Verteilte Oberflächenrekonstruktion aus 3D-Punktwolken

Distributed Surface Reconstruction from 3D Point Clouds

Thomas Wiemann, Isaak Mitschke

Mit dem Las Vegas Reconstruction Toolkit (LVR) der Universität Osnabrück lassen sich aus 3D-Punktwolken Polygonnetze erzeugen. Wenn besonders große Punktwolken polygonalisiert werden sollen, wird dies jedoch problematisch, da die Rekonstruktion sehr rechenintensiv ist und mit zunehmender Punktzahl viel Zeit in Anspruch nimmt. Häufig lassen sich Oberflächen auch gar nicht rekonstruieren, weil die Menge an Eingangsdaten zu groß für den Hauptspeicher eines durchschnittlichen PC ist. In diesem Beitrag wird ein Ansatz gezeigt, mit dem besonders große Punktwolken auf Computerclustern mithilfe des Message Passing Interfaces (MPI) in geeignete Teilpunktwolken zerteilt und rekonstruiert werden können. Das vorgestellte Verfahren wird anhand verschiedener Datensätze nach verschiedenen Kriterien wie Skalierbarkeit und geometrischer Genauigkeit evaluiert.

Schlüsselwörter: 3D-Oberflächenrekonstruktion, 3D-Laserscanning

The Las Vegas Surface Reconstruction Toolkit (LVR) allows to reconstruct polygonal meshes from 3D point clouds. If these point clouds become too large – both in number of points and spatial extension – the reconstruction cannot be done on a single standard PC, as the amount of data becomes too large to fit into RAM. Therefore, we present an approach for distributed surface reconstruction using the Message Passing Interface (MPI). The idea is to subdivide the large input data and process the single chunks on several nodes in a computing cluster. The reconstructions computed on the single nodes are then integrated into a global reconstruction. We evaluate our approach with respect to different criteria like geometric precision and scalability on different example data sets.

Keywords: Surface reconstruction, 3D laser scanning

1 EINLEITUNG

Bei der Vermessung großflächiger Areale mit 3D-Laserscannern fallen enorme Datenmengen in Form von 3D-Punktwolken an. Diese sind beim Einsatz moderner Geräte zwar sehr dicht, trotzdem werden keine mathematisch beschreibbaren Flächen aufgenommen, sondern nur sehr viele Stichproben von den erfassten Oberflächen. Diese Art der Darstellung ist für praktisch relevante Anwendungen wie effizientes Rendering oder Nutzung der Daten als Karten für autonome Fahrzeuge und mobile Roboter sehr ineffizient. Daher geht man dazu über, Oberflächen aus Punktwolken zu rekonstruieren und als Polygonnetze abzuspeichern. Für diese Aufgabe wurde an der Universität Osnabrück eine Open-Source-Software entwickelt: das Las Vegas Surface Reconstruction Toolkit (LVR)[1].

Diese Software wurde im Anwendungskontext der mobilen Robotik entwickelt. Daher wurde bei der Entwicklung besonderes Augenmerk auf eine effiziente Implementierung für moderne Mehrkernprozessoren gelegt. Beim Rekonstruieren von Oberflächen aus

[1] http://www.las-vegas.uni-osnabrueck.de

großvolumigen und hoch aufgelösten Punktwolken kann es dennoch zu Problemen kommen, wenn die Menge der Daten so groß ist, dass sie nicht mehr in den Hauptspeicher passt. Selbst wenn der Hauptspeicher ausreichend groß ist, dauert die Rekonstruktion von Scans mit mehreren hundert Millionen Datenpunkten je nach Ausdehnung der Daten mehrere Stunden. Um große und hoch aufgelöste 3D-Scans zu verarbeiten, wird oft die Anzahl der Punkte reduziert. Dies ist jedoch mit einem Detailverlust verbunden. Auf High-Performance-Computing-Clustern, die aus mehreren hundert Rechnern – sogenannten Knoten – bestehen, kann die Rekonstruktion parallel durchgeführt werden, wodurch sich die Dauer der Rekonstruktion verkürzt. Bisher war es mit dem Las Vegas Reconstruction Toolkit lediglich möglich, Flächennormalen auf solchen Verbundrechnern verteilt zu berechnen /Wiemann et al. 2016/. Allerdings werden dabei alle Datenpunkte im Hauptspeicher eines dedizierten Master-Knotens vorgehalten. In dieser Arbeit wurde das LVR-Toolkit dahingehend erweitert, dass die Daten nicht mehr durchgehend im Hauptspeicher vorhanden sein müssen, sondern räumlich indiziert auf der Festplatte abgelegt werden. So können sie bei Bedarf von den Rechnern im Cluster in den Hauptspeicher der jeweiligen Knoten geladen werden. Diese Teilvolumina können parallel verarbeitet werden, wodurch sich der Rekonstruktionsprozess drastisch beschleunigen lässt. Die von den Knoten berechneten Teilrekonstruktionen werden anschließend in einer konsistenten Gesamtrekonstruktion vereint. Dabei muss darauf geachtet werden, dass keine Redundanzen, wie mehrfaches Vorkommen von Vertices oder von Dreiecksdefinitionen, entstehen.

In diesem Beitrag wird die Datenstruktur zur Verwaltung des von den Punktwolken eingenommenen Volumens vorgestellt, die es erlaubt, eine Marching-Cubes-basierte Rekonstruktion auf mehrere Rechner verteilt vorzunehmen. Es wird die Performanz des implementierten Verfahrens an einem großräumigen und hoch aufgelösten Datensatz evaluiert und gezeigt, dass durch den neuen Ansatz keine geometrischen Inkonsistenzen in den rekonstruierten Dreiecksnetzen entstehen.

2 PROBLEMBESCHREIBUNG

Das Las Vegas Surface Reconstruction Toolkit stellt eine Open-Source-C++-Bibliothek zur Rekonstruktion von Dreiecksnetzen aus 3D-Punktwolken zur Verfügung. Die Software implementiert verschiedene Varianten des bekannten Marching-Cubes-Algorithmus. Für Szenen, die hauptsächlich aus ebenen Flächen existieren, wurden verschiedene Algorithmen zur Meshoptimierung implementiert. Sind die zugrunde liegenden Scans mit Farbinformationen von Kameras eingefärbt, können die Farbwerte in Form von Texturen auf die generierten Dreiecksnetze übertragen werden. Die Software wurde in den vergangenen Jahren erfolgreich in verschiedenen Anwendungskontexten (mobile Robotik, Dokumentation von historischen Gebäuden) eingesetzt, um kompakte und gleichzeitig realistische Umgebungsmodelle zu erzeugen. Eine ausführliche Evaluation bezüglich Genauigkeit und topologischer Konsistenz der erzeugten Meshes hat gezeigt, dass die Software in Umgebungen mit vielen planaren Anteilen deutlich besser arbeitet als andere frei verfügbare Rekonstruktionssoftware /Wiemann et al. 2015/.

Die Rekonstruktion erfolgt dabei im Wesentlichen in drei Schritten: Zunächst wird für alle Punkte eine Flächennormale geschätzt. Anschließend wird über das von den Punkten eingenommene Volumen eine Gitterstruktur mit vorgegebener Gitterkonstante gelegt. Für jeden Punkt des Gitters wird der orientierte Abstand zur gescannten Oberfläche geschätzt. Die Schätzung des Abstands erfolgt durch Projektion auf die nächste Tangentialebene, die durch den nächsten Messpunkt und seine Normale definiert ist. Zeigt die Normale in die Richtung eines Gittereckpunkts, bekommt die Distanz einen positiven Wert, ansonsten einen negativen („Signed Distance Funktion“ (SDF), /Hoppe 1992/). Ausgehend von diesen Distanzwerten und der Konfiguration der Vorzeichen kann nun innerhalb jeder Gitterzelle („Voxel“) der Verlauf der Oberfläche mittels vorberechneter Dreiecksmuster approximiert werden („Marching Cubes“, /Lorensen & Cline 1987/). Die Auflösung des Gitters definiert so die Auflösung des berechneten Dreiecksnetzes.

Eben diese Gitterstruktur, die den wesentlichen Schritten der Rekonstruktion zugrunde liegt, verursacht Probleme, wenn räumlich ausgedehnte Datensätze oder Punktwolken kleinerer Volumen mit sehr hoher Auflösung verarbeitet werden sollen. Mit naiven Ansätzen steigt die Anzahl der Gitterzellen kubisch mit der gewählten Gitterkonstante v. Daher ist es notwendig, Datenstrukturen zu verwenden, die den Speicherbedarf bei der Rekonstruktion reduzieren. Üblicherweise werden dazu Octrees verwendet. In diesen lassen sich benachbarte Zellen allerdings nur schwer identifizieren. Diese müssen für eine topologisch konsistente Rekonstruktion eines Dreiecksnetzes allerdings effizient aufgefunden werden können. Dazu wurde für das Las Vegas Reconstruction Toolkit eine hashbasierte Datenstruktur implementiert. Mit dieser lassen sich topologisch konsistente Rekonstruktionen effizient berechnen. Allerdings war dies bisher nur auf einem einzelnen Rechner möglich. In diesem Beitrag wird beschrieben, wie sich dieses Verfahren erweitern lässt, sodass es möglich wird, großflächige und hoch aufgelöste Rekonstruktionen verteilt auf verschiedenen Rechnern in einem Rechnerverbund zu berechnen. Dabei werden zwei Zielstellungen verfolgt: Erstens soll die Rekonstruktion parallelisiert durchgeführt werden können und zweitens soll es möglich sein, den benötigten Arbeitsspeicher zu begrenzen, sodass auf Rechnern mit wenig Arbeitsspeicher die Rekonstruktion durch Unterteilung der Daten in kleinere Datenpakete dennoch unter Inkaufnahme einer höheren Laufzeit durchgeführt werden kann. In diesem Beitrag wird davon ausgegangen, dass die Rekonstruktion in einem Rechnerverbund in einem lokalen Netzwerk erfolgt. Die Kommunikation der Rechner erfolgt mittels der offenen Message-Passing-Interface-(MPI-) Spezifikation.

Eine wichtige Fragestellung dabei ist, ob sich durch eine Unterteilung des Rekonstruktionsproblems in viele kleine Teilprobleme die Qualität der Rekonstruktion verschlechtert. Dieser Aspekt wird bei der Zusammensetzung des finalen Modells aus den einzelnen Teilrekonstruktionen besonders berücksichtigt und in den Experimenten entsprechend evaluiert.

3 VERTEILTE REKONSTRUKTION MIT VOXELHASHING

3.1 Stand der Forschung

Klassischerweise werden zur Aufteilung des Rekonstruktionsvolumens sogenannte Octrees verwendet. Diese Baumstruktur lässt sich leicht und speichereffizient implementieren /Elseberg et al. 2013/ und zum effizienten Rendering von großen Punktwolken in verschiedenen Auflösungsstufen verwenden. Prinzipiell lassen sich Octrees auch zur Oberflächenrekonstruktion verwenden. Ein Beispiel ist z. B. Poisson-Reconstruction /Kazhdan et al. 2006/, die einen Octree einer zuvor definierten Tiefe verwendet, um das Rekonstruktionsvolumen aufzuteilen. Allerdings ist die dort implementierte Struktur nicht für eine parallelisierte Rekonstruktion ausgelegt. Es muss immer die Repräsentation des gesamten Raums im Speicher gehalten werden, was eine hoch aufgelöste Rekonstruktion von ausgedehnten Gebieten mit diesem Ansatz auf derzeit verfügbarer Hardware unmöglich macht. Ein Octree hat zudem den Nachteil, dass sich die Nachbarn der erzeugten Voxel nur schwer unter Inkaufnahme eines signifikanten Speicher- und Laufzeitoverheads bestimmen lassen /Wiemann et al. 2016/. Dies ist aber notwendig, um die Erzeugung von redundanten Dreiecken zu vermeiden. Wie die Ergebnisse weiter unten zeigen werden, ist die Auffindung von benachbarten Elementen erforderlich, um eine konsistente Rekonstruktion berechnen zu können /Igelbrink et al. 2015/.

Eine Alternative zu Octrees sind Hashing-basierte Ansätze, wie das von /Nießner et al. 2013/ vorgestellte Voxelhashing. Bei diesem Ansatz wird ein Hashingverfahren verwendet, welches eine Hashfunktion der Form

$$H(x, y, z) = (x \cdot p_1 + y \cdot p_2 + z \cdot p_3) \bmod n \tag{1}$$

verwendet, um Punkte von Weltkoordinaten auf Positionen in einem Gitter zu mappen. Dabei sind x, y, z die Koordinaten eines Punkts, p_1, p_2 und p_3 sind große Primzahlen und n ist die Größe der Hashtabelle. Die Verwaltung der Tabelle erfolgt dabei auf der GPU und Kollisionen werden durch verlinkte Listen behandelt. Zwar ist bei diesem Ansatz die Ausdehnung des verwalteten Raums prinzipiell unbegrenzt, die Verwaltung der Punkte erfolgt aber auf der GPU, sodass auch hier die Menge der zu verwaltenden Daten durch den Hauptspeicher der Grafikkarte begrenzt ist. Eine Aufteilung der Daten ist nicht vorgesehen. Zudem können auch bei diesem Ansatz benachbarte Zellen nur schwer aufgefunden werden. Ein weiterer Ansatz wird in /Hoppe & Lefebvre 2006/ vorgestellt. Auch hier wird nicht auf das Verwalten von Nachbarschaftsbeziehungen eingegangen.

Um diese Nachteile auszugleichen, ist ein Ansatz entwickelt worden, der ein virtuelles Raumgitter mittels einer einfachen Hashfunktion verwaltet, die es erlaubt, benachbarte Zellen im Gitter effizient aufzufinden. Die Argumente der Hashfunktion können darüber hinaus verwendet werden, um Teilvolumina eindeutig für eine Aufteilung und anschließende Serialisierung zu definieren. Zudem ermöglicht es die Datenstruktur, in Bereichen, in denen die Punktdichte der Eingangsdaten niedrig ist, mögliche Fehlstellen im Gitter aufzufüllen, um die Qualität der Rekonstruktionen zu verbessern. Dieser Ansatz wird im Folgenden vorgestellt.

3.2 Gitterstruktur zur verteilten Rekonstruktion

Die Idee des hier verwendeten Ansatzes ist, ein virtuelles Raumgitter mit einer vorgegebenen Auflösung v zu erzeugen. Jeder Messpunkt fällt jetzt genau in ein Voxel. Das entsprechende Voxel ist durch drei Indizes i, j und k definiert, die seine Position im 3D-Gitter repräsentieren.

Beispielsweise befindet sich das Voxel mit den Indizes $i = 3$, $j = 2$ und $k = 5$ an dritter Position in x-Richtung, zweiter Position in y-Richtung und fünfter Position in z-Richtung, ausgehend vom Ursprung des jeweiligen Koordinatensystems. Für jeden Messpunkt p lässt sich die Zelle, in der er sich befindet, nach *Abb. 1* bestimmen:

$$i = \left\lfloor \frac{p_x - x_{\min}}{v} \right\rfloor, \quad j = \left\lfloor \frac{p_y - y_{\min}}{v} \right\rfloor, \quad k = \left\lfloor \frac{p_z - z_{\min}}{v} \right\rfloor. \tag{2}$$

Alle Zellen werden in einer Hashmap verwaltet. Diese erlaubt es, auf Zellen in konstanter Zeit zuzugreifen, und speichert nur Zellen, die auch Daten enthalten. Zur Adressierung der Zellen anhand des Index-Tripels wird folgende Hashfunktion verwendet:

$$H(i, j, k) = i \cdot \dim_x + j \cdot \dim_y + k. \tag{3}$$

Die Konstanten $\dim_x$ und $\dim_y$ bezeichnen dabei die Ausdehnung des Raumvolumens in x- bzw. in y-Richtung in Vielfachen der Gitterkonstante:

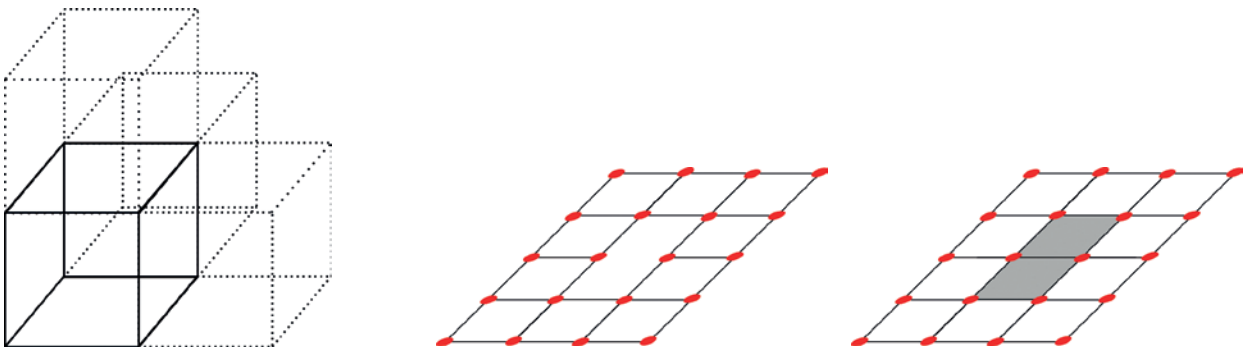

Abb. 1 | Erzeugung von Extrusionszellen zur konsistenten Gitterauffüllung. Wird eine Zelle in die Hashmap eingefügt (schwarz, linkes Bild), werden die grau skizzierten Nachbarzellen ebenfalls generiert. Solche Zusatzzellen helfen, Konfigurationen, wie in der Mitte gezeigt, aufzufüllen (graue Zellen im rechten Bild).

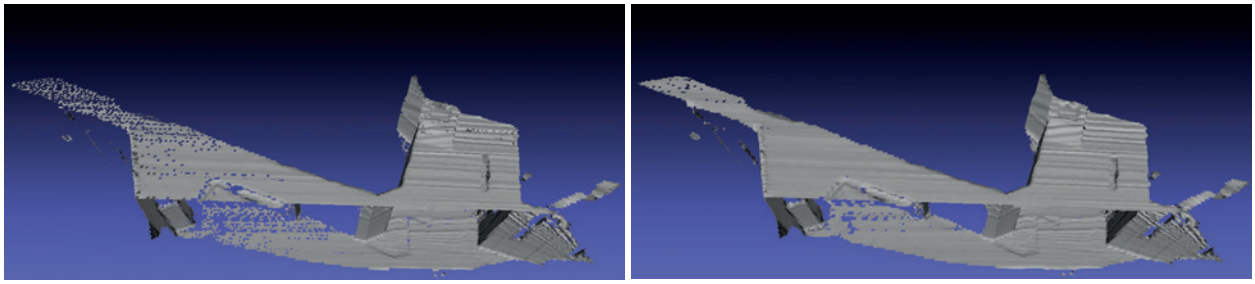

Abb. 2 | Einfluss der extrudierten Zellen auf das Ergebnis der Rekonstruktion. Im linken Bereich sind deutlich die in Abb. 1 skizzierten Fehlstellen in der Rekonstruktion zu erkennen. Durch das Auffüllen dieser Löcher wird eine durchgehende Fläche erzeugt.

$$\dim_x = \left\lfloor \frac{x_{\max} - x_{\min}}{v} \right\rfloor, \quad \dim_y = \left\lfloor \frac{y_{\max} - y_{\min}}{v} \right\rfloor. \tag{4}$$

Für alle Punkte der Punktwolke wird beim Einlesen die Zelle des Gitters bestimmt, in der sie sich befinden. Sollte in der Hashmap bereits eine Zielzelle für einen Punkt vorhanden sein, wird er dort hinzugefügt. Sollte keine Zelle vorhanden sein, wird eine neue Zelle erzeugt und in die Hashmap eingefügt. Auf diese Art und Weise lassen sich alle Punkte bei gegebener Gitterauflösung in eine Zelle einsortieren.

Der entscheidende Vorteil dieser Struktur ist, dass sich benachbarte Zellen in der Hashmap leicht auffinden lassen, indem die Indizes in der Raumrichtung, in der gesucht wird, erhöht oder erniedrigt werden. Aufgrund der durchgehenden Indizierung kann die Punktwolke nun leicht in räumlich zusammenhängende Bereiche unterteilt werden, indem alle Zellen in einem Bereich mit den dazugehörigen Punkten auf der Festplatte gespeichert werden. Diese Teilbereiche können dann bei der Rekonstruktion von den einzelnen Knoten im Cluster geladen und verarbeitet werden. Dadurch, dass zu jedem serialisierten Volumen die dazugehörigen Indexintervalle des globalen Gitters abgespeichert werden, lassen sich die Teilrekonstruktionen später wieder zu einem konsistenten Gesamtmodell zusammenfügen. Näheres dazu in Abschnitt 4.1.

3.3 Konsistentes Auffüllen von Fehlstellen im Gitter

Um Fehlstellen im Gitter auszugleichen, wird für jede Zelle eine weitere Zelle in positiver *i*, *j*, *k*-Richtung generiert, falls dort nicht schon ursprünglich eine existiert. So können Löcher im Gitter, die komplett von anderen Zellen umschlossen sind, konsistent aufgefüllt werden. Solche Fehlstellen können insbesondere in Randbereichen der aufgenommenen Scans, in denen die Punktdichte geringer ist, vorkommen. Diese Zusatzzellen werden im Folgenden als Extrusionszellen bezeichnet. Die Erzeugung solcher Zellen ist schematisch in *Abb. 1* dargestellt. Wird die im linken Teilbild dargestellte schwarze Zelle erzeugt, werden die gestrichelt dargestellten Zellen ebenfalls in das Gitter eingefügt, wenn sie nicht schon bereits erzeugt wurden. Die entsprechenden Positionen im Gitter können leicht über die fortlaufende Indizierung bestimmt und angefragt werden. Sollte eine neue Extrusionszelle angelegt werden, werden die SDF-Werte der Nachbarzellen übernommen. Sollte es sich um eine Randzelle handeln, werden die SDF auf einen ungültigen Wert gesetzt. Solche Zellen erfordern allerdings eine Sonderbehandlung bei der Rekonstruktion, wie in Abschnitt 4.2 beschrieben. Der Einfluss dieser Maßnahme auf die Rekonstruktionsqualität ist in den Beispielbildern in *Abb. 2* illustriert. Im linken Bild sind die in *Abb. 1* skizzierten Fehlstellen deutlich zu erkennen. Das Auffüllen durch Extrusionszellen sorgt für eine deutlich konsistentere Rekonstruktion (rechtes Bild).

4 ABLAUF DER REKONSTRUKTION

4.1 Aufteilung und Zusammenfügen der Teilrekonstruktionen

Dieser Abschnitt präsentiert Details zum eigentlichen Ablauf der verteilten Rekonstruktion. Die Software zur verteilten Rekonstruktion besteht im Wesentlichen aus zwei Komponenten: Einem Master-Knoten im Cluster, der für das Datenmanagement zuständig ist, und einer Menge von Slave-Knoten, die einzelne Teilvolumina rekonstruieren. Die Slave-Knoten berechnen jeweils parallel die Rekonstruktion für ein Teilvolumen, das ihnen vom Master zugeteilt wurde. Alle Teilvolumina werden in einem gemeinsamen Dateisystem abgelegt, auf das sowohl der Master- als auch die Slave-Knoten Zugriff haben.

Der Master lädt zunächst die Punktwolke und bestimmt die räumliche Ausdehnung des zu rekonstruierenden Bereichs, wodurch die relevanten Parameter für die Hashfunktion festgelegt werden. Dies geschieht, ohne die eigentlichen Punkte in den Hauptspeicher zu laden. Wenn die Ausdehnung bekannt ist, wird das Raumvolumen der Punktwolke in Teile vorgegebener Größe zerlegt und mit den dazugehörigen Punkten befüllt, indem diese in entsprechende Dateien auf dem gemeinsamen Dateisystem kopiert werden. Die Größe der Teilvolumina kann so gewählt werden, dass alle relevanten Daten bei der Rekonstruktion in den Arbeitsspeicher der Knoten passen. Dies erlaubt es, beliebig große Volumen zu bearbeiten, da immer nur ein Teil der Daten im Arbeitsspeicher der Knoten vorhanden ist. Derzeit nicht bearbeitete Daten werden immer in eigenen Dateien ausgelagert.

Die so vorsegmentierten Daten werden nach einem bestimmten Schema im gemeinsamen Dateisystem abgelegt. Das Schema dieser Zuteilung ist in *Abb. 3* dargestellt. Im dem dort präsentierten Beispiel werden drei Teilvolumina erstellt und verschiedenen Rechnern zugewiesen. Bei der Aufteilung wird darauf geachtet, dass sich die Teilbereiche jeweils um eine Zellebene überlappen. Dies ist

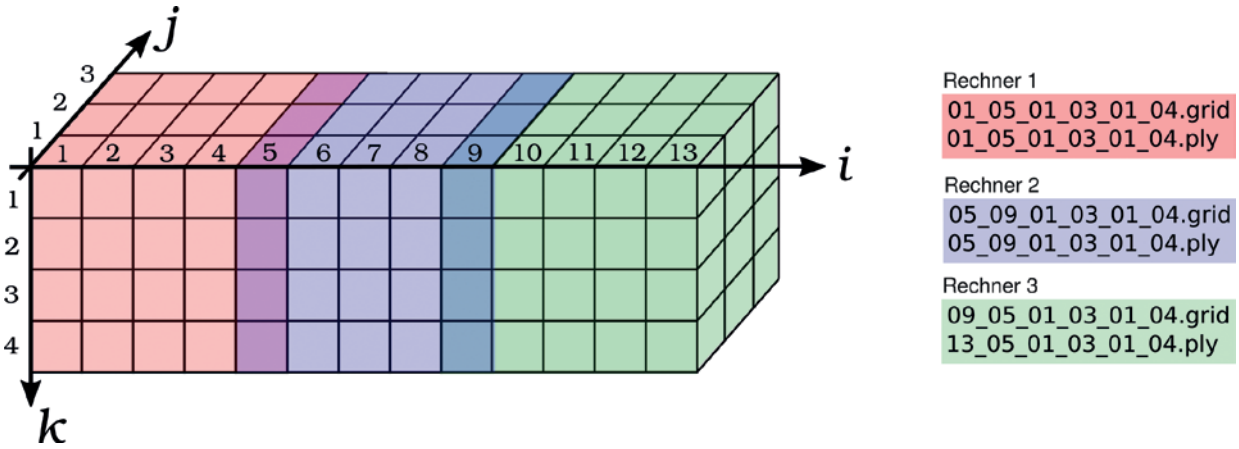

Abb. 3 | Aufteilung des globalen Gitters in sich teilweise überlappende Teilvolumina. Jeder Rechner im Rechnernetz ist für die Rekonstruktion eines ihm vom Master-Knoten zugewiesenen Bereichs zuständig. Die Ergebnisse der Berechnung (Rekonstruktion und Gitterbelegung) werden nach einem bestimmten Namensschema abgespeichert, sodass der Master die Ergebnisse zu einer konsistenten Gesamtrekonstruktion zusammensetzen kann.

notwendig, um Inkonsistenzen bei der Zusammenfügung zu vermeiden. Der blaue Bereich z. B. wird Rechner 2 im Netz zugewiesen. Er berechnet die SDF-Werte und Rekonstruktionen für den Bereich $i = 5, \ldots, 9$; $j = 1, \ldots, 3$ und $k = 1, \ldots, 4$. Entsprechend setzen sich die Dateinamen zusammen. Diese Informationen können bei der Zusammensetzung der Gesamtrekonstruktion im Master-Knoten benutzt werden, um relevante Teilbereiche zu bestimmen. Damit später ein konsistentes Mesh in diesem Bereich berechnet werden kann, muss sichergestellt sein, dass die SDF-Werte in den jeweils überlappenden Teilgittern gleich sind.

Um dies zu erreichen, werden die Distanzwerte an den Außenzellen zweier überlagerter Gitter interpoliert. Dies übernimmt der Master-Knoten, nachdem alle Gitter erzeugt und abgespeichert wurden. Dafür lädt der Master-Knoten nacheinander alle Teilgitter und interpoliert die SDF-Werte an den Eckpunkten der Gitterzellen durch Mittelwertbildung. Da zwei Gitter A und B gegenseitig Nachbarn zueinander sein können, muss aus Effizienzgründen sichergestellt werden, dass in den entsprechenden Überlappungsbereichen nicht doppelt interpoliert wird. Daher werden in einer $N \times N$-Marker-Matrix die schon miteinander verglichenen Gitter markiert. Die Größe N ist dabei $\max(i, j, k)$. Diese Matrix wird mit 0 initialisiert und eine Zelle i, k wird mit 1 markiert, wenn Gitter i und j miteinander interpoliert werden. Um Arbeitsspeicher zu sparen, geschieht dies in einer Sparse-Repräsentation.

Für eine solche Interpolation der SDF-Werte wird zunächst der Überlappungsbereich zweier benachbarter Gitter berechnet. Dann wird jede Zelle, die in diesem Bereich liegt, durchlaufen. Da die Zellen in einer Hashmap abgelegt sind, lassen sich die Zellen im Überlappungsbereich effizient finden, indem jede Position einer möglichen Zelle im Überlagerungsbereich generiert und dann in den zu vergleichenden Hashmaps nach dem Schlüssel gesucht wird. Bei der Interpolation der SDF-Werte überlappender Zellen müssen fünf Sonderfälle beachtet werden, die in *Abb. 4* skizziert sind.

1. Eine extrudierte Zelle von Gitter A liegt auf einer Zelle von B, und eine extrudierte Zelle von Gitter B liegt auf einer Zelle von Gitter A. Wenn dies der Fall ist, werden die Distanzwerte der

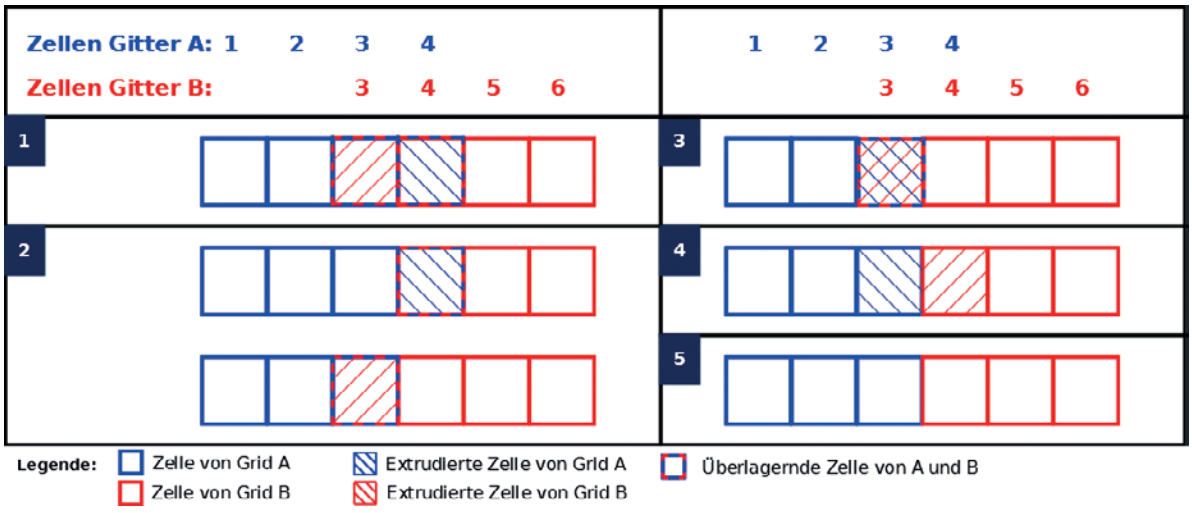

Abb. 4 | Bei der Berechnung des globalen Modells müssen unterschiedliche Konfigurationen in den Überlappungsbereichen berücksichtigt werden. Die verschiedenen Kombinationen von Zellenarten (reguläre Zellen, extrudierte Zellen, überlagernde Zellen), die dort aneinanderstoßen bzw. überlappen, sind hier schematisch aufgezeigt.

extrudierten Zellen von A und B auf einen ungültigen Wert gesetzt. Ein ungültiger Distanzwert sorgt dafür, dass die Zelle bei der Rekonstruktion ignoriert wird. Die SDF-Werte der nicht extrudierten Zellen von A und B werden an der Stoßkante gemittelt.
2. Eine extrudierte Zelle von A überlagert sich mit einer normalen Zelle von B oder umgekehrt. Bei diesem Fall werden auch wieder die SDF-Werte der äußeren Gitterschnittpunkte von der extrudierten Zelle auf einen ungültigen Wert gesetzt. Die äußeren SDF-Werte, die sich die normalen Zellen von Gitter A und B teilen, werden interpoliert.
3. In diesem Fall überschneiden sich nur die extrudierten Zellen der beiden Gitter. Hier wird jeder SDF-Wert der extrudierten Zelle in A mit dem entsprechenden der korrespondierenden Zelle in B interpoliert.
4. Dieser Fall kommt in der Praxis nur sehr selten vor. Dabei grenzen nur die extrudierten Zellen zweier Gitter aneinander. Hier werden die SDF-Werte der gültigen Gitterpunkte der beiden extrudierten Zellen interpoliert.
5. Dieser Fall tritt nur auf, wenn die Gitterstruktur nicht extrudiert wird. Bei diesem Fall schneiden sich keine Zellen. Daher müssen nur die SDF-Werte der Außengitterpunkte von A und B an den Stoßkanten miteinander interpoliert werden (analog zu Fall 1 ohne extrudierte Zellen).

Nachdem alle Teilgitter miteinander interpoliert wurden, haben auch die Überlappungsbereiche immer konsistente SDF-Belegungen. Die Teilgitter werden nun parallel auf den zur Verfügung stehenden Rechnerknoten mittels Marching Cubes trianguliert. Da die Distanzwerte an den Grenzen der Teilgitter konsistent sind, werden nun auch in den Überlappungsbereichen konsistente Triangulationen berechnet, denn die Vertexpositionen der Dreiecke werden im Marching-Cubes-Algorithmus lediglich anhand der SDF-Werte an den Ecken der Zellen bestimmt.

Ein weiteres Problem, das durch die Aufteilung und Zusammenführung der Daten entsteht, ist das Auftreten von doppelten Vertices im Polygonnetz. Diese treten an den Übergängen zwischen zwei Teilpolygonnetzen auf. Sie entstehen dadurch, dass während der Polygonnetzgenerierung nicht bekannt ist, welche Randpunkte in den Nachbardaten liegen. Durch die Interpolation der benachbarten Gitterzellen liegen die doppelt vorkommenden Punkte jedoch immer direkt aufeinander. Das Entfernen der redundanten Punkte geschieht, während das Gesamtpolygonnetz aus den einzelnen Teilnetzen zusammengesetzt wird.

4.2 Globale Zusammenführung

Benachbarte Zellen lassen sich in der vorgestellten Gitterstruktur relativ leicht finden, da sich benachbarte Zellen über ein Inkrement bzw. Dekrement der Zellenindizes direkt laden lassen. Das Problem bei der verteilten Rekonstruktion ist, dass sich benachbarte Zellen nicht unbedingt im Arbeitsbereich des gleichen Knotens befinden müssen. Daher können doppelt vorhandene Vertices erst nach der Rekonstruktion vom Master identifiziert und zusammengefasst werden. Nachdem die Teilrekonstruktionen vorliegen, wird im Masterprozess überprüft, ob die Vertices an den Rändern des jeweiligen Rekonstruktionsvolumens nicht schon bereits berechnet wurden. Dazu werden im Extrusionsbereich jeweils die Zellen aus dem benachbarten Bereichen dazugeladen, sodass bereits vorhandene Vertices wiederverwendet werden können. Die zu ladenden Daten können anhand des bei der Serialisierung generierten Dateinamens erkannt werden. Dieser Ansatz ermöglicht es nun, lokal überlappende Teilgitter zusammenzuführen, ohne das Redundanzen entstehen. Der Nachteil dieses Ansatzes ist, dass bei der Zusammensetzung der Gesamtrekonstruktion sehr viele Dateizugriffe erforderlich sind. Dies wirkt sich negativ auf die Gesamtlaufzeit aus, ist aber erforderlich, um konsistente Gesamtrekonstruktionen zu erhalten, wie im folgenden Abschnitt gezeigt wird.

5 EXPERIMENTELLE ANALYSE DES VERFAHRENS

Der Ansatz zur verteilten Flächenrekonstruktion ist anhand verschiedener Kriterien betrachtet worden. Zunächst werden qualitative Ergebnisse der Rekonstruktionen anhand eines repräsentativen Datensatzes präsentiert. Danach wird die Skalierbarkeit des Ansatzes in einem Cluster-Rechner evaluiert. Im dritten Teil wird die geometrische Genauigkeit der parallelisierten Rekonstruktion anhand eines Vergleichs mit der seriellen Variante analysiert. Alle hier vorgestellten Ergebnisse wurden auf einem Rechnerverbund aus drei Dell-PowerEdge-R530-Servern mit jeweils zwei Intel-Xeon-E5-2680-CPU mit 28 Kernen und 192 GB Arbeitsspeicher pro Rechner erzeugt. Insgesamt standen also 84 CPU-Kerne Verfügung. Jeder davon verfügt umgerechnet im Schnitt über ca. 6 GB RAM. Die Rechner sind mittels Gigabit-LAN untereinander in einem lokalen Netzwerk verbunden. Als Betriebssystem kam Ubuntu 16.04 zum Einsatz.

5.1 Qualitative Ergebnisse

Zunächst werden einige Beispielrekonstruktionen anhand eines Testdatensatzes gezeigt. Evaluiert wurde ein Datensatz des Bremer Marktplatzes, der mit einem Riegl VZ400 aufgenommen wurde. Dieser hat laut Datenblatt eine Messgenauigkeit von 5 mm auf 100 m Entfernung. Das Volumen der Punktwolke betrug ca. 300 m × 500 m × 300 m. Insgesamt enthält der Datensatz ca. 150 Mio. Messpunkte.

Eine Visualisierung der Punktwolke und der daraus berechneten Rekonstruktion ist in *Abb. 5* zu sehen. Das linke Bild zeigt die Eingangspunktwolke, das rechte Bild ein Rendering des berechneten Polygonnetzes. Die Rekonstruktion erfolgte mit einer Voxelgröße von 5 cm. Dieser Wert hat sich in der Praxis als sinnvoller Wert für eine bestmöglich aufgelöste Rekonstruktion auf diesem Datensatz erwiesen. Bei kleinerer Voxelgröße fragmentieren entfernte zusammenhängende Flächen, da die Gittergröße dann kleiner ist als die Punktdichte. Auch die Rekonstruktionszeit pro Knoten steigt signifikant mit kleinerer Auflösung (siehe *Tab. 1*).

Das so berechnete Dreiecksnetz besteht aus ca. 50 Mio. Dreiecken. Weitere Details aus der Rekonstruktion sind in *Abb. 6* gezeigt. Wie in den Abbildungen zu erkennen ist, sind in den Rekonstruktionen optisch keine Artefakte zu erkennen, die darauf schließen lassen, dass die Rekonstruktion parallel auf mehrere

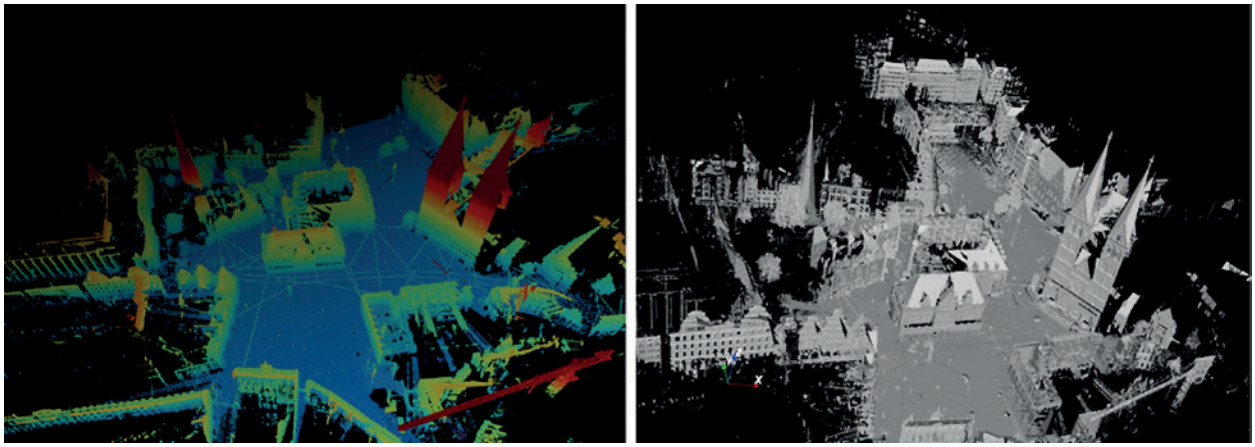

Abb. 5 | Der Testdatensatz des Bremer Marktplatzes als 3D-Punktwolke (links) und das rekonstruierte Dreiecksnetz (rechts)

Voxelgröße in cm	1	2	3	4	5	6	7	8	9	10
Laufzeit in mm:ss	55:54	22:44	14:32	10:24	07:58	04:29	04:18	04:05	04:11	03:58

Tab. 1 | Laufzeit der Rekonstruktion pro Knoten am Beispieldatensatz in Abhängigkeit der Voxelgröße (80 aktive Knoten im Cluster)

Rechner verteilt ablief. Die zusammenhängenden Flächen sind glatt ohne erkennbare Übergänge. Wären in diesem Bereich Inkonsistenzen vorhanden, würden sich diese an den Wandflächen durch „ausfransende" Dreiecke zeigen und wären in der gewählten Darstellung deutlich erkennbar.

Welchen Einfluss diese Interpolation auf das Gesamtergebnis hat, ist in dem in *Abb. 7* gezeigten Detailausschnitt veranschaulicht. Ohne die Interpolation innerhalb der überlappenden Bereiche (rot) liegen die Rekonstruktionen an den Stoßkanten der Teilgitter nicht übereinander und sorgen für eine inkonsistente Rekonstruktion, wie im linken Teilbild zu erkennen ist. Nach der Interpolation der

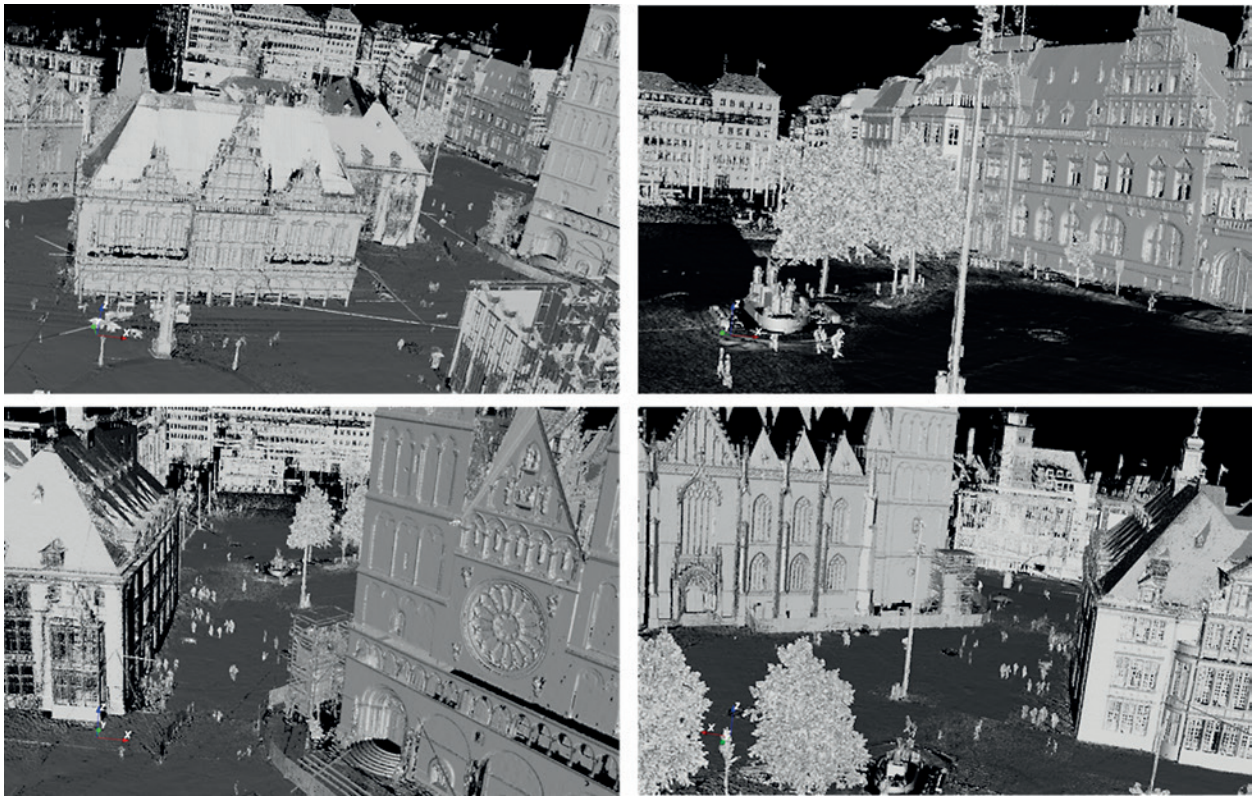

Abb. 6 | Detailansichten aus der Rekonstruktion des Bremer Marktplatzes

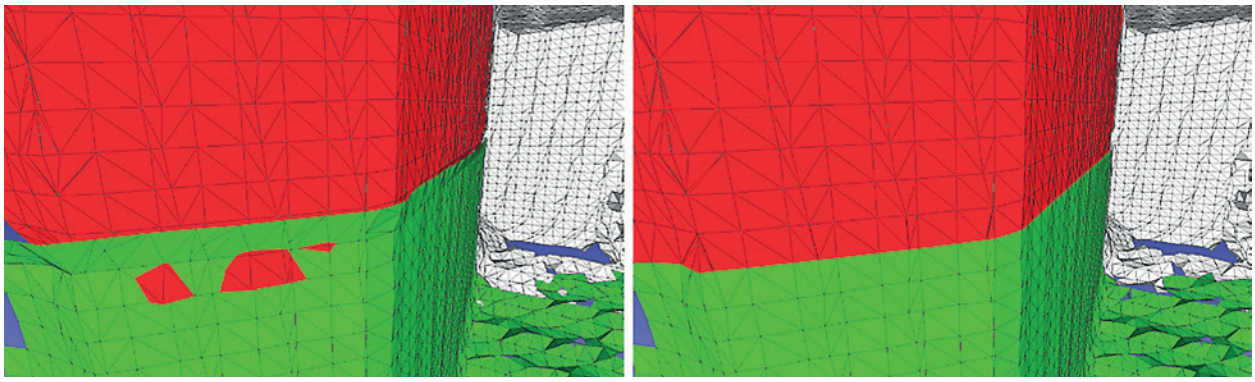

Abb. 7 | Einfluss der Interpolation der Distanzwerte in den Überlappungsbereichen auf die Qualität der Gesamtrekonstruktion

Distanzwerte in den Überlappungsbereichen liegen die erzeugten Dreiecke genau übereinander und sorgen für einen glatten Übergang zwischen den Teilrekonstruktionen.

5.2 Untersuchungen zur Skalierbarkeit des Ansatzes

Eine Frage bei der Implementierung paralleler Algorithmen ist die Skalierbarkeit mit der Anzahl der zur Verfügung stehenden CPU-Kerne. Idealerweise sollte der Overhead zur Verteilung und Verwaltung der generierten Datenpakete nicht den Gewinn an Laufzeit, der durch das Hinzufügen weiterer Rechner erzielt werden kann, übersteigen. Daher ist die Skalierbarkeit des Ansatzes über die Anzahl der zur Verfügung stehenden Slave-Nodes auf dem Dell-Rechnerverbund evaluiert worden.

Das Ergebnis für die Rekonstruktion bei einer Voxelgröße von 5 cm ist in *Abb. 8* links dargestellt. Es ist deutlich zu erkennen, dass die Rechenzeit bei der Rekonstruktion ohne Interpolation der Distanzwerte in den Überlappungsbereichen sehr gut mit der Anzahl der Slave-Nodes skaliert. Die Rechenzeit halbiert sich in etwa bei Verdoppelung der Slave-Zahl (rote Kurve). Dies spricht für eine sehr gute Parallelisierbarkeit des Verfahrens und einen geringen Kommunikationsoverhead in der vorliegenden Implementierung. Die Abflachung der Kurve bei vielen Knoten ist ein typischer Effekt bei parallelen Implementierungen, da die Effizienz aufgrund verschiedener Faktoren – z. B. zur Verfügung stehender Bandbreite – nicht beliebig gesteigert werden kann. In der Gesamtbetrachtung mit Interpolation (blaue Kurve) nimmt die Skalierbarkeit bei sehr vielen Knoten ab, was vor allem durch auf den I/O-Overhead beim Laden der Daten aus den Überlappungsbereichen zurückzuführen ist.

Wie oben erläutert, spielt die Interpolation der Vertices eine wichtige Rolle für die Qualität der Rekonstruktionen. Daher ist der Einfluss der Festplattenzugriffe auf die Gesamtlaufzeit noch einmal gesondert evaluiert worden. Dazu wurde die Zeit, die für das Laden und Speichern der Daten benötigt wird, gesondert erfasst. Das Ergebnis dieser Untersuchung ist im rechten Bild in *Abb. 8* dargestellt. Wie man sieht, steigt der Anteil der Zeit, die für das Laden und Speichern der Daten benötigt wird, mit der Anzahl der Prozesse. Dies ist vor allem darauf zurückzuführen, dass mit steigender Anzahl von Prozessen mehr Fragmente in vielen kleinen Dateien entstehen. Dementsprechend müssen bei der Suche nach gemeinsamen Vertices in der vorliegenden Implementierung sehr viele Dateien von den Festplatten der Rechner gelesen werden. Dies erzeugt einen signifikanten I/O-Overhead, der sich negativ auf die Gesamtlaufzeit auswirkt. Es ist geplant, durch die Implementierung einer geeigneten Metadatenstruktur, wie sie auch in den Arbeiten der Autoren zur großräumigen Rekonstruktion mit KinectFusion /Igelbrink et al. 2015/ verwendet wird, diesen Overhead zu minimieren.

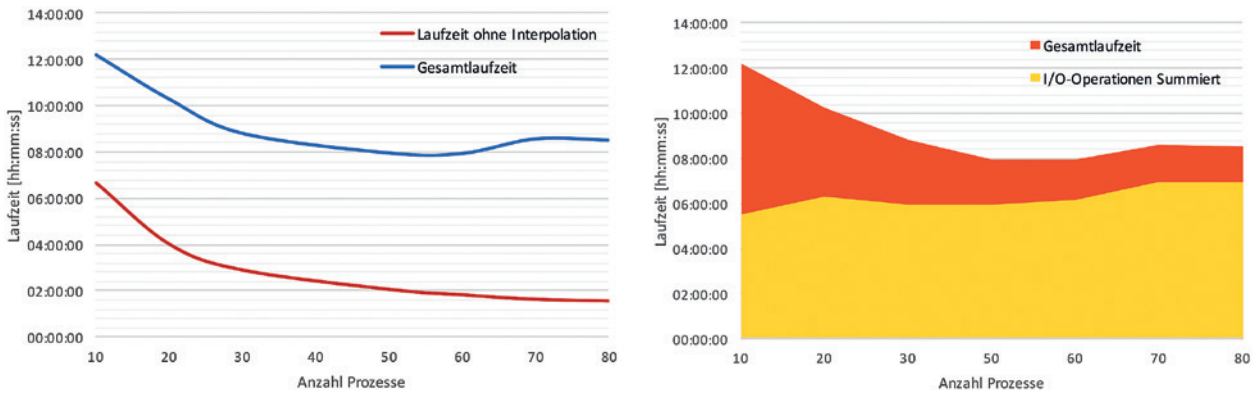

Abb. 8 | Laufzeitverhalten des Verfahrens mit und ohne Interpolation in den Übergangsbereichen (links) sowie der Anteil an I/O-Operationen (rechts) an der Gesamtlaufzeit

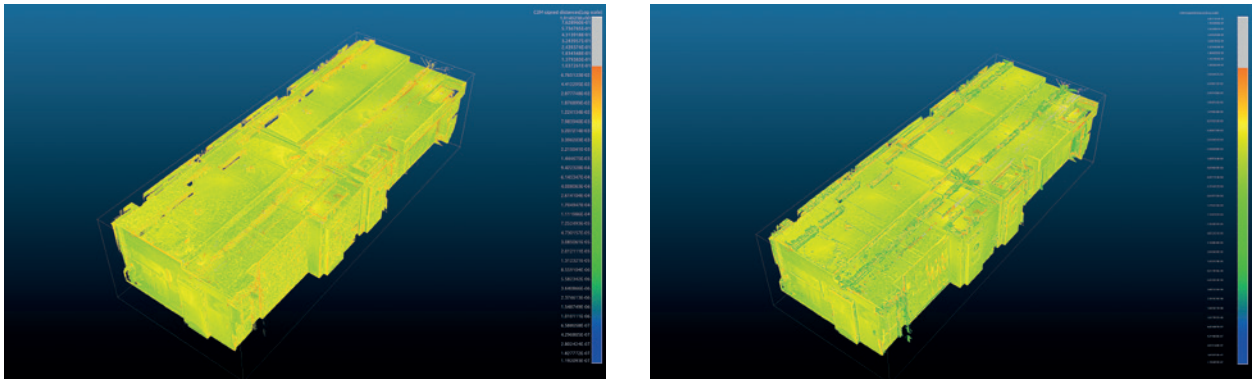

Abb. 9 | Vergleich der seriellen Rekonstruktion (rechts) mit der verteilten Rekonstruktion (links)

5.3 Vergleich mit dem seriellen Verfahren

In Abschnitt 5.1 zeigte die qualitative Analyse, dass das vorgestellte Verfahren subjektiv keine negativen Effekte auf die Qualität der Rekonstruktion hat. Um den Einfluss der Verteilung besser quantifizieren zu können, ist das verteilte Verfahren mit der Rekonstruktion ohne Verteilung der Daten verglichen worden. Dazu erfolgte die Aufnahme eines weiteren Datensatzes mit ca. 90 Mio. Punkten, der sich auf einem Desktop-PC mit 32 GB RAM gerade noch mit der Zielauflösung von 5 cm rekonstruieren lässt. Mit dieser Referenz sind die zwei Verfahren miteinander verglichen worden, indem dieser Datensatz einmal ohne Aufteilung und ein weiteres Mal mit dem verteilten Verfahren unter Verwendung derselben Parameter rekonstruiert wurde.

Dazu kam das Open-Source-Tool CloudCompare[2] zum Einsatz. Mit diesem Programm lassen sich Abweichungen zwischen Dreiecksnetzen und Punktwolken berechnen und visualisieren. Das Ergebnis dieses Vergleichs ist in *Abb. 9* dargestellt. Laut CloudCompare ergibt sich für die serielle Rekonstruktion eine mittlere Abweichung von 0,3 mm (Varianz 0,0017), für die verteilte Rekonstruktion wird ein mittlerer Fehler von 0,4 mm (Varianz 0,0052) ermittelt. Dies spiegelt sich auch in den Falschfarbenbildern in *Abb. 9* wider. Die Fehlerwerte sind im linken Bild, welches das Ergebnis der verteilten Rekonstruktion zeigt, minimal schlechter. Die Verteilung der Fehler ist in beiden Fällen konsistent, d. h., die größten Fehler treten an scharfen Kanten auf, was im Wesentlichen auf den verwendeten Marching-Cubes-Algorithmus zurückzuführen ist.

Insgesamt nähern die Rekonstruktionen die Eingangsdaten in beiden Fällen sehr gut an. Ein signifikanter Abfall der Rekonstruktionsqualität ist in der verteilten Variante nicht zu beobachten. Es ist allerdings zu beachten, dass die Eingangsdaten bereits mit einem Fehler von 5 mm belegt waren. Daher ist davon auszugehen, dass sich dieser auch in den Rekonstruktionen niederschlägt. In weiteren Experimenten müssten die rekonstruierten Umgebungen exakt vermessen und die entsprechenden Kennzahlen mit denen aus den Rekonstruktionen verglichen werden.

6 ZUSAMMENFASSUNG UND FAZIT

In diesem Beitrag ist ein Verfahren zur verteilten Rekonstruktion von Dreiecksnetzen aus 3D-Punktwolken vorgestellt und evaluiert worden. Es wurde gezeigt, dass die geometrische Genauigkeit der Rekonstruktion durch die Unterteilung der Daten in Teildatensätze, die parallel mittels MPI auf einem Rechnercluster bearbeitet und nach der Rekonstruktion wieder zusammengeführt werden, nicht signifikant abnimmt. Dies wird dadurch erreicht, dass die Daten so verteilt werden, dass sich zwischen den Teilstücken Überlappungen ergeben, in denen die berechneten Distanzwerte interpoliert werden. Diese Interpolation führt in der derzeitigen Implementierung dazu, dass die Rechenzeit aufgrund vieler Datenzugriffe steigt. Insgesamt skaliert diese Implementierung aber bereits gut mit der Anzahl der verwendeten CPU-Kerne. Zukünftige Erweiterungen werden darauf abzielen, die Zahl der Datenzugriffe zu minimieren, um die Laufzeit weiter zu verringern.

LITERATUR

Elseberg, J.; Borrmann, D.; Nüchter, A. (2013): One Billion Points in the Cloud – An Octree for Efficient Processing of 3D Laser Scans. In: ISPRS Journal of Photogrammetry and Remote Sensing, 76(2013), 76–88.

Hoppe, H.; DeRose, T.; Duchamp, T; McDonald, J.; Stuetzle, W. (1992): Surface Reconstruction from Unorganized Points. In: Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 92), 26.–31. Juli 1992, Chicago, IL, USA. ACM, New York, 71–78.

Igelbrink, T.; Wiemann, T.; Hertzberg, J. (2015): Generating Topologically Consistent Triangle Meshes from Large Scale Kinect Fusion. In: Proceedings of the European Conference on Mobile Robots 2015 (ECMR 2015), 2.–4. September 2015, Lincoln, UK. IEEE, New York, 252–257.

Kazhdan, M.; Bolitho, M.; Hoppe, H. (2006): Poisson Surface Reconstruction. In: Proceedings of the 4th Eurographics Symposium on Geometry Processing (SGP '06), 26.–28. Juni 2006, Cagliari, Italien. Eurographics Association, Genf, 61–70.

Lefebvre, S.; Hoppe, H. (2006): Perfect spatial hashing. In: Proceedings of the 33rd International Conference on Computer Graphics and Interactive Techniques (SIGGRAPH 06), 30. Juli–3. August 2006, Boston, MA, USA. ACM, New York, 579–588.

[2] http://www.danielgm.net/cc/

Lorensen, W. E.; Cline, H. E. (1987): Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In: Proceedings of the 14th Annual Conference on Computer Computer Graphics and Interactive Techniques (SIGGRAPH 87), 27.–31. Juli 1987, Anahein, CA, USA. ACM, New York, 163–169.

Nießner, M.; Zollhöfer, M.; Izadi, S.; Stamminger, M. (2013): Real-time 3D reconstruction at scale using voxel hashing. In: ACM Transactions on Graphics (ToG), 32(2013)6, Article 169.

Rinnewitz, K. O.; Schalk, S. K.; Wiemann, T.; Lingemann, K.; Hertzberg, J. (2012): Das Las Vegas Reconstruction Toolkit. In: Luhmann, Th.; Müller, Chr. (Hrsg.): Photogrammetrie – Laserscanning – Optische 3D-Messtechnik. Beiträge der Oldenburger 3D-Tage 2012. Wichmann, Berlin/Offenbach, 126–133.

Wiemann, T.; Annuth, H.; Lingemann, K.; Hertzberg, J. (2015): An Extended Evaluation of Open Source Surface Reconstruction Software for Robotic Applications. In: Journal of Intelligent and Robotic Systems, 77(2015)1, 149–170.

Wiemann, T.; Mrozinski, M.; Feldschnieders, D.; Lingemann, K.; Hertzberg, J. (2016): Data Handling in Large Scale Surface Reconstruction. In: Menegatti, E.; Michael, N.; Berns, K.; Yamaguchi, H. (Hrsg.): Intelligent Autonomous Systems 13. Proceedings of the 13th International Conference IAS-13. Springer International Publishing, Cham, 499–511.

Dr. Thomas Wiemann

UNIVERSITÄT OSNABRÜCK
INSTITUT FÜR INFORMATIK

Wachsbleiche 27 | 49090 Osnabrück
twiemann@informatik.uni-osnabrueck.de

B. Sc. Isaak Mitschke

UNIVERSITÄT OSNABRÜCK
INSTITUT FÜR INFORMATIK

Wachsbleiche 27 | 49090 Osnabrück
imitschke@uni-osnabrueck.de

Manuskript eingereicht: 10.07.2017 | Im Peer-Review-Verfahren begutachtet