloo, Ontario, Canada, Mai 2011. Canadian Human-Computer Communications Society.

[SLS+96] SHADE, JONATHAN, DANI LISCHINSKI, DAVID H. SALESIN, TONY DEROSE und JOHN SNYDER: *Hierarchical image caching for accelerated walkthroughs of complex environments*. In: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, SIGGRAPH '96, Seiten 75–82, New York, NY, USA, 1996. ACM.

[TS91] TELLER, SETH J. und CARLO H. SÉQUIN: *Visibility preprocessing for interactive walkthroughs*. In: *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '91, Seiten 61–70, New York, NY, USA, 1991. ACM.

[Ulr00] ULRICH, THATCHER: *Loose Octrees*. In: DELOURA, MARK (Herausgeber): *Game Programming Gems*, Game Programming Gems, Kapitel 4.11, Seiten 444–453. Charles River Media, Boston, MA, USA, 2000.

[WBSS04] WANG, ZHOU, ALAN CONRAD BOVIK, HAMID RAHIM SHEIKH und EERO P. SIMONCELLI: *Image quality assessment: from error visibility to structural similarity*. Image Processing, IEEE Transactions on, 13(4):600–612, April 2004.

[WFP+01] WAND, MICHAEL, MATTHIAS FISCHER, INGMAR PETER, FRIEDHELM MEYER AUF DER HEIDE und WOLFGANG STRASSER: *The randomized z-buffer algorithm: interactive rendering of highly complex scenes*. In: *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, SIGGRAPH '01, Seiten 361–370, New York, NY, USA, 2001. ACM.

[YGL97] YUAN, PING, MARK GREEN und RYNSON W. H. LAU: *A framework for performance evaluation of real-time rendering algorithms in virtual reality*. In: *Proceedings of the ACM symposium on Virtual reality software and technology*, VRST '97, Seiten 51–58, New York, NY, USA, 1997. ACM.

[ZMHH97] ZHANG, HANSONG, 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, Seiten 77–88, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.