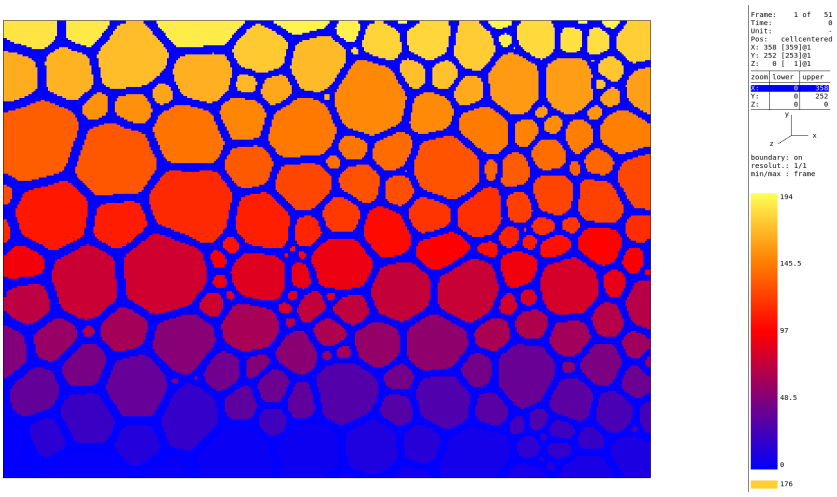
Bilddigitalisierung und Simulation von Schäumen

Nhat Phi Ho und Fynn Wagner

September 29, 2022

Abschlussdokumentation der Kooperationsphase 2021/22

Durchgeführt am Institut für Angewandte Materialien
Betreut von Jana Holland-Cunz und Matthieu Laqua



Inhaltsverzeichnis

Abstract

Metal foams are spongy and solid structures, where gas-filled pores are surrounded by either solid or liquid metal. They offer an alternative to common metal usage, due to their heat and energy absorption properties and because they are less heavy regarding their reduced density.
At the *Institute for Applied Materials - Microstructure Modelling and Simulation (IAM-MMS)* at the Karlsruhe Institue of Technologie, scientists work with experimental data of liquid foams. By application of a software called PACE3D[8] it is possible to create simulation runs, which behave similar to real foams with the aim to predict physical properties. The subject of this project is to enable an even closer connection betweeb experimental and simulated data, by developing a program which can be used to convert image files taken during experiments into special files usable by PACE3D. The goal is to successfully convert an image of a foam structure during an early stage of the experiment and create a simulation run, which mimicks the real temporal evolution of the foam.
It was possible to develop a computer program which can convert the pictures into the desired file format. In addition, it was possible to create simulation runs with the converted pictures as basis, and attempts to calibrate the simulation to fit the experiment have been made.

1 Einleitung

Unter Metallschäumen versteht man komplexe poröse Strukturen, in denen Gas durch feste Metallwände, oder flüssige Metall-Lamellen getrennt und eingeschlossen wird (nach [1]). Metallschäume bieten als Konstruktionswerkstoffe eine Alternative zu herkömmlichem Metall, besonders im Maschinenbau. Das liegt an ihrer geringen Dichte und entsprechend geringerem Gewicht. Auch haben sie eine sehr hohe Wärmeleitfähigkeit (nach [9]) und eine gute Absorptionsfähigkeit um Energie in Form von Schwingungen, Stoß oder Schall aufzunehmen (nach [2]). Metallschäume sind aber noch nicht ausreichend genug erforscht worden, um großflächig in der Industrie Verwendung zu finden.
Flüssigkeitstemplatierung beschreibt ein neues Verfahren, in dem Tropfen und/ oder Blasen sich in einer flüssigen Emulsion oder einem Schaum anordnen, bevor diese verfestigt werden. Das Verfahren wird angewendet, um Vorlagen zur Synthese von festen Schäumen zu erstellen.[10] Flüssige Schäume sind metastabil, das bedeutet, sie zerfallen mit der Zeit. Um die Entwicklung und auch den Zerfall der flüssigen Schäume vorherzusagen und so gezielt Schäume mit bestimmten Eigenschaften herzustellen, können Simulationen hilfreich sein. Sie dienen als Ersatz, aber auch als Begleitung von teuren, gefährlichen und zeitintensiven Experimenten, um Eigenschaften und Strukturen von Schäumen zu vorherzusagen.
Am Institut für Angewandte Materialien am KIT werden Daten aus Experimenten mit wässrigen Polymer-Tensid-Schäumen verarbeitet. Dabei handelt es sich um Bilderreihen von Momentaufnahmen eines zeitlichen Verlaufs, in dem der Schaum sich unter bestimmten Bedingungen ausbildet.
Dank des am Institut entwickelten Softwarepakets PACE3D[8] ist es möglich, den zeitlichen Verlauf der Entstehung von Schäumen in Simulationen abzubilden. Die Simulation mit PACE3D basiert auf der Phasenfeldmethode. Als Phasen gelten Bereiche mit gleichem chemischen Aufbau und gleichen physikalischen Eigenschaften wie z.B. Dichte (nach [7]). Bei flüssigen Schäumen bildet jede einzelne Blase und auch die Flüssigkeit, welche die Blasen trennt, jeweils eine Phase. Die Phasenfeldmethode ist ein mathematisches Verfahren mit der physikalische Eigenschaften durch Parameter ersetzt und Übergänge zwischen Phasen und ihre morphologische Entwicklung mittels mathematischer Gleichungen modelliert werden. PACE3D wird daher auch als Phasenfeldlöser bezeichnet (nach [8]).
Um Simulationsvorgänge und experimentelle Vorgänge zu verbinden, wurde in diesem Projekt ein Programm entwickelt, welches experimentelle Daten, wie Bilder, in Skalarfeld-Dateien im Format des Phasenfeldlösers umwandelt. Ziel ist es die Simulation durch Variieren der Parameter so gut wie möglich an die Experimentaldaten anzugleichen. Die Bilddigitalisierung ist als Zwischenschritt nötig, damit das vorliegende Bildmaterial in einem frühen Stadium des Experiments umgewandelt werden kann. Anschließend ist es möglich mittels PACE3D eine Simulation zu erstellen, welche die Morphologie des Schaums im zeitlichen Verlauf vorhersagt.

2 Material und Methoden

Zur Simulation von Polymer-Tensid-Schäumen ist es nötig, ein gegebenes Bild-digitalisierungsprogram um weitere Funktionen zu erweitern. Programmiert wurde in der Entwicklungsumgebung CLion in der Sprache C. Der Code, der in dieser Dokumentation zu finden ist, wurde im Institut für Zwecke wie Leserlichkeit nachträglich angepasst. Der wesentliche Teil des Programms, wie der Algorithmus, wurde aber in der Arbeit erstellt.

2.1 Beschreibung des Programms

Das Programm soll Dateien im PNG-Bildformat in ein Format umwandeln, welches für Simulationen weiterverwendet werden kann. Zur Erstellung des Programms wurde ein bereits bestehendes Programm als Basis genutzt, welches PNG-Bilder in dieses Format umwandeln kann. Es werden dabei die jeweiligen Farben der PNG-Dateien den entsprechenden Phasen zugeordnet. In Abb. 1 ist ein Beispiel des Bildmaterials zu sehen. Alle Blasen würden hier, da sie ihre Farbe teilen, der gleichen Phase angehören.

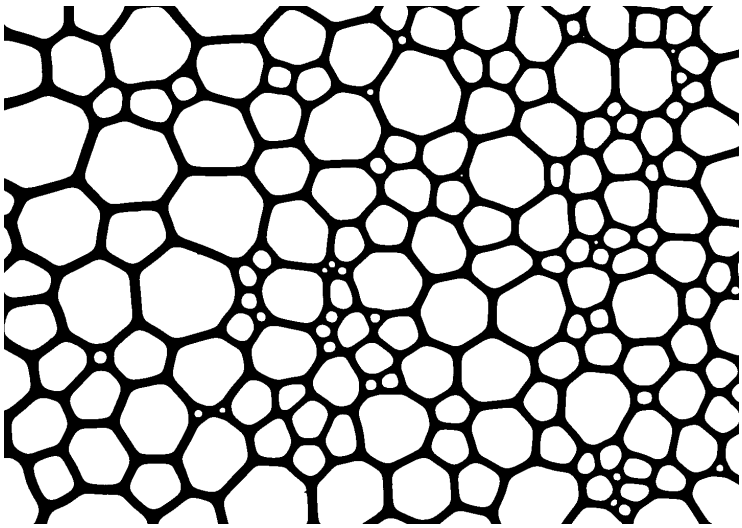


Abb. 1: Vorliegendes Bildmaterial

Eigentlich soll aber jede Blase eine eigene Phase darstellen. Das bereits bestehende Programm wurde erweitert, so dass es zusammenhängende Strukturen erkennen und als einzelne Phasen zuordnen kann.

2.1.1 Dateiformate

PNG ist ein Rastergrafikformat, wo jedem Pixel eine RGB-Farbe zugeordnet wird. Die Größe der Dateien hält sich durch verlustfreie Kompressionsverfahren in Maßen. RGB-Farben setzen sich zusammen aus Intensitätswerte der Farben Rot, Grün und Blau, welche zusammen in einer additiven Farbmischung die Intensität der einzelnen Farbkomponenten angeben. [3]

Phasen werden im Rahmen von PACE3D in einem **Skalarfeld** gespeichert. In einem Skalarfeld wird jedem Punkt im Raum (an einer bestimmten x, y, und z Koordinate in einem orthogonalen Koordinatensystem) eine reelle Zahl zugeordnet. Diese Zahl nennt sich Skalar. [4] Skalarfelder im Phasen-Dateiformat lassen sich mit dem Programm XSimViewer graphisch betrachten, wie man in Abb. 2 sehen kann, wo eine Blase eines Schaumes dargestellt wird. Koordinaten, die zu der Phase gehören besitzen den Skalarwert 1 und werden graphisch in gelb dargestellt. Andere Koordinaten die nicht zu der Phase gehören besitzen den Skalarwert 0.

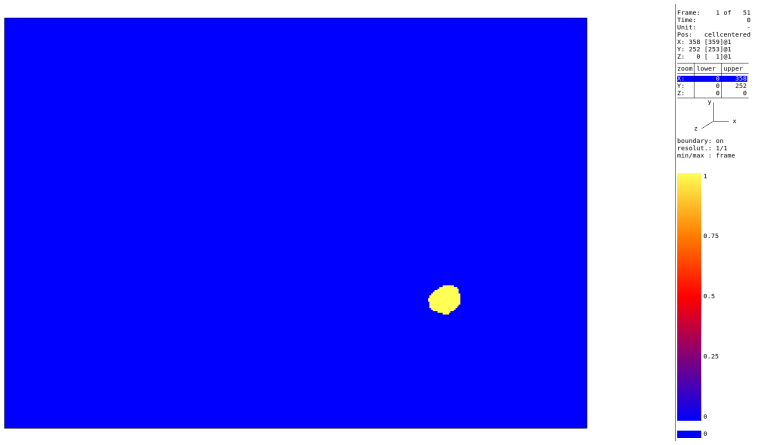


Abb. 2: Graphische Darstellung eines Skalarfelds in XSimViewer

Um einen kompletten Schaum zu bilden, werden Skalarfelder im Phasen-Dateiformat in einer separaten Datei in einem Index gelistet. Eine Zusammenstellung aller Phasen innerhalb dieses Index ist in Abb. 3 zu sehen. Jede Phase erhält hier ihren eigenen Skalarwert, und ist graphisch eingefärbt entsprechend der Legende auf der rechten Seite der Abbildung. Die Flüssigkeit ist die erste Phase 0, und die Blasen sind bis zu der gesamten Anzahl der Blasen durchnummeriert. Alle Skalare haben somit einen Wert von 0 oder höher. Dieses zusammengestellte Skalarfeld ist die letzendliche Ausgabe des Programms und wird für Simulationen weiterverwendet.

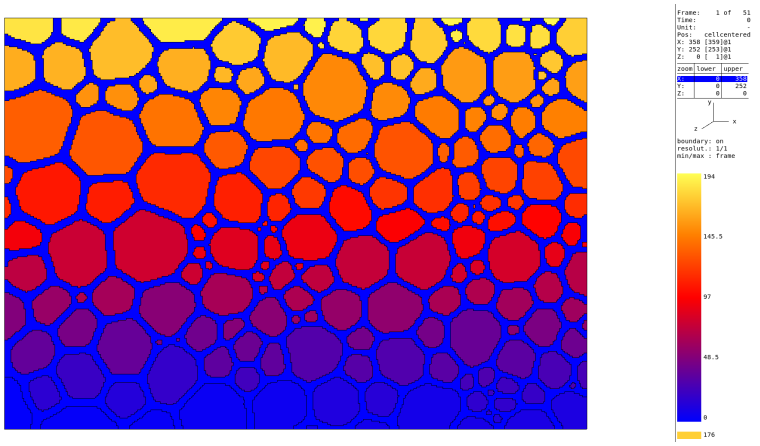


Abb. 3: Zusammenstellung des Phasen-Index

2.1.2 Erweiterung des bereitgestellten Programms

Wie vorher erläutert wurde, erstellt das bereitgestellte Programm ein Skalarfeld, welches Farbcodierungen durch entsprechenden Skalarwerte ersetzt. Die Breite und Höhe der PNG-Datei, oder die erste Datei bei mehreren PNG-Dateien, legen dabei die X und Y Dimensionen des Skalarfeldes fest. Die Z Dimension wird durch die Anzahl der PNG-Dateien bestimmt. Jedem Pixel in einer PNG-Datei entspricht also einem diskreten Raumbereich (Voxel) der in den Bildern abgebildeten Realität. Damit das Programm verschiedene Blasen unterscheidet, muss es zusammenhängende Segmente erkennen können. Für zwei Voxel, die zu dem gleichen Segment gehören, gelten folgende Bedingungen:

1. Die zwei Voxel in den müssen die gleiche Farbe haben
2. Sie müssen Nachbarn sein

2.1.3 Nachbarnprinzip

Als Nachbarn gelten alle Voxel, die direkt an eine Fläche des zu untersuchenden Voxels angrenzen. In Abb. 4 sieht man die als Nachbarn definierten Voxel eines zentralen Voxels (blau).
Dabei sind Voxel in diagonaler Richtung, welche nur an der Kante eines Voxel angrenzen, ausgeschlossen. Die sechs Nachbarn eines Voxels kann man also so definieren, dass sie sich in der X-, Y- oder Z-Richtung von dem untersuchten Voxel nur um eins unterscheiden.

2.1.4 Rekursive Schleife

Die Erkennung zusammenhängender Segmente wurde zuerst mit einem rekursiven Ansatz gelöst. Eine Rekursion in der Programmierung beschreibt eine Funktion, die sich selbst aufruft. In Abb. 5 wird das veranschaulicht: Der blaue Pfeil steht für den rekursiven, während der gelbe einen normalen Programm-Aufruf darstellt.

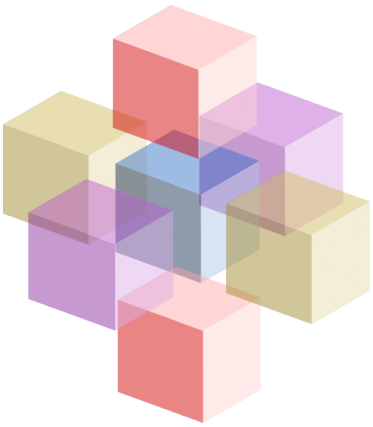


Abb. 4: Voxel (Blau) umgeben von Nachbarn

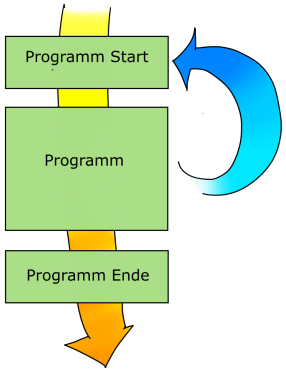


Abb. 5: Veranschaulichung eines rekursiven Aufrufs

Dabei kann von dem nach Farben kodierten Skalarfeld Gebrauch gemacht werden, um Farben zu vergleichen. Die zusammenhängenden Segmente werden in ein neues Skalarfeld eingetragen. Das ist veranschaulicht in Abb. 6. Im Folgenden wird das nach Farben kodierte Skalarfeld als Skalarfeld A bezeichnet und das nach Segmenten kodierte Skalarfeld als Skalarfeld B bezeichnet.
Die Funktion wird mit dem Start-Voxel als Parameter aufgerufen, und prüft mithilfe von Skalarfeld A für jeden Nachbar des übergebenen Voxels, ob dieser die gleiche Farbe wie der übergebene Voxel hat. Wenn ja, erhält der entsprechende Nachbar im Skalarfeld B den gleichen Skalar wie der übergebene Voxel, und die Funktion wird rekursiv aufgerufen mit dem Nachbar-Voxel als Parameter. Der rekursive Aufruf findet solange statt, bis keine Nachbarn mit gleicher Farbe zu finden sind. Somit wurde ein zusammenhängendes Segment erfasst.
Um zu verhindern, dass die Funktion sich unendlich oft aufruft, darf der gleiche Voxel nicht zweimal überprüft werden. Aus diesem Grund wird ein Kopie des Skalarfeld A erstellt, und im Verlauf wird der Skalar für jede bereits besuchte Koordinate auf -1 gesetzt. Ein Voxel wird nur untersucht, wenn sein Äquivalent in der Kopie keinen Skalar vom Wert -1 hat.

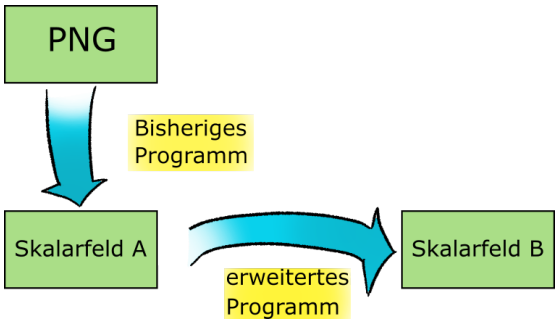


Abb. 6: Grobe Skizze von dem Verfahren

Die Verwendung einer zuvor mit -1 gefüllter Kopie des Skalarfelds A nutzt im Gegensatz zur ursprünglichen Version den Speicher effektiver.

2.1.5 Breitensuche

Ein weiteres, noch effizienteres Verfahren zur Nachbarnsuche als der rekursive Aufruf stellt die Breitensuche (BFS = engl. "*breadth-first search*") dar. Die Breitensuche ist ein Suchverfahren mit Graphen, wo für jeden Knoten zunächst alle angrenzenden Knoten untersucht werden, bevor man die Kinderknoten eines angrenzenden Knotens untersucht.[5] In Abb. 7 ist veranschaulicht, wie die Reihenfolge, in der der Algorithmus sich die Knoten eines Graphen anschaut, aussehen könnte.

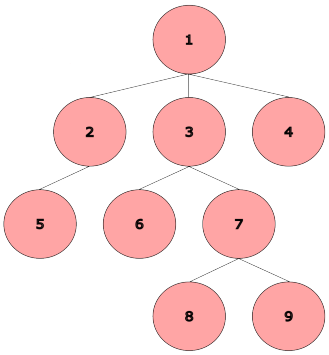


Abb. 7: Modell der Breitensuche

In einem Programm lässt sich dafür algorithmisch eine Warteschlange verwenden. Es wird immer ein Knoten in der Warteschlange gestellt und untersucht. Anschließend untersucht man die angrenzenden Knoten (engl. "*child*") und übergibt sie der Warteschlange, damit sie anschließend ebenfalls untersucht werden. Das sieht man auch an dem Pseudo Code des Algorithmus [5]. In dem Fall stoppt der Algorithmus, wenn er den gesuchten Knoten `goal_node` gefunden hat:

```
1  BFS(start_node, goal_node)
2  erzeuge eine leere Warteschlange queue
3  queue.enqueue(start_node);
4  markiere start_node als gesehen
5  while queue ist nicht leer
6      node = queue.dequeue();
7      if node == goal_node
8          return true;
9      foreach child in nachfolger(node)
10         if child ist nicht markiert als gesehen
11             queue.enqueue(child);
12             markiere child als gesehen
13 return false;
```

Für das Programm wurde der Algorithmus entsprechend angepasst. Als Knoten (node) fungiert die Struktur `PixelState` welche die Koordinaten enthält, mit der man den gewünschten Skalar in dem Feld referenzieren kann. Eine Warteschlange `bfsQueue` wird erstellt, welche den Start-Voxel `startPixel` auf-nimmt.

```
1  PixelState *startPixel_heap = (PixelState *) Malloc(sizeof(
       PixelState));
2  *startPixel_heap = startPixel;
3
4  Queue *bfsQueue = newQueue();
5  Queue_enqueue(bfsQueue, startPixel_heap);
```

Die anschließende BFS Schleife fängt mit dem `startPixel` an und untersucht dessen zugehörige Nachbarn. Solange ein Nachbar keine illegalen Koordinaten (Koordinaten außerhalb der Dimensionen des Skalarfeldes) besitzt, im Skalarfeld B, hier `result`, noch keinen Skalarwert ungleich -1 hat und auch die Farbe mit `startPixel` teilt, wird er ebenfalls der Warteschlange `bfsQueue` übergeben. Nachdem der Skalarwert des untersuchten Voxels in `result` in den Wert des Segments (`colorValueToSet`) geändert wurde, kann die Schleife den nächsten Voxel aus `bfsQueue` rausholen. Das wiederholt sich bis `bfsQueue` leer ist.

```
#define UNMARKED -1.0f

while (!Queue_isEmpty(bfsQueue)) {
    PixelState *currentPixel = Queue_dequeue(bfsQueue);

    for (int i = 0; i < 6; ++i) {
      long neighbour_x = currentPixel->x + n6_offset_list[i].x;
      long neighbour_y = currentPixel->y + n6_offset_list[i].y;
      long neighbour_z = currentPixel->z + n6_offset_list[i].z;

      if (SimGeo_isInsideDomain(original->simgeo, neighbour_x,
          neighbour_y, neighbour_z)
          && ScalarData_getValue(result, neighbour_x, neighbour_y,
          neighbour_z) == UNMARKED
          && ScalarData_getValue(original, neighbour_x,
          neighbour_y, neighbour_z) == colorValueOriginal) {

        PixelState *neighbourPixel =
              (PixelState *) Malloc(sizeof(PixelState));
        neighbourPixel->x = neighbour_x;
        neighbourPixel->y = neighbour_y;
        neighbourPixel->z = neighbour_z;
        Queue_enqueue(bfsQueue, neighbourPixel);

        ScalarData_setValue(result, neighbour_x, neighbour_y,
        neighbour_z, colorValueToSet);
        if (singlePhaseResult != NULL) {
          ScalarData_setValue(singlePhaseResult, neighbour_x,
              neighbour_y, neighbour_z, 1.0f);
        }
      }
    }
    Free(currentPixel);
  }
```

Die BFS bildet so das Kernstück des Programms, womit ein gesamtes zusammenhängendes Segment in einer Schleife in das Skalarfeld `result` geschrieben werden kann.

2.1.6 Generierung des Skalarfeldes

Zur tatsächlichen Generierung des Skalarfeldes sind mehrere Schritte nötig. Zuerst wird die PNG Datei binarisiert, so dass alle Pixel entweder schwarz oder weiß sind. Bei den Bildern aus den Experimenten sind die Blasen weiß und die Flüssigkeit schwarz, wie in Abb.8 gezeigt.
Für die Simulation ist es von Bedeutung, dass die erste Phase bzw. der Skalar 0 die Flüssigkeit ist. Darum muss dem Programm die Farbe der Flüssigkeit im Bild mitgegeben werden, so dass das Programm die Flüssigkeit als erste Phase in das neue Skalarfeld schreiben kann. Das kann durch eine Farbpalette geschehen, die als Parameter dem Programm beigegeben wird.
Bei der Farbpalette handelt es sich um eine Text-Datei, worin einer Phase einer bestimmten Farbe in der hexadezimalen Farbdefinition zugeordnet wird. Der Inhalt könnte z.B so aussehen:

```
Phasefield.N=2
Phasefield.Phases=(phase0, phase1)
phasecount=2
Colors=(0x000000, 0xFFFFFF)
```

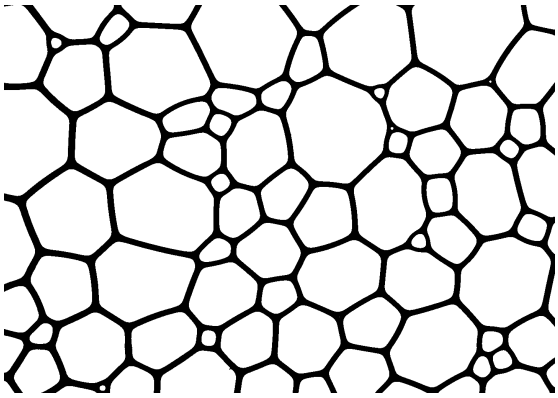


Abb. 8: Binarisiertes Bild des Experiments

Es ist möglich das binarisierte Bild mit dem bereits gegebenen Programm umzuwandeln und für das so erstellte Skalarfeld eine Farbpalette zu speichern. Diese Farbpalette muss dann nur angepasst werden, damit die Bildfarbe des binarisierten Bildes der Flüssigkeit der ersten Phase zugeordnet ist. Alternativ kann auch die Farbpalette manuell geschrieben werden. Mit der angepassten Farbpalette als Parameter kann das Bild anschließend erneut umgewandelt werden. Das Programm geht dabei alle Koordinaten durch und wendet die BFS, eingebettet in der Funktion `floodFillSegment`, bei jeder unbeschriebenen Koordinate an.

```
for (long z = 0; z < nz; ++z) {
      for (long y = 0; y < ny; ++y) {
        for (long x = 0; x < nx; ++x) {

          if (ScalarData_getValue(resultScalarData, x, y, z)
              == UNMARKED) {
            PixelState currentPixel = { .x = x, .y = y, .z = z, };

            float colorValueToSet = (float) next_color_value++;
            floodFillSegment(currentPixel, colorValueToSet,
                  out->scalardata[0], resultScalarData,
                  singlePhaseScalarField);
            }
        }
      }
}
```

2.2 Programmiersprache C

C ist eine universell anwendbare Programmiersprache, welche eher als eine maschinennahe Programmiersprache gesehen wird. C ist nützlich zur Programmierung von zeitkritischen Anwendungen, wie z.B Betriebssystemen oder Echtzeitprozessen. Anders als bei einer "*low-level language*" basiert C Code aber nicht auf einer bestimmten Hardware oder einem bestimmten Betriebssystem und ist daher davon unabhängig. Die Sprache enthält auch Eigenschaften von höheren Programmiersprachen und somit ist der Code einfacher zu lesen und auch einfacher zu debuggen (nach [6]).

Da Wert auf effizienten und doch übersichtlichen Code gelegt wird, wird C zur Programmierung der Anwendungen für die vorliegende Aufgabe. Dabei wurde auch im Verlauf des Projektes von innerhalb des Instituts entwickelten und verwendeten Bibliotheken und Programmen Gebrauch gemacht.

2.3 Entwicklungsumgebung CLion

Für eine effiziente Implementierung wurde die Integrierte Entwicklungsumgebung CLion genutzt, welche C als Programmiersprache unterstützt. CLion bietet eine übersichtliche Syntaxhervorhebung, Formatierungs-Tastenkürzel und Werkzeuge zum Anpassen von Code. Zusätzlich schlägt CLion Verbesserungsvorschläge bei Fehlern direkt vor und hat eine übersichtliche Navigationshilfe, auch zu Bibliotheksfunktionen.

2.4 Simulationsprogramme

Zur Simulation der Schäume wird das Softwarepaket PACE3D verwendet, welches als Kern einen Phasenfeldlöser sowie eine breit aufgestellte Auswahl an Pre- und Post-processing-Tools enthält. Unser Programm ist eingebettet in das Preprocessing-Tool `image2scalardata`, was auch zu PACE3D gehört.
Im Projekt wurde hauptsächlich von einem für 2-dimensionale Simulationen optimierten Löser zum Simulieren Gebrauch gemacht. Da bei der Kalibrierung viele Simulationen durchgeführt wurden, wurde, um Zeit zu sparen, eine effizientere Variante des Lösers genutzt.

2.5 Parameter bei Simulationen

Die Simulation lässt sich durch Parameter konfigurieren. Zur Einstellung der Parameter wird am Anfang einer Simulation eine Textdatei eingelesen. Sie enthält verschiedene Parameter über Faktoren wie Druck oder Schwerkraft, welche die Simulation beeinflussen. Die Parameter beeinflussen das Verhalten der Phasen zueinander und haben die Intention physikalische Parameter in diesem Rahmen simulieren zu können. Die Textdatei enthält auch die Information wie lange die Simulation laufen soll. Die Simulation berechnet jeweils für einen Zeitschritt die neue Morphologie des Schaumes. Wie viele Zeitschritte insgesamt die Simulation anhält und in welchen Abständen ein neues Skalarfeld mit der jeweiligen Morphologie ausgeschrieben werden soll, lässt sich auch in der Textdatei angeben. Im Projekt wurden hauptsächlich die Parameter Schwerkraft und Koaleszenz variiert.
Der Parameter **Schwerkraft** übt eine treibende Kraft in eine bestimmte Richtung auf die Phasen aus. Die Richtung hängt von dem Standort ab, welcher mittels Koordinaten angegeben werden kann. Abhänging von der Dichte der jeweiligen Phase wirkt sich diese Kraft unterschiedlich stark aus und resultiert in einer anderen treibenden Kraft, was zu einer Veränderung der Morphologie im Verlauf der Simulation führt. In den Simulationen wurde hauptsächlich die Intensität der Schwerkraft variiert, sowie der Ursprungsstandort der Schwerkraft. Der Standort wurde nur in der Tiefe variiert, und lag unterhalb des zu simulierenden Schaubildes.
Der Parameter **Koaleszenz** dient zur Beschreibung des Vorgangs, bei der Phasen miteinander verschmelzen. Ob dies geschieht, hängt von einer bestimmten

Wahrscheinlichkeit ab, die als Parameter festgelegt werden kann und in den Simulationen variiert wurde. Bei Schaum findet Koaleszenz durch das Platzen der Lamellen statt, die Blasen voneinander trennen. In Abb. 9 ist zu sehen, wie am Rand der Blase die Skalarwerte sich in einem Bereich zwischen 1 und 0 befinden (von Orange zu Rot). Wie in Abschnitt 2.1.1 beschrieben, gehören Skalare mit dem Wert 0 nicht zu der Phase. Im Phasenmodell ist es jedoch schwierig mit harten Übergängen von 1 auf 0 zu rechnen. Der rote-orangene Bereich stellt also einen Übergangsbereich der Phase dar, der im Phasenmodell anstatt einem harten Übergang genutzt wird. Überlappen solche Bereiche von zwei angrenzenden Phasen, besteht die Möglichkeit, dass diese miteinander verschmelzen.

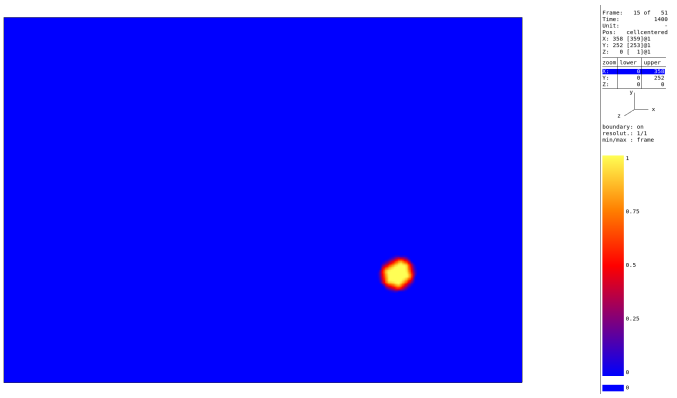


Abb. 9: Darstellung einer Blase

2.6 Untersuchungskriterien der Simulationen

Es ist wichtig, dass die Simulation reale Schäume möglichst realistisch simuliert. Dementsprechend wurde darauf Wert gelegt, dass Beobachtungen, die bei realen Schäumen auftreten, auch in der Simulation zu sehen sind.
Bei der Schwerkraft wurde darauf geachtet, dass Phasen sich natürlich ausbreiten und eine Kugelform anstreben, welches dem natürlichen Verhalten Minimaloberflächen auszubilden entspricht. Die Flüssigkeit soll auch in eine eindeutige Richtung durch die Schwerkraft verdrängt werden. Beobachten lässt sich das z.B durch dünner werdende Lamellen, wo die Flüssigkeit entweicht. Die entwichene Flüssigkeit sammelt sich in größere Bereiche an und bewegt sich in eine bestimmte Richtung. Der Ursprung der Schwerkraft sollte so klar zu sehen sein. Im besten Fall soll dieser Ursprung, den man durch Beobachten der Sammlung der Flüssigkeit ermittelt, auch mit dem Standort ungefähr übereinstimmen der als Parameter festgelegt wurde.
Bei der Koaleszenz ist es wichtig, dass Phasen miteinander verschmelzen, aber sich für das Gesamtsystem wieder ein Gleichgewicht einstellt. Verschmolzene Phasen nähern sich dabei wieder ihrer Kugelform an. Koaleszenz ist dabei ein Vorgang, welcher im Mikrosekundenbereich stattfindet, eine Gleichgewichtseinstellung kann Millisekunden bis Sekunden dauern. Es ist also wichtig dass der zeitliche Abstand zwischen Koaleszenz-Vorgängen ausreichend ist, damit auch eine Gleichgewichtseinstellung stattfinden kann.

Es ließ sich auch die Größenverteilung der Blasen analysieren. Im Idealfall entspricht diese einer Normalverteilung: mit einer großen Anzahl an Blasen mittlerer Größe, und einer Abnahme der Anzahl in beiden Richtungen (Abb 10).

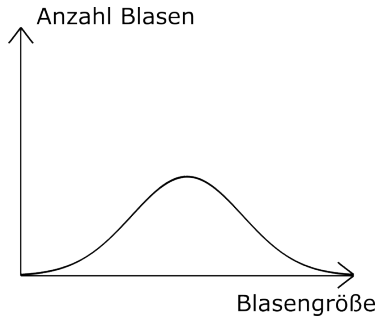


Abb. 10: Beispiel einer Normalenverteilung

Zusätzlich ist die Anzahl der Blasen ein wichtiges Kriterium, da die Blasen bei Koaleszenz-Vorgängen weniger werden.

3 Anwendung des Programms und weiterführende Simulationen

Das Programm wird dazu verwendet binarisierte Bilder in Skalarfelder umzuwandeln, damit diese analysiert und für Simulationen genutzt werden können.

3.1 Bilddigitalisierung

Die Aufnahmen der echten Schäume stammen von Cuifeng Zhao aus der Arbeitsgruppe von Prof. Krastev, welche an der Hochschule Reutlingen/NMI Tübingen tätig ist. Diese Aufnahmen wurden mit dem in Abschnitt 2.3 beschriebenen zusätzlichem Feature des Programms bilddigitalisiert und für Simulationen genutzt. In Abb. 11 ist eine der Aufnahmen vor und nach der Bilddigitalisierung zu sehen.

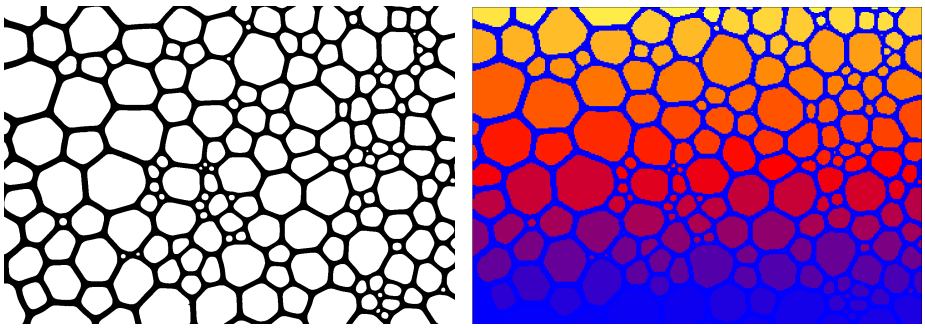


Abb. 11: Binarisiertes Bild und Skalarfeld

Die Dateien befinden sich nun im Format von PACE3D und können entsprechend weiterverwendet werden.

3.2 Simulationsergebnisse

Wie in Abschnitt 2.4 und 2.5 beschrieben, wird beim Anwenden des Phasenfeldlösers eine bilddigitalisierte Aufnahme und eine Textdatei mit den Parametern übergeben. Nach Ablauf der Simulation ergibt sich eine Reihe an Skalarfeldern, die in gleichmäßigen Zeitschritten den zeitlichen Verlauf des simulierten Verhaltens des Skalarfeldes darstellen. In Abb. 12 sind Ausschnitte einer solchen Reihe an Skalarfeldern zu sehen.

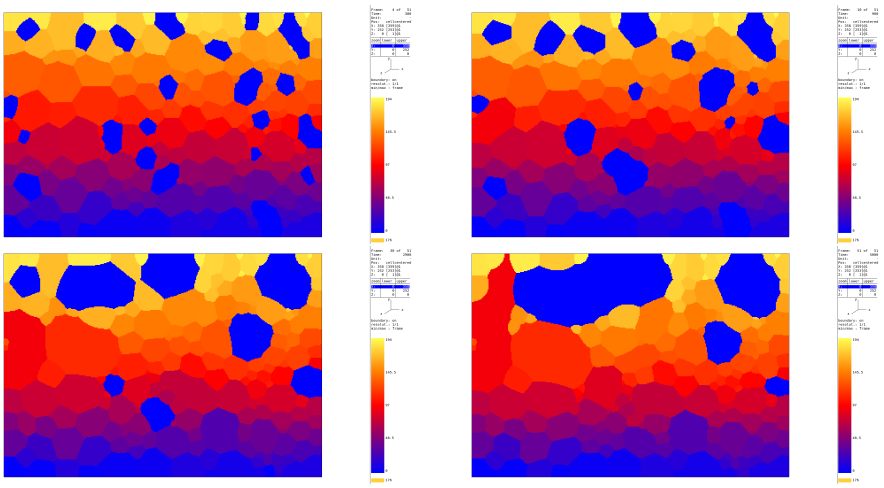


Abb. 12: Kalibrierte Simulation

Bei der Kalibrierung der Simulationen wurde darauf geachtet, dass Kriterien aus Abschnitt 2.6 in den zeitlichen Verläufen, die in den Bilderreihen dargestellt werden, erfüllt werden.

Die Kalibrierung selbst fand durch leichtes Abändern der Parameter der Simulation statt. An den Ergebnissen der Simulationen mit den jeweiligen Parametern konnten die Auswirkungen der Änderungen untersucht werden.

In Abb. 13 sind beispielsweise zwei Simulationen mit verschiedenen Intensitäten an Koaleszenz zu sehen, mit einer höheren Intensität auf der linken Seite. Da Koaleszenz in benachbarten Bereichen bei zu hoher Intensität kurz hintereinander stattfinden kann, kann das Gesamtsystem nicht rechtzeitig in den Gleichgewichtszustand gelangen. So bilden sich Formen aus, die deutlich von rein konvexen Flächen abweichen. Die Orte der Koaleszenz-Ereignisse unterscheiden sich auch von Simulation zu Simulation, da sie in der Simulation nur mit einer gewissen Wahrscheinlichkeit an einer Stelle auftreten.

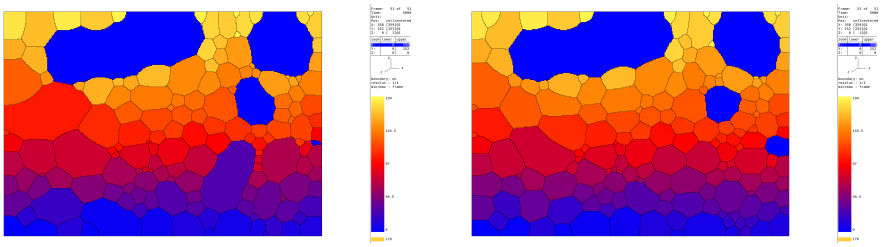


Abb. 13: Höhere und niedrigere Koaleszenz

In Abb. 14 ist der Einfluss der Schwerkraft zu sehen. Man kann erkennen, dass auf der rechten Seite mehr, auch etwas kleinere Flüssigkeitsblasen vorhanden sind. Auch befinden sich die größeren Flüssigkeitsblasen auf der linken Seite weiter oben. Das liegt an einer stärkeren Kraft nach oben auf der linken Seite, die zu einer stärkeren Verdrängung der Flüssigkeit führt. Die Blasen auf der linken Seite sind auch durchschnittlich größer.

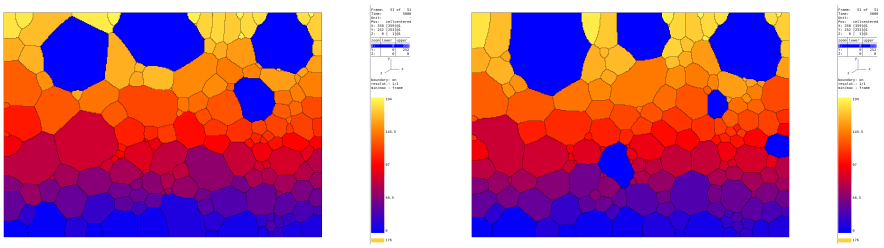


Abb. 14: Höhere und niedrigere Schwerkraft

Anhand der sichtbaren Veränderungen in den verschiedenen Simulationen werden sukzessive Anpassungen für die nächsten Simulationen festgelegt. Nach mehreren Anpassungen liegt schließlich eine kalibrierte Simulation vor.

3.3 3D-Rekonstruktion von Schaumstrukturen

Es ist möglich, anhand der umgewandelten Bilder von 2D Schäumen 3D Schäume zu rekonstruieren. Ein solcherart rekonstruierter 3D-Schaum kann ein Bild darüber liefern, wie der flüssige Schaum aussehen wird, wenn er verfestigt wird. Somit hilft die Rekonstruktion zum Verständnis der Struktur und Eigenschaften des Schaums.
Die Größenverteilung der Blasen des 2D Schaums wird analysiert, und basierend darauf wird eine Gauß'sche Normalverteilungskurve möglichst gut angepasst. Die Blasengrößenverteilung des Schaumes in Abb. 11 ist im Histogramm aus Abb. 15 dargestellt.

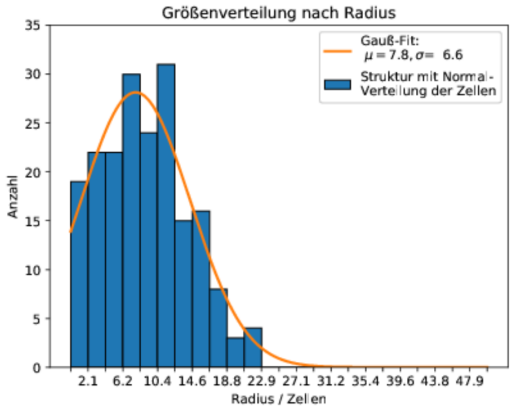


Abb. 15: Blasengrößenverteilung der 2D-Simulation und Anpassung der Normalverteilungskurve

Der Mittelwert μ der Normalverteilung wird als Parameter genutzt um den 3D Schaum algorithmisch zu generieren. Für die Rekonstruktion des 3D Schaums in Abb. 16 wurde der Mittelwert 7.8 aus der Größenverteilungsanalyse in Abb. 15 benutzt.
Der so rekonstruierte Schaum lässt sich auf verschiedene Art und Weise darstellen. In der Voronoi-Darstellung auf der linken Seite von Abb. 16 sind die ver-

schiedenen Blasen durch verschiedene Farben und ihre Form erkennbar. In der Isosurface-Darstellung auf der rechten Seite sind die innere Struktur und die Grenzen zwischen den Blasen als feste Stege sichtbar dargestellt.

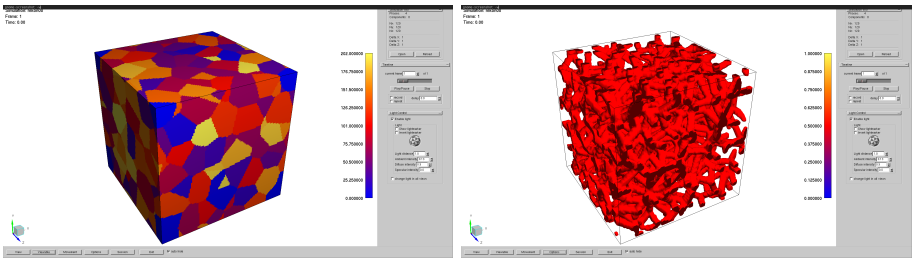


Abb. 16: Voronoi Darstellung

Eine weitere Rekonstruktion ist in Abb. 18 zu sehen. Hier wurde eine Momentaufnahme des Schaums in einem späteren Stadium des Experiments genutzt (Abb. 17)

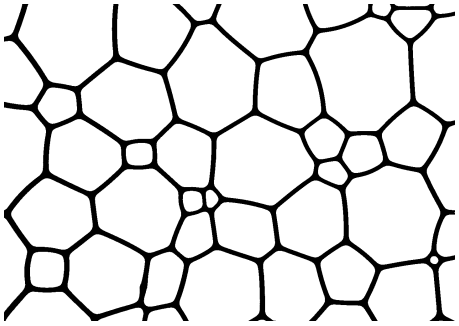


Abb. 17: Schaum im späteren Stadium

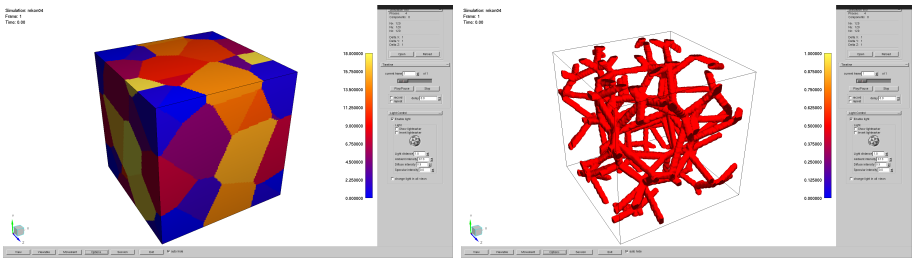


Abb. 18: Rekonstruktion eines weiteren Schaums

Daran kann man erkennen, dass, durch Koaleszenz-Ereignisse verursacht, die Blasen größer sind und weniger vorhanden sind. Wie man also sieht, finden die Koaleszenz-Ereignisse so wie sie simuliert werden, auch in den eigentlichen Experimenten statt und führen zu ähnlichen Resultaten.

4 Schlussbetrachtung

Es ist in der vorliegenden Arbeit gelungen ein bestehendes Programm so zu erweitern, dass Bilder in Skalarfelder umgewandelt werden, welche nicht nach Farben, sondern nach zusammenhängenden Segmenten kodiert sind. Mit Hilfe unserer Programmversion konnten Simulationen basierend auf experimentellen Aufnahmen durchgeführt werden. Dies ermöglichte die Kalibrierung der Parameter einer Schaumsimulation. Auch ist es uns gelungen ausgehend von den vorliegenden 2D-Aufnahmen realer Schäume eine 3D-Rekonstruktion des Schaumes anzufertigen. Der in dieser Arbeit implementierte Algorithmus stellt also eine gute Ergänzung bestehender Werkzeuge auf dem Feld der Schaumforschung da.

Verbesserung der Simulation und des Programms
In der Arbeit wurde die Kalibrierung durchgeführt um einem natürlichen Schaumverhalten nahezukommen. Weitere Kalibrierungen mit Material aus Experimenten sind nötig um das Verfahren zu optimieren und zu verbessern. Eine weitere noch ausstehende Aufgabe besteht darin, Parameter der Simulationen näher zu untersuchen, um sie einfacher mit physikalischen Parametern vergleichen zu können. Beispielsweise wirkt die Schwerkraft in unserer Simulation nach oben, obwohl der Ursprung unterhalb des simulierten Schaumes angegeben wurde.
Weitere Anpassungen der Bedienungsfreundlichkeit und effizienten Nutzung des Bilddigitalisierungsverfahrens müssen noch erfolgen. Eine Möglichkeit zum Beispiel wäre es, dass eine Koordinate der Flüssigkeit als Parameter angegeben werden wird, so dass, ausgehend von dieser Koordinate, mit der Breitensuche die Flüssigkeit auf diese Weise erkannt werden kann. Somit muss man nicht umständlich die Farbe der Flüssigkeit in dem Bild über eine Farbpalette angeben. Entweder das Schreiben oder Ausschreiben und Modifizieren einer Farbpalette kann so wegfallen.

Ausblick
Unser Verfahren lässt sich für alle phasengestützten Simulationen einsetzen. Simulationen können parallel oder als Ersatz zu normalen Experimenten durchgeführt werden und Forschungsprozesse effizienter, materialsparender und einfacher gestaltet werden.
Durch die Aufbereitung 2-dimensionaler Aufnahmen von realen Schäumen oder Simulationsergebnissen lassen sich 3D-Schäume rekonstruieren, was eine tiefere Analyse von Experimenten und die Vorhersage der Struktur fester Schäume ermöglicht.
Der Kalibrierungsvorgang kann mit künstlicher Intelligenz und maschinellem Lernen verbunden werden, welche mögliche Parameter für die Simulation zum Erstellen von gewünschten Schaumstrukturen mit gewünschten Eigenschaften vorschlagen können.

5 Danksagung

Wir danken Jana Holland-Cunz und Matthieu Laqua, die uns nicht nur dieses Projekt ermöglicht haben, sondern uns auch im Projekt intensiv unterstützt und betreut haben. Wir danken auch den Kursleitern des Hector-Seminars, insbesondere Norbert Krieg und Dietmar Gruber für die intensive Betreuung während des letzten Jahres, sowie Anke Richert und Paul Bischof für die wertvolle Unterstützung in den vergangenen Jahren. Ganz besonders danken wir Josephine und Dr. Hans-Werner Hector, ohne deren Stiftung dieses Projekt nicht möglich gewesen wäre.

6 Quellen

6.1 Textquellen

[1] URL: https://de.wikipedia.org/wiki/Schaum. (Zugriff: 30.07.2022).

[2] URL: https://www.iwu.fraunhofer.de/de/forschung/leistungsangebot/kompetenzen-von-a-bis-z/leichtbau/Werkstoffe/Metallische-Leichtbauwerkstoffe/Metallschaum.html. (Zugriff: 30.07.2022).

[3] URL: https://de.wikipedia.org/wiki/Portable_Network_Graphics. (Zugriff: 10.07.2022).

[4] URL: https://de.wikipedia.org/wiki/Skalarfeld. (Zugriff: 10.07.2022).

[5] URL: https://de.wikipedia.org/wiki/Breitensuche. (Zugriff: 10.07.2022).

[6] Dennis M. Ritchie Brian W. Kernighan. *The C Programming Language, 2nd Edition*. Addison-Wesley. Pearson, 1988. ISBN: 9780131103627.

[7] J. B. Clarke et al. "Definitions of terms relating to phase transitions of the solid state (IUPAC Recommendations 1994)". In: *Pure and Applied Chemistry* 66.3 (1994), pp. 577–594. DOI: doi:10.1351/pac199466030577. URL: https://doi.org/10.1351/pac199466030577.

[8] J. Hötzer et al. "The parallel multi-physics phase-field framework Pace3D". In: *Journal of Computational Science* 26 (2018), pp. 1–12. ISSN: 1877-7503. DOI: https://doi.org/10.1016/j.jocs.2018.02.011. URL: https://www.sciencedirect.com/science/article/pii/S1877750317310116.

[9] Dr. Franz Meyer. *Metallschaum - ein Werkstoff für die Wärmetechnik*. URL: https://api.deutsche-digitale-bibliothek.de/binary/b11cc4ed-d9bf-4a6a-8ac2-b80eb1bd5ede.pdf. (Zugriff: 30.07.2022).

[10] Cosima Stubenrauch et al. "Emulsion and Foam Templating—Promising Routes to Tailor-Made Porous Polymers". In: *Angewandte Chemie International Edition* 57.32 (2018), pp. 10024–10032. DOI: https://doi.org/10.1002/anie.201801466. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/anie.201801466.

6.2 Bildquellen

Alle Bilder sind Screenshots aus Institut-eigenen Programmen wie XSimViewer oder wurden mit Inkscape und Krita erstellt.

7 Selbständigkeitserklärung

Hiermit versichern wir, dass wir diese Arbeit unter der Beratung durch Jana Holland-Cunz, Matthieu Laqua und Norbert Krieg selbstständig verfasst haben und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt wurden, sowie Zitate kenntlich gemacht haben.

Nhat Phi Ho

Fynn Wagner

8 Anhang

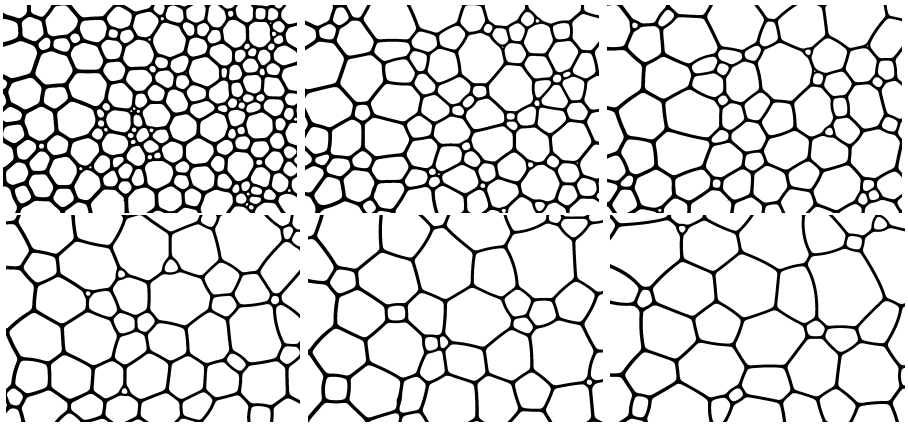


Abb. 19: Experimentelle Aufnahmen

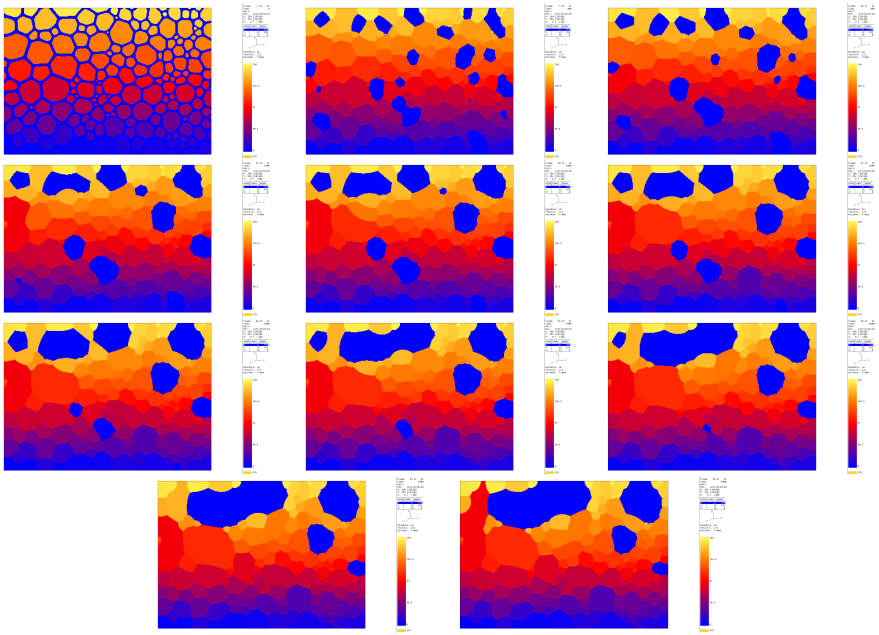


Abb. 20: Kalibrierte Simulation

```
#define UNMARKED -1.0f

typedef struct pixel_state_s {
  long x;
  long y;
  long z;
} PixelState;

PixelState n6_offset_list[6] = {
        {0, 0, 1},
        {0, 0, -1},
        {0 , 1 , 0},
        {0, -1, 0},
        { 1, 0, 0},
        { -1, 0, 0},
};

static void floodFillSegment(PixelState startPixel, float
    colorValueToSet, ScalarData* original, ScalarData* result,
    ScalarData* singlePhaseResult) {
  // save the color of the current segment in the original (
    basically what color the algorithm should "look for")
  float colorValueOriginal = ScalarData_getValue(original,
    startPixel.x, startPixel.y, startPixel.z);

  // put start pixel on the heap
  PixelState *startPixel_heap = (PixelState *) Malloc(sizeof(
    PixelState));
  *startPixel_heap = startPixel;

  Queue *bfsQueue = newQueue();
  Queue_enqueue(bfsQueue, startPixel_heap);
  // mark start pixel as visited
  ScalarData_setValue(result, startPixel.x, startPixel.y,
    startPixel.z, colorValueToSet);
  if (singlePhaseResult != NULL) {
    ScalarData_setValue(singlePhaseResult, startPixel.x, startPixel
    .y, startPixel.z, 1.0f);
  }

  // start BFS
  while (!Queue_isEmpty(bfsQueue)) {
    PixelState *currentPixel = Queue_dequeue(bfsQueue);

    // check neighbours
    for (int i = 0; i < 6; ++i) {
      long neighbour_x = currentPixel->x + n6_offset_list[i].x;
      long neighbour_y = currentPixel->y + n6_offset_list[i].y;
      long neighbour_z = currentPixel->z + n6_offset_list[i].z;#

      // if the pixel is inside the domain and not visited yet and
    of the same color as in the original, create a new
      // pixel on the heap and add it to the queue
      if (SimGeo_isInsideDomain(original->simgeo, neighbour_x,
    neighbour_y, neighbour_z) &&
          ScalarData_getValue(result, neighbour_x, neighbour_y,
    neighbour_z) == UNMARKED &&
          ScalarData_getValue(original, neighbour_x, neighbour_y,
    neighbour_z) == colorValueOriginal) {

        PixelState *neighbourPixel = (PixelState *) Malloc(sizeof(
    PixelState));
```

```
        neighbourPixel->x = neighbour_x;
        neighbourPixel->y = neighbour_y;
        neighbourPixel->z = neighbour_z;
        Queue_enqueue(bfsQueue, neighbourPixel);

        // mark neighbour as visited
        ScalarData_setValue(result, neighbour_x, neighbour_y,
    neighbour_z, colorValueToSet);
        if (singlePhaseResult != NULL) {
          ScalarData_setValue(singlePhaseResult, neighbour_x,
    neighbour_y, neighbour_z, 1.0f);
        }
      }
    }

    Free(currentPixel);
  }
}

if (individual_grains == true) {
    // create the result holding all the colored segments
    resultScalarData = createInitializedScalarfield(UNMARKED, out);

    // start with 1 to assure that black or the set color in '
    color_first_phase' gets to be the 0th phase
    int next_color_value = 0;
    bool zero_phase_found = false;
    char *phaseName = Calloc(64, sizeof(char));

    // first run for the zero phase
    for (long z = 0; z < nz; ++z) {
      for (long y = 0; y < ny; ++y) {
        for (long x = 0; x < nx; ++x) {
          // if the pixel was not yet visited AND THE PHASE IS THE
    ZERO PHASE, start a flood fill / simple bfs from there
          if (ScalarData_getValue(resultScalarData, x, y, z) ==
    UNMARKED &&
              ScalarData_getValue(out->scalardata[0], x, y, z) ==
    0.0f) { // the phase zero
            PixelState currentPixel = { .x = x, .y = y, .z = z, };

            // create a new scalar field for the individual phase
            ScalarData *singlePhaseScalarField =
    createInitializedScalarfield(0.0f, out);

            float colorValueToSet = (float) next_color_value++;
            floodFillSegment(currentPixel, colorValueToSet, out->
    scalardata[0], resultScalarData, singlePhaseScalarField);

            // add individual phase as scalar field to the simgeo
    and simbench
            String_createFormatedString(&phaseName,"phase%.0f",
    colorValueToSet);
            SimBench_addFileByFieldname(out, phaseName, FD_PHI);
            SimGeo_preliminaryAddFile(out->simgeo, &(out->
    fieldnamestable), &(out->fieldnamestablecount),
    FieldDescriptor_define(FD_PHI->modulename, phaseName, FD_PHI->
    filetype));
            //ScalarData_copy(out->scalardata[zero_phase_found ?
    next_color_value : next_color_value - 1],
    singlePhaseScalarField);
```

```
            ScalarData_copy(out->scalardata[next_color_value],
    singlePhaseScalarField);
            freeScalarData(singlePhaseScalarField);
          }
        }
      }
    }

    // second run for all other phases
    for (long z = 0; z < nz; ++z) {
      for (long y = 0; y < ny; ++y) {
        for (long x = 0; x < nx; ++x) {
          // if the pixel was not yet visited, start a flood fill /
     simple bfs from there
          if (ScalarData_getValue(resultScalarData, x, y, z) ==
    UNMARKED) {
            PixelState currentPixel = { .x = x, .y = y, .z = z, };

            // create a new scalar field for the individual phase
            ScalarData *singlePhaseScalarField =
    createInitializedScalarfield(0.0f, out);

            float colorValueToSet = (float) next_color_value++;
            floodFillSegment(currentPixel, colorValueToSet, out->
    scalardata[0], resultScalarData, singlePhaseScalarField);

            // add individual phase as scalar field to the simgeo
    and simbench
            String_createFormatedString(&phaseName,"phase%.0f",
    colorValueToSet);
            SimBench_addFileByFieldname(out, phaseName, FD_PHI);
            SimGeo_preliminaryAddFile(out->simgeo, &(out->
    fieldnamestable), &(out->fieldnamestablecount),
    FieldDescriptor_define(FD_PHI->modulename, phaseName, FD_PHI->
    filetype));
            ScalarData_copy(out->scalardata[next_color_value],
    singlePhaseScalarField);
            freeScalarData(singlePhaseScalarField);
          }
        }
      }
    }

    // finally overwrite the scalardata in the simbench
    ScalarData_copy(out->scalardata[0], resultScalarData);
    freeScalarData(resultScalarData);

    // because we added more files
    SimBench_finishInit(out, force);
    SimGeo_commitPreliminaryAddFile(out->simgeo, &(out->
    fieldnamestable), &(out->fieldnamestablecount));

    Free(phaseName);
}

static ScalarData *createInitializedScalarfield(float scalarValue,
    SimBench *simBench) {
  ScalarData *scalarData = newScalarData();
  ScalarData_init(scalarData, simBench->simgeo);

  long nx = (long) simBench->simgeo->Nx;
  long ny = (long) simBench->simgeo->Ny;
```

```
146   long nz = (long) simBench->simgeo->Nz;
147
148   for (long z = 0; z < nz; ++z) {
149     for (long y = 0; y < ny; ++y) {
150       for (long x = 0; x < nx; ++x) {
151         ScalarData_setValue(scalarData, x, y, z, scalarValue);
152       }
153     }
154   }
155   return scalarData;
156 }
```