UNIVERSITÄT
KOBLENZ · LANDAU

Fachbereich 4: Informatik

Point Rendering

Masterarbeit

zur Erlangung des Grades Master of Science (M.Sc.)
im Studiengang Computervisualistik

vorgelegt von

Alexander Seggebäing

Erstgutachter:	Prof. Dr.-Ing. Stefan Müller (Institut für Computervisualistik, AG Computergraphik)
Zweitgutachter:	Alexander Maximilian Nilles, M.Sc. (Institut für Computervisualistik, AG Computergraphik)

Koblenz, im November 2020

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

	Ja	Nein
Mit der Einstellung der Arbeit in die Bibliothek bin ich einverstanden.	☒	☐

.....

(Ort, Datum) (Unterschrift)

Zusammenfassung

In dieser Arbeit werden Verfahren zum Rendern von Punktdaten vorgestellt und miteinander verglichen. Die Verfahren lassen sich in zwei Kategorien unterteilen. Zum einen werden visuelle Verfahren behandelt, welche sich mit der reinen Darstellung von Punktprimitiven befassen. Hauptproblem ist dabei die Darstellung von Oberflächen, da Punktdaten im Gegensatz zu traditionellen Dreiecksnetzen keine Nachbarschaftsinformationen beinhalten. Zum anderen werden beschleunigende Datenstrukturen dargelegt, welche die echtzeitfähige Darstellung von großen Punktwolken ermöglichen. Punktwolken weisen häufig eine hohe Datenmenge auf, da diese meist durch 3D-Scanningverfahren wie z. B. Laserscanning und Photogrammetrie generiert werden.

Abstract

In this thesis different methods for rendering point data are shown and compared with each other. The methods can be divided into two categories. For one visual methods are introduced that strictly deal with the displaying of point primitves. The main problem here lies in the depiction of surfaces since point data, unlike traditional triangle meshes, doesn't contain any connectivity information. On the other hand data strucutres are shown that enable real-time rendering of large point clouds. Point clouds often contain large amounts of data since they are mostly generated through 3D scanning processes such as laser scanning and photogrammetry.

Inhaltsverzeichnis

1 Einleitung **1**

2 Grundlagen **3**
2.1 Punkte als Renderprimitiv 3
2.2 Generierung von Punktwolken 4
2.3 Out-of-Core-Algorithmen und Datenstrukturen 7
2.3.1 K-D-Tree 7
2.3.2 Octree 8
2.4 Splatting 8
2.4.1 Größe 9
2.4.2 Form 11

3 Punktrendering-Verfahren **14**
3.1 Visuelle Verfahren 14
3.1.1 Splat-Blending 14
3.1.2 Voronoi-Interpolation 17
3.2 Datenstrukturen 18
3.2.1 QSplat 18
3.2.2 Modifiable-Nested-Octree 23
3.2.3 Continuous-Level-of-Detail 26

4 Evaluation **31**
4.1 Testumgebung 31
4.2 Visuelle Verfahren 31
4.2.1 Punktwolken 31
4.2.2 Leistung 32
4.2.3 Darstellung 33
4.3 Datenstrukturen 36
4.3.1 Punktwolken 36
4.3.2 Speicheranforderungen 36
4.3.3 Leistung 37
4.3.4 Darstellung 43

5 Fazit **44**

1 Einleitung

3D-Scanningverfahren finden heutzutage in immer mehr Bereichen Anwendung. So werden diese unter anderem in den Geowissenschaften zur topografischen Vermessung von Geländestrukturen, im Bergbau zur Erfassung von Höhlen und Schächten, im Bauwesen zur Bauplanung und Baukontrolle, im Forstwesen zur Überwachung von Waldwachstum und in der Archäologie zur Erfassung von historischen Stätten genutzt [VM10]. Die verbreitetsten Verfahren sind dabei das 3D-Laserscanning und die Photogrammetrie. Beide Verfahren resultieren häufig in großen Punktwolken deren Darstellung und Handhabung ein eigenständiges Problem darstellt.

Eine grundlegende Schwierigkeit liegt dabei in der Nutzung von Punkten als Renderprimitiv. Da Punkte lediglich eine Position im Raum definieren und weder über Größe noch Nachbarschaftsinformation verfügen, ist die Darstellung von Oberflächen nicht trivial. Die Darstellung ist zusätzlich dadurch erschwert, dass 3D-Scanningverfahren rauschanfällig sind, die Abtastrate schwankt und Teile der Umgebung häufig überhaupt nicht erfasst werden. Eine Möglichkeit zur Darstellung von Punktwolken ist die Generierung von Polygonnetzen aus den Punktdaten. Diese können anschließend mithilfe von konventionellen Methoden gerendert werden. Problematisch ist jedoch, dass die Berechnung der Polygonnetze aufwändig ist und häufig eine manuelle Korrektur von Löchern und diversen anderen Artefakten erfordert [WS06]. Da in vielen Anwendungsbereichen die sofortige Betrachtung der Daten notwendig ist, ist es stattdessen von Interesse, die Punktdaten an sich und ohne aufwändige Vorverarbeitungsschritte zu rendern.

Ein weiteres Hauptproblem stellt die enorme Datenmenge dar, welche beim 3D-Scanning generiert wird. So können moderne 3D-Laserscanner mehrere Millionen Punkte pro Sekunde abtasten [Lei]. Die entstehenden Punktwolken überschreiten häufig die verfügbaren Arbeitsspeicherkapazitäten und verhindern eine interaktive Darstellung in Echtzeit. Eine triviale Möglichkeit solche Datensätze in Echtzeit darzustellen, ist es nur ausgewählte Teilbereiche zu rendern oder die Punktwolke durch Downsampling zu vereinfachen. Da Punktwolken in vielen Anwendungsfällen zur Vermessung der Umgebung oder Betrachtung von Details verwendet werden, sind diese Verfahren jedoch häufig aus praktischen Gründen keine Option. Es werden daher Verfahren benötigt, welche ein echtzeitfähiges Rendering der gesamten Punktwolke ermöglichen, wobei relevante Bereiche mit maximaler Detailstufe dargestellt werden.

In dieser Arbeit werden verschiedene Punktrenderingverfahren vorgestellt und miteinander verglichen. Dabei werden zum einen visuelle Verfahren behandelt, welche sich mit der reinen Darstellung von Punktprimitiven befassen. Zum anderen werden beschleunigende Datenstrukturen aufgezeigt, welche die echtzeitfähige Darstellung von großen Punktwolken

ermöglichen.

In Abschnitt 2 werden zunächst die für diese Arbeit notwendige Grundlangen behandelt. Dabei handelt es sich um die Nutzung von Punkten als Renderprimitiv, den Generierungsprozess von Punktwolken, die Funktionsweise von Out-of-Core Algorithmen und Datenstrukturen und das Rendern von Punkten durch Splatting. In Abschnitt 3 werden die in dieser Arbeit behandelten visuellen und beschleunigenden Punktrenderingverfahren vorgestellt. In Abschnitt 4 werden die Verfahren evaluiert und gegenübergestellt. Ein Fazit in Abschnitt 5 schließt die Arbeit ab.

2 Grundlagen

2.1 Punkte als Renderprimitiv

Das Dreieck hat sich in der Computergrafik als grundlegendes Renderprimitiv durchgesetzt, da es die einfachste geometrische Figur darstellt, welche eine planare Fläche definiert. Werden mehrere Dreiecke zu einem Netz (*Mesh*) verbunden, lassen sich beliebige 3D-Objekte modellieren. Beim Rendern werden die Objekte auf ein diskretes 2D-Pixelraster abgebildet, welches auf einem Bildschirm dargestellt werden kann. Mit steigender Komplexität der Objekte kommt es dabei vermehrt vor, dass einzelne Primitive auf einen Bereich der Größe eines Pixels oder noch kleiner abgebildet werden. Levoy und Whitted schlugen, unter anderem aus diesem Grund, erstmals vor, Punkte als Renderprimitive zu verwenden [LW85]. So stellten sie fest, dass Punktprimitive ab einer bestimmten Komplexität, keine visuellen Nachteile zu kohärenten Primitiven haben und gleichzeitig eine effizientere Rasterisierung ermöglichen (Abbildung 1). Durch die Abwesenheit von Nachbarschaftsinformation haben Punkte, im Vergleich zu anderen Primitiven, zusätzlich Speichervorteile und erlauben eine einfachere Berechnung von Level-of-Detail-Repräsentationen (*LOD*) [CH02].

(a) (b)

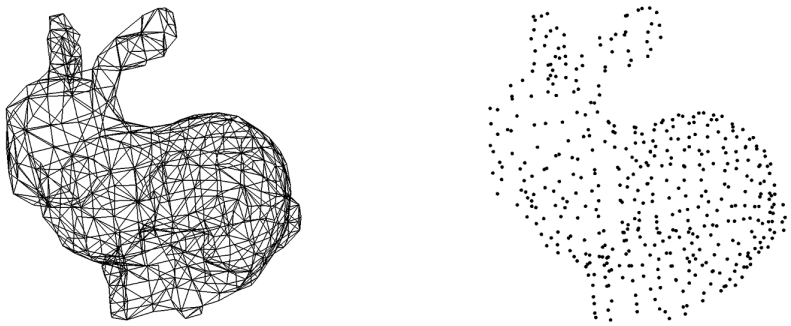
Abbildung 1: Rendering eines Drachen [XYZ] durch traditionelle Rasterisierung eines Meshs (a) und Rendering als Punktwolke, wobei jeder Vertex durch einen Pixel dargestellt wird (b). Die Komplexität des Models ist so hoch, dass die Primitive des Meshs bei der gewählten Kameraperspektive auf einen oder weniger Pixel abgebildet werden. Die Punktwolke kann das Model daher trotz fehlender Nachbarschaftsinformation als geschlossene Fläche darstellen und führt zu einem nahezu identischen Ergebnis wie das Rendern des Meshs.

Trotz mehrerer Fortschritte im Bereich des Punktrenderings, welche sowohl das Rendern von großen Punktwolken in Echtzeit erlauben als auch eine hohe visuelle Qualität ermöglichen, konnten sich Punkte nicht als allgemeines Renderprimitiv durchsetzen. Dies liegt unter anderem daran, dass Grafikhardware hauptsächlich an das Rendern von Dreiecken angepasst ist. Insbesondere vor der Einführung einer programmierbaren Shader-Pipe-

line mit OpenGL 2.0 [Khr], mussten Punktrenderer auf der CPU implementiert werden, wodurch diese deutliche Leistungsnachteile im Vergleich zu einem traditionellen Dreiecksrendering hatten [CH02]. Heutzutage erlaubt modernes OpenGL die effiziente Implementierung von Punktrendering auf der GPU [BHZK05]. Punktrendering wird jedoch nicht mehr als allgemeiner Ersatz für traditionelles Dreiecksrendering verfolgt. Stattdessen liegt der Fokus auf der Darstellung von großen Datensätzen, welche nur in Form von Punktwolken vorliegen und deren Konvertierung in Meshs aus praktischen Gründen nicht erwünscht oder möglich ist.

2.2 Generierung von Punktwolken

Viele der beim Punktrendering aufkommenden Probleme und Herausforderungen sind ein direktes Resultat der Punktwolkengenerierung [Sch14]. Die Generierung geschieht durch die Abtastung (*Sampling*) von geometrischen Objekten. Dabei können diese sowohl in digitaler Form vorliegen als auch aus der echten Welt stammen.



(a) (b)

Abbildung 2: Sampling des Stanford Bunny [Sta]. Die Vertices des Meshs (a) werden als Abtastpunkte verwendet (b). Die Abtastrate ist nicht gleichmäßig und muss beim Rendern der Punktwolke beachtet werden.

Bei digitalen Objekten kann es sich etwa um parametrisierte Geometrie handeln, welche in konstanten oder zufälligen Intervallen abgetastet werden kann oder um Meshs deren Vertices als Abtastpunkte dienen können. Ist das gewählte Abtastverfahren nicht spezifisch darauf abgestimmt, weisen die entstehenden Punktwolken i. d. R. keine gleichmäßige Abtastrate auf (Abbildung 2). Beim anschließenden Rendern muss diese Ungleichmäßigkeit beachtet werden, um die Darstellung von Flächen zu gewährleisten. Dazu kann den Punkten z. B. jeweils ein Radius zugeordnet werden, welcher abhängig von der Punktdichte ihrer Umgebung ist [RL00].

Zur Abtastung von Objekten aus der echten Welt werden 3D-Scanningverfahren verwendet. Die Erfassung der Daten geschieht dabei meist durch

Photogrammetrie oder 3D-Laserscanning und kann sowohl terrestrisch als auch luftgestützt durchgeführt werden.

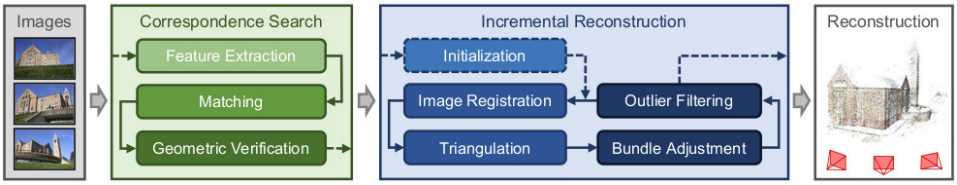


Abbildung 3: Die Structure-from-Motion-Pipeline. Abbildung aus [SF16].

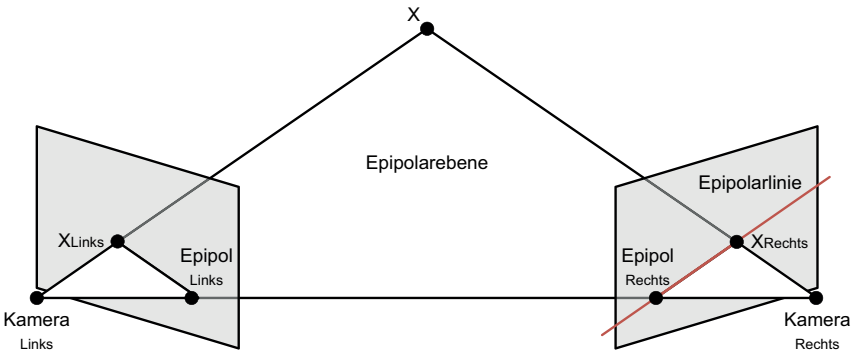


Abbildung 4: Die Epipolargeometrie. Ein Punkt X wird auf die Bildebene von zwei Kameras projiziert. Der Strahl vom Projektionszentrum der linken Kamera zur Abbildung des Punktes X_{Links} kann auf die Bildebene der rechten Kamera in Form einer Geraden projiziert werden und wird als Epipolarlinie bezeichnet. Die Abbildung von X auf der Bildebene der rechten Kamera liegt auf dieser Epipolarlinie.

Bei der Photogrammetrie wird das zu erfassende Objekt aus unterschiedlichen Positionen und Blickwinkeln fotografiert. Anschließend können 3D-Punkte aus der Bildmenge berechnet werden. Dazu benötigen moderne Structure-from-Motion-Algorithmen (*SfM*) weder intrinsisch kalibrierte Kameras noch Informationen über deren Posen. Abbildung 3 zeigt die SfM-Pipeline. Zunächst werden Merkmale aus den Bildern extrahiert und sich überschneidende Bildpaare identifiziert indem Punktkorrespondenzen in der Bildmenge gesucht werden. Um sicherzustellen, dass es sich bei den gefundenen Korrespondenzen tatsächlich um homologe Bildpunkte handelt, werden diese geometrisch verifiziert. Dazu wird mithilfe der Epipolargeometrie (Abbildung 4) eine Transformation gesucht, welche die Punkte von einem Bild in das andere überführt. Lassen sich genügend Punktpaare überführen, gelten die Bilder als geometrisch verifiziert. Ausgehend von einem Bildpaar lassen sich anschließend schrittweise weitere Bilder in das bestehende Model integrieren und 3D-Punkte durch Triangulierung der Punktkorrespondenzen berechnen. Um eine stabile 3D-Rekonstruktion

zu ermöglichen, werden dabei Ausreißer identifiziert und gefiltert. Aktuelle SfM-Algorithmen sind in der Lage große Bildmengen zu analysieren. Die entstehenden Punktwolken weisen jedoch häufig mehrere Probleme auf. So hängt die Punktdichte an einer Stelle von der Anzahl und Qualität der registrierten Bilder ab, welche diese abbilden. Ein häufig auftretendes Problem ist, dass eine Vielzahl der Bilder nicht registriert werden kann und dadurch große Teile des Objekts fehlen [SF16]. Des Weiteren sorgen reflektive Oberflächen, objektivbedingte Verzerrungen, Sensorrauschen und weitere Faktoren dafür, dass die Punktwolken häufig stark verrauscht sind [WKZ+16].

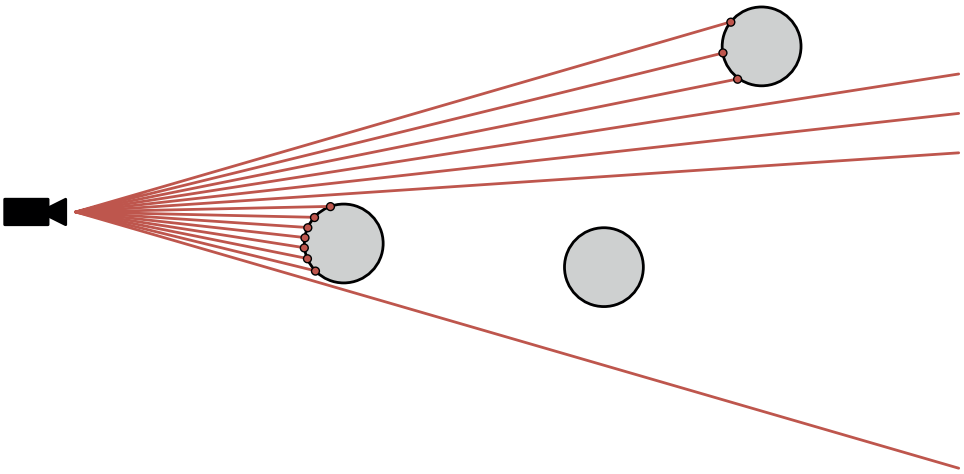


Abbildung 5: Sampling von Objekten mittels Laserscanning aus der Vogelperspektive. Dadurch, dass die Strahlen nach hinten hin auseinanderlaufen werden Objekte, welche sich in der Nähe des Scanners befinden mit einer höheren Auflösung abgetastet als Objekte welche weiter entfernt sind. Objekte, welche von anderen Objekten verschattet werden, werden überhaupt nicht erfasst.

Beim 3D-Laserscanning wird das Objekt erfasst, indem es mit einem Laser abgetastet wird. Dazu wird der Strahl eines im Scanner fest verbauten Lasers durch zwei oszillierende Spiegel abgelenkt und die vom Strahl bis zu dessen Kollision mit dem Objekt zurückgelegte Strecke gemessen [ST18]. Die Distanzmessung erfolgt entweder durch die Laufzeitmessung eines Pulslasers oder die Messung der Phasenverschiebung eines Dauerstrichlasers. Unter Beachtung der Spiegelausrichtung und der vom Laser zurückgelegten Strecke lässt sich die 3D-Position des abgetasteten Punktes berechnen. Des Weiteren verfügen Laserscanner häufig über eine Farbkamera, welche parallel zur Laserabtastung ein Farbbild des Objekts aufnimmt und den abgetasteten Punkten entsprechende Farbwerte zuweist. Laserscanning hat deutlich höhere Anschaffungskosten als Photogrammetrie, liefert i. d. R. jedoch eine höhere Präzision [WKZ+16]. Die generierten Punktwolken weisen dennoch häufig mehrere Probleme auf (Abbildung

5). Die Abtastrate von Objekten ist abhängig von ihrer Entfernung zum Scanner und nimmt nach hinten hin ab. Ein weiteres Problem, welches vermehrt beim terrestrischen Scanning auftritt, sind fehlende Daten durch Verschattungen. Wird ein Objekt von einem anderen Objekt verdeckt kann dieses nicht abgetastet werden. In der Punktwolke können dadurch große Bereiche entstehen, welche keine Punkte beinhalten. Wie auch bei der Photogrammetrie sind die entstehenden Punktwolken aufgrund von diversen Faktoren häufig stark verrauscht [JBPA17].

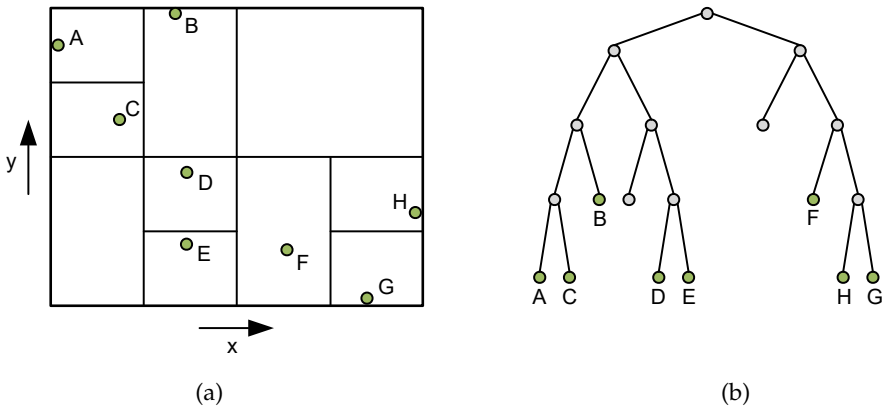
2.3 Out-of-Core-Algorithmen und Datenstrukturen

Die Generierung von Punktwolken resultiert häufig in gigantischen Datensätzen. Gigantisch beschreibt in diesem Kontext Punktwolken, deren Speicheranforderungen die verfügbaren Arbeits- und Grafikspeicherkapazitäten überschreiten. Zur Handhabung der gesamten Punktwolke sind in diesem Fall Out-of-Core-Algorithmen notwendig. Out-of-Core-Algorithmen verarbeiten Datensätze, indem die Gesamtheit der Daten auf einem externen Massenspeicher (*Out-of-Core*) gelagert wird und Teile der Daten bei Bedarf in den Arbeitsspeicher (*In-Core*) geladen werden. Hierbei muss beachtet werden, dass Zugriff und Datenübertragung bei Massenspeichern wie z. B. Festplatten deutlich mehr Zeit in Anspruch nehmen, als dies beim Arbeitsspeicher der Fall ist. Daher sollten die Zugriffe auf externe Daten so weit wie möglich minimiert werden. Out-of-Core-Algorithmen setzen voraus, dass die Daten durch eine geeignete Datenstruktur aufgeteilt werden. Die Anforderungen an die Datenstruktur hängen dabei vom spezifischen Anwendungsfall ab [Sch14]. Datenstrukturen haben auch außerhalb von Out-of-Core-Algorithmen Vorteile bei der Handhabung von Punktwolken. So können sie etwa den Gebrauch von Visibility-Culling ermöglichen, wodurch nicht sichtbare Teile der Punktwolke vor dem Rendern verworfen werden können. Besonders bei großen Datensätzen, welche bei Punktwolken üblich sind, kann die eingesparte Geometrie positive Auswirkungen auf die Renderzeit haben. Des Weiteren können Datenstrukturen genutzt werden, um die Punktwolke in unterschiedlichen Detailstufen zu speichern. Im Folgenden werden zwei Datenstrukturen vorgestellt, welche Grundlage für einige in dieser Arbeit behandelte Verfahren sind.

2.3.1 K-D-Tree

Beim K-d-tree handelt es sich um einen binären Baum, welcher eine Datenmenge im k-dimensionalen Raum aufteilt. Die Aufteilung geschieht dabei jeweils durch Hyperebenen, welche entlang der Koordinatensystemachsen ausgerichtet sind. Die spezifische Wahl der Hyperebenen ist dabei variabel [Sam06]. So ist etwa eine mögliche Strategie den Raum wiederholt zu halbieren und dabei durch die Koordinatenachsen in einer festgelegten Rei-

henfolge zu rotieren. Eine weitere Strategie ist es, den Raum immer entlang der längsten Achse zu halbieren (Abbildung 6). Ein K-d-tree teilt den Raum i.d.R. so lange auf, bis seine Blattknoten einzelne Punkte beinhalten. Alternativ ist es jedoch möglich den Raum nur zu teilen, wenn dieser mehr als eine festgelegte Anzahl an Punkten beinhaltet [Sam06]. In diesem Fall können Blattknoten mehrere Punkte beinhalten.



(a) (b)

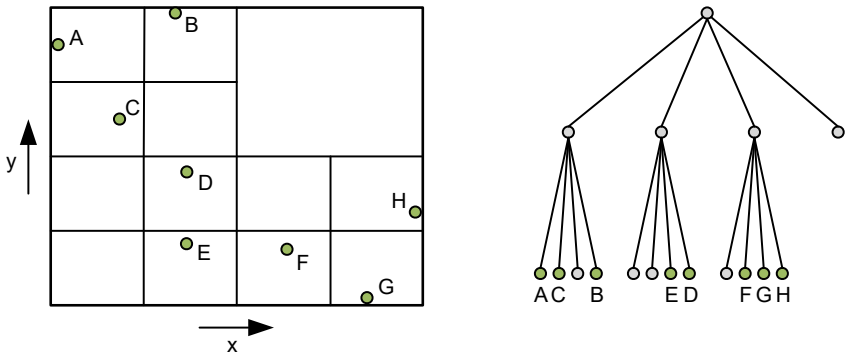
Abbildung 6: (a) Aufteilung einer Punktemenge durch einen K-d-tree im 2D-Raum. Die Teilräume werden immer entlang der längsten Achse halbiert. (b) Der resultierende binäre Baum.

2.3.2 Octree

Beim Octree handelt es sich um eine Datenstruktur, welche einen Datensatz im dreidimensionalen Raum rekursiv in jeweils acht Teilräume unterteilt. Die Knoten des resultierenden Baums haben daher entweder keine oder acht Kinder. Eine Teilung in unterschiedlich große Teilräume ist möglich, i.d.R. wird jedoch eine gleichmäßige Teilung durchgeführt (Abbildung 7). Wie beim K-d-tree kann eine Teilung des Raums entweder so lange fortgeführt werden bis die Blattknoten einzelne Punkte beinhalten oder nur, wenn der Raum eine festgelegte Anzahl an Punkten überschreitet [Sam06].

2.4 Splatting

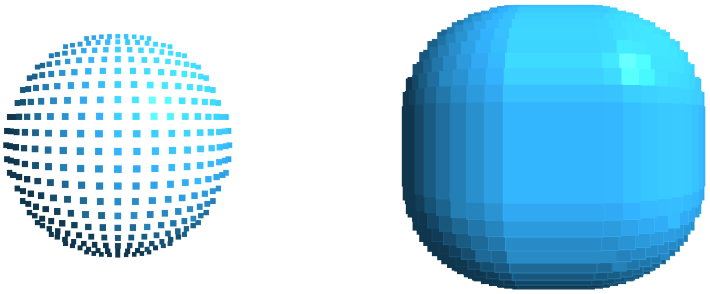
Da Punkte lediglich eine Position im Raum definieren und über keinerlei Nachbarschaftsinformation verfügen, können diese grundsätzlich keine Flächen definieren. Splatting ist ein Verfahren, welches trotz dessen die flächige Darstellung von Punktwolken ermöglicht. Dabei werden Punkte durch zweidimensionale geometrische Figuren (*Splats*) mit einer festgelegten Größe dargestellt. Der Name leitet sich von der dahinterstehenden Idee ab, Farbkugeln auf ein Objekt zu werfen, welche beim Aufprall zerplatzen und Farbflecken auf der Oberfläche hinterlassen. Die Form und Größe der



(a) (b)

Abbildung 7: (a) Aufteilung einer Punktemenge durch einen Quadtree, welcher im zweidimensionalen Raum analog zum Octree arbeitet. Die Teilräume werden immer in vier gleichmäßige Teile unterteilt. (b) Der resultierende Baum.

Splats lässt sich beliebig festlegen und hat starke Auswirkungen auf die Bildqualität und Renderzeit. In OpenGL werden `GL_POINTS` Punktprimitive als bildschirmausgerichtete Quadrate mit einer festgelegten Pixelgröße gerendert. Dadurch lässt sich Splatting trivial umsetzen. Jedoch haben Punktdichte und Kameraposition dabei starke Auswirkungen auf die visuelle Qualität. Um unabhängig davon eine konstante visuelle Qualität zu gewährleisten, ist es notwendig Form und Größe entsprechend zu steuern.



(a) (b)

Abbildung 8: Auswirkung der Splat-Größe auf die Bildqualität. (a) Eine zu kleine Splat-Größe sorgt für Löcher in der Oberfläche des Objekts. (b) Eine zu große Splat-Größe führt zu einer Überzeichnung und Aufblähung des Objekts.

2.4.1 Größe

Die Größe der Splats hat beim Splatting die größte Auswirkung auf die Darstellungsqualität. Ist diese zu klein gewählt, entstehen Löcher in Ober-

flächen von gerenderten Objekten. Ist sie dagegen zu groß gewählt, werden Objekte überzeichnet, wodurch ihre Form verfälscht wird (Abbildung 8). Dadurch dass die Splat-Größe eines Punktes in OpenGL im Screen-Space festgelegt wird, ist diese unabhängig von Kamera und perspektivischer Projektion. Ist jedem Splat eine feste Größe zugeordnet kommt es bei einer interaktiven Betrachtung daher unweigerlich zu Artefakten. In den meisten Fällen wird die Splat-Größe eines Punktes aus diesem Grund in Weltkoordinaten festgelegt und im Vertex-Shader auf Bildkoordinaten projiziert. Abbildung 9 zeigt die Projektion von Welt- in Bildkoordinaten. Zunächst wird das View-Frustum perspektivisch auf den Koordinatenbereich $[-1,1]^3$ abgebildet. Dieser wird durch eine parallele Projektion auf den 2D Koordinatenbereich $[-1,1]^2$ überführt, welcher anschließend auf Bildkoordinaten abgebildet wird [BHZK05]. Die Projektion eines Punktes hängt neben den Projektionsparametern n,f,t,b,h von seinem Weltradius und seiner Position ab. Da die Berechnung der exakten Projektion aufwändig ist und die dadurch resultierende Genauigkeit keine signifikanten Vorteile bietet, wird die Projektion vereinfacht, indem die X und Y Koordinaten des Punktes bei der Berechnung vernachlässigt werden [CN01]. In diesem Fall lässt sich die Pixelgröße eines Splats $size$ anhand seines Weltradius r_{world} und seiner Entfernung zur Kamera $depth$ durch die folgende Gleichung bestimmen:

$$size = 2r_{world} \cdot \frac{n}{depth} \cdot \frac{h}{t-b}. \tag{1}$$

Die berechnete Größe lässt sich im Vertex-Shader über die Built-In-Variable `gl_PointSize` festlegen und resultiert in einem Quadrat mit einer Kantenlänge von $size$ Pixeln.

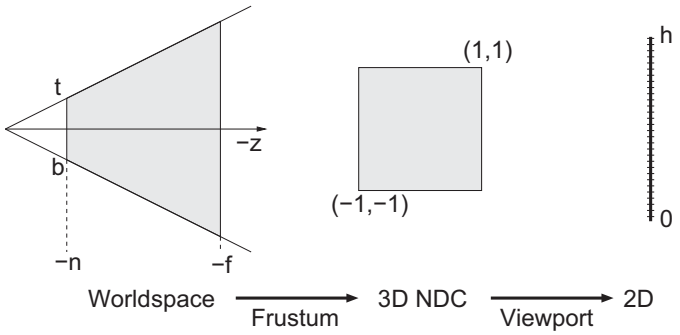
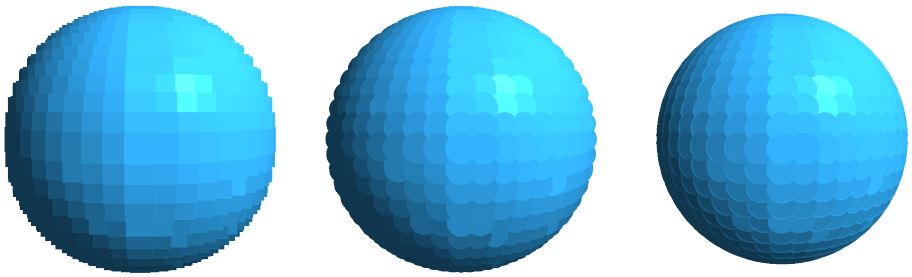


Abbildung 9: Projektion von Welt- in Bildkoordinaten. Das View-Frustum ist durch die Distanzen der Clipping Ebenen zur Kamera n, f und den Öffnungswinkeln t, b definiert. Der Parameter h gibt die Höhe des Viewports in Pixeln an. Abbildung aus [BHZK05].



(a) (b) (c)

Abbildung 10: Auswirkung der Splat-Form auf die Bildqualität. (a) Quadrate überzeichnen Objekte und sorgen durch ihre Kanten für starke Aliasingartefakte an Silhouettenkanten, Texturen und beleuchteten Oberflächen. (b) Scheiben überzeichnen Objekte ebenfalls, produzieren aber weniger Aliasingartefakte als Quadrate. (c) Normalenausgerichtete Scheiben bzw. Ellipsen weisen kaum Überzeichnung an Silhouettenkanten auf. Die Aliasingartefakte sind auf einem ähnlichen Niveau wie bei Scheiben.

2.4.2 Form

Da Punkteprimitive in OpenGL als bildschirmausgerichtete Quadrate gerendert werden, ist das Quadrat die einfachste Form, um Splats darzustellen. Durch ihre Ausrichtung neigen diese jedoch zum Überzeichnen an Silhouettenkanten, wodurch die Form des abgebildeten Objekts verfälscht wird. Aufgrund der kantigen Form entstehen an Silhouettenkanten, Texturen und beleuchteten Oberflächen zusätzlich Aliasingartefakte (Abbildung 10a). Aufgrund dessen werden i. d. R. Scheiben anstelle von Quadraten beim Splatting verwendet. OpenGL bietet nativ nicht die Möglichkeit Punktprimitive als Scheiben zu rendern. Diese lassen sich dennoch mit wenig Aufwand aus den quadratischen Splats erzeugen. Dazu werden im Fragment-Shader alle Fragmente, welche außerhalb des Kreisradius liegen verworfen. Die lokale Position des aktuellen Fragments kann aus der Built-In-Variable `gl_PointCoord` gelesen werden und liegt im Bereich $[0, 1]^2$. Durch die runde Form erzeugen Scheiben weniger Aliasingartefakte als Quadrate. Da sie trotzdem bildschirmausgerichtet sind, werden Objekte an ihren Silhouettenkanten dennoch überzeichnet dargestellt (Abbildung 10b). Um dem entgegenzuwirken, ist es möglich die Scheiben anhand der Oberflächennormale an ihrem Mittelpunkt auszurichten. Dies sorgt effektiv für elliptische Splats (Abbildung 10c). Diese lassen sich ähnlich wie Scheiben aus quadratischen Splats erzeugen, indem entsprechende Fragmente verworfen werden. Um festzustellen welche Fragmente Teil des Splats sind, wird ein Ray-Casting-Verfahren im View-Space angewendet. Dabei wird im Fragment-Shader ein Strahl von der Kamera zur Abbildung des Fragments auf der Near-Plane erzeugt und mit dem Splat geschnit-

ten. Der abbildende Punkt q_n wird bestimmt, indem die Bildkoordinaten des Fragments, welche aus der Built-In-Variable `gl_FragCoord` gelesen werden können, auf die Near-Plane projiziert werden. Sind x, y die Bildkoordinaten des Fragments, n die Distanz der Near-Plane zur Kamera, w_n, h_n die Breite und Höhe der Near-Plane und w_{vp}, h_{vp} die Breite und Höhe des Viewports, lässt sich der Punkt durch die folgende Gleichung bestimmen:

$$q_n = \begin{pmatrix} x \cdot \frac{w_n}{w_{vp}} - \frac{w_n}{2} \\ y \cdot \frac{h_n}{h_{vp}} - \frac{h_n}{2} \\ -n \end{pmatrix}. \tag{2}$$

Anschließend wird ein Strahl von der Kamera, welche im View-Space dem Ursprung entspricht, zum Punkt q_n erzeugt und mit der Splat-Ebene geschnitten, welche durch den Splat-Mittelpunkt c und seine Oberflächennormale n bestimmt ist (Abbildung 11). Der Schnittpunkt q auf der Splat-Ebene lässt sich wie folgt bestimmen:

$$q = q_n + \frac{c \cdot n}{n \cdot q_n}. \tag{3}$$

Indem die Entfernung von q zum Splat-Mittelpunkt c berechnet wird, lässt sich bestimmen, ob das Fragment innerhalb des Kreisradius des Splats liegt oder ob es verworfen werden soll, wodurch sich die elliptische Form des Splats ergibt. Eine Eigenschaft des Verfahrens ist es, dass die Tiefe der normalenausgerichteten Splats nicht korrekt ist. Dadurch dass diese aus bildschirmausgerichteten Quadraten erzeugt wurden, besitzen alle Fragmente die Tiefe des Splat-Mittelpunkts. Für einige Splatting-Verfahren, wie das in Abschnitt 3.1.1 behandelte Splat-Blending, ist es notwendig die Tiefe zu korrigieren. Durch das Ray-Casting ist die Position und somit auch die Tiefe q_z jedes Fragments im View-Space bereits bekannt. Sind n, f die Distanzen der Near- und Far-Plane zur Kamera, lässt sich die View-Space-Tiefe durch die folgende Gleichung in die nichtlineare Skala des Depth-Buffers überführen:

$$depth = \frac{1}{q_z} \cdot \frac{fn}{f-n} + \frac{f}{f-n}. \tag{4}$$

Der berechnete Wert kann durch die Built-In-Variable `gl_FragDepth` in den Depth-Buffer geschrieben werden.

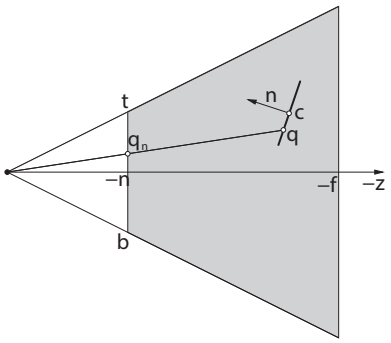


Abbildung 11: Der Punkt q im View-Space, welcher der Position des Fragments auf dem Splat entspricht, wird per Ray-Casting bestimmt. Dazu wird ein Strahl vom Ursprung zur Abbildung des Fragments auf der Near-Plane q_n erzeugt und mit der Splat-Ebene geschnitten. Die Splat-Ebene ist durch den Splat-Mittelpunkt c und dessen Normale n definiert. Abbildung aus [BSK04].

3 Punktrendering-Verfahren

In diesem Abschnitt werden die im Rahmen dieser Arbeit verfolgten Punktrendering-Verfahren vorgestellt und deren Implementierung erläutert. Die Verfahren lassen sich in visuelle Verfahren und Datenstrukturen unterteilen. Visuelle Verfahren befassen sich mit der reinen Darstellung von Punktprimitiven. Datenstrukturen behandeln die echtzeitfähige Darstellung von großen Punktwolken. Die vorgestellten visuellen Verfahren sind unabhängig von den vorgestellten Datenstrukturen. Zwei Verfahren aus unterschiedlichen Kategorien lassen sich daher beliebig kombinieren.

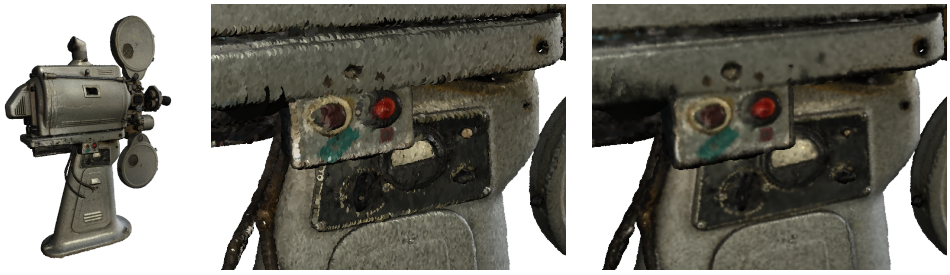
3.1 Visuelle Verfahren

Um beim Splatting eine lochfreie Darstellung von Oberflächen zu gewährleisten, muss die Größe der Splats zwangsläufig so hoch gewählt werden, dass es zu einer Überlappung der Splats kommt. Dies sorgt dafür, dass viele Splats, welche zur selben Oberfläche gehören, sich gegenseitig verdecken. Bei texturierten und beleuchteten Objekten führen diese Überlagerungen zu Aliasingartefakten. Die Artefakte werden auch dadurch verstärkt, dass Punktwolken, wie in Abschnitt 2.2 beschrieben, häufig verrauscht sind. Zusätzlich sorgt die Verdeckung bei bildschirmausgerichteten Splats für einen Flackereffekt, welcher durch Änderung der Kameraperspektive ausgelöst wird, da die Reihenfolge der Splats von dieser abhängig ist. Im Folgenden werden zwei Verfahren behandelt, welche Splats filtern und dadurch Verdeckungsartefakte vermindern.

3.1.1 Splat-Blending

Splat-Blending ist Verfahren zur Splat-Filterung. Dabei wird die Farbe eines Pixels im Bild nicht nur durch den vordersten Splat bestimmt, sondern aus allen Splats zusammengesetzt, welche zur abgebildeten Oberfläche gehören. Dies bewirkt eine Texturenglättung, welche Aliasingartefakte vermindert und die Detailwiedergabe verbessern kann (Abbildung 12). Flackereffekte werden ebenfalls vermieden, da an jedem Oberflächenpunkt alle zugehörigen Splats betrachtet werden und ihre Reihenfolge dadurch keine Relevanz hat. Splat-Blending kann mittels eines Multipass-Verfahrens auf der GPU implementiert werden [BWK02]. Es besteht aus zwei Rendervorgängen, welche auf der Geometrie arbeiten und einem Rendervorgang auf Pixelebene:

Tiefen-Pass. Um festzustellen welche Splats zur selben Oberfläche gehören und dementsprechend zur Farbe dieser beitragen, wird zunächst ein Rendervorgang ausgeführt, bei welchem lediglich in den Depth-Buffer geschrieben wird. Besonderheit ist hierbei, dass ein festgelegter Offset-Wert



(a) (b) (c)

Abbildung 12: (a) Splatting eines Filmprojektors [Nan]. (b) Nahaufnahme ohne Splat-Blending. Durch die gegenseitige Verdeckung der Splats kommt es in der Textur zu Aliasingartefakten. (c) Nahaufnahme mit Splat-Blending. Durch das Blending wird die Textur geglättet.

auf die Tiefenwerte addiert wird. Dieser Wert bestimmt wie nah Splats in ihrer Tiefe zusammenliegen müssen, um zu einer Oberfläche zu zählen. Da der Depth-Buffer nicht linear ist, reicht es nicht aus, den Offset-Wert auf den von OpenGL berechneten Tiefenwert zu addieren. Stattdessen wird der Offset-Wert von der View-Space-Tiefe subtrahiert und anschließend Gleichung 4 verwendet, um den neuen Tiefenwert zu berechnen. Dieser wird über die Built-In-Variable `gl_FragDepth` im Depth-Buffer gespeichert.

(a) (b)

Abbildung 13: Rendering eines Autowracks [Ren]. (a) Ist der Offset-Wert passend gewählt, setzt sich die Farbe der abgebildeten Oberfläche aus allen zugehörigen Splats zusammen. (b) Ein zu hoher Offset-Wert sorgt dafür, dass auch Splats von verdeckten Oberflächen in die Berechnung mit eingehen. Oberflächen werden dadurch transparent dargestellt.

Akkumulierungs-Pass. Im zweiten Durchlauf werden die Farben aller Splats, welche sich auf einer Oberfläche befinden, akkumuliert. Hierbei kommt der im vorherigen Durchlauf beschriebene Depth-Buffer zum Einsatz. Die-

ser wird mittels des OpenGL-Befehls `glDepthMask` auf einen Read-Only-Modus gestellt. Dies sorgt dafür, dass an jedem Pixel alle Fragmente betrachtet werden, welche innerhalb des festgelegten Tiefenbereichs liegen und somit zur abgebildeten Oberfläche gehören. Ist der Offset-Wert zu hoch gewählt, hat dies die Folge, dass Objekte transparent wirken, da auch Fragmente von verdeckten Oberflächen betrachtet werden (Abbildung 13). Die im Fragment-Shader berechneten Farben der Fragmente werden in einer RGBA-Textur akkumuliert. Im RGB-Teil werden die jeweiligen Farbwerte v_f multipliziert mit einer Gewichtung w_f aufaddiert, während die Gewichtungen im Alpha-Kanal summiert werden:

$$rgb = \sum w_f \cdot v_f, \tag{5}$$

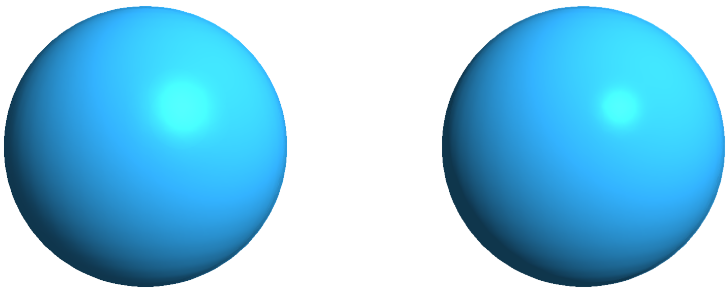
$$a = \sum w_f. \tag{6}$$

Die unterschiedlichen Blending-Funktionen des RGB-Teils und Alpha-Kanals lassen sich durch den OpenGL-Befehl `glBlendFuncSeparate` bestimmen. Die Gewichtung eines Fragments wird durch dessen Distanz zum Splat-Mittelpunkt bestimmt. Dabei wird eine Gaußsche Gewichtungsfunktion verwendet, welche ihren Höhepunkt am Splattmittelpunkt hat und nach außen hin abnimmt. Da die Gewichtung für jedes Fragment bestimmt werden muss, werden die Funktionswerte einmalig auf der CPU vorberechnet und in einem Shader-Storage-Buffer-Object (*SSBO*) auf der GPU gespeichert.

Normalisierungs-Pass. Im finalen Rendervorgang werden die akkumulierten Farbwerte der Pixel normalisiert. Dazu wird ein Screen-Filling-Quad gerendert. An jedem Pixel wird nun der entsprechende RGBA-Wert aus der im vorherigen Durchlauf gefüllten Textur gelesen. Die Normalisierung wird durchgeführt, indem die RGB-Kanäle durch den Alpha-Kanal geteilt werden. Die daraus resultierende Farbe entspricht der finalen Farbe der abgebildeten Oberfläche und wird in den Frame-Buffer geschrieben.

Erweiterung durch Deferred-Shading

Eine hochqualitative Beleuchtung mittels Phong-Shading setzt voraus, dass Positionen, Farben und Normalen zwischen benachbarten Punkten interpoliert werden. Durch die fehlenden Nachbarschaftsinformation ist dies bei Punktwolken jedoch nicht möglich. Dadurch, dass ein Splat nur einen Punkt auf der Oberfläche des Objekts darstellt, besitzen alle seine Fragmente die selben Werte. Es lässt sich daher trivial nur eine Beleuchtung per Flat-Shading erreichen. Wird Flat-Shading in Kombination mit Splat-Blending verwendet, lässt sich eine gouraud-shading-ähnliche Beleuchtung erreichen.



(a) (b)

Abbildung 14: Rendering einer Kugel mit Splat-Blending. (a) Flat-Shading der Splats mit anschließendem Splat-Blending erzeugt eine Beleuchtung, welche Gouraud-Shading gleicht. (b) Splat-Blending in Kombination mit Deferred-Shading ermöglicht eine mit Phong-Shading vergleichbare Beleuchtung.

Um trotz der fehlenden Interpolation hochqualitatives Shading per Phong zu ermöglichen, lässt sich das Multipass-Prinzip des Splat-Blendings durch einen Deferred-Shading-Ansatz erweitern [BHZK05]. Anstatt lediglich die Farben der Splats in einer Textur zu akkumulieren, werden auch Positionen und Normalen in jeweils eigenen Texturen akkumuliert. Der Normalisierungs-Pass wird zu einem Shading-Pass erweitert, bei welchem die gesammelten Attribute zunächst normalisiert werden und anschließend pro Pixel die eigentliche Beleuchtung per Phong berechnet wird. Durch das Blending von Positionen, Farben und Normalen, entstehen ähnliche Ergebnisse wie beim klassischen Phong-Shading. Ein weiterer Vorteil ist, dass die Beleuchtung durch den Deferred-Shading-Ansatz nur einmal pro Pixel berechnet werden muss. Beim einfachen Splat-Blending muss im Falle einer Beleuchtung, diese einmal pro Splat berechnet werden. Bei komplexen Datensätzen mit einer hohen Anzahl von Punkten oder komplexen Beleuchtungsmodellen kann die Kombination von Splat-Blending und Deferred-Shading daher einen signifikanten Leistungsvorteil haben [BHZK05].

3.1.2 Voronoi-Interpolation

Ein weiteres Verfahren zur Filterung von Splats ist es, diese in Form eines Voronoi-Diagramms zu rendern [SW15]. Das Ergebnis gleicht dabei einer Nearest-Neighbor-Interpolation (Abbildung 15). Im Vergleich zu dem, in Abschnitt 3.1.1 behandelten Splat-Blending, lässt sich das Verfahren effizient in nur einem Renderdurchgang auf der GPU umsetzen. Wie bei einem einfachen Splatting werden die Splats in Form von Quadraten, Scheiben oder Ellipsen gerendert. Die einzelnen Fragmente der Splats werden in ihrer Tiefe jedoch so verschoben, dass diese nach hinten hin eine kegelähnliche Form aufweisen. Die gegenseitige Verdeckung der Kegel sorgt ef-

fektiv für die Generierung eines Voronoi-Diagramms (Abbildung 16). Zur Berechnung der Tiefenverschiebung *offset* wird eine paraboloide Gewichtungsfunktion verwendet. Diese ist einfach zu berechnen und sowohl für quadratische als auch runde Splats definiert:

$$\mathit{offset} = 1 - (u^2 + v^2), \quad u, v \in [-1, 1]. \tag{7}$$

Die Parameter u, v entsprechen dabei der parametrisierten Position des Fragments auf dem Splat. Die berechnete Tiefenverschiebung wird anschließend mit dem Weltradius des Punktes multipliziert, sodass diese, unabhängig von der Größe des Splats, die gleichen Ergebnisse erzielt. Der berechnete Offset-Wert wird auf die View-Space-Tiefe des Fragments addiert und durch Gleichung 4 in die nichtlineare Skala des Depth-Buffers überführt. Das Ergebnis wird mittels der Built-In-Variable `gl_FragDepth` in den Depth-Buffer geschrieben.

(a) (b)

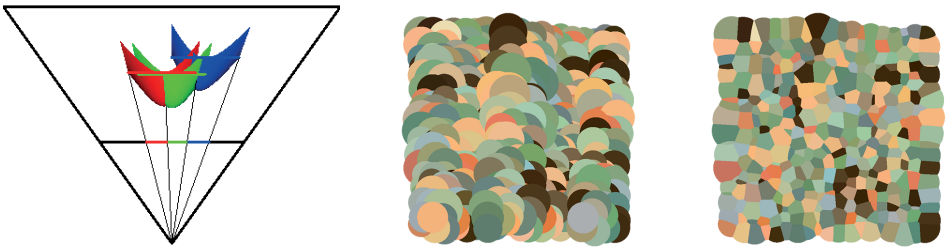
Abbildung 15: Splatting einer texturierten Oberfläche. (a) Bei einem einfachen Splatting mit scheibenförmigen Splats entstehen in kontrastreichen Bereichen der Textur Verdeckungsartefakte, welche die Sichtbarkeit von Details vermindern. (b) Eine Voronoi-Interpolation der Splats verhindert eine gegenseitige Verdeckung und verbessert die Detailwiedergabe.

3.2 Datenstrukturen

Wie in Abschnitt 2.2 erläutert, resultiert die Generierung von Punktwolken häufig in großen Datenmengen, welche eine echtzeitfähige Darstellung dieser verhindern. Im Folgenden werden drei beschleunigende Datenstrukturen vorgestellt, welche Punktwolken unterteilen und diese beim Rendern so traversieren, dass eine echtzeitfähige Betrachtung möglich ist.

3.2.1 QSplat

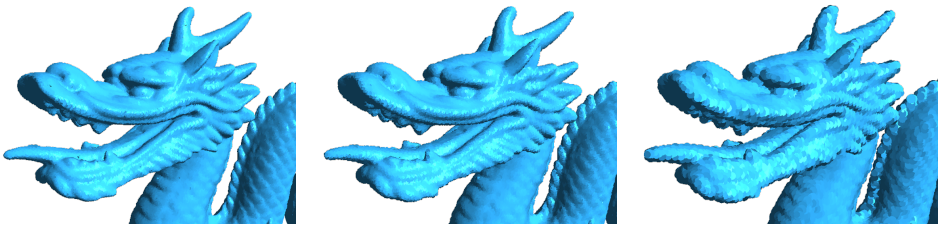
QSplat ist ein Verfahren zur echtzeitfähigen und interaktiven Darstellung von großen Punktwolken [RL00]. Es basiert auf einer Bounding-Sphere-Hierarchie (*BSH*), welche die Punktwolke räumlich aufteilt und benachbarte Punkte zu Bounding-Spheres zusammenfasst. Diese dienen zum einen



(a) (b) (c)

Abbildung 16: Funktionsweise der Voronoi-Interpolation. (a) Splatting mit Voronoi-Interpolation aus der Vogelperspektive. Die Splats werden als Paraboloide gerendert. (b) Das Splatting einer Oberfläche ohne Voronoi-Interpolation erzeugt splatting-typische Verdeckungsartefakte. (c) Paraboloide Splats erzeugen ein Voronoi-Diagramm und gleichen so einer Nearest-Neighbor-Interpolation der Punkte. Abbildung (a) aus [SW15].

als Bounding-Volumes und ermöglichen dadurch Visibility-Culling, zum anderen fungieren sie als neue Render-Primitive, welche eine Vereinfachung der Punktwolke darstellen. Jede Ebene der BSH bildet ein LOD, wobei die originale Punktwolke in den Blattknoten gespeichert ist. Hauptmerkmal von QSplat ist, dass beim Rendern nur Bounding-Spheres gezeichnet werden, deren abgebildete Pixelgröße auf der Bildebene kleiner oder gleich einem festgelegten Cutoff-Wert ist. Bei einem Cutoff-Wert von eins etwa, werden mehrere Punkte, welche auf den selben Pixel abbilden, durch eine einzige Bounding-Sphere repräsentiert. Dies reduziert die Anzahl der zu zeichnenden Punkte, bei gleichbleibender Bildqualität. Cutoff-Werte über eins können verwendet werden um, auf Kosten des LOD, eine höhere Bildrate zu erreichen (Abbildung 17).



(a) (b) (c)

Abbildung 17: Rendering des Stanford Dragon [Sta] mit dem QSplat-Verfahren in unterschiedlichen Detailstufen. (a) Cutoff-Wert von 1 Pixel. (b) Cutoff-Wert von 10 Pixeln. (c) Cutoff-Wert von 20 Pixeln.

Aufbau

Die BSH unterliegt der Struktur eines K-d-tree. So werden die Punkte ausgehend vom Wurzelknoten, welcher alle Punkte enthält, rekursiv entlang der längsten Achse ihrer Bounding-Box geteilt. Um die eigentliche BSH zu erstellen, werden die Punkte jedes Knotens zu jeweils einem neuen Punkt in Form einer Bounding-Sphere zusammenfasst. Eine Bounding-Sphere ist durch eine Position und einen Radius definiert. Die Position berechnet sich aus dem Mittelwert der Positionen aller zusammengefassten Punkte. Der Radius wird so gewählt, dass dieser alle Punkte einschließt. Falls vorhanden kann eine Bounding-Sphere zusätzliche Attribute (Farbe, Normale etc.) beinhalten, welche sich ebenfalls über die jeweiligen Mittelwerte der Punkte bestimmen lassen. Jede Ebene der BSH repräsentiert durch ihre Bounding-Spheres ein LOD der Punktwolke, wobei die Bounding-Spheres der Blattknoten den originalen Punkten der Punktwolke entsprechen. Um die Komplexität der BSH zu verringern und eine schnellere Traversierung zu ermöglichen, wird diese abschließend restrukturiert um einen durchschnittlichen Branching-Faktor von vier zu erhalten.

Rendervorgang

Die BSH wird beim Rendern in Breitensuchenreihenfolge traversiert. Dies ermöglicht im Falle einer Out-of-Core Implementierung die Darstellung der gesamten Punktwolke in einer geringeren LOD Stufe, auch wenn noch nicht alle zu rendernden Bounding-Spheres in den Arbeitsspeicher geladen wurden. Jede traversierte Bounding-Sphere wird zunächst per Frustum-Culling auf Sichtbarkeit geprüft. Ist diese nicht sichtbar, so werden sie und der von ihr ausgehende Teilbaum verworfen. Liegt die gesamte Bounding-Sphere innerhalb des Frustums, so wird dies auch für den restlichen Teilbaum notiert um wiederholte Tests zu vermeiden. Sind Normalen vorhanden, kann zusätzlich Backface-Culling angewendet werden. Fällt der Sichtbarkeitstest positiv aus wird geprüft, ob es sich bei der Bounding-Sphere um einen Blattknoten handelt. Ist dies der Fall, wird dieser gerendert. Handelt es sich nicht um einen Blattknoten, wird anhand des gewählten Cutoff-Werts bestimmt, ob die Bounding-Sphere die gewünschte Detailstufe liefert oder der Teilbaum weiter traversiert werden muss. Dazu wird die Größe der Bounding-Sphere mittels Gleichung 1 auf Bildkoordinaten projiziert. Ist die abgebildete Größe kleiner oder gleich dem Cutoff-Wert, wird die Bounding-Sphere gerendert und der Teilbaum nicht weiter traversiert. Ist die abgebildete Größe größer als der Cutoff-Wert wird die Bounding-Sphere nicht gerendert und der Teilbaum weiter traversiert.

Sequentialisierung

Während das QSplat Verfahren zur Zeit seiner Einführung deutliche Leistungssteigerungen ermöglichte, hat es auf moderner Hardware erhebliche

Nachteile. Durch die stetige Leistungssteigerung von GPUs, können diese immer mehr Punkte, ohne jegliche Datenstrukturen, in Echtzeit rendern. Die BSH im QSplat Verfahren muss dagegen in jedem Frame auf der CPU traversiert werden. Dies ist besonders problematisch, da jeder Knoten der BSH nur einen einzigen Punkt darstellt und diese somit sehr komplex ist. Durch die hierarchische Art der Traversierung lässt sich diese auch nicht ohne weiteres parallelisieren und auf die GPU verlagern. Auf moderner Hardware sorgt der durch die Traversierung auf der CPU entstehende Overhead daher in vielen Fällen für eine schlechtere Leistung, als bei einem direkten Rendern der gesamten Punktwolke. Um eine effiziente Traversierung der BSH auf der GPU zu ermöglichen, lässt sich die hierarchische Datenstruktur sequentialisieren [DVS03]. Die Sequentialisierung hat zwar den Nachteil, dass Visibilty-Culling von Teilbäumen nicht mehr möglich ist, ermöglicht jedoch das Rendern der Punktwolke mit variablem Detailgrad ohne eine Traversierung auf der CPU. Ein weiterer Nachteil der Sequentialisierung ist, dass die gesamte Datenstruktur in den Grafikspeicher geladen werden muss. Die sequentialisierte BSH ist an sich daher nicht zur Darstellung von gigantischen Punktwolken geeignet, welche aufgrund ihrer Speicheranforderungen auf Out-of-Core-Verfahren angewiesen sind. Die sequentialisierte BSH lässt sich jedoch mit anderen Datenstrukturen kombinieren um eine Out-of-Core-Implementierung zu ermöglichen [PSL05, WS06].

Die Grundidee der Sequentialisierung ist es alle Knoten der BSH durch die Shader-Pipeline der GPU zu schicken. Dabei wird im Vertex-Shader für jede einzelne Bounding-Sphere entschieden, ob diese gezeichnet werden soll oder nicht. Dies hängt, wie auch bei der ursprünglichen QSplat Traversierung, von dem gewählten Cutoff-Wert ab. Dazu wird im Vertex-Shader für jede Bounding-Sphere berechnet, welchen Weltradius diese mindestens haben müsste, um einen Splat mit dem Pixelradius des gewählten Cutoff-Werts zu erzeugen. Ist der Radius der Bounding-Sphere größer als der berechnete Radius, so wird diese nicht gerendert, da eine höhere Detailstufe an dieser Stelle erforderlich ist. Ist der Radius der Bounding-Sphere hingegen kleiner, so stellt diese eine ausreichende Detailstufe dar und wird gerendert. Dieser Test alleine reicht jedoch nicht aus, da durch die Sequentialisierung Informationen zu Kinder- bzw. Elternknoten verloren gehen. Es würden also auch alle Bounding-Spheres der tieferen (höher aufgelösten) Ebenen gerendert werden. Daher muss jede Bounding-Sphere neben ihrem eigenen Radius auch den Radius ihres Elternknotens speichern. Dies ermöglicht es im Vertex-Shader einen zweiten Test auszuführen, welcher nur Bounding-Spheres rendert, deren Elternradius größer als der berechnete Radius ist. So werden nur die Bounding-Spheres gerendert, welche für die gewählte Detailstufe notwendig sind.

Ein Nachteil der Sequentialisierung ist, dass alle Knoten der BSH auf der GPU bearbeitet werden müssen. Da das eigentliche Splatting beim Ren-

dern einer Punktwolke den zeitaufwändigsten Schritt darstellt, liefert das Verfahren bei vielen Punktwolken trotz der höheren Datenmenge schnellere Bildraten. Trotzdem lässt sich die Anzahl der zu bearbeitenden Knoten stark verringern, indem diese in einer geeigneten Reihenfolge sortiert werden. Dazu wird die BSH per Breitensuche sequentialisiert, wobei die Kinder jeder Ebene absteigend nach ihrem Elternradius sortiert werden. Dadurch kann in einem einfachen Schritt auf der CPU, bereits ein Großteil der Knoten verworfen werden. Dazu wird der nächste Oberflächenpunkt zur Kamera auf der Bounding-Sphere des Wurzelknotens bestimmt. Für diesen Punkt wird, wie auch auf der GPU, berechnet, welchen Radius ein Splat haben müsste, um im Bild die gewünschte Pixelgröße abzubilden. Bei dem berechneten Wert handelt es sich um den minimalen Radius, welchen eine Bounding-Sphere haben muss, um den gewählten Cutoff-Wert zu erreichen. Es wird anschließend von hinten durch die sortierte Liste iteriert und die Position i des ersten Knotens gespeichert, dessen Elternradius größer als der berechnete Wert ist. Beim eigentlichen Rendern müssen dann nur die Knoten bis zu dieser Position von der GPU bearbeitet werden. Es wird also nur die Liste im Bereich $[0..i]$ übergeben. Da sich im hinteren Teil der Liste die hoch aufgelösten Ebenen befinden und die Anzahl an Knoten von Ebene zu Ebene exponentiell ansteigt, können so in vielen Fällen die meisten nicht zu zeichnenden Knoten im Voraus verworfen werden (Abbildung 18).

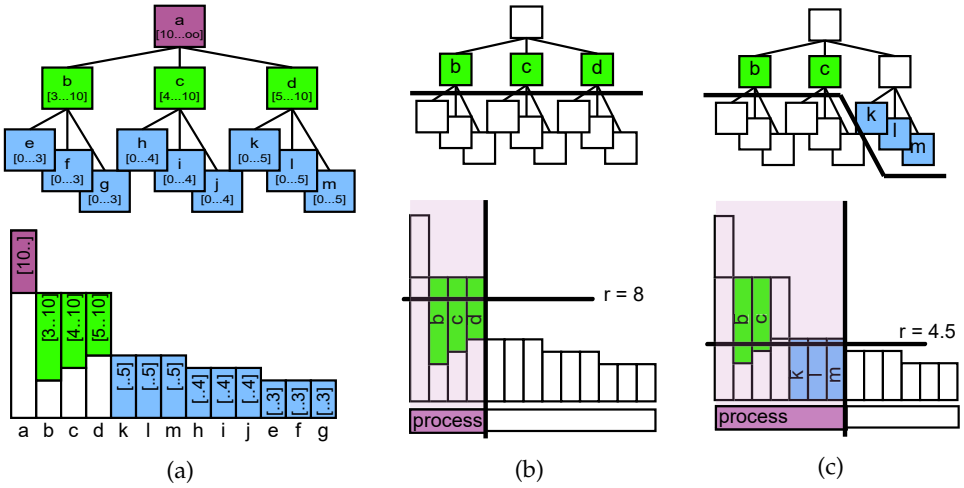


Abbildung 18: Sequentialisierung der BSH. (a) Eine beispielhafte BSH (oben) und ihre sortierte und sequentialisierte Form (unten). Jeder Knoten speichert seinen eigenen Radius und den Radius seines Elternknotens. (b,c) Durch die Sortierung der Liste muss, je nach berechnetem minimalem Radius *r*, nur ein Teil der Knoten auf der GPU bearbeitet werden. Der rechts von der Trennlinie stehende Teil kann in einem Schritt auf der CPU verworfen werden. Abbildung aus [DVS03].

3.2.2 Modifiable-Nested-Octree

Beim Modifiable-Nested-Octree (*MNO*) handelt es sich um eine hierarchische Baumstruktur, welche eine Punktwolke räumlich, rekursiv in jeweils acht Knoten unterteilt [SW11]. Jeder Knoten enthält eine Teilmenge der originalen Punktwolke, wobei jeder Punkt genau einem Knoten zugeordnet wird. Die Vereinigung aller Knoten des MNO stellt also die originale Punktwolke dar. Der MNO hat die Eigenschaft, dass sich die Punktdichte von Ebene zu Ebene verdoppelt. Enthält Ebene i eine maximale Anzahl von n^3 Punkten, so enthält Ebene $i+1$ eine maximale Anzahl von $2n^3$ Punkten. Der MNO erzeugt dadurch mit jeder Ebene, exponentiell höher aufgelöste LOD-Repräsentationen der Punktwolke, welche sich additiv aus den eigenen Punkten und den Punkten der vorherigen Ebenen zusammensetzen. Neben den zugeordneten Punkten speichert jeder Knoten seine Ebene im MNO und eine ID zwischen 0 und 7, welche seine Stelle räumlich in Bezug auf seinen Elternknoten angibt.

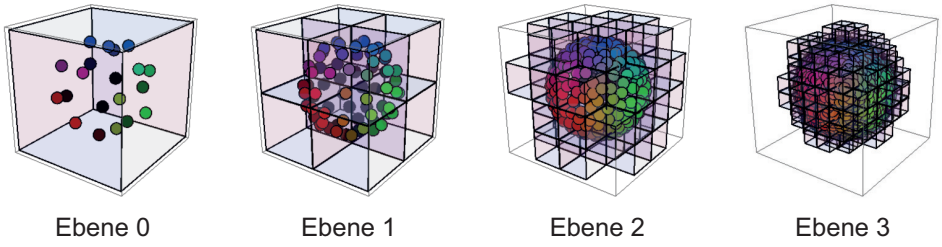


Abbildung 19: Aufbau der MNO Datenstruktur. Jede Ebene verdoppelt die Punktdichte der Knoten, wodurch sich die Detailstufe exponentiell erhöht. Die Vereinigung aller Ebenen entspricht der vollständigen Punktwolke. Abbildung aus [Sch16].

Aufbau

Der Aufbau des MNO beginnt mit der Erstellung des Wurzelknotens, dessen Größe durch die Bounding-Box der originalen Punktwolke bestimmt wird. Jeder Knoten des MNO ist in ein n^3 großes Gitter aufgeteilt. Ausgehend vom Wurzelknoten wird Punkt für Punkt bestimmt, in welche Zelle des Gitters dieser fällt. Die entsprechenden Indizes lassen sich einfach über die Bounding-Box des Knotens und die Auflösung n des Gitters bestimmen. Ist die Zelle leer, so wird der Punkt zu dieser hinzugefügt und der nächste Punkt bearbeitet. Ist die Zelle bereits von einem anderen Punkt belegt, wird bestimmt in welchen der acht Kinderknoten der aktuelle Punkt fällt und ob dieser bereits existiert. Existiert der Kinderknoten noch nicht, wird dieser erstellt. Anschließend beginnt die Zuordnung des Punktes im Gitter des Kinderknotens von neuem. Um wenig befüllte Knoten zu ver-

meiden, kann jedem Knoten zusätzlich ein Array hinzugefügt werden, in welchem Punkte gespeichert werden, welche keinen Platz im Gitter gefunden haben. In diesem Fall wird erst ein neuer Kinderknoten erstellt, wenn das Array eine festgelegte Anzahl an Punkten enthält.

Rendervorgang

Die Knoten des MNO werden, ausgehend vom Wurzelknoten, absteigend anhand der projizierten Pixelgröße ihrer Bounding-Box traversiert. Dies sorgt dafür, dass die Auflösung der Punktwolke nahe der Kamera am höchsten ist und nach hinten hin schrittweise geringer wird. Zunächst wird die Sichtbarkeit des aktuellen Knotens mittels Frustum-Culling überprüft. Ist dieser nicht sichtbar, wird er nicht gerendert und der von ihm ausgehende Teilbaum nicht weiter verfolgt. Zusätzlich wird die projizierte Pixelgröße überprüft. Liegt diese unter einem festgelegten Schwellwert, wird der Knoten ebenfalls verworfen. Andernfalls wird dieser gerendert.

Um selbst bei großen Datensätzen eine echtzeitfähige Darstellung der Punktwolke zu ermöglichen, erlaubt der MNO das Festlegen eines Punktlimits. Sobald die Anzahl der gerenderten Punkte das Limit überschreitet, wird eine weitere Traversierung des MNO gestoppt. So lässt sich indirekt die maximale Detailstufe der Punktwolke begrenzen.

Da die Knoten abhängig von ihrer Ebene im MNO unterschiedliche Punktdichten haben, ist es notwendig die Splat-Größe der Punkte ebenfalls abhängig von der Ebene ihrer Knoten zu wählen. Knoten in den tieferen Ebenen mit einer hohen Auflösung benötigen nur eine kleine Splat-Größe, um Löcher zu vermeiden, während Knoten aus höheren Ebenen mit einer geringen Punktdichte eine größere Splat-Größe erfordern. Da sich die Auflösung von Ebene zu Ebene verdoppelt, halbiert sich die Splat-Größe dementsprechend. Der Splat-Radius der Punkte eines Knotens r_{node} lässt sich abhängig von der Tiefe des Knotens d_{node} und des Splat-Radius des Wurzelknotens r_{root} wie folgt bestimmen:

$$r_{node} = \frac{r_{root}}{2^{d_{node}}}. \tag{8}$$

Da höheraufgelöste LOD-Repräsentationen der Punktwolke additiv aus mehreren Knoten zusammengesetzt werden, ist diese triviale Wahl der Splat-Größe jedoch nicht ausreichend. Punkte aus höheren Ebenen würden Punkte aus tieferen Ebenen durch ihren größeren Splat-Radius verdecken [Sch16]. Zusätzlich können einzelne Knoten im gerenderten Bild Punkte aus unterschiedlichen Detailstufen enthalten. Es ist daher notwendig, die Splat-Größe jedes Punktes dynamisch so zu bestimmen, dass sie der Splat-Größe des höchstaufgelösten, an dieser Stelle gerenderten Knotens entspricht (Abbildung 20). Dazu muss der gerenderte Teil des MNO für jeden Punkt im Vertex-Shader traversiert werden. Um die Traversierung auf der GPU zu ermöglichen, muss der sichtbare Teil des MNO an diese übergeben werden.

Dies geschieht in Form eines Arrays, welches in ein SSBO kopiert wird. Das Array besteht aus 3D-Integer-Vektoren, welche jeweils einen Knoten darstellen und in Breitensuchenreihenfolge gespeichert sind. Der X-Wert kodiert in 8 Bit die gerenderten Kinderknoten. Der Y-Wert gibt den Index im Array an, an welchem der erste gerenderte Kinderknoten kodiert ist. Der Z-Wert gibt die eigene ID an. Zusätzlich zu dem Array wird dem Vertex-Shader noch die Tiefe des MNO, die Ebene des aktuell zu rendernden Knotens und die Bounding-Box des Wurzelknotens übergeben. Mithilfe dieser Informationen lässt sich der sichtbare Teil des MNO auf der GPU traversieren, bis ein Blattknoten erreicht ist. Die Tiefe dieses Blattknotens gibt den Detailgrad an der Stelle des Punktes an und bestimmt seine Größe.

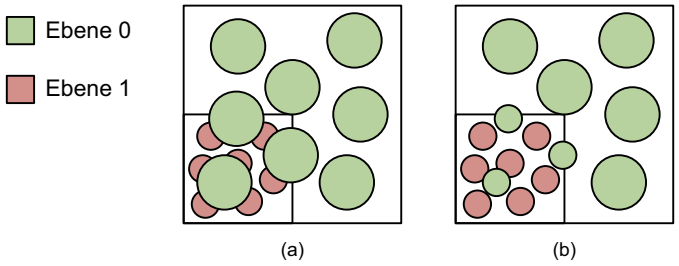


Abbildung 20: (a) Ebenenabhängige feste Splat-Größen. Splats aus Ebene 0 verdecken Splats aus Ebene 1. (b) Dynamische Splat-Größen, welche sich jeweils an den höchsten Detailgrad an den entsprechenden Stellen anpassen. Es kommt nicht mehr zu Verdeckungen durch höhere Ebenen.

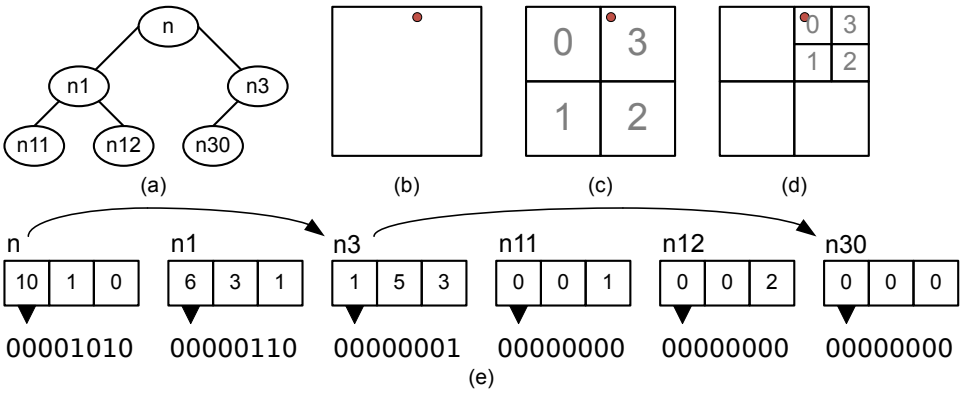


Abbildung 21: Kodierung und Traversierung einer MNO Datenstruktur. (a) Der MNO. (b,c,d) Traversierung des MNO für einen Punkt (rot). (e) Kodierung der Datenstruktur in einem Array bestehend aus 3D Integer Vektoren. Die X-Werte kodieren die gerenderten Kinderknoten und sind in der Abbildung zusätzlich im Binärsystem dargestellt. Abbildung nach [Sch16].

Abbildung 21 zeigt beispielhaft die Traversierung eines MNO auf der GPU im 2D-Fall. Zunächst wird ausgehend vom Wurzelknoten, also dem

ersten Eintrag des Arrays geprüft, in welchem Kinderknoten sich der Punkt befindet. In diesem Fall liegt der Punkt in dem Kinderknoten mit der ID 3. Im X-Wert des Vektors wird nun das Bit an der dritten Stelle überprüft. Da dieses in diesem Fall auf 1 gesetzt ist, wurde der entsprechende Kinderknoten gerendert und die Traversierung wird fortgesetzt. Dazu wird der Y-Wert gelesen, welcher die Stelle des ersten gezeichneten Kinderknotens im Array angibt, in diesem Fall 1. Es wird ab dieser Stelle durch das Array iteriert, bis ein Vektor gefunden wird, welcher im Z-Wert die gesuchte ID 3 gespeichert hat. Bei diesem Vektor handelt es sich um die Kodierung des gesuchten Kinderknotens. Die Traversierung wird nun auf die selbe Art und Weise fortgeführt, bis ein Blattknoten erreicht wird. In diesem Fall endet die Traversierung bei dem Blattknoten n30, welcher auf Ebene 2 liegt. Der Splat-Radius des Punktes lässt sich nun mithilfe von Gleichung 8 berechnen.

Die Berechnung der korrekten Splat-Größe pro Punkt sorgt zwar für einen Overhead im Vertex-Shader, hat aber gleichzeitig den Vorteil, dass insgesamt weniger Splats mit einer großen Splat-Größe gerendert werden. Es werden also weniger Fragmente generiert [Sch16].

3.2.3 Continuous-Level-of-Detail

Ein weiterer Ansatz zur echtzeitfähigen Darstellung von komplexen Punktwolken ist das Continuous-Level-of-Detail-Verfahren (*CLOD*) [SKW19]. Es ermöglicht in Echtzeit das kontinuierliche Reduzieren einer Punktwolke in betrachterabhängige Detailstufen. Das Verfahren arbeitet bei der LOD Berechnung, anders als der in Abschnitt 3.2.2 behandelte MNO, nicht auf Knoten- sondern auf Punktebene. So wird die originale Punktwolke kontinuierlich auf der GPU, unter Beachtung der Kameraposition zu einer neuen Punktwolke vereinfacht. Anders als bei anderen Verfahren sind die Übergänge zwischen den LOD Stufen dabei nicht hart, sondern gehen graduell ineinander über. Des Weiteren wird plötzlich aufploppende Geometrie bei dem Verfahren vermieden. Wie die in Abschnitt 3.2.1 beschriebene sequentielle BSH, erfordert das Verfahren, dass die gesamte Punktwolke auf der GPU gespeichert ist. Für eine Out-of-Core-Implementierung muss das CLOD-Verfahren daher mit anderen Datenstrukturen kombiniert werden [SKW19].

Aufbau

Grundlage für das Verfahren ist eine Unterteilung der originalen Punktwolke in mehrere Detailstufen. Der Aufbau der Detailstufen entspricht der in Abschnitt 3.2.2 vorgestellten MNO-Datenstruktur, mit dem Unterschied, dass keine räumliche Unterteilung der Punktwolke stattfindet. Jede Ebene enthält eine Teilmenge der Punktwolke, wobei sich die Punktdichte von

Ebene zu Ebene verdoppelt. Jeder Punkt ist genau einer Ebene zugeordnet, wodurch die Vereinigung aller Ebenen die originale Punktwolke ergibt. Die CLOD-Datenstruktur lässt sich aus der MNO-Datenstruktur berechnen, indem alle Knoten mit der selben Tiefe zu jeweils einem Knoten bzw. einer Ebene zusammengefasst werden. Nach dem Aufbau der Datenstruktur, werden die Ebenen sequentialisiert und in einen Vertex-Buffer kopiert (Abbildung 22). Dabei speichert jeder Punkt neben Position und anderen optionalen Attributen, zusätzlich welcher Ebene er zugeordnet ist.

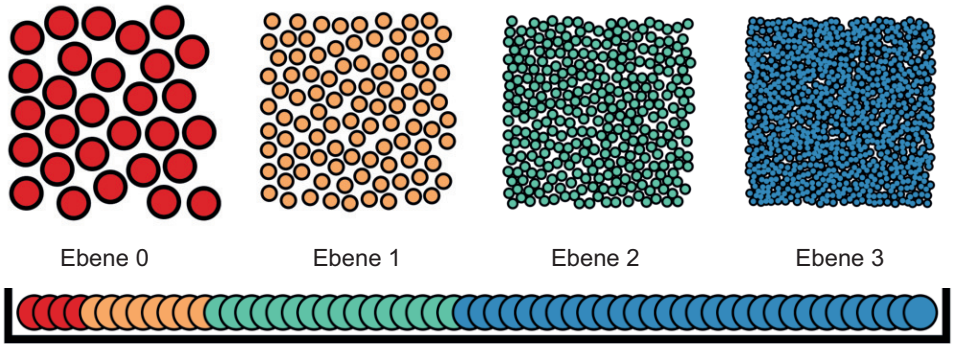


Abbildung 22: Die CLOD-Datenstruktur. Die Punktwolke wird in mehrere Ebenen unterteilt, wobei sich die Auflösung von Ebene zu Ebene verdoppelt. Nach der Erstellung werden die Punkte aller Ebenen sequentiell in einen Vertex-Buffer geschrieben. Abbildung aus [SKW19].

Rendervorgang

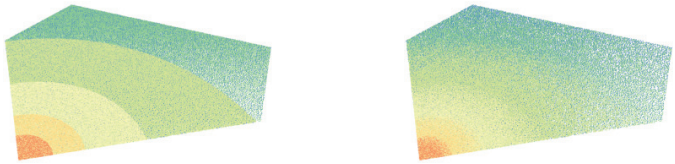
Der Rendervorgang besteht aus zwei Schritten. Im ersten Schritt wird die originale Punktwolke zu einer neuen Punktwolke mit kameraabhängigen und graduell ineinander übergehenden Detailstufen vereinfacht. Im zweiten Schritt wird die vereinfachte Punktwolke gerendert, wobei die Splat-Größen ebenfalls abhängig von den Detailstufen gewählt werden und graduell ineinander übergehen.

Die kontinuierliche Vereinfachung der Punktwolke geschieht auf Punktbasis in einem Compute-Shader, welcher ausgewählte Punkte in einen neuen Vertex-Buffer kopiert. Eine vom Nutzer festgelegte LOD-Distanz zur Kamera $DistanceLOD$ bestimmt dabei über welchen Entfernungsbereich die Ebenen bzw. Detailstufen unterteilt werden sollen. Abhängig davon lässt sich für jede Ebene eine Grenze $MaxDistance_{level}$ bestimmen, welche angibt bis zu welcher Distanz Punkte dieser Ebene gerendert werden sollen:

$$MaxDistance_{level} = \frac{DistanceLOD}{2^{level}}. \tag{9}$$

Für jeden Punkt wird in Abhängigkeit von dessen Ebene die maximal erlaubte Distanz berechnet. Ist die Distanz des Punktes kleiner als die berech-

nete Grenze, wird er in den neuen Vertex-Buffer kopiert, welcher später die vereinfachte Punktwolke beinhaltet. Da Punkte der Wurzelebene 0 die geringste Auflösung darstellen, besitzen sie keine Grenze und werden in jedem Fall in den neuen Vertex-Buffer kopiert. Um diskrete Schritte zwischen den Detailstufen zu vermeiden und einen graduellen Übergang zu erzeugen wird für jeden Punkt ein konstanter Zufallswert im Bereich von 0 bis 1 erzeugt und von seiner Ebene subtrahiert. Der Zufallswert sorgt dafür, dass die Punkte, zu einem jeweils zufälligen Maß, in den Bereich der vorhergehenden Ebene vordringen. Dies bewirkt einen ditheringähnlichen Effekt, welcher die Übergänge zwischen den Ebenen glättet (Abbildung 23). Zusätzlich zum Distanztest werden die Punkte auf Sichtbarkeit getestet und nur kopiert wenn sie im View-Frustum liegen.



(a) (b)

Abbildung 23: Aufteilung der Detailstufen im gerenderten Bild. (a) Durch die disktreten Abstufungen sind Übergänge zwischen Detailstufen sichtbar. (b) Durch die Subtraktion eines Zufallswerts zwischen 0 und 1 von der Ebene der Punkte, wird ein ditheringähnlicher Effekt erzeugt. Abbildung aus [SKW19].

Obwohl die Reduktion der Punktwolke durch die Parallelisierung auf der GPU effizient durchgeführt werden kann, kann diese abhängig von der vorhandenen Rechenleistung und der Größe der Punktwolke einen Leistungsengpass darstellen. In vielen Fällen ist es daher sinnvoll, die Reduktion über mehrere Frames zu verteilen (Abbildung 24). Dazu werden zwei Vertex-Buffer benötigt, welche wiederholt miteinander getauscht werden. Während ein Buffer vom Compute-Shader über mehrere Frames befüllt wird, wird der andere Buffer, welcher im vorherigen Schritt befüllt wurde, gerendert. Sobald der Reduktionschritt abgeschlossen ist, werden die Buffer getauscht und es wird der neu befüllte Buffer gerendert, während der alte Buffer erneut befüllt wird. Das Aufteilen des Reduktionsschritts über mehrere Frames hat die Auswirkung, dass die LOD-Darstellung nicht in jedem Frame aktuell ist. Dies ist bei einer Aufteilung über wenige Frames allerdings nicht erkennbar und der Eindruck einer kontinuierlichen LOD-Berechnung bleibt bestehen [RL00]. Da die View-Matrix im Compute-Shader nicht mehr in jedem Frame aktualisiert wird, ist es notwendig beim Sichtbarkeitstest ein erweitertes View-Frustum zu verwenden, um das fälschliche Filtern von Punkten zu verhindern. Da im Vereinfachungsschritt, in der Theorie, die gesamte originale Punktwolke in die neuen Vertex-Buffer kopiert werden kann, müssen diese entsprechend groß sein. Bei großen

Punktwolken wird i. d. R. jedoch nur eine kleine Teilmenge der originalen Punktewolke kopiert. Um einen übermäßigen Speicherverbrauch auf der GPU zu vermeiden, kann die Größe der Buffer daher begrenzt werden. Dazu wird ein Punktlimit festgelegt, welches angibt, wie viele Punkte maximal kopiert werden dürfen. Ist das Punktlimit höher gesetzt als die in Echtzeit darstellbare Anzahl an Punkten, hat dieses keine negativen Auswirkungen auf das Verfahren.

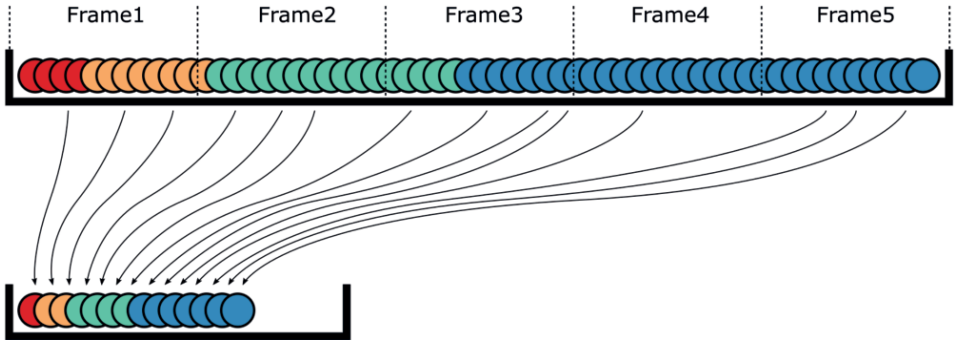


Abbildung 24: Der Reduktionsschrittes kann über mehrere Frames verteilt werden. Bei großen Punktwolken können so höhere Bildraten erreicht werden, während der Eindruck einer kontinuierlichen LOD-Berechnung bestehen bleibt. Abbildung aus [SKW19].

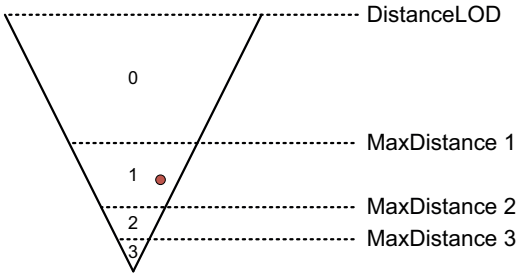


Abbildung 25: Aufteilung der Detailstufen und ihre Grenzen *MaxDistance* abhängig von einer nutzerdefinierten Distanz *DistanceLOD*. Um zu bestimmen welche Detailstufe an der Stelle des Punktes (rot) gerendert wird, wird von oben nach unten über die Ebenen iteriert. Die erste Ebene deren Grenze kleiner ist als die Entfernung des Punktes zur Kamera ist Ebene 2. Die Detailstufe an der Stelle des Punktes entspricht daher Ebene 1.

Nach Abschluss des Reduktionsschrittes werden die Punkte des neuen Vertex-Buffers gerendert. Da dieser komplett auf der GPU erstellt wurde und die CPU keine Informationen über die Anzahl an generierten Punkten hat, wird ein indirektes Rendering mittels des OpenGL-Befehls `glDrawArraysIndirect` durchgeführt. Dies verhindert einen überflüssigen Zwischenschritt, in welchem die Information erst an die CPU und daraufhin

wieder an die GPU geschickt werden. Da Punkte aus unterschiedlichen Ebenen zusammenkommen, um höheraufgelöste Teile der Punktwolke zu bilden, ist es notwendig die Splat-Größe der Punkte dynamisch zu bestimmen. So muss die Größe eines Punktes an die Größe der Punkte der höchstaufgelösten, an dieser Stelle gerenderten Ebene angepasst werden. Um die Ebene zu bestimmen, wird über alle Ebenen iteriert bis eine gefunden wird, deren Grenze kleiner ist als die Distanz des Punktes zur Kamera. Der Punkt liegt dann in der vorhergehenden Ebene und dessen Splat-Größe lässt sich mittels Gleichung 8 bestimmen (Abbildung 25). Um auch hier diskrete Schritte zwischen den Splat-Größen zu vermeiden und einen graduellen Übergang zu erzeugen, werden die Splat-Größen zwischen angrenzenden Detailstufen zusätzlich interpoliert.

4 Evaluation

In diesem Abschnitt werden die in dieser Arbeit vorgestellten Verfahren evaluiert und miteinander verglichen. Da die visuellen Verfahren und die beschleunigenden Datenstrukturen unabhängig voneinander arbeiten, wurden diese getrennt voneinander evaluiert. Im Fall von QSplat wurde die sequentialisierte Form der BSH getestet, welche im Folgenden einfachheitshalber nur als BSH bezeichnet wird.

4.1 Testumgebung

Beim Testsystem handelt es sich um ein Notebook mit einem Intel Core i7-6700HQ Prozessor, 16 GB Arbeitsspeicher und einer Nvidia GeForce GTX 970M Grafikkarte. Der Prozessor verfügt über 4 Kerne mit einer maximalen Taktfrequenz von 3,5 GHz. Die Grafikkarte verfügt über 1280 CUDA Kerne mit einer maximalen Taktfrequenz von 993 MHz und 3 GB GDDR5 Grafikspeicher. Getestet wurde auf Windows 10 Pro 64 bit Version 1909 mit Nvidia Treiberversion 382.05. Alle Tests wurden bei einer Renderauflösung von 1920 x 1080 ausgeführt.

4.2 Visuelle Verfahren

4.2.1 Punktwolken

Zur Auswertung der visuellen Verfahren wurden vier Punktwolken ausgewählt, welche bedingt durch ihre Generierung unterschiedliche Eigenschaften aufweisen:

Modell	Generierung	Punkte	Normalen	Farben
Bunny	Mesh-Sampling	34 834	Ja	Nein
Bagger	Mesh-Sampling	1 687 512	Ja	Ja
Melbourne	Laserscanning	10 803 733	Nein	Ja
Campus Koblenz	Laserscanning	29 314 103	Nein	Ja

Tabelle 1: Eigenschaften der getesteten Punktwolken.

Die Bunny (Abbildung 26a) und Bagger (Abbildung 26b) Punktwolken wurden jeweils durch Abtastung eines Meshs generiert. Als Abtastpunkte wurden die Vertices des Meshs verwendet, wodurch die Abtastrate nicht gleichmäßig ist. Durch die beim Generierungsprozess verfügbaren Nachbarschaftsinformation konnte den Punkten jeweils eine Größe und Normale zugeordnet werden. Dies erlaubt ein lochfreies Splatting mit elliptischen und beleuchteten Splats. Die Bagger Punktwolke verfügt zusätzlich über Farbinformationen. Die Melbourne (Abbildung 26c) Punktwolke wurde durch

luftgestütztes 3D-Laserscanning generiert. Die Abtastrate ist größtenteils gleichmäßig. Die Punkte verfügen weder über Größe noch Normale. Als Splat-Form wurden daher bildschirmausgerichtete Scheiben gewählt. Es wurde eine globale Splat-Größe gewählt, welche die meisten Löcher schließt ohne in einer auffälligen Überzeichnung zu resultieren. Die Campus Koblenz (Abbildung 26d) Punktwolke wurde durch terrestrisches 3D-Laserscanning generiert. Die Abtastrate variiert in Abhängigkeit von der Entfernung zum Scanner stark. Splat-Form und Größe wurden wie bei der Melbourne Punktwolke bestimmt. Aufgrund der unregelmäßigen Abtastrate konnten viele Löcher jedoch nicht geschlossen werden.

(a) (b) (c) (d)

Abbildung 26: Die zur Evaluation der visuellen Verfahren verwendeten Punktwolken. (a) Bunny [Sta]. (b) Bagger [Fov]. (c) Melbourne [Cit]. (d) Campus Koblenz [NPA].

4.2.2 Leistung

Im Folgenden wird die Leistung der Verfahren anhand der erreichten Bildfrequenz in Bildern pro Sekunde (*fps*) bewertet. Als Vergleichswert dient ein einfaches Splatting. Ziel ist eine möglichst geringe negative Auswirkung auf die Bildfrequenz. Tabelle 2 zeigt die Bildfrequenzen beim Rendern der Testmodelle mit einfachem Splatting, Splat-Blending und Voronoi-Interpolation.

Splat-Blending und Voronoi-Interpolation resultieren beide in einer Senkung der Bildfrequenz. Die Auswirkung des Splat-Blendings ist durch das notwendige doppelte Splatting der Szene besonders hoch. Abgesehen von der Bunny Punktwolke, welche eine Senkung von 24% aufweist, liegt eine Senkung von 67-78% vor.

Die Voronoi-Interpolation hat deutlich geringere Auswirkungen auf die Bildfrequenz. Im Fall der Bunny Punktwolke ist keine signifikante Änderung erkennbar. Bei den anderen Punktwolken liegt eine Senkung von 7-32% vor.

Modell	Splatting	Splat-Blending	Voronoi-Interpolation
Bunny	267	202 -24%	266 0%
Bagger	110	24 -78%	102 -7%
Melbourne	41	11 -73%	28 -32%
Campus Koblenz	10	4 -60%	9 -10%

Tabelle 2: Bildfrequenz in fps.

4.2.3 Darstellung

Im Folgenden werden die von den Verfahren gerenderten Bilder gegenübergestellt. Dabei werden die jeweils produzierten Bildartefakte erläutert und die Detailwiedergabe zwischen den Verfahren verglichen.

Abbildung 27a zeigt eine Nahaufnahme der Bunny Punktwolke. Einfaches Splatting erzeugt durch die gegenseitige Verdeckung der Splats deutliche Aliasingartefakte in der Beleuchtung. Des Weiteren sind die einzelnen Splats und ihre elliptische Form erkennbar. An einigen Stellen wird dadurch die Form der Oberfläche verfälscht und erscheint wellig. Splat-Blending resultiert in einer glatten Beleuchtung der Oberfläche und weist keine Aliasingartefakte auf. Bis auf einige Ausnahmen an den Silhouettenkanten, sind die einzelnen Splats nicht erkennbar. Voronoi-Interpolation resultiert in ähnlichen Aliasingartefakten wie einfaches Splatting, verhindert aber eine gegenseitige Verdeckung der Splats. Die elliptische Form der Splats ist dadurch nicht erkennbar und verfälscht die Oberflächenform nicht.

Abbildung 27b zeigt eine Nahaufnahme eines Baggerarms der Bagger Punktwolke. Einfaches Splatting erzeugt deutliche Verdeckungsartefakte. Da es sich bei dem Baggerarm um ein relativ kantiges Objekt handelt, ist dessen Form durch die elliptischen Splats stark verfälscht. Die Aufschrift ist ebenfalls schlecht lesbar. Splat-Blending gibt die Form des Arms gut wieder. Details, wie das am Arm entlanglaufende Rohr, sind erkennbar. Die Aufschrift ist etwas unscharf aber gut lesbar. Voronoi-Interpolation gibt die grundlegende Form des Arms gut wieder, Details sind jedoch kaum erkennbar. Die Aufschrift ist etwas besser lesbar als beim einfachen Splatting.

Abbildung 27c zeigt eine Nahaufnahme einer Fensterfront der Melbourne Punktwolke. Beim einfachen Splatting sind die einzelnen Fenster durch Verdeckungsartefakte nicht erkennbar. Beim Splat-Blending lassen sich die Fenster gut erkennen, die Textur ist aber etwas unscharf. Bei der Voronoi-Interpolation sind die Fenster ebenfalls erkennbar.

Abbildung 27d zeigt eine Nahaufnahme eines Hydrantenschilds der Campus Koblenz Punktwolke. Beim einfachen Splatting ist die feine Schrift des Schilds nicht lesbar. Splat-Blending sorgt für eine geglättete Textur, welche durch ihre Unschärfe nur schwer lesbar ist. Vornoi-Interpolation resul-

tiert in einer gut lesbaren Schrift.

Insgesamt resultiert einfaches Splatting in einer Vielzahl von Bildartefakten, welche hauptsächlich durch die gegenseitige Verdeckung von Splats verursacht werden. In vielen Fällen sorgt die Verdeckung dafür, dass die Form von Objekten an den Silhouettenkanten verfälscht wird. Details in Texturen können durch Verdeckung ebenfalls schwer erkennbar sein. Des Weiteren kommt es zu Aliasingartefakten bei texturierten und beleuchteten Oberflächen. Bei einer interaktiven Betrachtung kommt es, insbesondere bei bildschirmausgerichteten Splats, zu starken Flackereffekten. Splat-Blending verhindert die gegenseitige Verdeckung von Splats und dadurch auch daraus resultierende Artefakte. Silhouettenkanten sind gut erkennbar und Texturen weisen keine Aliasingartefakte auf. Die Glättung der Texturen resultiert jedoch auch in einer Unschärfe, welche feine Details schwerer erkennbar macht. Bei einer interaktiven Betrachtung werden Flackereffekte vollständig verhindert. Voronoi-Interpolation verhindert ebenfalls die Verdeckung von Splats, hat aber keinen glättenden Effekt. Die Form von Objekten wird i. d. R. besser wiedergegeben als beim einfachen Splatting. Details in Texturen werden ebenfalls besser wiedergegeben, weisen jedoch trotzdem Aliasingartefakte auf. Bei einer interaktiven Betrachtung werden Flackereffekte minimiert, aber nicht vollständig verhindert.

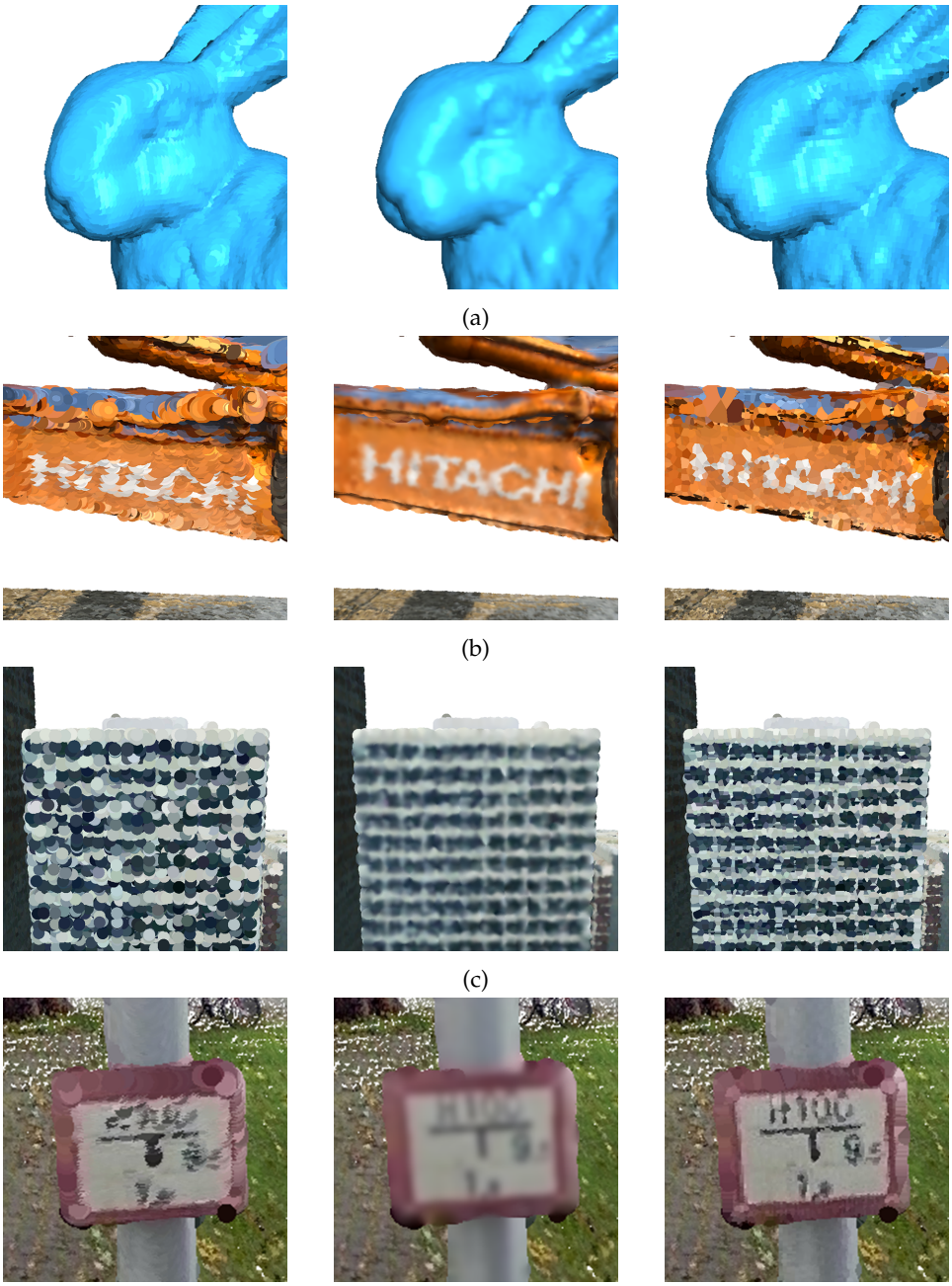


Abbildung 27: Nahaufnahmen der Punktwolken bei einem Splatting mit den getesteten Verfahren. V. l. n. r.: Splatting, Splat-Blending und Voronoi-Interpolation.

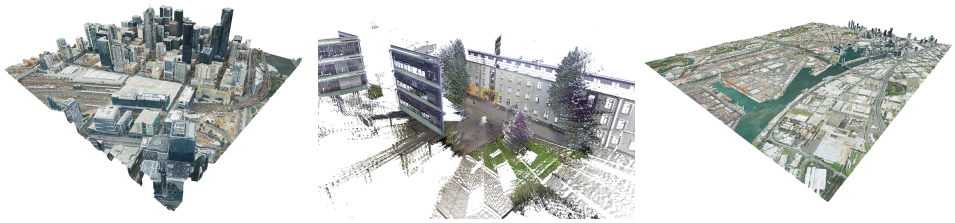
4.3 Datenstrukturen

4.3.1 Punktwolken

Zur Auswertung der Datenstrukturen wurden drei Punktwolken unterschiedlicher Größe ausgewählt (Abbildung 28):

Modell	Punkte	Größe (MB)
Melbourne	10 803 733	173
Campus Koblenz	29 314 103	469
Melbourne Groß	108 486 478	1735

Tabelle 3: Punktzahl und Datenmenge der Punktwolken.



(a) (b) (c)

Abbildung 28: Die zur Evaluation der Datenstrukturen verwendeten Punktwolken. (a) Melbourne [Cit]. (b) Campus Koblenz [NPA]. (c) Melbourne (Groß) [Cit].

4.3.2 Speicheranforderungen

Im Folgenden werden die Grafikspeicheranforderungen der Datenstrukturen bewertet. Vergleichswert ist dabei der Speicherverbrauch der Punktwolken ohne jegliche Datenstrukturen. Tabelle 4 zeigt den von den Punktwolken benötigten Grafikspeicher in Kombination mit den Datenstrukturen.

Die BSH resultiert in einer Steigerung des Speicherverbrauchs von 124% bei der Melbourne Punktwolke und 125% bei der Campus Koblenz Punktwolke. Die Steigerung folgt zum einen aus den zusätzlichen Punkten, welche bei der BSH generiert werden und zum anderen aus der Voraussetzung, dass jeder Punkt einen eigenen Radius und den Radius seines Elternknotens speichert. Die Melbourne (Groß) Punktwolke konnte nicht mit der BSH getestet werden, da die Speicherkapazität des Testsystems überschritten wurde.

Der MNO hat keine signifikante Auswirkung auf den Grafikspeicherverbrauch, da die Datenstruktur größtenteils auf der CPU gespeichert und

traversiert wird. Lediglich zur Bestimmung der Splat-Größe, wird der sichtbare Teilbaum in kodierter Form auf die GPU kopiert, die dabei entstehende Datenmenge ist jedoch unbedeutend.

Das CLOD-Verfahren benötigt bei den Melbourne und Campus Koblenz Punktwolke jeweils 200% zusätzlichen Grafikspeicher. Dies folgt aus den zusätzlich erforderlichen Vertex-Buffern, in welche die zu rendernden Punkte im Vereinfachungsschritt kopiert werden. Bei der Melbourne (Groß) Punktwolke beträgt die Steigerung des Speicherverbrauchs nur 55%, da die Größe der Vertex-Buffer auf jeweils 30000000 Punkte beschränkt wurde. Das Punktlimit ist hoch genug gewählt um keinen Einfluss auf das Verfahren zu haben.

Datenstruktur	Melbourne		Campus Koblenz		Melbourne (Groß)	
BSH	388	+124%	1054	+125%	-	
MNO	173	0%	469	0%	1735	0%
CLOD	519	+200%	1407	+200%	2695	+55%

Tabelle 4: Grafikspeicheranforderungen der Datenstrukturen in MB.

4.3.3 Leistung

Im Folgenden wird die Leistung der Datenstrukturen bewertet. Um einen sinnvollen Vergleich zu gewährleisten, wurde eine minimale Bildfrequenz von 60 fps als gemeinsames Leistungsziel festgelegt und die Anzahl an gerenderten Punkten gemessen. Alle Datenstrukturen bieten Parameter zur Beeinflussung der Bildrate. Die Bildfrequenz lässt sich bei der BSH durch die Wahl des Cutoff-Werts, beim MNO durch die Wahl des Punktlimits und beim CLOD-Verfahren durch die Wahl der LOD-Distanz beeinflussen. Sowohl beim MNO als auch bei der CLOD-Datenstruktur, wurde die Punktwolke in sieben Ebenen bzw. Detailstufen unterteilt. Der Vereinfachungsprozess wurde beim CLOD Verfahren über 5 Frames verteilt. Da die Anzahl an gerenderten Punkten je nach Datenstruktur abhängig von der Betrachterentfernung zur Punktwolke ist, wurden die Punktwolken aus jeweils zwei Kamerapositionen gerendert. Die erste Position zeigt einen Überblick der Punktwolke aus der Entfernung, während die zweite Position eine Detailaufnahme zeigt.

Tabelle 5 zeigt die Anzahl an gerenderten Punkten beim Splatting der Melbourne Punktwolke mit einer minimalen Bildfrequenz von 60 fps. Beim Splatting aus der Überblicksperspektive liegt das CLOD-Verfahren mit 64% gerenderten Punkten deutlich vor den anderen Verfahren. Der MNO steht mit 4% gerenderten Punkte an zweiter Stelle vor der BSH, welche 3% der Punktewolke rendert. Bei einem Splatting aus der Detailperspektive liefert das CLOD-Verfahren mit 99% gerenderten Punkten ebenfalls die beste

Leistung. Die BSH rendert 38% der Punktwolke und liegt vor dem MNO, welcher nur 4% der Punkte rendert. Abbildung 29 zeigt die gerenderten Bilder.

Datenstruktur	Bildfrequenz (fps)	Punkte (Überblick)		Punkte (Detail)	
BSH	≥60	288 393	3%	4 159 238	38%
MNO	≥60	400 675	4%	400 079	4%
CLOD	≥60	6 949 955	64%	10 749 621	99%

Tabelle 5: Leistung der Datenstrukturen beim Splatting der Melbourne Punktwolke.

Tabelle 6 zeigt die Anzahl an gerenderten Punkten beim Splatting der Campus Koblenz Punktwolke. Die BSH kann das Leistungsziel von 60 fps nicht erreichen. Stattdessen liegt die höchste zu erreichende minimale Bildfrequenz bei 49 fps, wodurch ein direkter Vergleich mit den anderen Verfahren nur eingeschränkt möglich ist. Aus der Überblicksperspektive liefert der MNO mit 16% gerenderten Punkten die beste Leistung, gefolgt von dem CLOD Verfahren mit 5% und der BSH mit 4%. Beim Splatting aus der Detailperspektive rendert die BSH mit 59% die meisten Punkte. Das CLOD-Verfahren folgt mit 51% gerenderten Punkten. Der MNO rendert mit 18% die wenigsten Punkte. Abbildung 30 zeigt die gerenderten Bilder.

Datenstruktur	Bildfrequenz (fps)	Punkte (Überblick)		Punkte (Detail)	
BSH	≥49	1 312 852	4%	17 294 123	59%
MNO	≥60	4 654 160	16%	5 213 283	18%
CLOD	≥60	1 517 331	5%	14 894 254	51%

Tabelle 6: Leistung der Datenstrukturen beim Splatting der Campus Koblenz Punktwolke.

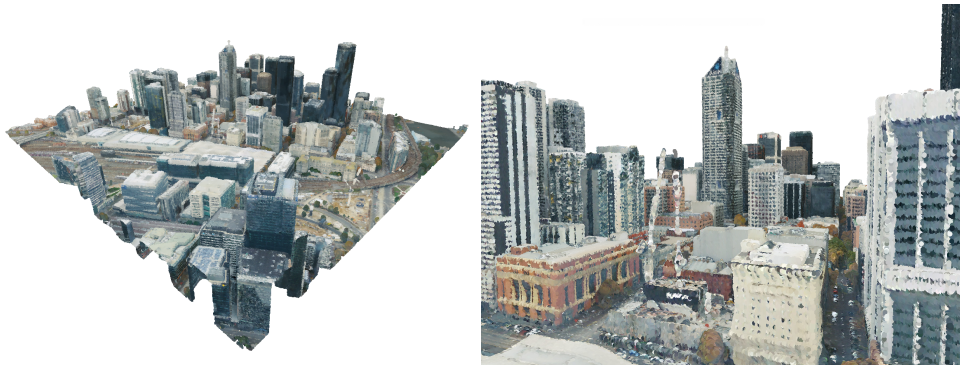
Tabelle 7 zeigt die Anzahl an gerenderten Punkten beim Splatting der Melbourne (Groß) Punktwolke mit einer minimalen Bildfrequenz von 60 fps. Wie in Abschnitt 4.3.2 erläutert, konnte die BSH nicht mit dieser Punktwolke getestet werden. Der MNO rendert aus der Überblicksperspektive 4% der Punktwolke und liegt damit vor dem CLOD Verfahren, welches nur unter 1% der Punktwolke rendert. Aus der Detailperspektive rendert das CLOD Verfahren 9% der Punktwolke und liegt damit vor dem MNO, welcher 4% der Punktwolke rendert. Abbildung 31 zeigt die gerenderten Bilder.

Insgesamt kann festgestellt werden, dass die Leistung der Verfahren in unterschiedlichem Maße von der Größe der Punktwolke und der Kameraperspektive abhängig ist. Die Bildfrequenz der BSH ist ab einer bestimmten Punktwolkengröße begrenzt. Dies liegt an der großen Menge an neuen

Datenstruktur	Bildfrequenz (fps)	Punkte (Überblick)		Punkte (Detail)	
BSH	-	-		-	
MNO	≥60	4 449 347	4%	4 450 533	4%
CLOD	≥60	24 413	<1%	9 720 625	9%

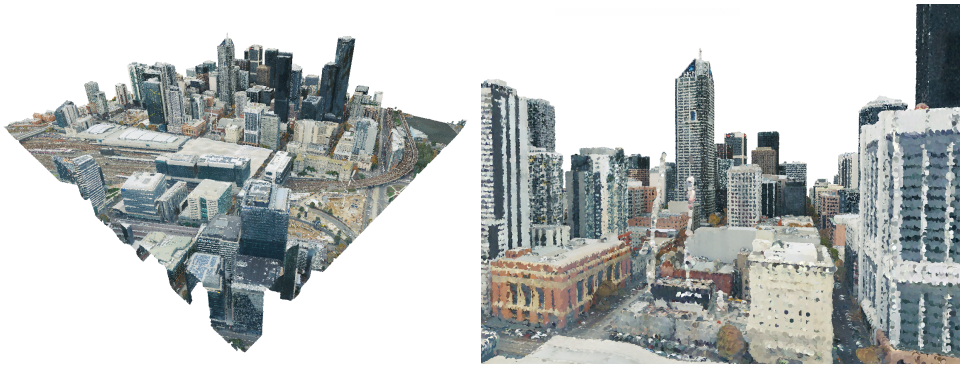
Tabelle 7: Leistung der Datenstrukturen beim Splatting der Melbourne (Groß) Punktwolke.

Punkten, welche beim Aufbau der BSH erstellt werden und zusätzlich zu den bestehenden Punkten in jedem Frame von der GPU bearbeitet werden müssen. Die Anzahl der gerenderten Punkte ist zusätzlich abhängig von der Kameraperspektive, da die zu rendernde Detailstufe in Abhängigkeit von der Entfernung der Punkte zur Kamera bestimmt wird. Die Leistung des MNO ist ebenfalls abhängig von der Größe der Punktwolke. Bei Punktwolken, welche auch ohne beschleunigende Strukturen in Echtzeit darstellbar sind ($\geq 30 fps$), kann die Traversierung des MNO, abhängig von dessen Tiefe, einen Leistungsengpass darstellen. Eine starke Reduktion der zu rendernden Punkte führt in diesem Fall nur zu einer geringen Erhöhung der Bildfrequenz. Die Kameraperspektive hat beim MNO keinen signifikanten Einfluss auf die Anzahl der gerenderten Punkte, da die Detailstufe durch ein Punktlimit begrenzt wird. Auch die Leistung des CLOD-Verfahrens ist abhängig von der Größe der Punktwolke, da alle Punkte im Reduktionschritt auf der GPU bearbeitet werden müssen. Durch die Verteilung des Reduktionsschritts über mehrere Frames lässt sich der Einfluss jedoch vermindern. Da die Detailstufe, wie auch bei der BSH, in Abhängigkeit von der Entfernung der Punkte zur Kamera bestimmt wird, hat die Kameraperspektive ebenfalls Einfluss auf die Zahl der gerenderten Punkte.



(a)

(b)



(c)

Abbildung 29: Splatting der Melbourne Punktwolke mit den getesteten Datenstrukturen. (a) Sequentielle Bounding-Sphere-Hierarchie. (b) Modifiable-Nested-Octree. (c) Continuous-Level-of-Detail.

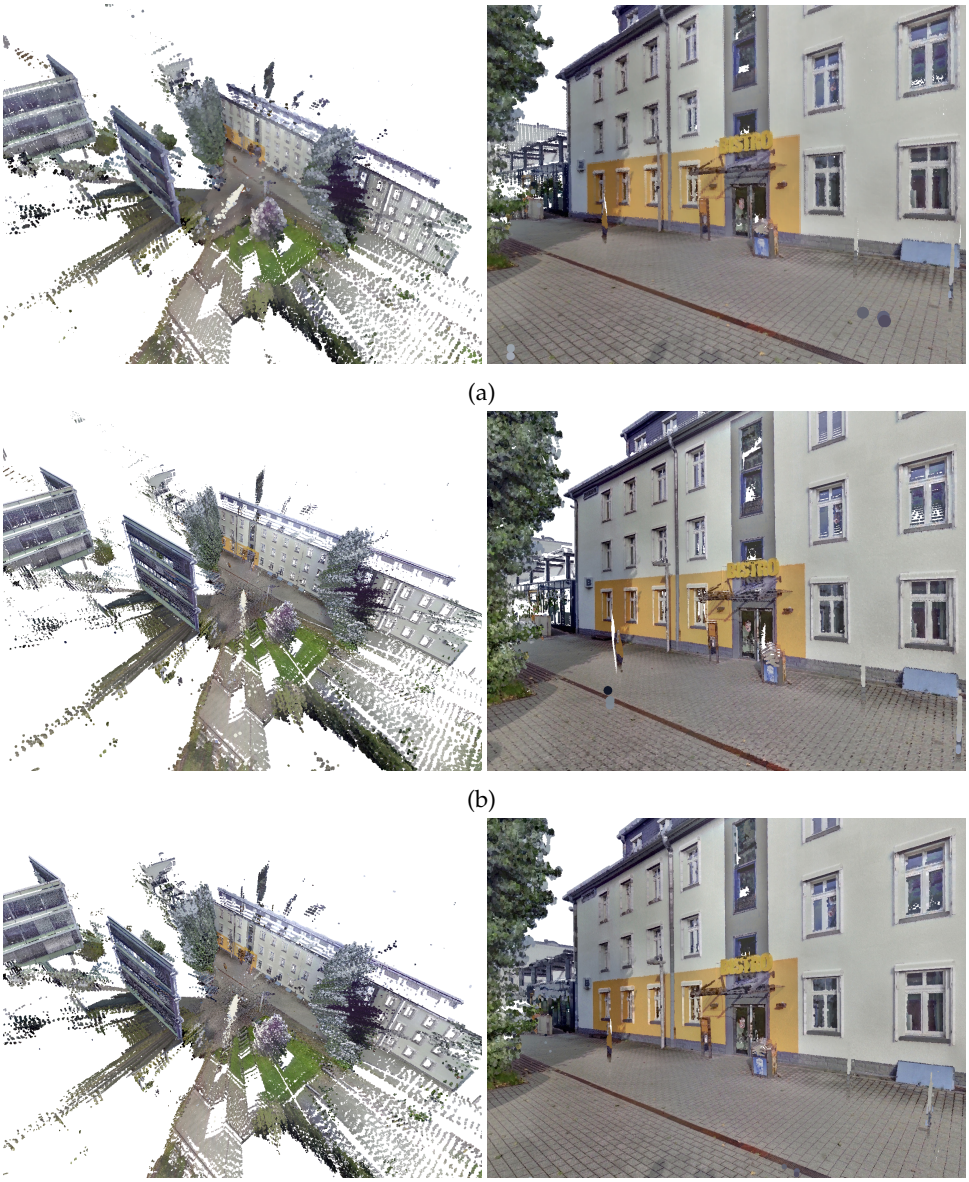


Abbildung 30: Splatting der Campus Koblenz Punktwolke mit den getesteten Datenstrukturen. (a) Bounding-Sphere-Hierarchie. (b) Modifiable-Nested-Octree. (c) Continuous-Level-of-Detail.

(a)

(b)

Abbildung 31: Splatting der Melbourne (Groß) Punktwolke mit den getesteten Datenstrukturen. (a) Modifiable-Nested-Octree. (b) Continuous-Level-of-Detail.

(a) (b) (c)

Abbildung 32: Splatting der Melbourne Punktwolke mit den verschiedenen Datenstrukturen. Alle Verfahren rendern die gleiche Anzahl an Punkten ± 500. (a) Bounding-Sphere-Hierarchie. (b) Modifiable-Nested-Octree. (c) Continuous-Level-of-Detail.

4.3.4 Darstellung

Die Datenstrukturen weisen, abhängig von ihrem Aufbau und ihrer Traversierung, unterschiedliche visuelle Eigenschaften auf, welche im folgenden erläutert werden.

Abbildung 32 zeigt ein Splatting der Melbourne Punktwolke, wobei die Parameter der Datenstrukturen so gewählt wurden, dass alle Verfahren etwa die gleiche Punktzahl rendern. Die BSH weist in Bereichen mit einem niedrigen LOD eine Art Texturenglättung auf, welche Aliasingeffekte vermindert. Grund dafür ist die Art, in welcher die Bounding-Spheres der BSH gebildet werden. Die Farbe einer Bounding-Sphere wird durch den Mittelwert aller eingeschlossenen Punkte bestimmt. Jeder originale Punkt der Punktwolke hat daher Einfluss auf das gerenderte Bild, wodurch der glättende Effekt zustande kommt. Beim MNO und dem CLOD-Verfahren werden Detailstufen dagegen durch Teilmengen der originalen Punktwolke repräsentiert. So kommt es insbesondere bei einem niedrigen LOD zu starken Aliasingartefakten.

Weitere Unterschiede lassen sich in der Verteilung der Detailstufen erkennen. Sowohl die BSH als auch das CLOD-Verfahren bestimmen das zu rendernde LOD auf Punktbasis. Die Verteilung ist daher optimal an die Position der Kamera angepasst. Der MNO bestimmt das zu rendernde LOD dagegen auf Knotenbasis, wodurch die Verteilung der Detailstufen von den Voxeln des Octree abhängig ist. Die unterliegende blockige Datenstruktur kann dadurch sichtbar werden.

Bei einer interaktiven Betrachtung kommt es bei der BSH und dem MNO zu plötzlich aufploppender Geometrie. Dies liegt an der sich sprunghaft ändernden Splat-Größe bei einer Änderung des LOD. Durch die knotenabhängige Verteilung der Detailstufen ist der Effekt beim MNO besonders stark ausgeprägt. Da die Splat-Größe beim CLOD-Verfahren zwischen den Detailstufen interpoliert wird, ist der Übergang zwischen angrenzenden Detailstufen hier weich und nicht erkennbar.

5 Fazit

Die größten Schwierigkeiten beim Punktrendering sind zum einen die fehlenden Nachbarschaftsinformationen, welche eine hochqualitative Darstellung erschweren und zum anderen die bei Punktwolken üblichen, großen Datenmengen, welche eine echtzeitfähige Darstellung verhindern. In dieser Arbeit wurden visuelle und beschleunigende Verfahren zur Darstellung von Punktwolken vorgestellt und miteinander verglichen.

Splat-Blending erlaubt ein hochqualitatives Splatting von Punkten, indem die Farbe von überlappenden Splats gewichtet summiert und anschließend normalisiert wird, um die Farbe der abgebildeten Oberfläche zu bestimmen. Splat-Blending glättet Texturen und verhindert Aliasingartefakte, welche normalerweise beim Splatting auftreten. Splat-Blending lässt sich vollständig auf der GPU umsetzen, erfordert jedoch ein Multipass-Rendering, bei welchem das Splatting der Szene zweifach durchgeführt werden muss. Es wurde eine Senkung der Bildfrequenz von 24-78% gemessen, wodurch das Verfahren nur bedingt zur interaktiven Darstellung von Punktwolken geeignet ist.

Voronoi-Interpolation erzeugt eine Nearest-Neighbor-Interpolation der Splats, indem diese in Form eines Voronoi-Diagramms gerendert werden. Dazu werden die Splats in ihrer Tiefe durch eine paraboloide Funktion verschoben. Das Voronoi-Diagramm wird durch gegenseitige Verdeckung der Paraboloide erzeugt. Voronoi-Interpolation verbessert die Detailwiedergabe von kontrastreichen Texturen, hat jedoch keinen glättenden Effekt. Das Verfahren kann in nur einem Renderpass durchgeführt werden und hat dadurch einen geringeren Einfluss auf die Bildfrequenz als das Splat-Blending. Es wurde eine Senkung der Bildfrequenz von 0-32% gemessen.

QSplat war das erste Verfahren, welches die interaktive Darstellung von großen Punktwolken in einer variablen Detailstufe ermöglichte. Die Punktwolke wird dabei in eine Bounding-Sphere-Hierarchie unterteilt, deren Bounding-Spheres als zusätzliche Punkte dienen, um niedriger aufgelöste Versionen der Punktwolke darzustellen. Die zu rendernde Detailstufe wird vom Nutzer mittels eines Pixelwerts festgelegt. Bei der Traversierung wird für jede Bounding-Sphere bestimmt, ob deren abgebildete Pixelgröße gleich oder kleiner dem festgelegten Wert ist. Ist dies der Fall, wird die weitere Traversierung des Teilbaums gestoppt. Aufgrund der hohen Komplexität der entstehenden Baumstruktur, kann das Verfahren auf moderner Grafikhardware jedoch keine wünschenswerten Ergebnisse erzielen. Da die Traversierung auf der CPU stattfinden muss, stellt diese einen deutlichen Leistungsengpass dar. In dieser Arbeit wurde daher eine Adaption des Verfahrens auf der GPU getestet. Bei diesem wird die BSH sequentialisiert und alle Knoten auf der GPU bearbeitet. Die sequentielle BSH ermöglicht wie das originale QSplat Verfahren eine Darstellung der Punktwolke in variabler Detailstufe. Dadurch, dass die hierarchischen In-

formationen der BSH verloren gehen, kann jedoch kein Visibility-Culling mehr verwendet werden. Des Weiteren muss die gesamte Datenmenge auf der GPU gespeichert werden. Zur Nutzung von Out-of-Core-Algorithmen, muss die sequentielle BSH daher mit anderen Datenstrukturen kombiniert werden. Durch die Generierung von neuen Punkten konnte bei der BSH eine Erhöhung des Grafikspeicherverbrauchs von 124-125% gemessen werden. Bei einer minimalen Bildrate von 60 fps konnte das Verfahren bis zu 4 159 238 Punkte rendern. Ab einer gewissen Punktwolkengröße konnte keine Bildrate von 60 fps erreicht werden, wodurch das Verfahren nicht zur interaktiven Darstellung von großen Punktwolken geeignet ist. Das Verfahren weist einen texturenglättenden Effekt bei der Darstellung von niedrigen Detailstufen auf. Bei einer interaktiven Betrachtung kommt es bei einer Änderung des LOD zu plötzlich aufploppender Geometrie.

Der Modifiable-Nested-Octree ist eine hierarchische Baumstruktur, welche auf einem Octree basiert und die Punktwolke räumlich unterteilt. Jeder Knoten speichert eine Teilmenge der originalen Punktwolke, wobei die Punktdichte von Ebene zu Ebene exponentiell erhöht wird. Der MNO stellt die Punktwolke dadurch von Ebene zu Ebene in immer höheren Detailstufen dar. Die Detailstufen setzten sich dabei additiv aus allen darunterliegenden Ebenen zusammen. Die Knoten des MNO werden in der Reihenfolge ihrer abgebildeten Pixelgröße im Bild traversiert, bis ein nutzerdefiniertes Punktlimit erreicht ist. Der MNO erfordert keinen zusätzlichen Grafikspeicher, da er auf der CPU gespeichert und traversiert wird. Eine Erweiterung durch Out-of-Core Algorithmen ist möglich. Bei einer minimalen Bildrate von 60 fps konnte das Verfahren bis zu 5 213 283 Punkte rendern. Bei einer interaktiven Betrachtung kommt es bei einer Änderung des LOD zu plötzlich aufploppender Geometrie, wobei die Struktur des Octree sichtbar wird.

Das Continuous-Level-of-Detail-Verfahren ermöglicht die interaktive Betrachtung von großen Punktwolken durch eine kontinuierliche Vereinfachung dieser auf der GPU. Die Punktwolke wird dazu wie beim MNO in mehrere Ebenen unterteilt, welche immer höher aufgelöste Detailstufen darstellen. Anders als beim MNO findet jedoch keine räumliche Unterteilung statt. In einem Compute-Shader werden kontinuierlich ausgewählte Punkte in einen neuen Vertex-Buffer kopiert, welcher anschließend gerendert wird. Die Auswahl der Punkte findet in Abhängigkeit von deren zugehöriger Ebene und Entfernung zur Kamera statt. In der Nähe der Kamera werden Punkte aus allen Ebenen ausgewählt, während nach hinten hin nur Punkte aus immer niedrigeren Ebenen ausgewählt werden. Die Anzahl der gerenderten Punkte lässt sich durch einen Entfernungswert beeinflussen, welcher die Verteilung der Detailstufen bestimmt. Da alle Punkte auf der GPU gespeichert werden, können Out-of-Core Algorithmen nur in Kombination mit anderen Datenstrukturen umgesetzt werden. Durch die zusätzlich erforderlichen Vertex-Buffer konnte beim CLOD Verfahren eine Erhö-

hung des Grafikspeicherverbrauchs von 55-200% gemessen werden, wobei der zusätzliche Speicherverbrauch ab einer bestimmten Punktwolkengröße konstant bleibt. Bei einer minimalen Bildfrequenz von 60 fps konnte das Verfahren bis zu 14 894 254 Punkte rendern. Bei einer interaktiven Betrachtung ist ein weicher Übergang bei einer Änderung des LOD gegeben.

Zusammenfassend haben alle Verfahren unterschiedliche Eigenschaften, welche je nach Anforderungen des Nutzers und Beschaffenheit der Punktwolke, Vor- oder Nachteile darstellen können. Eine generelle Rangfolge der Verfahren in Bezug auf Leistung und Darstellungsqualität kann daher nicht gegeben werden.

Literatur

[BHZK05] BOTSCH, Mario ; HORNUNG, Alexander ; ZWICKER, Matthias ; KOBBELT, Leif: High-quality surface splatting on today's GPUs. In: *Proceedings Eurographics/IEEE VGTC Symposium Point-Based Graphics, 2005.* IEEE, 2005, S. 17–141

[BSK04] BOTSCH, Mario ; SPERNAT, Michael ; KOBBELT, Leif: Phong splatting. In: *Proceedings of the First Eurographics conference on Point-Based Graphics*, 2004, S. 25–32

[BWK02] BOTSCH, Mario ; WIRATANAYA, Andreas ; KOBBELT, Leif: Efficient high quality rendering of point sampled geometry. In: *Rendering Techniques* 2002 (2002), S. 13th

[CH02] COCONU, Liviu ; HEGE, Hans-Christian: Hardware-accelerated point-based rendering of complex scenes. In: *Rendering Techniques*, 2002, S. 43–52

[Cit] CITY OF MELBOURNE OPEN DATA TEAM: *City of Melbourne 3D Point Cloud 2018*. `https://data.melbourne.vic.gov.au/City-Council/City-of-Melbourne-3D-Point-Cloud-2018/2dqj-9ydd`, Abruf: 30.10.20

[CN01] CHEN, Baoquan ; NGUYEN, Minh X.: POP: A hybrid point and polygon rendering system for large data. In: *Proceedings Visualization, 2001. VIS'01.* IEEE, 2001, S. 45–540

[DVS03] DACHSBACHER, Carsten ; VOGELGSANG, Christian ; STAMMINGER, Marc: Sequential point trees. In: *ACM Transactions on Graphics (TOG)* 22 (2003), Nr. 3, S. 657–662

[Fov] FOVEA: *Excavator 3d scan RAW*. `https://sketchfab.com/3d-models/excavator-3d-scan-raw-59a1365a0eda4aeab6795fb65a37e01f`, Abruf: 29.10.20. – License: Creative Commons BY-NC 4.0

[JBPA17] JAVAHERI, Alireza ; BRITES, Catarina ; PEREIRA, Fernando ; ASCENSO, João: Subjective and objective quality evaluation of 3D point cloud denoising algorithms. In: *2017 IEEE International Conference on Multimedia & Expo Workshops (ICMEW)* IEEE, 2017, S. 1–6

[Khr] KHRONOS GROUP: *History of OpenGL*. `https://www.khronos.org/opengl/wiki/History_of_OpenGL`, Abruf: 08.11.20

[Lei] LEICA GEOSYSTEMS: *Leica RTC360 3D Laser Scanner*. https://leica-geosystems.com/products/laser-scanners/scanners/leica-rtc360, Abruf: 07.11.20

[LW85] LEVOY, Marc ; WHITTED, Turner: *The use of points as a display primitive*. Citeseer, 1985

[Nan] NANO GAMES: *Cinema Movie Projector AP5 - 1961*. https://sketchfab.com/3d-models/cinema-movie-projector-ap5-1961-6ea63a63e75149fcaea83026451bf479, Abruf: 29.10.20. – License: Creative Commons BY-NC 4.0

[NPA] NEUHAUS, Frank ; PAULUS, Prof. Dr.-Ing. Dietrich ; ARBEITSGRUPPE AKTIVES SEHEN: *Universität Koblenz-Landau - Campus Koblenz*

[PSL05] PAJAROLA, Renato ; SAINZ, Miguel ; LARIO, Roberto: Xsplat: External memory multiresolution point visualization. (2005)

[Ren] RENAFOX: *Abandoned Euro Car (Raw Scan)*. https://sketchfab.com/3d-models/abandoned-euro-car-raw-scan-b288c431e9a54e9b9c4c80ca466231ca, Abruf: 30.10.20. – License: Creative Commons BY 4.0

[RL00] RUSINKIEWICZ, Szymon ; LEVOY, Marc: QSplat: A multiresolution point rendering system for large meshes. In: *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, 2000, S. 343–352

[Sam06] SAMET, Hanan: *Foundations of multidimensional and metric data structures*. Morgan Kaufmann, 2006

[Sch14] SCHEIBLAUER, Claus: *Interactions with gigantic point clouds*, Diss., 2014

[Sch16] SCHÜTZ, Markus: *Potree: Rendering large point clouds in web browsers*, Diplomarbeit, 2016

[SF16] SCHONBERGER, Johannes L. ; FRAHM, Jan-Michael: Structure-From-Motion Revisited. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016

[SKW19] SCHÜTZ, Markus ; KRÖSL, Katharina ; WIMMER, Michael: Real-time continuous level of detail rendering of point clouds. In: *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)* IEEE, 2019, S. 103–110

[ST18] SHAN, Jie ; TOTH, Charles K.: *Topographic laser ranging and scanning: principles and processing*. CRC press, 2018

[Sta] STANFORD UNIVERSITY COMPUTER GRAPHICS LABORATORY: *The Stanford Models*. `http://graphics.stanford.edu/data/3Dscanrep`, Abruf: 30.10.20

[SW11] SCHEIBLAUER, Claus ; WIMMER, Michael: Out-of-core selection and editing of huge point clouds. In: *Computers & Graphics* 35 (2011), Nr. 2, S. 342–351

[SW15] SCHÜTZ, Markus ; WIMMER, Michael: High-quality point-based rendering using fast single-pass interpolation. In: *2015 Digital Heritage* Bd. 1 IEEE, 2015, S. 369–372

[VM10] VOSSELMANN, George ; MAAS, Hans-Gerd: *Airborne and Terrestrial Laser Scanning*. Dunbeath : Whittles Publishing, 2010. – ISBN 978–1904445–87–6

[WKZ+16] WOLFF, Katja ; KIM, Changil ; ZIMMER, Henning ; SCHROERS, Christopher ; BOTSCH, Mario ; SORKINE-HORNUNG, Olga ; SORKINE-HORNUNG, Alexander: Point cloud noise and outlier removal for image-based 3D reconstruction. In: *2016 Fourth International Conference on 3D Vision (3DV)* IEEE, 2016, S. 118–127

[WS06] WIMMER, Michael ; SCHEIBLAUER, Claus: Instant Points: Fast Rendering of Unprocessed Point Clouds. In: *SPBG* Citeseer, 2006, S. 129–136

[XYZ] XYZ RGB INC.: *The XYZ RGB Models*. `http://graphics.stanford.edu/data/3Dscanrep`, Abruf: 30.10.20