Visualisierungsinstitut der Universität Stuttgart

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3443

Extraktion von möglichst regulären Isoflächen zur Vernetzung großer CT-Volumendatensätzen

Thomas Mezger

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Thomas Ertl
Betreuer/in:	Dipl.-Inf. Steffen Frey
Betreuer/in:	Dr. Guido Reina
Beginn am:	2013-01-29
Beendet am:	2013-07-31
CR-Nummer:	I.7.2

Kurzfassung

Anfang des Jahres 2004 wurde von der European Synchrotron Radiation Facility ein Verbundwerkstoff einer Aluminium-Legierung mit Aluminiumoxid-Keramikpartikel holotomographisch gescannt. Die Volumendaten dieser Aufnahme wurden dem Institut für Materialprüfung, Werkstoffkunde und Festigkeitslehre der Universität Stuttgart für ihre Forschung zur Verfügung gestellt. Um finite Elemente Analysen mit den Daten durchzuführen, wird eine Repräsentation der Trennfläche (Iso-Fläche) zwischen der Aluminium-Legierung und den Aluminiumoxid-Keramikpartikeln benötigt. Erweiterte Anforderungen an das Dreiecksnetz der Iso-Flächenrepräsentation hindern den Einsatz bereits etablierter Methoden.
In dieser Arbeit wurde ein Programm entwickelt, welches einen großen Datensatz verarbeiten kann und ein Dreiecksnetz berechnet, das den Anforderungen für einen weiteren Einsatz für Finite Elemente Analysen gerecht wird. Es wird ein Partikelsystem mit einem Energieminimierungsverfahren verwendet, um eine optimale Form und Verteilung der Dreiecke zu erreichen. Dieses Verfahren wird mittels räumlicher Segmentierung und einer parallelen Ausführung beschleunigt, um eine akzeptable Berechnungszeit zu erzielen.
Für die Handhabung großer Datensätze wird die Berechnung in separate räumliche Zellen eingeteilt, welche mittels Ghostlayer-Schichten einen jeweiligen Berechnungsschritt unabhängig voneinander durchführen können.
Die benötigte Qualität des Dreickesnetzes der Iso-Fläche wird mit dem Verfahren dieser Arbeit erreicht. Eine finite Elemente Analyse konnte bereits erfolgreich anhand eines kleineren Ausschnittes des Datensatzes von 2004 durchgeführt werden.

Abstract

At the beginning of the year 2004 the European Synchrotron Radiation Facility holotomographically scanned a composite material of an aluminium alloy and aluminiumoxide ceramic particles. The volumedataset of this scan was provided for research to the Institute for Materials Testing, Materials Science and Strength of Materials from the University of Stuttgart. To run finite element analysis on this data it requires a division surface (Iso-Surface) between the aluminium alloy and the aluminiuoxid particles. Advanced requiremens to the trianglemesh of the Iso-Surface restrain the usage of common known methods.
A software was developed in this paper to handle a large dataset and to calculate a trianglemesh for further use with finite element analysis. Therefore a particlesystem is used with an energy minimizing approach to achieve an trianglemesh with well-shaped triangles. This method is enhanced in speed by segmentation in space and a parallel execution to achieve an acceptable execution time.
The calculation is sperated in space to independent cells with ghostlayers to handle large datasizes. These ghostlayers are exchanged between the cells every pass of the particlesystem to ensure consistent data at any time.
The required quality of the trianglemesh of the Iso-Surface is achieved by the method of this paper. A finite element analysis of a part from the complete dataset of 2004 has already been accomplished.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Algorithmen

1 Einleitung

In einem Experiment der European Synchrotron Radiation Facility (ESRF) wurden Anfang 2004 zwei Aluminium/Aluminiumoxid-Verbundwerkstoffproben (Al/Al_2O_3) holotomographisch gescannt. Auf der Grundlage dieses Experimentes hat sich das Institut für Materialprüfung, Werkstoffkunde und Festigkeitslehre (IMWF) der Universität Stuttgart, vertreten durch Herrn Dr.-Ing. Ulrich Weber, an das Visualisierungsinstitut (VISUS) der Universität Stuttgart gewendet. Ziel dieser Kooperation ist es aus den skalaren Volumendatensätzen vom Experiment der ESRF eine Isoflächenrepräsentation zwischen Aluminium (Al) und Aluminiumoxid (Al_2O_3) zu gewinnen. In weiteren Schritten werden diese als Grundlage für FE-Berechungen (FE = Finite Elemente) und FE-Simulationen verwendet.
Diese Diplomarbeit soll die Grundlagen für die Gewinnung des für die FE-Berechnungen benötigten Dreiecks-Netzes zusammenführen sowie eine erste Implementierung in C++ liefern.
In diesem Kapitel wird zunächst ein Einblick über das vom ESRF durchgeführten Experiment gegeben sowie die hieraus entstandenen Datensätze vorgestellt. Die Anforderungen an FE-Meshes werden besprochen. In Kapitel 2 [Literatur] wird die Theorie des verwendeten Ansatzes zur Berechnung des Dreiecksnetzes präsentiert sowie die nötigen Ergänzungen und verwendeten Beschleunigungsstrategien behandelt. Die Aufteilung der Berechnung in einzelne Einheiten wird in Kapitel 3[Verteilte Berechnung] erläutert. In Kapitel 4 [Programm] wird das aus den Grundlagen entwickelte Programm vorgestellt. Die Ergebnisse der Berechnung werden in Kapitel 5 [Ergebnisse] besprochen. Kapitel 6 [Zusammenfassung und Ausblick] fasst die Ergebnisse der Arbeit zusammen und stellt Anknüpfpunkte vor.

1.1 Experiment IN432

In der ESRF werden Materialproben mit Röntgenstrahlen 3-Dimensional eingescannt. Hierfür wird eine Synchrotron-Strahlungsquelle verwendet. Die von einer Synchrotron-Strahlungsquelle der dritten Generation erzeugten Röntgenstrahlen bieten ein hohes Maß an Kohärenz. Ein solcher Strahl wird, wie in [Abb. 1.1] dargestellt, auf eine Materialprobe geleitet. Beim Durchdringen der Probe wird die planare Wellenstruktur verändert. Durch Variieren des Abstandes zwischen der Materialprobe und dem Detektor ist es möglich sogenannte Phasenbilder aufzuzeichnen. Mit holographischer Rekonstruktion kann anschließend aus den gemessenen Phasenbildern aus verschiedenen Lagen und Richtungen eine dreidimensionale tomographische Volumenrekonstruktion errechnet werden. Eine große Errungenschaft dieser Technik ist die hohe Sensitivität der Messung insbesondere bei Verbundwerkstoffen mit geringen Unterschieden in der gemessenen Dichte der jeweiligen

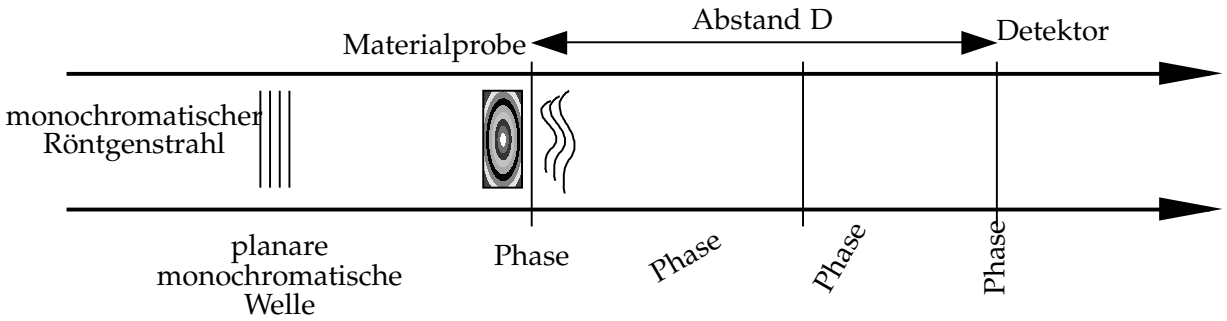


Abbildung 1.1: Schematische Darstellung des Experimentes *IN*432 der ESRF vom 16. Feb 2004

Materialien.
Der schematische Aufbau des Experimentes ist in [Abb. 1.1] beschrieben. Zur Aufzeichnung der Phasenbilder wurde eine CCD FReLoN Kamera mit einer Auflösung von $2048x2048$ und einer $1\mu m$ Optik verwendet. Die hieraus resultierende Pixelgröße beträgt 0.7 μm. Es wurden pro Winkelposition der Materialprobe je drei Scans mit drei verschiedenen Abständen D zwischen Materialprobe und Detektor aufgenommen ($D = 8mm$, $D = 56mm$ und $D = 120mm$).

1.2 Volumendaten

Nach der tomographischen Rekonstruktion ist das Volumen (2048^3) mit Floatwerten codiert. Da pro Float 4 Bytes benötigt werden, würde die Dateigröße des ganzen Volumens $2048^3 * 4 =$ 32Gb betragen. Durch Skalierung des Volumens von Float in 1 Byte pro Voxel, mit den Minimalwerten von 0 und den Maximalwerten von 255, wurde eine Dateigröße von 8Gb erreicht. Das ganze Volumen ist zusätzlich in acht einzelne Dateien entlang der z-Achse geteilt, die jeweils eine Scheibe mit einer Dicke von 256 Voxeln des Volumens abbilden.
In [Abb. 1.2] ist ein Ausschnitt der Volumendaten zu sehen. Gut zu erkennen sind die sehr losen einzelnen Strukturen der Al_2O_3 Partikel, dessen Isofläche zum Al berechnet werden soll.

1.3 Anwender-Kriterien

Berechnungen und Simulationen mit Finite-Elemente-Methoden werden heutzutage in vielen Bereichen eingesetzt. Grundlagen solcher Berechnungen ist die geometrische Aufteilung des Modells in endliche Elemente. Eine solche Geometrieaufteilung kann z. B. durch ein Netz aus Dreiecken (Mesh) der Oberfläche oder durch ein Netz aus Tetraeder des Volumens verwirklicht werden.
Die Qualität der Berechnung, d. h. wie genau die Berechnung abläuft und somit die Simulation die Realität abbildet, hängt stark von der Qualität des Meshes ab. Deswegen ist es wichtig,

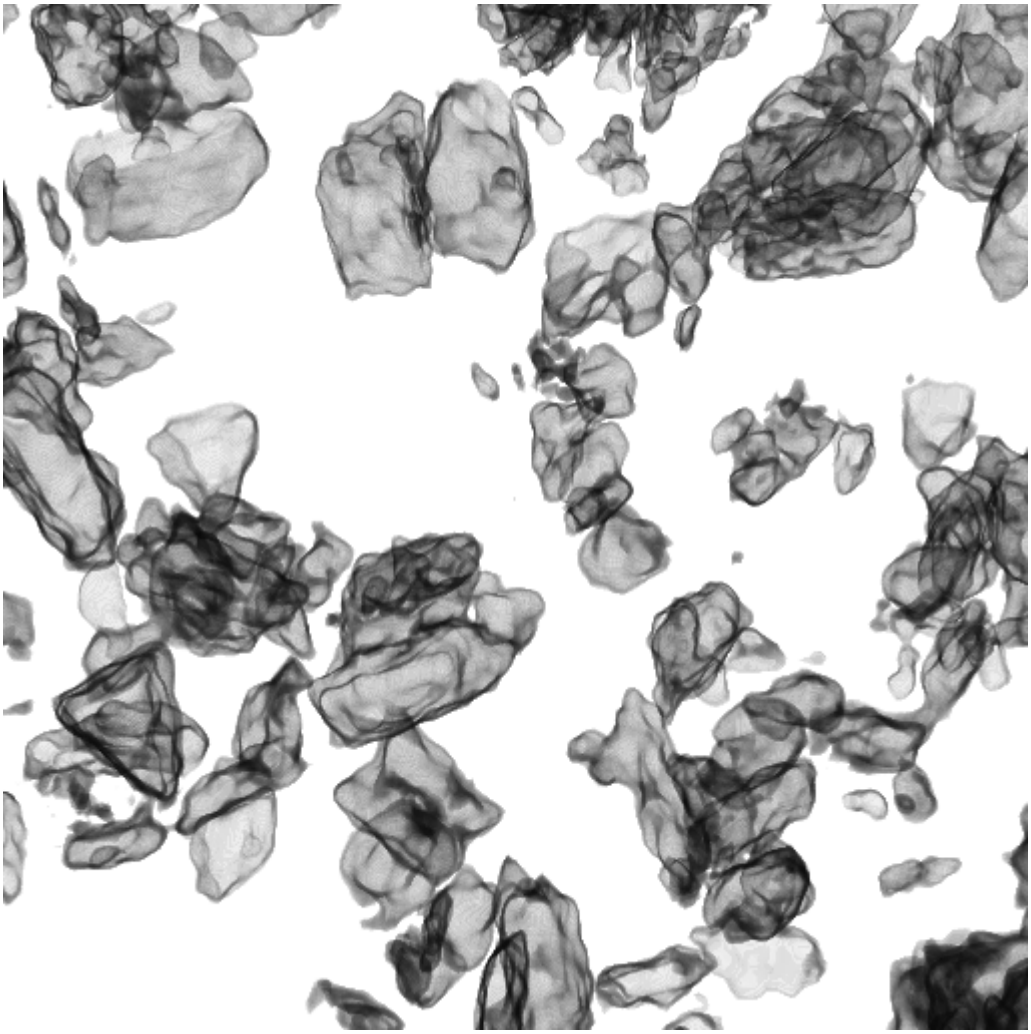


Abbildung 1.2: Raycasting Snapshot eines Ausschnittes (ca 200x200x64) der verwendeten Volumendaten mit 10% Al_2O_3. Dieses Bild wurde mit der Voreen rendering Engine [Uni] erstellt. Es zeigt eine Nahaufnahme einer 64 Voxel breiten Scheibe des Volumens.

dass das aus den Volumendaten extrahierte Mesh folgende Qualitätsmerkmale aufweist. Diese sind :

- möglichst gleichseitige Dreiecke mit 60° Winkel
- möglichst gleich große Dreiecke, maximale Abweichung von 2-facher Kantenlänge

Diese Merkmale sind für die Stabilität des Verfahrens wichtig. Wenn es zu große Abweichungen vom optimalen 60°-Winkel gibt oder die Größen der Dreiecke zu unterschiedlich sind, kann dieses das Ergebnis beeinträchtigen oder gar verfälschen.

2 Literatur

Zur Berechnung eines möglichst regulären Dreiecksnetzes wird das Partikelsystem von [MGW05] mit einer CUDA-Erweiterung von [KCH12] verwendet. Dieses Partikelsystem und dessen Erweiterungen werden in Kapitel 2.2 im Detail behandelt. Zunächst werden alternative Ansätze der Literaturrecherche vorgestellt und mit dem verwendeten Partikelsystem verglichen.

2.1 Bisherige Arbeiten

Es gibt eine Vielzahl an Algorithmen zur Berechnung einer Isofläche. Einige ausgewählte Repräsentanten werden nachfolgend vorgestellt.

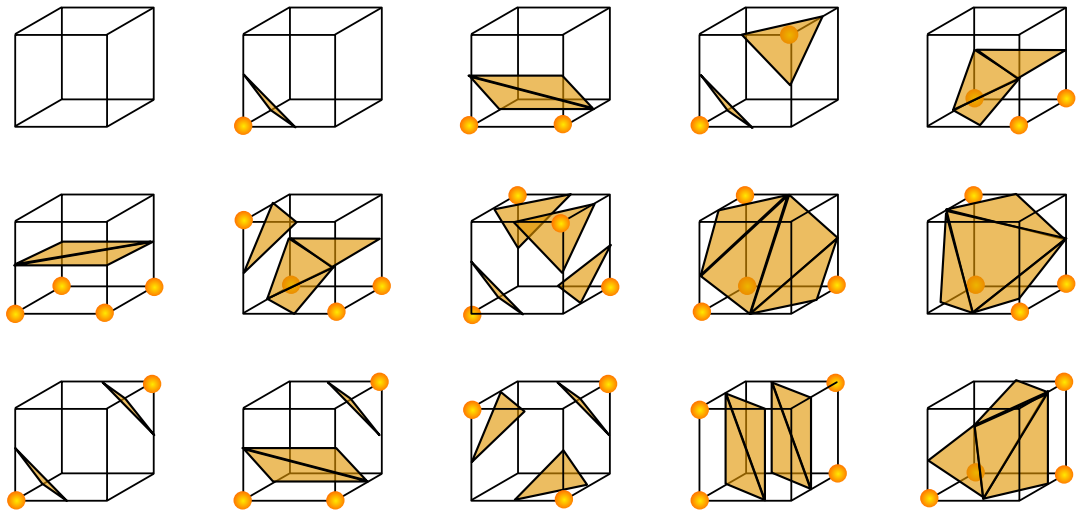


Abbildung 2.1: Alle 15 Möglichkeiten (ohne geometrische Redundanzen) der Aufteilung der Iso-Grenzfläche. Die Ecken der Würfel entsprechen den Voxeln. Gelb markierte Voxel liegen auf der selben Seite der Iso-Fläche. Für skalare Volumendaten werden die Vertexpositionen der Dreiecke zwischen den Voxeln noch linear interpoliert. Grafik von [Jmt].

2.1.1 Marching Cubes

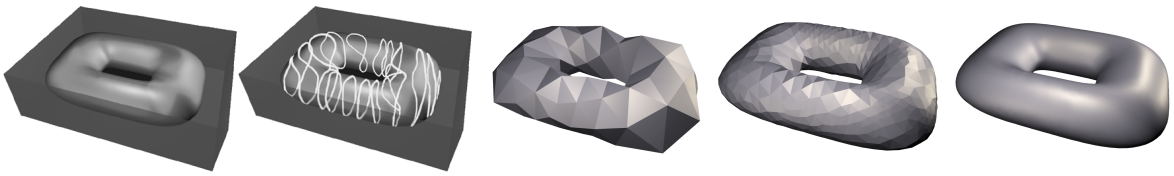
Eine der bekanntesten Arbeiten zur Berechnung einer Iso-Fläche aus einem Volumendatensatz ist Marching Cubes [LC87]. Dieser 1987 entwickelte Algorithmus durchschreitet das Volumen in jedem Voxel. Liegt eine trennende Iso-Fläche zwischen den angrenzenden Voxeln vor, wird eine Dreiecksaufteilung der Grenzschicht innerhalb der angrenzenden 8 Voxel (inkl. Ausgangs-Voxel) errechnet. Dies wird mit Hilfe einer Dreiecks-Lookup-Table der 256 möglichen Aufteilungen und linearer Interpolation, bei skalaren Voxeldaten, der resultierenden Vertices sehr effizient erreicht. Alle möglichen Aufteilungen (ohne Redundanz durch geometrische Symmetrien) sind in Abbildung 2.1 ohne Interpolation der Endpositionen der Vertices aufgeführt. Die resultierenden Dreiecke sind jedoch bei skalaren Voxeldaten, wie sie in diesem Fall vorliegen, oft sehr unförmig hinsichtlich eines möglichst gleichmäßigen Seitenverhältnisses und einer annähernd gleichmäßigen Größe.

2.1.2 Vereinfachungs-Methoden

Eine Methode um eine homogene Verteilung der Vertices auf der Trennfläche und damit ein gleichmäßiges Netz zu erreichen sind Algorithmen bei welchen ein bestehendes, nicht optimales Dreiecksnetz durch Verschmelzen und Teilen von Vertices, Kantentausch zwischen benachbarten Dreiecken und das Einfügen von neuen Vertices verbessert wird. [CMS97] gibt eine Übersicht zu einer Auswahl solcher Methoden. Diese Methoden arbeiten generell mit Qualitätskriteren für die Dreiecksnetzstruktur. Die einzelnen Einheiten, Dreieck, Vertex oder Kanten werden hinsichtlich den Qualitätskriterien verbessert bis eine Verteilung erreicht wurde bei welcher keine Einheit mehr gegen die angegebenen Regeln verstößt. Diese Methoden sind im Vergleich zum verwendeten Partikelsystem schnell in der Ausführung, allerdings qualitativ hinter dem Partikelsystem anzusiedeln.

2.1.3 Verfeinerungs-Ansätze

Ein Verfeinerungs-Ansatz extrahiert zunächst ein sehr grobes Dreiecksnetz des Zielobjektes und verfeinert anschließend iterativ die Dreiecke bis eine gewünschte Genauigkeit im Bezug auf den Fehler der Abweichung von Mesh zum Orginalobjekt erreicht wurde. Die Dreiecksunterteilung kann hierbei mit Dreiecken erfolgen, welche den Qualitätskriterien des Benutzers von vorneherein entsprechen. Ein Beispiel für solch einen Ansatz ist [LHMG02]. Für Datensätze, wie sie in dieser Diplomarbeit behandelt werden, ist ein solcher Algorithmus allerdings nicht praktikabel da für jedes separate Objekt (abgetrennte Iso-Oberfläche) schon auf der gröbsten Ausgangsebene eine getrennte Repräsentation vorhanden sein muss. Ebenso ist die hierarchische Strukturierung für die Zwecke dieser Diplomarbeit nicht relevant. Im Vergleich zum verwendeten Partikelsystem ist der Berechnungsaufwand deutlich geringer. Die Qualität solch einer Unterteilung kann durch eine unvorteilhafte Geometrie des Objektes leiden und ist damit hinter den Ergebnissen des verwendeten Partikelsystems anzusiedeln.



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

Abbildung 2.2: Ablauf der Isoflächenbestimmung mit Ribbons. In (a) sind die Ausgangsdaten zu sehen. (b) stellt die Einteilung des Objektes in die Ribbons dar. Das erste grobe Mesh zwischen den Ribbons ist in (c) zu sehen. Durch weitere Unterteilung entsteht schließlich (d). Das Endergebnis ist in (e) abgebildet. Die Grafiken sind direkt von [WSBDoo] übernommen.

2.1.4 Wavefront-Propagation mit Ribbons

In [WSBDoo] wird zunächst die Isofläche durch repräsentative „Ribbons“ konstruiert. Ausgehend von einem Surfel, einem Voxel welcher von der Isofläche durchschnitten wird, wird über eine Wavefront propagation Technik die Oberfläche durchlaufen und je nach Abstand zum Ausgangs-Surfel in „Ribbons“ geteilt. Diese Ribbons werden im weiteren Prozess miteinander vernetzt wodurch eine erste grobe Approximation der Isofläche entsteht. Anschließend wird das Dreiecksnetz adaptiv verfeinert und mit einem kräftebasierten Löser die Vertices gleichmäßig verteilt. Qualitativ ist das Ergebnis-Mesh mit dem des verwendeten Partikelsystems zu vergleichen. Die benötigte Laufzeit befindet sich in derselben Größenordnung, wie die des Partikelsystems. Für die in dieser Diplomarbeit zugrundeliegenden Ausgangsdaten ist dieses Verfahren jedoch nicht praktikabel, da für jedes einzelne Oberflächenobjekt ein separater Ausgangs-Surfel bestimmt werden müsste und die Behandlung jeder separaten Oberfläche räumlich nicht ohne weiteres getrennt werden kann.

2.2 Verwendeter Ansatz

Diese Diplomarbeit baut auf das Partikelsystem von [KCH12] auf um eine Isoflächen-Repräsentation eines skalaren Grauwert-Volumendatensatzes mit homogen verteilten Vertices zu berechnen. Hierfür wird zunächst eine Isofläche mit MarchingCubes errechnet. Diese dient zur Berechnung des verwendeten Distanzfeldes sowie als Startverteilung der Partikel. Anschließend werden über Kräfte zwischen den Partikeln die Bewegungen dieser einzelnen auf der Isofläche berechnet. Über diverse Korrekturmechanismen werden die Partikel nur auf der Isofläche bewegt. Die Dichte der Verteilung der Partikel wird mit einem weiteren Mechanismus kontrolliert. Somit können sich die Partikel auf der Isofläche im gewünschten Abstand verteilen. Nachdem die Partikel homogen auf der Isofläche verteilt sind werden die Partikel über ein externes Tool [DGG$^+$] (Theorie [DG03]) miteinander vernetzt. Bei idealer Verteilung entsteht durch die Partikelverteilung ein regelmäßiges hexagonales Gitter. Die einzelnen Kontrollmechanismen und die allgemeine Struktur und Verlauf des Berechnungsprozesses werden im Folgenden mathematisch und algorithmisch erläutert.

Algorithmus 2.1 Algorithmus zum parallelen Berechnen des Distanzfeldes

procedure BINPARTS($triangles_{1..m}$,$voxelData_{1..n}$, $isoValue$)
  $maxDistance_{1..n} \leftarrow 2\sqrt{3}$
  $direction_{1..n} \leftarrow 0$
  for $i = 1 \rightarrow m$ **in parallel do**
    $myAABB \leftarrow calculateAABB(triangle_i)$
    for all $j, voxel_j \in myAABB$ **do**
      $atomic(maxDistance_j \leftarrow min(maxDistance_j, calcDistance(voxel_j, triangle_i))$
    end for
  end for
  for $i = 1 \rightarrow m$ **in parallel do**
    $myAABB \leftarrow calculateAABB(triangle_i)$
    for $j, voxel_j \in myAABB$ **do**
      if $maxDistance_j = calcDistance(voxel_j, triangle_i)$ **then**
        $direction_i \leftarrow calcDirectionToTriangle(voxel_j, triangle_i)$
      end if
    end for
  end for
  for $i = 1 \rightarrow n$ **in parallel do**
    if $voxelData_i > isoValue$ **then**
      $maxDistance_i \leftarrow -(maxDistance_i)$
    end if
  end for
end procedure

2.2.1 Distanzfeld

Das Distanzfeld wird für den Kontrollmechanismus der die Partikel während der Bewegung auf der Oberfläche hält verwendet. Ein zusätzliches Normalenvolumen wird für die Isolierung von Bewegungskräften zweier sich auf lokal verschieden Oberflächen befindlichen sich gegenüber liegenden Partikel berechnet. Das Normalenvolumen wird mit einem $3D$ Sobel-Filter aus den Volumendaten errechnet.
Im Folgenden wird das Distanzfeld in einer diskretisierten Form behandelt. Die Auflösung entspricht hierbei der Auflösung des skalaren Volumendatensatzes. Konstruiert wird das diskrete Distanzfeld aus der Isoflächenrepräsentation des MarchingCubes Algorithmus. Die einzelnen Dreiecke der Isofläche werden analog zum Ansatz von [FL08] durchlaufen (siehe Algorithmus 2.1). Dabei wird für jedes Dreieck eine Achsen-orientierte Bounding Box (AABB) konstruiert. Diese AABB wird um einen zusätzlichen konstanten Faktor, der maximal möglichen räumlichen Bewegung eines Partikels des Partikelsystems bei der Bewegung, in jede Richtung erweitert (siehe Abbildung 2.3).
Nun wird für jeden sich in der erweiterten AABB befindlichen Voxel des Distanzfeldes (pro Voxel ein dreidimensionaler Richtungsvektor und einem skalaren Distanzwert) zunächst eine Abstandsberechnung zum Dreieck durchgeführt und im Distanzwert des Voxels des

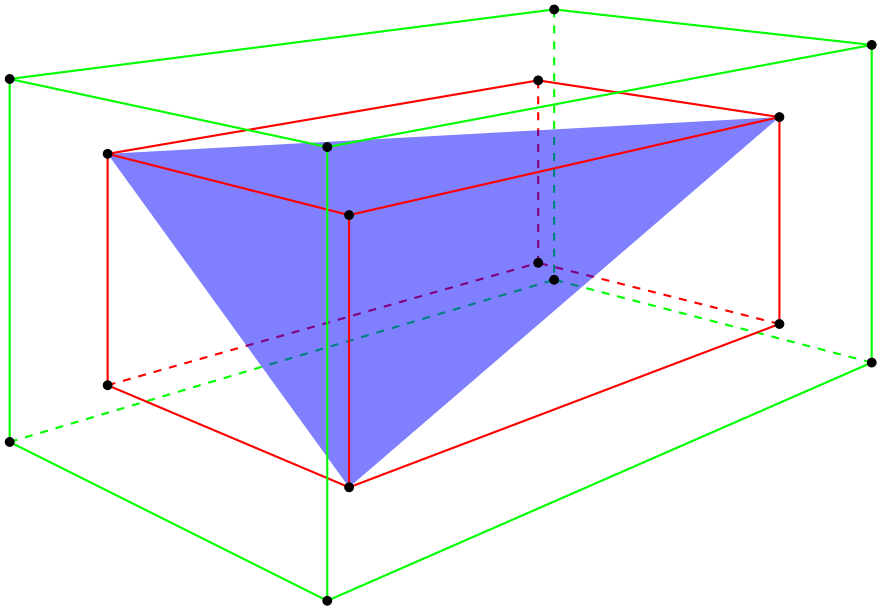


Abbildung 2.3: In blau ist ein Dreieck zur Distanzfeld-Berechnung abgebildet. Die rote Markierung ist die direkte AABB des Dreiecks. In grün ist die erweiterte AABB für die Distanzfeld-Berechnung dargestellt. Der Abstand auf allen Seiten zwischen roter und grüner AABB ist die maximale Strecke die ein Partikel des Partikelsystems aus Abschnitt 2.2.3 zurücklegen kann. Algorithmus aus [FL08].

Distanzfeldes gespeichert, sofern dieser geringer ist als der bisher Eingetragene. Diese Berechnung wird für alle Dreiecke und deren Voxel durchgeführt. In einem zweiten Durchlauf wird nun, wenn der gespeicherte minimale Abstandswert mit dem berechneten identisch ist, die Richtung im Distanzfeld vermerkt. In einem dritten Durchlauf durch die Voxel des Distanzfeldes wird anhand der skalaren Volumendaten für die Voxel des Distanzfeldes das Vorzeichen des Abstandes angepasst. Hierfür wird der skalare Wert für jeden Voxel des Distanzfeldes im zugehörigen Voxel des Roh-Volumens nachgeschlagen. Ist dieser größer als der Isowert wird das Vorzeichen der Entfernung im Distanzvolumen negiert. Das nun vorliegende Distanzfeld enthält in jedem Voxel den geringsten signierten (innen negativ /außen positiv) Abstand und genaue Richtung zur Isofläche, welche vom MarchingCubes Algorithmus berechnet wurde.

2.2.2 Energie-Berechnung

Um die Lage der Partikel untereinander qualitativ beurteilen zu können wurde von [MGW05] ein Energie-System eingeführt. Die globale Energie berechnet sich durch

$$E = \sum_{j=1}^{m} \sum_{i=j+1}^{m} E_{ij}(|d_{ij}|). \tag{2.1}$$

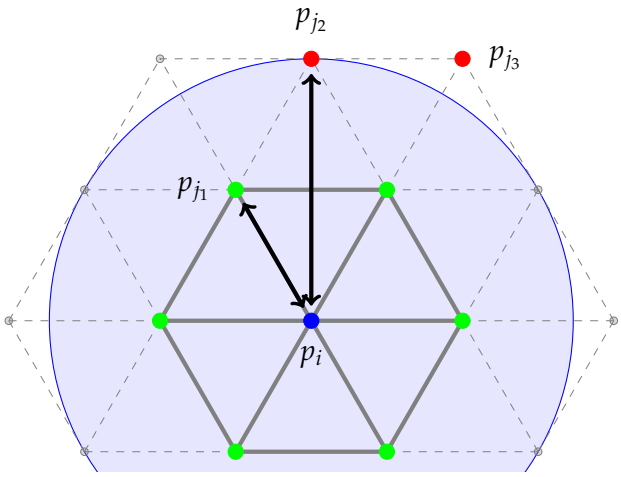


Abbildung 2.4: Eine ideale Partikelverteilung durch ein hexagonales Gitter. p_{j_2} und p_{j_3} sind hierbei außerhalb des Einflussbereiches von p_i mit einem Abstand von $d_{ij_2} = 1$ und $d_{ij_3} = \sqrt{3}$. Der Abstand $d_{ij_1} = \frac{\sqrt{3}}{2}$ zwischen Partikel p_i und p_{j_1} ist ideal. Der blaue Kreis gibt den Einflussbereich des Partikels p_i für dessen Energieberechnung an. Alle grün gefärbten Partikel befinden sich innerhalb, alle anderen außerhalb des Einflussbereiches von p_{i_r}.

E_{ij} ist die Energie zwischen einem Partikelpaar von Partikel p_i und Partikel p_j. Die Symmetrie der paarweisen Energie wurde in Formel 2.1 schon berücksichtigt. m ist die Anzahl der Partikel des gesamten Partikelsystems.

Die Energie E_{ij} eines Partikelpaares berechnet sich aus

$$E_{ij} = \begin{cases} \cot(|d_{ij}| \frac{\pi}{2}) + |d_{ij}| \frac{\pi}{2} - \frac{\pi}{2} & |d_{ij}| < 1 \\ 0 & |d_{ij}| \geq 1 \end{cases} \tag{2.2}$$

mit dem skalierten Abstand zweier Partikel

$$d_{ij} = \frac{|p_i - p_j|}{2 * \cos(\frac{\pi}{6}) * \min(h_i, h_j)} \tag{2.3}$$

wobei $\min(h_i, h_j)$ ein Faktor zur adaptiven Verteilungsdichte der Partikel nach der Krümmung der Isofläche ist. Dieser Krümmungs-Faktor kann für eine uniforme Verteilungsdichte auch durch die ideale Entfernung *idealSize*, wie in Gleichung 2.4, zweier Partikel ersetzt werden. Der Abstand zweier Partikel $|p_i - p_j|$ ist hierbei der euklidsche Abstand.

$$d_{ij} = \frac{|p_i - p_j|}{2 * \cos(\frac{\pi}{6}) * idealSize} \tag{2.4}$$

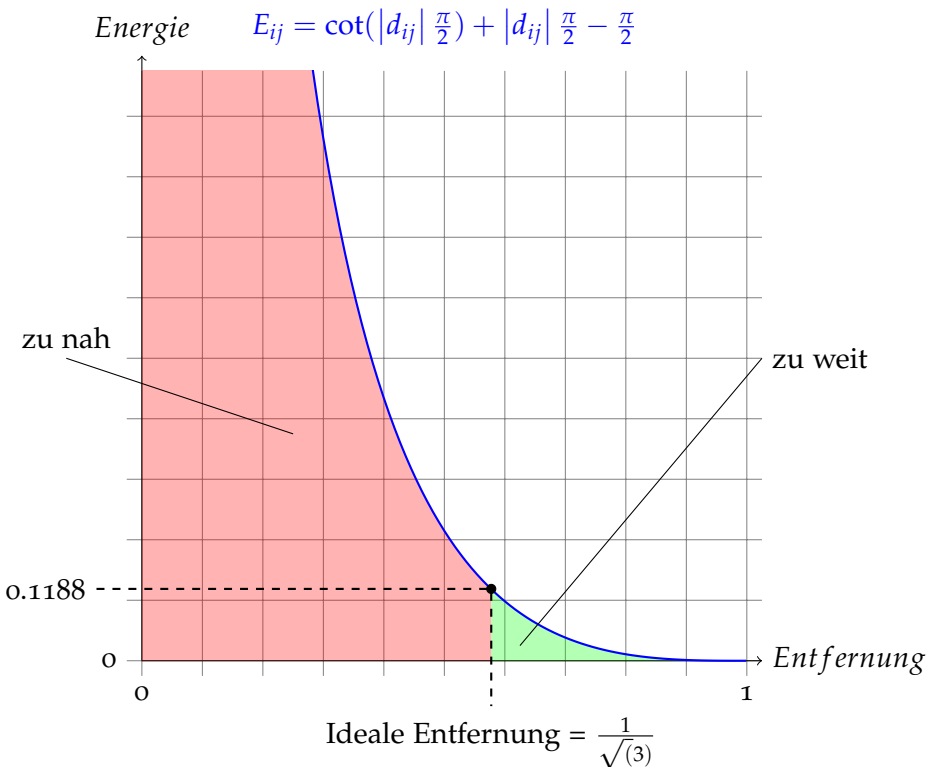


Abbildung 2.5: Ansicht der Energiekurve eines Partikelpaares in Abhängigkeit des Abstandes zueinander. Im rot markierten Bereich befinden sich die Partikel zu nah beieinander. Im grünen Abschnitt sind die Partikel zu weit voneinander entfernt. Der ideale Abstand und dessen Energieniveau sind eingetragen.

Eine ideale Verteilung der Partikel durch ein hexagonales Gitter ist in [Abb. 2.4] zu sehen. Alle grün markierten Partikel werden für die Energieberechnung von P_i berücksichtigt. Alle Partikel außerhalb des Einflussbereichs von P_i werden durch einen Abstand $d_{ij} \geq 1$ nach Formel 2.2 mit einer Energie von 0 bewertet und fließen somit nicht in die Berechnung der Partikelenergie mit ein.
In [Abb. 2.5] wird die Energie abhängig zum Abstand dargestellt. Partikel mit einem Abstand von $d_{ij} \geq 1$ haben keinen Einfluss auf die jeweilige Partikelenergie. Wenn sich die Partikel in einer idealen hexagonalen Anordnung befinden, wie in [Abb. 2.4] vorhanden, ergibt sich daraus eine ideale Partikelenergie von $E_{ideal} = 0.7128$. Auf dieses Energieniveau wird durch die Bewegung (wie in Abschnitt 2.2.3 beschrieben) und Regelung der Dichte der Partikel (wie in Abschnitt 2.2.4 beschrieben) hingearbeitet.

Cosinus Faktor

Bei sich gegenüberliegenden Isoflächen mit einem Abstand der jeweiligen Partikel $d_(p_{iso1}, p_{iso2}) < 1$ nach Formel 2.4 werden diese jeweils in der Energieberechnung und zur Berechnung der Bewegung (siehe 2.2.3) gegenseitig mit berücksichtigt. Dieses Verhalten

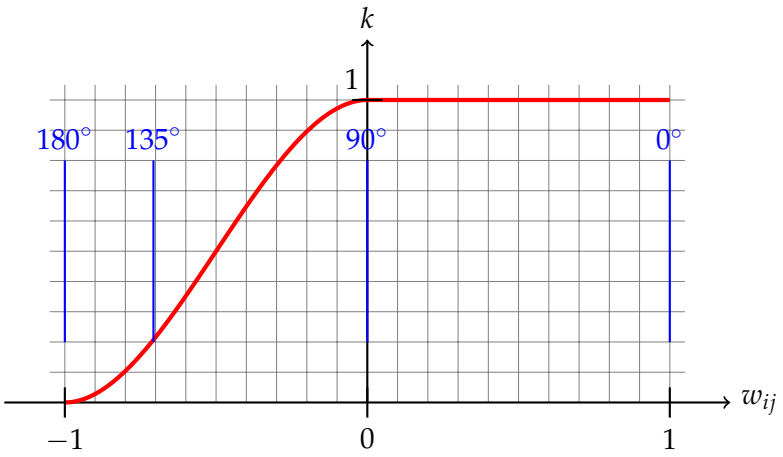


Abbildung 2.6: Funktionsplot von Gleichung 2.5. Stetig differenzierbare Funktion im Definitionsbereich $(0,1)$. $k_{ij}(-1) = 0$, $k_{ij}(0) = 1$. Zum besseren Verständnis wurden einige markante, zu w_{ij} entsprechende Winkel in der Darstellung blau eingefügt

kann zu einem fehlerhaften Ergebnis führen. Im Quellcode der Software SCIRun [Cen], mit einer Implementation von [KCH12], wurde daher ein Cosinus-Faktor der Partikelnormalen eingeführt um diese Beeinträchtigung zu unterbinden.
Dieser Faktor k wird wie folgt über den Winkel α der jeweiligen Normalen der Partikel definiert

$$k_{ij} = \begin{cases} -2w_{ij}^2(w_{ij} + \frac{3}{2}) + 1 & w_{ij} < 0 \\ 1 & w_{ij} \geq 0 \end{cases} \tag{2.5}$$

mit

$$w_{ij} = \frac{n_i \bullet n_j}{|n_i|\,|n_j|} = \cos(\alpha) \tag{2.6}$$

Mit diesem Faktor k werden sich gegenüberliegende Partikel nicht gegenseitig bei der Energieberechnung und damit bei ihrer Verteilung innerhalb ihres Einflussbereiches (siehe Abbildung 2.4) stören. Die Verringerung des Faktors k_{ij} und damit die gegenseitige Beeinflussung bei sich stark gegenüberliegenden Partikeln ist deutlich im Funktionsplot (Abb. 2.6) zu sehen. Damit ergibt sich die neue Partikelpaar-Energie E_{ij} zu

$$E_{ij}^{neu} = E_{ij} * k_{ij}. \tag{2.7}$$

Dies ist die Partikelpaar-Energie auf die, sofern nicht ausdrücklich anderst erwähnt, im weiteren Dokument Bezug genommen wird. Insbesondere gilt dies auch für die Gleichung 2.1.

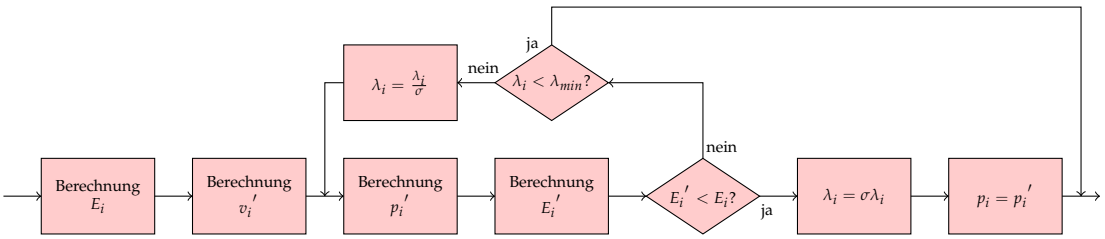


Abbildung 2.7: Übersicht des Bewegungsprozesses eines Partikels p_i.

2.2.3 Bewegung

Berechnung $v_i{}'$

Um eine ideale Verteilung der Partikel wie in [Abb. 2.4] zu erreichen, werden die Partikel im Verhältnis zur ihrer Position zu den anderen Partikeln bewegt. Der sich für einen Partikel p_i ergebende Bewegungsvektor v_i berechnet sich aus den einzelnen paarweisen Bewegungsvektoren v_{ij} zwischen den Partikeln p_i und p_j wie in Formel 2.9 nach [MGW05] und [KCH12] beschrieben.

$$v_i = \sum_{j \neq i} v_{ij} \tag{2.8}$$

mit

$$v_{ij} = -(\tilde{H}_i)^{-1}\left(\frac{\delta E_{ij}}{\delta \left|d_{ij}\right|}\frac{d_{ij}}{\left|d_{ij}\right|}\right) \tag{2.9}$$

und

$$\frac{\delta E_{ij}}{\delta \left|d_{ij}\right|} = \begin{cases} \frac{\pi}{2}\left[1 - \sin^{-2}\left(\left|d_{ij}\right| \frac{\pi}{2}\right)\right] & \left|d_{ij}\right| < 1 \\ 0 & \left|d_{ij}\right| \geq 1 \end{cases} \tag{2.10}$$

Die hessische Matrix H_i wurde entsprechend dem Levenberg-Marquard Algorithmus (L-M) in der Diagonale angepasst und ergibt hier die modifizierte hessische Matrix $\tilde{H}_i$. In jedes Diagonalelement der Matrix H_i wird dabei ein Faktor λ_i eingefügt. Bei jeder neuen Positionsberechnung des Partikels p_i wird dieser Faktor λ_i des Partikels entsprechend angepasst. Da die Berechnung der inversen modifizierten hessischen Matrix zu viel Rechenzeit benötigen würde, wurde diese Berechnung von den Autoren der Literatur [KCH12] und [MGW05] vereinfacht, jedoch nicht im entsprechenden wissenschaftlichen Artikel dokumentiert.

Die Konvergenz-Charakteristik des L-M wurde erhalten indem der Faktor λ_i aus der modifizierten hessischen Matrix direkt vor die Summe der einzelnen Partikelvektoren gezogen wurde. Die inverse hessische Matrix wurde anschließend weggelassen. Daraus ergibt sich der neue Bewegungsvektor $v_i^{'}$ zu

$$v_i^{'} = \lambda_i \sum_{j \neq i} v_{ij}^{'} \tag{2.11}$$

mit

$$v_{ij}^{'} = \frac{\delta E_{ij}}{\delta \left| d_{ij} \right|} \frac{d_{ij}}{\left| d_{ij} \right|}. \tag{2.12}$$

Berechnung $p_i^{''}$

Die neue Position $p_i^{''}$ wird nun durch das Verschieben des alten Punktes p_i um $v_i^{'}$ erreicht. Zusätzlich wird der neu berechnete Punkt $p_i^{''}$ auf die durch den Marching Cubes Algorithmus berechnete Isofläche mithilfe des Distanzfeldes projiziert um die endgültige Position $p_i^{'}$ zu erhalten.

$$p_i^{''} = p_i + v_i^{'} \tag{2.13}$$

$$p_i^{'} = p_i^{''} + n - \frac{(p_{vox} - p_i^{''}) \bullet n_{vox}}{n_{vox} \bullet n_{vox}} n_{vox} \tag{2.14}$$

Zur Projektion wird zunächst der zugehörige Voxel im Distanzfeld berechnet. Anschließend wird die anteilige Strecke des Richtungsvektors zwischen konkreter Voxelposition (Zentrum des Voxelbereiches) und berechneter Partikel Position $p_i^{''}$ berechnet. Zur Projektion wir nun die neue Position nach 2.13 und nach 2.14 auf die Oberfläche projiziert. Somit bleiben die Partikel immer konsistent auf der Isofläche. Das Schema der Projektion wird in [Abb. 2.8] dargestellt.

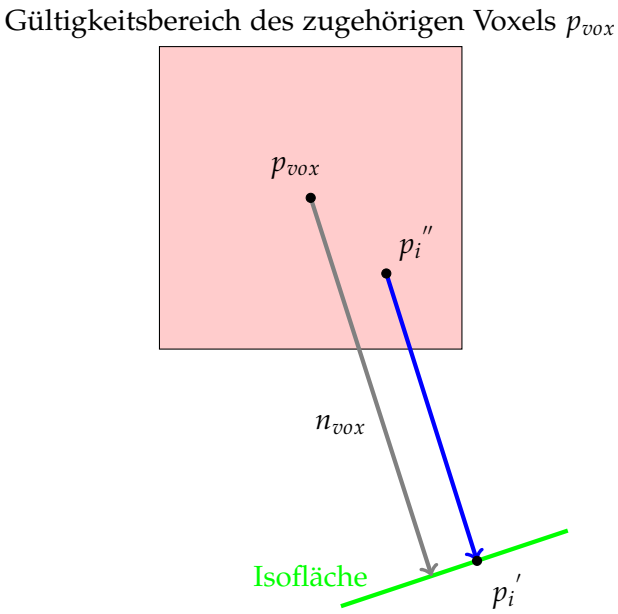


Abbildung 2.8: Beispiel (in $2D$) einer Projektion eines Partikels p_i'' auf die Isofläche mit Hilfe des Distanzvolumens und des zugeordneten Voxels des konkreten Distanzfeldes mit Zentrum p_{vox} und Normalen n_{vox}. Der projezierte Partikel ist p_i'.

Anpassung λ_i

Der Faktor λ_i jedes Partikels wird während dessen Bewegung analog zum L-M Algorithmus angepasst. Wurde ein Partikel erfolgreich an eine neue Position p_i' bewegt, die neue Energie E_i' ist niedriger als an der vorherigen Position p_i mit der Energie E_i, wird λ_i um den Faktor σ erhöht. Dies ergibt eine um σ größere Schrittweite beim nächsten Durchlauf. Wurde in der aktuellen Berechnung kein niedrigeres Energieniveau für die Partikelposition p_i' erreicht, wird λ_i um den Faktor σ verringert. Die Berechnung der neuen Position p_i' wird mit einer durch einen geringeren Faktor λ_i geringere Schrittweite wiederholt. Wurde eine definierte minimale Schrittweite unterschritten und keine neue Position mit einem niedrigeren Energieniveau gefunden, wird der Partikel nicht bewegt.

2.2.4 Kontrolle der Dichte der Partikelverteilung

Da die Anzahl an benötigten Partikel für eine gleichmäßige, den Anwenderkriterien (Abschnitt 1.3) entsprechende, ideale Verteilung nicht a priori bekannt ist und erst im laufenden Anpassungsprozess gefunden werden kann, wird ein Mechanismus benötigt um die Anzahl an Partikel im System zu kontrollieren. Im Partikelsystem von [KCH12] wird das Energieniveau E_i der einzelnen Partikel gegen das bei einer idealen Partikelverteilung vorliegende ideale Energieniveau $E_{ideal} = 0.7128$ (siehe Abschnitt 2.2.2) abgeglichen. Ist die Energie des Partikels P_i mit $E_i < 0.75E_{ideal}$ zu gering und damit die Dichte der Partikel im Umfeld

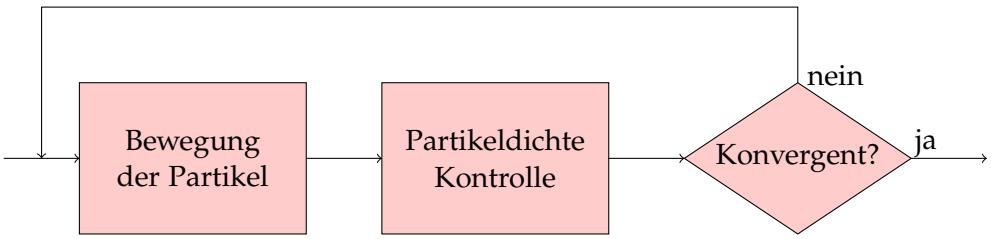


Abbildung 2.9: Ablauf der Berechnung einer optimalen Verteilung der Partikel des verwendeten Partikelsystems

von P_i zu gering wird ein weiterer Partikel eingefügt. Wenn die Energie des Partikels mit $E_i > 1.35E_{ideal}$ zu groß und damit die Partikeldichte zu hoch ist, wird dieser Partikel entfernt. Um ein mehrfaches Einfügen oder Löschen von Partikeln im gleichen Gebiet zu verhindern werden diese Operationen wie in [WH05] eingeführt, von [KCH12] verwendet, mit einem zufälligen Bias versehen.
Zum Einfügen des Partikels wird die Position des Partikels mit zu niedrigem Energielevel als Ausgangspunkt festgelegt. Der neue Partikel wird in Richtung der berechneten Bewegungsrichtung v_i des Ausgangspartikels eingefügt. Hierfür wird analog zum Bewegungsprozess die neue Position berechnet und schrittweise der Lambda-Faktor verkleinert bis ein Energieniveau für den neuen Partikel erreicht wurde, welches unterhalb des kritischen Energieniveaus zum Löschen eines Partikels liegt. Diese Beschränkung verhindert, dass der neue Partikel im nächsten Durchlauf gleich wieder entfernt werden könnte.

2.2.5 Zusammenfassung des Partikelsystems

Der grundlegende Ablauf des Partikelsystems ist in Abbildung 2.9 dargestellt. Die Partikel werden nacheinander bewegt. Hierbei werden alle Partikel bei der Bewegung berücksichtigt. Wenn der Bewegungsprozess für alle Partikel abgeschlossen ist, wird die Dichte der Partikel angepasst. Ist das Partikelsystem nach einem Durchlauf konvergent, haben die Partikel eine homogene Verteilung erreicht und können über ein externes Tool vernetzt werden.
Die naive Umsetzung dieses Partikelsystems ist nicht sehr performant. Dabei müssen für die Energie- und Bewegungsberechnung alle Partikel miteinander verrechnet werden. Eine Verbesserung wird nun im Folgenden besprochen.

2.3 Beschleunigung durch Binning

Wie von [Hec97] eingeführt, können die Partikel des Partikelsystems in eine räumliche Struktur eingeteilt werden, die der maximalen noch zur Berechnung sensiblen Entfernung der Partikel untereinander entspricht. Hierbei wird das Volumen in sogenannte Bins eingeteilt mit einer Kantenlänge die einem Abstand $d_{ij} \geq 1$, also $d_{bin} \geq \sqrt{(3)}idealSize$ entspricht. Damit ist es möglich zu einem Partikel durch die zugrunde liegende Struktur sehr schnell seine Nachbarschaft, also die zur Berechnung der Energie und Bewegungsrichtung relevanten

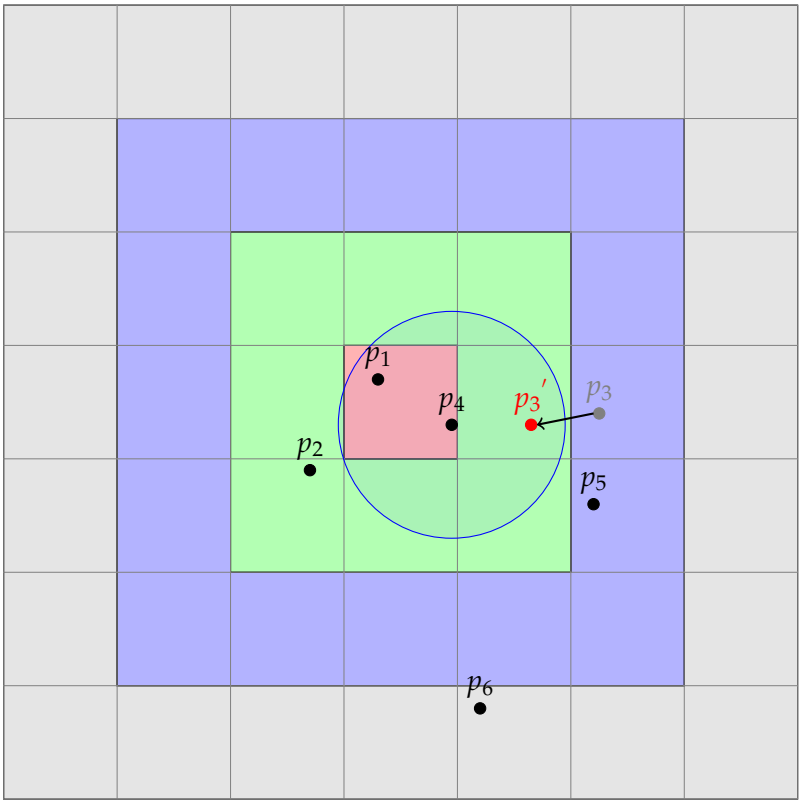


Abbildung 2.10: 2*D* Darstellung der verwendeten Binstruktur. Das rote Quadrat in der Mitte markiert den aktiven Bin, die grünen Bins bilden die unmittelbare Nachbarschaft. Die blauen Bins bilden die erweiterte und notwendige Nachbarschaft. Der blaue Kreis kennzeichnet den Einflussbereich von p_4 bei der Energie- und Bewegungsberechnung. Der rote Partikel $p_3{}'$ wurde von seiner alten Position (grau) in einem vorherigen Schritt an seine jetzige Position verschoben.

Partikel zu bestimmen.
Die Nachbarschaft eines Bins umfasst die umliegenden $5x5x5$ Bins wie in Abbildung 2.10 für 2*D* dargestellt. Die grüne Zone der Nachbarschaft ist notwendig, weil sich dort Partikel befinden, die mit dem Partikel des aktiven Bins (mitte) interagieren. Die blau markierten Bins sind wichtig, wenn die Binstruktur nicht während des Bewegungsprozesses ständig auf den neuesten Stand gebracht wird, sondern nachdem der Bewegungsprozess der Partikel für einen Durchlauf abgeschlossen ist, erneut aufgebaut wird (siehe Abbildung 2.9). Diese getrennte Berechnung der Bewegung der Partikel und dem Einteilen der Partikel in die Binstruktur ist für die Beschleunigung durch parallelisieren (siehe Kapitel 2.4) notwendig.
Wenn die Binstruktur nicht konsistent während des Bewegungsprozesses gehalten wird, kann es vorkommen, dass sich ein Partikel von seinem Ausgangsbin in einen anderen Bin hinein bewegt. Allerdings ist dieser Partikel noch seinem Ausgangsbin zugewiesen. Diese Situation wurde in Abbildung 2.10 mit Partikel p_3 dargestellt. Hierbei hat sich der Partikel p_3 von seinem Ausgangsbin (Mitte, Rechts, Blau) in einen benachbarten Bin (Mitte, Rechts, Grün) an die Position $p_3{}'$ bewegt. Nach dem Bewegen und vor dem erneuten Sortieren der

Partikel in die einzelnen Bins ist dieser Partikel $p_3{}'$ noch in seiner Ausgangszelle verlinkt. Um diesen und andere mögliche Partikel nun bei der Bewegungsberechnung, als Beispiel des Partikels p_4 mit einzuschließen, muss die komplette bis 2 Bins entfernte in Abb. 2.10 blau eingefärbte zwiebelartige Binstruktur mit berücksichtigt werden.
In dieser Abbildung muss der Bin, in dem sich Partikel p_6 befindet, nicht für die Nachbarschaft der Partikel im aktiven roten Bin berücksichtigt werden, weil dieser sich auch bei maximal möglicher Bewegung nicht in den Einflussbereich eines Partikels bewegen kann, welcher sich im aktiven Bin befindet. Dies gilt für alle Partikel in den grau eingefärbten und noch weiter entfernten Bins.
Es ist durchaus möglich, dass Partikel in der Nachbarschaft eines Bins bzw. eines Partikels sind, aber nicht im Einflussbereich der einzelnen zu berechnenden Partikel. In Abbildung 2.10 wird für die Berechnung von p_4 auch der Partikel p_5 berücksichtigt, da dieser sich in der Nachbarschaft von p_4 befindet. Diese immer noch zusätzlichen eigentlich unnötigen Berechnungen werden im Gegenzug zur simplen Beschleunigungsstrategie toleriert.

2.4 Parallelisierung

2.4.1 CUDA

Der in dieser Diplomarbeit verwendete Ansatz, um aus den skalaren Volumendaten ein qualitativ hochwertiges Dreiecksnetz (im Sinne des Anwenders siehe Abschnitt 1.3) zu extrahieren, verlangt sehr viel Rechenaufwand. Für den Datensatz des ERSF beläuft sich dies auf einige Tage mit einer normalen heute üblichen Desktop-Workstation. Um die Berechnung und damit das wissenschaftliche Arbeiten zu beschleunigen wird der grundlegende Algorithmus wie in Kapitel 2.4.2 beschrieben auf eine parallele Berechnung ausgelegt.
NVIDIA bietet mit seinen Grafikkarten und der Sprache CUDA für seine GPGPUs (general purpose graphic processing unit) eine Plattform für paralleles Rechnen. Auf einer Workstation mit einer aktuellen Grafikkarte kann mit GPGPU-Computing schon eine Rechenleistung im Tera-Flop-Bereich erzielt werden. Die Energie-Effizienz von Flops/Watt spricht auch für den Einsatz von Grafikkarten gegenüber einem normalen Rechencluster mit CPUs. Eine Einschränkung von GPGPUs gegenüber CPUs ergibt sich durch den beschränkten Befehlssatz der GPGPUs, welcher aber für einfache numerische Berechnungen nicht weiter hinderlich ist. Durch den Einsatz von CUDA kann die Berechnung des Dreiecksnetzes gegenüber einer Single-CPU-Variante um ca. das 40-fache beschleunigt werden [KCH12].

Architektur

In Abbildung 2.11 ist die Prozessorarchitektur der Fermi-Prozessorgeneration von NVIDIA zu sehen. Dieser Architektur liegt der Cuda Computecapability von 2.0 oder höher zugrunde. Die Implementierung dieser Diplomarbeit ist auf diese minimale Computecapability 2.0

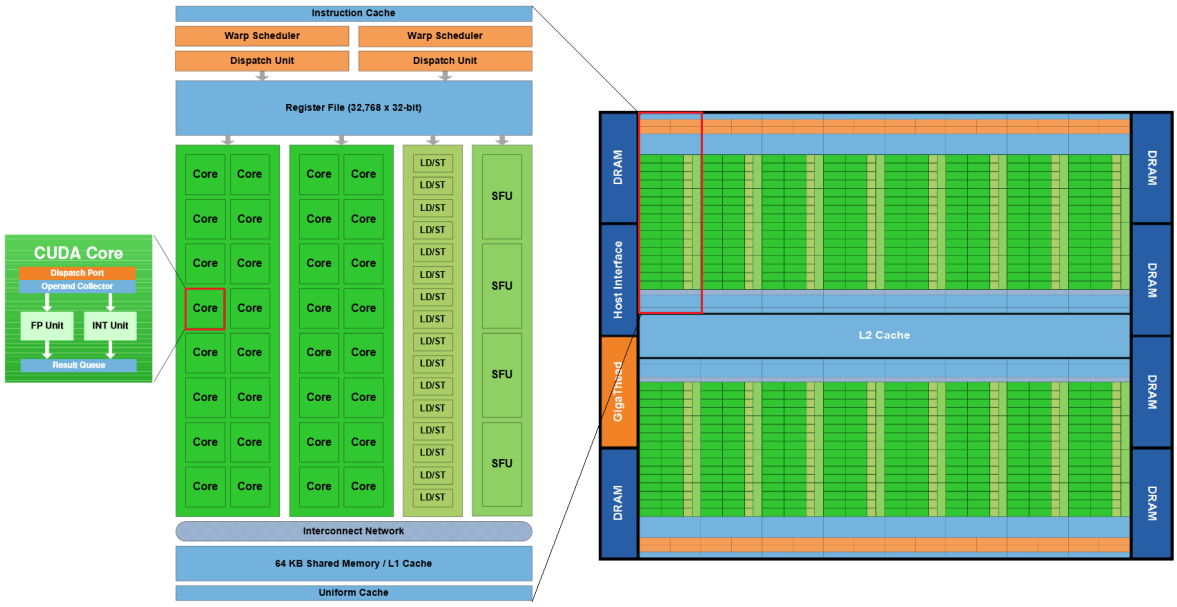
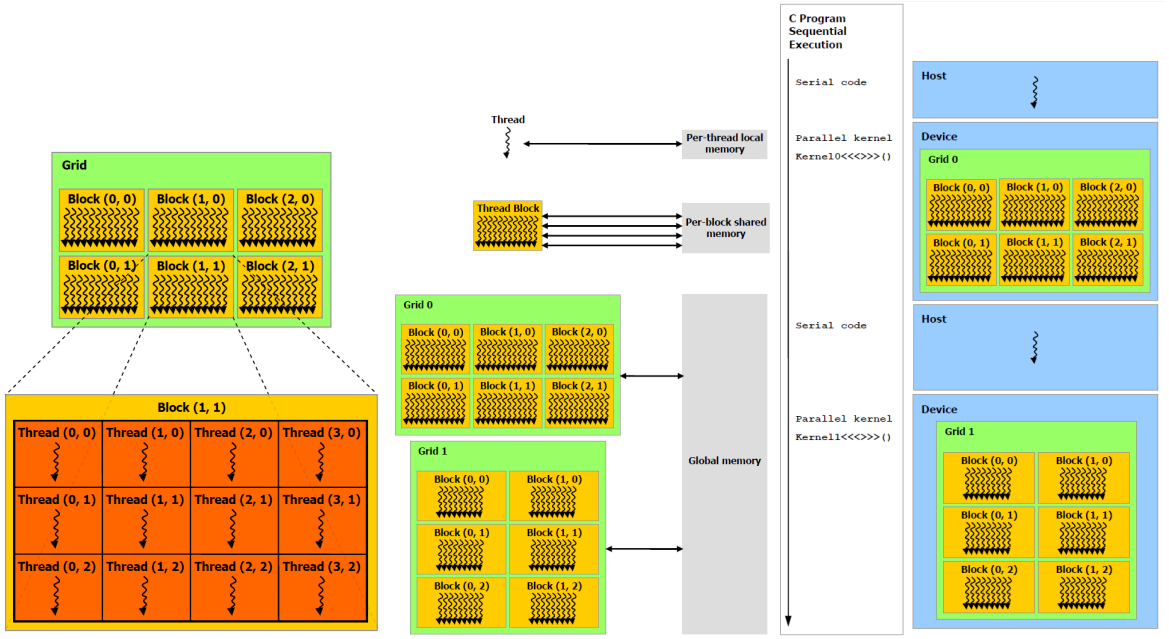


Abbildung 2.11: Schematische Darstellung der Prozessorarchitektur der NVIDIA Fermi Prozessorfamilie. Links ist ein einzelner Rechenkern mit Floatingpoint und Integer Einheit zu sehen. In der Mitte ist ein Streaming Multiprozessor (SM) zu sehen. Auf der rechten Seite ist die Gesamtübersicht des Prozessors mit seinen SMs und dem *L2* Chache abgebildet. Quelle: [NVIb]

ausgelegt.
Es ist in Abbildung 2.11 deutlich die große Anzahl an Cuda Rechenkernen zu sehen, die in den Streaming Multiprozessoren (SM) gruppiert sind. Die neueren Grafikchips der Kepler Architektur unterscheiden sich von diesen durch eine höhere Anzahl an Cuda-Kernen aber mit weniger SM's. Die generelle Aufteilung ist jedoch ähnlich. Jeder Rechenkern innerhalb eines SM's wird in SIMD (single instruction multiple data) ausgeführt. Jeder SM kann unabhängig der anderen SM's seine Cuda-Kerne instruieren. Einem SM wird also eine Gruppe mit Threads zugewiesen, die auf den Cuda-Kernen dann bearbeitet werden. Diese Gruppe mit Threads nennt man einen Thread Block. In Abbildung 2.12 (a) ist die Einteilung der Threads in einem Block (unten) sichtbar, dabei können Threads innerhalb eines Blocks in einem *3D* Gitter angeordnet werden. Zusätzlich können die Threadblöcke in einem 3D Grid angeordnet werden. Diese Blöcke von Threads werden auf den freien SM's der GPGPU verteilt. Im Speichermodell in Abbildung 2.12 (b) ist die Zuordnung der Speicherbereiche zu den einzelnen Threadhierachien dargestellt. Für jeden Thread gibt es einen lokalen Speicher für Register. Jeder Thread-Block hat einen geteilten Speicher für den Austausch innerhalb des Thread-Blocks und zur Synchronisation der Threads. Auf den globalen Speicher kann von allen Threads aller Threadblöcke zugegriffen werden. Threads können nur innerhalb eines Thread-Blocks synchronisiert werden.
Der sequentielle Ablauf eines Cuda-Programmes ist in Abbildung 2.12 zu sehen. Vom seriellen Host-Code (CPU) wird der parallele Device-Code (GPGPU) aufgerufen. Der Ausführung des Device-Codes kann asynchron oder synchron zum Host-Code erfolgen. Ein Flaschen-



(a) Grid/Thread Modell **(b)** Speicher Hierarchie **(c)** Sequentieller Programmablauf

Abbildung 2.12: Architektur und Funktionsprinzip der NVIDIA Cuda Technologie. (a) beschreibt die Einteilung in Grid und Thread-Blöcke, (b) skizziert das Speichermodell der Strukturen und (c) gibt einen Überblick über den sequentiellen Programmablauf mit parallelen Cuda-Berechnungsschritten (Kernel). Die Grafiken sind aus [NVIa] und basieren auf der Fermi Prozessor-Architektur von NVIDIA.

hals bei der GPGPU-Programmierung ist der Transfer der Daten vom Host-Speicher zum Device-Speicher, da die GPGPU nicht auf den Arbeitsspeicher des Host-Systems zugreifen kann. Dabei müssen die benötigten Arbeitsdaten vor der Ausführung des Device-Codes in den globalen Speicher des Devices transferiert werden und das Ergebnis der Berechnung nach der Ausführung wieder zurück vom Device in den Arbeitsspeicher des Host-Systems kopiert werden. Diese Aktion kann auch asynchron ausgeführt werden.

2.4.2 Cuda-Binstruktur

Für die Beschleunigung der Berechnung des Partikelsystems mit Cuda hat [KCH12] die Bin-Struktur aus Abschnitt 2.3 für Cuda strukturiert. Für einen zusammenhängenden Speicherzugriff der Partikel von aufeinanderfolgenden Threads werden aus Performance-Gründen ([NVIa]) die Partikel Bin-weise in den Speicher der Grafikkarte geschrieben. Zunächst werden die Partikel unsortiert vom Host-Speicher in den Device-Speicher kopiert. Während eines parallelen Durchlaufs aller Partikel (siehe Algorithmus 2.2) wird die Anzahl an Partikel pro Bin berechnet. Anschließend werden die Partikel in einem zweiten Durchlauf in einen zusammenhängenden Speicherbereich des Bins, wie in Abbildung 2.13 schematisch dargestellt, in den Speicher geschrieben.

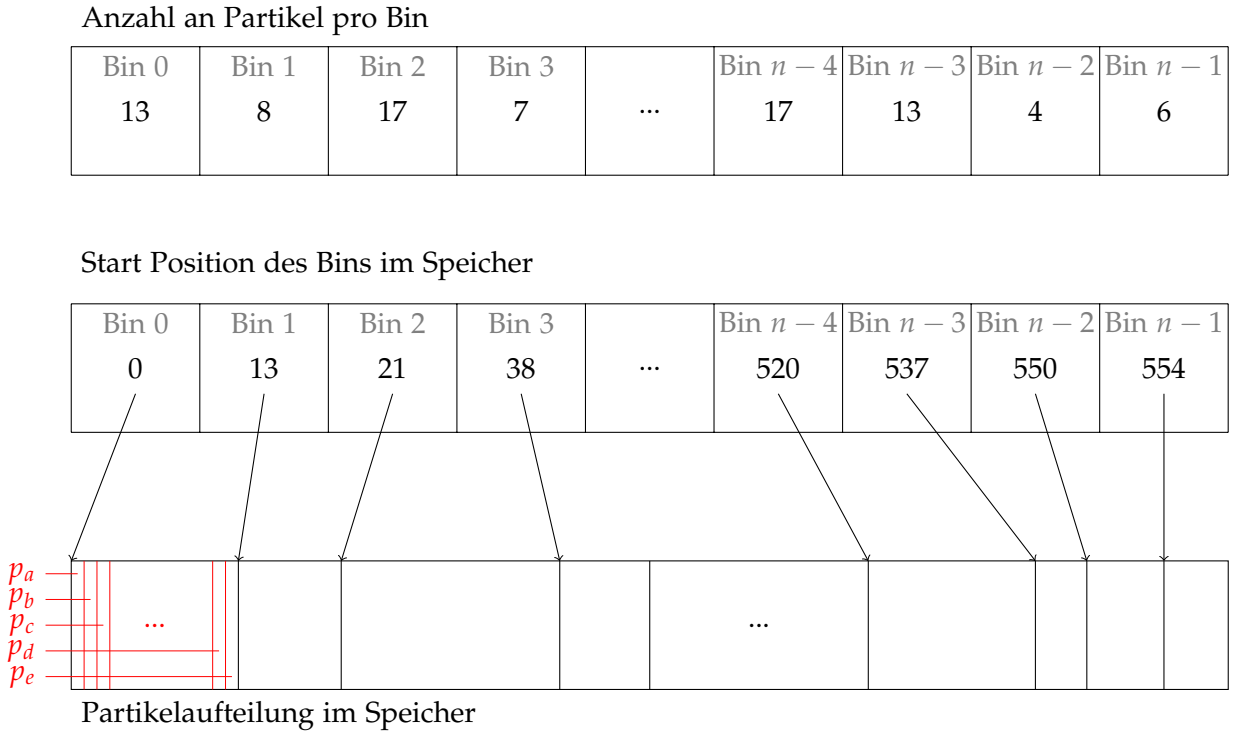


Abbildung 2.13: Bin-Verteilung im Speicher der Grafikkarte wie in [KCH12]. Im Beispiel sind 560 Partikel auf n Bins verteilt. Hierbei spielt die Sortierung der Bins keine Rolle. Die Partikel p_a, p_b, p_c bis p_d und p_e sind dabei räumlich innerhalb des Bins B_0 angeordnet. Die Reihenfolge der Partikel innerhalb eines Bins ist irrelevant.

Algorithmus 2.2 Algorithmus zur parallelen Einteilung von Partikel in Bins.

procedure BINPARTS($p_{1..m}$,$B_{1..n}$)
  $NumParts_{1..n} \leftarrow 0$
  $ActualWriteOffset_{1..n} \leftarrow 0$
  for $i = 1 \rightarrow m$ **in parallel do**
    $j =$ IndexOf(B_j) containig p_i
    $NumParts_j \leftarrow atomic(NumParts_j + 1)$
  end for
  $BinIndex_0 \leftarrow 0$
  for $i = 1 \rightarrow n$ **do**
    $BinIndex_i = BinIndex_{i-1} + NumParts_{i-1}$
  end for
  for $i = 1 \rightarrow m$ **in parallel do**
    $j =$ IndexOf(B_j) containig p_i
    $MyWriteOffset \leftarrow atomic(ActualWriteOffset_j + 1)$
    $Particles[BinIndex_j + MyWriteOffset] \leftarrow p_i$
  end for
end procedure

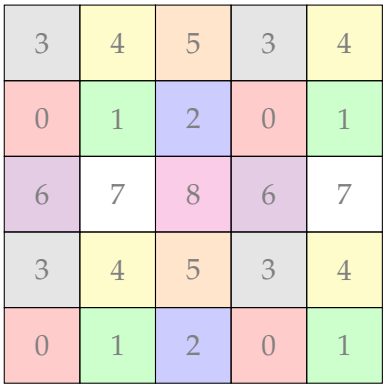
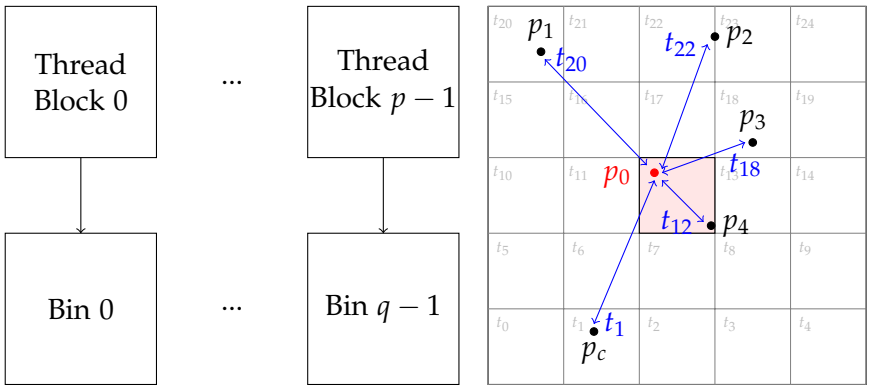


Abbildung 2.14: 2D Darstellung der verwendeten Binstruktur. Aufteilung der Bins in parallel zur Berechnung ausführbare Bins nach [KCH12]. Alle Partikel der Bins mit gleicher Farbe/Nummer können im selben Schritt binweise parallel berechnet werden ohne dass sich Komplikationen durch Nebenläufigkeit in der Berechnung ergeben.

Die Partikel des Partikelsystems werden seriell abgearbeitet. Dies folgt aus dem Energie-Minimierungsproblem des Partikelsystems und der exakten und stabilen Berechnung der Partikelpositionen. Es soll kein Partikel an eine Position bewegt werden, die ein höheres Energieniveau aufweist als die bisherige Position. Daher können die angrenzenden Partikel, welche einen Partikel p_i in seiner Position beeinflussen wegen Nebenläufigkeit in der parallelen Berechnung nicht gleichzeitig bearbeitet werden. [KCH12] hat die parallele Berechnung der Partikel deshalb binweise organisiert. Innerhalb eines Bins werden die Partikel seriell abgearbeitet, die Berechnung der einzelnen paarweisen Energien zu einem Partikel und Bewegungsvektoren werden parallel ausgeführt. Bins die sich nach Abbildung 2.14 nicht beeinflussen, können parallel zueinander abgearbeitet werden. Eine Einteilung der Bins die gleichzeitig abgearbeitet werden können sind in Abbildung 2.14 markiert. In einem 3D-Grid mit Bins ergeben sich daher 27 Einteilungen der Bins. Die Parallelisierung auf Partikelebene erfolgt innerhalb eines Cuda Thread-Blocks. Dabei wird jedem Bin innerhalb der Nachbarschaft des aktiven Bins ein eigener Thread zugewiesen, welcher die Wechselwirkung aller Partikel innerhalb seines Bins mit dem jeweils aktiven Partikel berechnet. Die Parallelisierung der Bins erfolgt mit Thread-Blöcken in einem Grid. Ein Bin wird also jeweils einem Thread-Block und daher einem SM der GPGPU zugewiesen.



(a) Zuteilung Threadblocks - Bins **(b)** Zuteilung Threads - Bins

Abbildung 2.15: In (a) ist die Einteilung der Thread-Blocks (Anzahl p) zu den Bins zu sehen, hierbei werden jeweils nur die aktiven Bins (Anzahl q) mit gleichem Index (siehe Abbildung 2.14) berücksichtigt und auf die Thread-Blöcke verteilt. Idealerweise gilt $p = q$. In (b) ist die Zuteilung der Threads in $2D$ eines Thread-Blocks innerhalb der Nachbarschaft eines Bins zu sehen. Der aktive Bin des zugehörigen Thread-Blocks sowie der gerade aktive Partikel p_0 sind in rot gefärbt. Die Anzahl an Partikeln innerhalb der Nachbarschaft des aktiven Bins beträgt c.

3 Verteilte Berechnung

Heutige Grafikkarten haben eine Speichergröße von zwei bis sechs GigaByte. Die Größe der Ausgangsdaten des ESRF-Projektes beträgt acht Gigabyte. Die zur Berechnung nötigen Daten (Distanzfeld und Partikel) belaufen sich auf etwa 140 GB. Die Herausforderung dieser Diplomarbeit besteht darin, das Gesamtvolumen und deren Berechnung so zu zerlegen, dass die Berechnung einzelner Teile jeweils unabhängig voneinander möglich ist, aber die Berechnung dennoch konsistent zum Vorgehen des Partikelsystems abläuft.

3.1 Aufteilung der Berechnung

Das gesamte Volumen wird daher in einzelne Berechnungs-Zellen eingeteilt, welche über eine sich mit den angrenzenden Zellen überlappende Schicht verfügen. Diese Schicht nennt sich Ghostlayer. Dort sind die Partikel und Strukturen der angrenzenden Zellen enthalten, werden aber nicht aktiv berechnet. Die Stuktur ist in Abbildung 3.1 für zwei benachbarte Zellen dargestellt. Die aktive Zelle ist in rot gefärbt. Dies ist der Kern der Zelle. Die Partikel im Kern werden, wie in den Abschnitten zuvor erläutert, berechnet. Der Kern der benachbarten Zelle ist in blau gefärbt. Zur besseren Ansicht wurde ein Bin-Gitter in diese angrenzende Zelle eingezeichnet.
Die Auflösung an Bins pro Zelle richtet sich nach dem verfügbaren Speicher auf der Grafikkarte. Die grüne Zelle mit Ghostlayern hat eine Auflösung von $(sizeCore_x + 4, sizeCore_y + 4, sizeCore_z + 4)$. Die Breite der Ghostlayer bestimmt sich durch die zwei jeweils zusätzlich nötigen Bins, bei einer Berechnung eines Bins (siehe Abschnitt 2.3) an den Außenseiten des Zellkerns.

Vor der Berechnung innerhalb einer Zelle wird die Ghostlayer-Schicht der Zelle aufgebaut. Hierfür werden bei allen umliegenden Zellen die Partikel in den entsprechenden Gebieten kopiert. Es müssen alle direkt angrenzenden Zellen berücksichtigt werden. Die Zellen werden, analog zu den Bins, nacheinander berechnet um Fehler im Partikelsystem zu vermeiden. Der Ablauf der Berechnung einer Zelle ist in Algorithmus 3.1 dargestellt.

3.2 Zellübergreifende Partikelbewegung

Da sich die Partikel über die Zellgenzen hinweg bewegen können, müssen solche Partikel nach der Berechnung einer Zelle gesondert behandelt werden. Hierfür werden die Positionen der einzelnen Partikel überprüft. Liegt ein Partikel außerhalb des Zellkerns wird dieser in

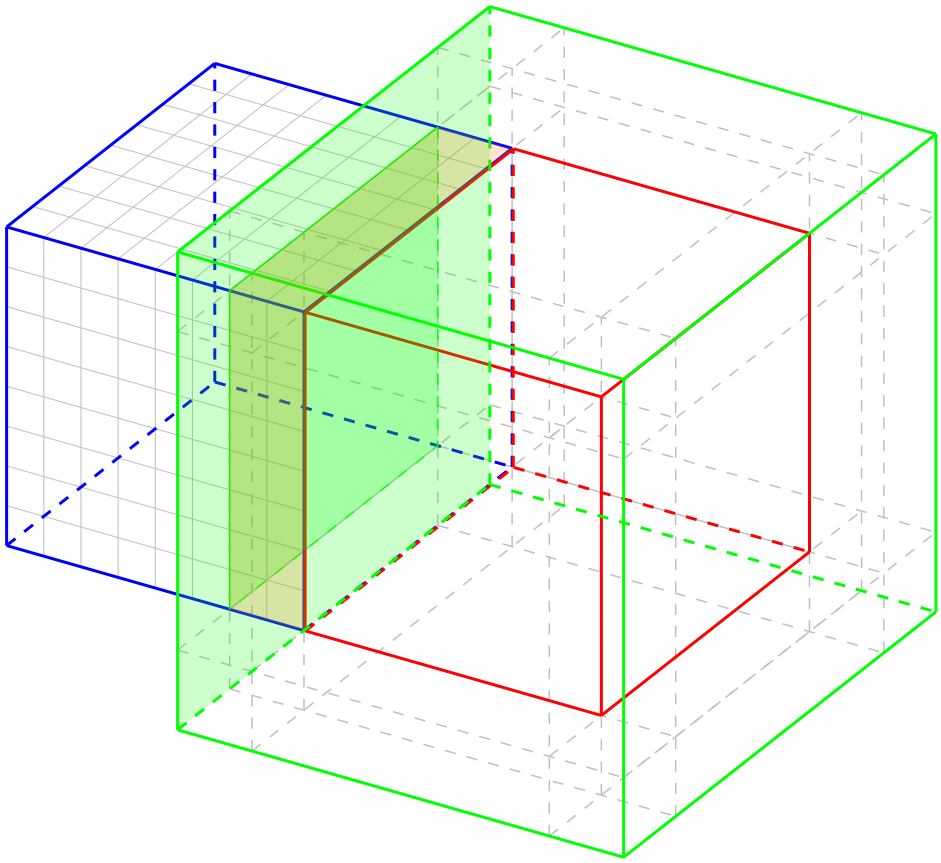


Abbildung 3.1: Eine aktive Berechnungszelle der Aufteilung der Berechnung (rot) und eine benachbarte Zelle (blau). Für die blaue Zelle wurde ein Bin-Gitter eingezeichnet. Die umhüllende grüne Zelle der aktiven roten Zelle ist das gesamt zur Berechnung notwendige Volumen mit Ghostlayer.

Algorithmus 3.1 Algorithmus der Berechnung einer Zelle.

```
procedure CalculateCell(p_{1..m})
    for x = 1 → 3 do
        for y = 1 → 3 do
            for z = 1 → 3 do
                ghostParticles ← ghostParticles + getGhostParticles(Cell_{x,y,z})
                addNewParticlesToCell(getExternParticles(Cell_{x,y,z}))
            end for
        end for
    end for
    calculateCoreBins()
    externParticles ← getParticlesMovedOutOfCoreVolume()
    delete ghostParticles
end procedure
```

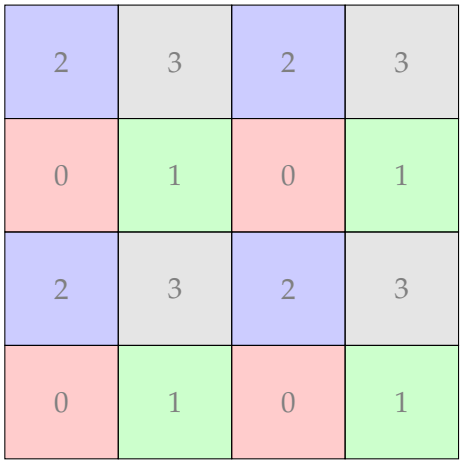


Abbildung 3.2: 2D Darstellung der Einteilung der Zellen in unabhängige Berechnungsgruppen.

einem gesonderten Bereich gespeichert. Ähnlich wie beim Kopieren der Ghostlayer werden die Partikel in der Vorbereitung zur Berechnung von den umliegenden Zellen kopiert. Da die Partikel sich nur in die direkten Nachbarzellen innerhalb eines Berechnungsvorganges bewegen können, reichen diese Zellen aus. Der Austausch der Partikel zwischen Zellen ist in Algorithmus 3.1 dargestellt.

3.3 Parallelisierung der Zell-Berechnung

Die Berechnung diverser Zellen kann unabhängig voneinander stattfinden. In Abbildung 3.2 ist eine Einteilung für den 2D Fall gegeben. In 3D sind es acht verschiedene unabhängige Gruppen. Die Einteilung in diese Gruppen ergibt sich aus der Nutzung von Ghostlayern welche den aktuellen Stand der Partikel in den umliegenden Zellen überträgt. Würde sich während der Berechnung der Partikel des Zellkerns die Verteilung der Partikel eines benachbarten Zellkerns ändern, so wäre die Übergangszone inkonsistent und führt zu falschen Berechnungen an den Zellgrenzen. Daher sind die direkt angenzenden Zellkerne während einer Berechnung des aktiven Zellkerns gesperrt. Die Koordination der Berechnung der einzelnen Zellen wird wie in Algorithmus 3.2 durchgeführt. Jede Zelle hat anhand ihrer Position eine Indentifikationsnummer (ID) zwischen $0 \leq Cell_{j_{1D}} < 8$ analog zu Abbildung 3.2 für den 3D-Fall.

3.4 Parallelisierung GPU/CPU

Um eine optimale Ausnutzung von CPU und GPU zu erreichen werden die Berechnungen der Zellen nach den vorhandenen Ressourcen verteilt. Es werden alle zur Berechnung parallel ausführbaren Zellen (mit gleichem Berechnungsindex siehe [Abb. 3.2]) in einem

Algorithmus 3.2 Algorithmus der verteilen Berechnung der Zellen.

```
procedure CalculateMesh(Cell_{1..n})
    repeat
        for i = 0 → 7 do
            for j = 1 → n in parallel do
                if Cell_{jID} = i then
                    CalculateCell(Cell_j)
                end if
            end for
        end for
    until convergent(GlobalEnergy(ParticleSystem))
end procedure
```

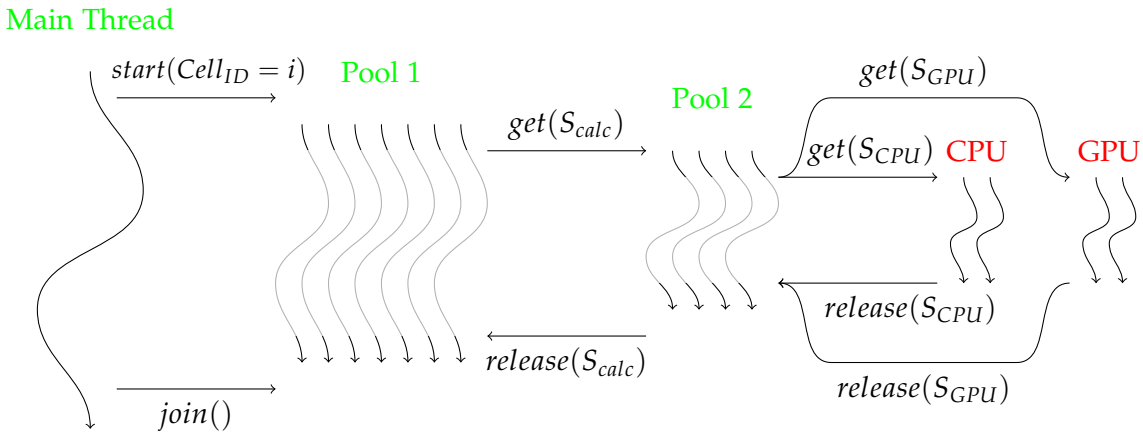


Abbildung 3.3: Schematischer Ablauf des Threadings. Pool 1 beinhaltet alle Ausführbaren Zellen mit gleicher Zell-ID (siehe [Abb. 3.2). Pool 2 beinhaltet n Threads die um die vorhandenen Ressourcen konkurrieren. Der Main-Thread wartet auf das Beenden der von ihm gestarteten Threads ($join()$).

eigenen Thread gestartet. Damit die konkurrierenden Threads sich nicht gegenseitig blockieren und beim warten Speicher belegen wird ihre Ausführung durch eine zählende Semaphore auf die maximale Anzahl an zur Verfügung stehenden Berechnungsressourcen ($CPUKerne + GPUs = n$) begrenzt.
Für die einzelnen Berechnungen des Distanzfeldes, dem Binning, der Bewegung und der Kontrolle der Population werden jeweils getrennt die Ressourcen über zwei zählende Semaphoren je Ressource angefragt. Bei der Berechnung der Bewegung und der Populationskontrolle wird zunächst die GPU-Ressource angefragt, ist dies nicht möglich wird die CPU-Ressource angefragt. Dieser Vorgang wird solange wiederholt, bis eine der Ressourcen zur Verfügung steht. Die Anfragen für das Binning und die Berechnung des Distanzfeldes erfolgt blockierend an die GPU. In [Abb. 3.3] ist die Threading-Struktur schematisch dargestellt. Die wiederholten Zugriffe einzelner Threads aus Thread-Pool 2 auf die beiden Ressourcen CPU und GPU wurden aus Gründen der Übersichtlichkeit zusammengefasst.

Die Anzahl der verfügbaren Ressourcen wird durch das Programm bestimmt und kann durch den Anwender weiter begrenzt werden.

4 Programm

Das Verfahren zum Erstellen eines möglichst regelmäßigen Dreiecksnetzes aus Volumendaten, wie vorangegangen vorgestellt, wurde als Erweiterung des Plugins TriSoup vom Projekt MegaMol [Meg] erstellt. Das Projekt und die Software werden zunächst vorgestellt.

4.1 MegaMol und Trisoup

MegaMol ist eine Software zur Visualisierung von punktbasierten molekularen Datensätzen. Das Projekt wurde 2006 im Rahmen des Sonderforschungsbereiches 716 an der Universität Stuttgart vom Viusalisierungsinstitut begonnen. Die MegaMol Middleware ist aus verschiedenen Modulen zusammengesetzt. Die einzelnen Module werden erst zur Laufzeit miteinander verlinkt.
Zur Visualisierung wird das Modul View3D verwendet, dieses wird vom Modul TriSoupRenderer mit der Darstellung der Dreiecke versorgt. Das Modul TriSoupRenderer fordert die zu Berechnenden Dreiecke vom Modul TriangleData an. Dieser Ablauf ist in Abbildung 4.1 schematisch dargestellt. Ein interaktives Interface zum Ändern von Parameter ist bereits über die AntTweakBar [ant] integriert. Die Benutzeroberfläche ist in Abbildung 4.2 mit der AntTweakBar und den vom Anwender zu wählenden Parametern dargestellt. Die Präsentation der Daten kann über bereits vorhandene Parameter in der AntTweakBar auf die eigenen Bedürfnisse eingestellt werden.

4.2 Programmablauf

Da der Iso-Wert des Datensatzes für die Iso-Fläche zur Trennung zweier Materialien nicht von vornherein festgelegt ist, muss dieser Parameter vom Anwender vor dem Beginn der Be-

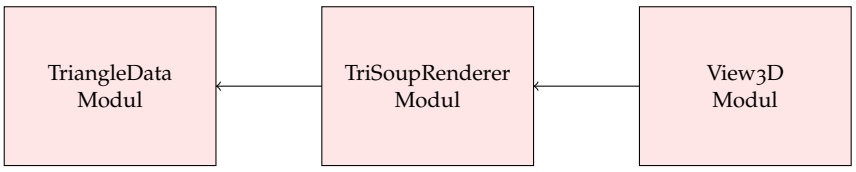
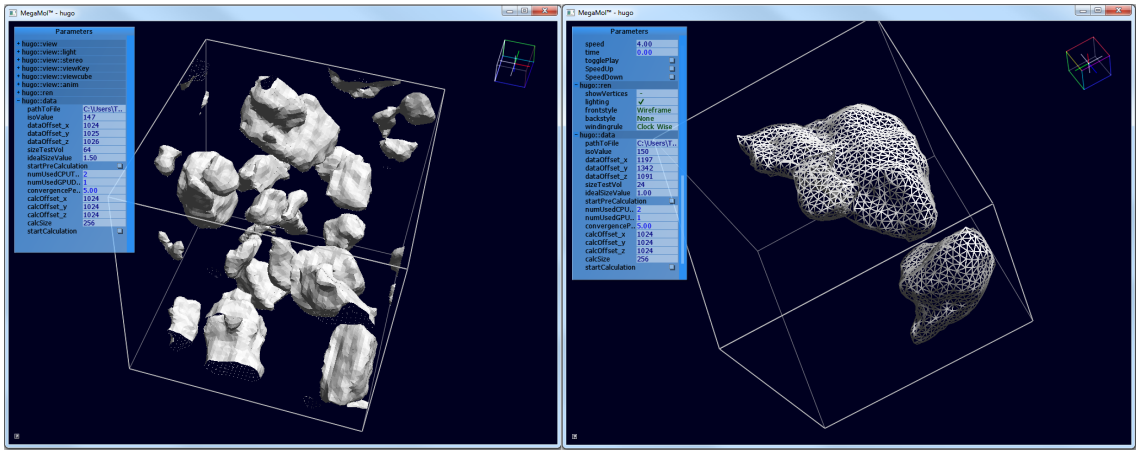


Abbildung 4.1: Modularer Aufbau des Programmes. View3D zeigt die Daten des TriSoupRenderers an. Der TriSoupRenderer fordert die Daten von dem Modul TriangleData an. In TriangleData wird das Mesh zum Anzeigen berechnet.



(a) **(b)**

Abbildung 4.2: Screenshot des Programms MegaMol mit dem Plugin TriSoup. (a) ist die Ansicht beim Start und während der Wahl des Iso-Wertes. In (b) wurde bereits ein Testmesh berechnet, die Oberfläche wird als Drahtgitter angezeigt.

rechnung manuell bestimmt werden. Für diesen Prozess ist es notwendig, dass der Anwender ein direktes Feedback über die Position der Iso-Fläche zum Verhältnis des Iso-Wertes erhält. Um dieses direkte Feedback zu gewährleisten wird für die Berechnung der Iso-Fläche der Marching Cubes Algorithmus von [LC87] verwendet. Die berechnete Iso-Fläche dient dabei lediglich zur Wahl des richtigen Iso-Wertes um die Grenzfläche zwischen den Werkstoffen zu identifizieren. Die dabei errechneten Dreiecke unterstehen daher nicht den Qualitätskriterien (siehe Kapitel 1.3) des Anwenders. Um ein interaktives Feedback ($> 1 \frac{Frame}{Sekunde}$) zu gewährleisten, wird zum beschleunigen des Feedback-Prozesses die Berechnung hoch parallel auf einer GPGPU ausgeführt.

4.2.1 Parameter bestimmen

Der voreingestellte Startwert für den Iso-Wert für Daten mit Werten zwischen 0 und 256 wurde auf 128 festgelegt. Dies sollte dem Anwender eine erste Übersicht der im Volumen enthaltenen Objekten geben. Für die Anpassung des Iso-Wertes wird ein Regler betätigt mit dem sich der Iso-Wert zwischen den Minimal- und Maximalwerten bewegen lässt. Bei jeder Änderung des Iso-Wertes durch den Anwender wird sofort eine erneute Berechnung durchgeführt und dem Anwender präsentiert. Dieser Prozess ist in Abbildung 4.3 dargestellt. Dadurch erhält der Anwender direkt ein Feedback über die Auswirkung der Parameteränderung. Diesen Vorgang wiederholt der Anwender, bis er mit der Trennfläche zwischen den Objekten im Volumen zufrieden ist. Es ist auch möglich den Ausschnitt für das Feedback im Volumen interaktiv zu verschieben und die Größe anzupassen um ein Gebiet im Volumen mit für den Anwender relevanten Daten zu observieren. Nachdem ein Iso-Wert bestimmt wurde kann der Anwender, in einem zum vorherigen Prozess analogen Ablauf,

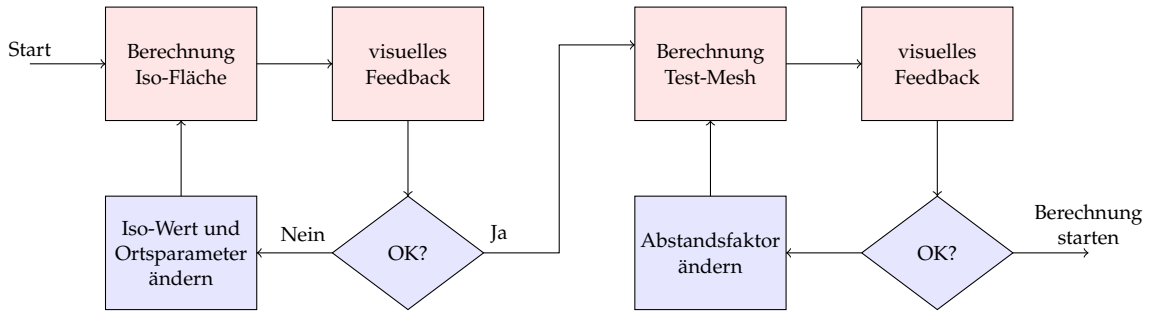


Abbildung 4.3: Prozess zum bestimmen der Betriebsparameter. Dafür werden vom Anwender der Iso-Wert und die Ortsparameter des Ausschnittes sowie die angestrebte ideale Größe der Dreiecke innerhalb des Volumens des Datensatzes interaktiv angepasst. In blau sind die Aktionen des Anwenders, in rot die Prozesse des Computers dargestellt.

die angestrebte ideale Kantenlänge der Dreiecke im Verhältnis zur Größe der Auflösung der Ausgangsdaten bestimmen. Hierbei entspricht ein Faktor von 1.0 dem Abstand zweier Voxel des Ausgangsvolumens. Dieser Vorgang benötigt Zeit bei der Berechnung weshalb nur ein Ausschnitt der Daten betrachtet wird, um das Feedback für den Anwender schnell zu erhalten.

4.2.2 Berechnung des Meshes

Ist der Anwender mit der Qualität des berechneten Dreiecksnetzes der Iso-Fläche des Test-Volumens zufrieden, kann er die Berechnung des Meshes von einer Position im Volumen mit beliebiger, den Grenzen der Daten entsprechenden Größe starten. Das Programm beginnt dann die Daten in die Berechnungszellen aufzuteilen und diese, wie in Kapitel 3 beschrieben, zu berechnen. Nachdem der Vorgang abgeschlossen ist, liegt das Mesh in einem **.OFF*-Format vor. Dieses kann dann vom Anwender unabhängig von MegaMol verwendet werden. Im weiteren wurde ein Formatwandler vom **.OFF*-Format zum *Patran Normal File*-Format entwickelt, welcher die weitere Verwendung des Meshes von Finite Elementen Analysen mit der Software *Patran* [MSC] ermöglicht.

5 Ergebnisse

In diesem Kapitel werden die Ergebnisse der Diplomarbeit behandelt. Da für den Anwender die Qualität des Meshes wichtig ist, werden die Ergebnisse speziell in dieser Hinsicht nach dessen Kriterien beurteilt. Für die Berechnung des Meshes wird der Ressourcenverbrauch bestimmt und die Laufzeiten des Programmes der sequentiellen und Zell-basierten Berechnung miteinander verglichen.

5.1 Mesh-Qualität

Um die Qualität der Form der Dreiecke zu bestimmen werden die Winkel der Dreiecke berechnet und in einem Histogramm dargestellt. Für diese Auswertung wurden eine Berech-

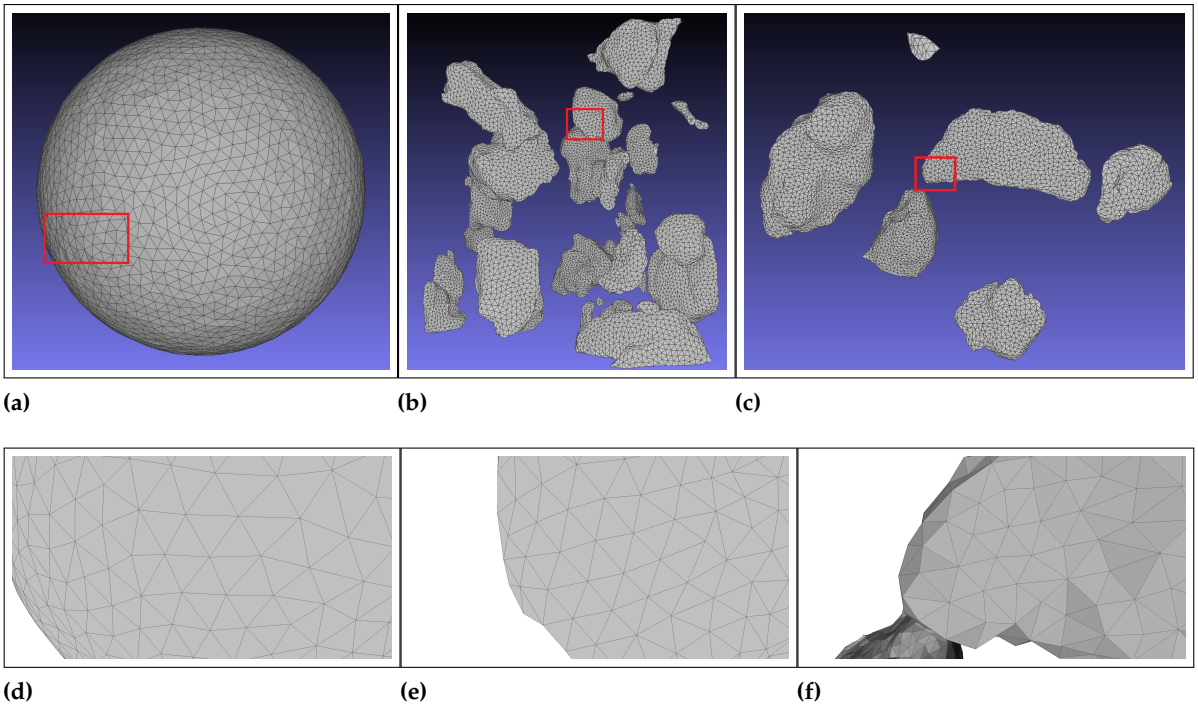
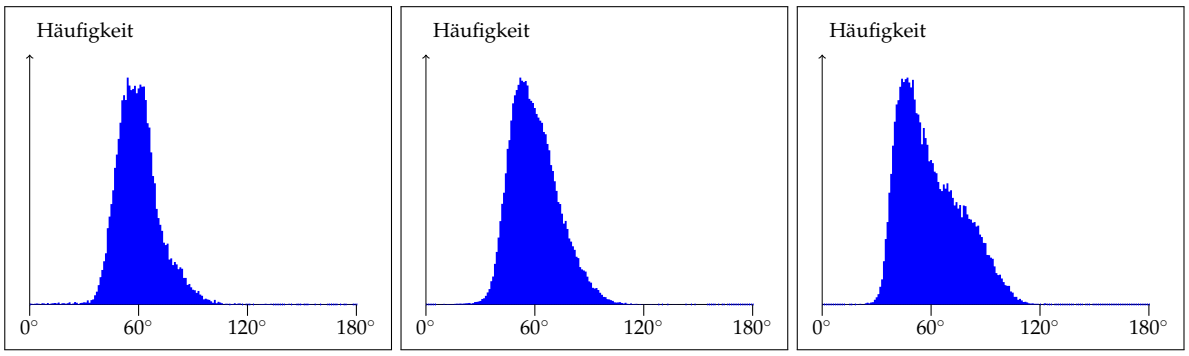
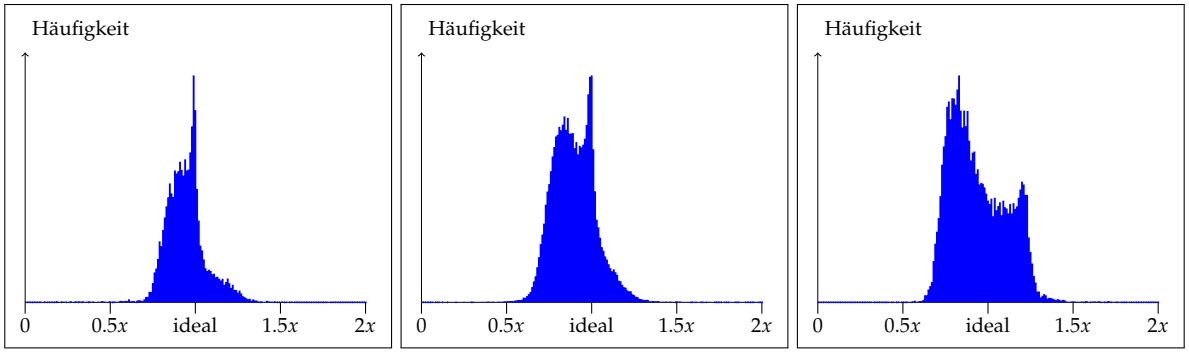


Abbildung 5.1: Berechnete Beispielmeshes. (a) ein künstlicher Datensatz einer Kugel. (b) und (c) sind Ausschnitte aus dem Originaldatensatz mit unterschiedlicher Größe und Position. (d), (e) und (f) sind die zugehörigen markierten Ausschnitte aus (a), (b) und (c).



(a) **(b)** **(c)**

Abbildung 5.2: Histogramme der Winkel innerhalb der Dreiecke zu den Beispielen aus Abbildung 5.1. Ein Winkel von 60° entspricht dem idealen angestrebten Winkel.



(a) **(b)** **(c)**

Abbildung 5.3: Histogramme der Kantenlänge zu den Beispielen aus Abbildung 5.1. Eine Kantenlänge von 1.0 entspricht der ideal angestrebten Länge.

nung eines generischen Datensatzes, einer Kugel, sowie zwei Berechnungen mit verschiedenen Positionen und Größen aus dem Volumen des Datensatzes vom ESRF verwendet. Die Ergebnisse dieser Berechnungen sind in Abbildung 5.1 mit den zugehörigen Histogrammvolumen der Winkel (Abbildung 5.2) und der Kantenlängen der Dreiecke (Abbildung 5.3) dargestellt.
Die angestrebte perfekte hexagonale Anordnung der Dreiecke aus Kapitel 2.2.2 ist in allen Beispielen sehr gut zu erkennen. Durch die Beschaffenheit der Oberfläche ist es jedoch nicht möglich diese Verteilung für alle Dreiecke zu erreichen. In Abbildung 5.1 (d) sind auf der linken Seite auf halber Höhe Vertices zu erkennen die nicht mit der optimalen Anzahl von sechs weiteren Vertices verbunden sind. Im zugehörigen kompletten Mesh (a) sind große Flächen mit annähernd idealer Vernetzung zu erkennen. An den Übergangsstellen zwischen solchen idealen Flächen kann es dabei zu geringen Abweichungen der optimalen Form kommen. Diese Abweichungen bewegen sich jedoch in einem akzeptablen Bereich. In Beispiel (b) und (c) mit den zugehörigen Ausschnitten (e) und (f) wird die hervorragende

Verteilung der Vertices auch auf komplexeren Oberflächen demonstriert.
Die Histogramme in Abbildung 5.2 belegen die Qualität der Dreiecke hinsichtlich eines angestrebten Winkels von 60°. Der überwiegende Teil der Dreiecke besitzt Winkel in einem Bereich mit weniger als 30° Abweichung.
In Abbildung 5.3 wird die Verteilung der Kantenlängen der Dreiecke zu den Beispielen aus Abbildung 5.1 präsentiert. Mit einer maximalen Abweichung von $< 30\%$ des Großteils der Kantenlängen ist dieses Kriterium des Anwenders ebenfalls erfüllt. Sehr gut zu erkennen ist das quasi Fehlen von Kanten mit einer Länge kleiner als die Hälfte der idealen Länge oder mehr als das Anderthalbfache der idealen Länge.
Die Dreiecksnetze aus mehrzelligen Berechnungen sind mit den Netzen aus einer einzelligen Berechnung nahezu identisch. Bei beiden Vorgängen liegt die Binstruktur zugrunde weshalb es keine grundlegenden Unterschiede oder erkennbare Übergänge der Netze gibt. Lediglich durch die Reihenfolge der Threads beim ausführen und den Zufallszahlen bei der Dichtekontrolle der Partikel könnten Unterschiede auftreten. Diese sind aber hinsichtlich der Qualität der Ergebnisse obsolet.

5.2 Ressourcenverbrauch

Bei der Effizienz des Verfahrens ist auch der Ressourcenverbrauch ein entscheidender Faktor. Dieser gliedert sich in den verwendeten maximalen Speicher sowie in die zur Berechnung benötigte Zeit. Zunächst wird das Verfahren mit der in Kapitel 3 verwendeten Erweiterung für große Datensätze hinsichtlich des Speicherbedarfs bewertet.

5.2.1 Speicher

Durch die verwendete Binning-Struktur zur Beschleunigung der Berechnung (siehe Kapitel 2.3) werden je Bin innerhalb der Berechnungszelle folgende Daten benötigt:

- Anzahl der Partikel in diesem Bin (je Bin $4Byte$)
- Position des ersten Partikels dieses Bins im Speicher (je Bin $4Byte$)

Dies ergibt 8 Byte je Bin. Durch eine Berechnung der Anzahl an Partikeln je Bin aus den Startpositionen der Partikel der Bins könnte dieser Faktor noch halbiert werden. Für jede Berechnungszelle werden weiterhin diese Daten benötigt:

- Positionen der Partikel (je Partikel $12Byte$)
- Lambdawerte der Partikel (je Partikel $4Byte$)
- Flag zum merken ob ein Partikel gelöscht werden soll (je Partikel $1Byte$)

Dies ergibt je $17Byte$ pro Partikel.
Für die Projektion der Partikel auf die Oberfläche und die Berechnung des Cosinusfaktores (siehe Kapitel 2.2.2 und 2.2.3) wird ein Distanzfeld sowie ein Normalenvolumen benötigt welches sich aus

- Richtung und Entfernung des Voxels zur Oberfläche (je Voxel $16Byte$)
- Normale des Voxels (je Voxel $12Byte$)
- Rohdaten des Ausgangsvolumens (je Voxel $1Byte$)

zusammensetzt. In der Summe sind dies $29Byte$ je Voxel des Ausgangsvolumens.

Berechnung

Für einen Faktor von 1.0 der idealen Größe der Dreiecke werden je zwei Voxel für die Ghostlayer benötigt. Die Anzahl der Bins ist hierbei mit der Anzahl an Voxeln identisch, da die Bin-Größe der Größe der Voxel der Ausgangsdaten entspricht.
Für eine Zellgröße von 256 Voxeln werden also $(256+4)*(256+4)*(256+4) = 17.576.000$ Bins und ebenso viele Voxel des Distanzfeldes benötigt. Dies ergibt eine Gesamtmenge von $17.576.000 * (8Byte + 29Byte) \approx 650MByte$ der notwendigen Metadaten einer einzigen Berechnungszelle. Mit etwa 150.000 Partikeln je Berechnungszelle werden zusätzliche $150.000 * 17Byte \approx 2,5MByte$ für die Partikel benötigt. Diese Größe kann während der Berechnung variieren, der Einfluss auf den gesamten Speicherbedarf ist aber verschwindend gering.
Eine deutlich größere Zellgröße ist mit heutigen Desktop-Grafikkarten nicht möglich, da dies sonst den verfügbaren Speicher überschreiten würde. Um bei einem großen zu berechnenden Volumen mit mehreren maximal großen Zellen auch den Arbeitsspeicher zu entlasten werden die Metadaten bei jedem Berechnungsschritt der Zelle erneut berechnet. Außer den Partikeldaten bleiben dabei lediglich die Rohdaten des Ausgangsvolumens mit je $1Byte$ pro Voxel im Speicher erhalten. Dies erspart das erneute, verhältnismäßig langsame Laden der Daten aus dem Hauptspeicher beim neuberechnen der Metadaten.

Beispiel

Für ein zu berechnendes Volumen von 2048^3 ergibt sich daraus eine Speicherauslastung von etwa $512 * (260 * 260 * 260) * 1Byte + 512 * 2.550.000Byte = 10.304.512.000Byte \approx 10,3GB$ permanente Daten im Arbeitsspeicher. Mit einer modernen CPU mit 8 Kernen sowie zwei CUDA-fähigen Grafikkarten ergeben sich 10 aktive Zellen mit zusammen $10 * 650.312.000Byte = 6.503.120.000Byte \approx 6,5GB$ Arbeitsdaten. Weiterhin muss für das Betriebssystem und andere Dienste ein gewisser Speicheranteil eingeplant werden, weshalb sich der benötigte Arbeitsspeicher des Systems auf etwa $20GB$ beläuft.
Das erneute berechnen der Metadaten ist bei großen Ausschnitten deshalb notwendig. Dies führt allerdings zu einem erhöhten Zeitbedarf welcher im Verhältnis zur benötigten Gesamtzeit jedoch marginal ist.

Zell-größe	Anzahl Zellen	parallel ausführbar	idealSize Faktor	Einsatz CPU / GPGPU	Zeit (in Sekunden)		
					Gesamt	Initialisierung	Berechnung
256	1	1	1.0	GPGPU	2082.9	23.7	2059.2
128	8	1	1.0	GPGPU	2172.3	44.9	2127.4
128	8	1	2.0	GPGPU	2504.6	44.9	2459.7
128	8	1	4.0	GPGPU	4706.4	44.9	4661.5
64	64	8	1.0	GPGPU	2571.8	86.5	2485.3
64	64	8	1.0	CPU + GPGPU	2269.2	86.5	2182.7

Tabelle 5.1: Die Parameter mit entsprechenden Zeiten der Berechnung. Die Zellgöße ist die Größe der einzelnen Berechnungszellen. Aufgeführt sind die daraus zum Endvolumen entstehende Anzahl an Zellen sowie die Anzahl an parallel ausführbaren Zellen. Der Faktor der idealen Größe der Dreiecke sowie der Einsatz von zusätzlichen CPU-threads ist für jeden Eintrag aufgeführt. Unter den Zeiten wird die Zeit zur Initialisierung der Zellen und die eigentliche Berechnungszeit sowie die Summe beider angegeben.

5.2.2 Zeit

Für die Zeitmessungen des Programmes wird ein Ausschnitt aus der Mitte des Originaldatensatzes von 256^3 verwendet. Die Berechnung durchläuft jeweils zehn Schritte des gesamten Volumens (Siehe dazu Abbildung 2.9). Es wird sowohl die Größe der Berechnungszellen bei den Zeitmessungen variiert, als auch der Einsatz von zusätzlichen CPU-Threads zu einer rein GPU-basierten Berechnung. Die Zeiten für die Initialisierung und die Berechnung werden gemessen. Das bei allen Varianten gleiche Vernetzen der Partikel des externen Programms sowie kleinere analoge Nachbearbeitungsschritte werden nicht in die Zeitmessungen mit aufgenommen. Für eine ausgewählte Konfiguration wird ebenso die anzustrebende ideale Größe variiert.

Testsystem

Das für die Zeitmessung verwendete Testsystem ist

- CPU - AMD Athlon 64 *X*2 Dual Core Prozessor 4000+
- RAM - 8*GB DDR*2 Speicher
- GPGPU - NVIDIA GeForce GTX 650 Ti mit 2*GB GDDR*5 Speicher
- Betriebssystem - Windows 7 64*bit*

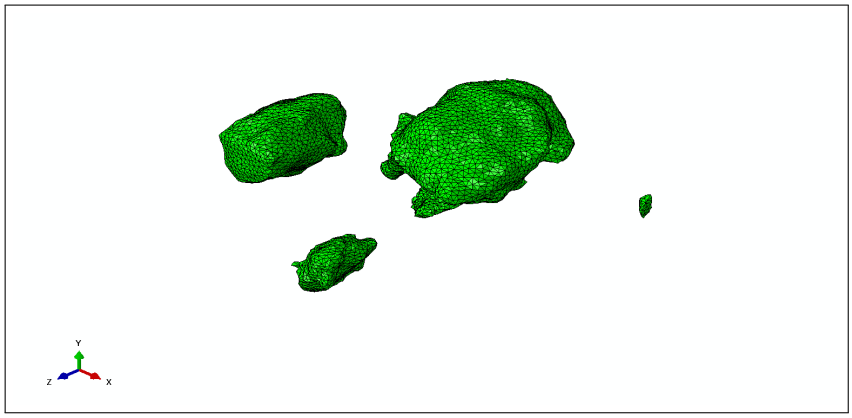
Auswertung

Die Zeiten der Berechnung sind in Tabelle 5.1 aufgeführt. Je größer die Berechnungszellen, desto geringer ist die Berechnungszeit. Bei größeren Zellen ist der zusätzliche Aufwand zum Austausch der Daten sowie die Berechnung der Metadaten geringer. Ebenso wird die Auslastung der Grafikkarte mit größeren Daten erhöht. Dabei wird das Verhältnis der Zeit

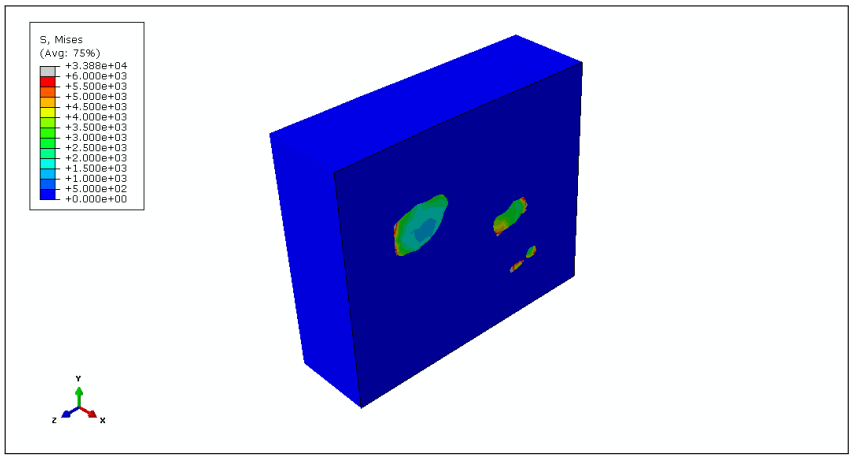
zum Austausch zwischen Host (CPU) und Device (GPGPU) und der eigentlichen Berechnung mit größeren Zellen verringert.
Eine deutliche Beschleunigung durch zusätzliche CPU-Threads ist in den Zeiten zu erkennen. Das Testsystem ist nicht ausgewogen hinsichtlich der Leistung von CPU und GPGPU. Dennoch ist die Beschleunigung um etwa 12% erkennbar. Mit einer modernen CPU mit 8 oder mehr Rechenkernen könnte diese Beschleunigung deutlich erhöht werden. Um das Potential der Parallelisierung ausschöpfen zu können sollten mindestens 64 parallel ausführbare Berechnungszellen zur Verfügung stehen, sowie eine Zellgröße von 128^3 oder 256^3 verwendet werden. Die Zellgröße sollte nicht zu klein gewählt werden da sich sonst der zusätzliche Aufwand bei Berechnung und Speicherbedarf deutlich erhöhen.
Bei einer größeren idealen Kantenlänge der Dreiecke ist ein deutlicher Anstieg des Berechnungsaufwandes zu erkennen. Dies folgt aus der größeren Bin-Größe und dem größeren Einflussbereich bei der Energieberechnung der Partikel. Durch die gleichbleibend dichte Streuung der Partikel unabhängig der idealen Kantenlänge zu Beginn der Berechnung durch den Marching Cubes Algorithmus ist die Anzahl an wechselwirkenden Partikeln deutlich erhöht. In den ersten Durchläufen bei der Berechnung ist der Aufwand deutlich erhöht, dies kann auch bei geringerem Aufwand mit weniger Gesamtpartikel in den späteren Durchläufen nicht kompensiert werden.

5.3 Finite-Elemente Analyse

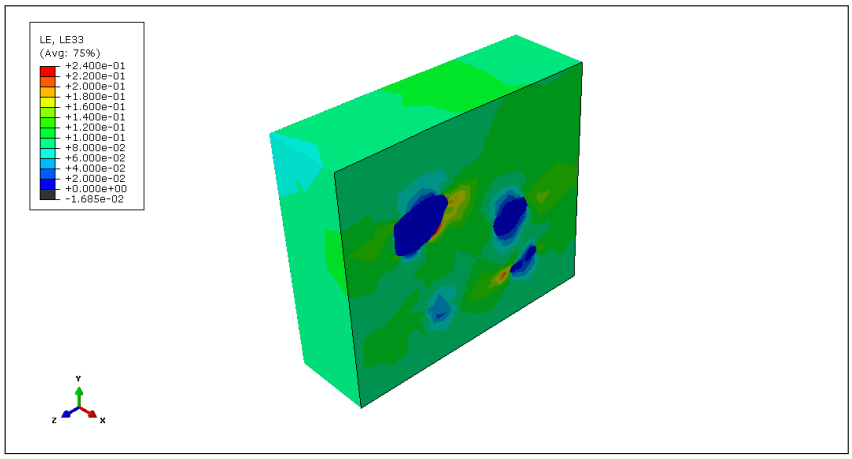
Das Dreiecksnetz des Beispieles von Abbildung 5.1(c) wurde von Dr.-Ing. Ulrich Weber für eine Analyse mit finiten Elementen (FEA) verwendet. Eine Visualisierung der durchgeführten FE-Berechnung ist in Abbildung 5.4 dargestellt. Die Auswertung der Berechnungen und die Aussagen über die Ergebnisse im Folgenden wurden von Dr.-Ing. Ulrich Weber erstellt.
Bei diesem Werkstoff handelt es sich um einen partikelverstärken Verbundwerkstoff. Die Partikel bestehen aus Al_2O_3, das die Partikel umgebende Volumen besteht aus der Aluminium-Legierung $AA6061$. Der Volumenausschnitt mit den abgebildeten Partikeln wurde in ein repräsentatives Volumen für die Berechnung eingebettet.
Dieses Würfel-Volumen wurde in Z-Richtung um 10% verlängert. Das Ergebnis dieser Berechnung mit den Von-Mises-Spannungen ist in Abbildung 5.4(b) zu sehen. Man erkennt, dass die festen Keramikpartikel die Spannungen des Verbundwerkstoffs aufnehmen.
In Abbildung 5.4(c) ist eine Drehung des Volumens dargestellt. Wegen der Verformungsfähigkeit der Aluminium-Matrix gegenüber der Keramikpartikel werden die Drehungen von der Aluminium-Matrix ausgenommen. Dabei entstehen die Verformungslokalitäten in der Umgebung der Keramikpartikel.



(a)



(b)



(c)

Abbildung 5.4: Visualisierung einer FE-Berechnung. Diese Bilder wurden von Herrn Dr.-Ing. Ulrich Weber erstellt. In (a) ist das verwendete Modell zu sehen. In (b) ist in einem Schnitt durch das Volumen die Von-Mises-Spannung bei 10% globaler Dehnung in Z-Richtung dargestellt. In (c) sind die Spannungen bei einer globalen Drehung von 10% in Z-Richtung abgebildet.

6 Zusammenfassung und Ausblick

Im Folgenden wird die Arbeit und das Erreichte kurz zusammengefasst. Anschließend werden Anknüpfpunkte für zukünftige Weiterentwicklungsmöglichkeiten vorgestellt.

6.1 Zusammenfassung

Ausgehend der Arbeit von [MGW05] wurde ein Partikelsystem implementiert welches auf einer Isofläche in einem Volumen Partikel gleichmäßig verteilt. Diese Partikel werden über ein externes Programm [DGG+] vernetzt. Bei einer optimalen Verteilung der Partikel ergibt sich ein hexagonales Dreiecksnetz (Kapitel 2.4).
Mit einer Beschleunigungstrategie, dem Binning (siehe Kapitel 2.3), werden die benötigten Zeiten der Berechnungen des Partikelsystems durch räumliche Einteilung der Partikel verringert. Da die durch Binning verringerte Berechnungszeit für große Volumen immer noch zu hoch ist, wurde das Verfahren auf eine hoch parallele Ausführung auf GPGPUs ausgelegt (siehe Kapitel 2.4).
Die dabei erzielte Geschwindigkeit ist für ein großes Volumen akzeptabel, jedoch durch den Speicherbedarf (siehe Kapitel 5.2.1) nicht ausführbar. Zur Lösung dieses Problems wurde eine Aufteilung des Gesamtvolumens in einzelne Berechnungszellen durchgeführt (siehe Kapitel 3). Der Austausch der Berechnungsdaten an den Grenzschichten der Zellen sowie die Ausführung nach gewissen Beschränkungen wurden präsentiert.
Eine weitere Parallelisierungsebene bei der Ausführung der Berechnungsschritte der einzelnen Berechnungszellen wurde eingeführt (Kapitel 3.3) und bewertet (Kapitel 5.2.2). Die Qualität des Meshes wurde mit den Anforderungen des Anwenders verglichen und bewertet (Kapitel 5.1).
Zum Verständnis wofür das berechnete Mesh eingesetzt wird, wurde in Kapitel 5.3 eine von Herrn Dr.-Ing. Ulrich Weber erstellte Simulation mit Patran [MSC] anhand eines berechneten Meshes vorgestellt. Die angeforderte Qualität des Dreiecksnetzes wurde erreicht. Eine angemessene Berechnungsdauer konnte erzielt werden. Die Anforderungen an den verfügbaren Arbeitsspeicher befinden sich in einem akzeptablen Bereich.

6.2 Ausblick

Während der Bearbeitung des Themas dieser Diplomarbeit haben sich weitere Felder eröffnet, welche nicht im Rahmen dieser Arbeit behandelt werden konnten.

Vernetzung

Für die Vernetzung der Partikel wird die Software TightCocone [DGG+] verwendet. Dieses externe Programm vernetzt die Partikel nur aufgrund deren Positionen innerhalb des Volumens. Dies kann unter Umständen zu einer inkorrekten Vernetzung führen. Hierbei könnte ein eigener Algorithmus zum vernetzen der Partikel entwickelt werden, welcher anhand der vorhandenen zusätzlichen Metadaten (Normalen- und Distanzvolumen) ein besseres Ergebnis erzielen kann.

Voxelisierung

Für die finite-Elemente Analysen (FEA) aus Kapitel 5.3 wird ein Volumenmesh aus Thetraedern benötigt. Dies kann von Patran [MSC], der verwendeten Software für die FEA, direkt erstellt werden. Hier kann mit einem eigenen internen Volumen-Meshingalgorithmus das Volumen bereits vernetzt werden. Alternativ kann eine Schnittstelle zur Software TetGen [tet] eingerichtet werden, damit dem Anwender direkt ein Volumenmesh zur Verfügung gestellt werden kann.

Transferfunktion

Zur besseren Abgrenzung der Isofläche in nicht eindeutig entscheidbaren Bereichen könnte eine Transferfunktion implementiert werden. Dabei könnte in weiteren Schritten zwischen mehreren Materialien in einem Volumen unterschieden werden.

GPGPU-Cluster

Für eine schnellere Berechnung des Gesamtvolumens kann die Berechnung der einzelnen Zellen auf einem GPGPU-Cluster stattfinden. Die entwickelte Struktur der Berechnungszellen ist dafür bereits ausgelegt. Dabei könnten Berechnungszellen einzelnen Knoten zugeordnet werden und über Message Passing Interfaces (MPI) oder Infiniband der Austausch der Ghostlayer stattfinden. Dies könnte die Berechnung weiter beschleunigen.

Literaturverzeichnis

[ant] A light and intuitive graphical user interface for graphic applications based on OpenGL. URL http://anttweakbar.sourceforge.net/. AntTweakBar website http://anttweakbar.sourceforge.net/. (Zitiert auf Seite 41)

[Cen] Center for Integrative Biomedical Computing (CIBC) University of Utah. SCIRun. URL http://www.sci.utah.edu/cibc-software/scirun.html. (Zitiert auf Seite 22)

[CMS97] P. Cignoni, C. Montani, R. Scopigno. A Comparison of Mesh Simplification Algorithms. *Computers & Graphics*, 22:37–54, 1997. (Zitiert auf Seite 16)

[DG03] T. K. Dey, S. Goswami. Tight cocone: a water-tight surface reconstructor. In *Proceedings of the eighth ACM symposium on Solid modeling and applications*, SM '03, S. 127–134. ACM, New York, NY, USA, 2003. doi:10.1145/781606.781627. URL http://doi.acm.org/10.1145/781606.781627. (Zitiert auf Seite 17)

[DGG+] T. K. Dey, J. Giesen, S. Goswami, J. Hudson, W. Zhao. tightCocone Software. URL http://www.cse.ohio-state.edu/~tamaldey/cocone.html. (Zitiert auf den Seiten 17, 53 und 54)

[FL08] K. R. B. Fagerjord, T. V. Lochehina. GPGPU: Fast and easy Distance Field computation on GPU, 2008. URL http://difi.freya.no/. (Zitiert auf den Seiten 7, 18 und 19)

[Hec97] P. Heckbert. Fast surface particle repulsion. In *SIGGRAPH*, Band 97, S. 95–114. Citeseer, 1997. (Zitiert auf Seite 26)

[Jmt] Jmtrivial at Wikipedia.org. Marching Cubes cases. URL http://commons.wikimedia.org/wiki/File:MarchingCubes.svg. (Zitiert auf den Seiten 7 und 15)

[KCH12] M. Kim, G. Chen, C. Hansen. Dynamic particle system for mesh extraction on the GPU. In *Proceedings of the 5th Annual Workshop on General Purpose Processing with Graphics Processing Units*, GPGPU-5, S. 38–46. ACM, New York, NY, USA, 2012. doi:10.1145/2159430.2159435. URL http://doi.acm.org/10.1145/2159430.2159435. (Zitiert auf den Seiten 8, 15, 17, 22, 23, 25, 26, 28, 30, 31 und 32)

[LC87] W. E. Lorensen, H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, 1987. doi:10.1145/37402.37422. URL http://doi.acm.org/10.1145/37402.37422. (Zitiert auf den Seiten 16 und 42)

[LHMG02] U. Labsik, K. Hormann, M. Meister, G. Greiner. Hierarchical Iso-Surface Extraction. *Journal of Computing and Information Science in Engineering*, 2(4):323–329, 2002. Selected Papers of SMA 2002. (Zitiert auf Seite 16)

[Meg] MegaMol Software. URL http://www.visus.uni-stuttgart.de/megamol. MegaMol project website http://www.visus.uni-stuttgart.de/megamol/. (Zitiert auf Seite 41)

[MGW05] M. D. Meyer, P. Georgel, R. T. Whitaker. Robust Particle Systems for Curvature Dependent Sampling of Implicit Surfaces. In *Proceedings of the International Conference on Shape Modeling and Applications 2005*, SMI '05, S. 124–133. IEEE Computer Society, Washington, DC, USA, 2005. doi:10.1109/SMI.2005.41. URL http://dx.doi.org/10.1109/SMI.2005.41. (Zitiert auf den Seiten 15, 19, 23 und 53)

[MSC] MSC Software. URL http://www.mscsoftware.com/product/patran. Patran software website http://www.mscsoftware.com/product/patran. (Zitiert auf den Seiten 43, 53 und 54)

[NVIa] NVIDIA Corporation. Cuda C programming guide v5.0. URL http://docs.nvidia.com/cuda/cuda-c-programming-guide/. (Zitiert auf den Seiten 8 und 30)

[NVIb] NVIDIA Corporation. NVIDIA Fermi architecture whitepaper. URL http://www.nvidia.com/content/PDF/fermi_white_papers/NVIDIA_Fermi_Compute_Architecture_Whitepaper.pdf. (Zitiert auf den Seiten 8 und 29)

[tet] A Quality Tetrahedral Mesh Generator and a 3D Delaunay Triangulator. URL http://wias-berlin.de/software/tetgen/. TetGen website http://wias-berlin.de/software/tetgen/. (Zitiert auf Seite 54)

[Uni] University of Münster and Linköping Univeristy. Open source volume rendering engine. URL http://www.voreen.org/. (Zitiert auf den Seiten 7 und 13)

[WH05] A. P. Witkin, P. S. Heckbert. Using particles to sample and control implicit surfaces. In *ACM SIGGRAPH 2005 Courses*, SIGGRAPH '05. ACM, New York, NY, USA, 2005. doi:10.1145/1198555.1198656. URL http://doi.acm.org/10.1145/1198555.1198656. (Zitiert auf Seite 26)

[WSBD00] Z. J. Wood, P. Schröder, D. Breen, M. Desbrun. Semi-regular mesh extraction from volumes. In *Proceedings of the conference on Visualization '00*, VIS '00, S. 275–282. IEEE Computer Society Press, Los Alamitos, CA, USA, 2000. URL http://dl.acm.org/citation.cfm?id=375213.375254. (Zitiert auf den Seiten 7 und 17)

Alle URLs wurden zuletzt am 29. 07. 2013 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift