Massiv parallelisierte Oberflächenrekonstruktion aus großvolumigen und hochaufgelösten 3D-Punktwolken

Thomas WIEMANN und Isaak MITSCHKE

Zusammenfassung

Mit dem Las Vegas Reconstruction Toolkit (LVR) (RINNEWITZ ET AL. 2012) der Universität Osnabrück lassen sich aus 3D-Punktwolken Polygonnetze erzeugen. Wenn besonders große 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 parallel verarbeitet werden können. Dafür werden die Punktwolken räumlich zusammenhängend aufgeteilt und anschließend auf der Festplatte abgespeichert. Nach dieser räumlichen Aufteilung werden sie dann bei der Rekonstruktion von den einzelnen Prozessen zur weiteren Verarbeitung bei Bedarf nachgeladen. Wir zeigen die implementierte Funktionalität beispielhaft anhand eines räumlich ausgedehnten und hochaufgelösten Musterdatensatzes. Für die Berechnungen stand ein SGI UV 2000 Hochleistungsrechner zur Verfügung, auf dem die hier präsentierten Ergebnisse erzielt wurden.

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, dennoch werden keine mathematisch zusammenhängenden Flächen aufgenommen, sondern nur sehr viele Stichproben von den zu erfassenden 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).

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 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 einem Beispieldatensatz.

2 Gitterstruktur zur Rekonstruktion

Die Polygonalisierung der Punktwolken erfolgt nach dem Marching-Cubes-Verfahren (LORENSEN & CLINE 1987). Dabei wird das Volumen, das von den Messpunkten eingenommen wird, in gleichmäßige kubische Zellen (Voxel) unterteilt. Diese Voxel haben jeweils acht Eckpunkte, für die die relative Orientierung und der Abstand zur vermessenen Oberfläche bestimmt werden. Anhand der Abstände und Orientierungen kann dann innerhalb einer Zelle ein vorberechnetes Muster von Dreiecken verwendet werden, um den Verlauf der gemessenen Oberfläche innerhalb der Zelle zu approximieren. Zur Bestimmung von Orientierung und Abstand wird Hoppes Distanzfunktion verwendet (HOPPE 1992). Dabei wird an die k nächsten Nachbarn eines Referenzpunktes eine Ebene interpoliert. Der Abstand des Punktes zu dieser Ebene definiert seine Distanz, die Normale der Ebene seine relative Position.

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:

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

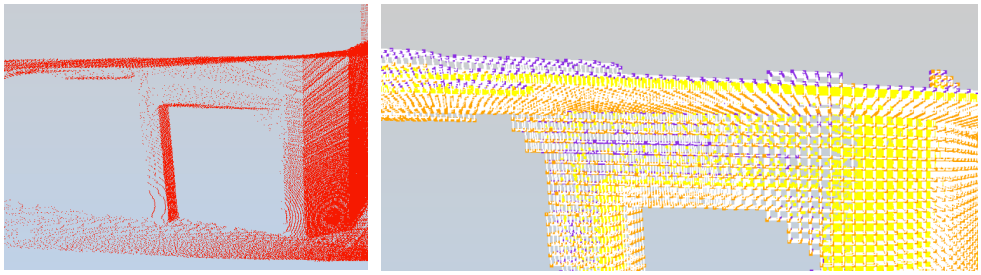

Abb.1: Ausschnitt aus einem 3D-Scan eines Gebäudes (links). Die dazugehörigen Gitterzellen mit der relativen Orientierung der Zellvertices zur gescannten Oberfläche sind im rechten Bild dargestellt (blau: oberhalb der Fläche, orange unterhalb).

Alle Zellen werden in einer Hashmap verwaltet. Zur Adressierung der Zellen anhand des Index-Tripels wird folgende Hashfunktion verwendet

$$H(i,j,k) = i \cdot dim_x + j \cdot dim_y + k \tag{2}$$

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

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

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 Abbildung 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.

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.

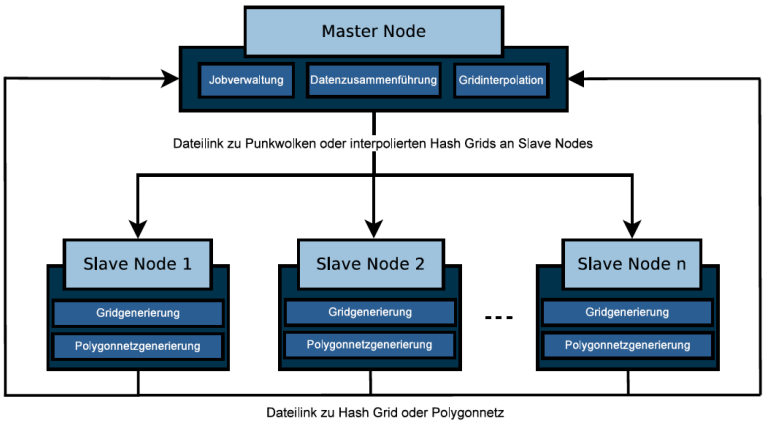

Abb. 2: Prinzipieller Ablauf der Rekonstruktion. Eine Master-Node verwaltet die Jobs, die an die anderen Nodes im Cluster weitergegeben werden. Die berechneten Polygonnetze werden von den Slaves wieder an den Master zurückgesendet und dort fusioniert.

3 Ablauf der Rekonstruktion

3.1 Verteilung der Teilvolumina zur parallelen Rekonstruktion

In diesem Abschnitt präsentieren wir Details zum eigentlichen Ablauf der verteilten Rekonstruktion. Die Anwendung zur verteilten Rekonstruktion besteht im Wesentlichen aus zwei Komponenten: eine Master-Node im Cluster, die für das Datenmanagement zuständig ist, und eine 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 beschriebenund 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.

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.

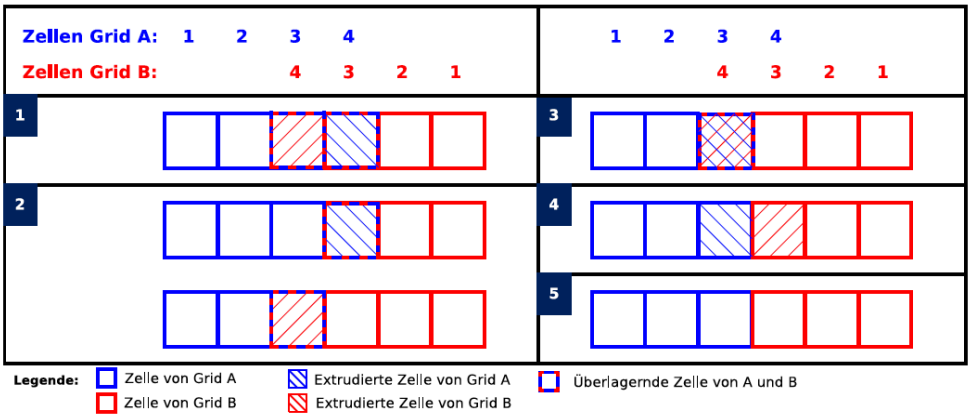

Abb. 3: Übersicht über die unterschiedlichen Konfigurationen der überlappenden Zellen, die bei der Interpolation der Distanzwerte berücksichtigt werden müssen.

3.2 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 diese 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 eine $N \times N$-Matrix die schon miteinander verglichenen Gitter markiert. Um Speicher zu schonen, 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, wie in Abbildung 3 dargestellt ist.

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 Punkten 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. Dabei wird bei der Zusammenführung von Teilrekonstruktionen mittels einer Set-Datenstruktur sichergestellt, dass jeder räumliche Punkt nur einmal in das globale Dreiecksnetz eingefügt wird. Prinzipiell könnten die redundanten Vertices auch direkt bestimmt werden, wie in IGELBRINK ET AL. (2015) gezeigt wurde. In der vorliegenden Implementierung wurden diese Ansätze aber leider noch nicht umgesetzt.

4 Experimente und Diskussion

Die folgenden Ergebnisse wurden auf dem SGI UV2000 Cluster der Universität Osnabrück berechnet. Die Tests erfolgten auf einem hochaufgelösten und räumlich ausgedehnten Scan des Bremer Markplatzes[1]. Eine Visualisierung der Punktwolke und der berechneten Rekonstruktion ist in Abbildung 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.

[1] http://kos.informatik.uni-osnabrueck.de/3Dscans/ (Datensatz Nr. 12)

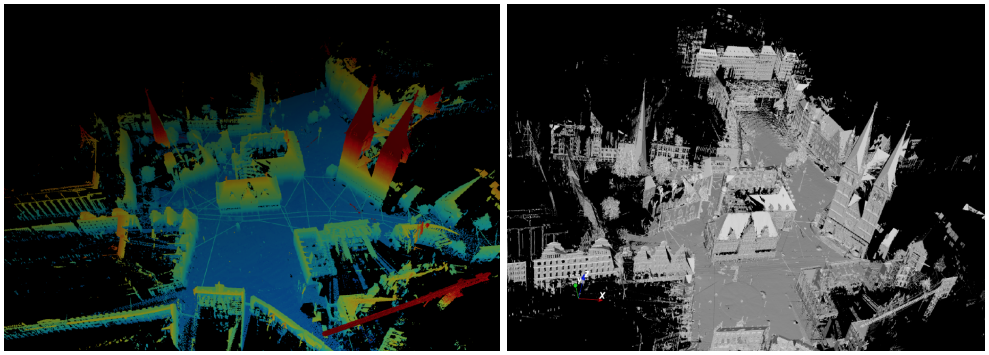

Abb. 4: Der Testdatensatz des Bremer Markplatzes als 3D-Punktwolke (links) und polygonale Rekonstruktion (rechts).

Zunächst wurde anhand dieser Testdaten untersucht, wie sich die oben vorgestellte Interpolation auf die Konsistenz der globalen Rekonstruktion auswirkt. Die Ergebnisse sind in Abbildung 5 dargestellt. Ohne die Interpolation der überlappenden Bereiche liegen die Rekonstruktionen an den Stoßkanten der Teilgitter nicht übereinander und sorgen für eine inkonsistente Rekonstruktion (links). Nach der Interpolation der Distanzwerte in den Überlappungsbereichen liegen die erzeugten Dreiecke genau übereinander.

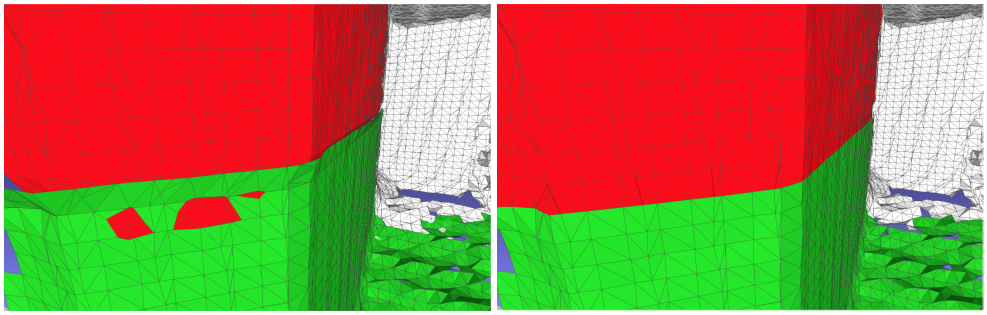

Abb. 5: Vergleich der Rekonstruktionen in den Überlappungsbereichen vor und nach der Interpolation der Distanzwerte.

Darüber hinaus haben wir die Skalierbarkeit unseres Ansatzes mit der Anzahl an zur Verfügung stehenden Slave-Nodes evaluiert. Das Ergebnis für die Rekonstruktion mit und ohne Filterung redundanter Vertices bei einer Zielgröße von 2 Mio. Datenpunkten in den Teilgittern ist in Abbildung 6 dargestellt. Man kann deutlich erkennen, dass die Rechenzeit bei der Rekonstruktion ohne Filterung sehr gut mit der Anzahl der Slave-Nodes skaliert. Die Rechenzeit halbiert sich in etwa bei Verdoppelung der Slavezahl. Dies spricht für eine sehr gute Parallelisierbarkeit des Verfahrens und einen geringen Kommunikationsoverhead in der vorliegenden Implementierung. Mit Filterung der redundanten Vertices sieht das Ergebnis weniger gut aus. Zwar sinkt die effektive Rechenzeit auch hier mit steigender Slavezahl, allerdings sind die Gesamtrechenzeiten größer und das Verfahren skaliert nicht mehr so gut.

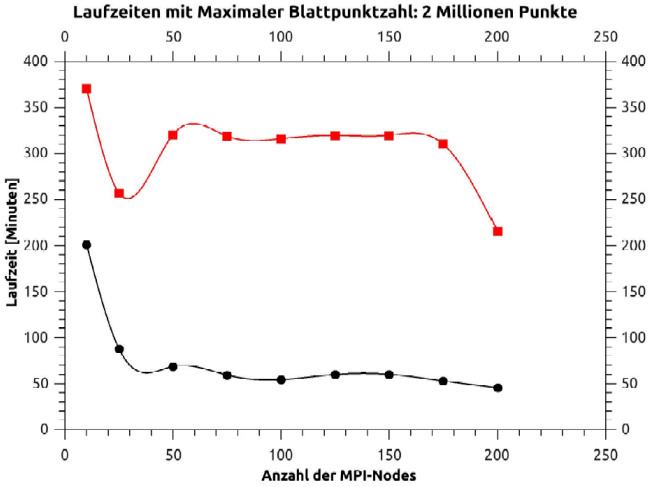

Abb. 6: Laufzeit der Rekonstruktion in Abhängigkeit der zur Verfügung stehenden Slave-Nodes mit und ohne Filterung redundanter Vertices.

Dies ist auf die relativ ineffiziente Suche nach redundanten Vertices in der vorliegenden Implementierung zurückzuführen. Wir planen durch die Integration des direkten Suchverfahrens aus IGELBRINK ET AL (2015), diesen Flaschenhals zu entschärfen.

Literatur

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. & 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

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

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

Autoren:

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

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