Institut für Parallele und Verteilte Systeme
Simulation Großer Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit Nr. 205

Die Finite–Elemente–Methode mit dynamisch–adaptiven kartesischen Gittern

Benjamin Maier

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Miriam Mehl
Betreuer/in:	Dipl.-Inf. Michael Lahnert
Beginn am:	1. April 2015
Beendet am:	1. September 2015
CR-Nummer:	G.1.8

Kurzfassung

In dieser Arbeit wird ein zweidimensionales Strömungsproblem, beschrieben durch die Navier–Stokes–Gleichungen, auf einem dynamisch adaptiven Gitter mithilfe der Finite–Elemente–Methode berechnet. Es wird der komplette Ablauf der Berechnung anhand einer Implementierung vorgestellt. Als Datenstruktur werden Quadtrees verwendet, die mit einem bottom–up–Algorithmus nach [SSB08] parallel erzeugt werden können. Basierend auf der Vorticity wird das Gitter während der Simulation verfeinert oder vergröbert. Es wird die parallele Skalierbarkeit untersucht und für ein reguläres Gitter ein Laufzeitvergleich mit einer Referenzimplementierung ohne Quadtrees durchgeführt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Algorithmen

Verzeichnis der Dateien

1. Einführung

Zur Simulation des zeitlichen Verhaltens eines Stoffes ist eine Auflösung auf verschiedenen Skalen möglich. Bei Wahl der Mikroskala wird in *Molekulardynamik*–Simulationen die Substanz durch die Summe ihrer Atome oder Moleküle dargestellt, die miteinander über Kräfte wechselwirken. Je nach Fragestellung werden auch quantenchemische Effekte berücksichtigt. Die potentielle Energie wird häufig basierend auf dem *Lennard–Jones*–Potential modelliert. Charakteristisch für diese Art der Simulation sind die Möglichkeit, intrinsische Größen, wie Wärmekapazitäten oder Leitfähigkeiten zu bestimmen, sowie die notwendige Einschränkung auf kurze Zeitintervalle bei kleinsten räumlichen Gebieten. Da variable Systemgrößen flukturieren, muss zur Auswertung ein Mittelwert gebildet werden. [And80, CP85].

Auf der nächst höheren Skala, der Mesoskala, wird meist mit gröberen Einheiten als Atomen simuliert. Die Wechselwirkungen werden dabei entsprechend abstrahiert. Die Eigenschaft der Fluktuation der beschriebenen Größen besteht ebenfalls. Ein häufiger Vertreter für die Simulation von Strömungen auf der Mesoskala ist die *Lattice–Boltzmann*–Methode [CD98].

Im Gegensatz dazu wird auf der Makroskala ein Kontinuum betrachtet, das durch entsprechende partielle Differentialgleichungen beschreibbar ist, welche makroskopische Systemgrößen wie Druck, Temperatur sowie Geschwindigkeit zueinander in Beziehung setzen.

Ebenfalls möglich ist ein heterogener *Multiskalen*–Ansatz, bei dem die Beschreibung auf mehreren Skalen erfolgt [ELR$^+$07].

In dieser Arbeit wird die Beschreibung der Strömung eines kontinuierlichen Fluids auf der Makroskala betrachtet. Dies beinhaltet das Aufstellen und Lösen partieller Differentialgleichungen mit sinnvollen Randbedingungen auf einem vorgegebenen Gebiet. Dabei stellt sich das Finden einer analytischen Lösung häufig schwierig dar, wie im Beispiel der Navier–Stokes–Gleichungen, zu denen bisher noch nicht einmal die Existenz einer vernünftigen allgemeine Lösung nachgewiesen werden konnte [Fef00]. Um dennoch die gewünschten Informationen zu erhalten, können numerische Verfahren, wie die Finite–Elemente–Methode (FEM) zu Hilfe genommen werden. Dabei wird der betrachtete Zeitraum und das betrachtete Rechengebiet diskretisiert. Die beschreibende Differentialgleichung wird in eine sogenannte schwache Form überführt, deren Einhaltung dann in jedem Teilgebiet der räumlichen Zerlegung gefordert wird.

Die Art der räumlichen Diskretisierung hat Auswirkungen auf die *Genauigkeit* der so ermittelten numerischen Lösung und auf die *Effizienz* und damit Praktikabilität des Verfahrens. Deshalb ist es von Interesse, die Diskretisierung unter diesen beiden Gesichtspunkten möglichst gut zu wählen.

Ein möglicher Ansatz zum Erreichen hoher Genauigkeit ist die Verwendung einer lokal adaptiven Diskretisierung, die eine genauere Auflösung in interessanteren Teilbereichen des Simulationsgebietes ermöglicht. Wenn klare Kriterien für den erwarteten Informationsgehalt an einem Punkt bekannt sind, können die Teilbereiche mit feinerer Diskretisierung auch während der Simulation automatisch angepasst werden. Dies wird als *dynamische Adaptivität* bezeichnet.

Der Gesichtspunkt der Effizienz ist wichtig, um umfangreiche Probleme in praktikabler Zeit berechnen zu können. Die Grenzen der Größe von praktisch berechenbaren Problemen werden von Höchstleistungsrechnern erreicht, die oftmals über 10^5 Prozesse nebenläufig ausführen können. Um einen Algorithmus auf solchen High-Performance-Rechnern ausführen zu können, muss er eine gute parallele Skalierbarkeit aufweisen.

Im Rahmen dieser Arbeit wurde ein Programm entwickelt, das die Strömung einer Flüssigkeit in einem zweidimensionalen Gebiet auf der Makroskala simuliert. Dabei wird die FEM mit einer dynamisch adaptiven Diskretisierung angewandt. Es wird darauf geachtet, dass die Algorithmen dieser Berechnung effizient parallel ausgeführt werden können.

Im folgenden Kapitel 2 werden zunächst die verwendeten Datenstrukturen beschrieben. Das Modell zur Beschreibung des Strömungsproblems wird aufgestellt und in die schwache Form zur Anwendung der Finiten–Elemente–Methode überführt. Anschließend wird erläutert, wie die Berechnung auf der Datenstruktur umgesetzt wird. Im nächsten Kapitel 3 wird dann zunächst die Bedienung des Programms erklärt. Dabei wird auf die Spezifizierung von Geometrie und Parametern in Eingabedateien und sowie die Auswertung der Ausgabedateien eingegangen. Es wird daraufhin ein Überblick über die Implementierung gegeben und die verwendeten Lösungsverfahren werden genauer beschrieben. In Kapitel 4 findet schließlich eine Validierung statt. Außerdem werden Laufzeituntersuchungen angestellt und die Verwendung der dynamischen Adaptivität anhand von Beispielen demonstriert. Schließlich wird in Kapitel 5 das Vorgehen zusammengefasst und es wird ein Ausblick auf mögliche fortsetzende Untersuchungen gegeben.

2. Methoden

2.1. Verwendetes Gitter

Um die Finite–Elemente–Methode auf einem zweidimensionalen Berechnungsgebiet anwenden zu können, wird zunächst eine räumliche Diskretisierung vorgenommen. Diese wird hier durch die Zerlegung des Rechengebiets in quadratische Elemente erreicht. Um räumliche Adaptivität zu erhalten, können die Elemente unterschiedliche Seitenlängen haben, die jedoch immer ganzzahlige Vielfache voneinander sind. Der hier verwendete Ansatz, der eine Diskretisierung mit diesen Anforderungen liefert, ist die Verwendung eines *Quadtrees*. Im folgenden Kapitel 2.2 werden Quadtrees eingeführt. Die Darstellung folgt dabei den Ausführungen in [SSA+07]. Wie dort ebenfalls beschrieben wird, können Quadtrees effizient parallel erzeugt werden. Der in [SSA+07] vorgestellte Algorithmus zum Erzeugen von Quadtrees wird in Kapitel 2.2.3 anhand von Beispielen skizziert.

2.2. Quadtrees

2.2.1. Grundlegende Begriffe

Als *Quadtree* bezeichnet man einen Baum, bei dem jeder Knoten maximal 4 Kindknoten besitzt. Diese werden *Quadranten* genannt. Enthält jeder Knoten entweder keine oder genau 4 Kinder, bezeichnet man den Quadtree als *vollständig*. Es gibt einen *Wurzelknoten*, der als einziger keinen Vorgänger hat. Alle anderen Knoten haben genau einen Vorgänger, der auch *Vaterknoten* genannt wird. Knoten ohne Nachfolger heißen *Blätter*, alle anderen Knoten *innere Knoten*. Die maximal 4 Nachfolger eines Knotens sind zueinander *Geschwister*. Alle Vorgänger auf dem Pfad von der Wurzel zu einem bestimmten Knoten werden als die *Vorfahren* dieses Knotens bezeichnet.

Die Länge des Pfades von der Wurzel zu einem bestimmten Knoten wird als dessen *Level* eingeführt. Damit befindet sich die Wurzel auf Level 0. Ein vollständiger Quadtree mit maximalem Level l_{max} hat somit höchstens $4^{l_{max}}$ Blätter.

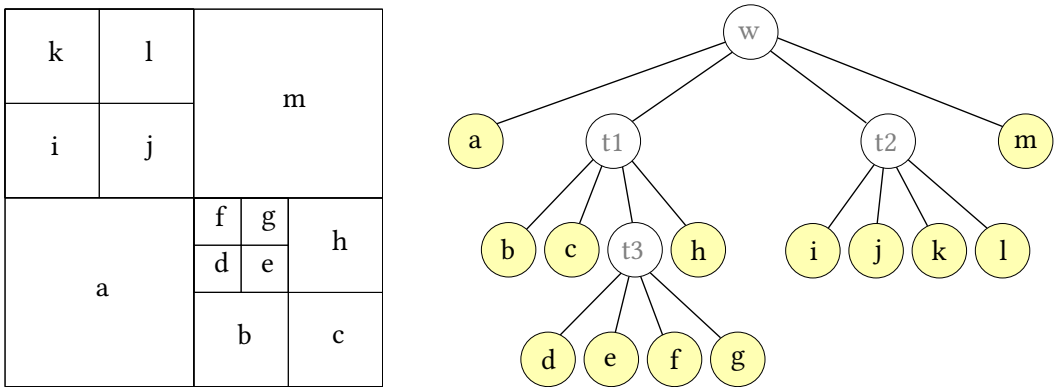
Vollständige Quadtrees eignen sich dazu, ein quadratisches Gebiet in kleinere Quadrate, die den Quadranten entsprechen, zu zerlegen. Mit der Wurzel wird dabei das gesamte Gebiet assoziiert. Die 4 Kinder dieses Knotens zerteilen das Gebiet in 4 Teilquadrate gleicher Größe. Diese Vorschrift wird auf alle weiteren Knoten angewandt, sodass die Teilgebiete, die zu den Blättern gehören, zusammen gerade eine Zerlegung des gesamten Gebiets bilden. Unter dieser Betrachtungsweise werden im Folgenden die Quadranten auch *Elemente* genannt, da sie später die räumliche Diskretisierung der Finite–Elemente–Methode bilden.

In Abbildung 2.1 ist ein beispielhafter vollständiger Quadtree als Zerlegung eines Gebiets in Elemente und als Baum dargestellt.

Für die Zerlegung dreidimensionaler Körper lassen sich analog Oktalbäume verwenden, die nur Knoten enthalten, welche maximal acht Kindknoten besitzen. Die analoge Struktur im Eindimensionalen ist der Binärbaum.

Die Eckpunkte eines Elements werden *Knoten* genannt. Die Kindelemente eines Vaterknotens und die Knoten eines Elements werden wie in Abbildung 2.2 durchnummeriert.

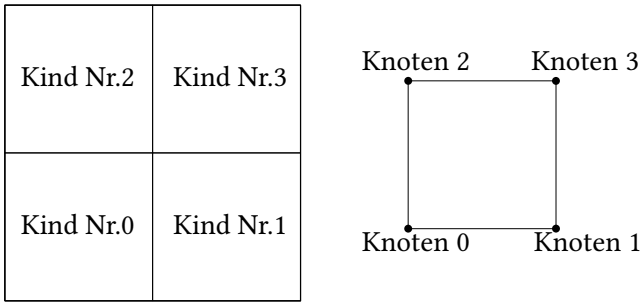
Abbildung 2.1. Ein vollständiger Quadtree mit maximalem Level $l_{max} = 3$.



(a) Quadtree als Zerlegung eines Gebiets

(b) Quadtree als Baum mit Wurzel w, inneren Knoten t1,t2 und t3 und Blättern in gelb

Abbildung 2.2. Nummerierung der Kinder eines Elements und der Eckpunkte



2.2.2. Repräsentation im Speicher

Die Gebietszerlegung durch einen Quadtree ist rekonstruierbar, wenn zu jedem quadratischen Teilgebiet dessen Größe und Position bekannt ist. Nimmt man als ursprüngliches Gebiet die Menge $\Omega = [0,1]^2$ an, lässt sich die Seitenlänge h eines Teilgebiets aus dem Level l des zugehörigen Blattknotens als $h = 2^{-l}$ bestimmen. Die Position kann durch Angabe der Koordinaten der linken unteren Ecke des Teilquadrates spezifiziert werden. Dieser Punkt wird *Anker* des Elements genannt.

Ein Quadtree lässt sich somit im Computer repräsentieren, indem für jeden Blattknoten dessen Level l und die Position des Ankers gespeichert wird. Für die Koordinaten (x, y) des Ankerpunktes reicht die Angabe in Vielfachen der kleinsten Elementgröße $h_{min} = 2^{-l_{max}}$, wenn diese ebenfalls bekannt ist.

Eine mögliche Darstellung dieser Information für jedes Blatt wird durch die *Morton*–Kodierung erreicht. Hier werden zunächst die Binärdarstellungen der Anker–Koordinaten x und y zu einer neuen Binärzahl z vereint, indem nacheinander abwechselnd das nächste Bit aus der Darstellung von y und von x in z eingefügt wird. Anschließend wird die Binärdarstellung des Levels des Knotens angefügt, wie in Abbildung 2.3 dargestellt ist. Man erhält eine binäre ID mit $s = 2\,a + b$ Stellen, wobei a die maximale Stellenanzahl der Binärdarstellung einer Koordinate und b die maximale Stellenanzahl der Binärdarstellung des Levels ist. Es gilt für Quadtrees mit maximalem Level l_{max}:

$$a = l_{max}, \qquad b = \lfloor \log_2(l_{max}) \rfloor + 1.$$

Das größtmögliche Level, für das diese Darstellung funktioniert, hängt von der Anzahl s zur Verfügung stehender Bits pro Blattknoten ab. Für eine Länge von 32 Bit liegt das maximale Level bei $l_{max} = 13$, für 64 Bit ist $l_{max} = 29$ möglich.

Der Quadtree wird nun als sortierte Liste seiner Morton–IDs gespeichert. Diese Art der Darstellung hat die Eigenschaft, dass die Knoten in der Reihenfolge vorliegen, die der Tiefensuche über die Blätter des Quadtrees entspricht. Übertragen auf die Gebietszerlegung entsteht eine charakteristische *Morton-Kurve*, wie in Abbildung 2.4 dargestellt. Da für jeden beliebigen Punkt im Gebiet und einen maximalen Abstand ε eine entsprechende Diskretisierung gefunden werden kann, sodass die Morton–Kurve diesem Punkt näher als ε kommt, handelt es sich um eine raumfüllende Kurve.

Abbildung 2.3. Konstruktion der *Morton*–Kodierung für das Element d des Quadtrees aus Abbildung 2.1(a)

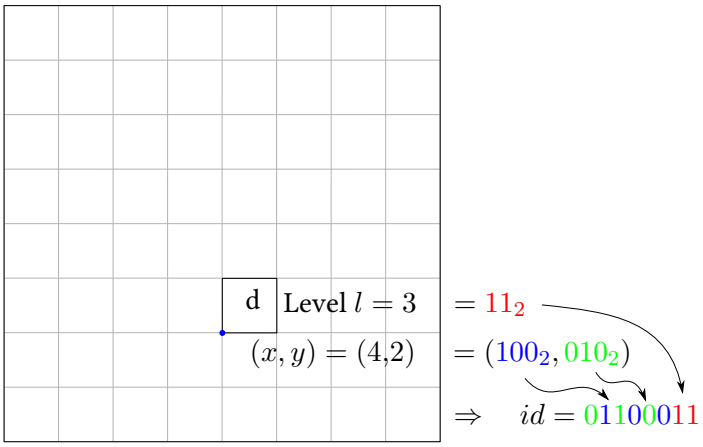
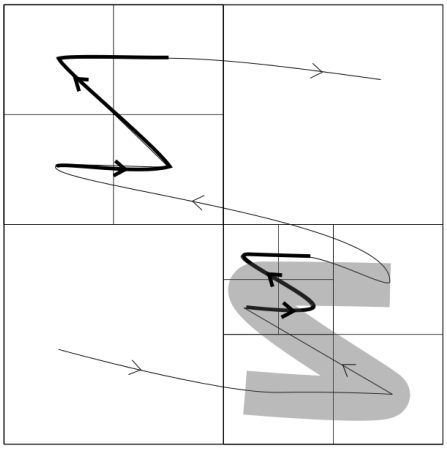


Abbildung 2.4. *Morton*–Kurve. Bei aufsteigender Sortierung der Element–IDs ergibt sich die dargestellte Reihenfolge. Hervorgehoben sind die „Z"–förmigen Strukturen auf unterschiedlichen Ebenen.



Durch diese lineare Repräsentation des Quadtrees im Speicher sind Geschwister- oder Nachbarschaftsbeziehungen nicht explizit gespeichert. Dies steht im Gegensatz zu einer Baum–Datenstruktur, bei der auch alle inneren Knoten gespeichert werden und Nachfolger–Beziehungen durch Zeiger gegeben sind. Um für ein Blatt des Quadtrees also die Nachbarn zu finden oder in einem zweiten Quadtree die Knoten zu ermitteln, die den Vorfahren eines Knotens im ersten Quadtree entsprechen, ist eine Suche nötig. Aus dem bekannten Anker wird als Suchschlüssel eine Morton–Kodierung dieses Ankerpunkts erzeugt. Dafür wird auch ein Wert für das Level benötigt. Ist kein Level vorgegeben, wie z. B. bei der Suche nach Elementen beliebiger Größe an einer bestimmten Stelle, kann als Level der Wert Null verwendet werden. Durch Bisektion in der sortierten Liste der IDs kann dieser Suchschlüssel in maximal $\lceil \log_2 n \rceil$ Schritten gefunden werden,

wenn der Quadtree n Elemente enthält. Ist kein Element mit dieser Element-ID vorhanden, z. B. weil als Level der Wert Null verwendet wurde, ist das Element mit dem gesuchten Anker, falls es ein solches gibt, dasjenige, welches auf die Suchschlüssel–ID folgt.

Es gilt $n \leq 4^{l_{max}}$ und somit:

$$\lceil \log_2 n \rceil \leq \lceil \log_2 4^{l_{max}} \rceil = 2\, l_{max}.$$

Bei vollständiger Ausnutzung eines 32- oder 64-Bit-Datentyps für die Element-IDs benötigt die Suche also maximal 26 bzw. 58 Schritte. Häufig ist die Anzahl jedoch deutlich geringer, da der Quadtree in der Praxis oft nicht nur aus Knoten mit maximalem Level besteht.

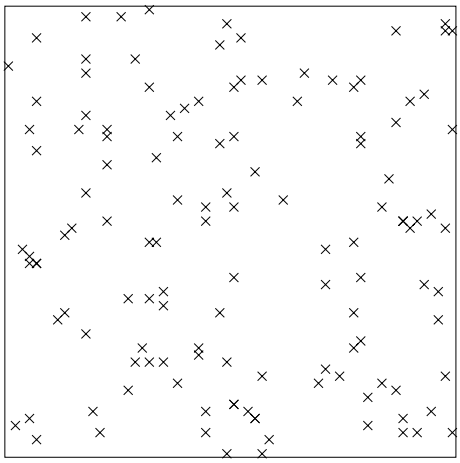
2.2.3. Paralleles Erzeugen des Quadtree

Im Folgenden wird beschrieben, wie ein solcher Quadtree parallel, d.h. durch nebenläufige Ausführung eines Programms in mehreren Prozessen, erzeugt werden kann. Der Algorithmus stammt aus [SSA+07], eine detailliertere Behandlung ist dort zu finden.

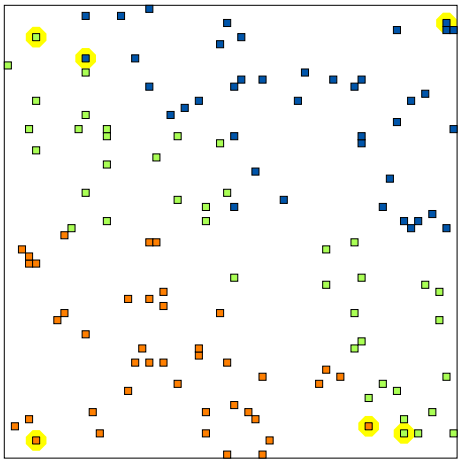
Als Eingabe in den Algorithmus dienen im Gebiet liegende Punkte. Es wird versucht, auf dieser Grundlage einen in der Tiefe durch l_{max} beschränkten Quadtree mit möglichst wenigen Elementen zu erzeugen, sodass in jedem Element maximal ein Punkt liegt. Dementsprechend kann durch die lokale Punktdichte die Diskretisierungsfeinheit an der entsprechenden Stelle vorgegeben werden.

In Abbildung 2.5(a) sind beispielhaft zufällige Punkte aufgeführt, anhand derer der Ablauf des Algorithmus' im Folgenden nachvollzogen wird. Als erstes werden die Punkte gleichmäßig auf alle Prozesse verteilt und dort jeweils Elemente auf Level l_{max} erzeugt, sodass jeder Punkt in einem Element liegt. Diese Elemente werden als *Eingabeelemente* bezeichnet. Der sich dadurch ergebende unvollständige Quadtree ist in Abbildung 2.5(b) dargestellt. Die Farben der Elemente entsprechen dem Prozess, auf dem sie sich befinden.

Abbildung 2.5. Eingabe in den Algorithmus



(a) Punkte als Eingabe in den Algorithmus



(b) Eingabeelemente, d.h. Quadranten mit maximalem Level, verteilt auf 3 Prozesse: Orange = Prozess 0, Grün = Prozess 1, Blau = Prozess 2. Das jeweils erste und letzte Element auf jedem Prozess ist gelb hinterlegt.

Die Elemente werden aufsteigend nach ihrer Morton–Kodierung sortiert. Im nächsten Schritt wird die Region zwischem dem nach dieser Sortierung ersten und dem letzten Quadranten vervollständigt. Diese

Start- und Schlussknoten sind in Abbildung 2.5(b) für jeden Prozess durch gelbe Umrandung hervorgehoben. Die Vervollständigung geschieht, indem im Baum ausgehend vom Startknoten zunächst alle Geschwister mit größerer ID hinzugefügt werden, dann zum Vaterknoten gewechselt wird und wieder alle Geschwister mit größerer ID hinzugefügt werden. Dies wird wiederholt, bis der folgende Geschwisterknoten ein Vorfahre des Schlussknotens ist. Dann wird in ähnlicher Weise im Baum abgestiegen, bis der Schlussknoten schließlich erreicht ist. Das Resultat für das betrachtete Beispiel ist in Abbildung 2.6(a) zu sehen.

Nun werden auf jedem Prozess die jeweils größten Elemente, d.h. die Quadranten mit kleinstem Level, gesondert abgespeichert. Diese werden nun *Blöcke* genannt. Alle Blöcke zusammen decken das Gesamtgebiet noch nicht vollständig ab, da im Bereich der Start- und Schlussknoten jedes Prozesses potentiell kleinere Elemente erzeugt wurden. Deshalb erzeugt nun jeder Prozess, wenn nötig, einen Block der gleichen Größe, um den Beginn seines Bereiches nahtlos an den Bereich des vorhergehenden Prozesses anzuschließen. Der letzte Prozess erzeugt eventuell auch noch einen Block am Ende seines Bereiches, sodass in der Vereinigung nun das komplette Gebiet durch Blöcke zerlegt ist. Diese Blöcke bilden eine grobe Approximation des zu erzeugenden feineren Quadtrees. In Abbildung 2.6(b) sind die extrahierten Blöcke für das Beispiel zu sehen. Anzumerken ist, dass Blöcke auf unterschiedlichen Prozessen durchaus unterschiedliche Größen haben können. Die Größe der Blöcke eines Prozesses sind nach Konstruktion jedoch untereinander gleich.

Die Anzahl an Eingabeelementen, die ein Block überdeckt, die also Nachfahren des Blocks sind, werden im Folgenden für jeden Block gezählt. Danach werden die Blöcke mitsamt ihren Eingabeelementen so neu auf die Prozesse verteilt, dass die summierte Anzahl pro Prozess möglichst nahe am Durchschnitt liegt. Ab diesem Schritt liegt also eine Lastbalancierung vor, die auf der Heuristik beruht, dass der grobe Approximations-Quadtree aus den Blöcken bestmöglich verteilt ist.

In dem betrachteten Beispiel ist die Anzahl Eingabeelemente pro Block in Abbildung 2.6(b) eingetragen. Da die Punkte zu Beginn schon gleichmäßig verteilt waren, findet hier keine Umverteilung der Blöcke mehr statt. Im Allgemeinen ist denkbar, dass die Punkte zu Beginn durch verschiedene Prozesse erzeugt werden und sich die lokale Dichte unterscheidet. Dann hat der Lastbalancierungschritt Auswirkungen.

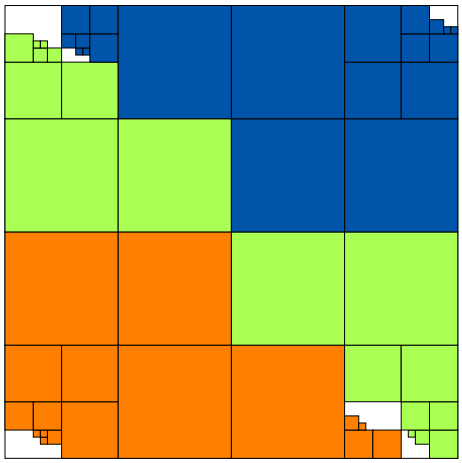
Der nächste Schritt besteht darin, dass nun jedes Element, angefangen mit den Blöcken, solange durch seine 4 Kinder ersetzt wird, wie es noch mehr als ein Eingabeelement als Nachfahre hat. Anders ausgedrückt wird der Quadtree so weit verfeinert, bis er die Eingabeelemente als Blätter enthält. Das Resultat, zusammen mit den ursprünglichen Punkten, ist in Abbildung 2.6(c) abgebildet. Man erkennt, dass in Bereichen mit größerer Punktdichte auch mehr Elemente entstanden sind.

2.2.4. Herstellung der 2:1–Balance

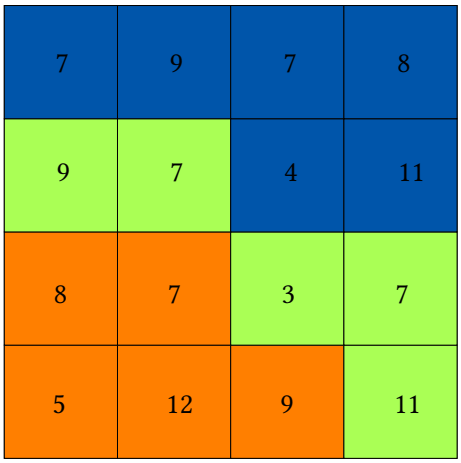
Bei der Finite–Elemente–Methode werden Berechnungen an den Eckpunkten der Elemente, genannt *Knoten*, durchgeführt. Um einen Quadtree hierfür verwenden zu können, darf es nur eine beschränkte Anzahl Knotentypen geben. Der Typ eines Knotens hängt von dem Verhältnis der Level der maximal 4 angrenzenden Elemente ab. Im bisher betrachteten Quadtree sind die Level, die an einem Eckpunkt aneinanderstoßen können, bis auf die Beschränkung durch l_{max} beliebig. Es wird nun gefordert, dass sich diese Level an jedem Knoten nur um jeweils eins unterscheiden dürfen. Es dürfen sich also keine zwei Elemente, von denen eines eine mehr als doppelt so große Kantenlänge wie das andere hat, eine Kante oder Ecke teilen. Diese Eigenschaft wird *2:1–Balance* genannt. Bei Verletzung der Eigenschaft liegt eine *Dysbalance* vor.

Im zweidimensionalen Fall lässt sich neben der hier verwendeten Einschränkung, dass Elemente, deren Level sich um maximal eins unterscheiden, keine gemeinsame Ecke haben, alternativ auch die schwächere Forderung keiner gemeinsamen Kante aufstellen. Im dreidimensionalen Fall ist zudem auch noch die Beschränkung auf „keine gemeinsame Fläche“ möglich. Alle diese möglichen Forderungen haben gemeinsam,

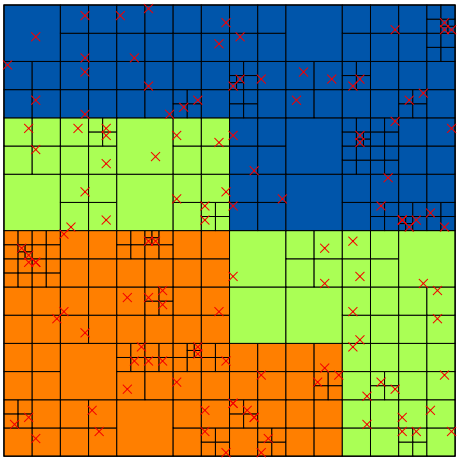
Abbildung 2.6. Erzeugung von Blöcken und anschließende Verfeinerung



(a) Vervollständigte Bereiche



(b) Extrahierte Blöcke mit Anzahl überdeckter Eingabeelemente. Prozess 0 (orange) überdeckt insgesamt 41 Eingabeelemente, Prozess 1 (grün) 37, Prozess 2 (blau) 46.



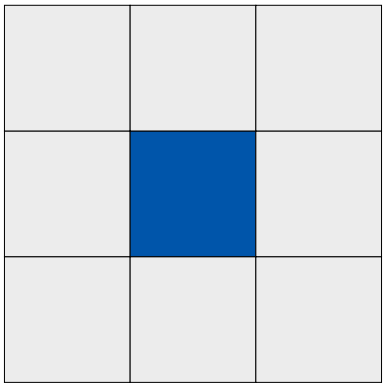
(c) Verfeinerte Blöcke, sodass pro Element maximal ein Punkt liegt

dass sie die Anzahl an Möglichkeiten, wie sich die Umgebung eines Knotens gestaltet, d.h. die Anzahl der *Knotentypen*, einschränken.

Der in [SSA+07] betrachtete Algorithmus muss als nächstes diese Balance herstellen. Dabei werden entsprechende Elemente, die die Balanceeigenschaft verletzen, verfeinert. Im Allgemeinen können durch die dabei hinzukommenden feineren Elemente weitere Dysbalancen entstehen, die ebenfalls behoben werden müssen. Der Balancierungsvorgang an einer Stelle kann sich somit auf nicht benachbarte Elemente auswirken. Dieser Effekt wird als *Ripple–Effekt* bezeichnet.

Ob ein Quadrant verfeinert wird, hängt jedoch nicht von beliebig weit entfernten Dysbalancen ab. Der Bereich, innerhalb dessen Dysbalancen zu einer Verfeinerung des Quadranten führen können, umschließt genau die 8 möglichen direkten Nachbarn des Quadranten auf gleichem Level, wie in Abbildung 2.7 dargestellt. Dieser Bereich wird als die *Isolationsschicht* des Quadranten bezeichnet. Aus diesem Grund kann die globale Balancierung zunächst parallel auf Teilgebieten durchgeführt werden. Danach ist nur noch die Anpassung der *Randquadranten* dieser Teilgebiete notwendig, d.h. es müssen die Quadranten balanciert werden, deren Isolationsschicht in das benachbarte Teilgebiet reicht.

Abbildung 2.7. Elemente der Isolationsschicht des blauen Elements



Zur Herstellung der 2:1–Balance geht der Algorithmus in 3 Stufen vor. In der ersten Stufe wird die Balanceeigenschaft in jedem Block sichergestellt. In der zweiten Stufe folgt dann die Balancierung entlang der Grenzen der lokalen Blöcke, sodass nach diesem Schritt der Teilquadtree auf jedem Prozess für sich die Balanceeigenschaft einhält. In der letzten Stufe wird dann die Balancierung zwischen den Grenzen der Teilgebiete der Prozesse durchgeführt, um globale Balance zu erreichen.

Die Balancierung innerhalb der Blöcke in der ersten Stufe verläuft auf jedem Prozess seriell, ohne notwendige Kommunikation. Angefangen mit den Elementen auf dem feinsten Level l_{max} wird über alle Level bis zum Level des Blocks iteriert. Für bestimmte Quadranten auf Level l werden die Nachbarn auf Level $l - 1$ hinzugenommen. So entsteht eine Menge überlappender Quadranten, die am Ende durch einen Linearisierungsprozess in einen gültigen Teilquadtree überführt wird. Um die Anzahl gleicher Quadranten, die dann entfernt werden müssen, zu reduzieren, wird das erwähnte Hinzufügen der Nachbarn nur einmal für Elemente durchgeführt, die nicht Geschwister eines bereits behandelten Elements sind. Nachdem dies für alle Level durchgeführt wurde, ist im Block die Balanceeigenschaft eingehalten.

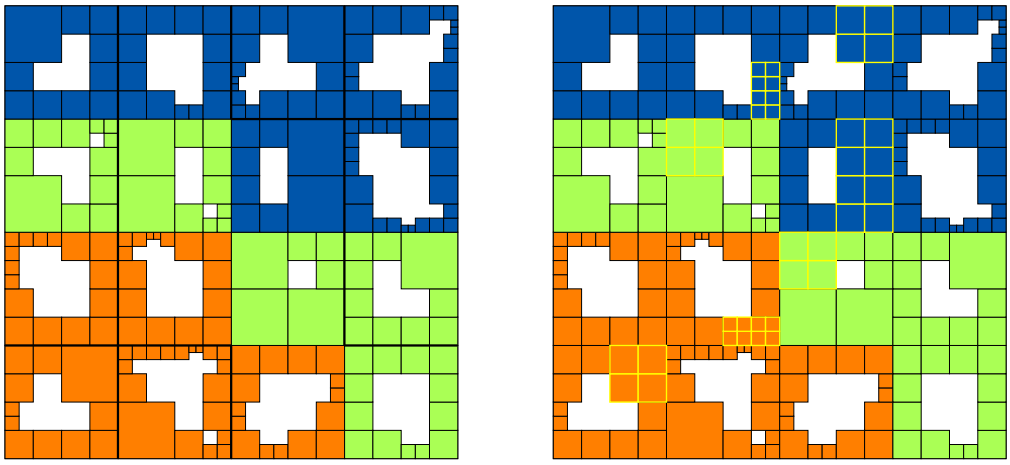
Für die zweite Stufe werden zunächst in jedem Block die Randquadranten ermittelt. Das sind genau diejenigen Elemente, die mindestens eine Kante mit dem Rand des Blocks teilen. Für jede Verletzung der Balanceeigenschaft auf diesen *Intraprozess*–Randquadranten wird das entsprechende Element verfeinert. In Abbildung 2.8 ist der Vorgang für das Beispiel dargestellt.

Die dritte Stufe verläuft analog zur zweiten Stufe mit dem Unterschied, dass die untersuchten Elemente nun auf die Prozesse verteilt sind. Jeder Prozess bestimmt also die Grenzquadranten zu anderen Prozessen (siehe Abbildung 2.9), die nun an die angrenzenden Prozesse versendet werden. An Stellen, wo die Gebiete dreier oder mehrerer Prozesse aneinander stoßen, werden Quadranten auch an mehrere Prozesse versendet, wie in [SSB08] näher beschrieben wird. Nun führt jeder Prozess die Balancierung an den Interprozessgrenzen durch. Die Balancierung an der Grenze der Teilgebiete zweier Prozesse wird also von beiden Prozessen unabhängig durchgeführt und liefert für jeden Prozess die korrekten Quadranten innerhalb des Prozessgebiets. Die fremden Grenzquadranten, die der Prozess zur Balancierung von anderen Prozessen bekommen hat, werden in diesem Prozess möglicherweise auch verfeinert. Sie entsprechen danach jedoch nicht zwangsläufig den korrekten Quadranten, die der jeweilige besitzende Prozess erzeugt hat.

Deshalb werden nach der Prozedur die Grenzquadranten erneut ausgetauscht, sodass in Folge jeder Prozess neben seinen eigenen Elementen nun auch die korrekten benachbarten Grenzquadranten besitzt. Diese werden als *Ghost*–Quadranten bezeichnet und sind später zur Erzeugung der Finite–Elemente–Datenstruktur nützlich.

In Abbildung 2.10 sind die Grenzquadranten abgebildet, wie sie auf jedem Prozess nach der Balancierung entstanden sind. Die grau dargestellten Elemente auf einem Prozess müssen nicht mit den farbigen Elementen mit gleichem Anker auf dem jeweiligen besitzenden Prozess übereinstimmen. In diesem Beispiel

Abbildung 2.8. Intraprozess–Grenzquadranten. Die Grenzen zwischen Blöcken auf einem Prozess sind mit dickerer Linie hervorgehoben.

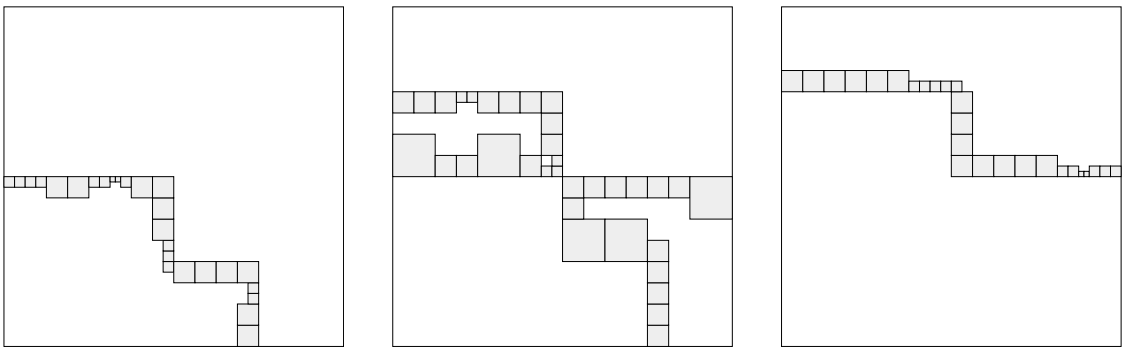


(a) Intraprozess–Randquadranten

(b) Balancierte Intraprozess–Randquadranten. Verfeinerte Elemente sind mit gelber Umrandung hervorgehoben.

ist das jedoch der Fall. In Abbildung 2.11 sind schließlich die Elemente und Ghostelemente abgebildet, die jeder Prozess nach Beendigung des Algorithmus' zum Erzeugen und Balancieren besitzt.

Abbildung 2.9. Interprozess–Grenzquadranten auf den jeweiligen Prozessen



(a) Prozess 0

(b) Prozess 1

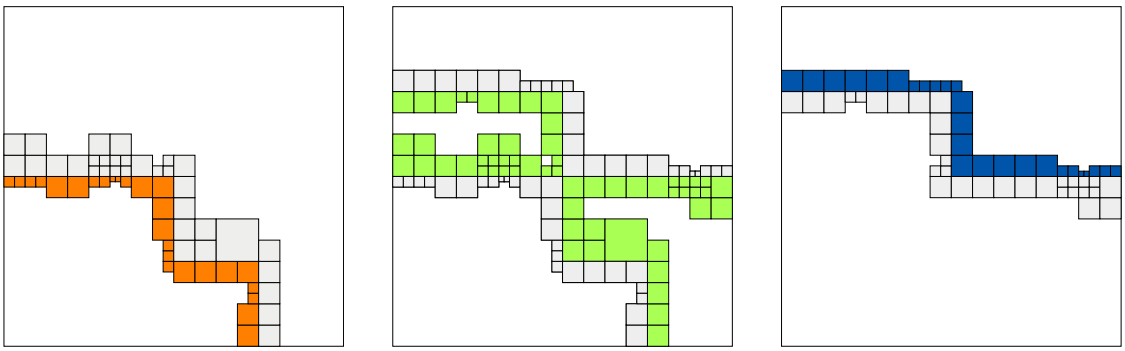
(c) Prozess 2

2.3. Das Finite–Elemente–Gitter

2.3.1. Anforderungen

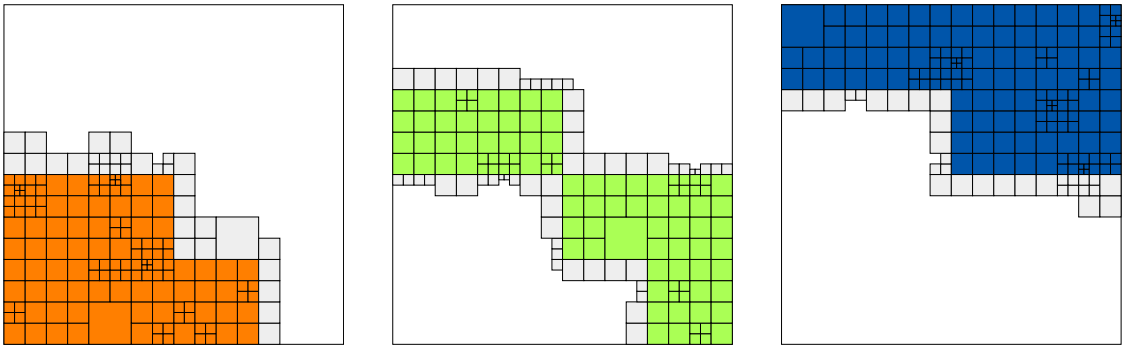
Für die Berechnung durch die Finite–Elemente–Methode ist eine Datenstruktur notwendig, die Werte mit den Elementen der Gebietszerlegung und Werte mit den Knoten assoziert. Bei Verwendung eines durch die sortierte Folge seiner Element–IDs repräsentierten Quadtrees ist die Assoziation von Werten mit den Elementen trivial, da die Werte einfach in derselben Reihenfolge wie die Elemente abgespeichert werden können. Für ein brauchbares Finite–Elemente–Gitter wird nun noch die Assoziation von Daten mit Knoten und die Verknüpfung der Elemente zu ihren jeweiligen adjazenten Knoten benötigt. Dafür wird das Vorgehen in [SSA+07] verwendet.

Abbildung 2.10. Balancierte Interprozess–Grenzquadranten. In Farbe sind die jeweiligen Elemente dargestellt, die der Prozess besitzt, in grau die von anderen Prozessen erhaltene Grenzquadranten, die zur Balancierung notwendig sind.



(a) Prozess 0 (b) Prozess 1 (c) Prozess 2

Abbildung 2.11. Kompletter balancierter Quadtree, aufgeteilt auf 3 Prozesse mit Ghost–Quadranten in grau.



(a) Prozess 0 (b) Prozess 1 (c) Prozess 2

Bei den Knoten wird zwischen normalen Knoten und *hängenden Knoten* unterschieden. Ein Knoten wird hängend genannt, wenn er auf einer Kante eines Elements, aber auf keiner Ecke des Elements liegt. Ein hängender Knoten tritt also an der Grenze von zwei feineren und einem gröberen Element auf, dort, wo die Kanten eine „T“–Kreuzung bilden. Der in Abbildung 2.12 abgebildete Quadtree enthält vier hängende Knoten, die durch ungefüllte Punkte dargestellt sind.

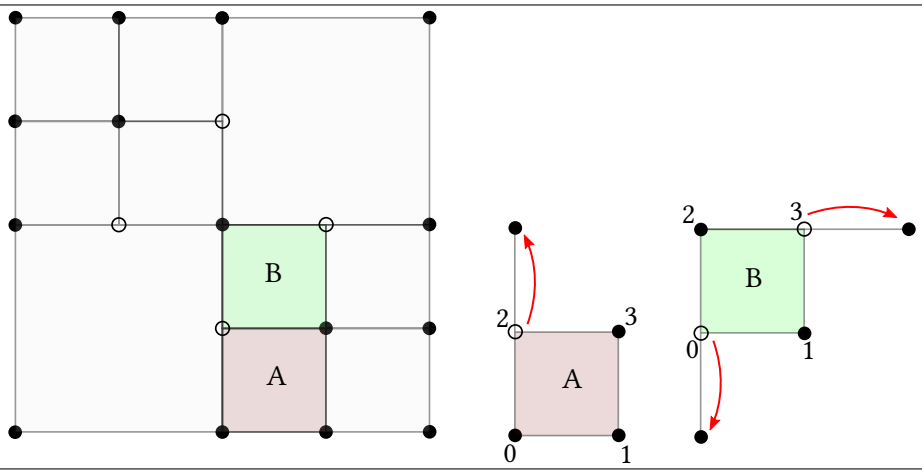
Bei der Diskretisierung enthält der Raum der zulässigen Geschwindigkeiten gerade die Geschwindigkeiten, die an nicht–hängenden Knoten bestimmte Werte annehmen und dazwischen bilinear interpoliert werden. An hängenden Knoten ist dann der Werte der Geschwindigkeit durch den Mittelwert der beiden Werte des größeren angrenzenden Elements festgelegt. Da Stetigkeit in der globalen Geschwindigkeitsfunktion gefordert wird, darf keine davon abweichende Festlegung der Werte an hängenden Knoten erfolgen. Dort befindet sich also kein Freiheitsgrad des Systems. Es ist jedoch denkbar, auch für die Werte an hängenden Knoten Variablen einzuführen, für die anschließend die Einhaltung von entsprechenden Zwangsbedingungen gefordert wird, sodass dort Stetigkeit erzwungen wird.

Wie in [Wan01] erläutert, lässt sich dies geschickt vermeiden. Es wird kein Wert mit hängenden Knoten assoziiert, stattdessen erhält jedes Element, das hängende Knoten an seinen Eckpunkten besitzt, einen Verweis auf den nächsten Knoten außerhalb des Elements, von dem der Wert an dem hängenden Knoten abhängt. Somit besitzt jedes entsprechende Element die Information über den Wert an hängenden Knoten,

da es zwei andere Knotenwerte kennt, als deren Mittelwert sich der Wert des hängenden Knotens berechnen lässt.

Der in Abbildung 2.12 dargestellte Quadtree enthält Elemente mit einem und zwei hängenden Knoten. Element A besitzt einen hängenden Knoten an Position 2. Mit dieser Position wird der darüberliegende Knoten assoziert, der auch zu Element B an Position 2 gehört. Element B enthält zwei hängende Knoten an den Positionen 0 und 3, die, wie dargestellt, mit weiteren Knoten verknüpft werden.

Abbildung 2.12. Finite–Elemente–Gitter mit hängenden Knoten. Gesondert dargestellt sind die beiden Element A und B, die einen bzw. zwei hängende Knoten besitzen. Die dort beginnenden Pfeile verweisen auf die nichthängenden Knoten, die stattdessen mit der entsprechenden Ecke verknüpft werden.



2.3.2. Erzeugen des Finite–Elemente–Gitters

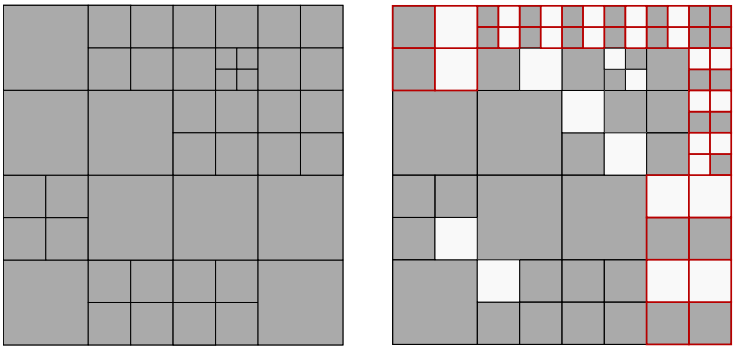
Die Datenstruktur der Knoten wird durch einen weiteren Quadtree, den *Quadtree der Knoten* gebildet, der aus dem *Quadtree der Elemente* entsteht. Interpretiert man die Element–IDs des Quadtrees der Elemente als Repräsentation des jeweiligen Ankerpunkts, lassen sich durch Verwendung einer Kopie des Quadtrees der Elemente als Quadtree der Knoten schon die meisten nicht–hängenden Knoten eindeutig darstellen. Lediglich die Knoten, die auf dem rechten und oberen Rand des Gebiets liegen sind nicht Ankerpunkt eines Elements und haben somit keine Repräsentation im so erhaltenen Quadtree der Knoten.

Deshalb werden die Elemente, die an den rechten oder oberen Rand des Gebiets stoßen, einmal verfeinert. Jedes dieser Randelemente wird also durch seine 4 Kindelemente ersetzt, von denen das unten links im Folgenden für den Knoten des ursprünglichen Elements steht. Der neu zu repräsentierende Knoten befindet sich an der unteren rechten oder oberen linken Ecke des zu ersetzenden Elements, je nach dem, ob sich das Element am rechten oder oberen Rand des Gebiets befindet. Dementsprechend wird der Knoten nun durch das hinzukommende Element unten rechts oder oben links dargestellt. Eine Ausnahme bildet das Element oben rechts, das die rechte obere Ecke des gesamten Gebiets bildet und somit 3 Knoten auf dem Rand besitzt. Dieses wird in 4 Elemente verfeinert, die allesamt jeweils einen Knoten repräsentieren. Die sonstigen Elemente am rechten oder oberen Rand werden durch 4 Elemente verfeinert, von denen nur 2 Elemente Knoten repräsentieren.

Der durch diese Verfeinerungsaktion erzeugte vorläufige Quadtree der Knoten enthält also Elemente am rechten und oberen Rand, die keine Knoten repräsentieren. Außerdem gibt es Quadranten, die hängende Knoten darstellen und ebenfalls nicht benötigt werden. Alle Quadranten, die keine nicht–hängenden Knoten repräsentieren, werden nun im nächsten Schritt entfernt und man erhält den endgültigen Quadtree der Knoten, der für jeden nichthängenden einen Eintrag enthält.

Ein beispielhafter Quadtree der Elemente ist in Abbildung 2.13(a) dargestellt. Nach Ausführung der beschriebenen Verfeinerung erhält man daraus den in Abbildung 2.13(b) dargestellten vorläufigen Quadtree der Knoten. Die weiß eingezeichneten Elemente werden entfernt, um den endgültigen Quadtree der Knoten zu erhalten.

Abbildung 2.13. Quadtree der Elemente und daraus erzeugter vorläufiger Quadtree der Knoten



(a) Quadtree der Elemente

(b) Vorläufiger Quadtree der Knoten. Rot umrandet sind Elemente, die durch den Verfeinerungsschritt entstanden sind. Weiß gefärbte Elemente repräsentieren keinen nicht–hängenden Knoten und werden entfernt.

Das Erzeugen des Quadtrees der Knoten soll parallel ausgeführt werden, da der Quadtree der Elemente bereits auf mehrere Prozesse verteilt ist. Da auf jedem Prozess zu den Elementen auch noch benachbarte Randquadranten vorliegen, kann der Großteil der Quadranten des Quadtrees der Knoten durch Kopie des lokalen Quadtrees der Elemente und anschließende Verfeinerung am rechten und oberen Gebietsrand erstellt werden. Es ist jedoch möglich, dass danach noch nicht alle Knoten dargestellt werden. Liegt ein hängender Knoten auf der Grenze der Gebiete der Prozesse, so kann es vorkommen, dass kein Element mit diesem Punkt als Ankerpunkt in den lokalen Ghost-Quadranten vorkommt. In solch einem Fall wird das benötigte Element durch Kommunikation übermittelt.

Ein Beispiel für einen solchen Fall ist durch Abbildung 2.14 angegeben. Der rot eingezeichnete Knoten auf Prozess 0 ist nicht Ankerpunkt eines dem Prozess bekannten Elements und muss von Prozess 1 zu Prozess 0 gesendet werden. Alle schwarz markierten nicht–hängenden Knoten können dagegen ohne Kommunikation im lokalen Quadtree der Knoten erzeugt werden.

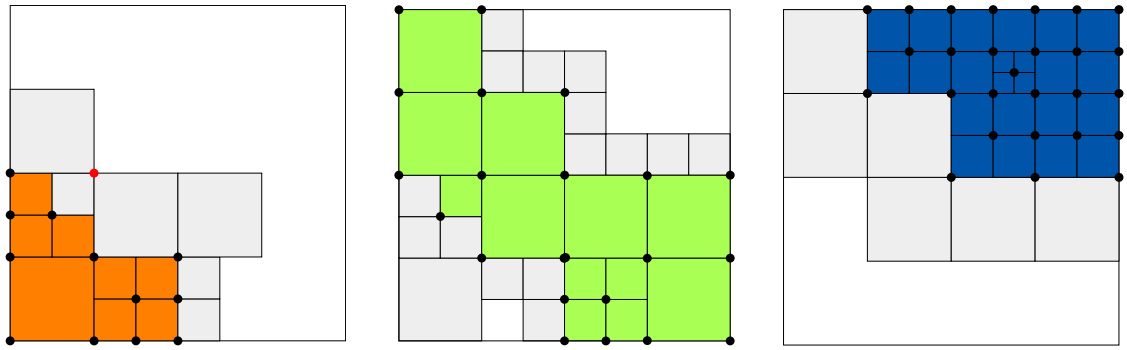
In Abbildung 2.15 ist der resultierende Quadtree der Knoten abgebildet, nachdem auch die ungültigen Elemente entfernt wurden. Das schraffierte Element repräsentiert den roten Knoten aus Abbildung 2.14(a) und wurde von Prozess 1 zu Prozess 0 gesendet.

2.3.3. Verknüpfung von Knoten und Elementen

Nachdem Elemente und Knoten durch eigene Quadtrees repräsentiert werden, ist als nächstes eine Verknüpfung von Knoten und Elementen notwendig. Dazu wird eine Tabelle erzeugt, die zu jedem Element die 4 assoziierten Knoten speichert. Eine direkte Ermittlung der ein bis sechs Elemente, die zu einem Knoten gehören, wird dadurch nicht gewährleistet. Sie müsste jeweils durch mehrere Suchen im Baum erfolgen. In der vorgelegten Implementierung der Finite–Elemente–Methode wird sie jedoch nicht benötigt.

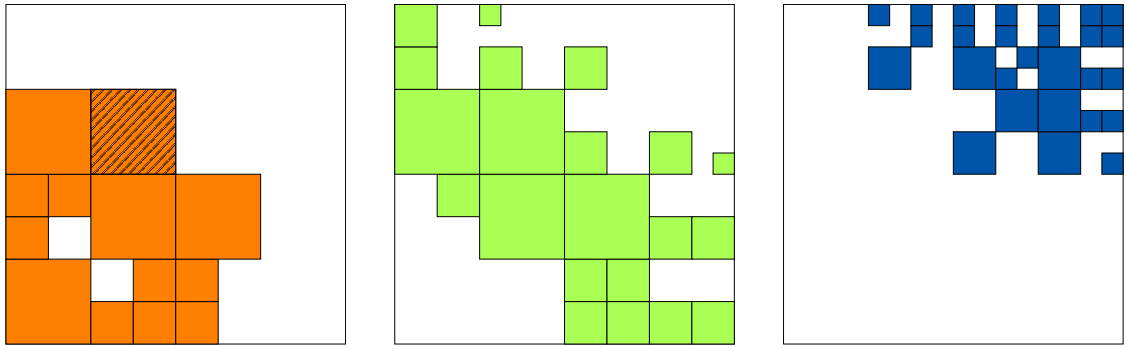
Das Erzeugen der Tabelle erfolgt durch eine Schleife über alle Elemente. Für die 4 Eckpunkte jedes Elements wird zunächst herausgefunden, ob es sich um einen hängenden Knoten handelt. Abhängig von der Nummer

Abbildung 2.14. Verteilter Quadtree der Elemente. Beispiel mit einem Knoten, deren Quadrant im Quadtree der Knoten nicht unter den Ghost–Quadranten vorliegt



(a) Prozess 0 (b) Prozess 1 (c) Prozess 2

Abbildung 2.15. Knoten–Quadtree nach Entfernen der ungültigen Quadranten



(a) Prozess 0 (b) Prozess 1 (c) Prozess 2

des Elements unter seinen Geschwistern kann dies für die Hälfte der Knoten direkt ausgeschlossen werden. Beispielsweise sind für das Kindelement Nr. 0 (siehe 2.2) die Knoten 0 und 3 sicher keine hängenden Knoten. Für die anderen Knoten wird das relevante Nachbarelement gesucht und anhand dessen Level festgestellt, ob es sich um einen hängenden Knoten handelt. Falls dies der Fall ist, wird ein Verweis zu einem dem hängenden Knoten benachbarten Knoten, wie in Kapitel 2.3.2 beschrieben, in die Tabelle gespeichert. Dabei muss beachtet werden, dass dieser Knoten eventuell auf einem anderen Prozess liegen kann. Solche Knoten werden während des Schleifendurchlaufs gesammelt und anschließend durch Kommunikation mit den entsprechenden Prozessen ausgetauscht.

Die Verweise zu den benötigten Knoten bestehen aus den Indices der Knoten in der Morton–ID–Liste des linearen Knoten–Quadtrees. Diese erhält man durch eine dortige Suche, wobei die Position des Such–Ankers aus der Position des Knotens hervorgeht. Es ist dabei lediglich eine Anpassung des Such–Ankers am rechten und oberen Rand des Gebiets notwendig.

Die Anzahl der Suchen innerhalb der Schleife lässt sich signifikant reduzieren, indem einmal für die benachbarten Elemente gefundene Knoten, sofern bereits vorhanden, für das aktuelle Element wiederverwendet werden.

2.4. Modellierung und Berechnung

In diesem Kapitel werden die verwendeten Gleichungen beschrieben und anschließend die Finite–Elemente–Methode für diese Gleichungen hergeleitet. Die Beschreibung basiert auf Standardliteratur wie beispielsweise [Zie79] für die Behandlung von Strömungsproblemen oder [Bra13] für die Grundlagen der Finite–Elemente–Methode.

2.4.1. Die Navier–Stokes–Gleichungen

Fluide, also Flüssigkeiten und Gase unterscheiden sich u.a. bezüglich ihres Widerstands gegen Formänderung. Zur Erzeugung einer *Couette*–Strömung befindet sich das Fluid im Spalt zwischen zwei parallelen Platten, die relativ zu einander bewegt werden, so, dass ihr Abstand konstant bleibt. In der Folge stellt sich im Fluid eine senkrecht zu den Platten linear ansteigende Geschwindigkeit ein. Deren Steigung wird als Deformationsgeschwindigkeit bezeichnet. Bei diesem Experiment tritt im Fluid eine räumlich konstante Schubspannung auf, deren Größe von der Deformationsgeschwindigkeit abhängt. Ist die Schubspannung proportional zur Deformationsgeschwindigkeit, ordnet man das Fluid in die Klasse der *newtonschen* Fluide ein. Beispiele hierfür sind Wasser, Öl oder Luft [Zie79].

Beschrieben werden soll nun die Strömung eines solchen newtonschen Fluids. Es wird als inkompressibel angenommen, was für Flüssigkeiten allgemein eine gute Näherung ist. Gesucht für jeden Zeitpunkt $t \in [0, t_{end}]$ sind Beziehungen für die Geschwindigkeit $\mathbf{u}(\mathbf{x}, t) = (u(\mathbf{x}, t), v(\mathbf{x}, t))^\top$ in die beiden Koordinatenrichtungen x_1 und x_2 sowie für den skalaren Druck $p(\mathbf{x}, t)$ im Rechengebiet $\Omega \subset \mathbb{R}^2$. Vektoren, wie z. B. die Position $\mathbf{x} \in \Omega$ werden als fett gesetzte Symbole dargestellt. Durch die Anwendung des Newtonschen Grundgesetzes auf ein Massenelement des Fluids lassen sich die Navier–Stokes–Gleichungen herleiten, die diese Beschreibung liefern. Sie setzen sich aus den Impulsgleichungen:

$$\begin{aligned} u_t &= \frac{1}{Re}\left(u_{x_1x_1} + u_{x_2x_2}\right) - (v\,u)_{x_2} - (u^2)_{x_1} - p_{x_1} + g_1, \\ v_t &= \frac{1}{Re}\left(v_{x_1x_1} + v_{x_2x_2}\right) - (u\,v)_{x_1} - (v^2)_{x_2} - p_{x_2} + g_2 \end{aligned} \tag{2.1}$$

und der Kontinuitätsgleichung:

$$u_{x_1} + v_{x_2} = 0 \tag{2.2}$$

zusammen. Die Indizierung der Variablen steht hier für die Ableitung nach der jeweiligen Variable. Es bezeichnet hierbei $\mathbf{g} = (g_1, g_2)$ eine über das Gebiet verteilt angreifende, externe, massenbezogene Kraft, wie z. B. die Gravitation. Die Reynoldszahl Re ist eine charakteristische Größe der Strömung, die das Turbulenzverhalten beschreibt. Sie ist proportional zur Dichte des Fluids sowie zu den geometrischen Abmessungen und der Geschwindigkeit des betrachteten Strömungsproblems. Zudem ist sie antiproportional zur Viskosität des Fluids.

Mithilfe der Kontinuitätsgleichung (2.2) lassen sich die Gleichungen (2.1) umformen. Es gilt:

$$(v\,u)_{x_2} + (u^2)_{x_1} = v_{x_2}\,u + v\,u_{x_2} + 2\,u\,u_{x_1} = \underbrace{(v_{x_2} + u_{x_1})}_{=0}\,u + u\,u_{x_1} + v\,u_{x_2} = u\,u_{x_1} + v\,u_{x_2}$$

und analog:

$$(u\,v)_{x_1} + (v^2)_{x_2} = v\,v_{x_2} + u\,v_{x_1}.$$

Damit lassen sich die Navier–Stokes–Gleichungen in vektorieller Form darstellen:

$$\mathbf{u}_t = \frac{1}{Re}\,\Delta\mathbf{u} - (\mathbf{u}\cdot\nabla)\,\mathbf{u} - \nabla p + \mathbf{g} \qquad \textit{Impulserhaltung} \tag{2.3a}$$

$$\nabla\cdot\mathbf{u} = 0 \qquad \textit{Kontinuität} \tag{2.3b}$$

Dabei ist der Laplaceoperator Δ komponentenweise zu verstehen.

Die Impulsgleichungen lassen sich wie folgt interpretieren: Die zeitliche Änderung der Geschwindigkeit in Gleichung (2.3a) setzt sich zusammen aus den Termen auf der rechte Seite: dem mit der Reynoldszahl gewichteten Anteil aus Diffusion $1/Re\,\Delta\mathbf{u}$, dem Konvektionsterm $(\mathbf{u}\cdot\nabla)\,\mathbf{u}$, der die Änderung durch Herantransport von Fluid anderer Geschwindigkeit beschreibt, sowie dem Einfluss aus einem Druckgradienten ∇p und der externen Kraft $\mathbf{g}$.

2.4.2. Die Finite Elemente Methode

Die Navier–Stokes–Gleichungen (2.3) sollen nun diskretisiert und mit der Finiten–Elemente–Methode gelöst werden.

Kern der Finiten–Elemente–Methode ist, dass das Problem in eine schwache Form überführt wird, in der die Gleichungen mit einer Testfunktion multipliziert wurden und nur für alle zulässigen Testfunktionen in einem integralen Sinn gelten soll.

In diesem Kapitel werden zunächst geeignete Testräume eingeführt und anschließend die Impulsgleichungen (2.1) und die Kontinuitätsgleichung (2.2) in die schwache Form überführt. Durch Definition von geeigneten Matrizen lassen sich diese schließlich in einer kompakten Form darstellen.

Ansatzräume

Als Raum der Testfunktionen $U = \{\phi \in H^1(\Omega) \mid \phi(\partial\Omega) = 0\}^2$ werden die zweidimensionalen Funktionen zugelassen, deren Komponenten auf dem Gebiet schwach differenzierbar sind und auf dem Rand des Gebiets den Wert 0 annehmen. Für nähere Erläuterungen wird auf [Bra13] verwiesen.

Ausgehend von den Gleichungen (2.3) erhält man nach Multiplikation mit einer Testfunktion $\boldsymbol{\varphi} \in U$ und Integration über das Gebiet Ω die folgende schwache Form:

$$\begin{aligned}\int_\Omega \mathbf{u}_t\cdot\boldsymbol{\varphi}\,\mathrm{d}\Omega \;&= \frac{1}{Re}\int_\Omega \Delta\mathbf{u}\cdot\boldsymbol{\varphi}\,\mathrm{d}\Omega - \int_\Omega (\mathbf{u}\cdot\nabla)\mathbf{u}\cdot\boldsymbol{\varphi}\,\mathrm{d}\Omega \\ &\quad - \int_\Omega \nabla p\cdot\boldsymbol{\varphi}\,\mathrm{d}\Omega + \int_\Omega \mathbf{g}\cdot\boldsymbol{\varphi}\,\mathrm{d}\Omega \qquad\qquad \forall\boldsymbol{\varphi}\in U.\end{aligned} \tag{2.4}$$

Da die Testfunktion $\boldsymbol{\varphi}$ auf dem Rand des Gebiets verschwindet, liefert die Anwendung der Greenschen Formel auf die Komponenten des Diffusionsterms:

$$\begin{aligned}\int_\Omega \mathbf{u}_t\cdot\boldsymbol{\varphi}\,\mathrm{d}\Omega \;&= -\frac{1}{Re}\int_\Omega \nabla\mathbf{u} : \nabla\boldsymbol{\varphi}\,\mathrm{d}\Omega - \int_\Omega (\mathbf{u}\cdot\nabla)\mathbf{u}\cdot\boldsymbol{\varphi}\,\mathrm{d}\Omega \\ &\quad - \int_\Omega \nabla p\cdot\boldsymbol{\varphi}\,\mathrm{d}\Omega + \int_\Omega \mathbf{g}\cdot\boldsymbol{\varphi}\,\mathrm{d}\Omega \qquad\qquad \forall\boldsymbol{\varphi}\in U.\end{aligned} \tag{2.5}$$

Im nächsten Schritt wird der Raum der Testfunktionen U durch einen diskretisierten Raum U^h approximiert. Außerdem werden endliche Ansatzräume V^h und S^h benötigt, in denen die Lösung der diskretisierten

schwachen Form für die Geschwindigkeit $\mathbf{u}^h(\mathbf{x})$ und den Druck $p^h(\mathbf{x})$ gefunden werden soll. Beim *Ritz–Galerkin*–Verfahren wird für U^h und V^h derselbe Raum verwendet.

Wie bereits beschrieben, wird die räumliche Diskretisierung für ein zweidimensionales Problem durch eine Zerlegung des Gebiets in M quadratische Elemente mit potentiell unterschiedlicher Seitenlänge erhalten. Die Anzahl der dabei enstehenden nicht–hängenden Knoten sei N. Pro so erhaltenes Element wird eine auf dem Element konstante Ansatzfunktion $\psi_i : \mathbb{R}^2 \to \mathbb{R}$ und pro Knoten A eine bilineare Ansatzfunktion $\phi_A : \mathbb{R}^2 \to \mathbb{R}$ definiert. Für ein reguläres Gitter lautet die Definition:

$$\psi_i(\mathbf{x}) = \begin{cases} 1 & \mathbf{x} \text{ ist in Element } i \\ 0 & \text{sonst,} \end{cases}$$

$$\phi_A(\mathbf{x}) = \max\left\{0, 1 - \frac{|x_1 - x_1^A|}{h}\right\} \cdot \max\left\{0, 1 - \frac{|x_2 - x_2^A|}{h}\right\}$$

mit h : Gitterweite des regulären Gitters,

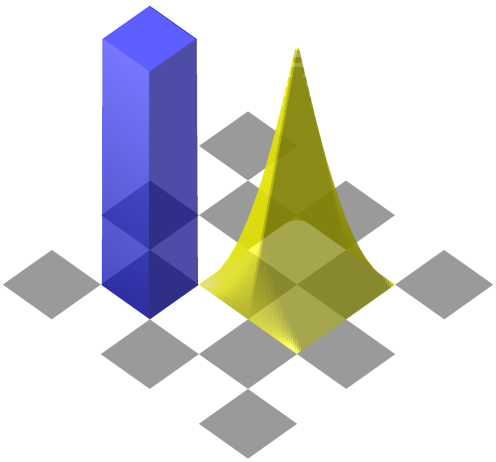
(x_1^A, x_2^A) : Position des Knotens A.

Die Funktionen sind in Abbildung 2.16 dargestellt. Für das zweidimensionale Geschwindigkeitsfeld werden dann die daraus abgeleiteten Funktionen $\boldsymbol{\varphi}_{iA} : \mathbb{R}^2 \to \mathbb{R}^2$ mit:

$$\boldsymbol{\varphi}_{iA} : \mathbf{x} \mapsto \phi_A(\mathbf{x})\, \mathbf{e}_i$$

für $i \in \{1,2\}$ verwendet. Hier bezeichnet $\mathbf{e}_i$ den kanonischen i–ten Einheitsvektor.

Abbildung 2.16. Ansatzfunktionen auf regulärem Gitter. Links (blau) $\psi_A(\mathbf{x})$, rechts (gelb) $\phi_A(\mathbf{x})$.



Die Gesamtheit dieser Ansatzfunktionen bildet die Ansatzräume $U^h = V^h$ und S^h, sodass gilt:

$$\boldsymbol{\varphi}_{iA} \in U^h, \quad \text{für } i \in \{1,2\}, A \in \{1, \ldots, N\},$$

$$\psi_i \in S^h, \quad \text{für } i \in \{1, \ldots, M\}.$$

Die diskretisierten Variablen für die Geschwindigkeit und den Druck werden als Linearkombination der Ansatzfunktionen dargestellt. Sie lauten mit der Dimensionsanzahl $d = 2$:

$$\mathbf{u}^h(\mathbf{x}, t) = \sum_{i=1}^{d} \sum_{A=1}^{N} u_{iA}(t)\, \boldsymbol{\varphi}_{iA}(\mathbf{x}), \qquad p^h(\mathbf{x}, t) = \sum_{i=1}^{M} p_i(t)\, \psi_i(\mathbf{x}). \tag{2.6}$$

Damit wird das Geschwindigkeitsfeld durch Angabe der Vektoren

$$\mathbf{u}_h = \left(u_{11}, u_{12}, \ldots, u_{1N}\right)^\top$$

$$\mathbf{v}_h = \left(u_{21}, u_{22}, \ldots, u_{2N}\right)^\top$$

und das Druckfeld durch Angabe des Vektors

$$\mathbf{p}_h = \left(p_1, \ldots, p_M\right)^\top$$

vollständig beschrieben.

Schwache Form der Impulsgleichungen

Die diskretisierte schwache Form der Impulsgleichungen (2.5) lautet mit den diskretisierten Variablen:

$$\begin{aligned}\int_\Omega \mathbf{u}_t^h \cdot \boldsymbol{\varphi}\,\mathrm{d}\Omega \quad &= -\frac{1}{Re}\int_\Omega \nabla\mathbf{u}^h : \nabla\boldsymbol{\varphi}\,\mathrm{d}\Omega - \int_\Omega (\mathbf{u}^h\cdot\nabla)\mathbf{u}^h\cdot\boldsymbol{\varphi}\,\mathrm{d}\Omega \\ &\quad - \int_\Omega \nabla p^h\cdot\boldsymbol{\varphi}\,\mathrm{d}\Omega + \int_\Omega \mathbf{g}\cdot\boldsymbol{\varphi}\,\mathrm{d}\Omega \qquad\qquad \forall\boldsymbol{\varphi}\in U^h.\end{aligned}$$

Da diese Gleichung linear in $\boldsymbol{\varphi}$ ist, bleibt die Aussage äquivalent, wenn gefordert wird, dass sie statt für beliebige Funktionen des Raums U^h nur für alle Elemente einer Basis von U^h gilt. Nach Konstruktion von U^h bildet $\{\boldsymbol{\varphi}_{iA} \mid i = 1{,}2,\ A = 1,\ldots,N\}$ eine solche Basis.

Das zu lösende Problem, das aus den Impulsgleichungen hergeleitet wurde, ist nun endlichdimensional und lautet:

„Finde $\mathbf{u}^h \in U^h, p^h \in S^h$, sodass

$$\begin{aligned}\int_\Omega \mathbf{u}_t^h \cdot \boldsymbol{\varphi}_{jB}\,\mathrm{d}\Omega \quad &= -\frac{1}{Re}\int_\Omega \nabla\mathbf{u}^h : \nabla\boldsymbol{\varphi}_{jB}\,\mathrm{d}\Omega - \int_\Omega (\mathbf{u}^h\cdot\nabla)\mathbf{u}^h\cdot\boldsymbol{\varphi}_{jB}\,\mathrm{d}\Omega \\ &\quad - \int_\Omega \nabla p^h\cdot\boldsymbol{\varphi}_{jB}\,\mathrm{d}\Omega + \int_\Omega \mathbf{g}\cdot\boldsymbol{\varphi}_{jB}\,\mathrm{d}\Omega\end{aligned} \tag{2.7}$$

für $j = 1{,}2,\ B = 1,\ldots,N$ gilt“.

Nach Einsetzen der Beschreibung von $\mathbf{u}^h$ und p^h mittels der Ansatzfunktionen (2.6) erhält man folgendes Problem:

„Finde u_{iA} für $A = 1,\ldots,N,\ i = 1{,}2$ und p_A für $A = 1,\ldots,M$,
sodass für $B = 1,\ldots,N,\ j = 1{,}2$ gilt:

$$\begin{aligned}\sum_{i=1}^{d}\sum_{A=1}^{N} u_{t;iA}\int_\Omega \phi_A\,\phi_B\,\mathrm{d}\Omega = \sum_{i=1}^{d}\Big(\quad &-\frac{1}{Re}\sum_{A=1}^{N} u_{iA}\int_\Omega \sum_{k=1}^{d}\phi_{A,x_k}\,\phi_{B,x_k}\,\mathrm{d}\Omega \\ &- \sum_{A=1}^{N} u_{iA}\sum_{G=1}^{N}\sum_{k=1}^{d} u_{kG}\int_\Omega \phi_G\,\phi_{A,x_k}\,\phi_B\,\mathrm{d}\Omega \\ &+ \sum_{A=1}^{M} p_A\int_\Omega \psi_A\,\phi_{B,x_i}\,\mathrm{d}\Omega \quad + \int_\Omega \mathbf{g}\cdot\boldsymbol{\varphi}_B\,\mathrm{d}\Omega\Big).“\end{aligned} \tag{2.8}$$

Hierbei wurden abgeleitete Größen durch ein Komma im Index notiert, beispielsweise bedeutet ϕ_{A,x_k} die Ableitung von ϕ_A nach dem k–ten Ortsvektor. Die Integrale über Ansatzfunktionen in (2.8) sind konstant und können berechnet werden. Damit handelt es sich um ein Gleichungssystem mit $2N$ Gleichungen für die $2N + M$ Unbekannten $\mathbf{u}_h, \mathbf{v}_h$ und $\mathbf{p}_h$. Um Lösbarkeit zu gewährleisten, muss als zusätzliche Bedingung die Kontinuitätsgleichung erfüllt werden.

Schwache Form der Kontinuitätsgleichung

Analog zur Bildung der schwachen Form für die Impulsgleichungen wird nun auch die Kontinuitätsgleichung (2.3b) in die schwache Form überführt. Nach Multiplikation mit einer Testfunktion $\psi_B \in S^h$ und Integration über das Gebiet Ω lautet die Gleichung:

$$\int_\Omega \nabla \cdot \mathbf{u}^h \, \psi_B \, \mathrm{d}\Omega = 0, \quad \text{für } B = 1, \ldots, M.$$

Nach Anwendung des Gaußschen Integralsatzes erhält man:

$$\int_\Omega \mathbf{u}^h \cdot \nabla \psi_B \, \mathrm{d}\Omega = 0, \quad \text{für } B = 1, \ldots, M. \tag{2.9}$$

Auch hier wird die Beschreibung von $\mathbf{u}^h$ durch die Ansatzfunktionen (2.6) vorgenommen. Es ergibt sich:

$$\sum_{A=1}^{N} \int_\Omega (u_{1A} \, \phi_A \, \psi_{B,x} + u_{2A} \, \phi_A \, \psi_{B,y}) \, \mathrm{d}\Omega = 0, \quad B = 1, \ldots, M. \tag{2.10}$$

Mit (2.8) und (2.10) zusammen erhält man nun ein lösbares Gleichungssystem. Zur Vereinfachung der Notation werden die konstanten Werte der auftretenden Integrale für verschiedene Ansatzfunktionen in Matrizen notiert. Es werden die Matrizen M, D, C, A_1, A_2 sowie die Vektoren $\mathbf{f}_1$ und $\mathbf{f}_2$ wie folgt eingeführt:

$$M = (m_{k,j})_{k,j=1}^{N}, \quad D = (d_{k,j})_{k,j=1}^{N}, \quad C = (c_{k,j})_{k,j=1}^{N}, \quad A_i = (a_{i,k,j})_{k,j=1}^{M,N}, \quad \mathbf{f}_i = (f_{i,k})_{k=1}^{N}$$

mit:

$$\begin{aligned}
m_{A,B} &= \int_\Omega \phi_A \, \phi_B \, \mathrm{d}\Omega, \\
d_{A,B} &= \frac{1}{Re} \int_\Omega \nabla \phi_A \cdot \nabla \phi_B \, \mathrm{d}\Omega = \frac{1}{Re} \int_\Omega \sum_{k=1}^{d} \phi_{A,x_k} \, \phi_{B,x_k} \, \mathrm{d}\Omega, \\
c_{A,B}(\mathbf{u}^h) &= \sum_{G=1}^{N} \int_\Omega \nabla \phi_A \cdot \mathbf{u}_G \, \phi_G \, \phi_B \, \mathrm{d}\Omega = \sum_{G=1}^{N} \sum_{k=1}^{d} u_{kG} \int_\Omega \phi_G \, \phi_{A,x_k} \, \phi_B \, \mathrm{d}\Omega, \\
a_{i,A,B} &= \int_\Omega \psi_A \, \phi_{B,x_i} \, \mathrm{d}\Omega, \\
f_{i,B} &= \int_\Omega g_i \, \phi_B \, \mathrm{d}\Omega.
\end{aligned} \tag{2.11}$$

Dabei wird M als Steifigkeitsmatrix bezeichnet. Aus dem Diffusionsterm ist D entstanden und aus dem Konvektionsterm C. Die Matrizen $A_i, i = 1{,}2$ entsprechen zusammen als Operator der diskreten Divergenz und die transponierten Matrizen $A_i^\top$ entsprechen der diskreten Version des Gradienten. Der Vektor der rechten Seite $\mathbf{f}_i$ beschreibt den Einfluss der externen Kraft.

Die Gleichungen der schwachen Form (2.8) und (2.10) lassen sich nun in Form der folgenden Matrizengleichungen notieren (vgl. auch [DH03, S.84]):

$$M \, \dot{\mathbf{u}}_h + D \, \mathbf{u}_h + C(\mathbf{u}_h, \mathbf{v}_h) \, \mathbf{u}_h - A_1^\top \mathbf{p}_h = \mathbf{f}_1, \tag{2.12a}$$

$$M \, \dot{\mathbf{v}}_h + D \, \mathbf{v}_h + C(\mathbf{u}_h, \mathbf{v}_h) \, \mathbf{v}_h - A_2^\top \mathbf{p}_h = \mathbf{f}_2, \tag{2.12b}$$

$$A_1 \, \mathbf{u}_h + A_2 \, \mathbf{v}_h = 0. \tag{2.12c}$$

Dabei bezeichnen $\dot{\mathbf{u}}_h$ und $\dot{\mathbf{v}}_h$ die Koeffizientenvektoren der Diskretisierung der zeitlichen Ableitungen von $\mathbf{u}$ und $\mathbf{v}$. Wie diese Diskretisierung vorgenommen wird, wird im nächsten Abschnitt beschrieben.

2.4.3. Zeitliche Diskretisierung

Die hergeleiteten Gleichungen (2.12) können noch nicht gelöst werden, da $\dot{\mathbf{u}}_h$ und $\dot{\mathbf{v}}_h$ noch nicht eingeführt wurden. Ausgehend von Startwerten $\mathbf{u}^{(0)}, \mathbf{v}^{(0)}$ und $\mathbf{p}^{(0)}$ sollen die Geschwindigkeiten $\mathbf{u}^{(k)}, \mathbf{v}^{(k)}$ und Drücke $\mathbf{p}^{(k)}$ zu Zeitpunkten $k \cdot \delta t, k \in \mathbb{N}$ berechnet werden.

Die zeitliche Diskretisierung von $\dot{\mathbf{u}}_h$ und $\dot{\mathbf{v}}_h$ geschieht durch einen expliziten Euler-Schritt, sodass gilt:

$$\begin{aligned} M\left(\frac{\mathbf{u}^{(n+1)} - \mathbf{u}^{(n)}}{\delta t}\right) + D\,\mathbf{u}^{(n)} + C_1(\mathbf{u}^{(n)}, \mathbf{v}^{(n)})\mathbf{u}^{(n)} - A_1^\top \mathbf{p}^{(n+1)} &= \mathbf{f}_1, \\ M\left(\frac{\mathbf{v}^{(n+1)} - \mathbf{v}^{(n)}}{\delta t}\right) + D\,\mathbf{v}^{(n)} + C_2(\mathbf{u}^{(n)}, \mathbf{v}^{(n)})\mathbf{v}^{(n)} - A_2^\top \mathbf{p}^{(n+1)} &= \mathbf{f}_2. \end{aligned}$$

Somit können die jeweils nächsten Geschwindigkeiten aus den vorherigen berechnet werden:

$$\begin{aligned} \mathbf{u}^{(n+1)} &= \mathbf{u}^{(n)} + \delta t\, M^{-1} \underbrace{\left(\mathbf{f}_1 - D\,\mathbf{u}^{(n)} - C(\mathbf{u}^{(n)})\mathbf{u}^{(n)}\right)}_{=:\mathbf{a}^{(n)}} + \delta t\, M^{-1} A_1^\top \mathbf{p}^{(n+1)}, \\ \mathbf{v}^{(n+1)} &= \mathbf{v}^{(n)} + \delta t\, M^{-1} \underbrace{\left(\mathbf{f}_2 - D\,\mathbf{v}^{(n)} - C(\mathbf{v}^{(n)})\mathbf{v}^{(n)}\right)}_{=:\mathbf{b}^{(n)}} + \delta t\, M^{-1} A_2^\top \mathbf{p}^{(n+1)}. \end{aligned} \tag{2.13}$$

Dabei wurden zwei Hilfsterme $\mathbf{a}^{(n)}$ und $\mathbf{b}^{(n)}$ definiert. Außerdem werden nun vorläufige Geschwindigkeiten $\mathbf{u}_{\mathrm{v}}^{(n)}, \mathbf{v}_{\mathrm{v}}^{(n)}$ eingeführt:

$$\begin{aligned} \mathbf{u}_{\mathrm{v}}^{(n)} &:= \mathbf{u}^{(n)} + \delta t\, M^{-1}\mathbf{a}^{(n)} \\ \mathbf{v}_{\mathrm{v}}^{(n)} &:= \mathbf{v}^{(n)} + \delta t\, M^{-1}\mathbf{b}^{(n)} \end{aligned} \tag{2.14}$$

Die Berechnung der Geschwindigkeiten im nächsten Zeitschritt nach (2.13) lässt sich mit (2.14) somit wie folgt notieren:

$$\begin{aligned} \mathbf{u}^{(n+1)} &= \mathbf{u}_{\mathrm{v}}^{(n)} + \delta t\, M^{-1} A_1^\top \mathbf{p}_h^{(n+1)}, \\ \mathbf{v}^{(n+1)} &= \mathbf{v}_{\mathrm{v}}^{(n)} + \delta t\, M^{-1} A_2^\top \mathbf{p}_h^{(n+1)}. \end{aligned} \tag{2.15}$$

Die Werte für den Druck im neuen Zeitschritt $\mathbf{p}^{(n+1)}$ wurden bisher noch nicht festgelegt. Außerdem halten die vorläufigen Geschwindigkeiten $\mathbf{u}_{\mathrm{v}}^{(n)}, \mathbf{v}_{\mathrm{v}}^{(n)}$ die Kontinuitätsbedingung eventuell nicht ein. Da die Kontinuitätsgleichung jederzeit eingehalten werden soll, wird sie als Bedingung für $\mathbf{u}^{(n+1)}$,$\mathbf{v}^{(n+1)}$ aufgestellt, aus der entsprechende Werte für den Druck folgen.

Die Kontinuitätsgleichung (2.12c) lautet für den Zeitschritt $n+1$:

$$A_1\,\mathbf{u}^{(n+1)} + A_2\,\mathbf{v}^{(n+1)} = 0.$$

Durch Einsetzen von (2.15) erhält man:

$$A_1\,\mathbf{u}_{\mathrm{v}}^{(n)} + \delta t\, A_1\, M^{-1} A_1^\top \mathbf{p}^{(n+1)} + A_2\,\mathbf{v}_{\mathrm{v}}^{(n)} + \delta t\, A_2\, M^{-1} A_2^\top \mathbf{p}^{(n+1)} = 0$$

und somit nach Umformung:

$$\underbrace{(A_1\, M^{-1}\, A_1^\top + A_2\, M^{-1}\, A_2^\top)}_{=:L}\, \mathbf{p}^{(n+1)} = -\frac{1}{\delta t}\,(A_1\, \mathbf{u}_{\mathrm{v}}^{(n)} + A_2\, \mathbf{v}_{\mathrm{v}}^{(n)}). \tag{2.16}$$

Dabei wurde die Matrix L eingeführt, die einen diskreten Differentialoperator darstellt. Es handelt sich um ein diskretes Poissonproblem, das mit entsprechenden homogenen Dirichlet-Randbedingungen für jeden Zeitschritt gelöst werden muss.

Zusammenfassend ergibt sich aus (2.15), (2.16) und der Definition der vorläufigen Geschwindigkeiten (2.13) folgendes Verfahren:

Für $n = 1,2,\dots$ berechne:

$$\begin{cases} \mathbf{u}_{\mathrm{v}}^{(n)} = \mathbf{u}^{(n)} + \delta t\, M^{-1}(\mathbf{f}_1 - D\,\mathbf{u}^{(n)} - C(\mathbf{u}^{(n)}, \mathbf{v}^{(n)})\,\mathbf{u}^{(n)}) \\ \mathbf{v}_{\mathrm{v}}^{(n)} = \mathbf{v}^{(n)} + \delta t\, M^{-1}(\mathbf{f}_2 - D\,\mathbf{v}^{(n)} - C(\mathbf{u}^{(n)}, \mathbf{v}^{(n)})\,\mathbf{v}^{(n)}) \end{cases}$$

Löse $L\mathbf{p}_h^{(n+1)} = -\frac{1}{\delta t}\,(A_1\, \mathbf{u}_{\mathrm{v}}^{(n)} + A_2\, \mathbf{v}_{\mathrm{v}}^{(n)})$ nach $\mathbf{p}^{(n+1)}$ (mit homogenen Neumann–Randbedingungen)

$$\begin{cases} \mathbf{u}^{(n+1)} = \mathbf{u}_{\mathrm{v}}^{(n)} + \delta t\, M^{-1} A_1^\top \mathbf{p}^{(n+1)} \\ \mathbf{v}^{(n+1)} = \mathbf{v}_{\mathrm{v}}^{(n)} + \delta t\, M^{-1} A_2^\top \mathbf{p}^{(n+1)} \end{cases}$$

(2.17)

2.4.4. Alternative Berechnungsvorschrift

Statt dem in (2.17) dargestellten Algorithmus gibt es eine äquivalente Variante, bei der die rechte Seite des Poissonproblems anders berechnet wird. Die rechte Seite von (2.16) lässt sich nach Einsetzen der Definition der vorläufigen Geschwindigkeiten (2.14) wie folgt umformen:

$$\begin{aligned} & -\frac{1}{\delta t}\,(A_1\, \mathbf{u}_{\mathrm{v}}^{(n)} + A_2\, \mathbf{v}_{\mathrm{v}}^{(n)}) \\ = \; & -\frac{1}{\delta t}\left(A_1\,(\mathbf{u}^{(n)} + \delta t\, M^{-1}\mathbf{a}^{(n)}) + A_2\,(\mathbf{v}^{(n)} + \delta t\, M^{-1}\mathbf{b}^{(n)})\right) \\ = \; & \underbrace{-\frac{1}{\delta t}\,(A_1\, \mathbf{u}^{(n)} + A_2\, \mathbf{v}^{(n)})}_{=0} - A_1\, M^{-1}\mathbf{a}^{(n)} - A_2\, M^{-1}\mathbf{b}^{(n)} \end{aligned}$$

Wenn die Kontinuitätsbedingung im Zeitschritt n erfüllt ist, entfällt der erste Term, der der diskretisierten Kontinuitätsgleichung (2.12c) entspricht. Somit erhält man schließlich eine weitere Möglichkeit, die rechte Seite der Poissongleichung zu berechnen:

$$-\frac{1}{\delta t}\,(A_1\, \mathbf{u}_{\mathrm{vorl}}^{(n)} + A_2\, \mathbf{v}_{\mathrm{vorl}}^{(n)}) = -(A_1\, M^{-1}\mathbf{a}^{(n)} + A_2\, M^{-1}\mathbf{b}^{(n)}) \tag{2.18}$$

Dabei ist jedoch gefordert, dass die Kontinuitätsgleichung im vorigen Zeitschritt erfüllt ist. Dies ist im ersten Zeitschritt je nach Anfangswerten der Geschwindigkeit nicht unbedingt gegeben.

Im damit entstehenden alternativen Verfahren ist es geschickt, die Definition von **a** und **b** leicht so zu ändern, dass nun die Vormultiplikation mit M^{-1} enthalten ist. Dann lautet das Verfahren:

Für $n = 1,2,\dots$ berechne:

$$\begin{cases} \mathbf{a}^{(n)} = M^{-1}\left(\mathbf{f}_1 - D\,\mathbf{u}^{(n)} - C(\mathbf{u}^{(n)}, \mathbf{v}^{(n)})\mathbf{u}^{(n)}\right) \\ \mathbf{b}^{(n)} = M^{-1}\left(\mathbf{f}_2 - D\,\mathbf{v}^{(n)} - C(\mathbf{u}^{(n)}, \mathbf{v}^{(n)})\mathbf{v}^{(n)}\right) \end{cases}$$

Löse $L\mathbf{p}_h^{(n+1)} = -\left(A_1\,\mathbf{a}^{(n)} + A_2\,\mathbf{b}^{(n)}\right)$ nach $\mathbf{p}^{(n+1)}$ (mit homogenen Neumann–Randbedingungen)

$$\begin{cases} \mathbf{u}^{(n+1)} = \mathbf{u}^{(n)} + \delta t\left(\mathbf{a}^{(n)} + M^{-1}A_1^\top \mathbf{p}^{(n+1)}\right) \\ \mathbf{v}^{(n+1)} = \mathbf{v}^{(n)} + \delta t\left(\mathbf{b}^{(n)} + M^{-1}A_2^\top \mathbf{p}^{(n+1)}\right) \end{cases}$$

Im Vergleich zur ersten Variante erfolgt bei dieser Version die Berechnung der rechten Seite durch weniger Operationen. Insbesondere entfällt die Multiplikation und anschließende Division mit der Zeitschrittweite δt, die als kleiner Wert angenommen wird. Dies führt auf einen geringeren numerischen Fehler in der rechten Seite des Poissonproblems. Dafür ist das Verfahren nur korrekt, wenn die Kontinuitätsgleichung im vorherigen Schritt erfüllt ist. Bei häufiger Anwendung dieser Variante hintereinander kann sich so der Fehler im Residuum der Kontinuitätsgleichung mit der Zeit aufsummieren. Deshalb ist es sinnvoll, die erste Variante mindestens in bestimmten Abständen anzuwenden, um die Kontinuität wieder herzustellen.

Die Anzahl der Matrix–Vektor–Multiplikationen und Vektoradditionen insgesamt ist bei dieser Variante gleich wie beim ersten Verfahren. Lediglich die Anzahl der Skalar–Vektor–Multiplikation ist um eins höher. Somit liegt bei dieser Variante eine geringe Laufzeitverschlechterung vor.

Diese Variante kann zur Validierung herangezogen werden, um z. B. die Implementierung der Berechnung der rechten Seite in der ersten Variante zu überprüfen.

2.5. Berechnung der Operatoren

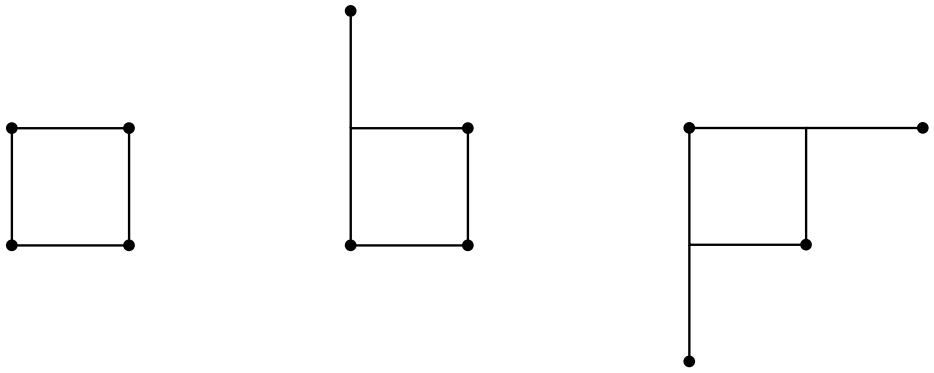
2.5.1. Knoten- und Elementtypen

Als nächstes müssen die in (2.11) definierten Matrizen berechnet werden. Dies geschieht nach deren Definition, indem die entsprechenden Integrale über Ansatzfunktionen und deren Ableitungen ausgewertet werden. Dabei muss beachtet werden, dass die Ansatzfunktionen je nach deren Träger unterschiedliche Formen annehmen. Um systematisch vorgehen zu können, werden nun verschiedene *Elementtypen* und *Knotentypen* eingeführt.

Wie bereits in Abbildung 2.12 gesehen, gibt es Elemente, unter deren Eckpunkten sich bis zu zwei hängende Knoten befinden. In solchen Fällen werden die hängenden Knoten durch die nächsten nicht–hängenden Knoten ersetzt. Durch die Anzahl hängender Knoten lässt sich jedes Element einem der drei Elementtypen A,B oder C zuordnen. Enthält das Element keine hängenden Knoten, ist es vom Typ A, bei einem hängenden Knoten gehört es zu Typ B und bei zweien zu Typ C. In Abbildung 2.17 sind Stellvertreter der drei Typen aufgeführt. Jedes Element lässt sich durch Rotation oder Spiegelung in eines der drei abgebildeten Elemente überführen.

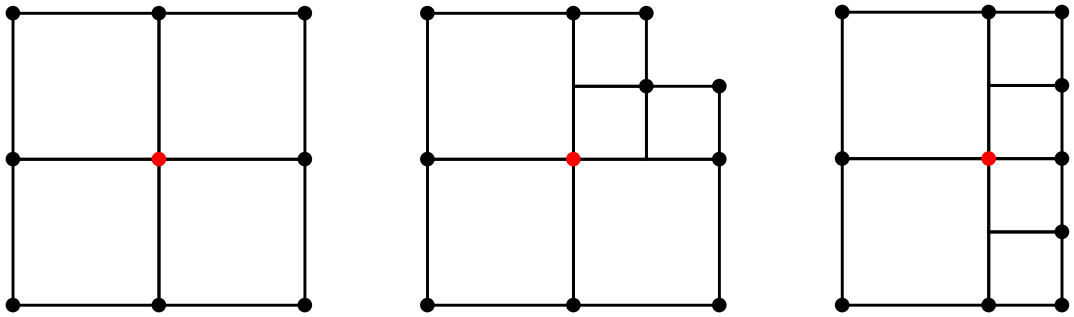
Der Typ eines Knotens wird durch Anordnung der Elemente um den Knoten herum bestimmt. Dabei treten Elemente mit maximal zwei verschiedenen Levels auf. Es werden die 5 Knotentypen A, B, C_1, C_2 und

Abbildung 2.17. Elementtypen

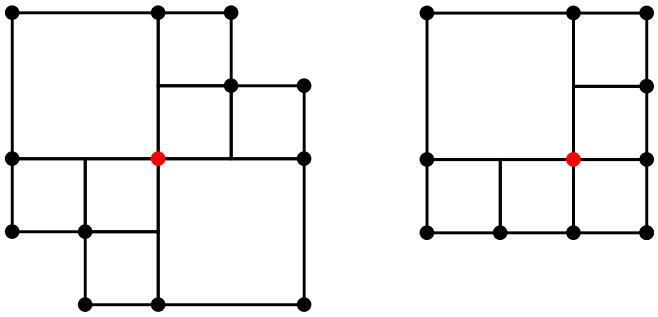


(a) Element Typ A (b) Element Typ B (c) Element Typ C

Abbildung 2.18. Knotentypen des jeweils rot markierten Knotens.



(a) Knoten Typ A (b) Knoten Typ B (c) Knoten Typ C_1



(d) Knoten Typ C_2 (e) Knoten Typ D

D nach Anzahl der „großen“ Elemente mit kleinerem Level eingeführt. Knoten, unter deren adjazenten Elementen ein, zwei, drei oder vier „große“ Elemente sind, gehören zu den Typen D, C, B oder A, wobei bei C eine weitere Unterscheidung in C_1 und C_2 ausgeführt wird.

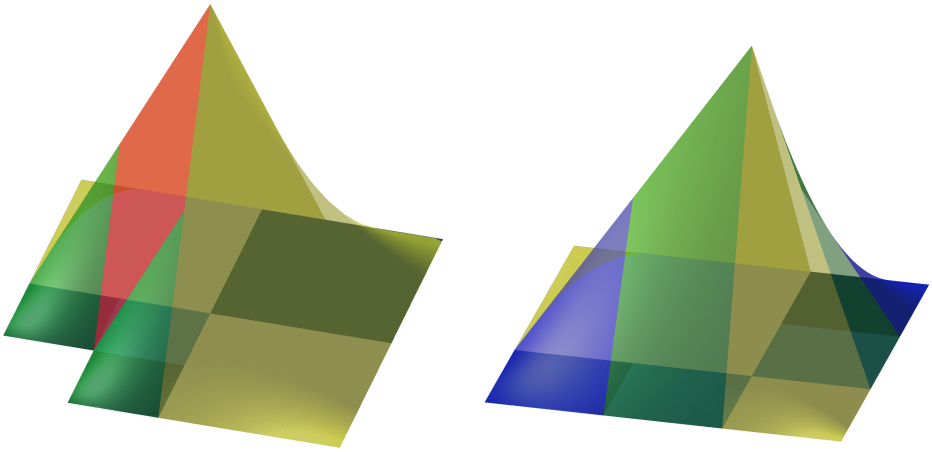
Die genaue Definition der Knotentypen geschieht durch die Stellvertreter in Abbildung 2.18. Wieder werden einem Knotentyp die Anordnungen zugeordnet, die durch Rotation und Spiegelung auf die dargestellte Anordnung abgebildet werden können.

Der Träger einer Ansatzfunktion $\phi_A(\mathbf{x})$ eines Knotens A erstreckt sich jeweils genau über die Elemente, die den Knotentyp definieren und die in Abbildung 2.18 dargestellt sind. Es sind genau die Elemente, denen der entsprechende Knoten zugeordnet ist. Auf jedem der zugehörigen Elemente nimmt die Ansatzfunktion jeweils an dem rot dargestellten Knoten den Wert 1 an und auf den anderen drei Knoten den Wert 0. Dazwischen verläuft sie bilinear.

In Abbildung 2.19(a) ist die Ansatzfunktion für einen Knoten vom Typ B dargestellt. Der gelbe Teil der Ansatzfunktion entspricht einer klassischen bilinearen Ansatzfunktion auf regulärem Gitter, wie z. B. in Abbildung 2.16. Die grünen und der rote Bereich entstehen dagegen, wie beschrieben, aus den zugrundeliegenden Elementen. Man erkennt, dass die Funktion an den Übergängen zwischen den farbigen Bereichen Kanten besitzt, sodass sie dort nicht differenzierbar ist. Die Stetigkeit ist jedoch überall gegeben.

Analog ist in Abbildung 2.19(b) eine Ansatzfunktion für einen Knoten vom Typ D abgebildet.

Abbildung 2.19. Ansatzfunktionen $\phi_A(\mathbf{x})$



(a) Ansatzfunktion für Knotentyp B (b) Ansatzfunktion für Knotentyp D

2.5.2. Berechnung von D und C

In den Matrizen $N \times N$–Matrizen M, D und C, so wie sie in (2.11) definiert wurden, gehört jeder Eintrag zu einem Paar zweier Ansatzfunktionen. Genauer, der Eintrag lässt sich als Integral über das Produkt von zwei Ansatzfunktionen bzw. deren Ableitungen berechnen. Das Integral ist nur jeweils dann ungleich Null, wenn sich die Träger der beiden Ansatzfunktionen überlappen. Bei einem regulären Gitter ergeben sich beispielsweise 9 Möglichkeiten, wie sich zwei Träger überlappen können. Drei davon sind in Abbildung 2.20 dargestellt. Die zugehörigen Matrizen haben in diesem Fall also höchstens 9 Nicht–Null–Einträge pro Zeile und Spalte. Somit handelt es sich um dünnbesetzte Matrizen.

Statt jede mögliche Überlappung zwischen zwei der fünf unterschiedlichen Ansatzfunktionen zu betrachten, ist es geschickter, die Beiträge zu den Integralen auf Elementebene zu berechnen. Um also einen Nicht–Null–Eintrag der Matrix M oder D zu erhalten, welcher in der Zeile steht, die dem Knoten i entspricht, werden die Beiträge aus den vier bis sechs am Knoten i angrenzenden Elemente zusammengezählt. Der Beitrag eines Elements kann im Voraus berechnet und in einer Koeffizientenmatrix abgespeichert werden. Er ist nur vom Elementtyp und dem Paar $(i, j) \in \{1, \ldots, 4\}^2$ zweier Knoten des Elements abhängig. Bei der Berechnung einer Matrix–Vektor–Multiplikation mit der Matrix M oder D werden die Nicht–Null–Einträge auf dieselbe Weise berechnet und, mit den entsprechenden Einträgen des zu multiplizierenden Vektor gewichtet, zusammengezählt.

Bei der Konvektionsmatrix C, die von $\mathbf{u}_h$ und $\mathbf{v}_h$ abhängig ist, ist für die Berechnung eines Eintrags die Summation über 4 Beiträge notwendig, die noch mit Geschwindigkeitswerten gewichtet werden müssen.

Zur Verdeutlichung des Vorgehens bei der Matrix–Vektor–Multiplikation ist in Algorithmus 2.1 ein Code–Abschnitt angegeben, der die Berechnung von `du`=$D\mathbf{u}$, `dv`=$D\mathbf{v}$ und `cu`=$C(\mathbf{u}, \mathbf{v})\mathbf{u}$, `cv`=$C(\mathbf{u}, \mathbf{v})\mathbf{v}$ demonstriert. In einer äußeren Schleife wird über alle Elemente iteriert. Pro Element laufen die nächsten beiden Schleifen

Code–Abschnitt 2.1 Matrix–Vektor–Multiplikation der Diffusions- und Konvektionsmatrizen

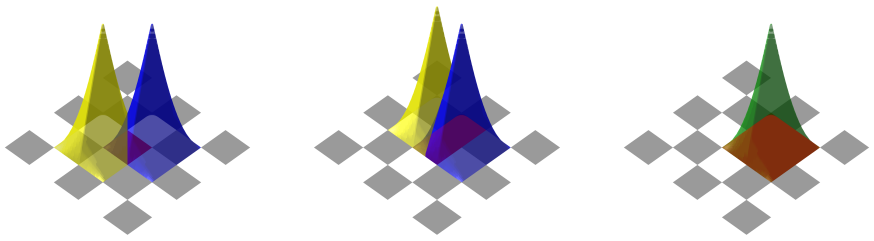
```
for el from 1 to N do            # Schleife über alle Elemente
    for i from 1 to 4 do         # Schleife über die 4 Knoten des Elements el
        for j from 1 to 4 do     # Schleife über die 4 Knoten des Elements el
            du(el,i) := du(el,i) + Dkoeff[el](p(el,i,j)) * u(el,j)
            dv(el,i) := dv(el,i) + Dkoeff[el](p(el,i,j)) * v(el,j)

            for k from 1 to 4 do      # Schleife über die 4 Knoten des Elements el
                cu(el,i) := cu(el,i) + vz(el) * C1koeff[el](p(el,i,j,k)) * u(el,j) * u(el,k)
                                     + vz(el) * C2koeff[el](p(el,i,j,k)) * u(el,j) * v(el,k)
                cv(el,i) := cv(el,i) + vz(el) * C1koeff[el](p(el,i,j,k)) * v(el,j) * u(el,k)
                                     + vz(el) * C2koeff[el](p(el,i,j,k)) * v(el,j) * v(el,k)
            end
        end
    end
end
```

über alle Paare von Knoten dieses Elements (i, j). Da jedem Element seine 4 Knoten zugeordnet sind, kann über die Elementnummer `el` und die lokale Knotennummer i auf den entsprechenden Wert der Variablen `x(el,i)` zugegriffen werden. Aus einer zum Elementtyp passenden 4×4–Matrix `Dkoeff[el]` werden die benötigten Koeffizienten ausgelesen. Dabei muss eine Abbildung `p(el,i,j)` zunächst den Index (i, j) so transformieren, dass das Element `el` die gleiche Ausrichtung wie das Referenzelement aus der Definition des Elementtyps bekommt, für den die Einträge in der Koeffizientenmatrix gelten.

Entsprechend wird für die Matrix C vorgegangen. Hier ist die Multiplikation mit zwei Knotenwerten erforderlich. Außerdem beinhaltet die Anpassung der Elementausrichtung neben der Abbildung `p(el,i,j,k` noch die Anpassung des Vorzeichens `vz(el)`, das aus einer eventuellen Spiegelung herrührt. Dies war bei der Diffusionsmatrix D nicht notwendig, da dort das Vorzeichen auch bei Spiegelung gleich bleibt.

Die Koeffizientenmatrizen `Dkoeff`, `C1koeff` und `C2koeff` sind für die Elementtypen im Anhang A aufgeführt.

Abbildung 2.20. Zwei Ansatzfunktionen $\phi_A(\mathbf{x})$ und $\phi_B(\mathbf{x})$ auf regulärem Gitter, deren Träger sich überlappen



(a) Überlappung über Eck (b) Halbe Überlappung (c) volle Überlappung für $A = B$

2.5.3. Berechnung von M^{-1}

Auch die Steifigkeitsmatrix M ließe sich wie D berechnen. Im beschriebenen Algorithmus kommt jedoch nur deren Inverse M^{-1} vor. Auch die Steifigkeitsmatrix ist dünnbesetzt. Bei Bildung der Inverse kann

diese Eigenschaft verloren gehen. Da es nicht praktikabel ist, die komplette Matrix abzuspeichern und zu invertieren, wird eine Näherung für M verwendet, deren Inverse einfach zu bestimmen ist. Man nähert die Matrix durch eine Diagonalmatrix $\tilde{M}$ an, wobei für jeden Diagonaleintrag die Summe aller Einträge der jeweiligen Zeile verwendet wird. Dieses Vorgehen wird als *lumping* bezeichnet. Um davon die Inverse zu bilden, müssen nur noch die Diagonaleinträge invertiert werden.

Die so erhaltene Matrix $\tilde{M}$ ist konstant und die Diagonaleinträge hängen nur vom entsprechenden Knotentyp ab. Nach Ausführung der entsprechenden Integration und Aufsummieren über die Nicht–Null–Einträge in jeder Zeile erhält man die in Tabelle 2.1 dargestellten Werte.

Knotentyp Knoten i	$\sum_j \tilde{M}_{ij}$
Typ A	h^2
Typ B	$\frac{15}{16}\,h^2$
Typ C_1	$\frac{3}{4}\,h^2$
Typ C_2	$\frac{7}{8}\,h^2$
Typ D	$\frac{9}{16}\,h^2$

Tabelle 2.1.: Einträge der genäherten Steifigkeitsmatrix $\tilde{M}$, abhängig vom Knotentyp. h bezeichnet die Elementseitenlänge für das jeweils größte Element in der Definition der Knotentypen.

2.5.4. Berechnung von A und $A^\top$

Bei M Elementen und N Knoten haben die Matrizen A_1 und A_2 die Dimension $M \times N$. Bei Anwendung auf einen Geschwindigkeitsvektor $\mathbf{u}$, dessen N Werte an den Knoten lokalisiert sind, erhält man M Werte auf den Elementen. Auch hier ist der Integrand in der Definition (2.11) nur dort ungleich Null, wo sich die Träger der Ansatzfunktionen ψ_i und ϕ_j überlappen. Da der Träger von ψ_i gerade dem Element i entspricht, berechnet die Matrix den Wert eines Elements gerade aus den Werten der angrenzenden Knoten.

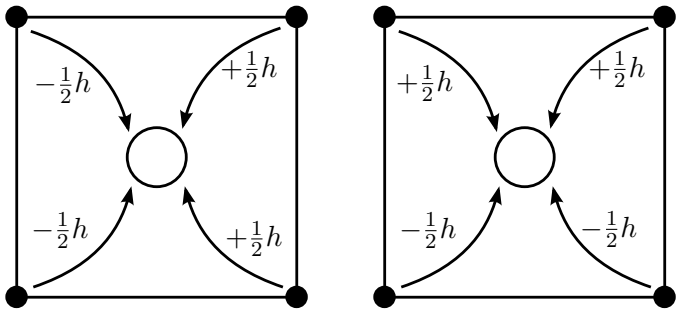
Für ein Element vom Typ A ist dieser Vorgang in Abbildung 2.21(a) und 2.21(b) verdeutlicht. Zur Berechnung von $\mathbf{p} = A\,\mathbf{u}$ werden die Werte von $\mathbf{u}$ an den Knoten wie angegeben mit $+1/2\,h$ und $-1/2\,h$ multipliziert und aufaddiert, um den Wert von $\mathbf{p}$ im Element zu erhalten. Für die weiteren Elementtypen sind die jeweiligen Faktoren im Anhang A in Abbildung A.4 aufgeführt.

Anschaulich entspricht die Anwendung von A_1 dem Bilden der Ableitung in x_1–Richtung und die Anwendung von A_2 dem Bilden der Ableitung in x_2–Richtung. Bis auf den Faktor h entsprechen die beiden oberen Faktoren sowie die beiden unteren Faktoren in Abbildung 2.21(a) jeweils einem Differenzenquotienten. Die beiden Werte werden schließlich durch den Faktor $1/2$ gemittelt, um den Wert auf den Elementmittelpunkt zu beziehen. Analog verhält es sich in 2.21(b) für den Operator A_2.

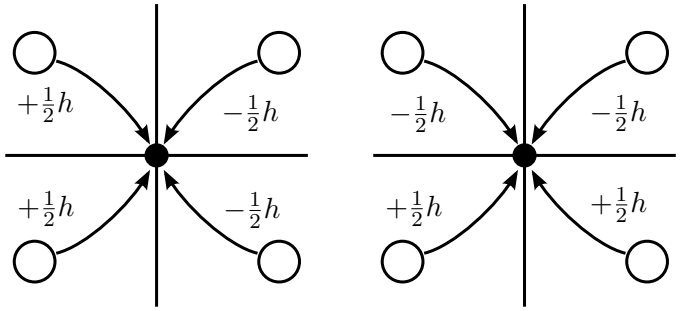
Die Anwendung von $A_1\,\mathbf{u} + A_2\,\mathbf{v}$ entspricht also, unter Vernachlässigung des Faktors h, der Bildung der Divergenz von $(u, v)^\top$.

Während die Matrix A aus Werten an den Knoten Werte am Elementmittelpunkt berechnet, ist bei der transponierten Matrix $A^\top$ das Umgekehrte der Fall. Da es sich um nun eine $N \times M$–Matrix handelt, werden

Abbildung 2.21. Anwendung der Operatoren A und $A^\top$ bei Elementen vom Typ A. h bezeichnet die Seitenlänge eines Elements.



(a) Anwendung von A_1 (b) Anwendung von A_2



(c) Anwendung von $A_1^\top$ (d) Anwendung von $A_2^\top$

Werte auf den Elementen in Werte an den Knoten überführt. Aus dem bereits bei A geltenden Argument der überlappenden Träger im Integranden der Definiton von A berechnet sich hier der Wert eines Knotens nur aus den Werten der angrenzenden Elemente. Für Knoten vom Typ A ist die Berechnungsvorschrift für $A_1^\top$ und $A_2^\top$ in den Abbildungen 2.21(c) und 2.21(d) dargestellt. Die Faktoren für alle Knotentypen befinden sie sich im Anhang A.

Auch hier liegt eine anschauliche Deutung nahe. Ähnlich wie bei A berechnen $A_1^\top$ und $A_2^\top$ die negativen Ableitungen in x_1- bzw. x_2–Richtung der durch die Werte an den Elementmittelpunkten gegebenen Funktion.

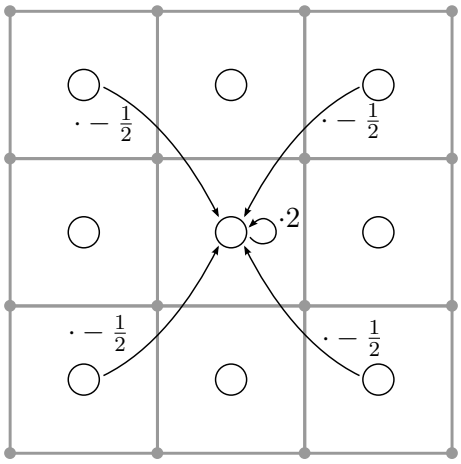
Nun lässt sich die Deutung auf den diskreten Operator $L = A_1 M A_1^\top + A_2 M A_2^\top$, wie er in (2.16) vorkommt, erweitern. Bei Verwendung der gelumpten Matrix $\tilde{M}$ für M entspricht die Anwendung von $\tilde{M}^{-1}$ auf einen Wert an einem Knoten vom Typ A der Division durch h^2. Somit beschreibt der Operator $\tilde{L} = A_1 \tilde{M} A_1^\top + A_2 \tilde{M} A_2^\top$ die Anwendung der Divergenz auf den negativen Gradienten, also den negativen Laplace–Operator. Die überschüssigen Faktoren h in A_i und $A_i^\top$ werden durch $\tilde{M}^{-1}$ wieder ausgeglichen.

Fasst man die Berechnungsvorschriften in Abbildung 2.21 zusammen, ergibt sich für $\tilde{L}$ das Schema in Abbildung 2.22. Um den Wert für das mittlere Element zu berechnen werden die Werte der diagonal angrenzenden Elemente mit $-1/2$ multipliziert und aufaddiert. Außerdem wird der eigene Wert mit 2 multipliziert und dazuaddiert. Die Werte der vier seitlich links, rechts, oben und unten angrenzenden Elemente leisten keinen Beitrag.

Bei der Anwendung von $\tilde{L}$ auf einen Vektor kann der Transport der Beiträge der benachbarten Elemente über die Knoten erfolgen. Ein Element muss damit nicht direkt auf die Werte seiner Nachbarn zugreifen. So lässt sich die Berechnung mit der in Kapitel 2.3 beschriebenen Datenstruktur ausführen. Auch für die

Anwendung der anderen Operatoren muss nur die Verknüpfung der Elemente zu seinen Knoten bekannt sein.

Abbildung 2.22. Anwendung des Operators $\tilde{L}$ auf Elemente vom Typ A.



3. Programmtechnische Umsetzung

3.1. Bedienung des Programms

In diesem Kapitel werden die Schritte beschrieben, die notwendig sind, um mit dem Programm eine Simulation durchzuführen. Dies beinhaltet im Pre–Processing die Spezifikation der Geometrie mit Positionierung von Randbedingungen in einer Grafikdatei sowie die Definition von Parametern in einer Textdatei. Anschließend erfolgt die Ausführung des Programms und danach die Auswertung der gewonnenen Daten im Post–Processing.

3.1.1. Spezifikation von Geometrie und Randbedingungen

Die Form des Berechnungsgebietes sowie verschiedene Randbedingungen werden dem Programm über eine Grafikdatei übergeben. Die Geometrie soll frei gewählt werden können. Das Simulationprogramm kann diese durch Quadtree–Elemente auf sehr feinem Level approximieren. Beispielsweise soll die Vorgabe eines Kreisbogens möglich sein, der dann in einstellbarer Genauigkeit durch entsprechende Elemente angenähert wird.

Bei Verwendung eines Pixel–basierten Dateiformats für die Grafikdatei muss die Pixelauflösung mindestens so fein wie die gewünschte Umsetzung durch das Simulationsprogramm sein. Für Elemente auf Level 20 bedeutet dies beispielsweise eine Breite von $2^{20} \approx 10^6$ Pixel, die die Rastergrafik haben muss, um Geometrie im Gebiet des kompletten Quadtrees zu spezifizieren. Für solch hohe Level ist dies nicht mehr praktikabel und auch für geringere Level steigt die Dateigröße schnell an. Deshalb verwendet das Programm nur Vektorgraphiken, bei denen die Ränder der Geometrie durch Pfade spezifiziert werden und somit eine nahezu beliebige Auflösung bei geringer Dateigröße möglich ist.

Das verwendete Dateiformat ist *svg*. Es ist XML–basiert, sodass die Dateien auch für Menschen lesbar sind. Empfohlen vom W3C, dem Gremium zur Standardisierung der Technik im WWW, ist es inzwischen so etabliert, dass jeder moderne Webbrowser svg–Grafiken darstellen kann. Zudem gibt es mehrere Editoren, mit deren Hilfe svg–Graphiken erstellt und bearbeitet werden können [W3C].

Linien werden bei diesem Format als Pfade bezeichnet. Ein Pfad kann sich aus geraden Linien, ellipsenförmigen Abschnitten und Bézier–Kurven vom Polynomgrad 2 und 3 zusammensetzen. Abgespeichert werden dafür nur bestimmte Werte, wie z. B. Koordinaten des Anfangs- und Endpunktes einer Linie, Mittelpunkt und Radien der Ellipse oder Kontrollpunkte der Bézier–Kurve. Neben diesen geometrischen Information können auch Darstellungsparameter wie Liniendicke und -farbe oder Füllung der umschlossenen Fläche angegeben werden.

Erlaubt in der Eingabedatei für das Programm sind beliebige zusammenhängende Gebiete. Insbesondere sind auch „Löcher“ möglich, z. B. um Hindernisse für das Fluid zu definieren. Auch das Format der Graphik kann beliebige Seitenverhältnisse annehmen. Das Gebiet wird durch komplette Umrandung durch Pfade definiert. Dabei können mehrere Pfade verwendet werden. Es ist besonders an den Übergangsstellen zwischen zwei Pfaden darauf zu achten, dass sich diese (in der Größenordnung der Diskretisierungsfeinheit) nicht überschneiden und auch keine Klaffung bilden.

Die Definition von Randbedingungen geschieht durch unterschiedliche Farbgebung des Pfads. Implementiert sind die Randbedingungen *noslip*, *inflow*, *outflow* und *slip* für die Geschwindigkeit sowie Dirichlet–Druckrandbedingungen.

Bei *noslip* und *inflow* handelt es sich um Dirichlet–Randbedingungen. An einem Rand mit *noslip*–Randbedingungen werden die Geschwindigkeiten auf Null gesetzt. Bei *inflow*–Rändern wird der zu setzende Geschwindigkeitsvektor so, wie er in der Parameterdatei definiert ist, verwendet. Die *outflow*–Randbedingung beschreibt dagegen eine homogene Neumann–Randbedingung. An Rändern mit *slip*–Bedingung werden die beiden Typen kombiniert. Die Geschwindigkeitskomponente in die Richtung orthogonal zum Rand wird auf Null gesetzt, für die Komponente tangential zum Rand gilt eine homogene Neumann–Randbedingung.

Um eine *noslip*–Randbedingung festzulegen, muss der Pfad, entlang derer sie gelten soll, einen *blauen* Farbton haben. Für *inflow* wird *grüne* Farbe verwendet, für *outflow rot* und für *slip magenta*.

Da in den beschreibenden Gleichungen jeweils nur Gradienten des Drucks vorkommen, ist der absolute Wert des Drucks nicht eindeutig festgelegt. Die Verwendung einer einzelnen Dirichlet–Druckrandbedingung ist somit nicht zielführend, um einen Druckverlauf vorzugeben. Deshalb können für den Druck zwei verschiedene Werte, die wieder in der Parameterdatei anzugeben sind, an zwei verschiedenen Stellen am Rand festgelegt werden. Diese beiden Stellen werden durch *gelben* und *orange*–farbenen Randpfad spezifiziert.

Bei der Interpretation des Farbtons verwendet das Programm eine gewisse Toleranz, sodass nicht ein genauer Farbton getroffen werden muss. Jede Farbe, die für das menschliche Auge der beschrieben Farbe entspricht, sollte zur jeweiligen Randbedingung führen. Sinnvoll ist auch eine Kontrolle der Ausgabedateien, ob die gewünschte Geometrie mit den gewünschten Randbedingungen erzeugt wurde.

Abbildung 3.1 zeigt zur Demonstration eine beispielhafte Eingabe–Vektorgraphik. Modelliert wird ein gebogenes Rohr, das im linken unteren Bereich Ablagerungen enthält, welche zu einer unregelmäßigen Kontur führen. Diese wurde mit dem Programm *Inkscape* durch „Freihandzeichnung“ erhalten, was quadratische Bézierkurven mit einigen Kontrollpunkten erzeugt. Der Rohrbogen unten rechts wird durch einen ellipsenförmigen Bogen gebildet. Die Verjüngung des aufsteigenden Rohrs rechts ist wiederum eine Bézierkurve, die dann in eine gerade Linie mündet. Die Rohrwand in blauer Farbe erzeugt dort *noslip*–Randbedingungen, an den beiden Rohröffnungen werden durch *gelb* und *orange* Druckwerte vorgegeben.

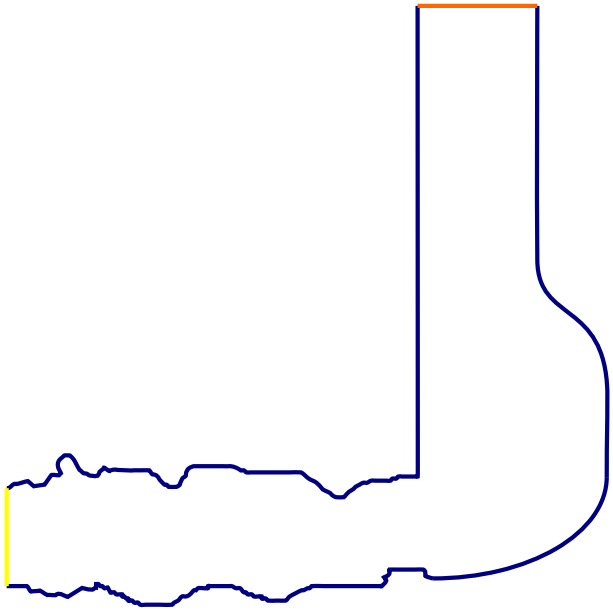
3.1.2. Eingabeparameter

Weitere Eingabe in das Programm bildet eine Datei, in der Parameter definiert sind. Dies ist eine Textdatei, die pro Zeile jeweils eine Angabe in der Form `<Bezeichner> = <Wert>` beinhaltet. Die gültigen Bezeichner sind vom Programm vorgegeben, die Verwendung von Klein- oder Großschreibung, sowie die Unterbringung von Unterstrichen im Bezeichner sind beliebig. Beispielsweise haben die folgenden beiden Zeilen, die für *inflow*–Randbedingungen die Geschwindigkeitskomponente in x_1–Richtung festlegen, den gleichen Effekt:

```
bc_velocity_dirichlet_u=1.0
BCVelocityDirichletU = 1
```

Der Wert ist je nach Parameter eine ganze Zahl, eine Kommazahl oder bei den Parametern, die Dateinamen bezeichnen, eine Zeichenkette. Bei einigen Parametern wird die Zahl `0` oder `1` erwartet, um die Option aus oder einzuschalten. Bei Kommazahlen ist das Dezimalzeichen der Punkt, durch Anhängen von `e` und einer ganzen Zahl wird die entsprechende Zehnerpotenz angenommen, wie z. B. bei `epsilon = 1e-15`. Zeichenketten werden ohne Anführungszeichen angegeben. Den Werten vorangestellte oder nachfolgende

Abbildung 3.1. Eingabegraphik `bent_pipe.svg` zur Spezifikation eines gebogenen Rohrs



Leerzeichen werden ignoriert. Um die Parameterdatei übersichtlich zu halten, können Leerzeilen und Kommentare durch Voranstellen von „#“ eingefügt werden.

Jeder der Parameter besitzt einen Standardwert, der angenommen wird, falls der Parameter nicht definiert wurde. Deshalb ist es möglich, nur die Parameter in der Eingabedatei anzugeben, die nicht den Standardwerten entsprechen. Mit Datei 1, S. 40 ist für das gebogene Rohr eine beispielhafte Eingabedatei gegeben, die die wichtigsten Parameter spezifiziert. Eine komplette Tabelle aller Parameter und ihrer Standardwerte befindet sich in Anhang B.

Der Parameter `scenario` bestimmt die Geometrie. Die Werte 1 und 2 erzeugen Beispiel–Geometrien. Für den Wert 3 wird ein quadratisches Gebiet durch Elemente auf einem festen Level diskretisiert, was einem regulären Gitter entspricht. Auch für den Wert 4 wird ein quadratisches Gebiet angenommen. Hier wird der Algorithmus zur Erzeugung des Quadtree mit zufälligen und im Gebiet gleichverteilten Punkten gestartet. Die Angabe `scenario = 5` sorgt hier dafür, dass die unter `input_file` angegebene Vektorgraphik geladen wird. Es wird nun ein Finite–Elemente–Gitter erzeut, das die Geometrie approximiert. Die auftretenden Level der Elemente werden dabei durch `d_min` und `d_max` beschränkt.

Die gesamte Länge der Geometrie entspricht dem durch `domain_width` angegebenen Parameter. Mit `re` und `t_end` werden Reynoldszahl und Dauer des Strömungsproblems definiert. Als Löser wird mit der Angabe von `solver_type = 1` das konjugierte Gradienten–Verfahren (CG) ausgewählt. Die Abbruchbedingung des iterativen Verfahrens ist bei Unterschreiten der Toleranz `epsilon` im Residuum erfüllt, wobei jedoch maximal `max_iter` Iterationen durchgeführt werden. Mit `delta_t` kann eine feste Zeitschrittweite vorgegeben werden. Wird dieser Wert, wie hier, auf 0 gesetzt, wird δt adaptiv so gewählt, dass kein Fluidteilchen in einem Zeitschritt eine größere Strecke zurücklegt als die kleinste Gitterweite. Anschließend wird der so gewonnene Zeitschritt noch mit dem Sicherheitsfaktor `delta_t_safety_factor` multipliziert, der ≤ 1 sein sollte.

Zur Anpassung der Ausgabe können sehr viele Parameter angegeben werden. Im Beispiel sind 4 wichtige herausgegriffen. `delta_t_output` bestimmt das Zeitintervall, nach dessen Erreichen die nächste Ausgabedatei geschrieben wird. Bei paralleler Ausführung des Programms werden pro Prozess eine *.vtu–Ausgabedatei und zusätzlich insgesamt eine *.pvtu–Datei erzeugt. Diese können beispielsweise mit Paraview angezeigt werden. Da dieses Programm aber manche Daten in dieser Form nicht richtig lädt, kann die Option `combine_vtu` aktiviert werden. Sie sorgt dafür, dass nach Ausgabe der parallelen Dateien diese seriell

Datei 1 Eingabedatei `settings.txt` für das Beispiel

```
# Bent Pipe CG

# Geometry
scenario = 5                # predefined scenario (5=input_file)
input_file = bent_pipe.svg  # name of input file
d_min = 3                   # minimum allowed depth
d_max = 10                  # maximum possible depth

# Problem and Solver
domain_width = 1.0          # width of the full domain
re = 1000                   # reynolds number
t_end = 10                  # end time
solver_type = 1             # which solver to use, 0=SOR, 1=CG, 2=Cholesky, 3=LU
epsilon = 1e-15             # error tolerance of solver
max_iter = 2000             # maximum number of solver iterations
delta_t = 0.0               # time step width, 0 = adaptive
delta_t_safety_factor = 0.5 # safety factor for time step width


# Output Parameters
delta_t_output = 0.01        # time interval for writing of output files
combine_vtu = 1              # combine parallel output to one single file
output_svg = 1               # output a descriptive *.svg file
output_complete_quadtree = 1 # output full quadtree

# Boundary Conditions
use_global_bc = 0               # use (0) for bounday conditions and not (1)

# (0)
bc_velocity_dirichlet_u = 1     # velocity value u for dirichlet b.c.
bc_velocity_dirichlet_v = 0     # velocity value v for dirichlet b.c.

bc_pressure_dirichlet_1 = 0     # pressure value for first dirichlet b.c. (yellow)
bc_pressure_dirichlet_2 = 1     # pressure value for 2nd dirichlet b.c. (orange)

# (1)
boundary_left_dirichlet_u   = 0  # value for u for dirichlet b.c. on outer left side
boundary_left_dirichlet_v   = 0  # value for v for dirichlet b.c. on outer left side
boundary_right_dirichlet_u  = 0  # value for u for dirichlet b.c. on outer right side
boundary_right_dirichlet_v  = 0  # value for v for dirichlet b.c. on outer right side
boundary_top_dirichlet_u    = 1  # value for u for dirichlet b.c. on outer top side
boundary_top_dirichlet_v    = 0  # value for v for dirichlet b.c. on outer top side
boundary_bottom_dirichlet_u = 0  # value for u for dirichlet b.c. on outer bottom s.
boundary_bottom_dirichlet_v = 0  # value for v for dirichlet b.c. on outer top side
```

wieder eingelesen und zu einer gemeinsamen *.vtu–Datei kombiniert werden, die die gesamte Information enthält. Diese kann dann problemlos angezeigt werden. Mit den beiden Parametern `output_svg` und `output_complete_quadtree` werden die Erzeugung von zwei svg–Vektorgraphiken aktiviert, die nach Einlesen der Geometrie ausgegeben werden. Dabei enthält eine Datei eine Darstellung der zur Berechnung verwendeten Elemente und der Randbedingungen, die andere Datei stellt den kompletten Quadtree dar.

Als letztes werden in der Datei Parameter für die Randbedingungen gesetzt. Der Schalter `use_global_bc` entscheidet dabei die Art der Angabe der Randbedingungen. Ist er, wie hier, auf 0 gesetzt, wird die Position der Randbedingungen aus der eingelesenen Vektorgraphik bestimmt. Die in der Parameterdatei unter `# (0)` gesetzten Parameter werden zur weiteren Beschreibung herangezogen und die Parameter unter `# (1)` werden nicht beachtet. Der Vektor der *inflow*–Randbedingung kann nun durch Spezifikation der Komponenten `bc_velocity_dirichlet_u` und `bc_velocity_dirichlet_v` erfolgen. Die beiden Werte für die Druckrandbedingungen sind durch `bc_pressure_dirichlet_1` und `bc_pressure_dirichlet_2` anzugeben.

Falls jedoch `use_global_bc` auf 1 gesetzt ist, werden die Parameter unter `# (1)` statt derjenigen unter `# (0)` betrachtet. Diese Wahl ist möglich, wenn es sich um ein quadratisches Berechnungsgebiet handelt. Dies

ist beispielsweise der Fall, wenn `scenario = 3` gewählt wurde und keine Vektorgraphik mit Informationen zu Randbedingungen vorliegt. Dann werden an den äußeren Rändern des quadratischen Gebiets für die Geschwindigkeit die Dirichlet–Randbedingungen angenommen, die durch die aufgeführten Parameter bestimmt werden.

3.1.3. Ausführen des Programms

Das Programm nimmt als Kommandozeilen–Argument den Dateinamen der Parameter–Datei entgegen. Wird kein Parameter übergeben, so ist der Standardwert „settings.txt“. Genau wie bei der Vektorgraphik wird nach der Datei zunächst im aktuellen Verzeichnis gesucht, dann im übergeordneten und so weiter, bis zum 3 Ebenen übergeordneten Verzeichnis. Es wird auf jeder Ebene auch jeweils in einem möglichen Unterordner „input“ gesucht. Dies ermöglicht das Anlegen einer Ordnerstruktur mit verschiedenen Parameter–Eingabedateien. Die Vektorgrafiken für verschiedene Beispiele können dagegen in einem einzigen „input“–Verzeichnis gespeichert sein. Wenn nun das Simulationsprogramm von den verschiedenen Verzeichnissen aus aufgerufen wird, werden jeweils die benötigten Vektorgraphiken gefunden.

Kann das Programm keine Parameterdatei finden, werden für alle Parameter die Standardwerte verwendet. Dies entspricht der Simulation des Beispiels „lid driven cavity“ auf regulärem Gitter mit 32×32 Elementen.

Bei diesem Szenario handelt es sich um die Strömung in einem quadratischen Gebiet. Wie in Abbildung 3.2 gezeigt, werden an den Rändern unten, links und rechts für die Geschwindigkeit *noslip*–Randbedingungen vorgegeben. Am oberen Rand wird eine Dirichlet–Randbedingung mit $\mathbf{u} = (1,0)^\top$ gesetzt. Nach Beginn der Simulation bildet sich im oberen rechten Bereich des Gebiets ein im Uhrzeigersinn rotierender Wirbel, der sich im Laufe der Zeit zur Mitte hin vergrößert. Zum Zeitpunkt $t = 20$ hat das System den in Abbildung 3.2(b) gezeigten stationären Zustand erreicht, bei dem der Wirbel das komplette Gebiet ausfüllt. Bei entsprechend feinem Gitter bilden sich in den unteren Ecken Gegenwirbel. Der Druck nimmt das in 3.2(c) gezeigte Feld an.

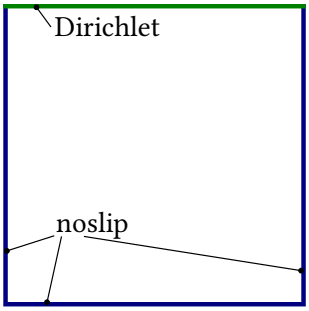
Jeder Parameter, der in der Parameterdatei möglich ist, kann auch direkt beim Aufruf über die Kommandozeile gesetzt werden. Dafür werden dem Programm Parameter der Form `-p<Bezeichner>=<Wert>` übergeben, die jeweils die mögliche Wertbelegung in der Parameterdatei ersetzen.

Ein möglicher Aufruf des Programms mit 4 Prozessen unter Verwendung der Datei 1 als „settings.txt“, wobei die maximale Diskretisierungsfeinheit `d_max` noch auf 6 verringert wird, sieht wie folgt aus:

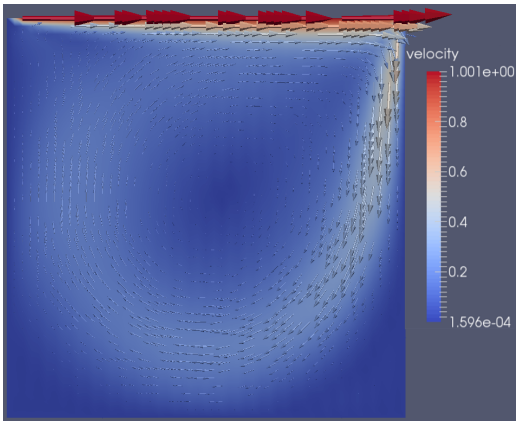
```
[root@localhost bent_pipe]# mpirun -n 4 ../../../qtree -pdmax=6
Starting with 4 processes.
Reading from "settings.txt".
Scenario 5 (input file "bent_pipe.svg")
Write to file output/complete_quadtree.svg"...
process 0: time to build FEMesh: 660 ms = 0 min 0 s
process 1: time to build FEMesh: 662 ms = 0 min 0 s
process 3: time to build FEMesh: 662 ms = 0 min 0 s
process 2: time to build FEMesh: 663 ms = 0 min 0 s
process 3: max level: 6, number of elements: 389, number of stored nodes: 438, memory: 81.516 kB
process 1: max level: 6, number of elements: 332, number of stored nodes: 389, memory: 71.287 kB
process 2: max level: 6, number of elements: 466, number of stored nodes: 516, memory: 96.626 kB
process 0: max level: 6, number of elements: 260, number of stored nodes: 304, memory: 55.876 kB
Write to file "output/bent_pipe_out.svg"...

3:50 (7%) [3.421%] t=0.746, delta_t_=0.00175 solved in 305 iterations, res=8.36462e-15
```

Das Programm zeigt zunächst einige Informationen über die verwendeten Dateien an. Anschließend gibt jeder Prozess eine Statistik zu seinem erzeugten Finite–Elemente–Gitter aus. Wenn dann die eigentliche Simulation läuft, werden in der letzten Zeile fortlaufend Informationen zum aktuellen Zustand angezeigt.

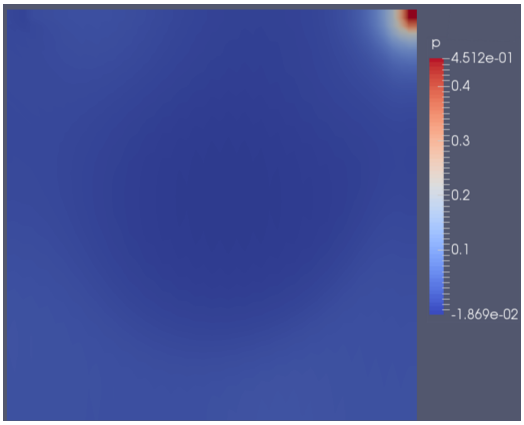
Abbildung 3.2. Szenario „driven cavity“



(a) Randbedingungen



(b) Geschwindigkeitsfeld. Farblich dargestellt ist der Betrag der Geschwindigkeit, Pfeile geben die Richtung des Vektorfelds an.



(c) Druckfeld

Die erste Angabe `3:50` liefert die erwartete Dauer bis zum Ende der Simulation bei Erreichen von `t_end`. In diesem Beispiel sind es also noch 3 Minuten 50 Sekunden. Der nächste Wert `(7%)` ist der aktuelle Fortschritt. Der darauffolgende Wert in eckigen Klammern `[3.421%]` gibt den Zeitfortschritt in der Berechnung im Verhältnis zur wirklichen Dauer an. Dies ist also ein Maß für die Schnelligkeit der Berechnung. 100% würden einer Echtzeit–Simulation entsprechen. Darauf folgt die aktuelle Simulationszeit und die Zeitschrittweite. Anschließend ist angegeben, wie viele Iterationen der Löser für die Berechnung des letzten Zeitschritts benötigte und welchen Wert das Residuum am Ende annahm. So kann auf einen Blick festgestellt werden, ob der Löser divergiert ist.

3.1.4. Ausgabe

Die Ausgabedateien, die das Programm erzeugt, werden im Unterordner output abgelegt. Zwei verschiedene Typen von Ausgabedateien werden erzeugt: zum einen svg–Graphiken, die die Geometrie und gesetzten Randbedingungen veranschaulichen. Diese werden zu Beginn der Simulation erzeugt. Zum anderen werden Dateien ausgegeben, die die Systemgrößen Druck und Geschwindigkeit für diskrete Zeitpunkte beinhalten.

Eine der erzeugten Vektorgraphiken ist in Abbildung 3.3 dargestellt. Zu sehen sind die erzeugten Elemente, die die Geometrie approximieren. Die gesetzten Randbedingungen sind beschriftet und die Elemente, in denen die Druck–Dirichletrandbedingungen gelten, sind farblich markiert. Die *noslip*–Randbedingungen sind durch blaue Markierungen an den Knoten eingezeichnet. Dass manche Markierungen außerhalb des Gebiets zu liegen scheinen, liegt daran, dass diese zu hängenden Knoten gehören, aber an der Position der

entsprechenden Ersatzknoten eingezeichnet sind. In der zweiten erzeugten Vektorgraphik ist der komplette Quadtree, auch außerhalb des Gebiets, dargestellt. In Abbildung 3.4 ist dieser für das Problem mit einem maximalen Level von 10 dargestellt. Man erkennt auch den minimalen Level `d_min = 3`, den die groben Elemente links oben annehmen.

Durch weitere Parameter lassen sich auch die Element- und Knotenindices sowie die Element–IDs in den Quadtrees darstellen.

Abbildung 3.3. Ausgabedatei `bent_pipe_out.svg`

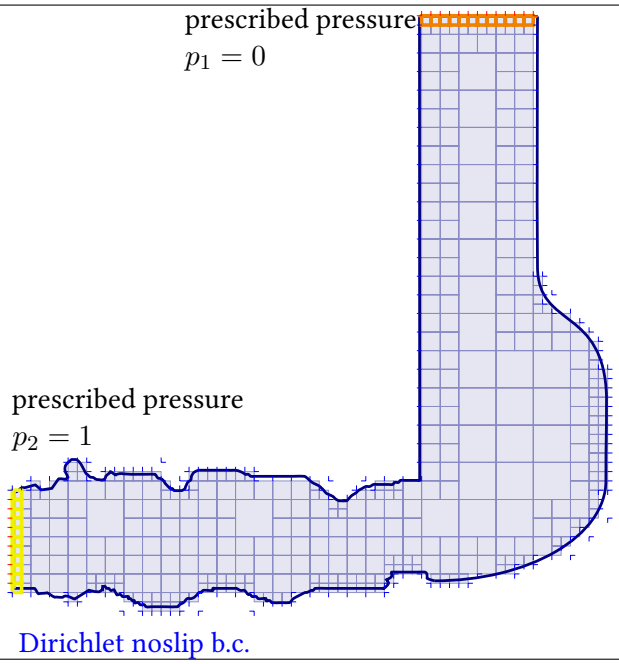
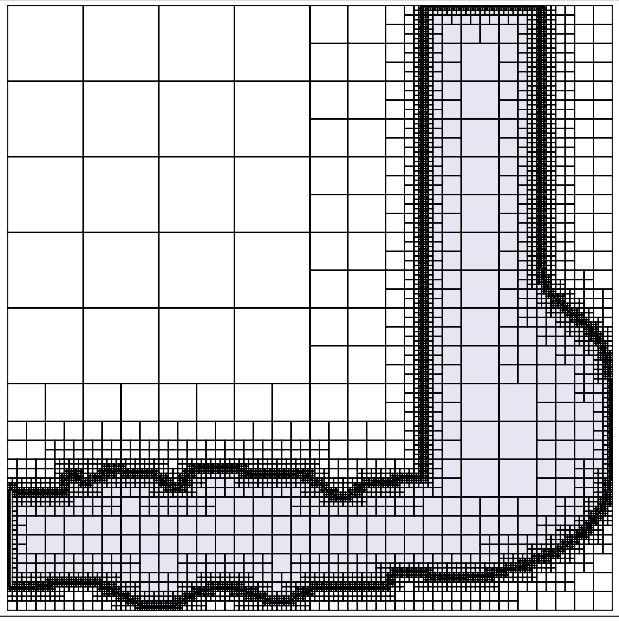


Abbildung 3.4. Ausgabedatei `complete_quadtree.svg`, hier für `d_max = 10`.



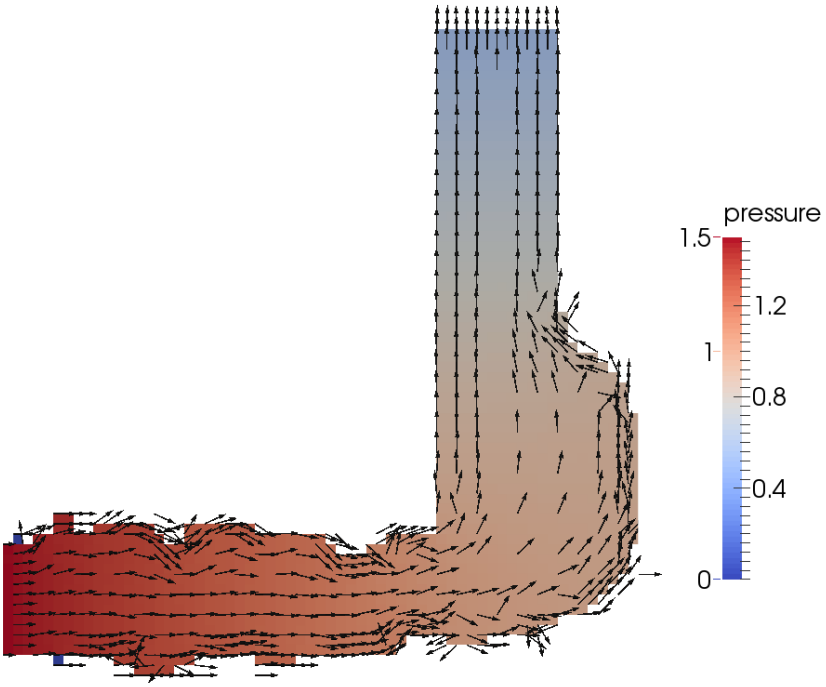
Nach Abschluss der Simulation befinden sich im „output“–Verzeichnis außerdem Dateien der Form

```
output_x_0.vtu    output_x_1.vtu    output_x_2.vtu    output_x_3.vtu
output_x.pvtu     output_x_combined.vtu
```

wobei x eine fortlaufende Nummer ist. Die 4 Dateien in der ersten Zeile sind die jeweiligen Ausgabedateien der 4 Prozesse mit den Daten des entsprechenden Zeitschritts. Es handelt sich um das parallele VTK–Dateiformat, wobei die Daten als „Unstructured Grid“ repräsentiert werden. Als Datenfelder sind die Geschwindigkeit, der Druck sowie die rechte Seite der Poisson–Gleichung und die Vorticity enthalten. Außerdem ist für jedes Element die Prozessnummer seines bearbeitenden Prozesses gespeichert.

Zu diesen *.vtu–Dateien gehört die Datei mit der Endung *.pvtu, die eine Auflistung der pro Datei gespeicherten Felder und eine Referenzierung der 4 Dateien enthält. Diese Datei kann mit dem *Visualization Toolkit* oder auch mit Paraview geladen werden. Bei der Verwendung von Paraview fällt auf, dass die Prozessnummern, die ganzzahlige Werte sind, nicht richtig eingelesen werden. Das Problem tritt bei der Datei `output_x_combined.vtu`, die die Daten der 5 anderen Dateien beinhaltet, nicht auf, sodass diese stattdessen verwendet werden kann. In Abbildung 3.5 ist eine mit Paraview generierte Ansicht der Simulation für $t = 0\ 02$ zu sehen. Man erkennt den Druckabfall entlang der Strömungsrichtung durch den Farbverlauf von rot nach blau. Am Beginn des Rohrs unten links sind auch zwei Randelemente mit blauer Füllung zu sehen, in denen der Druck den Wert Null annimmt. Dies folgt aus deren exponierter Position am Rand, die dazu führt, dass für jeweils drei Knoten die Geschwindigkeit gemäß der *noslip*–Randbedingung auf Null gesetzt wird.

Abbildung 3.5. Darstellung des Drucks als heatmap und des Geschwindigkeitsfeldes durch unskalierte Pfeile

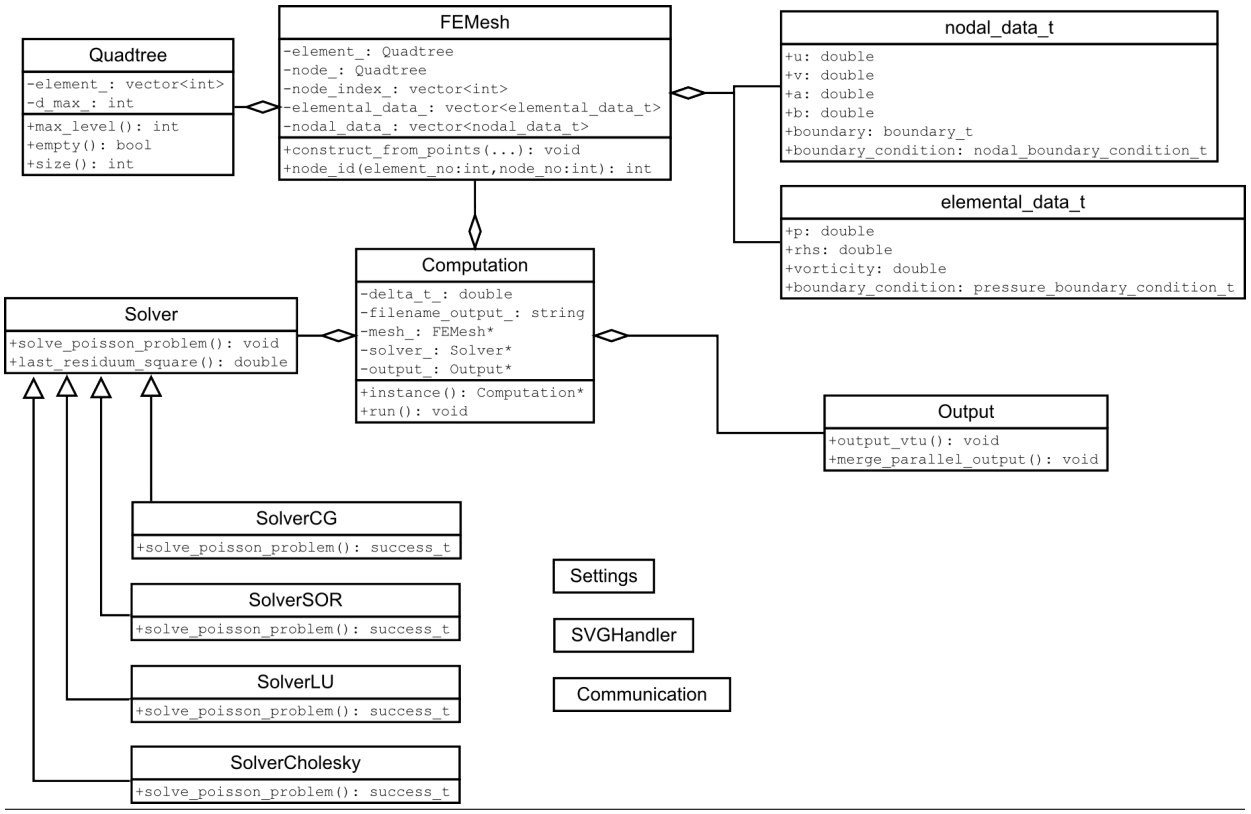


3.2. Implementierungsdetails

Die für das Programm verwendete Programmiersprache ist C++, für die Parallelisierung wurde das Message Passing Interface (MPI) verwendet. In diesem Abschnitt werden Aspekte der Implementierung vorgestellt. Nach einem Überblick über die Klassenstruktur wird die Umsetzung des Einlesens der Geometrie und anschließend das Lösen des Poissonproblems genauer betrachtet.

3.2.1. Klassenstruktur

Abbildung 3.6. UML Klassendiagramm der wichtigsten Klassen



Im Klassendiagramm 3.6 sind die wesentlichen Klassen aufgeführt. Die Klasse **`Quadtree`** beschreibt einen als Vektor seiner Element-IDs repräsentierten linearen Quadtree. Die Klasse enthält alle Methoden, um einen Quadtree aus im Gebiet liegenden Punkten parallel zu erstellen, zwischen den Prozessen aufzuteilen und die 2:1–Balancierungseigenschaft herzustellen, wie in Kapitel 2.2 beschrieben.

Das Finite–Elemente–Gitter wird durch die Klasse **`FEMesh`** repräsentiert. Die Struktur des Gitters ist in den beiden Quadtrees `element_` und `node_` enthalten, die den beschriebenen Quadtrees der Elemente und der Knoten entsprechen. Die Verknüpfung zwischen einem Element und seinen 4 Knoten geschieht durch den Vektor `node_index_`. Bei N Elementen enthält er $4N$ Einträge, sodass der Eintrag $4i + j$ den Index des Knoten Nummer j im Element i enthält. Dies ist der Index für den entsprechenden Eintrag im Elementvektor des linearen Quadtree. In ähnlicher Weise sind für jeden Knoten Informationen gespeichert, ob es sich um einen hängenden Knoten handelt und wenn ja, in welcher Richtung sich der Ersatzknoten befindet. Die eigentlichen Daten, also die Größen, die für die Berechnung verwendet werden, sind in zwei weiteren Vektoren `elemental_data_` und `nodal_data_` gespeichert. Der Vektor `elemental_data_` enthält für jedes Element die Daten, die in der Struktur **`elemental_data_t`** zusammengefasst sind. Neben dem Druck `p` und der rechten Seite `rhs` ist dies die Information in `boundary_condition`, ob das Element eine Dirichlet–Druckrandbedingung besitzt und wenn ja, welcher Wert dafür gilt. Außerdem sind noch weitere Hilfsfelder, die beim CG–Verfahren benötigt werden, enthalten. Analog dazu enthält die Struktur **`nodal_data_t`** die Größen, die an den Knoten positioniert sind. In der `FEMesh`–Klasse wird jedem Knoten durch den Vektor `nodal_data_` ein Objekt dieser Struktur zugeordnet. Neben den Geschwindigkeitsgrößen `u`, `v`, `a` und `b` wird an jedem Knoten auch wieder die Information über eine mögliche Randbedingung, die durch `boundary_condition` gespeichert wird, benötigt. Die Variable `boundary` enthält ein Flag–Feld, das angibt, an welcher Seite des Rands sich der Knoten befindet.

Die Klasse `FEMesh` enthält noch einige weitere Informationen wie z. B. für jeden Prozess das erste und das letzte Element, dass der Prozess besitzt. Zudem wird auch ermöglicht, effizient über bestimmte Knoten zu iterieren. Dies geschieht, indem Vektoren der entsprechenden Indices vorgehalten werden. Beispielsweise enthält ein Vektor alle Indices der Knoten, die eine *noslip*–Randbedingung besitzen. Um bei der Berechnung die Geschwindigkeiten an diesen Knoten auf Null zu setzen, kann über den Vektor der Indices iteriert werden und über den jeweiligen Index auf die knotenweisen Daten zugegriffen werden. Solche Vektoren mit Indices existieren für alle Randbedingungen, sowohl für Knoten als auch für Elemente. Außerdem wird für jeden Prozess ein solcher Vektor gespeichert, der die Indices der Knoten enthält, die der lokale Prozess mit dem jeweiligen Prozess gemeinsam hat. Dieser wird verwendet, um Knotenwerte an den Ränder zwischen Prozessen zusammenzuführen.

Diese Klasse ist die umfangreichste des Projekts, da sie viele Methoden enthält. Sie stellt Funktionalitäten bereit, um das Finite–Elemente–Gitter parallel aufzubauen und die Knoten mit den Elementen zu verknüpfen. Zusätzlich enthält sie auch Prozeduren zur Erzeugung eines weiteren Finite–Elemente–Gitters aus dem aktuellen mit an bestimmten Stellen verfeinerten oder vergröberten Elementen.

Die eigentliche Berechnung findet in der Singleton–Instanz der Klasse **`Computation`** statt. Bei Aufruf der Hauptmethode `run()` wird zunächst, je nach Parametrisierung, das Einlesen der Eingabedateien durchgeführt. Je nach Parameter wird ein Objekt der entsprechenden Löser–Klassen `SolverCG`, `SolverSOR`, `SolverLU` oder `SolverCholesky` erzeugt, das später zum Lösen des Poissonproblems verwendet wird.

Das Finite–Elemente–Gitter `mesh_` wird durch Aufruf seines Konstruktors erzeugt. Die Simulation wird anschließend durchgeführt, indem in einer Schleife über die Zeitschritte iteriert wird. Die Struktur dieser Schleife ist in Code–Abschnitt 3.1 dargestellt. Die Klasse `Computation` enthält Methoden, die die Schritte in der Schleife ausführen, die Ausgabe ausgenommen. Dafür wird ein Objekt der Klasse **`Output`** verwendet. Dieses Objekt berechnet die Position der hängenden Knoten, interpoliert deren Werte und schreibt sie dann zusammen mit den Werten der regulären Knoten in die Ausgabedateien.

Außerdem existieren noch die Hilfsklassen `Settings`, `SVGHandler` und `Communication`, die jeweils als Singleton ausgeführt sind. Die Klasse **`Settings`** verwaltet alle Parameter. Sie liest die Parameterdatei sowie die Kommandozeilenparameter ein und setzt die Standardwerte für die Parameter, die nicht angegeben wurden. Die Klasse **`SVGHandler`** kümmert sich um das Einlesen und Schreiben von svg–Vektorgraphiken. Sie erzeugt Punkte auf dem Rand des in der Vektorgraphik definierten Gebiets, welche als Eingabe in den Algorithmus zur Erzeugung des Quadtrees verwendet werden. Außerdem ist das Objekt dieser Klasse für das Erzeugen der Vektorgraphiken, die das Finite–Elemente–Gitter darstellen, zuständig. Die dritte Hilfsklasse **`Communication`** verwaltet den MPI–Zustand und stellt einige Kommunikationsmethoden bereit, die von den Klassen `Quadtree`, `FEMesh`, `Computation` und den Solver–Klassen aufgerufen werden.

Code–Abschnitt 3.1 Hauptschleife der Simulation

```
u,v,p = 0
Ausgabe des Startzustands
while t < t_end do
    t = t + δt
    setze Randbedingungen für u, v
    berechne a, b
    berechne rhs
    löse Poissonproblem
    berechne neue Geschwindigkeiten u, v
    berechne neues δt
    berechne vorticity
    verfeinere/vergröbere evtl. Gitter
    Ausgabe der neuen Werte
end
```

3.2.2. Einlesen der Geometrie

Im Folgenden wird genauer beschrieben, wie die Informationen aus der Vektorgrafik eingelesen werden. Dies beinhaltet die Exktraktion der Geometrie, mit der das Finite–Elemente–Gitter erzeugt wird, außerdem die Identifizierung von Zellen innerhalb und außerhalb des Gebiets und schließlich das Setzen der Randbedingungen an den entsprechenden Knoten. Diese Aktionen finden alle in der Klasse `SVGHandler` statt.

Der erste Schritt besteht darin, die svg–Datei zu parsen. Die Beispiel–svg–Datei 2 beschreibt eine quadratische Geometrie mit einem grünen Rand oben und blauen Rändern links, unten und rechts, wenn der Koordinatenursprung unten links angenommen wird. Die Graphik entspricht Abbildung 3.2(a). Das Quadrat hat eine Seitenlänge von etwas weniger als 1000, sodass sich der Rand nicht exakt auf der Kante der 1000×1000 großen Zeichenfläche befindet, sondern etwas weiter innen. Dies soll sicherstellen, dass die Ränder bei der Darstellung der Graphik auch angezeigt werden und durch numerische Ungenauigkeiten nicht außerhalb des Bildes geraten.

Die vom Programm erkannten svg–Tags stellen eine Teilmenge des svg–Standards dar. Zum einen werden für die Geometriebeschreibung irrelevante Tags, wie z. B. für die Darstellung von Text oder die Definition von Animationen nicht beachtet. Zum anderen kann eine Geometrie auf unterschiedliche Weise repräsentiert werden, z. B. kann ein vollständiger Kreis sowohl durch die Tags `<cicle />`, `<ellipse />` als auch durch ein `<path />`–Element dargestellt werden. Das `<path />`–Element stellt das allgemeinste Geometrieelement dar, mit dem alle mit svg darstellbaren Geometrien beschrieben werden können. Der Umfang an Tags, die das Programm verarbeiten kann, ist auf die Elemente `<g></g>`, `<path />` und `<rect />` beschränkt. Wenn die Vektorgraphik mithilfe des svg–Editors *Inkscape* erzeugt wird, kommen nur diese Elemente vor, sodass dieser Editor für die Erzeugung der Eingabegraphiken zu empfehlen ist.

Datei 2 svg–Vektorgrafik mit zwei Pfaden

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<svg
   xmlns="http://www.w3.org/2000/svg"
   width="1000"
   height="1000">
  <g transform="translate(0.05,0.05)">
    <path
       style="fill:none;stroke:#000080;stroke-width:0.02;"
       d="M 0,1 0,0 1,0 l 0,1"
           transform="scale(999.9)" />
    <path
       style="fill:none;stroke:#008000;stroke-width:0.02;"
       d="M 0,1 1,1"
           transform="scale(999.9)" />
  </g>
</svg>
```

Anhand der Beispiel–svg–Datei 2 wird nun die Struktur einer svg–Vektorgraphik vorgestellt. Als Element auf höchster Ebene kommt ein `<svg></svg>`–Element vor, das alle weiteren Tags enthält. Es besitzt als Optionen die Breite und Höhe der Fläche, auf der im Folgenden die Pfade definiert werden. Diese Angaben werden vom Programm benötigt, um später bei der Simulation die Längenangaben der Geometrie so zu skalieren, dass die Breite auf die Größe des Parameters `domain_width` abgebildet wird. In diesem Element ist in der Beispieldatei ein `<g></g>`–Element enthalten. Es wird allgemein verwendet, um Pfade zu gruppieren. Es kann die Option `transform` besitzen, durch die alle Koordinaten der Elemente dieser Gruppe verschoben, rotiert, skaliert oder anders transformiert werden. Darin enthalten sind schließlich zwei `<path />`–Elemente, die mit ihrer `d`–Option die Geometrie der Pfade beschreiben. Ein weiteres mögliches Element wäre `<rect />`, das ein Rechteck beschreibt. Beiden Elementen gemeinsam sind die möglichen Optionen `transform`, welche

wieder eine Transformation, die auf die Koordinaten angewandt wird, darstellt, sowie `style`, mit welchem die Farbe festlegt werden kann.

Um einen Quadtree zu erstellen, der am Rand des Gebiets eine vorgegebene Auflösung hat, werden auf dem Rand Punkte in einem bestimmten Abstand erzeugt, die dann als Eingabeelemente für den Algorithmus aus Kapitel 2.2.3 verwendet werden. Der Abstand kann entweder durch den Parameter `sample_width_border` direkt angegeben werden. Eine andere Möglichkeit, ist es dessen Inverse `inverse_sample_width_border` zu spezifizieren. Bei einer Gebietsgröße von $1{,}0$ entspricht der inverse Wert gerade der Anzahl Elemente über die Breite des Gebiets. Soll z. B. der Rand mit Elementen auf Level 10 approximiert werden, kann `inverse_sample_width_border` einfach auf $2^{10} = 1024$ gesetzt werden. Mit dieser Einstellung erhält man den Rand mit entsprechend feinen Elementen abgebildet. Im Inneren treten gröbere Elemente auf, so dass die 2:1–Balancierungseigenschaft erfüllt ist. Möchte man auch im Inneren ein spezielles Elementlevel vorgeben, kann analog der Parameter `inverse_sample_width_interior` angegeben werden. Ist dieser ungleich Null, werden über das gesamte Gebiet Punkte mit dem entsprechenden Abstand erzeugt, die ebenso als Eingabe in den Erzeugungsalgorithmus für den Quadtree aufgenommen werden. Es muss jeweils darauf geachtet werden, dass die Parameter `d_min` und `d_max` die gewünschten Level zulassen.

Im Anschluss erzeugt die Methode der Klasse `FEMesh` unter Verwendung von Methoden der Klasse `Quadtree` die beiden Quadtrees für die Elemente und Knoten und ordnet die nicht–hängenden Knoten den Elementen zu. Bisher befinden sich auch außerhalb des Gebietes noch Elemente und Knoten. Diese Knoten sind noch notwendig, um die Knotentypen zu bestimmen und abzuspeichern. Ist dies erledigt, können die Knoten und Elemente außerhalb gelöscht werden. Die Überprüfung, ob sich ein Element innerhalb des Gebiets befindet, geschieht wieder innerhalb der Klasse `SVGHandler` anhand der aus der Vektorgraphik eingelesenen Pfade.

Pro Element werden 3×3 Punkte so gleichmäßig platziert, dass der Abstand zwischen zwei Punkten und zwischen einem Punkt und dem Rand des Elements immer gleich ist. Für jedes Element wird dann untersucht, ob es sich innerhalb des Gebiets befindet. Dies passiert durch Verfolgung eines Strahls von dem Punkt in die positive x_1–Richtung. Dabei wird die Anzahl an Überschneidungen des Strahls mit den Pfaden des Rands gezählt. Wenn diese Anzahl gerade ist, gab es gleichviele Überquerungen des Randpfades, wo die Seite von innerhalb des Gebiets nach außerhalb welchselte wie Überquerungen, wo der Strahl von außen nach innen wechselte. Da sich der Strahl am Ende sicher außerhalb befindet, war auch der Startpunkt außerhalb. Dies bedeutet, dass die Punkte innerhalb gerade diejenigen sind, bei denen es eine ungerade Anzahl an Überschneidungen gab.

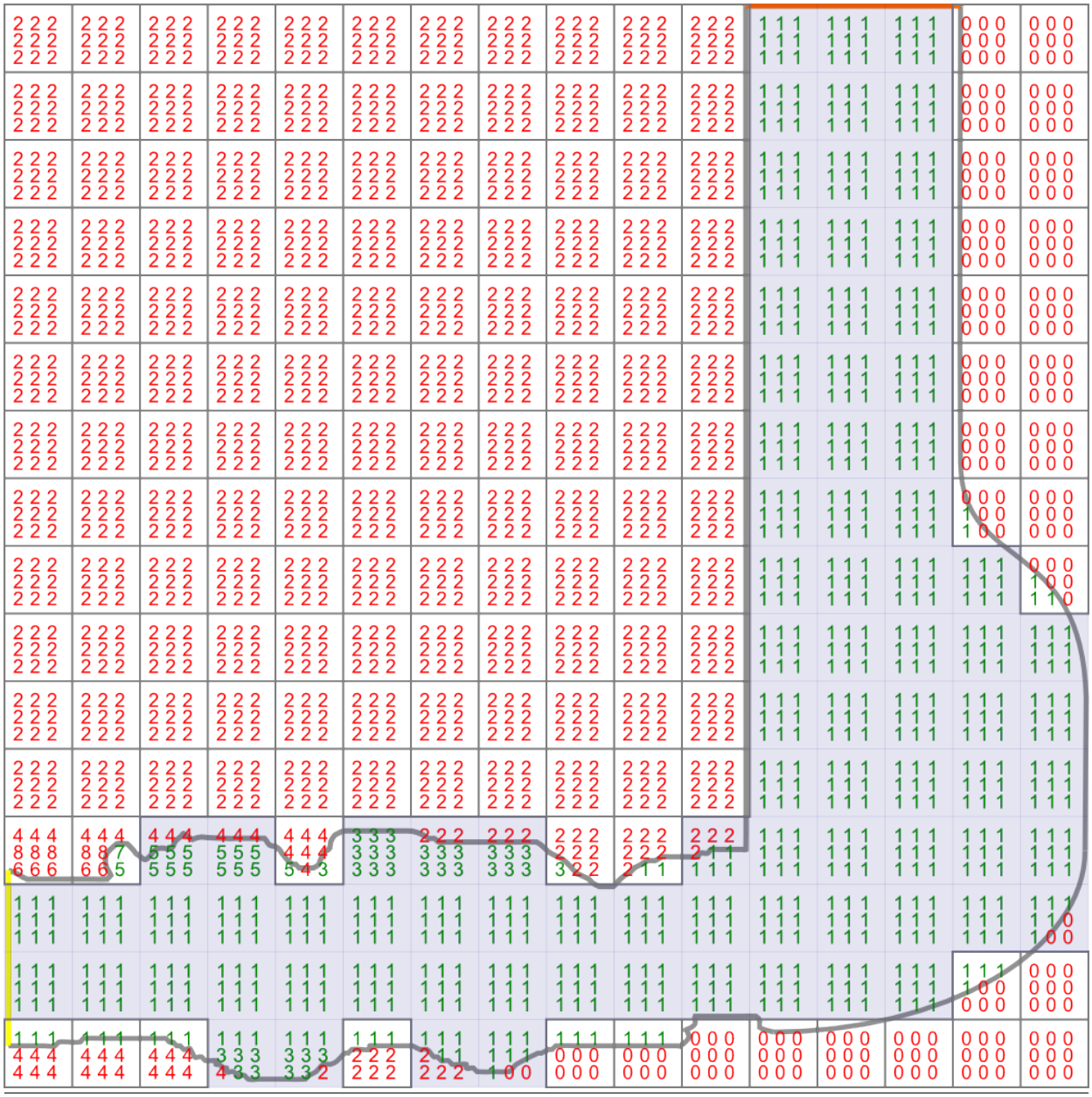
Für jeden Punkt wird bei diesem Verfahren über alle Pfad–Elemente iteriert, um festzustellen, ob sie die x_2–Komponente des Punkts überqueren. Da die Anzahl an Pfaden in einer Eingabedatei meist gering ist, stellt dieses Vorgehen kein Problem für die Laufzeit des Programms dar.

Ein Element wird als innerhalb des Gebietes angesehen, wenn die Mehrheit seiner Punkte, also mindestens 5, sich innerhalb befindet. In Abbildung 3.7 sind die 9 Punkte in jedem Element durch Zahlen dargestellt, die die Anzahl an Schnitten des Strahls mit dem Rand angeben. Für die Punkte, die sich danach innerhalb des Gebiets befinden, ist die Zahl grün gefärbt, für Punkte außerhalb rot. Man erkennt, dass in einigen Elementen, die vom Rand geschnitten werden, manche Punkte innerhalb und manche außerhalb liegen. Dann wird die Mehrheit der Punkte betrachtet.

3.2.3. Löser

Nach Erzeugen des Finite–Elemente–Gitters kann die Berechnung nach dem vorgestellten Algorithmus 3.1 erfolgen. Die Berechnung der vorläufigen Geschwindigkeiten $\mathbf{u}_v$ und $\mathbf{v}_v$ erfolgt durch Matrixmultiplikation mit den Operatoren C und D, wie in Kapitel 2.4.3 dargestellt. Der Wert wird in die Variablen `u` und `v` gespeichert. Diese Berechnung ist durch eine Schleife über alle Knoten möglich, die der jeweilige Prozess

Abbildung 3.7. Veranschaulichung des Algorithmus zur Findung der Elemente im Inneren. Dargestellt ist ein Quadtree, wobei in jedem Element die 3×3 Punkte durch Zahlen dargestellt werden, die die Anzahl der Schnitte mit der Kontur entlang des Strahls vom Punkt aus in positiver x_1–Richtung beschreibt. Ungerade Zahlen sind grün, gerade rot. Sind mindestens 5 Zahlen pro Element grün, so befindet sich das Element innerhalb des Gebiets und wird grau markiert.



besitzt. Danach haben die Knoten im Inneren das lokalen Rechengebiets bereits die richtigen Werte für $\mathbf{u}_v$ und $\mathbf{v}_v$. In die Werte der Knoten auf dem Rand zwischen Teilgebieten verschiedener Prozesse sind jedoch nur die Beiträge des lokalen Prozesses eingeflossen. Zudem sind diese Knoten auf mehreren Prozessen vorhanden und haben dort möglicherweise unterschiedliche Werte erhalten. Hier wird also ein Kommunikationsschritt benötigt, bei dem die Werte desselben Knotens auf verschiedenen Prozessen addiert werden und die Summe wieder zu den Prozessen zurück gesendet wird.

Als nächstes wird die rechte Seite $rhs = -1/\delta t \, (A_1 \, \mathbf{u}_v + A_2 \, \mathbf{v}_v)$ berechnet. Dies ist durch eine Schleife über die Elemente möglich, wobei jedes Element die Beiträge der Operatoren A_1 und A_2 seiner vier Knoten aufaddiert.

Dann kann das diskrete Poissonproblem

$$\tilde{L}\, p = rhs \qquad \text{mit } \tilde{L} = A_1 \, \tilde{M}^{-1} \, A_1^\top + A_2 \, \tilde{M}^{-1} \, A_2^\top$$

berechnet werden. Dies ist ein potentiell großes lineares Gleichungssystem mit der dünnbesetzten Matrix $\tilde{L}$.

Um $\tilde{L}$ auf einen Vektor p mit Werten in den Elementen anzuwenden, wird Algorithmus 3.2 verwendet. In einer Schleife über alle Elemente werden zunächst die elementweisen Beiträge der Operatoren $A_1^\top$ und $A_2^\top$ auf den Wert des Elements angewandt und das Ergebnis in den Knotenvariablen a und b aufsummiert. Auch hier ist in der Folge ein Kommunikationsschritt notwendig, um die Werte der Knoten, die sich auf mehreren Prozessen befinden, zusammenzuführen. Anschließend werden alle Knotenwerte mit den entsprechenden inversen gelumpten Einträgen der Matrix M multipliziert. Es handelt sich um jeweils eine Multiplikation für a und b für jeden Knoten. Nach diesem Schritt liegen in den Knoten die Werte $a = \tilde{M}^{-1} A_1^\top p$ und $b = \tilde{M}^{-1} A_2^\top p$ vor.

Als nächstes wird wieder über die Elemente iteriert. Der Wert p' im Element wird aus der Anwendung von A_1 auf a plus der Anwendung von A_2 auf b erhalten.

Code–Abschnitt 3.2 Berechnung der Matrix–Vektor–Multiplikation $p' = \tilde{L}\, p$ mit der Matrix $\tilde{L} = A_1 \, \tilde{M}^{-1} \, A_1^\top + A_2 \, \tilde{M}^{-1} \, A_2^\top$

```
a(kn) = b(kn) = 0    ∀Knoten kn

for Element el do
    a(el, 1) = a(el, 1) + A_1^⊤koeff[1] * p(el),    b(el, 1) = b(el, 1) + A_2^⊤koeff[1] * p(el)
    a(el, 2) = a(el, 2) + A_1^⊤koeff[2] * p(el),    b(el, 2) = b(el, 2) + A_2^⊤koeff[2] * p(el)
    a(el, 3) = a(el, 3) + A_1^⊤koeff[3] * p(el),    b(el, 3) = b(el, 3) + A_2^⊤koeff[3] * p(el)
    a(el, 4) = a(el, 4) + A_1^⊤koeff[4] * p(el),    b(el, 4) = b(el, 4) + A_2^⊤koeff[4] * p(el)
end

for Knoten kn auf Interprozessrand do
    Reduziere a(kn)
    Reduziere b(kn)
end

for Knoten kn do
    a(kn) = a(kn) * M̃^-1(kn)
    b(kn) = b(kn) * M̃^-1(kn)
end

for Element el do
    p'(el) =  A_1koeff[1] * a(el, 1) + A_1koeff[2] * a(el, 2) + A_1koeff[3] * a(el, 3) + A_1koeff[4] * a(el, 4)
            + A_2koeff[1] * b(el, 1) + A_2koeff[2] * b(el, 2) + A_2koeff[3] * b(el, 3) + A_2koeff[4] * b(el, 4)
end
```

Zur Lösung des Gleichungssystems sind zwei iterative Löser, das *Successive Over Relaxation*–Verfahren (SOR) und die *Methode der konjugierten Gradienten* (CG) umgesetzt. Zum Vergleich wurden auch zwei direkte Löser, unter Verwendung von *LU* und *Cholesky*–Zerlegung, implementiert.

SOR–Verfahren

Beim SOR–Verfahren handelt es sich um ein modifiziertes Gauss–Seidel–Verfahren. Die Grundidee zur Lösung des Gleichungsystems $A\,\mathbf{x} = \mathbf{b}$ ist die Aufteilung der Systemmatrix in drei Teilmatrizen D, L und

R sodass gilt: $A = D - L - R$. Damit folgt aus $A\,\mathbf{x} = \mathbf{b}$ die Gleichung $(D - L)\,\mathbf{x} = \mathbf{b} + R\,\mathbf{x}$. Daraus wird die Iterationsvorschrift

$$(D - L)\,\mathbf{x}^{(k+1)} = \mathbf{b} + R\,\mathbf{x}^{(k)} \tag{3.1}$$

abgeleitet. Angefangen mit einem Startwert $\mathbf{x}^{(0)}$ erhält man durch sukzessive Anwendung von (3.1) die Lösung. Die Iteration wird beendet, falls das Residuum $\mathbf{r}^{(k)} := b - A\,\mathbf{x}^{(k)}$ in einer Norm einen Toleranzwert ε unterschreitet. Durch Umformung erhält man aus (3.1):

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + D^{-1}\big(\mathbf{b} + L\,\mathbf{x}^{(k+1)} + R\,\mathbf{x}^{(k)}\big). \tag{3.2}$$

Um hierbei die Matrix D mit geringem Aufwand invertieren zu können, wird sie als Diagonalmatrix mit den Einträgen auf der Diagonale von A gewählt. L und R werden so gewählt, dass der Algorithmus nur einen Vektor zur Speicherung der Werte $\mathbf{x}^{(k)}$ und $\mathbf{x}^{(k+1)}$ verwenden muss. Die neuen Werte von $\mathbf{x}^{(k+1)}$ überschreiben dann jeweils die alten. Da in der Iterationsvorschrift (3.2) jedoch $\mathbf{x}^{(k)}$ auf der rechten Seite auftaucht, darf das Resultat der Matrixmultiplikation mit R nur von Werten abhängig sein, die zuvor noch nicht überschrieben wurden. Genauso können bei der Multiplikation von $L\,\mathbf{x}^{(k+1)}$ nur solche Werte des Vektors verwendet werden, die bereits auf den Schritt $k+1$ aktualisiert wurden. Dies hängt jeweils von der Reihenfolge ab, in der die Komponenten des Vektor $\mathbf{x}$ abgearbeitet werden. Ist diese Reihenfolge o.B.d.A so festgelegt, dass die Komponenten in aufsteigender Nummerierung nacheinander bearbeitet werden, muss L als untere Dreiecksmatrix und R als obere Dreiecksmatrix gewählt werden. Deren Einträge entsprechen denen aus der Systemmatrix A, um die Zerlegungseigenschaft einzuhalten.

Das SOR–Verfahren führt nun einen Parameter ω ein, der den Korrekturschritt skaliert. Für $\omega \in (0{,}2)$ ändert sich die Konvergenzeigenschaft nicht. Die Konvergenzgeschwindigkeit kann sich jedoch erhöhen, wenn der Parameter geschickt gewählt wird.

Die Iterationsvorschrift für das SOR–Verfahren lautet somit:

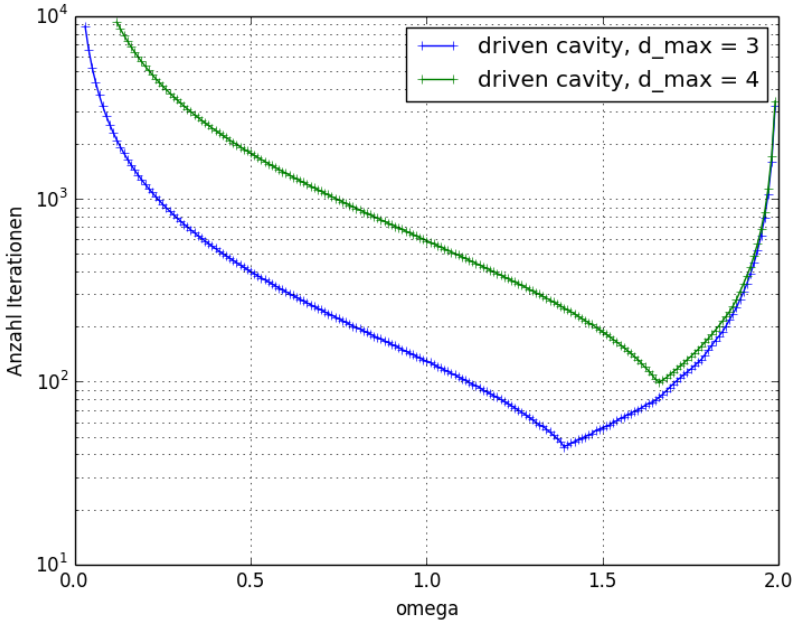
$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \omega\,D^{-1}\big(\mathbf{b} + L\,\mathbf{x}^{(k+1)} + R\,\mathbf{x}^{(k)}\big). \tag{3.3}$$

Der Parameter ω kann nun empirisch so gewählt werden, dass die Anzahl der Iterationen, bis die Toleranzgrenze erreicht ist, möglichst gering ist. Der optimale Parameter hängt von der Systemmatrix ab. In Abbildung 3.8 ist die benötigte Anzahl Iterationen bei verschiedenen Werten für Omega für zwei beispielhafte Probleme aufgetragen. Man erkennt, dass sich der beste Wert für Omega unterscheidet, dass er jedoch beides mal über 1 liegt. Im Programm lässt sich der Wert mit dem Parameter `omega` einstellen, der Standardwert ist $1{,}4$.

Falls Dirichlet–Druckrandbedingungen vorliegen, können die entsprechenden Elemente nach jedem Iterationsschritt auf die vorgegebenen Werte gesetzt werden. Dadurch wird eine Lösung gefunden, die die Randbedingungen einhält. Die Neumann–Randbedingungen werden bei der Auswertung des Differenzensterns am Rand berücksichtigt.

Für ein reguläres Gitter lässt sich das SOR–Verfahren über einen Zwei–Farben–Ansatz [AO82] gut parallelisieren. Da der Differenzenstern für die Matrix L die in Abbildung 2.22 dargestellte Form hat, ist es von Vorteil, das Ausführen der Iterationsvorschrift (3.3) in zwei aufeinanderfolgenden Schritten auszuführen. Der Vektor $\mathbf{x}$, der für jedes Element einen Eintrag enthält, wird gemäß der Struktur eines Schachbrettmusters aufgeteilt. Im ersten Schritt werden alle Vektorkomponenten der gleichen Farbe nach der Iterationsvorschrift aktualisiert. Im zweiten Schritt folgen die restlichen Komponenten der anderen Farbe. In jedem Schritt werden auch nur die Komponenten der jeweiligen Farbe gelesen, was eine parallele Abarbeitung innerhalb des Schritts durch Aufteilung des Gebets auf mehrere Prozesse möglich macht. Bei den hier betrachteten adaptiven Gittern treten je nach lokaler Zusammensetzung verschiedener Elementgrößen kompliziertere Abhängigkeiten in den Differenzensternen auf. Außerdem sind in der Datenstruktur

Abbildung 3.8. Anzahl Iterationen (logarithmisch) über SOR–Parameter ω für das „lid-driven cavity"–Problem auf regulärem Gitter, einmal für `d_max=3`, einmal für `d_max=4`.



Verknüpfungen zu den Nachbarelementen nicht explizit vorgesehen. Deshalb ist die Parallelisierung hier nicht so effizient möglich.

Im Programm erfolgt die Abarbeitung einer Iteration deshalb seriell, wobei nacheinander alle Komponenten von $\mathbf{x}^{(k+1)}$, die auf den Prozessen verteilt sind, berechnet werden. Dabei sind jedes Mal Kommunikationsschritte nötig, da die Berechnung eines Vektoreintrags auch von Werten abhängig sein kann, die auf anderen Prozessen vorliegen.

Als Ausweg lässt sich das CG–Verfahren verwenden, das eine bessere Parallelisierung ermöglicht.

CG–Verfahren

Der Algorithmus für das konjugierte–Gradienten–Verfahren ist in Algorithmus 3.3 angegeben. Jede Anweisung wird von den Prozessen auf den Variablen des ihnen zugeordneten Teilgebiets durchgeführt. Nur bei der Matrix–Vektor–Multiplikation und den 3 markierten Vektor–Vektor–Multiplikation sind Kommunikationsschritte notwendig, wobei erstere zu Beginn des Kapitels beschrieben wurde. Bei den Vektor–Vektor–Multiplikationen müssen jeweils nur die lokalen Ergebnisse parallel aufsummiert und diese Werte anschließend über alle Prozesse zusammengezählt werden. Dies wird mit der MPI–Routine `MPI_Allreduce(...,MPI_SUM,...)` bewerkstelligt.

Unter anderem wegen der guten Parallelisierbarkeit ist dieser Algorithmus bei paralleler Ausführung im Allgemeinen schneller und dem SOR–Verfahren vorzuziehen.

Anders als beim SOR–Verfahren können hier Dirichletrandbedingungen nicht durch Festsetzen der Druckwerte in jeder Iteration erzwungen werden. Stattdessen wird im Programm die rechte Seite angepasst. Formal lassen sich die Systemmatrix L und der Druckvektor p in zwei Teile aufteilen, von denen der

Code–Abschnitt 3.3 CG–Verfahren für das System $L\,p = rhs$

```
p = 0
r = rhs - L*p
s = r
rtr = r^T * r                # Vektor-Vektor-Multiplikation

while rtr < epsilon do
    ls = L * s               # Matrix-Vektor-Multiplikation
    sls = s^T * ls           # Vektor-Vektor-Multiplikation
    alpha = rtr / sls

    p = p + alpha * s
    r = r - alpha * ls
    rtr_new = r^T * r        # Vektor-Vektor-Multiplikation

    beta = rtr_new / rtr
    rtr = rtr_new

    s = r + beta * s
end
```

zweite gerade die Elemente mit Dirichletrandbedingung enthält. Dabei stehen die Indices u und c für *unconstrained* und *constrained*.

$$\begin{aligned} & L\,\mathbf{p} = \mathbf{b} \\ \Leftrightarrow \quad & \begin{pmatrix} L_{uu} & L_{uc} \\ L_{cu} & L_{cc} \end{pmatrix} \begin{pmatrix} \mathbf{p}_u \\ \mathbf{p}_c \end{pmatrix} = \begin{pmatrix} \mathbf{b}_u \\ \mathbf{b}_c \end{pmatrix}. \end{aligned} \tag{3.4}$$

Für die eingeschränkten Variablen gilt eine Randbedingung in der Form $\mathbf{p}_c = \tilde{\mathbf{p}}$. Der erste Block der Gleichung ergibt nach Umformung:

$$L_{uu}\,\mathbf{p}_u = \mathbf{b}_u - L_{uc}\,\mathbf{p}_c = \mathbf{b}_u - L_{uc}\,\tilde{\mathbf{p}}.$$

Dies ergibt ein Gleichungssystem mit modifizierter rechter Seite, das nun mit dem CG–Verfahren gelöst werden kann. Damit auch die zweite Block–Zeile der Gleichung (3.4) gilt, muss die Randbedingung entsprechend gewählt sein. Dies wird durch das Programm nicht überprüft.

Direkte Löser

Grundlage der direkten Löser ist eine Darstellung der Systemmatrix $A \in \mathbb{R}^{n\times n}$ als Matrizenprodukt zweier Dreiecksmatrizen. Die Erzeugung dieser Darstellung geschieht mit einer Zeitkomplexität von $\mathcal{O}(n^3)$ und muss jeweils nur durchgeführt werden, nachdem das Finite–Elemente–Gitter neu erzeugt wurde. Dies trifft zu Beginn der Simulation zu, ebenso wie nach Verfeinerungen und Vergröberungen der Gitters bei eingeschalteter dynamischer Adaptivität. Da die Systemmatrix während der Zeit zwischen Gitterverfeinerungen konstant bleibt, kann dann in jedem Zeitschritt das Gleichungssystem mithilfe der Darstellung in $\mathcal{O}(n^2)$ gelöst werden.

Bei der LU–Zerlegung mit Pivotisierung geschieht die Darstellung der Matrix als $A = P^\top\,L\,U$. Dabei ist L eine untere Dreiecksmatrix mit Einsen auf der Diagonale. U ist eine obere Dreiecksmatrix. Um die Stabilität des Verfahrens zu verbessern, wird eine Permutationsmatrix P verwendet. Da diese orthogonal ist, lässt sich das Zerlegungsproblem als $P\,A = L\,U$ schreiben, wobei die Vormultiplikation von P an die Systemmatrix in dieser Zeilenvertauschungen bewirkt. Die Permutationsmatrix wird so gewählt, dass während des Algorithmus der LU–Zerlegung die Pivotelemente, durch die diviert wird, möglichst große Werte annehmen.

Wenn die Matrizen erzeugt sind, kann das Gleichungsystem $A\,\mathbf{x} = \mathbf{b}$ in zwei Schritten gelöst werden. Zuerst ermittelt man aus $L\,\mathbf{y} = P\,\mathbf{b}$ den Hilfsvektor $\mathbf{y}$ durch Vorwärtseinsetzen. Dann erhält man die Lösung $\mathbf{x}$ aus $U\,\mathbf{x} = \mathbf{y}$ durch Rückwärtseinsetzen.

Um mit möglichst wenig Speicheraufwand auszukommen, wird in der Implementierung eine $n \times n$–Datenstruktur verwendet, um die Matrizen L und U zu speichern. Die Permutationsmatrix wird als Vektor π abgespeichert, der die Matrix angewandt auf den Vektor $(1,2,3,\ldots,n)^\top$ enthält. Dann entspricht der Eintrag (i,j) der permutierten Matrix $P\,A$ dem Eintrag $(\pi(i),j)$ der Matrix A. So kann vermieden werden, dass ganze Zeilen von A vertauscht, d.h. umkopiert werden müssen.

Eine ungefähre Halbierung dieses Speicherbedarfs ist mit der *Cholesky*–Zerlegung möglich. Symmetrisch positiv definite Matrizen A lassen sich als $A = L\,L^\top$ darstellen, wobei L eine reelle untere Dreiecksmatrix ist. Der Vorteil ist, dass hierbei nur eine Dreiecksmatrix gespeichert werden muss. Nach Erlangung der Zerlegung folgt das Lösen wieder dem gleichen Schema mit Vorwärts- und Rückwärtseinsetzen, wie bei der LU–Zerlegung.

Für die betrachteten Strömungsprobleme ergeben sich aus der Finite–Elemente–Methode immer symmetrische Steifigkeitsmatrizen. Die positive Definitheit ist jedoch bei beliebigen adaptiven Gittern mit Randbedingungen nicht immer gegeben. In einem solchen Fall lässt sich die Cholesky–Zerlegung trotzdem anwenden, wobei dann rein imaginäre Werte auf der Diagonale von L auftreten können. Dementsprechend müssen auch beim Vorwärts- und Rückwärtseinsetzen und bei der Speicherung von $\mathbf{y}$ komplexe Werte ermöglicht werden. Ist die Matrix regulär, erhält man wieder eine Lösung $\mathbf{x}$ mit reellen Werten.

Neben der hohen Zeitkomplexität beim Erstellen der Zerlegung haben diese direkten Lösungsverfahren den Nachteil, dass die Systemmatrix erst aus dem adaptiven Gitter extrahiert werden muss und als vollbesetzte Matrix abgespeichert werden muss. Zudem ist eine Parallelisierung nicht effizient. Im Programm können diese Verfahren deshalb nur bei serieller Ausführung verwendet werden. Sie dienen hauptsächlich der Validierung der iterativen Verfahren. Es hat sich gezeigt, dass sie für beinahe singuläre Probleme teilweise bessere Lösungen liefern. Außerdem kann mithilfe der Cholesky–Zerlegung ermittelt werden, ob die Systemmatrix positiv definit bzw. überhaupt regulär ist.

3.2.4. Adaptive Verfeinerung des Finite–Elemente–Gitters

Nachdem der Lösungsschritt in der Simulationsschleife in Algorithmus 3.1 erfolgt ist, können die neuen Geschwindigkeiten nach Gleichung 2.15 berechnet werden. Anschließend wird das Maximum u_{max} der absoluten Geschwindigkeitswerte an jedem Knoten bestimmt. Die neue Zeitschrittweite δt wird dann so gewählt, dass $u_{max}\,\delta t < \alpha\,h$ ist, wobei h die kleinste vorkommende Gitterweite ist. Der Sicherheitsfaktor α kann durch den Parameter `delta_t_safety_factor` festgelegt werden.

Der nächste Schritt ist die Berechnung der Vorticity w in jedem Element. Diese ist definiert als

$$w = \frac{\partial \mathbf{v}}{\partial x_1} - \frac{\partial \mathbf{u}}{\partial x_2}$$

und ist ein Maß für die Wirbelstärke. Ein hoher positiver Wert steht dabei für eine starke Rotationsbewegung entgegen dem Uhrzeigersinn. Der Wert für ein Element wird erhalten, indem die Ableitungen als Differenzenquotienten der Knotenwerte approximiert werden. Die genaue Formel ist in Algorithmus 3.4 angegeben. Die Variablen `u(el,1)` bis `u(el,4)` bezeichnen die Geschwindigkeitswerte, die sich auf die Eckpunkte der quadratischen Elemente beziehen mit der Nummierung wie in Abbildung 2.12. Dafür muss zuvor bei hängenden Knoten noch die entsprechende Interpolation ausgeführt werden, um die Werte an der richtigen Position zu erhalten.

Code–Abschnitt 3.4 Berechnung der Vorticity

```
for Element el do
    w(el) = 1/(2*h) * (v(el,4)-v(el,3)+v(el,2)-v(el,1))
            - 1/(2*h) * (u(el,3)-u(el,1)+u(el,4)-u(el,2))
end
```

Falls die entsprechende Option aktiviert ist, verfeinert oder vergröbert das Programm das Rechengitter in bestimmten Intervallen (alle `refine_interval` Zeitschritte). Dies geschieht, indem eine neues Finite–Elemente–Gitter erzeugt wird, das fortan für die Berechnung verwendet wird. Ein Element wird übernommen, wenn sich seine absolute Vorticity zwischen den beiden Schwellwerten `vorticity_min_threshold` und `vorticity_max_threshold` befindet. Ist sie höher, befindet sich an der Stelle des Elements ein Wirbel, der so interessant ist, dass es sinnvoll ist, das Finite–Elemente–Gitter hier zu verfeinern. Im neuen Gitter wird das Element deshalb durch die 4 Kindelemente ersetzt. Ist die Vorticity in 4 Elementen innerhalb des Rechengebiets, die zueinander Geschwister sind, jeweils unter dem Schwellwert, so werden diese 4 Elemente im neuen Gitter durch ihr Vaterelement ersetzt. So werden, abhängig von der Vorticity, an den Positionen der Elemente entsprechende Eingabeelemente generiert, mit deren Hilfe das in Kapitel 2.2.3 vorgestellte Verfahren das neue Gitter erzeugt. So ist sichergestellt, dass das neue Gitter auch wieder die vorgestellte Lastbalancierung besitzt. Der Nachteil daran ist, dass nun der Prozess, der ein Element besitzt, wechseln kann, was bedeutet, dass die entsprechenden Werte vom bisherigen Prozess zum neuen Prozess kommuniziert werden müssen.

Für die Elemente, für die der Prozess gleich bleibt, ist das Kopieren der Werte vom alten zum neuen Gitter in linearer Zeitkomplexität möglich. Dazu werden 2 nebeneinander durchlaufene Schleifen verwendet. Es wird zum einen über den linear abgespeicherten neuen Quadtree der Elemente iteriert. Ebenso wird auch über den alten Quadtree der Elemente iteriert. Für Element, die sowohl im alten als auch im neuen Quadtree vorkommen, werden die Druck- und Geschwindigkeitswerte kopiert. Falls ein neues Element aus seinen 4 Kindern hervorgeht, bekommt es den Mittelwert der 4 bisherigen Druckwerte. Handelt es sich um ein verfeinertes Element, bekommt es den Wert des Vaterelements im alten Gitter.

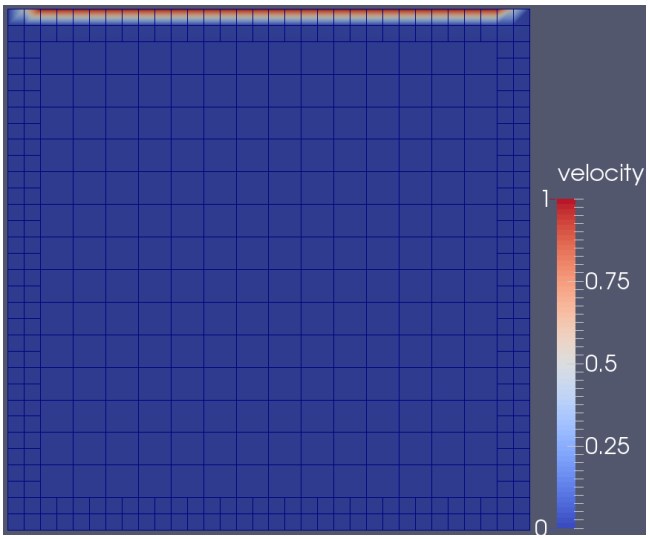
Auch die Zuweisung der Werte an den Knoten kann in linearer Zeit erfolgen. Dafür werden die selben beiden Schleifen wie für die Elementwerte verwendet. Es wird jeweils der Wert des neuen Knotens am Ankerpunkt des neuen Elements nachgeschlagen. Falls sich an der Position im alten Gitter kein Knoten befand, muss der Wert von den umliegenden Knoten bilinear interpoliert werden. Gelangt man zu einem Element, dass rechts von sich keine weiteren Elemente hat, so wird auch der Knoten, der sich an Knotenposition 1 des Elements (also rechts unten) befindet, mit seinem Wert belegt. Diese Werte sind dem Referenzelement im alten Gitter bekannt. Analog behandeln Elemente, die keine weiteren Elemente mehr über sich und rechts von sich haben, die Knoten an den Positionen 3 und 4. Durch dieses Vorgehen können fast alle Knotenwerte aus dem alten Gitter in das neue gebracht werden. An Ecken des Gebiets ist es manchmal nicht möglich, einen Knotenwert nur von dem aktuellen Element der Schleife über die alten Elemente zu erhalten. Dann muss in logarithmischer Zeit eine Suche nach dem Knoten an der entsprechenden Stelle stattfinden.

In Abbildung 3.9 ist das Resultat des „driven cavity“–Beispiels dargestellt. Eine quadratischen Geometrie mit Seitenlänge 1 erhält unten und an den Seiten *noslip*–Ränder. Wie bereits beschrieben, gilt am oberen Rand eine *inflow*–Randbedingung mit $(u, v) = (1,0)$. Das Fluid wird also am oberen Rand nach rechts bewegt. Infolge dessen erhält man im Gebiet einen großen Wirbel, der sich im Uhrzeigersinn bewegt. Die Reynoldszahl beträgt $Re = 1000$, als Löser wurde das CG–Verfahren mit $\varepsilon = 10^{-15}$ gewählt. `d_max` wurde auf 5 gesetzt. Es sind also maximal $2^5 = 32$ Zellen pro Richtung möglich. Mithilfe von `inverse_sample_width_border = 32` und `inverse_sample_width_interior = 16` wurde die in Abbildung 3.9(a) dargestellte Ausgangskonfiguration erreicht, bei der der Rand mit jeweils 32 Elementen und das Innere mit doppelt so großen Element diskretisiert

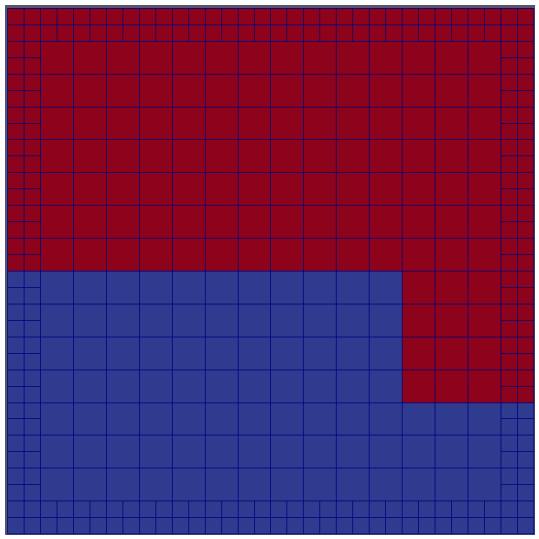
wird. Das Verfeinerungskriterium wird alle 5 Zeitschritte überprüft, die Toleranzgrenzen für die Vorticity betragen $0{,}01$ und $0{,}1$.

In Abbildung 3.9(a) ist der Ausgangszustand zu sehen. Nur am oberen Rand, wo die *inflow*–Randbedingung gesetzt ist, sind die absoluten Werte der Geschwindigkeit ungleich Null. In Abbildung 3.9(b) ist die dazugehörige Verteilung der Elemente auf 2 Prozesse dargestellt. Zum Zeitpunkt $t = 5$ ergeben sich die beiden Bilder 3.9(c) für die absolute Geschwindigkeit und 3.9(d) für die Prozessaufteilung. Am rechten Rand beginnt das Fluid abwärts zu fließen, was in einer dort erkennbar höheren Geschwindigkeit resultiert. In der Folge ist dort die Vorticity höher, sodass das Gitter in der Umgebung verfeinert wurde. Das aktuelle Gitter ist außerdem, wie in Abbildung 3.9(d) zu sehen, anders auf die Prozesse verteilt. Der Bereich rechts oben enthält eine höhere Elementdichte, obwohl er nur ein Viertel des Gebiets einnimmt. Er ist Prozess 1 zugeteilt. Zum Zeitpunkt $t = 19{,}18$ hat sich ein stationärer Zustand eingestellt. Man erkennt in Abbildung 3.9(f) den Wirbel, der sich über das gesamte Gebiet erstreckt. Besonders im unteren Bereich wurden gröbere Elemente erzeugt, eine Häufung von feinen Elementen befindet sich am oberen Rand sowie am oberen linken und rechten Rand. Die dort befindlichen Stellen mit höherer Vorticity werden somit feiner aufgelöst. Bei einer feineren Diskretisierung in der linken unteren und rechten unteren Ecke würde man dort auch noch jeweils einen Wirbel in die Gegenrichtung erkennen können. Diese Bereiche werden durch die aktuelle Einstellung nicht verfeinert. Durch die dort befindlichen groben Elemente beginnen diese Wirbel nicht zu entstehen, was wiederum die Vorticity nicht erhöht und nicht zur Verfeinerung führt. Dieses Problem ließe sich umgehen, indem das gröbstmögliche Elementlevel auf einen feineren Wert eingestellt wird, sodass es zum Ansatz dieser Wirbel ausreicht.

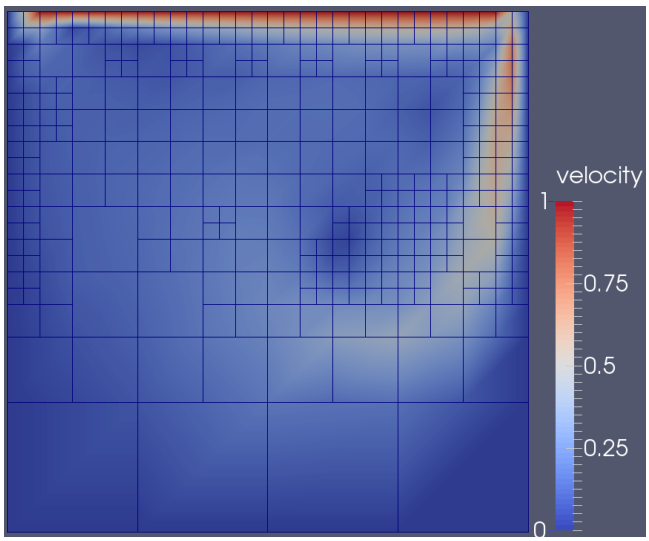
Abbildung 3.9. Beispiel „driven cavity“ mit adaptiver Verfeinerung, absoluter Wert der Geschwindigkeit und Gitter mit Aufteilung auf die 2 Prozesse zu verschiedenen Zeitpunkten



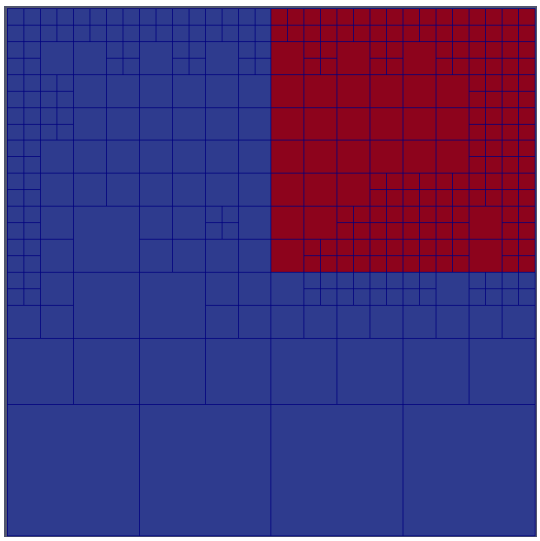
(a) Geschwindigkeit für $t = 0$



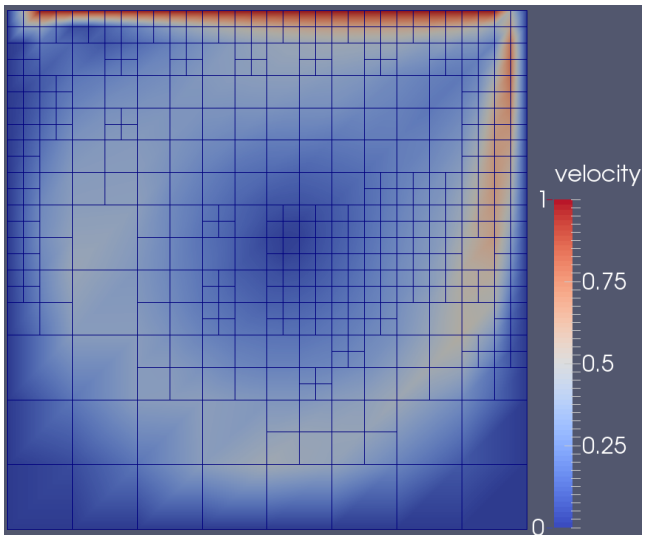
(b) Partionierung für $t = 0$



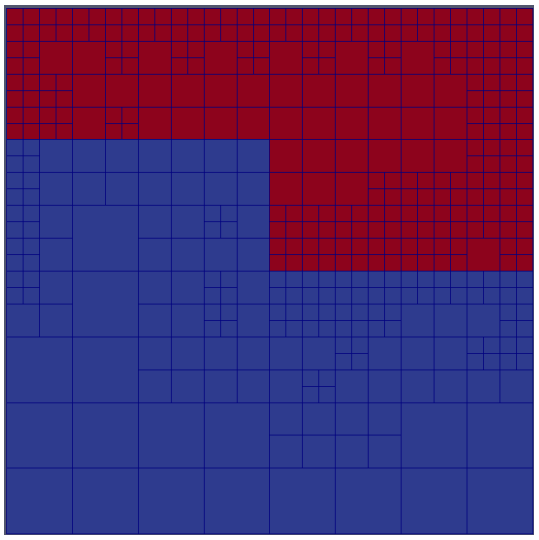
(c) Geschwindigkeit für $t = 5$



(d) Partionierung für $t = 5$



(e) Geschwindigkeit für $t = 19{,}18$



(f) Partionierung für $t = 19{,}18$

4. Beispiele und Auswertung

4.1. Validierung

Die Validierung wird ausgeführt, um zu überprüfen, ob die Simulationsergebnisse die Realität gut abbilden und somit verwertbare Aussagen liefern können. Dabei vergleicht man das Simulationsresultat mit Experimenten, die einfach durchzuführen sind und bei denen man die simulierten Größen messen kann. Eine dabei auftretende Abweichung kann mehrere Gründe haben. Zuerst basiert das Modell der Strömung auf Annahmen, wie der Inkompressibilität und dass es sich um ein Newtonsches Fluid mit einer bestimmten Reynoldszahl handelt. Auch die Randbedingungen beinhalten Annahmen über das Verhalten des Fluids am Rand des Gebiets. Z.B. bildet eine *noslip*–Randbedingung den Fall ab, dass Fluidteilchen an der entsprechenden Stelle an der Wand haften bleiben. Diese Annahmen können bei dem Experiment in der Realität verletzt sein. Daraus ergibt sich der Modellfehler. Doch auch wenn dieser gering ist, können weitere Fehler das Endergebnis verfälschen. Durch die Diskretisierung und das numerische Verfahren ergeben sich Diskretisierungs- und Approximationsfehler, außerdem treten durch die beschränkte Maschinengenauigkeit während des Verfahrens Rundungsfehler auf. Im Folgenden wird an einem Beispiel untersucht, inwieweit die Fehler zwischen dem Modell und dem Endergebnis große Abweichungen ergeben.

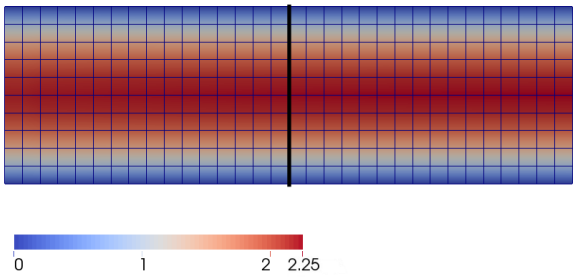
Für die laminare Strömung durch ein gerades Rohr aufgrund eines Druckunterschieds haben Experimente eine gute Übereinstimmung mit dem Modell, d.h. der Beschreibung durch die Navier–Stokes–Gleichungen, ergeben. Bei diesem auch als *Hagen–Poiseuille*–Fluss bezeichneten Experiment werden an den Ränder des Rohres *noslip*–Randbedingungen vorgegeben. Das zweidimensionale Rohr hat die Abmessungen $L_1 \times L_2$. Am linken Ende wird der Druck auf den Wert p_0, am rechten Ende auf p_1 festgesetzt, sodass entlang des Rohres eine Druckdifferenz $\Delta p = p_1 - p_0$ auftritt. Für diese Problemdefinition existiert eine analytische Lösung der Navier–Stokes–Gleichungen:

$$\begin{aligned} u(x_1, x_2) &= \frac{Re\,\Delta p}{2\,L_1}\, x_2\,(x_2 - L_2), \\ v(x_1, x_2) &= 0, \\ p(x_1, x_2) &= p_0 + \frac{x_1}{L_1}\,\Delta p \end{aligned} \tag{4.1}$$

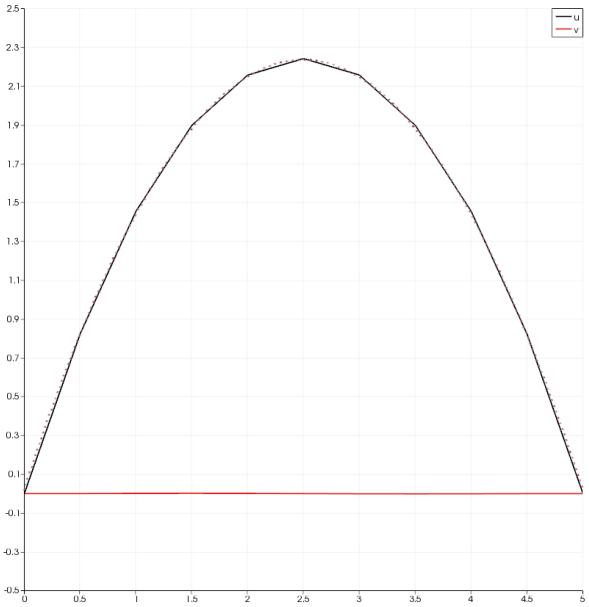
Es lässt sich leicht verifizieren, dass damit in der Impulsgleichung (2.3a) der Konvektionsterm und die Ableitungen der Geschwindigkeit in x_1–Richtung verschwinden. Der Term $1/Re\, u_{x_2x_2}$ entspricht gerade der negativen Ableitung des Drucks in x_1–Richtung, sodass die rechte Seite den Wert Null ergibt und damit einen stationären Zustand beschreibt. Auch die Kontinuitätsgleichung ist erfüllt.

In der Simulation kann entweder mit dem gesamten Gebiet oder, unter Ausnutzung der Symmetrie, nur mit der oberen Hälfte gerechnet werden. Abbildung 4.11 zeigt die Simulation einer vollständigen Poiseuilleströmung der Reynoldszahl $Re = 200$ in einem Feld mit den Abmessungen $L_1 = 1, L_2 = 0{,}3$. Die Druckdifferenz beträgt $\Delta p = 1$, sodass sich das Maximum der analytischen Lösung (4.1) auf $u(x_1, L_2/2) = 2{,}25$ beläuft. Das Gebiet wurde mit 32×10 Elementen diskretisiert. In Abbildung 4.1(b) ist der sich aus der Simulation ergebende Geschwindigkeitsverlauf entlang eines vertikalen Schnitts dargestellt. Zum Vergleich ist die parabelförmige analytische Lösung für $u(x_2)$ aufgetragen. Man erkennt, dass die Lösung

Abbildung 4.1. Stationäres Geschwindigkeitsfeld der Poiseuille–Strömung für $Re = 200$, Abmessungen $1 \times 0{,}3$, `d_max=5`



(a) Farbliche Kodierung des Betrags der Geschwindigkeit



(b) Verlauf entlang des schwarzen vertikalen Schnitts in (a), schwarze Kurve: $u(x_2)$, rot: $v(x_2)$, die Referenzkurve für $u(x_2)$ ist als gepunktete Linie eingezeichnet und liegt in etwa auf der schwarzen.

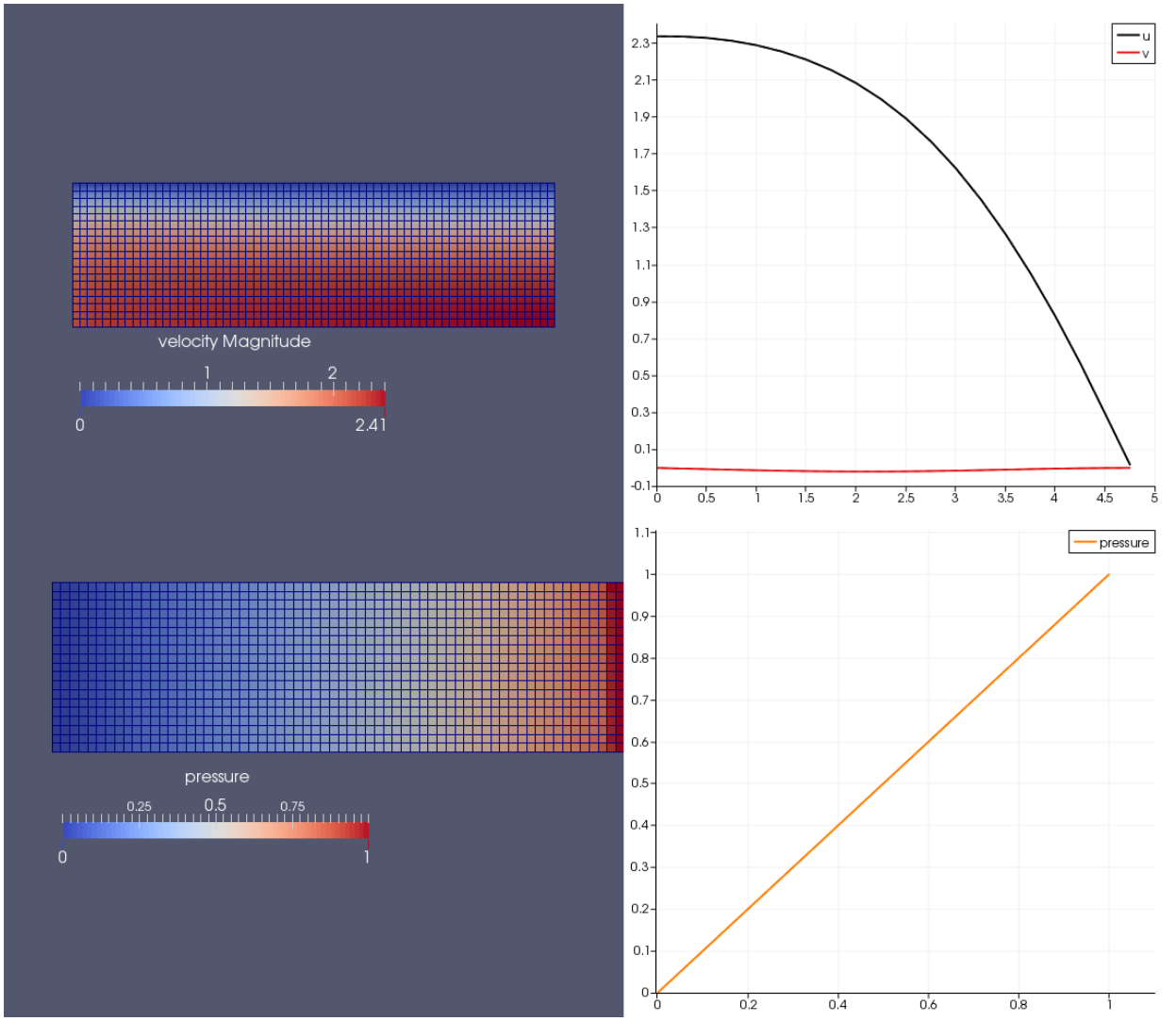
der Simulation sehr nahe an dieser Kurve verläuft. Auch die Geschwindigkeitskomponente v ist sehr nahe bei Null, was der analytischen Lösung entspricht. Das Simulationsergebnis stimmt also gut mit der analytischen Lösung überein.

In Abbildung 4.2 ist die Simulation für $t = 10$ eines halben Hagen–Poiseuille–Flusses dargestellt. Hier wurde die Randbedingung am unteren Rand auf *slip* gesetzt. Wie im Diagramm dargestellt, ergibt sich hierfür eine halbe Parabel für die Geschwindigkeit u entlang eines Schnitts in x_2–Richtung. Der Druck steigt in x_1–Richtung linear an, was mit der analytischen Lösung übereinstimmt.

In Abbildung 4.3 wurde erneut ein Hagen–Poiseuille–Fluss im kompletten Rohr simuliert. Bei der Diskretisierung wurden zwei unterschiedliche Elementgrößen verwendet, sodass auch hängende Knoten entstehen. Auch für dieses Beispiel zeigen die Diagramme wieder eine parabelförmige Geschwindigkeit $u(x_2)$ und für den Druckverlauf eine Gerade. Für das Hagen–Poiseuille–Flussproblem kann die Simulation also die Werte der analytischen Lösung gut reproduzieren.

Für das in Kapitel 3.1.3 eingeführte Beispiel „driven cavity“ lassen sich ebenfalls Versuche durchführen. Ein Bild einer Versuchsdurchführung mit einer nicht quadratischen Geometrie ist in Abbildung 4.4 zu sehen. Anhand der Bewegung der in der Flüssigkeit befindlichen Partikel lässt sich auf das Geschwindigkeitsfeld schließen. Bei einer Reynoldszahl von 1000 ergibt sich dabei ein großer Wirbel, der das gesamte Gebiet ausfüllt, sowie zwei kleinere Wirbel in die Gegenrichtung an den unteren Ecken. Das Simulationsergebnis ist in Abbildung 4.5 dargestellt. Es wurde eine feine Diskretisierung von 128×128 Elementen gewählt. Zur Verdeutlichung der Wirbel wurden außerdem *Streamlines* eingezeichnet. Dies sind Linien, die an jeder Stelle in Richtung des Geschwindigkeitsfelds verlaufen. Für den dargestellten Zeitpunkt $t = 20$ hat sich ein stationärer Zustand eingestellt, sodass die Streamlines den Pfaden entsprechen, die in der Flüssigkeit befindliche Partikel beschreiben würden. Auch hier liefert der Vergleich mit Experimenten gute Übereinstimmung.

Abbildung 4.2. Halbe Poiseuille–Strömung für $Re = 100$, $\Delta p = 1$, Abmessungen $1 \times 0{,}3$, `d_max=6`. Links oben: farbliche Kodierung des Betrags der Geschwindigkeit, Links unten: farbliche Kodierung des Drucks, rechts oben: Verlauf der Geschwindigkeitskomponenten entlang eines vertikalen Schnitts, schwarz: u, rot: v, rechts unten: Verlauf des Drucks entlang eines horizontalen Schnitts

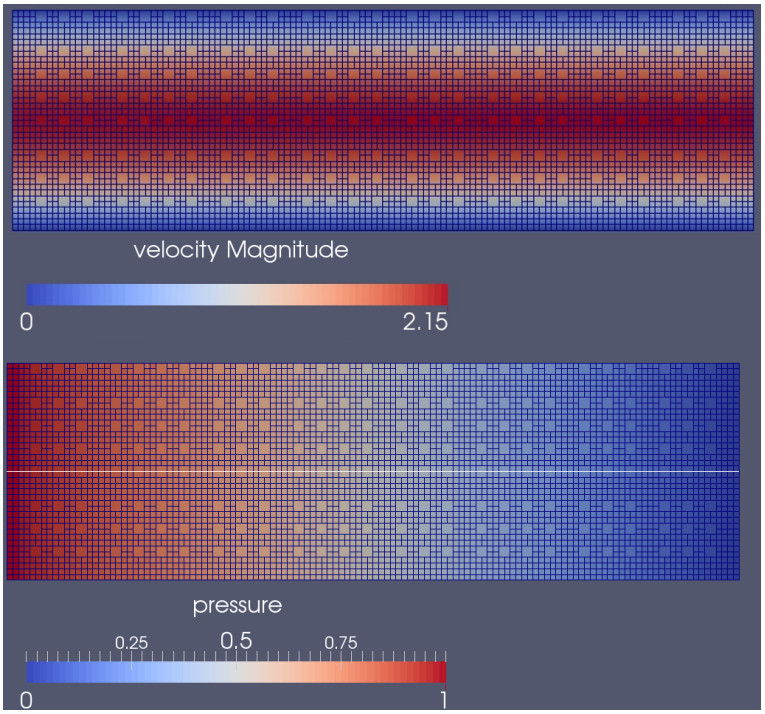


4.2. Laufzeituntersuchungen

4.2.1. Parallele Skalierbarkeit und Lastbalancierung

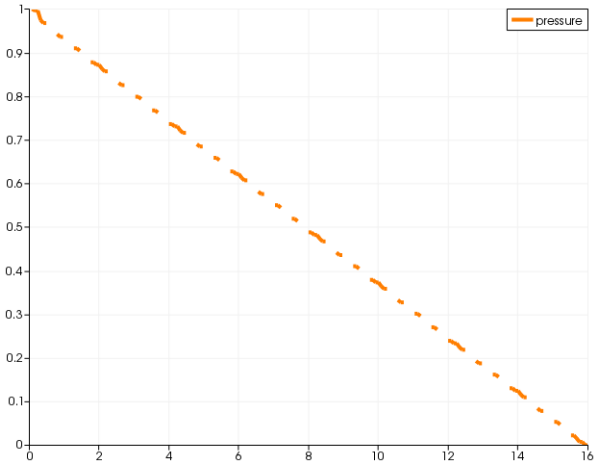
Wenn ein System mit mehreren Prozessoren zur Ausführung des Programms zur Verfügung steht, ist die Veränderung der Laufzeit bei Verwendung von mehr Prozessen von Interesse. Dies wird als starke Skalierung bezeichnet. Als Testfall wurde das in Abbildung 4.5 gezeigte „driven cavity“–Szenario bis zur Zeit $t = 2$ berechnet. Das betrachtete Problem wurde mit $2^{14} = 16384$ Elementen diskretisiert. Dabei ergeben sich 16641 Knoten. Als Lösungsverfahren wurde das CG–Verfahren verwendet. Das verwendete System enthält 12 Intel® Xeon® E5-2620 Prozessoren, wobei durch Hyperthreading 24 CPUs verwendbar sind. Die Größe des Hauptspeichers beträgt ca. 200GB mit L1/L2/L3–Cachegrößen von 32KB/256KB/15360KB. Die gesamte Ausführungsdauer vom Einlesen der Parameter bis zur Beendigung der Simulation wurde für verschiedene Prozessanzahlen gemessen und ist in Abbildung 4.6(a) über die Anzahl der verwendeten Prozesse aufgetragen. Die Dauer für den Aufbau der Finite–Elemente–Datenstruktur, die sich im Wesentli-

Abbildung 4.3. Stationäre Poiseuille–Strömung für $Re = 200$, `d_max=7`, `d_min=6`, Abmessungen $1 \times 0{,}3$, Druckrandbedingungen: $p_0 = 1$, $p_1 = 0$



(a) Oben: Farbliche Kodierung des Betrags der Geschwindigkeit, unten: Farbliche Kodierung des Drucks

(b) Geschwindigkeit entlang des vertikalen Schnitts, schwarz: $u(x_2)$, rot: $v(x_2)$



(c) Druck entlang des horizontalen Schnitts

chen aus der Laufzeit der Generierung der Quadtrees für die Elemente und Knoten sowie anschließend dem Finden der Zuordnung von Knoten zu Elementen zusammensetzt, beträgt dabei bei einem Prozess etwa eine Sekunde, bei mindestens zwei Prozessen weniger als eine halbe Sekunde. Im Vergleich zu der Berechungsdauer ist dies vernachlässigbar kurz.

Man erkennt, dass die Ausführungsdauer mit Erhöhung der Prozessanzahl zunächst abnimmt. Ab etwa 8 Prozessen erreicht die Kurve einen Sättingswert. Zur genaueren Untersuchung wird der *parallele Speedup* $S_p = T(1)/T(p)$ definiert, der die Laufzeit mit p Prozessoren $T(p)$ ins Verhältnis zur seriellen Laufzeit $T(1)$ setzt. Er ist als blaue Kurve im Diagramm 4.6(b) dargestellt. Hier erkennt man zunächst einen etwa

Abbildung 4.4. Experiment zu „driven cavity“

Bildquelle: http://www5.in.tum.de/forschung/simlab/course2010_files/course_mat/20101004_Benk_Belgrad_CFD_rework.pdf, S.19

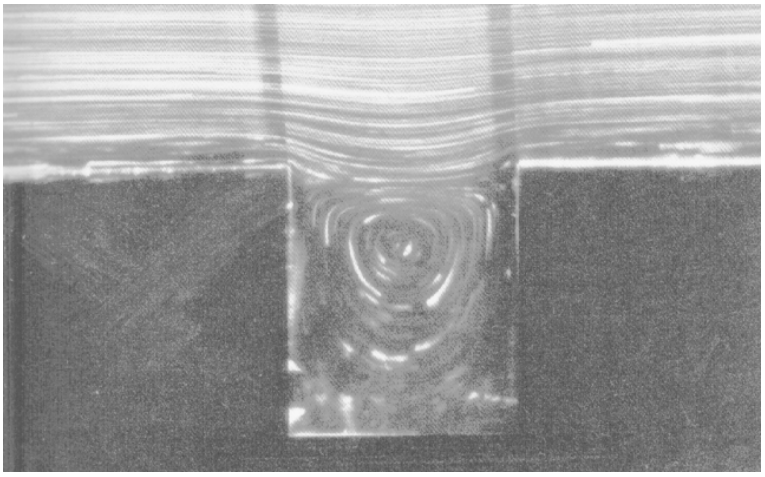
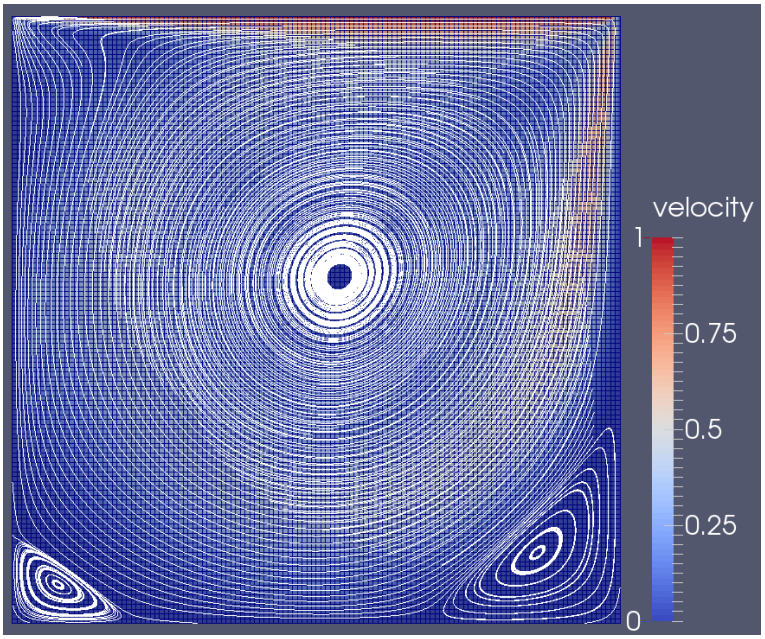


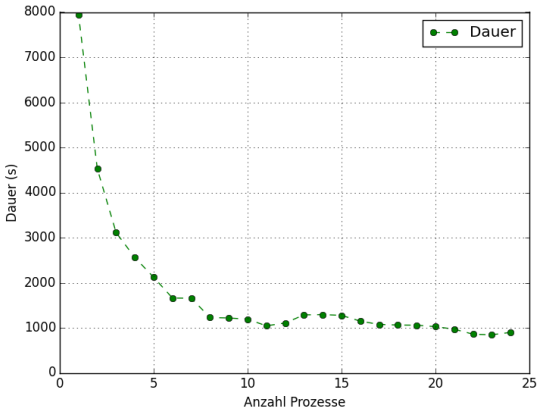
Abbildung 4.5. Geschwindigkeit und streamlines für „driven cavity“, $t = 20, Re = 1000, L_1 = L_2 = 1$, d_max=7



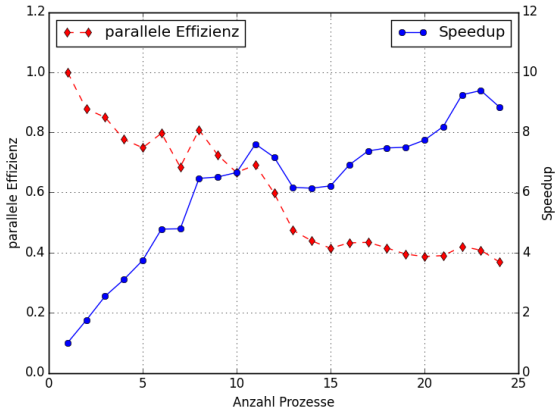
linearen Anstieg, der sich anschließend abschwächt. Der maximale Wert $S_p = 9{,}4$ wird für $p = 23$ Prozesse erreicht. Für Prozessanzahlen in dieser Größenordnung liefert die Hinzunahme weiterer Prozesse jedoch nur geringe Auswirkungen auf die Gesamtlaufzeit, weshalb der Nutzen in der Praxis fraglich ist. Um dies zu quantifizieren, wird die *parallele Effizienz* $E_p = S_p/p$ angegeben, die den parallelen Speedup durch die Anzahl Prozesse p teilt. Im Diagramm 4.6(b) ist diese als rote Kurve eingetragen. Man erkennt die fallende Tendenz mit Erreichen eines Sättigungswertes von etwa 40% für $p \geq 14$.

Das Erreichen des Sättingswertes ist darauf zurückzuführen, dass für viele Prozesse der Kommunikationsaufwand ansteigt. Außerdem nimmt gleichzeitig die Größe des Teilgebietes, die ein Prozess bearbeitet, ab. Somit wird die Einsparung an Rechenzeit durch die erhöhte Dauer für die Kommunikation kompensiert. Der Umfang des Kommunikationsaufwands während der Berechnung hängt direkt von der Anzahl an Randknoten ab, die ein Prozess besitzt. Wie in Abbildung 4.7(a) zu sehen ist, steigt der Anteil an Randknoten mit zunehmender Prozessanzahl, da dann die Anzahl an Knoten des Teilgebiets sinkt.

Abbildung 4.6. Untersuchung der starken Skalierung



(a) Starke Skalierung: Laufzeit über Anzahl Prozesse



(b) Speedup und parallele Effizienz

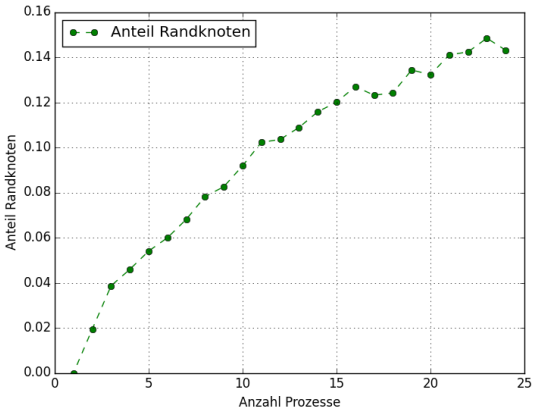
Es stellt sich außerdem die Frage, inwieweit sich die Heuristik der Lastbalancierung auf die Skalierbarkeit auswirkt. Die Aufteilung der Elemente auf die Prozesse erfolgt wie in Kapitel 2.2.3 beschrieben, wobei die Anzahl der Elemente, die ein Prozess besitzt, von der optimalen Anzahl einer gleichmäßigen Verteilung der Elemente abweichen kann. Der versprochene Nutzen liegt in der Verringerung der Grenzen zwischen den Gebieten mehrerer Prozesse bei gleichzeitiger Aufrechterhaltung einer genügend ausgeglichenen Elementaufteilung.

Zur Untersuchung wurde zunächst die Abweichung der Elementanzahl vom Optimum für jeden Prozess bestimmt und so normiert, dass eine Abweichung von -1 bedeutet, dass der Prozess gar keine Elemente zugeteilt bekommt und $+1$ bedeutet, dass er doppelt so viele Elemente, wie bei gleichmäßiger Verteilung, erhält. Durch Quadrierung erhält man die Varianz, die anschließend über alle Prozesse gemittelt wird. Nach Ziehen der Wurzel ergibt sich die durch die grüne Kurve in Abbildung 4.7(b) dargestellte Standardabweichung. Das jeweilige Maximum über alle Prozesse ist in rot eingezeichnet. Demnach schwankt die durchschnittliche prozentuale Abweichung von der optimalen Verteilung. Die durchschnittliche Abweichung beträgt dabei immer zwischen 5% und 20%, wobei einzelne Prozesse auch Abweichungen bis über 30% besitzen. Es ist außerdem kein Trend über die Erhöhung der Prozessanzahl zu bemerken. Daraus lässt sich schließen, dass die Heuristik auch für größere Prozessanzahlen gleich gut funktioniert. Außerdem erhält man einen Eindruck, wie sehr die Heuristik von der gleichmäßigen Elementverteilung abweicht.

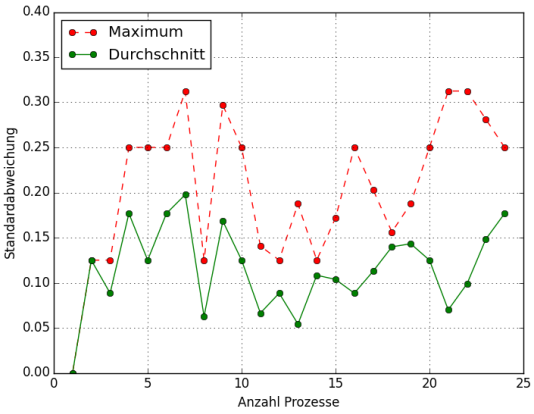
4.2.2. Laufzeit der Gittergenerierung

Als nächstes wird die Laufzeit für die Erzeugung der Finite–Elemente–Datenstruktur untersucht. Den Testfall bildet die Generierung für die in Abbildung 4.8 angegeben Geometrie (das Signet, das Teil des Logos der Universität Stuttgart ist), die sich aus vielen kleinen rautenförmigen Merkmalen zusammensetzt und somit zu einem an diesen Stellen entsprechend feinen Gitter führt. Die relevanten Parameter `d_max` und `inverse_sample_width_border` werden für den Testlauf entsprechend erhöht, um ein immer feineres Finite–Elemente–Gitter zu erzeugen. Dabei wird jeweils ein Prozess verwendet. Das Ergebnis ist in Abbildung 4.9 dargestellt. In der doppelt–logarithmischen Darstellung 4.9(a) ist eine Gerade mit Steigung 1 zu erkennen, was einem linearem Verlauf entspräche. Bei genauerer Betrachtung im nicht–logarithmischen Diagramm 4.9(b) erkennt man, dass die rote Kurve der Laufzeit besser mit der orangenen Referenzkurve einer $n\ \log(n)$–Funktion übereinstimmt, als mit dem blau dargestellten linearen Verlauf. Dies passt zu der Zeitkomplexität des Algorithmus von $\mathcal{O}(n\ \log(n))$, der aus der benötigten in logarithmischer Zeit ablaufenden Suche bei

Abbildung 4.7. Lastbalancierung

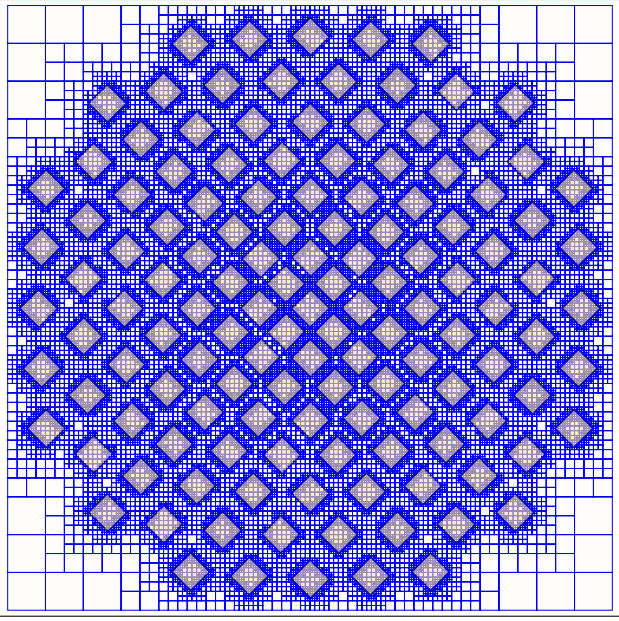


(a) Anteil Randknoten an den Knoten eines Prozesses, gemittelt über alle Prozesse



(b) Standardabweichung der von der Heuristik erzeugten Elementanzahl pro Prozess von der gleichmäßigen Verteilung. Grün: gemittelt über alle Prozesse, rot: maximaler Wert

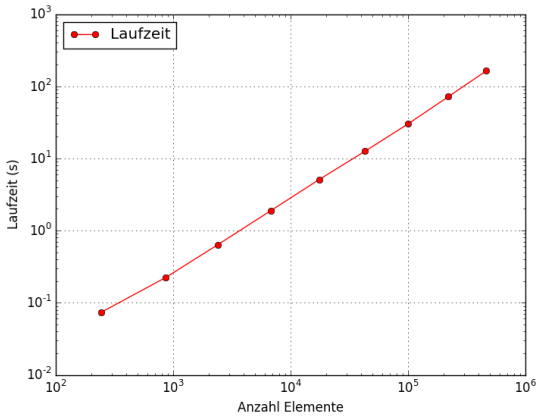
Abbildung 4.8. Geometrie mit erzeugtem Finite–Elemente–Gitter für `d_max=9`. Die Gitter innerhalb der Geometrie sind die gelb gefärbten Elemente, die sich innerhalb der auf der Spitze stehenden Quadrate befinden.



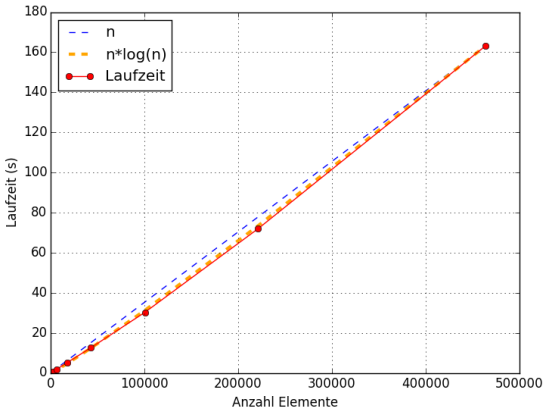
der Zuordnung der Knoten zu den Elementen herrührt, die für jedes der n Elemente durchgeführt werden muss.

Hält man dagegen die Gitterparameter fest und untersucht die Dauer des Gitteraufbaus für verschiedene Prozessanzahlen, erhält man das Resultat in Abbildung 4.10. Die benötigte Laufzeit nimmt mit zunehmender Prozessanzahl ab und erreicht wiederum einen Sättingswert.

Abbildung 4.9. Laufzeit für die Erzeugung eines Finite–Elemente–Gitters über die Anzahl an Elementen

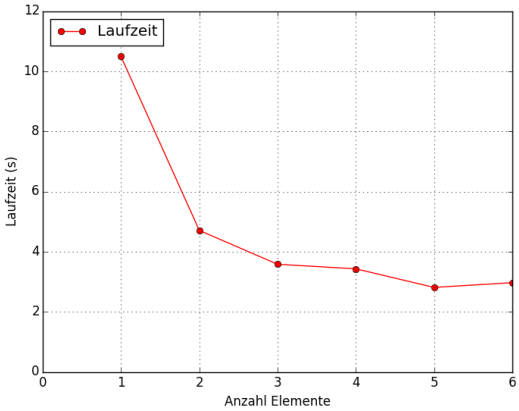


(a) Laufzeit für das Erzeugen des Finite–Elemente–Gitters bei serieller Ausführung



(b) Darstellung im Vergleich zu den Funktionen $c_1 \cdot n \log(n)$ (orange) und $c_2 \cdot n$ (blau) mit entsprechenden Werten für c_1, c_2.

Abbildung 4.10. Laufzeit für paralleles Erzeugen des Gitters aus Abbildung 4.8, `d_max=9`



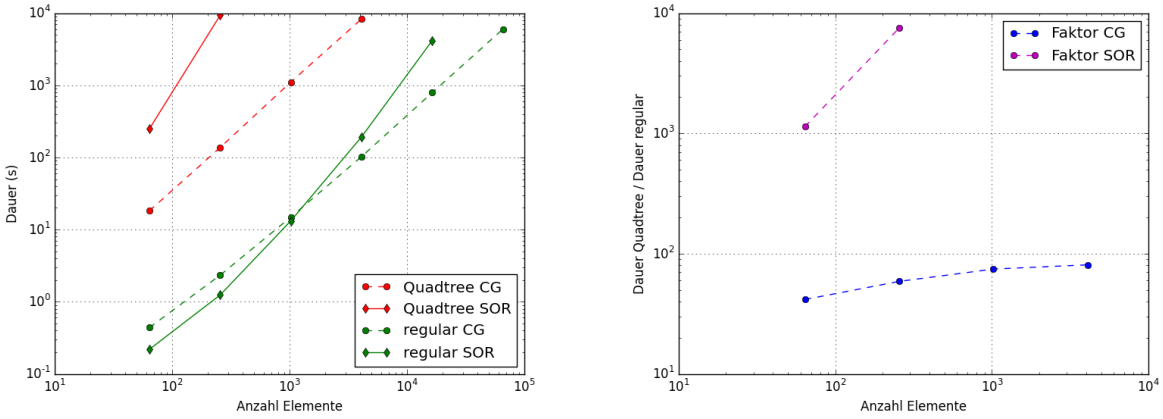
4.2.3. Laufzeit der Berechnung

Nachdem im letzten Kapitel die Laufzeit der Gitter–Generierung untersucht wurde, wird nun noch die Laufzeit der Berechnung genauer beleuchtet. Wie bereits bei dem „driven cavity"–Beispiel in Kapitel 4.2.1 erwähnt, dauert das Lösen des Poissonproblems in der Regel um mehrere Größenordnungen länger, als die Erzeugung des Gitters zu Beginn.

Als Testbeispiel wurde wieder die „driven cavity" mit $Re = 1000$, $L_1 = L_2 = 1{,}0$, mit einer Simulationszeitspanne bis $t_{end} = 2$ und einer festen Zeitschrittweite von $\delta t = 0{,}01$ verwendet. Das Programm wurde seriell gestartet. Der Parameter `d_max` wurde unterschiedlich gewählt, um Werte für verschiedene Anzahlen an Elementen zu erhalten.

In Diagramm 4.11(a) ist in rot die Laufzeit für das Lösen über die Anzahl Elemente aufgetragen. Die durchgezogene obere Linie kennzeichnet die Dauer bei Verwendung des SOR–Verfahrens, die gestrichelte Kurve beschreibt die Verwendung des CG–Verfahrens. Es ist ersichtlich, dass die Berechnung mittels des CG–Verfahrens schneller ist. Die Steigung der gestrichelten roten Geraden für die Laufzeit des CG–Verfahrens hat in diesem doppeltlogarithmischen Diagramm eine Steigung von größer eins (genauer

Abbildung 4.11. Vergleich der Laufzeit bei serieller Ausführung zwischen der Implementierung mit Quadtrees und der direkten Implementierung als reguläres Gitter



(a) Vergleich der Laufzeiten für CG und SOR–Verfahren

(b) Faktor zwischen den Implementierungen

$\approx 3/2$), was auf eine polynomielle konvexe Funktion schließen lässt. Bei Verdoppelung der Elementanzahl dauert die Berechnung also mehr als doppelt so lange.

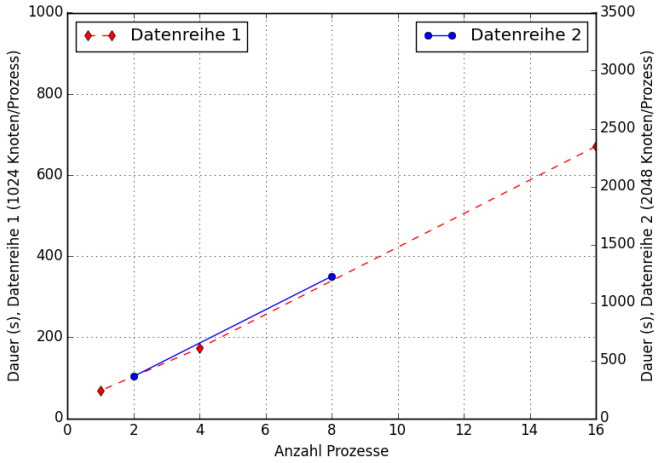
Falls bei der verwendeten Diskretisierung keine räumliche Adaptivität benötigt wird, man also mit einem regulären Gitter auskommen kann, enthält die Implementierung mit Quadtrees einen Overhead im Vergleich zu einer naiven Abspeicherung des Gitters als zweidimensionales Feld. Die Berechnungsdauer unter Verwendung eines solchen direkt abgespeicherten regulären Gitters ist in Abbildung 4.11(a) durch die grünen Kurven angegeben. Auch hier steht die durchgezogene Kurve für das SOR–Verfahren und die gestrichelte für das CG–Verfahren. Das Diagramm bestätigt, dass die Berechnung mit dieser Implementierung für beide Verfahren deutlich schneller ist. Bei Festlegung auf ein Verfahren lässt sich der Unterschied als Faktor angeben, indem man die Laufzeiten durcheinander teilt. Dies wurde in Abbildung 4.11(b) durchgeführt. Während der Faktor bei Vergleich des SOR–Verfahrens deutlich nicht konstant ist, variiert der Faktor bei Festlegung auf das CG–Verfahren mit steigender Elementanzahl weniger stark. Für 4096 Elemente liegt er bei ca. 80. Unter der hypothetischen Annahme eines linearen Zusammenhangs zwischen Elementanzahl und Berechnungsdauer kann man bei Problemen dieser Größerordnung statt des adaptiven Gitters also ein reguläres Gitter mit der 80–fachen Elementanzahl verwenden und erhält die gleiche Laufzeit. Da der lineare Zusammenhang, wie bereits erwähnt wurde, nicht gilt, liegt der Faktor jedoch etwas niedriger. Die Verwendung des adaptiven Gitters lohnt sich also erst, wenn die Adaptivität auch stark ausgenutzt wird, wenn das adaptive Gitter also weitaus weniger Elemente besitzt, als ein volles Gitter mit der kleinsten auftretenden Gitterweite.

4.2.4. Schwache Skalierung

Statt der eingangs untersuchten starken Skalierung lässt sich die schwache Skalierung betrachten, wenn man an der Laufzeit größerer Probleme bei gleichzeitiger Verwendung mehrerer Prozessoren interessiert ist. Dabei bleibt die durchschnittliche Anzahl Elemente pro Prozess gleich.

Um diese Untersuchung durchzuführen, wurde erneut die „driven cavity“ unter Verwendung des CG–Verfahrens mit den gleichen Parametern wie in Kapitel 4.2.3 berechnet. Um die Elementanzahl einfach festlegen zu können, wurde dieses Szenario mit einem regulären Gitter verwendet. Als Parameter für die Anzahl der Elemente n steht ausschließlich das maximale Level `d_max` zur Verfügung. Es gilt der

Abbildung 4.12. Untersuchung der schwachen Skalierung



Zusammenhang $2^{2\,d_{max}} = n$. Wird als konstante Elementanzahl pro Prozess ein Wert $c = 2^{k_c}$ festgelegt, gilt für die Prozessanzahl $p = 2^{k_p}$:

$$c = \frac{n}{p} = \frac{2^{2\,d_{max}}}{2^{k_p}} = 2^{k_c}$$

und somit $2\,d_{max} - k_p = k_c$ und deshalb für den Parameter `d_max`:

$$d_{max} = \frac{1}{2}\,(k_c + k_p).$$

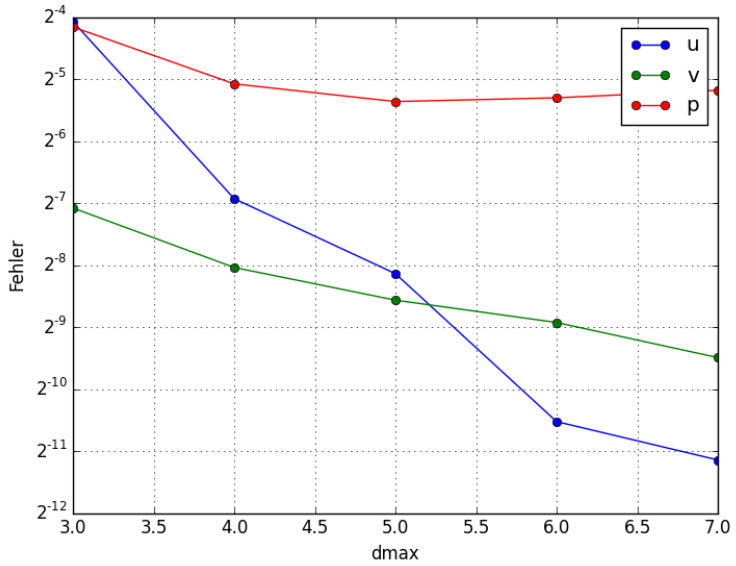
Für einen festen Wert von k_c erhält man ganzzahlige Werte für d_{max} also nur für jeden zweiten Wert von k_p. Dies entspricht Prozessanzahlen p, die sich um den Faktor 4 unterscheiden. Der Testlauf des Programms wurde deshalb für zwei verschiedene Datenreihen durchgeführt: Einmal für $k_c = 10$ und mit 1, 4 und 16 Prozessen. Dies erfordert die Parameterwerte $d_{max} = 5$, 6 und 7. Das zweite mal für $k_c = 11$ und mit 2 und 8 Prozessen mit den Parameterwerten $d_{max} = 6$ und 7. Das Resultat ist in Abbildung 4.12 dargestellt. Die beiden Datensätze wurden – ersichtlich an den Achsenbeschriftungen – unterschiedlich skaliert, sodass die Kurven aufeinander liegen.

Man erkennt mit ansteigender Prozess- und gleichzeitig ansteigender Elementanzahl, dass die Laufzeit etwa linear zunimmt. Während die Laufzeit bei serieller Ausführung stärker als linear mit der Elementanzahl ansteigt, kann die Linearität bei Verwendung entsprechend mehrerer Prozesse wiederhergestellt werden, zumindest im Bereich bis 16 Prozesse. Um ein doppelt so großes Problem zu berechnen, muss trotz doppelter Prozessanzahl also auch die doppelte Laufzeit in Kauf genommen werden. Deshalb stellt dies nur eine mäßige schwache Skalierung dar.

4.3. Konvergenz

Für die Poiseuille–Strömung wird nun untersucht, wie sich der Fehler bei Verringerung der Gitterweite verhält. Da die analytische Lösung bekannt ist, kann der Fehler des Resultats eines Simulationslaufs einfach bestimmt werden. Als Maß wird der Fehler in der $\mathcal{L}_2$–Norm betrachtet. Abbildung 4.13 zeigt den Fehler für die Geschwindigkeiten `u` und `v` sowie für den Druck `p`. Simuliert wurde die vollständige Poiseuille–Strömung mit Parametern wie für die Validierung in Abbildung 4.11. Der Druck und die Geschwindigkeit `v`

Abbildung 4.13. Konvergenzanalyse der Poiseuilleströmung für verschiedene Gitterweiten



werden zum Zeitpunkt $t = 100s$ entlang eines horizontalen Schnittes, die Geschwindigkeit `u` entlang eines vertikalen Schnittes, jeweils in der Mitte des Gebiets, ausgewertet.

Die rote Kurve für den Fehler des Drucks zeigt keine Konvergenz. Dies liegt daran, dass sich zwar der korrekte lineare Druckabfall einstellt, dieser aber zur analytischen Lösung leicht verschoben ist. Dies hängt mit der Realisierung der Dirichlet–Druckrandbedingungen zusammen, die am linken und rechten Rand des Gebietes auf einem Streifen mit jeweils endlicher Ausdehnung erzwungen wird. Die Geschwindigkeit `v` sollte entlang des Schnittes komplett verschwinden, stattdessen nimmt sie noch kleine Werte an, was darauf hindeutet, dass der stationäre Zustand noch nicht komplett erreicht ist. Deshalb ist deren Verlauf im Fehlerdiagramm wenig aussagekräftig. Die Geschwindigkeit `u` lässt sich durch eine Gerade mit Steigung -1 annähern. Im Diagramm ist die Achse des Fehlers logarithmisch zur Basis 2 skaliert, während auf der Abszisse das maximale Elementlevel aufgetragen ist. Bei der Erhöhung des Elementlevels um eins verdoppelt sich die Elementanzahl entlang einer Koordinatenrichtung, sodass sich die gesamte Anzahl vervierfacht. Zwischen den eingezeichneten halben Inkrementen von `dmax` liegt also eine Verdoppelung der Elementanzahl vor. Hier kann also eine lineare Konvergenz in der Anzahl der Elemente festgestellt werden.

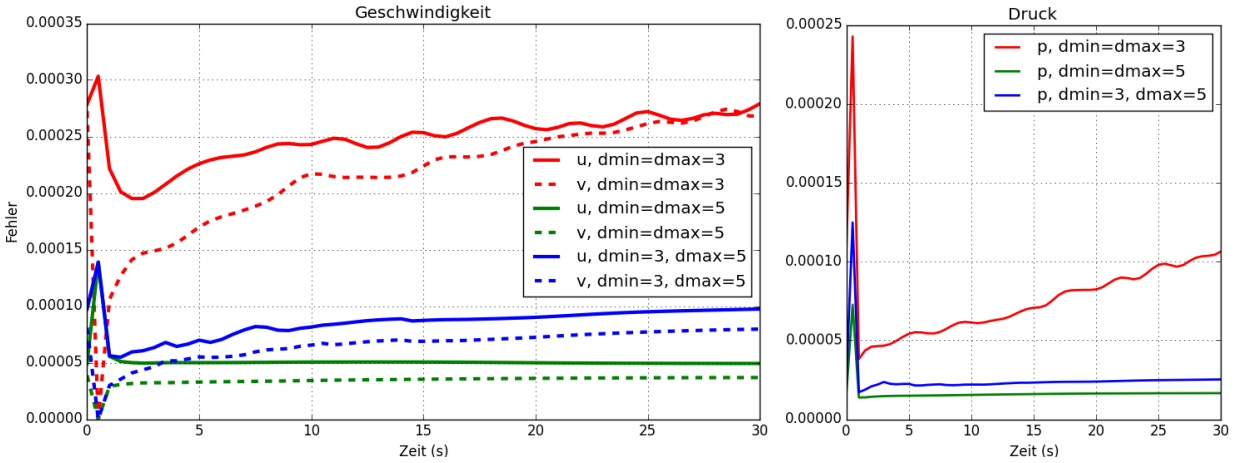
Als nächstes wird der Fehler, der bei Anwendung der dynamischen Adaptivität auftritt, untersucht. Dafür wird das in Kapitel 3.1.3 in Abbildung 3.2 eingeführte Szenario der „driven cavity“ verwendet. Da hier keine analytische Lösung vorliegt, wurde als Referenzlösung das Simulationsergebnis für ein reguläres Gitter mit `d_min=d_max=7` verwendet. Der diesbezügliche Fehler der Berechnung mit dynamisch adaptivem Gitter mit Level zwischen `d_min=3` und `d_max=5` ist in Abbildung 4.14 über den Verlauf der Simulation im Zeitintervall von 30 Sekunden dargestellt. Zum Vergleich sind außerdem die Fehler bei regulären Gittern mit Level 3 und 5 eingetragen. Zum Endzeitpunkt der Berechnung $t = 30s$ wird das Gebiet, ähnlich wie in Abbildung 3.9(e), durch Elemente dargestellt, die der Verteilung in Tabelle 4.1 folgen.

Man erkennt, dass der Fehler der Geschwindigkeiten in Abbildung 4.14(a) sowie der Fehler des Drucks in Abbildung 4.14(b) jeweils zwischen den beiden Vergleichswerten liegt. Die Fehler der Berechnungen mit regulären Gittern mit den Gitterweiten, zwischen denen sich die adaptive Gitterweite bewegt, definieren also eine untere und eine obere Schranke für den Fehler der adaptiven Berechnung. Durch Tabelle 4.1 ist ersichtlich, dass die Aufteilung der Fläche auf Elemente der drei möglichen Größen etwa gleich ist.

Level	Anzahl Elemente	Fläche der Elemente
3	23	0.359
4	75	0.293
5	356	0.348

Tabelle 4.1.: Elementverteilung des adaptiven Gitters der „driven cavity“–Simulation für $t = 30$

Abbildung 4.14. Fehleranalyse bei dynamischer Adaptivität für das „driven cavity“–Problem. Als Referenzdaten dient das Ergebnis für `d_max=d_min=7`.



(a) Fehler in der Geschwindigkeit

(b) Fehler im Druck

Man könnte deshalb einen Fehler erwarten, der in der Mitte der feinen und der groben Diskretisierung liegt. Stattdesen liegt der Fehler im Diagramm 4.14 jeweils näher bei dem Vergleichswert der genaueren Diskretisierung. Dies zeigt, dass das adaptive Verfeinerungskriterium gut gewählt ist.

In Kapitel 2.4.4 wurde eine Variante der Berechnungsvorschrift der rechten Seite des diskreten Poissonproblems eingeführt, die die Erfüllung der Kontinuitätsgleichung im vorherigen Zeitschritt erfordert. Im ersten Zeitschritt muss deshalb auf die davor beschriebene kanonische Variante zurückgegriffen werden, die diese Einschränkung nicht benötigt. Wird ab dem nächsten Zeitschritt jedoch immer die Variante verwendet, summiert sich der Fehler in der Erfüllung der Kontinuität auf. Dies wird in Abbildung 4.15 gezeigt. Als Szenario dient die übliche „driven cavity“ mit regulärem Gitter auf Level 5 und Reynoldszahl 1000 mit Simulationszeitraum von 20 Sekunden. Zu sehen ist das Residuum r der Kontinuitätsgleichung in der schwachen Form:

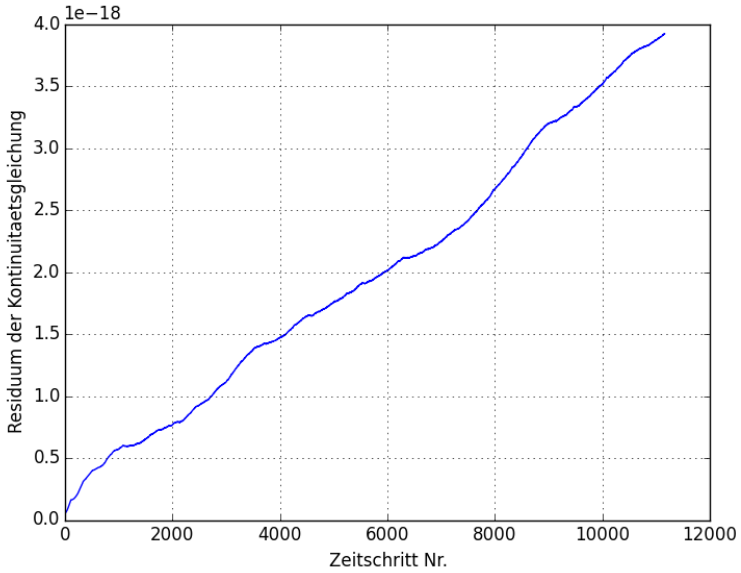
$$r = A_1 \, \mathbf{u}^{(n)} + A_2 \, \mathbf{v}^{(n)},$$

aufgetragen über die Berechnungszeitpunkte n. Erkennbar ist der Drift, der sich im Laufe der Zeit ergibt.

4.4. Anwendungsbeispiele

In diesem Kapitel werden zwei weitere Szenarien vorgestellt, die unter Verwendung der dynamischen Adaptivität berechnet wurden.

Abbildung 4.15. Drift bei Verwendung der Variante für die Berechnung der rechten Seite des Poissonproblems. Dargestellt ist das Residuum der Kontinuitätsgleichung über die Anzahl Iterationen



4.4.1. Strömung über eine Stufe

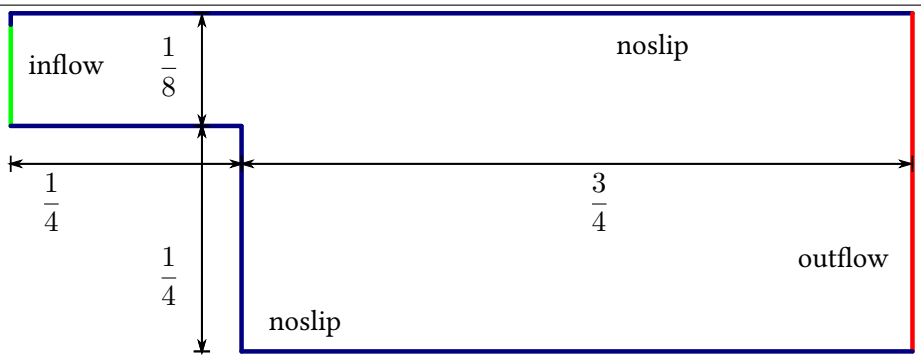
Das erste Szenario bildet die Strömung über eine Stufe, wie in Abbildung 4.16 dargestellt. Durch eine *inflow*–Randbedingung strömt die Flüssigkeit durch einen Kanal der Höhe $1/8$ und der Länge $1/4$ mit *noslip*–Rändern. Dieser erweitert sich daraufhin und bildet nach unten eine Stufe der Höhe $1/4$. Nach einem Bereich der Länge $3/4$ kann das Fluid das Gebiet durch eine *outflow*–Randbedingung verlassen.

Für drei herausgegriffene Zeitpunkte ist der Geschwindigkeits- und Druckverlauf in Abbildung 4.17 dargestellt. In Abbildung 4.17(a) zum Zeitpunkt $t = 0{,}54$ bildet sich hinter der Stufe ein Bereich mit geringerem Druck, sodass das Fluid in der Folge dort nach unten strömt. Es ensteht ein Wirbel im Uhrzeigersinn, der langsam weiter nach rechts unten wandert. In Abbildung 4.17(b) ist das Vektorfeld der Geschwindigkeit durch Pfeile angegeben. Man erkennt für $t = 1{,}40$ den Wirbel hinter der Stufe, außerdem bildet sich ein Wirbel am oberen Rand des Gebiets. Zum Zeitpunkt $t = 2{,}77$ in Abbildung 4.17(c) befindet sich der Bereich mit niedrigem Druck weiter rechts. Im Geschwindigkeitsfeld ist zu sehen, dass das Fluid nach Erreichen der Stufe zunächst weiter geradeaus strömt, bevor es dann die Richtung leicht nach unten ändert. Auch die Position des rückwärtsgerichteten Wirbels unter der Hauptströmung ist sichtbar. Mit weiter voranschreitender Zeit schwächen sich die beiden Wirbel ab, was in Abbildung 4.17(d) sichtbar ist. Wegen der *outflow*–Bedingung kann das Fluid an der komplette rechte Seite entweichen. Der stationäre Zustand, der sich nach weiteren 15 Sekunden einstellt, besteht darin, dass der Flüssigkeitsstrahl auch nach der Stufe gerade weiter strömt und das Gebiet leicht aufgefächert am rechten Rand oben verlässt.

Bei diesem Beispiel wurde das Finite–Elemente–Gitter adaptiv alle 5 Zeitschritte angepasst. Dabei wurde die Vorticity w für jedes Element berechnet. Elemente, deren Absolutwert der Vorticity unter dem Schwellwert von `vorticity_min_threshold=0.01` liegen, werden in dem Zeitschritt vergröbert, wenn die Bedingung auf alle jeweiligen 4 Geschwisterelemente zutrifft. Elemente mit einer absoluten Vorticity von über `vorticity_max_threshold=0.02` werden verfeinert. Danach wird die 2:1–Balanceeigenschaft wieder hergestellt.

In Abbildung 4.18 ist das Finite–Elemente–Gitter zu verschiedenen Berechnungszeitpunkten dargestellt. Elemente, deren absolute Vorticity unter dem Schwellwert liegen, sind hellgrau dargestellt und werden gegebenenfalls verfeinert. In Abbildung 4.18(a) sind dies einige Elemente im mittleren rechten Bereich. Dass

Abbildung 4.16. Geometrie und Randbedingungen des Szenarios „Strömung über Stufe“



diese nicht schon durch gröbere Elemente ersetzt wurden, liegt an der einzuhaltenden Balanceeigenschaft. Elemente, in denen die absolute Vorticity zwischen den beiden Grenzwerten liegt, sind dunkelgrau gefärbt. Diese werden nicht geändert und befinden sich an den Rändern der Wirbel. Elemente mit größerer absoluter Vorticity sind orange und rot. Falls die Vorticity einen Wert kleiner als $-0{,}2$ hat, ist die das Element orange gefärbt und wird im nächsten Zeitschritt verfeinert, falls es nicht schon das maximale Level erreicht hat. Diese Elemente treten in dem sich hinter der Stufe bildenden, im Uhrzeigersinn drehenden Wirbel auf. Die rote Farbe kennzeichnet die Elementen mit einem Vorticitywert über $+0{,}2$, was am oberen Rand und z. B. in Abbildung 4.18(b) im Gegenwirbel oben rechts gegeben ist. Man erkennt, dass sich in Abbildung 4.18(c) für $t = 26{,}27$ die Bereiche höherer Vorticity nach rechts verschoben haben. Hinter der Stufe treten nur noch schwache Verwirbelungen auf, sodass dort Elemente eingespart werden, indem gröbere Elemente verwendet werden.

4.4.2. Karman’sche Wirbelstraße

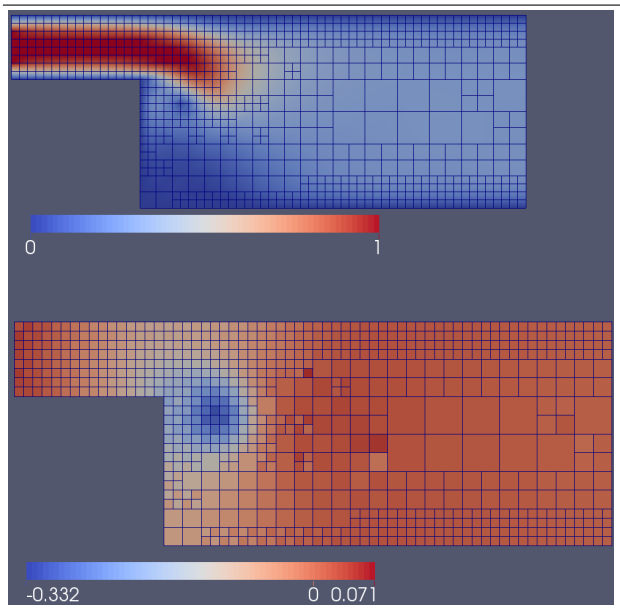
Das nächste Beispiel umfasst die Strömung um ein asymmetrisches Hindernis, das sich in einem Kanal befindet. Für eine ausreichend hohe Reynoldszahl bildet sich hinter dem Hindernis eine *Karman’sche Wirbelstraße*. Diese besteht aus Wirbeln, die sich vom Hindernis ablösen und in Flussrichtung fortbewegen. Die Drehrichtung zweier aufeinanderfolgender Wirbel ist dabei entgegengesetzt. Für $Re = 1000$ erhält man das in Abbildung 4.19(a) dargestellte Geschwindigkeitsfeld. Für dessen Simulation wurde auf der linken Seite eine *inflow*–Randbedingung mit $(u, v) = (1{,}0)$ und auf der rechten Seite eine *outflow*–Randbedingung festgesetzt. An den Wänden oben und unten gelten *noslip*–Randbedingungen. Die Länge des Kanals beträgt $L_1 = 1$, die Breite beträgt $L_2 = 0{,}3$. Für die erste Simulation in Abbildung 4.19(a) wurde ein reguläres Gitter verwendet, das aus Elementen mit Level `d_max=8` besteht.

Für die zweite Simulation in Abbildung 4.19(b) wurde die adaptive Gitteranpassung eingeschaltet. Das Gitter wird alle 10 Zeitschritte vergröbert bzw. verfeinert. Als Schwellwerte für die Vorticity wurde wieder $0{,}01$ und $0{,}02$ verwendet. Die Grenzen der Levels des Quadtrees der Elemente liegen bei `d_min=6` und `d_max=9`.

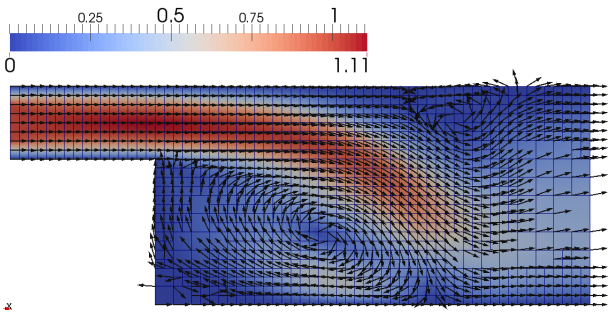
Man erkennt in Abbildung 4.19(b), dass sich vor dem Hindernis und zum Ende des Kanals grobe Elemente bilden. Zum abgebildeten Zeitpunkt $t = 0{,}57$ haben sich zwei Wirbel mit Drehrichtung entgegen dem Uhrzeigersinn und ein Wirbel mit Drehrichtung im Uhrzeigersinn vom Hindernis abgelöst. Der zweite befindet sich an der Oberkante des Hindernisses. In Abbildung 4.19(c) ist ein vergrößerter Ausschnitt des Simulationsgebiets zu sehen, der den Bereich der Wirbelablösung hinter dem Hindernis enthält. Durch Pfeile ist die Richtung des Geschwindigkeitsfelds und damit auch der Wirbel veranschaulicht.

In den Bereichen der Wirbel liegt die Vorticity über dem Schwellwert, sodass hier feine Elemente verwendet werden. Im Bereich zwischen den Wirbeln sind Elemente, deren Level um eins geringer als das maximale

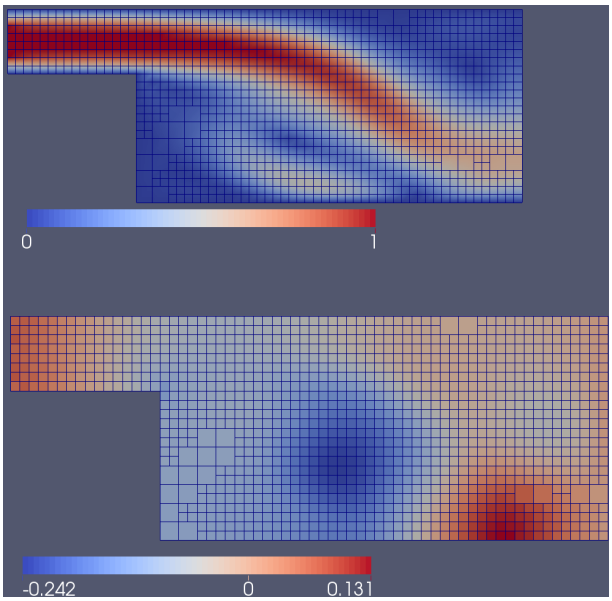
Abbildung 4.17. Strömung über eine Stufe, $Re = 1000, L_1 = 1, L_2 = 3/8$, `d_min=3`, `d_max=6`, `vorticity_min_threshold=0.01`, `vorticity_max_threshold=0.02`, `refine_interval=5`



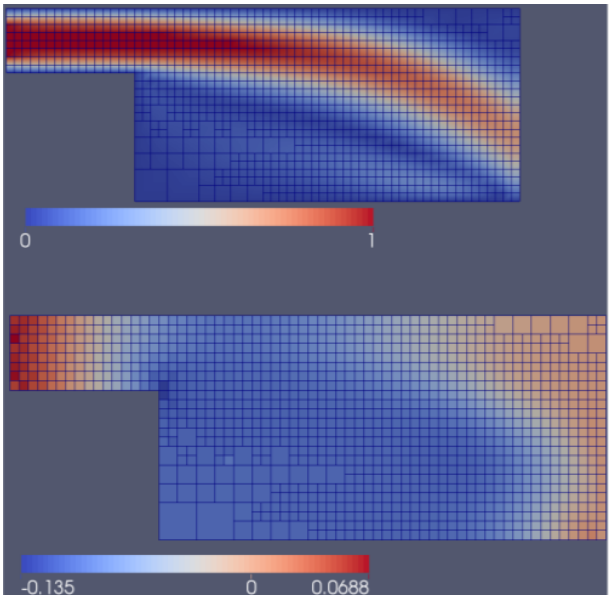
(a) Zeitpunkt $t = 0{,}54$, oben: Geschwindigkeitsfeld, Farbgebung nach Betrag der Geschwindigkeit, unten: Druckfeld



(b) Zeitpunkt $t = 1{,}40$, Geschwindigkeitsfeld, Farbgebung nach Betrag der Geschwindigkeit, die unskalierten Pfeile geben die Richtung an



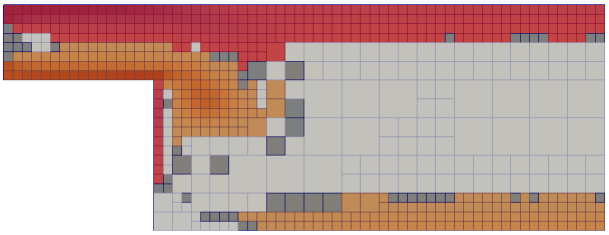
(c) Zeitpunkt $t = 2{,}77$, Oben: Geschwindigkeitsfeld, Farbgebung nach Betrag der Geschwindigkeit, unten: Druckfeld



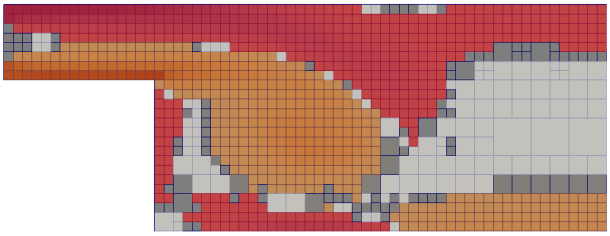
(d) Zeitpunkt $t = 23{,}36$, Oben: Geschwindigkeitsfeld, Farbgebung nach Betrag der Geschwindigkeit, unten: Druckfeld

Level ist, zu finden. Außerdem haben sich entlang des Randes aufgrund der *noslip*–Randbedingung feine Elemente gebildet. Zum Zeitpunkt der Abbildung enthält das Finite–Elemente–Gitter 35 223 Elemente, was etwa 45% der maximal möglichen Elementanzahl im Gebiet für `d_max=9` entspricht. Da sich bei der verwendeten Reynoldszahl von 1000 keine Wirbel am Rand des Kanals bilden, sind die dort eingesetzten feinen Elemente nicht erforderlich. Durch eine andere Verfeinerungsstrategie könnte die Entstehung dieser Elemente eingeschränkt werden.

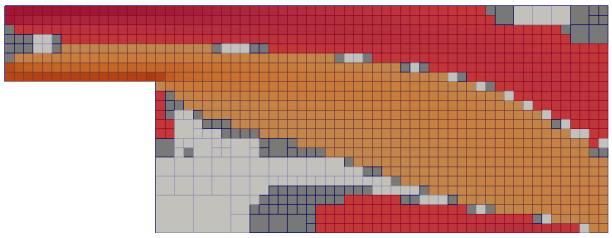
Abbildung 4.18. Strömung über eine Stufe, Farbgebung gemäß Vorticity w.
Hellgrau: $|w| < 0{,}1$, dunkelgrau: $0{,}1 \leq |w| \leq 0{,}2$, orange: $w < -0{,}2$, rot: $w > 0{,}2$



(a) $t = 0{,}33$

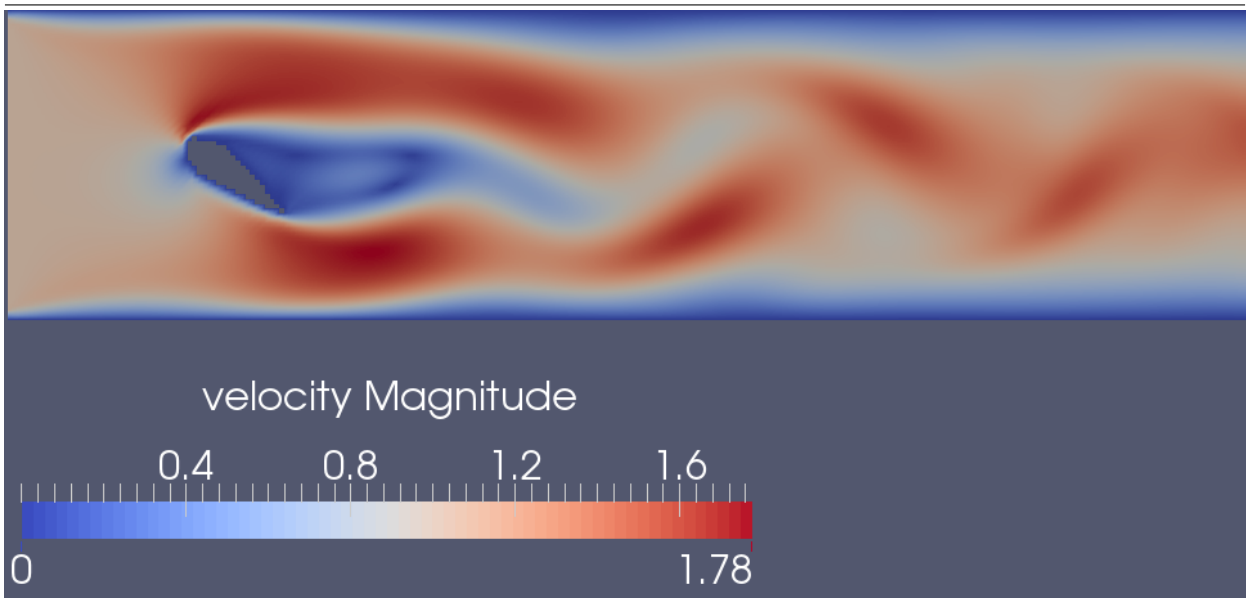


(b) $t = 1$

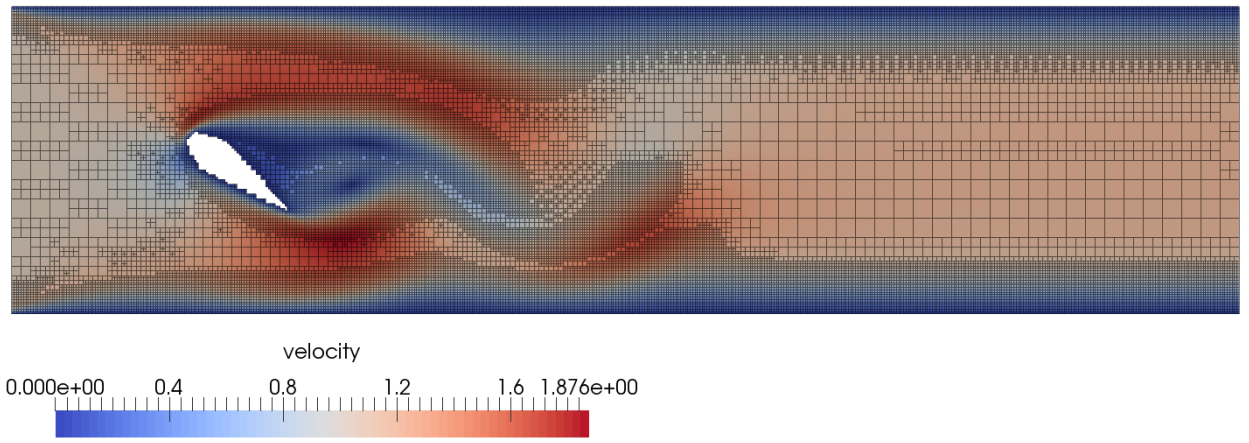


(c) $t = 26{,}27$

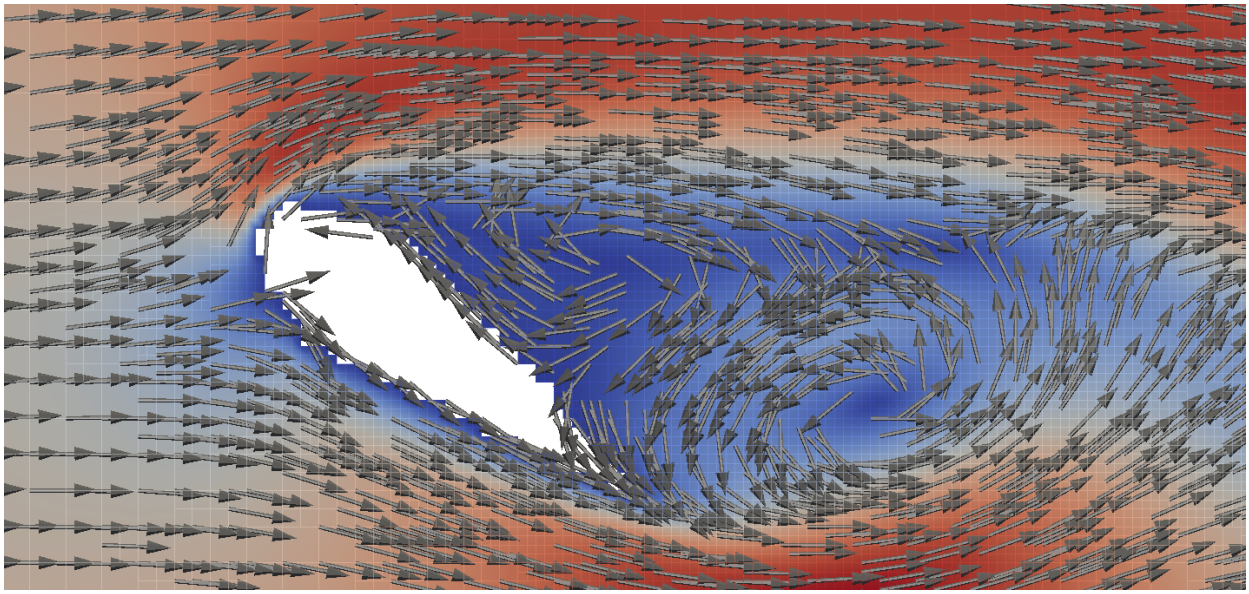
Abbildung 4.19. Karman'sche Wirbelstraße



(a) Zustand für $t = 3{,}73$, Geschwindigkeitsfeld



(b) Gesamtansicht der adaptiven Simulation für $t = 0{,}57$



(c) Vergrößerter Bereich hinter dem Hindernis, in dem die Wirbelablösung stattfindet

5. Zusammenfassung und Ausblick

In dieser Arbeit werden Methoden zur Berechnung einer Strömung mit der Finite–Elemente–Methode auf einem dynamisch adaptiven Gitter erläutert und implementiert. Dies ermöglicht eine feine Diskretisierung von beliebig verlaufenden Rändern, während im Inneren gröbere Elemente eingesetzt werden können. Außerdem lässt sich das Gitter während der Simulation in bestimmten Zeitintervallen neu erzeugen, sodass bei sich fortbewegenden Wirbeln die Feinheit der Diskretisierung entsprechend mitbewegt werden kann.

Um die dafür benötigten quadratischen Elemente verschiedener Größe abzuspeichern, wird eine schlanke Datenstruktur unter Verwendung von Quadtrees eingeführt, die ohne Zeiger zwischen den Elementen auskommt. Zu Beginn dieser Arbeit werden Quadtrees und deren lineare Repräsentation im Speicher als sortierte Liste von Knoten–IDs eingeführt. Anschließend werden die in [SSB08] vorgeschlagenen parallelen Algorithmen zur Erzeugung eines Quadtrees und zur Herstellung der 2:1–Balancierung anhand von Beispielen erläutert. Dabei spielen sogenannte Blöcke eine wichtige Rolle, die zusammen eine grobe Approximation des Quadtrees liefern. Diese Approximation wird für eine heuristische Lastbalancierung verwendet, die versucht, die Elemente gleichmäßig auf die Prozesse zu verteilen, sodass die Ränder der Teilgebiete gleichzeitig klein sind.

Um die Finite–Elemente–Methode anwenden zu können, muss die Datenstruktur neben den Werten auf den Elementen auch Werte in den Knoten bereitstellen. Um dies zu erreichen, wird ein weiterer Quadtree erzeugt, dessen Einträge die Knoten repräsentieren. Dabei werden hängende Knoten, die an „T“–Kreuzungen der Elementkanten entstehen, nicht abgespeichert, sondern bei der Berechnung entsprechend berücksichtigt.

Als nächstes werden in der vorliegenden Arbeit die Navier–Stokes–Gleichungen zur Behandlung des Strömungsproblems eingeführt. Der darauf basierende Finite–Elemente–Ansatz wird hergeleitet. Es werden bilineare Ansatzfunktionen für die Geschwindigkeit und konstante Ansatzfunktionen für den Druck verwendet. Nach Durchführung der zeitlichen Diskretisierung wird die sich ergebende Berechnungsvorschrift zusammengefasst. Anschließend wird darauf eingegangen, wie die Gleichungen auf dem adaptiven Finite–Elemente–Gitter gelöst werden können. Es werden Element- und Knotentypen definiert, für die eine jeweils angepasste Berechnung notwendig ist.

Ebenso wird die Umsetzung im Programm näher erläutert. Das Programm liest Geometrieinformation aus svg–Vektorgraphiken ein und schreibt Ausgabedateien im vtk–Dateiformat. Nach der Beschreibung der Bedienung des Programms und einem Überblick über dessen Klassenstruktur werden die Möglichkeiten der Lösung des linearen Gleichungssystems beschrieben. Das Programm erlaubt die iterative Lösung durch SOR- und CG–Verfahren, sowie das direkte Lösen.

Um während der Simulationszeit bewegliche Merkmale der Strömung, wie z. B. Wirbel, auflösen zu können, kann das Gitter basierend auf Werten der Vorticity adaptiv verfeinert oder vergröbert werden. Dabei findet eventuell eine Neuzuweisung der Elemente an die Prozesse statt, um die Lastbalancierung zu gewährleisten.

Nach der Validierung mittels eines *Hagen–Poiseille*–Flusses und der „driven cavity“ werden Laufzeituntersuchungen durchgeführt. Es wird eine gute starke parallele Skalierung festgestellt, während die schwache Skalierung mittelmäßig ausfällt. Dies liegt vorallem an der Zeitkomplexität des Lösers, die schlechter als linear verläuft. Es wird gezeigt, dass die Gittergenerierung meist einen vernachlässigbar kleinen Anteil

an der Laufzeit besitzt, jedoch ebenfalls eine gute parallele Skalierung aufweist. Es wird außerdem ein Vergleich unter Verwendung eines regulären Gitters durchgeführt, das einmal mithilfe der vorgestellten Datenstruktur gespeichert wird und einmal gemäß einer klassischen Implementierung verwendet wird. Dabei ergibt sich, dass die Quadtree–Datenstruktur um einen Fakter in der Größenordnung von 80 langsamer ist.

Die Verwendung der dynamischen Adaptivität wird schließlich anhand der Strömung über eine Stufe und anhand einer Karman'schen Wirbelstraße demonstriert.

Erweiterungen des Programms können auf die Verbesserung der Laufzeit der Löser hinzielen, z. B. indem ein Mehrgitterverfahren verwendet wird. Die Laufzeit des Programms bei paralleler Ausführung kann noch verringert werden, indem z. B. die Kommunikation komplett nicht–blockierend ausgeführt wird und damit Berechnung und Kommunikation überlappen können. Zudem kann die Lastbalancierung genauer untersucht werden, auch im Hinblick auf die auftretenden Umverteilungen der Elemente auf die Prozesse bei dynamischer Adaptivität. Als Kriterien für die Verfeinerung oder Vergröberung können weitere Strategien angewendet werden. Bezüglich des Gitters sind Erweiterungen unter Verwendung von dreieckigen Elementen möglich, z. B. um Gebietsränder mit weniger Elementen diskretisieren zu können.

Prinzipiell ist auch eine Verwendung der implementierten adaptiven Datenstruktur für andere Differentialgleichungen, z. B. für die Beschreibung von Problemen der Strukturmechanik, möglich.

A. Koeffizentenmatrizen

Die Matrizen `Dkoeff`, `C1koeff` sowie `C2koeff` sind in den Abbildungen A.1, A.2 und A.3 dargestellt. h bezeichnet die Seitenlänge des Elements.

Der Eintrag (i, j) in `Dkoeff` entspricht dem Beitrag des Knotenpaars (i, j), wobei die Nummerierung wie folgt gewählt wird: Der erste Knoten ist links unten, der zweite rechts unten, der dritte links oben und der vierte rechts oben.

Für die Matrizen `C1koeff` und `C2koeff` gilt diese Nummerierung ebenfalls. Hier ist der Eintrag (i, j) ein Vektor aus vier Werten $k = 1, \ldots, 4$, die wiederum für die vier Knoten stehen. Ein Eintrag entspricht dem Tupel von drei Knoten (i, j, k).

Abbildung A.1. Matrix `Dkoeff` (ohne Vorfaktor $1/Re$)

$$\begin{bmatrix} \frac{2}{3} & -\frac{1}{6} & -\frac{1}{6} & -\frac{1}{3} \\ -\frac{1}{6} & \frac{2}{3} & -\frac{1}{3} & -\frac{1}{6} \\ -\frac{1}{6} & -\frac{1}{3} & \frac{2}{3} & -\frac{1}{6} \\ -\frac{1}{3} & -\frac{1}{6} & -\frac{1}{6} & \frac{2}{3} \end{bmatrix}$$

(a) Elementtyp A

$$\begin{bmatrix} \frac{2}{3} & -\frac{1}{3} & \frac{1}{12} & -\frac{5}{12} \\ -\frac{1}{3} & \frac{2}{3} & -\frac{1}{6} & -\frac{1}{6} \\ \frac{1}{12} & -\frac{1}{6} & \frac{1}{6} & -\frac{1}{12} \\ -\frac{5}{12} & -\frac{1}{6} & -\frac{1}{12} & \frac{2}{3} \end{bmatrix}$$

(b) Elementtyp B

$$\begin{bmatrix} \frac{1}{6} & -\frac{1}{12} & 0 & -\frac{1}{12} \\ -\frac{1}{12} & \frac{2}{3} & -\frac{1}{2} & -\frac{1}{12} \\ 0 & -\frac{1}{2} & \frac{1}{2} & 0 \\ -\frac{1}{12} & -\frac{1}{12} & 0 & \frac{1}{6} \end{bmatrix}$$

(c) Elementtyp C

Die Faktoren für die Operatoren A_1 und A_2 sind in Abbildung A.4 dargestellt. Die Operatoren $A_1^\top$ und $A_2^\top$ sind in den Abbildungen A.5 und A.6 abgebildet.

Abbildung A.2. Matrix `C1koeff`

$$\begin{bmatrix}
\begin{bmatrix} -\frac{1}{12}h\sim \\ -\frac{1}{24}h\sim \\ -\frac{1}{36}h\sim \\ -\frac{1}{72}h\sim \end{bmatrix} & \begin{bmatrix} \frac{1}{12}h\sim \\ \frac{1}{24}h\sim \\ \frac{1}{36}h\sim \\ \frac{1}{72}h\sim \end{bmatrix} & \begin{bmatrix} -\frac{1}{36}h\sim \\ -\frac{1}{72}h\sim \\ -\frac{1}{36}h\sim \\ -\frac{1}{72}h\sim \end{bmatrix} & \begin{bmatrix} \frac{1}{36}h\sim \\ \frac{1}{72}h\sim \\ \frac{1}{36}h\sim \\ \frac{1}{72}h\sim \end{bmatrix} \\
\begin{bmatrix} -\frac{1}{24}h\sim \\ -\frac{1}{12}h\sim \\ -\frac{1}{72}h\sim \\ -\frac{1}{36}h\sim \end{bmatrix} & \begin{bmatrix} \frac{1}{24}h\sim \\ \frac{1}{12}h\sim \\ \frac{1}{72}h\sim \\ \frac{1}{36}h\sim \end{bmatrix} & \begin{bmatrix} -\frac{1}{72}h\sim \\ -\frac{1}{36}h\sim \\ -\frac{1}{72}h\sim \\ -\frac{1}{36}h\sim \end{bmatrix} & \begin{bmatrix} \frac{1}{72}h\sim \\ \frac{1}{36}h\sim \\ \frac{1}{72}h\sim \\ \frac{1}{36}h\sim \end{bmatrix} \\
\begin{bmatrix} -\frac{1}{36}h\sim \\ -\frac{1}{72}h\sim \\ -\frac{1}{36}h\sim \\ -\frac{1}{72}h\sim \end{bmatrix} & \begin{bmatrix} \frac{1}{36}h\sim \\ \frac{1}{72}h\sim \\ \frac{1}{36}h\sim \\ \frac{1}{72}h\sim \end{bmatrix} & \begin{bmatrix} -\frac{1}{36}h\sim \\ -\frac{1}{72}h\sim \\ -\frac{1}{12}h\sim \\ -\frac{1}{24}h\sim \end{bmatrix} & \begin{bmatrix} \frac{1}{36}h\sim \\ \frac{1}{72}h\sim \\ \frac{1}{12}h\sim \\ \frac{1}{24}h\sim \end{bmatrix} \\
\begin{bmatrix} -\frac{1}{72}h\sim \\ -\frac{1}{36}h\sim \\ -\frac{1}{72}h\sim \\ -\frac{1}{36}h\sim \end{bmatrix} & \begin{bmatrix} \frac{1}{72}h\sim \\ \frac{1}{36}h\sim \\ \frac{1}{72}h\sim \\ \frac{1}{36}h\sim \end{bmatrix} & \begin{bmatrix} -\frac{1}{72}h\sim \\ -\frac{1}{36}h\sim \\ -\frac{1}{24}h\sim \\ -\frac{1}{12}h\sim \end{bmatrix} & \begin{bmatrix} \frac{1}{72}h\sim \\ \frac{1}{36}h\sim \\ \frac{1}{24}h\sim \\ \frac{1}{12}h\sim \end{bmatrix}
\end{bmatrix}$$

(a) Elementtyp A

$$\begin{bmatrix}
\begin{bmatrix} -\frac{5}{32}h\sim \\ -\frac{17}{288}h\sim \\ -\frac{11}{288}h\sim \\ -\frac{11}{288}h\sim \end{bmatrix} & \begin{bmatrix} \frac{17}{144}h\sim \\ \frac{7}{144}h\sim \\ \frac{1}{48}h\sim \\ \frac{1}{48}h\sim \end{bmatrix} & \begin{bmatrix} -\frac{11}{288}h\sim \\ -\frac{1}{96}h\sim \\ -\frac{5}{288}h\sim \\ -\frac{5}{288}h\sim \end{bmatrix} & \begin{bmatrix} \frac{11}{144}h\sim \\ \frac{1}{48}h\sim \\ \frac{5}{144}h\sim \\ \frac{5}{144}h\sim \end{bmatrix} \\
\begin{bmatrix} -\frac{17}{288}h\sim \\ -\frac{7}{72}h\sim \\ -\frac{1}{96}h\sim \\ -\frac{1}{24}h\sim \end{bmatrix} & \begin{bmatrix} \frac{7}{144}h\sim \\ \frac{1}{12}h\sim \\ \frac{1}{144}h\sim \\ \frac{1}{36}h\sim \end{bmatrix} & \begin{bmatrix} -\frac{1}{96}h\sim \\ -\frac{1}{72}h\sim \\ -\frac{1}{288}h\sim \\ -\frac{1}{72}h\sim \end{bmatrix} & \begin{bmatrix} \frac{1}{48}h\sim \\ \frac{1}{36}h\sim \\ \frac{1}{144}h\sim \\ \frac{1}{36}h\sim \end{bmatrix} \\
\begin{bmatrix} -\frac{11}{288}h\sim \\ -\frac{1}{96}h\sim \\ -\frac{5}{288}h\sim \\ -\frac{5}{288}h\sim \end{bmatrix} & \begin{bmatrix} \frac{1}{48}h\sim \\ \frac{1}{144}h\sim \\ \frac{1}{144}h\sim \\ \frac{1}{144}h\sim \end{bmatrix} & \begin{bmatrix} -\frac{5}{288}h\sim \\ -\frac{1}{288}h\sim \\ -\frac{1}{96}h\sim \\ -\frac{1}{96}h\sim \end{bmatrix} & \begin{bmatrix} \frac{5}{144}h\sim \\ \frac{1}{144}h\sim \\ \frac{1}{48}h\sim \\ \frac{1}{48}h\sim \end{bmatrix} \\
\begin{bmatrix} -\frac{11}{288}h\sim \\ -\frac{1}{24}h\sim \\ -\frac{5}{288}h\sim \\ -\frac{5}{72}h\sim \end{bmatrix} & \begin{bmatrix} \frac{1}{48}h\sim \\ \frac{1}{36}h\sim \\ \frac{1}{144}h\sim \\ \frac{1}{36}h\sim \end{bmatrix} & \begin{bmatrix} -\frac{5}{288}h\sim \\ -\frac{1}{72}h\sim \\ -\frac{1}{96}h\sim \\ -\frac{1}{24}h\sim \end{bmatrix} & \begin{bmatrix} \frac{5}{144}h\sim \\ \frac{1}{36}h\sim \\ \frac{1}{48}h\sim \\ \frac{1}{12}h\sim \end{bmatrix}
\end{bmatrix}$$

(b) Elementtyp B

$$\begin{bmatrix}
\begin{bmatrix} -\frac{1}{96}h\sim \\ -\frac{1}{96}h\sim \\ -\frac{11}{576}h\sim \\ -\frac{1}{576}h\sim \end{bmatrix} & \begin{bmatrix} \frac{1}{48}h\sim \\ \frac{1}{48}h\sim \\ \frac{11}{288}h\sim \\ \frac{1}{288}h\sim \end{bmatrix} & \begin{bmatrix} -\frac{1}{72}h\sim \\ -\frac{1}{72}h\sim \\ -\frac{1}{32}h\sim \\ -\frac{1}{288}h\sim \end{bmatrix} & \begin{bmatrix} \frac{1}{288}h\sim \\ \frac{1}{288}h\sim \\ \frac{7}{576}h\sim \\ \frac{1}{576}h\sim \end{bmatrix} \\
\begin{bmatrix} -\frac{1}{96}h\sim \\ -\frac{1}{24}h\sim \\ -\frac{7}{288}h\sim \\ -\frac{1}{144}h\sim \end{bmatrix} & \begin{bmatrix} \frac{1}{48}h\sim \\ \frac{1}{12}h\sim \\ \frac{7}{144}h\sim \\ \frac{1}{72}h\sim \end{bmatrix} & \begin{bmatrix} -\frac{1}{72}h\sim \\ -\frac{1}{18}h\sim \\ -\frac{1}{24}h\sim \\ -\frac{1}{72}h\sim \end{bmatrix} & \begin{bmatrix} \frac{1}{288}h\sim \\ \frac{1}{72}h\sim \\ \frac{5}{288}h\sim \\ \frac{1}{144}h\sim \end{bmatrix} \\
\begin{bmatrix} -\frac{11}{576}h\sim \\ -\frac{7}{288}h\sim \\ -\frac{5}{96}h\sim \\ -\frac{5}{576}h\sim \end{bmatrix} & \begin{bmatrix} \frac{11}{288}h\sim \\ \frac{7}{144}h\sim \\ \frac{5}{48}h\sim \\ \frac{5}{288}h\sim \end{bmatrix} & \begin{bmatrix} -\frac{1}{32}h\sim \\ -\frac{1}{24}h\sim \\ -\frac{7}{48}h\sim \\ -\frac{1}{32}h\sim \end{bmatrix} & \begin{bmatrix} \frac{7}{576}h\sim \\ \frac{5}{288}h\sim \\ \frac{3}{32}h\sim \\ \frac{13}{576}h\sim \end{bmatrix} \\
\begin{bmatrix} -\frac{1}{576}h\sim \\ -\frac{1}{144}h\sim \\ -\frac{5}{576}h\sim \\ -\frac{1}{288}h\sim \end{bmatrix} & \begin{bmatrix} \frac{1}{288}h\sim \\ \frac{1}{72}h\sim \\ \frac{5}{288}h\sim \\ \frac{1}{144}h\sim \end{bmatrix} & \begin{bmatrix} -\frac{1}{288}h\sim \\ -\frac{1}{72}h\sim \\ -\frac{1}{32}h\sim \\ -\frac{1}{72}h\sim \end{bmatrix} & \begin{bmatrix} \frac{1}{576}h\sim \\ \frac{1}{144}h\sim \\ \frac{13}{576}h\sim \\ \frac{1}{96}h\sim \end{bmatrix}
\end{bmatrix}$$

(c) Elementtyp C

Abbildung A.3. Matrix `C2koeff`

$$\left\|\begin{array}{cccc}
\begin{bmatrix} -\frac{1}{12}h\sim \\ -\frac{1}{36}h\sim \\ -\frac{1}{24}h\sim \\ -\frac{1}{72}h\sim \end{bmatrix} & \begin{bmatrix} -\frac{1}{36}h\sim \\ -\frac{1}{36}h\sim \\ -\frac{1}{72}h\sim \\ -\frac{1}{72}h\sim \end{bmatrix} & \begin{bmatrix} \frac{1}{12}h\sim \\ \frac{1}{36}h\sim \\ \frac{1}{24}h\sim \\ \frac{1}{72}h\sim \end{bmatrix} & \begin{bmatrix} \frac{1}{36}h\sim \\ \frac{1}{36}h\sim \\ \frac{1}{72}h\sim \\ \frac{1}{72}h\sim \end{bmatrix} \\
\begin{bmatrix} -\frac{1}{36}h\sim \\ -\frac{1}{36}h\sim \\ -\frac{1}{72}h\sim \\ -\frac{1}{72}h\sim \end{bmatrix} & \begin{bmatrix} -\frac{1}{36}h\sim \\ -\frac{1}{12}h\sim \\ -\frac{1}{72}h\sim \\ -\frac{1}{24}h\sim \end{bmatrix} & \begin{bmatrix} \frac{1}{36}h\sim \\ \frac{1}{36}h\sim \\ \frac{1}{72}h\sim \\ \frac{1}{72}h\sim \end{bmatrix} & \begin{bmatrix} \frac{1}{36}h\sim \\ \frac{1}{12}h\sim \\ \frac{1}{72}h\sim \\ \frac{1}{24}h\sim \end{bmatrix} \\
\begin{bmatrix} -\frac{1}{24}h\sim \\ -\frac{1}{72}h\sim \\ -\frac{1}{12}h\sim \\ -\frac{1}{36}h\sim \end{bmatrix} & \begin{bmatrix} -\frac{1}{72}h\sim \\ -\frac{1}{72}h\sim \\ -\frac{1}{36}h\sim \\ -\frac{1}{36}h\sim \end{bmatrix} & \begin{bmatrix} \frac{1}{24}h\sim \\ \frac{1}{72}h\sim \\ \frac{1}{12}h\sim \\ \frac{1}{36}h\sim \end{bmatrix} & \begin{bmatrix} \frac{1}{72}h\sim \\ \frac{1}{72}h\sim \\ \frac{1}{36}h\sim \\ \frac{1}{36}h\sim \end{bmatrix} \\
\begin{bmatrix} -\frac{1}{72}h\sim \\ -\frac{1}{72}h\sim \\ -\frac{1}{36}h\sim \\ -\frac{1}{36}h\sim \end{bmatrix} & \begin{bmatrix} -\frac{1}{72}h\sim \\ -\frac{1}{24}h\sim \\ -\frac{1}{36}h\sim \\ -\frac{1}{12}h\sim \end{bmatrix} & \begin{bmatrix} \frac{1}{72}h\sim \\ \frac{1}{72}h\sim \\ \frac{1}{36}h\sim \\ \frac{1}{36}h\sim \end{bmatrix} & \begin{bmatrix} \frac{1}{72}h\sim \\ \frac{1}{24}h\sim \\ \frac{1}{36}h\sim \\ \frac{1}{12}h\sim \end{bmatrix}
\end{array}\right\|$$

(a) Elementtyp A

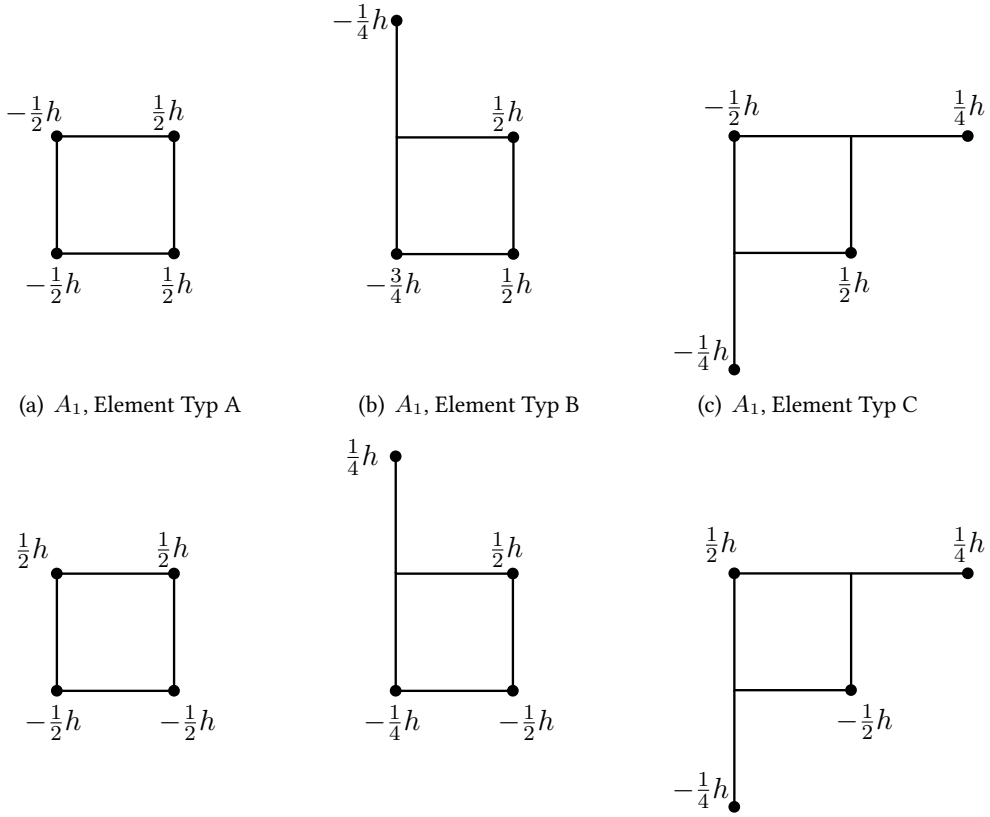
$$\left\|\begin{array}{cccc}
\begin{bmatrix} -\frac{7}{96}h\sim \\ -\frac{5}{288}h\sim \\ -\frac{1}{48}h\sim \\ -\frac{1}{72}h\sim \end{bmatrix} & \begin{bmatrix} -\frac{7}{144}h\sim \\ -\frac{5}{144}h\sim \\ -\frac{1}{72}h\sim \\ -\frac{1}{36}h\sim \end{bmatrix} & \begin{bmatrix} \frac{7}{96}h\sim \\ \frac{5}{288}h\sim \\ \frac{1}{48}h\sim \\ \frac{1}{72}h\sim \end{bmatrix} & \begin{bmatrix} \frac{7}{144}h\sim \\ \frac{5}{144}h\sim \\ \frac{1}{72}h\sim \\ \frac{1}{36}h\sim \end{bmatrix} \\
\begin{bmatrix} -\frac{5}{288}h\sim \\ -\frac{1}{72}h\sim \\ -\frac{1}{288}h\sim \\ -\frac{1}{144}h\sim \end{bmatrix} & \begin{bmatrix} -\frac{5}{144}h\sim \\ -\frac{1}{12}h\sim \\ -\frac{1}{144}h\sim \\ -\frac{1}{24}h\sim \end{bmatrix} & \begin{bmatrix} \frac{5}{288}h\sim \\ \frac{1}{72}h\sim \\ \frac{1}{288}h\sim \\ \frac{1}{144}h\sim \end{bmatrix} & \begin{bmatrix} \frac{5}{144}h\sim \\ \frac{1}{12}h\sim \\ \frac{1}{144}h\sim \\ \frac{1}{24}h\sim \end{bmatrix} \\
\begin{bmatrix} -\frac{1}{48}h\sim \\ -\frac{1}{288}h\sim \\ -\frac{1}{96}h\sim \\ -\frac{1}{144}h\sim \end{bmatrix} & \begin{bmatrix} -\frac{1}{72}h\sim \\ -\frac{1}{144}h\sim \\ -\frac{1}{144}h\sim \\ -\frac{1}{72}h\sim \end{bmatrix} & \begin{bmatrix} \frac{1}{48}h\sim \\ \frac{1}{288}h\sim \\ \frac{1}{96}h\sim \\ \frac{1}{144}h\sim \end{bmatrix} & \begin{bmatrix} \frac{1}{72}h\sim \\ \frac{1}{144}h\sim \\ \frac{1}{144}h\sim \\ \frac{1}{72}h\sim \end{bmatrix} \\
\begin{bmatrix} -\frac{1}{72}h\sim \\ -\frac{1}{144}h\sim \\ -\frac{1}{144}h\sim \\ -\frac{1}{72}h\sim \end{bmatrix} & \begin{bmatrix} -\frac{1}{36}h\sim \\ -\frac{1}{24}h\sim \\ -\frac{1}{72}h\sim \\ -\frac{1}{12}h\sim \end{bmatrix} & \begin{bmatrix} \frac{1}{72}h\sim \\ \frac{1}{144}h\sim \\ \frac{1}{144}h\sim \\ \frac{1}{72}h\sim \end{bmatrix} & \begin{bmatrix} \frac{1}{36}h\sim \\ \frac{1}{24}h\sim \\ \frac{1}{72}h\sim \\ \frac{1}{12}h\sim \end{bmatrix}
\end{array}\right\|$$

(b) Elementtyp B

$$\left\|\begin{array}{cccc}
\begin{bmatrix} -\frac{1}{96}h\sim \\ -\frac{1}{144}h\sim \\ -\frac{13}{576}h\sim \\ -\frac{1}{576}h\sim \end{bmatrix} & \begin{bmatrix} -\frac{1}{144}h\sim \\ -\frac{1}{72}h\sim \\ -\frac{5}{288}h\sim \\ -\frac{1}{288}h\sim \end{bmatrix} & \begin{bmatrix} \frac{1}{72}h\sim \\ \frac{1}{72}h\sim \\ \frac{1}{32}h\sim \\ \frac{1}{288}h\sim \end{bmatrix} & \begin{bmatrix} \frac{1}{288}h\sim \\ \frac{1}{144}h\sim \\ \frac{5}{576}h\sim \\ \frac{1}{576}h\sim \end{bmatrix} \\
\begin{bmatrix} -\frac{1}{144}h\sim \\ -\frac{1}{72}h\sim \\ -\frac{5}{288}h\sim \\ -\frac{1}{288}h\sim \end{bmatrix} & \begin{bmatrix} -\frac{1}{72}h\sim \\ -\frac{1}{12}h\sim \\ -\frac{7}{144}h\sim \\ -\frac{1}{48}h\sim \end{bmatrix} & \begin{bmatrix} \frac{1}{72}h\sim \\ \frac{1}{18}h\sim \\ \frac{1}{24}h\sim \\ \frac{1}{72}h\sim \end{bmatrix} & \begin{bmatrix} \frac{1}{144}h\sim \\ \frac{1}{24}h\sim \\ \frac{7}{288}h\sim \\ \frac{1}{96}h\sim \end{bmatrix} \\
\begin{bmatrix} -\frac{13}{576}h\sim \\ -\frac{5}{288}h\sim \\ -\frac{3}{32}h\sim \\ -\frac{7}{576}h\sim \end{bmatrix} & \begin{bmatrix} -\frac{5}{288}h\sim \\ -\frac{7}{144}h\sim \\ -\frac{5}{48}h\sim \\ -\frac{11}{288}h\sim \end{bmatrix} & \begin{bmatrix} \frac{1}{32}h\sim \\ \frac{1}{24}h\sim \\ \frac{7}{48}h\sim \\ \frac{1}{32}h\sim \end{bmatrix} & \begin{bmatrix} \frac{5}{576}h\sim \\ \frac{7}{288}h\sim \\ \frac{5}{96}h\sim \\ \frac{11}{576}h\sim \end{bmatrix} \\
\begin{bmatrix} -\frac{1}{576}h\sim \\ -\frac{1}{288}h\sim \\ -\frac{7}{576}h\sim \\ -\frac{1}{288}h\sim \end{bmatrix} & \begin{bmatrix} -\frac{1}{288}h\sim \\ -\frac{1}{48}h\sim \\ -\frac{11}{288}h\sim \\ -\frac{1}{48}h\sim \end{bmatrix} & \begin{bmatrix} \frac{1}{288}h\sim \\ \frac{1}{72}h\sim \\ \frac{1}{32}h\sim \\ \frac{1}{72}h\sim \end{bmatrix} & \begin{bmatrix} \frac{1}{576}h\sim \\ \frac{1}{96}h\sim \\ \frac{11}{576}h\sim \\ \frac{1}{96}h\sim \end{bmatrix}
\end{array}\right\|$$

(c) Elementtyp C

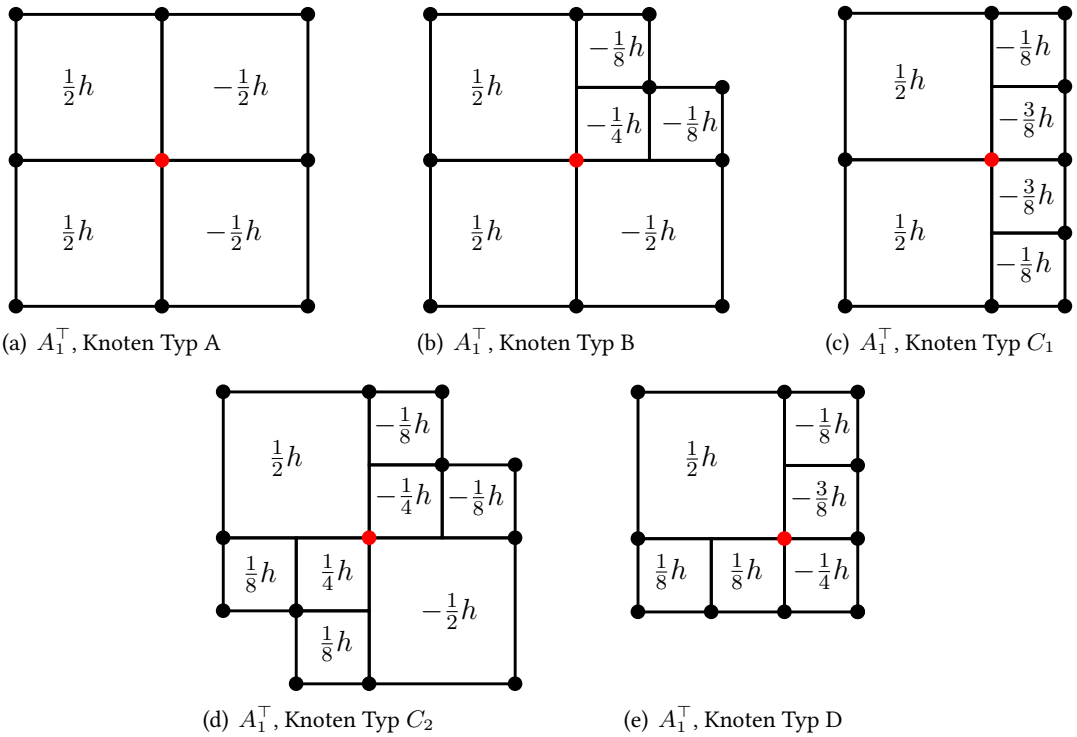
Abbildung A.4. Differenzensterne für die Operatoren A_1, A_2 für die verschiedenen Elementtypen



(a) A_1, Element Typ A (b) A_1, Element Typ B (c) A_1, Element Typ C

(d) A_2, Element Typ A (e) A_2, Element Typ B (f) A_2, Element Typ C

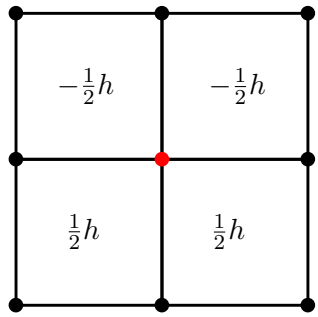
Abbildung A.5. Differenzensterne für den Operator $A_1^\top$ für die verschiedenen Knotentypen



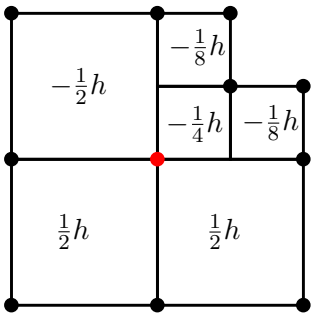
(a) $A_1^\top$, Knoten Typ A (b) $A_1^\top$, Knoten Typ B (c) $A_1^\top$, Knoten Typ C_1

(d) $A_1^\top$, Knoten Typ C_2 (e) $A_1^\top$, Knoten Typ D

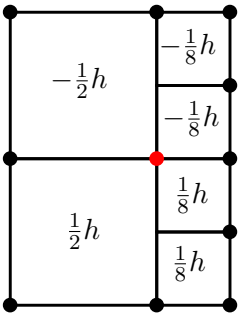
Abbildung A.6. Differenzensterne für den Operator $A_2^\top$ für die verschiedenen Knotentypen



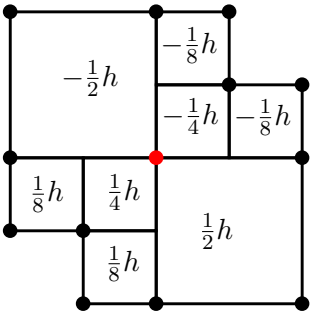
(a) $A_2^\top$, Knoten Typ A



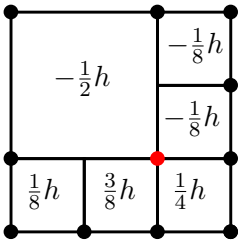
(b) $A_2^\top$, Knoten Typ B



(c) $A_2^\top$, Knoten Typ C_1



(d) $A_2^\top$, Knoten Typ C_2



(e) $A_2^\top$, Knoten Typ D

B. Parameter

Im Folgenden sind alle Parameter der Eingabedatei mit Typ, Standardwert und Beschreibung aufgelistet. Bei dem Typ *bool* wird als Wert 0 oder 1 erwartet, nicht *true* oder *false*.

Bezeichner	Typ	Standard-wert	Beschreibung
`scenario`	int	3	Art, wie die Geometrie erzeugt wird. 1,2 = Beispiele, 3 = reguläres Gitter, 4 = zufällige Punkte, 5 = Einlesen der Vektorgrafik
`d_max`	int	5	maximaler Elementlevel
`d_min`	int	0	minimaler Elementlevel
`domain_width`	double	1.0	Breite des Berechnungsgebiets
`reynolds_number`	double	1000.0	Reynoldszahl
`omega`	double	1.4	Überrelaxationsfaktor ω für SOR–Löser
`epsilon`	double	1e-15	Fehlertoleranz des Residuum für iterative Löser (SOR und CG)
`epsilon_progress`	double	0	Wert, um den das Residuum sich mindestens pro Iteration (bei SOR und CG) verkleinern muss, damit der Lösungsprozess nicht abgebrochen wird. 0=deaktiviert
`max_iterations`	int	10000	Maximale Iterationsanzahl (SOR und CG)
`delta_t`	double	0.0	Zeitschrittweite δt, 0 = adaptiv
`t_end`	double	20.0	Endzeitpunkt
`delta_t_output`	double	0.01	Intervall für das Schreiben der Ausgabedateien
`delta_t_safety_factor`	double	0.5	Sicherheitsfaktor α, der bei adaptiver Berechnung von δt aufmultipliziert wird
`navier_stokes`	bool	1	1 = Navier–Stokes–Gleichungen werden berechnet, 0 = Stokes–Gleichungen
`exit_on_max_iter`	bool	0	Ob die Simulation abgebrochen wird, falls der Löser in einem Zeitschritt die maximale Iterationszahl erreicht hat
`create_mesh_and_exit`	bool	0	1 = Das Programm liest die Geometrie ein, schreibt die *.svg–Ausgabedateien und beendet, 0 = Kompletter Ablauf
`enable_duration_log`	bool	1	Erzeugt eine Datei `log.txt`, die Informationen über die Laufzeit enthält.
`enable_continuity_log`	bool	1	Erzeugt eine Datei `log_continuity.txt`, die Informationen über die Erfüllung der Kontinuitätsgleichung enthält.

enable_load_balancing_log	bool	0	Erzeugt eine Datei log_load_balancing.txt, die Information über die Lastbalancierung und die Verteilung von Knoten und Elementen auf die Prozesse enthält.
settings_file	string	settings.txt	Dateiname der Eingabedatei, deshalb ist dies kein Parameter innerhalb der Eingabedatei. Es wird auch in den Ordnern bis drei Level über dem aktuellen Ordner gesucht.
verbose	bool	0	Zusätzliche Ausgabe der Löser, z. B. Residuum in jedem Iterationsschritt. Wenn gesetzt, hebt es die Wirkung des -v Kommandozeilenparameters auf
force_without_conti	bool	1	Die Version des Algorithmus, die die Kontinuitätsgleichung im vorigen Schritt nicht erfordert wird immer verwendet.
force_with_conti	bool	0	Die Version des Algorithmus, die die Kontinuitätsgleichung im vorigen Schritt erfordert wird immer verwendet. Wenn keine der beiden Switches gesetzt sind, wird immer diese Version verwendet, bis auf alle zehn Schritte
solver_type	int	1	Auswahl des Lösers, 0 = SOR–Verfahren (iterativ), 1 = CG–Verfahren (iterativ), 2 = komplexwertige Cholesky–Zerlegung (direkt), 3 = LU Zerlegung mit Pivotisierung (direkt)
cg_enable_pressure_bc	bool	1	Ob Druckrandbedingungen beim Lösen berücksichtigt werden
cg_start_with_0_pressure	bool	0	Ob der Druck zu Beginn des jedes Lösungsprozesses auf 0 gesetzt wird
cg_start_with_random_pressure	bool	0	Ob der Druck zu Beginn des jedes Lösungsprozesses auf zufällige Startwerte gesetzt wird
cg_output_matrix	bool	0	Ausgabe der Systemmatrix, rechten Seite und Lösung in *.csv–Dateien (Achtung, dauert bei großen Systemen lange)
cg_debug_constant_rhs	bool	0	Ob die rechte Seite auf konstant Null gesetzt wird (zum Debugging)
outflow_with_pressure_dirichlet	bool	1	Ob bei Geschwindigkeits–*outflow*–R.B. der Druck auf Null gesetzt wird. (Falls nicht, gibt es evtl. Konvergenzschwierigkeiten)
debug_border_coarser	bool	1	Ob alle Pfade der Vektorgraphik bis auf den ersten halb so fein diskretisiert werden
input_file	string		Dateiname der svg–Vektorgraphik,
complete_quadtree	bool	0	Ob der komplette Quadtree gespeichert bleibt und die Bereiche außerhalb der Geometrie nicht gelöscht werden sollen (nicht empfohlen)
sample_width_border	double	0.01	Sample–Länge des Rands der Geometrie, 0 = deaktiviert
sample_width_interior	double	0.01	Sample–Länge des Inneren der Geometrie, 0 = deaktiviert

inverse_sample_width_border	double	-	Inverse sample–Länge des Rands der Geometrie, d.h. Anzahl der Punkte auf dem Rand pro Breite des Gebiets, 0 = deaktiviert
inverse_sample_width_interior	double	-	Inverse sample–Länge des Inneren der Geometrie, 0 = deaktiviert
output_complete_quadtree	bool	1	Ob der komplette Quadtree nach Erzeugen in complete_quadtree.**svg** gespeichert wird
output_svg	bool	1	Ob eine <input_file>_out.**svg** Datei angelegt wird, die die Geometrie enthält
disable_file_output	bool	0	Ob die Ausgabe von svg–Dateien mit Quadtrees und vtk–Dateien mit Werten deaktiviert wird (deaktiviert output_svg, output_complete_quadtree)
combine_vtu	bool	1	Ob die Ausgabedateien *.vtu und *.pvtu bei paralleler Ausführung zu einer Datei *.vtu kombiniert werden
refine_interval	bool	0	Zeitintervall, nach dem das Finite–Elemente–Gitter verfeinert oder vergröbert wird
vorticity_max_threshold	double	0.1	Maximale vorticity, so dass ein Element nicht verfeinert wird
vorticity_min_threshold	double	0.01	Minimale vorticity, so dass ein Element nicht verfeinert wird, d.h. ein Element wird genau dann nicht verfeinert, wenn min < |vorticity| < max
smooth_dirichlet_bc	bool	1	Setzt *inflow* Randbedingungen neben *noslip* auf den halben Wert (empfohlen)
disable_outflow_bc	bool	0	deaktiviert das Kopieren von Geschwindigkeitswerten zur Realisierung von *outflow*–Rändern
use_global_bc	bool	1	Wahl der Randbedingungsart, 0 = Randbedingungen so wie in der Vektorgraphik, 1 = Randbedingungen am äußeren Rand des quadratischen Rechengebiets, Werte durch boundary_*_dirichlet_*
bc_velocity_dirichlet_u	double	1.0	*inflow*–Rand Wert für Geschwindigkeit in x_1–Richtung (nur falls use_global_bc=0)
bc_velocity_dirichlet_v	double	0.0	*inflow*–Rand Wert für Geschwindigkeit in x_2–Richtung (nur falls use_global_bc=0)
bc_pressure_dirichlet_1	double	0.0	Wert der ersten (gelb) Dirchlet–Randbedingung für den Druck (nur falls use_global_bc=0)
bc_pressure_dirichlet_2	double	1.0	Wert der zweiten (orange) Dirchlet–Randbedingung für den Druck (nur falls use_global_bc=0)
boundary_left_dirichlet_u	double	0.0	Wert der Dirchletrandbedingung am linken Rand in x_1–Richtung (nur falls use_global_bc=1)
boundary_left_dirichlet_v	double	0.0	Wert der Dirchletrandbedingung am linken Rand in x_2–Richtung (nur falls use_global_bc=1)

boundary_right_dirichlet_u	double	0.0	Wert der Dirchletrandbedingung am rechten Rand in x_1–Richtung (nur falls use_global_bc=1)
boundary_right_dirichlet_v	double	0.0	Wert der Dirchletrandbedingung am rechten Rand in x_2–Richtung (nur falls use_global_bc=1)
boundary_top_dirichlet_u	double	1.0	Wert der Dirchletrandbedingung am oberen Rand in x_1–Richtung (nur falls use_global_bc=1)
boundary_top_dirichlet_v	double	0.0	Wert der Dirchletrandbedingung am oberen Rand in x_2–Richtung (nur falls use_global_bc=1)
boundary_bottom_dirichlet_u	double	0.0	Wert der Dirchletrandbedingung am unteren Rand in x_1–Richtung (nur falls use_global_bc=1)
boundary_bottom_dirichlet_v	double	0.0	Wert der Dirchletrandbedingung am unteren Rand in x_2–Richtung (nur falls use_global_bc=1)
output_element_numbers	bool	0	Ob in der svg–Datei Elementnummern enthalten sind
output_element_ids	bool	0	Ob in der svg–Datei Element–IDs (Morton–IDs) enthalten sind
output_node_numbers	bool	0	Ob in der svg–Datei Knotennummern enthalten sind
output_subelement_numbers	bool	0	Ob in der svg–Datei pro Element 9 Nummern enthalten sind, die den Algorithmus zur Bestimmung, ob sich das Element innerhalb befindet, beschreiben
output_bc_text	bool	1	Ob die Randbedingungen mit Text versehen werden
output_geometry_discretization	bool	0	Ob die Samplingpunkte auf den Pfaden angezeigt werden
output_seed_points	bool	0	Ob die Samplingpunkte im Inneren angezeigt werden
output_entrance_exit_points	bool	0	Ob Schnittpunkte der Pfade mit den Elementgrenzen angezeigt werden
output_process_color	bool	1	Ob die Elemente nach Prozesszugehörigkeit gefärbt werden
output_rhs	bool	0	Ob bei Ausgabe von Werten auch die rechte Seite ausgegeben wird (gilt nur wenn output_ab=1)
output_ab	bool	0	Ob bei Ausgabe von Werten a und b ausgegeben werden
open_svg_in_inkscape	bool	0	Ob nach Erzeugen die Datei complete_quadtree.**svg** automatisch mit inkscape geöffnet wird
output_quadtree_svg	bool	0	Ob mehrere svg–Dateien der Quadtrees während der Algorithmus sie erzeugt, angelegt werden
output_laplace_stencil	bool	0	Ob die Differenzensterne des Laplace–Operators generiert werden (nur für sehr kleine Probleme sinnvoll)
output_bc_markers	bool	1	Ob in den svg–Dateien die Randbedingungen markiert werden sollen

node_font_size	int	10	Schriftgröße der Knotennummern
element_number_font_size	int	13	Schriftgröße der Elementnummern
nodal_value_font_size	int	8	Schriftgröße der Knotenwerte (Geschwindigkeiten)
elemental_value_font_size	int	10	Schriftgröße der Elementwerte (Druck und rechte Seite)
stroke_width	double	0.25	Liniendicke der Elemente in der svg–Datei

Literaturverzeichnis

[And80] H. C. Andersen. Molecular dynamics simulations at constant pressure and/or temperature. *The Journal of chemical physics*, 72(4):2384–2393, 1980. (Zitiert auf Seite 9)

[AO82] L. Adams, J. Ortega. A multi-color SOR method for parallel computation. In *ICPP*, S. 53–56. Citeseer, 1982. (Zitiert auf Seite 51)

[Bra13] D. Braess. *Finite Elemente: Theorie, schnelle Löser und Anwendungen in der Elastizitätstheorie*. Springer-Verlag, 2013. (Zitiert auf den Seiten 23 und 24)

[CD98] S. Chen, G. D. Doolen. Lattice Boltzmann method for fluid flows. *Annual review of fluid mechanics*, 30(1):329–364, 1998. (Zitiert auf Seite 9)

[CP85] R. Car, M. Parrinello. Unified approach for molecular dynamics and density-functional theory. *Physical review letters*, 55(22):2471, 1985. (Zitiert auf Seite 9)

[DH03] J. Donea, A. Huerta. *Finite element methods for flow problems*. Wiley, Chichester, 2003. URL http://swbplus.bsz-bw.de/bsz105966207cov.htm. (Zitiert auf den Seiten 27 und 78)

[ELR+07] B. Engquist, X. Li, W. Ren, E. Vanden-Eijnden, et al. Heterogeneous multiscale methods: a review. *Communications in Computational Physics*, 2(3):367–450, 2007. (Zitiert auf Seite 9)

[Fef00] C. L. Fefferman. Existence and smoothness of the Navier-Stokes equation. *The millennium prize problems*, S. 57–67, 2000. (Zitiert auf Seite 9)

[SSA+07] H. Sundar, R. S. Sampath, S. S. Adavani, C. Davatzikos, G. Biros. Low-constant parallel algorithms for finite element simulations using linear octrees. In *Proceedings of the 2007 ACM/IEEE conference on Supercomputing*, S. 25. ACM, 2007. (Zitiert auf den Seiten 11, 14, 16 und 18)

[SSB08] H. Sundar, R. S. Sampath, G. Biros. Bottom-up construction and 2: 1 balance refinement of linear octrees in parallel. *SIAM Journal on Scientific Computing*, 30(5):2675–2708, 2008. (Zitiert auf den Seiten 3, 17 und 77)

[W3C] W3C SVG Working Group Wiki - Implementations. URL http://www.w3.org/Graphics/SVG/WG/wiki/Implementations. (Zitiert auf Seite 37)

[Wan01] W. Wang. Special bilinear quadrilateral elements for locally refined finite element grids. *SIAM Journal on Scientific Computing*, 22(6):2029–2050, 2001. (Zitiert auf Seite 19)

[Zie79] J. Zierep. *Grundzüge der Strömungslehre*. Springer, 1979. (Zitiert auf Seite 23)

Alle URLs wurden zuletzt am 01.08.2015 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