Verteilte Oberflächenrekonstruktion aus 3D-Punktwolken

Distributed Surface Reconstruction from 3D Point Clouds

Thomas Wiemann, Isaak Mitschke

Zusammenfassung

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 PCs ist. In diesem Beitrag zeigen wir einen Ansatz, mit dem besonders große Punktwolken auf Computerclustern mit Hilfe von MPI in geeignete Teilpunktwolken zerteilt und rekonstruiert werden können. Das vorgestellte Verfahren wird nach verschiedenen Kriterien wie Skalierbarkeit und geometrischer Genauigkeit evaluiert.

Schlüsselwörter: Oberflächenrekonstruktion, 3D-Laserscanning

Abstract

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 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.

Keywords: Surface Reconstruction, 3D Laser Scanning

1 EINLEITUNG

Bei der Vermessung großflächiger Areale mit 3D Laserscannern fallen enorme Datenmengen in Form von 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 großvolumigen und hochaufgelösten 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 also große, hochaufgelöste 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 Nodes – bestehen, kann man die Rekonstruktion parallel durchführen, 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 Datenunkte im Hauptspeicher einer dedizierten Master-Node 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 Node geladen werden. Diese Teilpunktwolken können

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

so parallel verarbeitet werden. Die von den Knoten berechneten Teilrekonstruktionen werden anschließend in einer konsistenten Gesamtrekonstruktion vereint. Dabei muss darauf geachtet werden, dass keine Redundanzen, wie z.B. mehrfaches Vorkommen eines gleichen Dreieckspunkts oder einer gleichen Dreiecksdefinition, entstehen.

Im Folgenden werden wir die Datenstruktur zur Vorverarbeitung der Punktwolken und Serialisierung für die Knoten im Gitter vorstellen. Anschließend präsentieren wir den Ablauf des Rekonstruktionsvorgangs. Zum Schluss zeigen wir die Performanz des implementierten Verfahrens an verschiedenen Beispieldatensätzen.

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 zugrundeliegenden 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 geladenen Punkte eine Flächennormale geschätzt. Anschließend 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 Gittereckpunktes, bekommt die projizierte Distanz einen positiven Wert, ansonsten einen negativen („Signed Distance Funktion“, /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 und Cline 1987/). Die Auflösung des Gitters definiert so die Auflösung des berechneten Dreiecksnetzes.

Ebendiese 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. 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 Artikel beschreiben wir, wie sich diese erweitern lässt, so dass es möglich ist, großflächige und hochaufgelöste Rekonstruktionen verteilt auf verschiedenen Rechnern in einem Rechnerverbund zu berechnen. Dabei werden zwei Zielstellungen verfolgt: erstens soll die Rekonstruktion parallelisiert durchgeführt werden können; zweitens soll es möglich sein, den benötigten Arbeitsspeicher zu begrenzen, so dass auf Rechnern mit wenig Arbeitsspeicher die Rekonstruktion durch Unterteilung der Daten in kleinere Datenpakete ebenfalls, unter Inkaufnahme einer höheren Laufzeit, trotzdem eine Rekonstruktion vorgenommen werden kann. In diesem Papier gehen wir davon aus, dass die Rekonstruktion in einem Rechnerverbund in einem lokalen Netzwerk erfolgt. Die Kommunikation der Rechner erfolgt mittels der offenen 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.

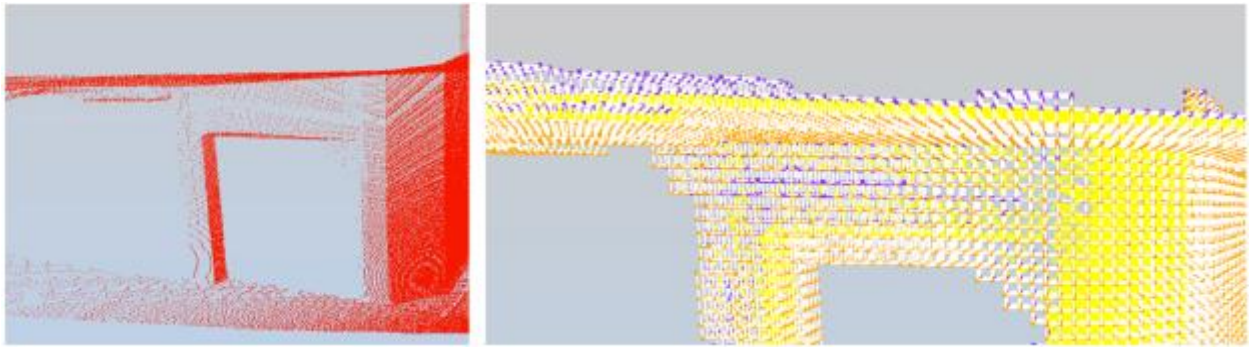

Abb.1: Ausschnitt aus einem 3D-Scan eines Gebäudes (links). Die dazugehörigen Gitterzellen mit der relativen Orientierung der Zellvertices zur gescannten Oberfläche sind im rechten Bild dargestellt (blau: oberhalb der Fläche, orange unterhalb).

3 GITTERSTRUKTUR ZUR VERTEILTEN REKONSTRUKTION

Klassischerweise werden zur Aufteilung des Rekonstruktionsvolumens sogenannte Octrees verwendet. Diese Baumstruktur lässt sich leicht und speichereffizient implementieren /Elseberg et al. 2013/. Ein Octree hat allerdings den Nachteil, dass sich die Nachbarn der erzeugten Voxel nur schwer unter Inkaufnahme eines signifikanten Speicher- und Laufzeitoverheads bestimmen lassen /Wiemann et al. 2016/. Um diesen Nachteil auszugleichen, wurde für das Las Vegas Reconstruction Toolkit ein Voxelhashing-Ansatz implementiert. Die Idee dabei 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, wie folgt bestimmen:

$$i = \left\lfloor \frac{p_x - x_{min}}{v} \right\rfloor, j = \left\lfloor \frac{p_y - y_{min}}{v} \right\rfloor, k = \left\lfloor \frac{p_z - z_{min}}{v} \right\rfloor \quad (1)$$

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

$$H(i,j,k) = i \cdot \ dim_x + j \cdot dim_y + k \quad (2)$$

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

$$dim_x = \left\lfloor \frac{x_{max} - x_{min}}{v} \right\rfloor, dim_y = \left\lfloor \frac{y_{max} - y_{min}}{v} \right\rfloor \quad (3)$$

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. Ein Ausschnitt aus einem Gitter mit den zur Rekonstruktion bestimmten relativen Orientierungen ist in Abb. 1 gezeigt.

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 die Festplatte gespeichert werden. Diese Teilbereiche können dann bei der Rekonstruktion von den einzelnen Nodes 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.

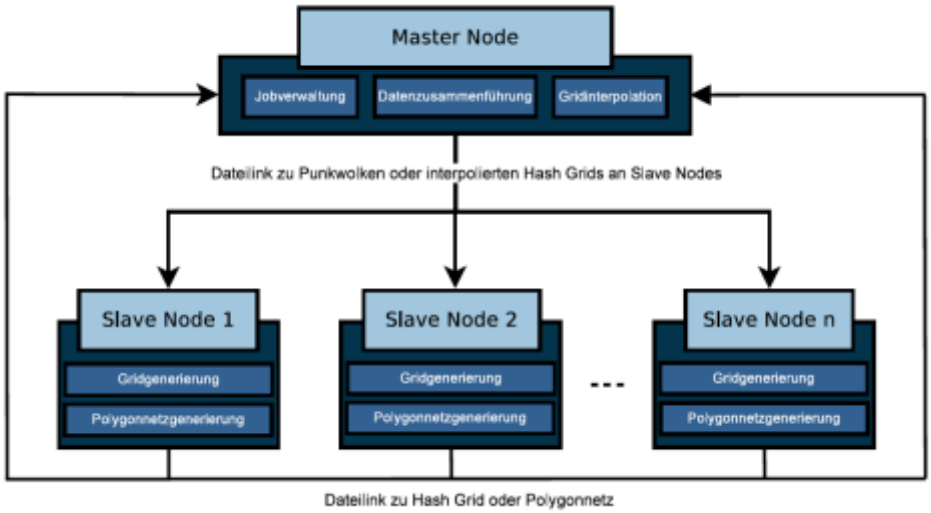

Abb. 2: Prinzipieller Ablauf der Rekonstruktion. Eine Master-Node verwaltet die Jobs, die an die anderen Nodes im Cluster weitergegeben werden. Die berechneten Polygonnetze werden von den Slaves wieder an den Master zurückgesendet und dort fusioniert.

Um Fehlstellen im Gitter auszugleichen, wird um alle erstellten Zellen eine weitere Zelle erstellt. 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. Mit diesem Mittel wird die Qualität der Rekonstruktion in diesen Bereichen deutlich verbessert. Ebenso werden beim Serialisieren an den Schnittkanten weitere Schichten, die sich mit den Nachbarvolumen überlappen, generiert. Dies macht es später möglich, einen konsistenten Übergang zwischen den Teilrekonstruktionen aus den verteilten Volumina zu berechnen. Diese Zusatzzellen bezeichnen wir im Folgenden als Extrusionszellen.

4 ABLAUF DER REKONSTRUKTION

4.1 Aufteilung und Zusammenfügen der Teilrekonstruktionen

In diesem Abschnitt präsentieren wir Details zum eigentlichen Ablauf der verteilten Rekonstruktion. Die Anwendung zur verteilten Rekonstruktion besteht im Wesentlichen aus zwei Komponenten: einer Master-Node im Cluster, die für das Datenmanagement zuständig ist, und einer Menge von Slave-Nodes, die einzelne Teilvolumina rekonstruieren. Die Slave-Nodes 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 zerteilt zunächst die Punktwolke wie im vorherigen Abschnitt beschrieben und erzeugt die Gitterstruktur im Rekonstruktionsvolumen. Anschließend wird dieses in Teile vorgegebener Größe zerlegt und in einzelnen Dateien auf die Festplatte des Systems abgelegt. Nach dieser Serialisierung werden den verfügbaren Slave-Nodes Dateilinks zu den abgelegten Teilvolumina gesendet. Die entsprechenden Dateien werden dann von den Slaves geladen. Dabei wird zunächst das Gitter im Arbeitsspeicher des entsprechenden Rechners im Cluster rekonstruiert. In diesem Teilgitter findet dann die Polygonalisierung der Teilpunktwolken mit dem Marching-Cubes-Algorithmus statt. Diese Teilrekonstruktionen werden ebenfalls auf der Festplatte gespeichert. Ein Link auf die entsprechenden Dateien wird als Antwort von den Slaves an den Master gesendet (Abb. 2).

Wenn alle Teilpunktwolken verarbeitet wurden, wird die globale Rekonstruktion im Master zusammengesetzt. Dieser ist insbesondere dafür zuständig, dafür zu sorgen, dass die Teilrekonstruktionen an den Schnittkanten konsistent zusammengefügt werden. Dabei kann es zu verschiedenen Problemen kommen, die im folgenden Abschnitt detaillierter beschrieben werden.

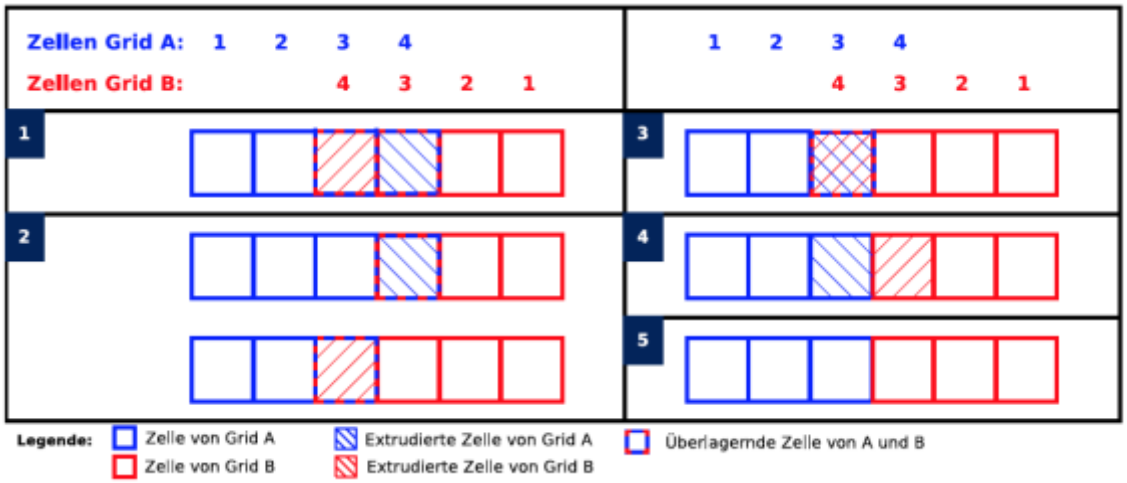

Abb. 3: Übersicht über die unterschiedlichen Konfigurationen der überlappenden Zellen, die bei der Interpolation der Distanzwerte berücksichtigt werden müssen.

4.2 Interpolation der Distanzwerte

Für eine global konsistente Rekonstruktion reicht es nicht aus, die vorher aufgeteilten Daten einfach wieder zusammenzufügen. Die zuvor erzeugten Überlagerungsbereiche müssen so angepasst werden, dass sich die dort erzeugten Rekonstruktionen exakt überlagern. Um dies zu erreichen, werden die Distanzwerte an den Außenzellen zweier benachbarter Gitter interpoliert. Dies übernimmt die Master-Node, nachdem alle Gitter erzeugt und abgespeichert wurden. Dafür lädt die Master-Node nacheinander alle Teilgitter und interpoliert die Signed-Distance-Werte durch Mittelwertbildung mit ihren Nachbarn. Da zwei Gitter A und B gegenseitig Nachbarn zueinander sind, muss aus Effizienzgründen sichergestellt werden, dass nicht doppelt interpoliert wird. Daher werden in einer $N \times N$-Marker-Matrix die schon miteinander verglichenen Gitter markiert. Um Arbeitsspeicher zu sparen, geschieht dies in einer Sparse-Repräsentation.

Für die Interpolation der Randzellen 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 mögliche Überlappungszellen effizient finden, indem jede Position einer möglichen Zelle im Überlagerungsbereich generiert und dann in beiden Hashmaps nach dem Schlüssel gesucht wird. Bei der Interpolation der Nachbarzellen müssen fünf Sonderfälle beachtet werden, die in Abbildung 3 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 äußeren Gitterpunkte von Zelle A4 und Zelle B4 auf einen ungültigen Wert gesetzt. Ein äußerer Gitterpunkt wird definiert als Punkt, der nur zu einer Zelle in einem lokalen Gitter gehört. Ein ungültiger Distanzwert sorgt dafür, dass die Zelle bei der Rekonstruktion ignoriert wird. Die Distanzwerte der Gitterpunkte, die sowohl in A3 als auch B3 liegen, werden interpoliert. Dabei wird einfach das arithmetische Mittel beider Distanzwerte als neuer Distanzwert gesetzt.
2. Eine extrudierte Zelle von A überlagert sich mit einer normalen Zelle von B oder umgekehrt. Bei diesem Fall werden auch wieder die Distanzwerte der äußeren Gitterschnittpunkte von der extrudierten Zelle auf einen ungültigen Wert gesetzt. Die äußeren Gitterschnittpunkte, 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 Distanzpunkt der extrudierten Zelle in A mit 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 äußeren 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 die Außengitterpunkte von A und B miteinander interpoliert werden.

Nachdem alle Teilgitter miteinander interpoliert wurden, können diese an die Slave-Nodes verschickt werden. Die Gitter werden dann von den Slave-Nodes zur Generierung von Polygonnetzen verwendet. Da die

Distanzwerte an den Grenzen der Teilgitter konsistent sind, werden nun auch in den Überlappungsbereichen konsistente Triangulationen berechnet, denn die Interpolationsergebnisse im Marching-Cubes-Algorithmus werden lediglich von den berechneten Entfernungswerten 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.

4.3 Globale Zusammenführung

Benachbarte Zellen lassen sich in der vorgestellten Gitterstruktur relativ leicht finden, da sich benachbarte Zellen über ein Inkrement bzw. Dekrement der Zellenindices direkt laden lassen. Das Problem bei der verteilten Rekonstruktion ist, dass sich benachbarte Zellen nicht unbedingt im Arbeitsbereich der gleichen Node befinden müssen. Daher können doppelt vorhandene Vertices erst nach der Rekonstruktion vom Master identifiziert und zusammengefasst werden. Nachdem die Teilrekonstruktionen vorliegen, muss im Masterprozess überprüft werden, 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, so dass bereits vorhandene Vertices wiederverwendet werden können. Die zu ladenden Daten können anhand des bei der Serialisierung generierten Datennamens 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, im konsistente Gesamtrekonstruktionen zu erhalten, wir im folgenden Abschnitt gezeigt wird.

5 EXPERIMENTELLE ANALYSE DES VERFAHRENS

Wir haben unseren Ansatz zur verteilten Flächenrekonstruktion anhand verschiedener Datensätze und Kriterien evaluiert. Zunächst präsentieren wir qualitative Ergebnisse der Rekonstruktionen anhand eines repräsentativen Datensatzes. Anschließend evaluieren wir die Skalierbarkeit des vorgestellten Ansatzes bei der Verwendung eines Cluster-Rechners. Im dritten Teil analysieren wir die geometrische Genauigkeit der parallelisierten Rekonstruktion anhand eines Vergleichs mit der seriellen Varianten.

Alle hier vorgestellten Ergebnisse wurden auf einem Rechnerverbund bestehend aus drei Dell PowerEdge R530 Servern mit jeweils 2 Intel Xeon E5-2680 CPUs mit insgesamt 28 Kernen pro Rechner und 192 GB Arbeitsspeicher. Insgesamt standen also 84 CPU-Kerne im Cluster zur Verfügung. 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 zeigen wir einige Beispielrekonstruktionen anhand eines Testdatensatzes. Evaluiert wurde ein Datensatz des Bremer Marktplatzes, der mit einem RieglVZ 400i aufgenommen wurde. Die Rekonstruktion wurde mit einer Voxelgröße von 5 cm rekonstruiert. Eine Visualisierung der Punktwolke und der berechneten Rekonstruktion ist in Abb. 4 zu sehen. Das linke Bild zeigt die Eingangspunktwolke, das rechte Bild zeigt ein Rendering des berechneten Polygonnetzes. Die Rekonstruktion erfolgte mit einer Voxelauflösung von 5 cm. Das Volumen der Punktwolke betrug ca. 300 m x 500 m x 300 m. Insgesamt enthält der Datensatz ca. 150 Mio. Messpunkte. Das daraus berechnete Dreiecksnetz besteht aus ca. 50 Mio. Dreiecken. Weitere Details aus dem Datensatz sind in Abb. 5 gezeigt.

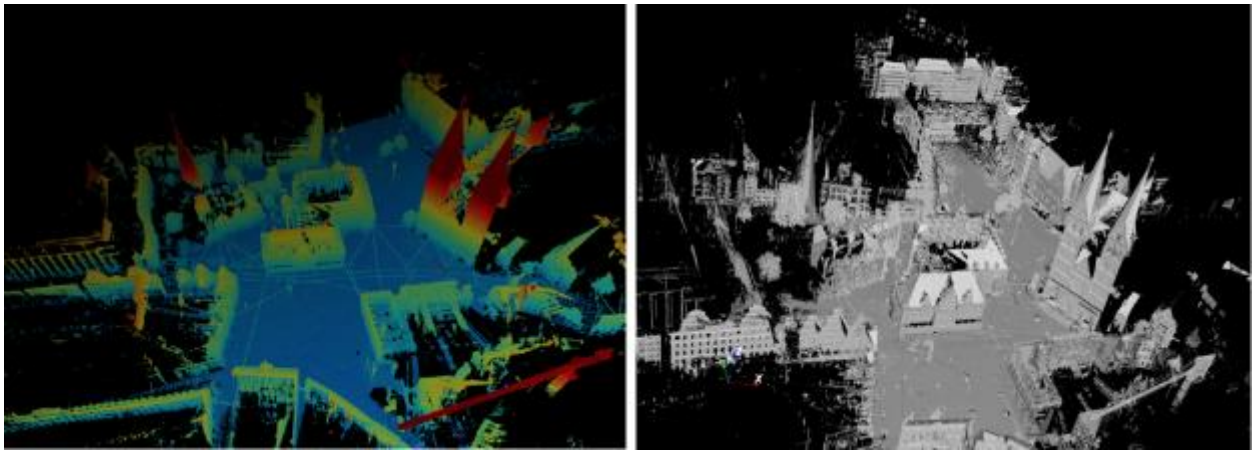

Abb. 4: Der Testdatensatz des Bremer Markplatzes als 3D-Punktwolke (links) und das rekonstruierte Dreiecksnetz (rechts).

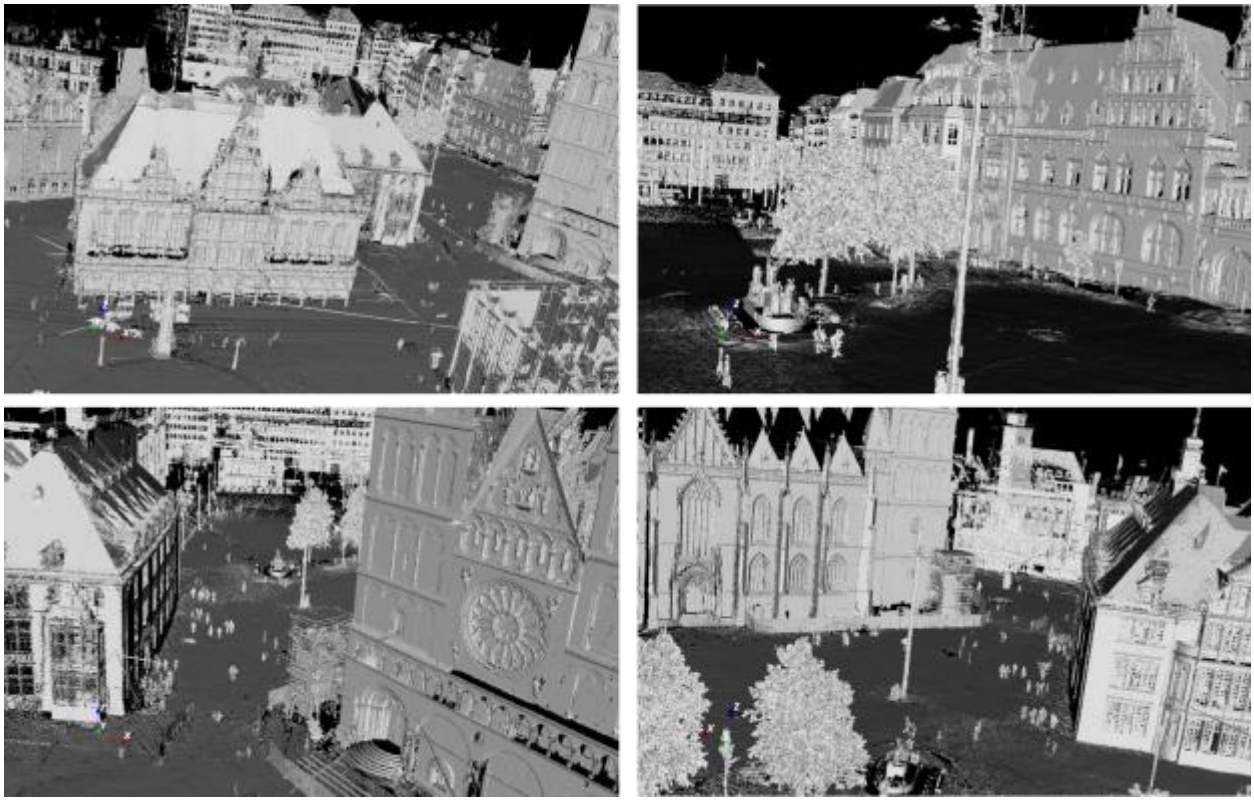

Abb 5: Detailansichten aus der Rekonstruktion des Bremer Marktplatzes.

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 Rechner verteilt ablief. Die zusammenhängenden Flächen sind glatt ohne erkennbare Übergänge. Dies ist auf die in Abschnitt 3.2 vorgestellte Interpolation der Distanzwerte in den Übergangsbereichen zwischen den Teilrekonstruktionen zurückzuführen. Welchen Einfluss diese Interpolation auf das Gesamtergebnis hat, ist in Abb. 6 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 Distanzwerte in den Überlappungsbereichen liegen die erzeugten Dreiecke genau übereinander und sorgen für einen glatten Übergang zwischen den Teilrekonstruktionen.

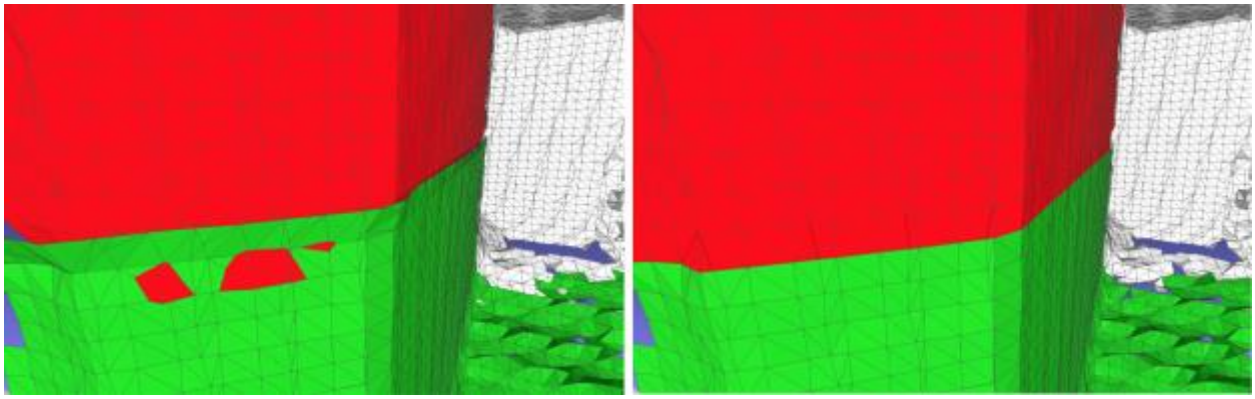

Abb. 6: Einfluss der Interpolation der Distanzwerte in den Überlappungsbereichen auf die Qualität der Gesamtrekonstruktion.

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 haben wir die Skalierbarkeit unseres Ansatzes über die Anzahl der zur Verfügung stehenden Slave-Nodes auf dem Dell-Rechnerverbund evaluiert. Das Ergebnis für die Rekonstruktion bei einer Zielgröße von 2 Mio. Datenpunkten in den Teilgittern ist in Abb. 7 dargestellt. Man kann deutlich 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. 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.

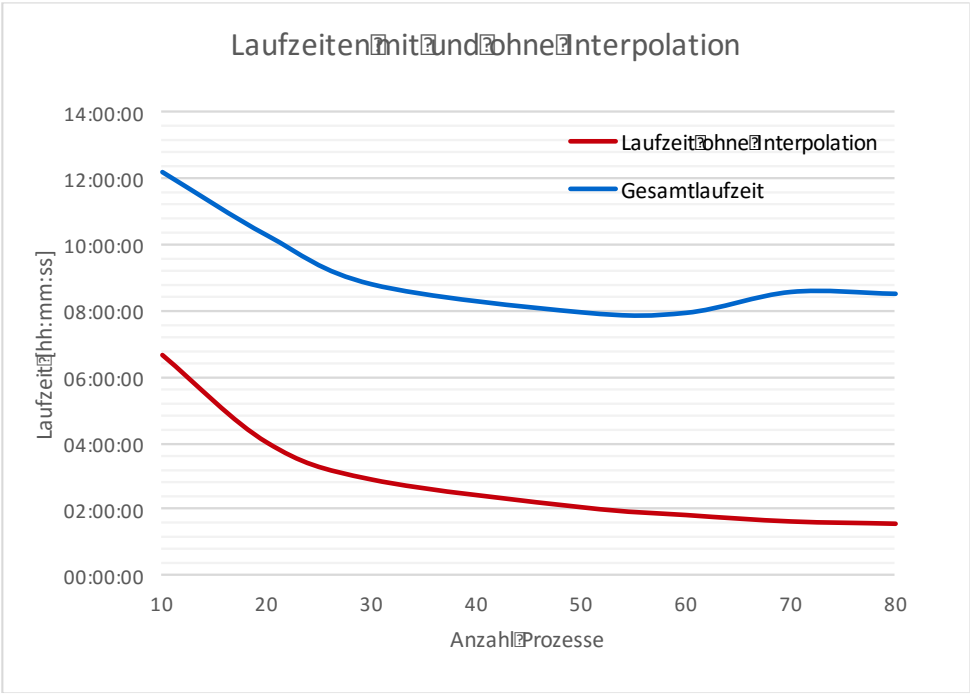

Abb. 7: Laufzeitverhalten des Verfahrens auf verschiedenen Datensätzen mit steigenden Anzahl an CPU-Kernen.

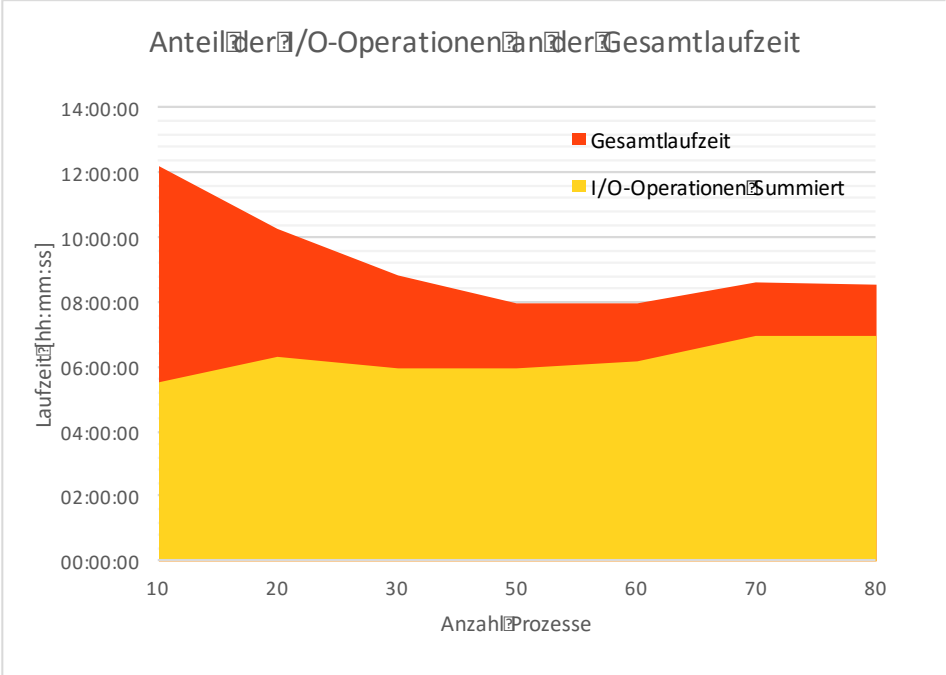

Abb. 8.: Laufzeit des Verfahrens bei Erhöhung der Anzahl der CPU-Kerne mit Interpolation in den Randbereichen (rot) und ohne (schwarz)

Wie oben erläutert, spielt die Interpolation der Vertices eine wichtige Rolle für die Qualität der Rekonstruktionen. Daher haben wir den Einfluss der Festplattenzugriffe auf die Gesamtlaufzeit noch einmal gesondert evaluiert. Dazu haben wir die Zeit, die für das Laden und Speichern der Daten benötigt wird, gesondert erfasst. Das Ergebnis dieser Untersuchung ist 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 die Suche nach gemeinsamen Vertices in der vorliegenden Implementierung erfordert, dass sehr viele Dateien vor dem Vergleich von den Festplatten der Rechner gelesen werden müssen. Dies erzeugt einen signifikanten I/O-Overhead, der sich negativ auf die Gesamtlaufzeit auswirkt. Wie planen durch die Implementierung einer geeigneten Metadatenstruktur, wie sie auch in unseren Arbeiten zur großräumigen Rekonstruktion mit KinectFusion /Igelbrink et al. 2015/ verwendet wird, diesen Overhead zu minimieren.

5.3 Vergleich mit dem seriellen Verfahren

In Abschnitt 4.1 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, haben wir das verteilte Verfahren mit der Rekonstruktion ohne Verteilung der Daten verglichen. Dazu haben wir einen Datensatz 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, als Referenz genommen und die zwei Verfahren miteinander verglichen. Dazu haben wir das OpenSource-Tool CloudCompare[2] verwendet. 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

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

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.

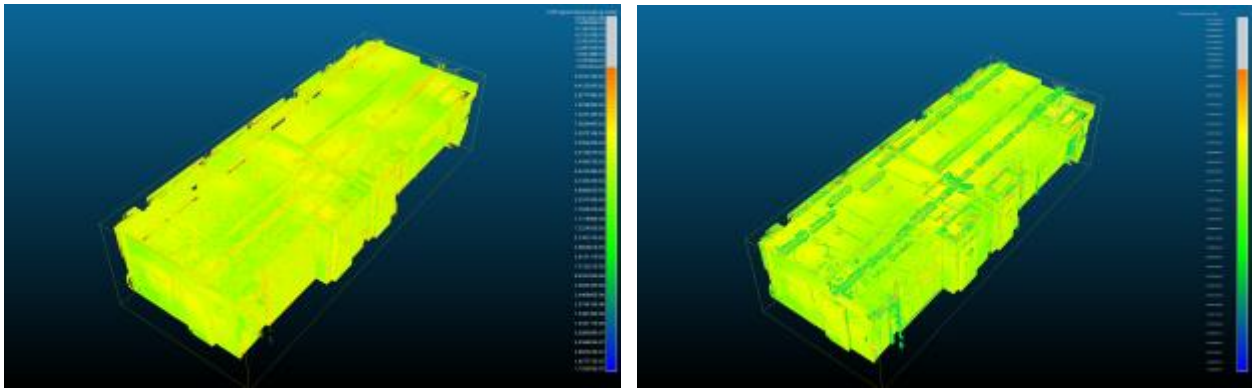

Abb. 9: Vergleich der seriellen Rekonstruktion (rechts) mit der verteilten Rekonstruktion (links).

6 ZUSAMMENFASSUNG UND FAZIT

In diesem Papier haben wir ein Verfahren zur verteilten Rekonstruktion von Dreiecksnetzen aus 3D-Punktwolken vorgestellt und evaluiert. 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 viele 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. ISPRS Journal of Photogrammetry and Remote Sensing, vol. 76, Elsevier, 2013

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), ACM, 1992

Igelbrink, T.; Wiemann, T.; Hertzberg, J. (2015): Generating Topologically Consistent Triangle Meshes from Large Scale Kinect Fusion. Proceedings of the European Conference on Mobile Robots 2015 (ECMR 2015), Lincoln (UK), Sep 2-4, 2015

Lorensen, W. E.; Cline, H. E. (1987): Marching Cubes: A High Resolution 3D Surface Construction Algorithm. In: ACM SIGGRAPH, 1987

Rinnewitz K.O.; Schalk, S. K.; Wiemann, T.; Lingemann K.; Hertzberg, J. (2012): Das Las Vegas Reconstruction Toolkit. Photogrammetrie, Laserscanning, Optische 3D-Messtechnik. Beiträge der Oldenburger 3D-Tage 2012, Wichmann Verlag, 2012.

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

Wiemann, T.; Mrozinski, M.; Feldschnieders, D.; Lingemann K.; Hertzberg, J. (2016): Data Handling in Large Scale Surface Reconstruction, Autonomous Systems 13. Springer, Advances in Intelligent Systems and Computing, vol. 302, pp. 499-511, 2016

Autoren:

Dr. Thomas Wiemann,
Institut für Informatik, Arbeitsgruppe Wissensbasierte Systeme,
Universität Osnabrück,
Wachsbleiche 27, 49090 Osnabrück,
twiemann@informatik.uni-osnabrueck.de

Isaak Mitschke, B.Sc.
Institut für Informatik,
Arbeitsgruppe Wissensbasierte Systeme,
Universität Osnabrück, Wachsbleiche 27,
49090 Osnabrück,
imitschke@uni-osnabrueck.de