Filterung von 3D-Neuronenaufnahmen durch nichtlineare anisotrope Diffusion

Diplomarbeit
in Mathematik
mit Ausrichtung
wissenschaftliches Rechnen

von
Roland Schulte

Betreuer: Prof. Dr. Gabriel Wittum

Ruprechts-Karls-Universität Heidelberg
Fakultät für Mathematik und Informatik

August 2003

Hvis menneskets hjerne var så enkel at vi kunne forstå den, da ville vi vært så dumme at vi ikke forstod den likevel.

Wenn das Gehirn des Menschen so einfach wäre, dass wir es verstehen könnten, dann wären wir so dumm, dass wir es trotzdem nicht verstehen würden.

Jostein Gaarder: *Sofies verden*

Inhaltsverzeichnis

1 Einleitung

Das menschliche Gehirn ist die wohl komplexeste Ansammlung von Materie auf dieser Erde. Aufgebaut aus einem gewaltigen Netzwerk von untereinander kommunizierenden Nervenzellen, den sogenannten Neuronen, ist es zusammen mit dem Rückenmark und dem peripheren Nervensystem für alle Aspekte der Wahrnehmung und Interaktion mit der Umwelt verantwortlich. Doch auch Lernvorgänge, Erinnerung und das Bewusstwerden des Menschen seiner selbst werden durch das Gehirn geleistet.
Gerade seiner Komplexität wegen sind viele Funktionen des Gehirns jedoch bis heute noch unverstanden. Die Erforschung von Träumen und Schlaf hält mehr Rätsel bereit, als sie bislang lösen konnte. Das Verständnis von Krankheiten wie Alzheimer, Creutzfeldt-Jacob oder Epilepsie steht erst am Anfang. Auch Vorgänge des Lernens und der Erinnerung stellen viele unbeantwortete Fragen.

Um die Funktionsweise des Gehirns zu verstehen, ist unter anderem ein Verständnis der aus den Neuronen aufgebauten Netzwerke und der Kommunikation innerhalb dieser sowie eine genauere Klärung des Zusammenhangs zwischen Morphologie und Funktion von Neuronen erforderlich. Einen wesentlichen Fortschritt stellt in diesem Zusammenhang die Entwicklung neuerer hochauflösender Mikroskope dar. Dabei kommt der konfokalen Zwei-Photonen-Mikroskopie eine besondere Bedeutung zu, da mit ihrer Hilfe dreidimensionale Aufnahmen von lebenden Nervenzellen möglich sind.
Neben den Beobachtungen und Experimenten an lebenden Zellen interessiert man sich - angeregt durch die rasche Entwicklung der Möglichkeiten numerischer Computersimulationen - auch für eine realistische Simulation der Informationsausbreitung auf den Neuronen und den durch sie gebildeten Netzwerken. Diese Informations- oder auch Signalleitung geschieht interzellulär mittels chemischer, intrazellulär mittels elektrischer Signale.
Aus den mikroskopisch gewonnen Aufnahmen kann man die Geometrie eines Neurons rekonstruieren. Bislang geschieht das noch manuell mit Unterstützung spezieller graphischer Software. Auf den rekonstruierten Neuronen lässt sich dann die elektrische Signalausbreitung simulieren. Dabei werden bislang jedoch nur eindimensionale Geometrien verwendet; es wird die Leitung von elektrischen Signalen auf im Raum angeordneten durchmesserlosen Kabeln mit Verzweigungen simuliert.

Unbefriedigend an diesem Vorgehen ist die Tatsache, dass die Signalleitung auf der realen Geometrie des Neurons ein anderes Verhalten zeigen wird als auf den eindimensionalen Geometrien. Das ist allein schon an der Tatsache ersichtlich, dass der elektische Widerstand R antiproportional zum Durchmesser des den Strom leitenden Kabels ist. Dieser Durchmesser dürfte also nicht vernachlässigt werden.
Weiter ist die manuelle Rekonstruktion der Neuronen einerseits sehr zeitintensiv, andererseits fehleranfällig und nicht unbedingt reproduzierbar. Sie eignet sich da-

mit auch schlecht für eine Klassifizierung der Morphologie und der Untersuchung von deren Zusammenhang mit der Funktion von Neuronen, bei der eine große Zahl von diesen rekonstruiert werden muss, und zwar mit einem vom Rekonstrukteur unabhängigen Ergebnis. Auch kann es wie bei der Erforschung von Lernvorgängen nötig sein Neuronen in Echtzeit zu rekonstruieren, um Veränderungen in ihrer Struktur zu beobachten.

In Zusammenarbeit mit dem Max-Plank-Institut für medizinische Forschung (MPI) wurde daher am Interdisziplinären Institut für wissenschaftliches Rechnen (IWR) der Universität Heidelberg ein Projekt ins Leben gerufen, das die Entwicklung eines Algorithmus zur automatischen 3D-Rekonstruktion von Neuronen aus mikroskopisch gewonnenen Datensätzen zum Ziel hat. Im ersten Schritt soll dabei erst einmal nur die eindimensionale Geometrie erzeugt, im zweiten darauf aufbauend dann die reale Geometrie extrahiert und später in einem weiteren Schritt die Signalausbreitung auf dieser simuliert werden. Das Fernziel ist die Simulation kleinerer Netzwerke.

Allein der erste Schritt unterteilt sich in die folgenden vier Teilschritte:

Filterung: Entfernung des durch die Aufnahme entstandenen Rauschens bei gleichzeitiger Erhaltung oder sogar Verstärkung der aufgenommenen Struktur, sowie Schließung von Strukturlücken, soweit es möglich ist.

Segmentierung: Auftrennung des Bildes in Struktur und Hintergrund.

Skelettierung: Markierung von Endpunkten der Struktur und Reduzierung ihrer Breite auf einen Voxel.

Graphrekonstruktion: Generierung eines Graphen im mathematischen Sinne. Als Graphenknoten dienen Endpunkte, Verzweigungspunkte und Zellkörper.

Die letzten beiden Schritte wollen wir auch einfach unter dem Begriff Rekonstruktion zusammenfassen.

Die Filterung der Aufnahmen vor der eigentlichen Rekonstruktion ist zwingend erforderlich, da man ohne sie keine brauchbaren Ergebnisse erhält. Die durch die Mikroskopie gewonnenen Aufnahmen sind derartig verrauscht, dass Signale teilweise darin verschwinden und daher entweder nicht erkannt werden, oder aber das Rauschen fälschlicherweise zusätzlich als Struktur erkannt wird. Weiter können in den Aufnahmen Unterbrechungen in den Zellstrukturen auftreten, falls diese an einer Stelle durch ein Fremdgewebe, z.B. ein Blutgefäß, überlagert werden. Soweit möglich, sollten diese Lücken durch die Filterung geschlossen werden, da die Rekonstruktion sonst keine zusammenhängende Zellstruktur liefern kann.
Als hervorragend geeignet für diese Aufgabe wird sich die Filterung durch nichtlineare anisotrope Diffusion erweisen. Dabei wird eine Diffusion auf dem Bild

durchgeführt, die in Abhängigkeit der lokalen Struktur nur in bestimmte Richtungen wirkt. So kann Rauschen geglättet werden, ohne die Strukturen zu zerstören, und gleichzeitig eine Schließung von Lücken in der Struktur bewirkt werden. Zur lokalen Bestimmung der Richtungen, in welche die Diffusion wirken soll, wird sich die Benutzung der aus der Physik bekannten Trägheitstensoren als günstig herausstellen.

Das Ziel dieser Arbeit besteht nun darin, einen auf nichtlinearer anisotroper Diffusion beruhenden Filter zu implementieren und in den Rahmen des Gesamtprojektes der Neuronenrekonstruktion zu integrieren, um diese überhaupt zu ermöglichen.

Kurzer Überblick über die einzelnen Kapitel dieser Arbeit:

In Kapitel 2 werde ich den Aufbau von Neuronen und das Prinzip der Zwei-Photonen-Mikroskopie kurz vorstellen.
Kapitel 3 beschäftigt sich mit dem Prinzip der Filterung durch nichtlineare anisotrope Diffusion sowie der Diskretisierung der zugrundeliegenden partiellen Differentialgleichung. Außerdem wird die Verwendung des Trägheitstensors zur lokalen Strukturerkennung erläutert.
Kapitel 4 stellt die Implementierung des Filters vor.
Kapitel 5 zeigt die Ergebnisse der Anwendung des Filters auf die Mikroskopaufnahmen und testet verschiedene Parameter der Implementierung aus.
Kapitel 6 zeigt zuletzt die Wirkung und damit die Notwendigkeit der Filterung für die Rekonstruktion.

2 Mikroskopie von Neuronen

2.1 Aufbau von Neuronen

Das Nervensystem mit all seinen hochkomplexen und in vielen Teilen noch unverstandenen Funktionsweisen ist hauptsächlich aus zwei Zelltypen, *Neuronen* und *Gliazellen*, aufgebaut. Diese werden nun kurz vorgestellt. Zur Darstellung der Signalleitung auf den Neuronen verweisen wir auf Campbell[2].

Neuronen
Die Neuronen sind für die Informationsweiterleitung und -verarbeitung innerhalb des Nervensystems verantwortlich und bilden so die Grundlage allen Erkennens, Denkens und Handelns. Zur Erfüllung dieser Funktionen bilden sie untereinander gewaltige Netzwerke unvergleichlicher Komplexität. In einem einzigen Kubikzentimeter des menschlichen Gehirns befinden sich mehrere Millionen Neuronen. Jedes einzelne davon kann mit tausenden anderen Neuronen verbunden sein.

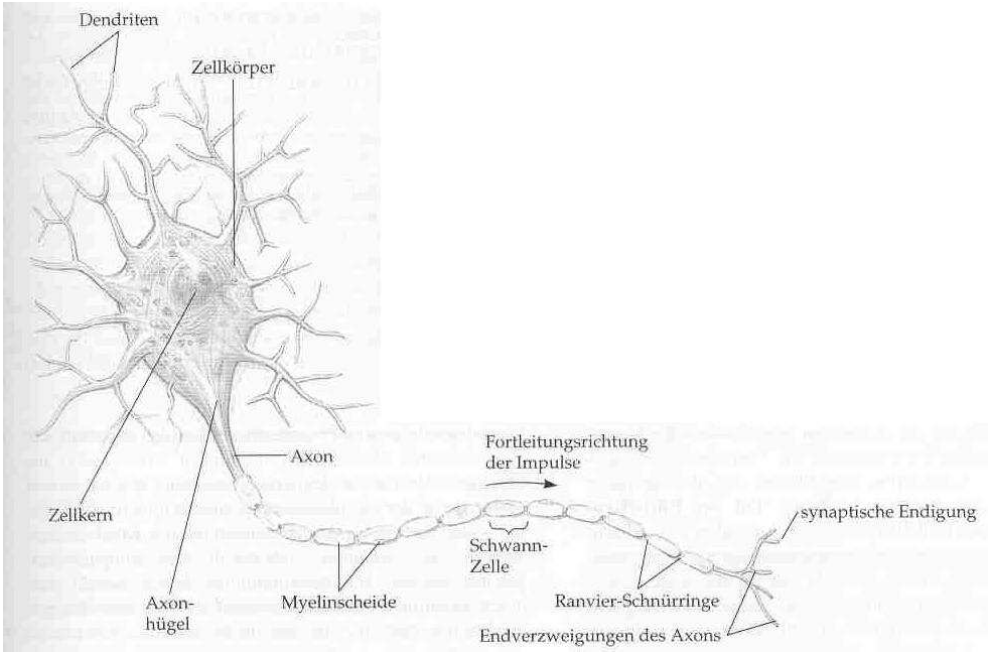

Abbildung 2.1: Aufbau eines Neurons. Entnommen aus Campbell[2]

Neuronen besitzen einen relativ großen Zellkörper, der den Zellkern und die übrigen Zellorganellen enthält. Weiterhin verfügen sie über lange Fortsätze, bei denen man die zwei Typen *Dendriten* und *Axone* unterscheidet.
Dendriten (griech.: *τό δένδρον* - der Baum) leiten eingehende Signale in Richtung Zellkörper. Sie sind zumeist stark verzweigt und vergrößern so die Zelloberfläche. Die Signale werden innerhalb eines Neurons elektrisch durch Veränderung

von Ionenkonzentrationen weitergegeben.
Axone, von denen ein Neuron jeweils höchstens eines besitzt, leiten ausgehende Signale vom Zellkörper weg. Der Bereich des Neurons, an dem das Axon dem Zellkörper entspringt, wird als *Axonhügel* bezeichnet. Axone können sich verzweigen und jede Verzweigung kann hunderte oder tausende sogenannte *synaptische Endigungen* aufweisen. An diesen werden die Signale durch Freisetzung von chemischen Botenstoffen, den *Neurotransmittern*, an eine andere Zelle übertragen und diese so untereinander vernetzt.
Neuronen lassen sich in drei funktionale Gruppen einteilen. Die *Sensorischen Neuronen* übertragen Signale aus den Sinneszellen in das Zentralnervensystem. Die *Interneuronen* interagieren ausschließlich mit anderen Neuronen und die *Motoneuronen* senden Signale vom Zentralnervensystem an Effektorzellen, wie z.B. Muskeln. Durch ihre unterschiedliche Funktion unterscheiden sich die drei Zelltypen hinsichtlich ihrer Morphologie, wobei es auch innerhalb der drei Typen noch weitere Unterscheidungen gibt.

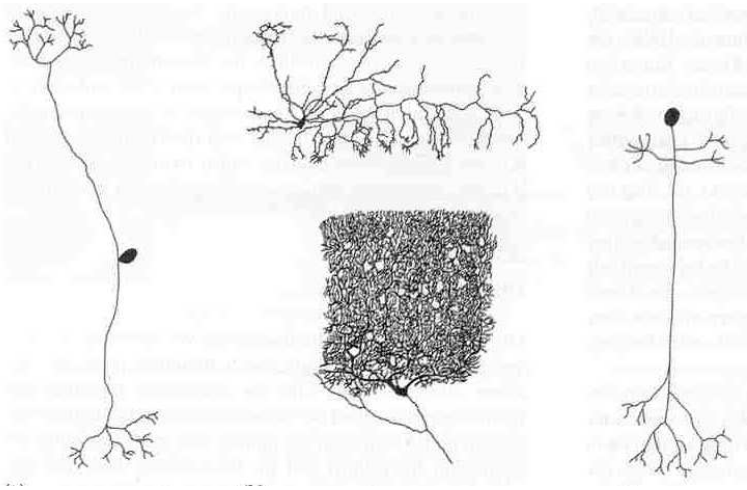

Abbildung 2.2: Neuronen unterschiedlicher Morphologie. Entnommen aus Campbell[2]

Gliazellen
Die Gliazellen (griech.: $\acute{\eta}\ \gamma\lambda\acute{\iota}\alpha$ - der Leim) sind für den Schutz, die Isolierung und die Ernährung der Neuronen und damit für ihre Funktionalität verantwortlich. Sie sind zehn bis fünfzig mal häufiger als Neuronen.
Spezielle Gliazellen bilden elektrisch isolierende Myelinschichten um das Axon der Neuronen, ohne die die Weiterleitung der Signale nicht möglich wäre.

2.2 Zwei-Photonen Mikroskopie

Bei der Mikroskopie mit herkömmlichen Lichtmikroskopen, auch *Transmissionsmikroskope* genannt, wird eine Probe von einfachem, das heißt für das menschliche Auge sichtbarem, Licht durchstahlt. Das transmittierte Licht, also derjenige Anteil, welcher beim Durchqueren der Probe nicht absorbiert wurde, erzeugt auf einer Projektionsebene ein Bild der Probe. Durch Bündelung des Lichtes mit Linsen kann ein scharfes Bild einer bestimmten Fokusebene der Probe erzeugt werden.
Eine Abwandlung von diesem Prinzip stellt das *Epifluoreszenzmikroskop* dar. Bei diesem muss die Probe vor der mikroskopischen Betrachtung mit einem Fluoreszenzfarbstoff präpariert werden. Dann wird sie mit einem kurzwelligen Anregungslicht bestrahlt, welches den Farbstoff zur Fluoreszenz anregt. Das Prinzip der Fluoreszenz besteht darin, dass die Moleküle des Fluoreszenzfarbstoffes durch Aufnahme einer bestimmten Energiemenge ΔE auf ein engergetisch höheres Niveau angehoben werden, jedoch nach der Energieaufnahme sofort wieder auf das alte Niveau zurückfallen, wobei sie die aufgenommene Energie als Fluoreszenzlicht abstrahlen. Dieses wird dann bei der Mikroskopie genutzt, um auf einer Projektionsebene ein Bild der Probe zu erzeugen. Um nicht auch das Anregungslicht im Bild wiederzufinden, wird vor die Projektionsebene ein Filter gesetzt, der nur das Fluoreszenzlicht transmittiert.
Beide Verfahren haben den Nachteil, dass das Bild der Fokusebene, auf der die Probe scharf abgebildet wird, durch die unscharfen Bilder außerhalb der Fokusebene überlagert werden.

Die auf der Fluoreszenzmikroskopie basierenden *Laser-Raster-Miskroskope* bilden nicht mehr die Probe als ganzes ab, sondern messen nur noch die Fluoreszenzintensität in genau einem bestimmten fokussierten Punkt. Durch Abtastung der Probe nach einem festgelegten Raster kann so wieder ein dreidimensionales Bild aufgebaut werden. Als Lichtquelle werden dabei Laser benutzt, welche in die Probe fokussiert werden. Bei der *Ein-Photonen-Laser-Raster-Mikroskopie* sind dies Laser mit einer Wellenlänge, welche gerade zur Anregung des Fluoreszenzfarbstoffes ausreicht. Leider wird dadurch nicht nur der Farbstoff im Fokuspunkt angeregt, sondern auf dem ganzen Weg des Lasers durch die Probe.
Um diesem Problem zu begegnen, wird auf der Projektionsebene eine Lochblende eingesetzt, welche Licht von außerhalb des Fokuspunktes abblockt. Man spricht von *konfokaler Mikroskopie*.

Das Verfahren der *Zwei-Photonen-Mikroskopie* verwendet Laser mit einer Wellenlänge, die der doppleten Wellenlänge der bei der Ein-Photonen-Mikroskopie verwendeten Laser entspricht. Die Energie der einzelnen Photonen ist damit nicht mehr ausreichend, um in der Probe Fluoreszenz anzuregen. Zwei solcher Photonen, die fast gleichzeitig auf ein Fluoreszenzmolekül treffen, können in einer Art Addition ihrer Energien jedoch die Fluoreszenz bewirken. Das kann allerdings nur im Fokuspunkt, in den die Laser fokussiert werden, passieren.

Der Vorteil dieser Zwei-Photonen-Anregung liegt darin, dass die Lochblende, die das Licht von außerhalb des Fokuspunktes abblockt, unnötig wird, da eine Anregung ohnehin nur noch im Fokuspunkt auftreten kann. Weiter wird die Schädigung der Probe wegen der geringeren Laserenergie vermindert, was die Untersuchung lebender Zellen begünstigt.
In der Abbildung 2.3 ist der schematische Aufbau eines Zwei-Photonen-Mikroskopes gezeigt.

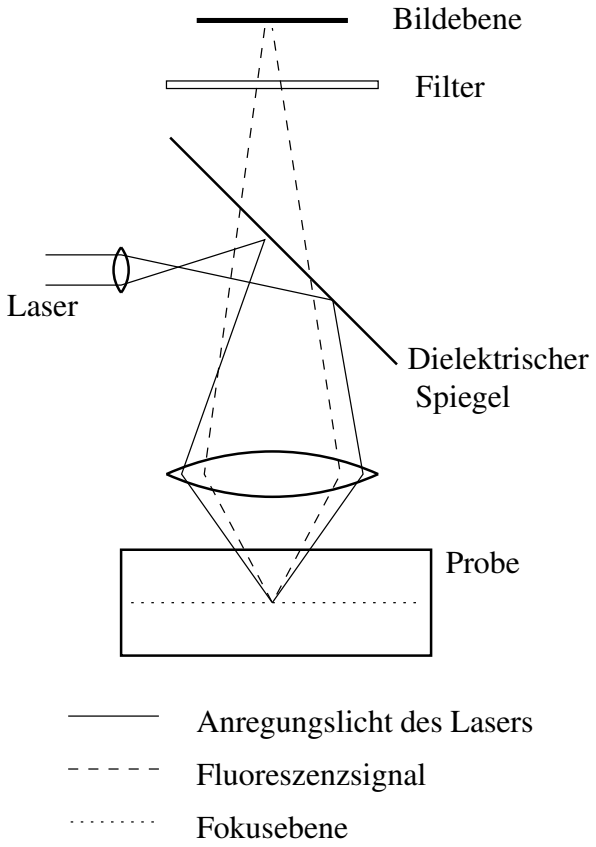

Abbildung 2.3: Schematischer Aufbau eines Zwei-Photonen-Mikroskopes

Die obigen Darstellungen der unterschiedlichen Mikroskopie-Methoden orientieren sich an Fricke[4].

Unter Verwendung der Zwei-Photonen-Mikroskopie werden am Max-Plank-Institut für medizinische Forschung dreidimensionale Bilder lebender Neuronen in Mäusegehirnen aufgenommen. Aus diesen werden bislang manuell die Geometrien der jeweiligen Neuronen rekonstruiert.
Die Rasterung des Bildes bei der Aufnahme geschieht dabei nicht in alle drei Raumrichtungen gleich; in die (vom Mikroskop aus gesehene) Tiefenrichtung,

die wir als z-Richtung bezeichnen wollen, ist der Abstand zwischen den einzelnen Bildpunkten knapp doppelt so groß gewählt wie in die anderen beiden Raumrichtungen. Wir werden bei der Filterung der Bilddaten jedoch immer von einem Raster ausgehen, das in den drei Raumrichtungen den gleichen Abstand der Bildpunkte aufweist. Wir werden also auf einem in z-Richtung gestauchten Bild rechnen. Den Abstand der Bildpunkte in die einzelnen Raumrichtungen definieren wir für unsere Rechnungen als 1.
Das Spektrum des aufgenommenen Fluoreszenzlichtes wird auf das übliche Grauwertintervall $[0, 255]$ skaliert, wobei der Wert 255 einer hohen Intensität, der Wert 0 entsprechend einer niedrigen Intensität entspricht.

Um aus den einzelnen Bildpunkten wieder ein kontinuierliches Bild zu erhalten, konstruiert man um jeden Bildpunkt einen sogenannten *Voxel*, der diesen als Mittelpunkt besitzt und deren Kantenlängen genau der Rasterung des Bildes entsprechen. Auf jedem einzelnen Voxel wird nun der Grauwert als konstant angenommen und entspricht dem Grauwert des in ihm enthaltenen Bildpunktes. Mit dem oben angegebenen Raster, welches wir für die Rechnungen verwenden werden, haben die Voxel immer die Kantenlänge und das Volumen 1.

In der Abbildung 2.4 ist ein typischer unbehandelter Datensatz gezeigt, wie er am Max-Plank Institut aufgenommen wurde.

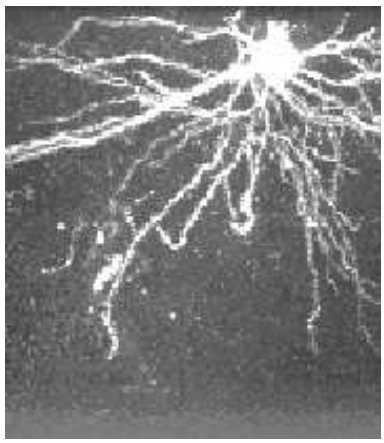

Abbildung 2.4: Ungefilterte Aufnahme eines Neurons

Deutlich ist zu erkennen, dass das Signal teilweise im Grauwertbereich des Rauschens verschwindet. Man spricht auch von einem schlechten *Signal-zu-Rausch-Verhältnis*.

3 Filterung durch nichtlineare anisotrope Diffusion

3.1 Einleitung

Die durch die Zwei-Photonen-Mikroskopie gewonnenen Aufnahmen sind qualitativ zu schlecht, um die Rekonstruktion der Nervenzellengeometrie erfolgreich auf ihnen durchführen zu können. Das liegt zum einen an dem schon erwähnten schlechten Signal-zu-Rausch-Verhältnis, so dass entweder viele Teile der Zelle bei der Segmentierung verloren gehen, oder aber zu viel Rauschen als Struktur erkannt wird. Andererseits können bei der Aufnahme Unterbrechungen in der Zellstruktur entstehen. Solche Unterbrechungen wirken auf den Rekonstruktionsalgorithmus fatal, da dieser an solchen Unterbrechungen stoppt und so ganze Teile der Nervenzelle verloren gehen können.
Vor der Weiterverarbeitung der Aufnahmen muss die Bildqualität daher entscheident verbessert werden. Diesen Vorgang nennt man *Filterung* des Bildes. Dabei wird der Bildvektor der Aufnahmedaten $\tilde{u}$ durch einen Filter f in einen modifizierten Bildvektor u überführt:

$$f(\tilde{u}) = u$$

Die Filterung von Bilddaten ist eine der Hauptaufgaben der digitalen Bildverarbeitung.
Im Fall unserer Aufnahmen benötigen wir einen Filter, der einerseits das Rauschen aus den Aufnahmen entfernt, ohne dabei die aufgenommenen Zellstrukturen zu zerstören, andererseits vorhandene Lücken in der Struktur schließt, soweit dies möglich ist.

3.2 Konstruktion des Filters

3.2.1 Rauschelimination durch Schwellenwertbildung

Für die Rauschelimination wäre es ideal, wenn die Grauwerte der verrauschten Voxel einen bestimmten Schwellenwert s_1 nicht überschritten und die Grauwerte des Signals einen bestimmten Schwellenwert s_2 nicht unterschritten. Wäre dann s_1 nur wenig größer oder sogar kleiner als s_2, so wäre die Entfernung des Rauschens mit wenig oder sogar gar keinem Verlust von Signaldaten möglich. Indem nämlich die Grauwerte aller Voxel, die s_1 nicht überschreiten, auf Null gesetzt würden, würde das Rauschen auf einfachste Art entfernt.
In unseren Aufnahmen ist diese klare Trennung von Rauschen und Signal jedoch nicht vorhanden und daher dieses Vorgehen nicht durchführbar.
Bei Betrachtung des in Abbildung 3.1 gezeigten Histogramms[1] fällt zwar eine dominierende Glockenkurve aus Hintergrundrauschen auf, von dem sich das Signal,

[1] Normalerweise entspricht der Grauwert 0 der Farbe Schwarz. Hier ist die Skala umgedreht und 0 entspricht Weiß, also dem Signal. Das ist jedoch nur in dieser Histogrammdarstellung so. In den realen Bilddaten entspricht der Grauwert 255 nach wie vor einer hohen Signalintensität.

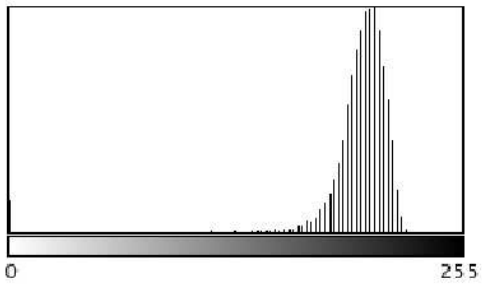

Abbildung 3.1: Histogramm für einen unbehandelten Datensatz

das hier ganz am Rand des Grauwertbereichs liegt, abhebt. Allerdings ist auch der Grauwertbereich, in dem das Signal liegt, noch stark mit Rauschen behaftet. Das wird im folgenden deutlich werden.

Wenn wir das Rauschen schon nicht ganz eliminieren können, so wenigstens einen gewissen Anteil davon. Wir unterteilen den Grauwertbereich in drei Intervalle, die durch die Werte s_u und s_o mit $s_u < s_o$ getrennt werden. Alle Werte unterhalb von s_u setzen wir auf 0, alle Werte oberhalb von s_o auf 1. Die dazwischenliegenden werden linear auf $[0, 1]$ skaliert[2].
Dieses Vorgehen macht Sinn, wenn sich das Signal hauptsächlich im oberen Intervall aufhält, nur teilweise im mittleren und gar nicht im unteren Intervall. Wir erhalten dann eine Kontrastverstärkung.

Abbildung 3.2: Datensatz nach Kontrastverstärkung durch Schwellenwertbildung

[2] Alternativ könnte man auch nichtlineare Skalierungen wählen.

In der Abbildung 3.2 ist das Ergebnis dieser Kontrastverstärkung mit den Grenzen $s_u = 10$ und $s_o = 40$ zu sehen. Deutlich sind hier Unterbrechungen der Struktur und im Grauwertbereich des Signals liegendes Rauschen zu erkennen.

Wir werden diese Art der Kontrastverstärkung mit den obigen Schwellenwerten standartmäßig vor jeder weiteren Behandlung der Aufnahmen verwenden. Dies sei hiermit vorrausgesetzt, auch wenn wir es an den jeweiligen Stellen nicht mehr explizit erwähnen werden.

Da wir das Rauschen also nicht beseitigen können, möchten wir es zumindest glätten und auf diese Weise im Idealfall einen einheitlichen Hintergrund erhalten, von dem sich das Signal abhebt. Einen solchen Hintergrund könnte ein Segmentierungsalgorithmus problemlos vom eigentlichen Objekt unterscheiden.

3.2.2 Medianfilter

Der Vorgang der Glättung läuft auf eine Mittelwertbildung hinaus. Einen einfachen Filter dieser Art stellt der sogenannte *Medianfilter* dar. Dabei wird der Grauwert eines Voxels x durch den Mittelwert der Grauwerte in einer festzulegenden Umgebung $\Omega(x)$ ersetzt:

$$u(x) = \frac{1}{|\Omega(x)|} \sum_{y \in \Omega(x)} \tilde{u}(y)$$

Es ist klar, dass dieses Vorgehen nicht nur das Rauschen eliminieren, sondern auch Kanten und Strukturen verwischen wird. Der Medianfilter eignet sich daher zwar für die Elimination einzelner gestörter Bildpunkte inmitten größerflächiger Gebiete mit konstanten Grauwerten, nicht jedoch für stark verrauschte Bilder mit feinen Strukturen, wie Nervenzellen sie darstellen.

3.2.3 Gaussfilter

Schaut man sich den Medianfilter an, so fällt auf, dass es sich dabei eigentlich um eine diskrete Faltung handelt:

$$u = (\tilde{u} * g)(x) = \sum_{y \in \Omega(x)} \tilde{u}(y) \cdot g(y - x), \quad g \equiv \frac{1}{|\Omega(x)|}$$

Die Gewichtsfunktion ist dabei konstant. Aber warum sollte man bei der Mittelwertbildung alle Voxel der Umgebung $\Omega(x)$ gleich gewichten? Weiter entfernte Punkte sollten weniger stark gewichtet in die Summe einfließen. Der Charakter der Mittelung bliebe erhalten, jedoch würden Kanten weniger stark in Mitleidenschaft gezogen, da sie weniger weiträumig verschmiert würden.

Um eine solche Gewichtung zu erhalten, wird in der Bildverarbeitung normalerweise die Gaussfunktion

$$G_\sigma(x) = \frac{1}{(2\pi\sigma^2)^{\frac{n}{2}}}\, e^{-\frac{x^2}{2\sigma^2}} \tag{3.1}$$

benutzt. Dabei ist n die Dimension des Bildes. Der durch Anwendung dieser Gewichtsfunktion entstehende Filter heißt *Gaussfilter*.
Bevor wir uns jedoch den Effekt dieses Filters ansehen, wollen wir feststellen, daß er auch auf eine andere Weise erreicht werden kann.

3.2.4 Die Diffusionsgleichung

Wir betrachten die Diffusionsgleichung

$$\partial_t u = \Delta u \quad \text{auf } \mathbb{R}^+ \times \mathbb{R}^n \tag{3.2}$$

Diese partielle Differentialgleichung modelliert den physikalischen Prozess der Diffusion, zum Beispiel die zeitliche und räumliche Verteilung der Konzentration eines Stoffes in Flüssigkeit oder Gas, oder die Verteilung von Wärme in einem Körper. Sie entsteht aus zwei physikalischen Gesetzmäßigkeiten:

1. Der Vorgang der Diffusion gleicht Unterschiede in der Konzentrationsverteilung aus. Ein Konzentrationsgradient erzeugt somit einen Fluss ϕ, der diesen Gradienten auszugleichen sucht:

   $$\phi = -\nabla u$$

   Diese Beziehung heißt *Fick'sches Gesetz*. Da der Fluss linear vom Gradienten abhängt, heißt auch die Diffusion linear.

2. Masse bleibt erhalten, wird also weder vernichtet, noch erzeugt. Dieses Gesetz der Massenerhaltung formuliert sich mathematisch als

   $$\partial_t u = -\nabla \cdot \phi$$

   Anschaulich wird dieses, wenn man zur integralen Darstellung übergeht und das Integral dann als Randintegral schreibt:

   $$\int_V \partial_t u \, dx = -\int_V \nabla \cdot \phi \, dx = -\int_{\partial V} \phi \cdot \vec{n} \, ds$$

   Die zeitliche Veränderung einer Konzentration innerhalb eines bestimmten Gebietes V ist gleich dem Fluss über den Rand.

Aus diesen beiden Gesetzmäßigkeiten setzt sich die Gleichung zusammen.
Bei Angabe einer Anfangsbedingung

$$u(x,0) = u_0(x) \quad \text{auf } \mathbb{R}^n$$

hat diese die Lösung (vgl. Strauss[14])

$$u(x,t) = \int_{\mathbb{R}^n} u_0(y) \cdot \frac{1}{(4\pi t)^{\frac{n}{2}}} e^{-\frac{(x-y)^2}{4t}} dy \tag{3.3}$$

3.2.5 Lineare Diffusion und Gaussfilter

Schreiben wir die Faltung des Bildes mit der Gaussfunktion in der kontinuierlichen Form

$$u(x,\sigma) = (\tilde{u} * G_\sigma)(x) = \int_{\mathbb{R}^n} \tilde{u}(y) \cdot \frac{1}{(2\pi\sigma^2)^{\frac{n}{2}}} e^{-\frac{(x-y)^2}{2\sigma^2}} dy \tag{3.4}$$

so sehen wir, daß diese für $\sigma = \sqrt{2t}$ der Lösung der Diffusionsgleichung mit $u_0 = \tilde{u}$ entspricht. Demnach erhalten wir unabhängig davon, ob wir auf dem verrauschten Bild eine Gaussfaltung ausführen oder aber einen linearen Diffusionsprozess starten, dasselbe Resultat. Der Zeitschritt in der Diffusionsgleichung korreliert dabei mit der Standartabweichung der Gaussfunktion. Wenn man weiter entfernte Punkte möglichst schwach gewichten will, so muß man eine kleine Standartabweichung σ bzw. einen kleinen Zeitschritt t wählen. Physikalisch ist das sofort einsichtig: Je kleiner die betrachtete Zeitspanne der Diffusion, desto weniger Einfluss hat ein weiter entfernter Punkt auf den betrachteten.
Die im ersten Moment vielleicht ungewohnte Betrachtungsweise, Diffusion auf Bilddaten anzuwenden, wird sofort einsichtig, wenn man sich die einzelnen Grauwerte als Konzentrationen in den Bildpunkten vorstellt.

Leider müssen wir feststellen, dass der Gaussfilter beziehungsweise die lineare Diffusion sich auch nicht als geeignetes Mittel herausstellen, um die Bildqualität zu verbessern. Zwar glätten sie das Rauschen gut heraus, zerstören aber noch viel zu sehr die Strukturen. Vor allem feine Strukturen gehen schnell verloren.

Abbildung 3.3: Ergebnis der linearen Diffusion

3.2.6 Nichtlineare Diffusion

Um die Kanten der Strukturen und damit diese selbst zu erhalten, muss die Diffusion an den Kanten gestoppt werden. Dann hätte man einerseits die Glättung des Rauschens durch die Diffusion erreicht, andererseits kein Verschmieren und schließliches Verschwinden der Strukturen mehr.
Um eine solche Diffusion zu erhalten, muss man das Fick'sche Gesetz modifizieren. Der Fluss soll nicht mehr linear vom Gradienten abhängen, sondern an den Kanten der Strukturen sehr klein sein. Dazu muss man natürlich zunächst einmal diese Kanten erkennen. Die einfachste Art der Kantendetektion stellt der Gradient selbst dar. Das Fick'sche Gesetz wird nun folgendermaßen modifiziert:

$$\phi(x,t) = g(|\nabla u(x,t)|) \cdot \nabla u(x,t)$$

Dabei soll $g(.)$ eine monoton fallende Funktion sein mit $g(0) = 1$. Ist nun der Gradient ∇u groß, so ist der Wert der Funktion g klein, woraus ein kleiner Fluss resultiert. Kanten werden so erhalten, da die Diffusion an ihnen verlangsamt wird oder sogar je nach Wahl von g stoppt. Die Diffusionsgleichung ist nun nichtlinear und hat die Form

$$\partial_t u = \nabla \cdot (g(|\nabla u|)\nabla u) \tag{3.5}$$

Dieses Vorgehen stammt von Perona und Malik, weswegen die Gleichung (3.5) auch als *Perona-Malik-Diffusion* bezeichnet wird.
Zur Wahl der Funktion $g(.)$ sind in der Literatur verschiedene Varianten vorgeschlagen worden, wovon Mrázek[9] die folgenden zusammenstellt:

1. Die Perona-Malik-Diffusivität

   $$g(s) = \frac{c}{1 + (s/\lambda)^{\alpha+1}}$$

   mit einer Konstanten c und $\alpha > 0$. Sie ist für $c = 1$, $\alpha = 1$ und $\lambda = 0.3$ in Abbildung 3.4 gezeigt.

2. Die Weikert-Diffusivität

   $$g(s) = 1 - \exp\left(\frac{c}{(s/\lambda)^m}\right)$$

   mit Konstanten c und m. Sie ist für $c = -2.33667$, $m = 4$ und $\lambda = 0.3$ in Abbildung 3.4 gezeigt.

3. Die Black-Sapiro Diffusivität

   $$g(s) = \begin{cases} \frac{1}{2}\left(1 - (\frac{s}{\lambda\cdot\sqrt{5}})^2\right)^2 & |s| \leq \lambda \cdot \sqrt{5} \\ 0 & \text{sonst} \end{cases}$$

   Diese ist für $\lambda = 0.3$ in Abbildung 3.4 gezeigt.

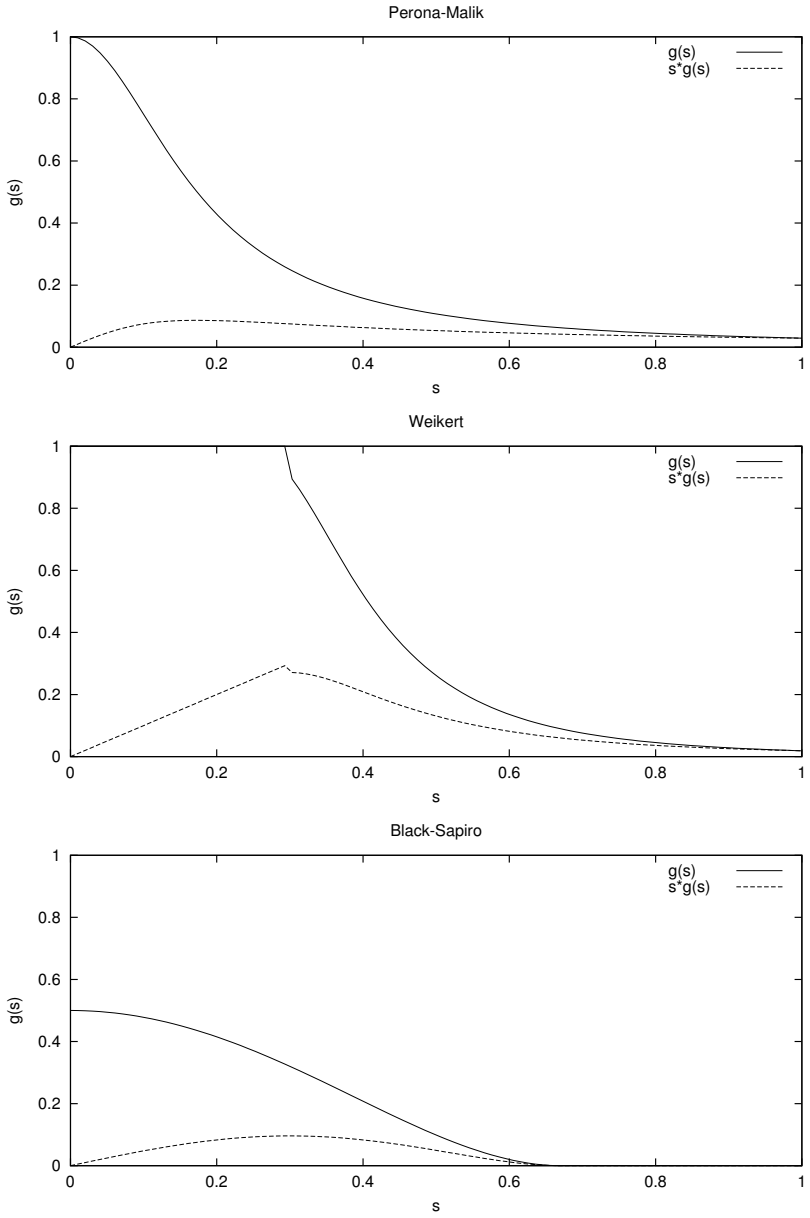

Abbildung 3.4: Unterschiedliche Diffusivitätsfunktionen $g(s)$ und ihre zugehörigen Flussfunktionen $s \cdot g(s)$ jeweils für $\lambda = 0.3$

Die Perona-Malik Diffusivität hat an der Stelle λ einen Wendepunkt und die resultierende Flussfunktion $s \cdot g(s)$ ein lokales Maximum, so dass sie monoton steigend ist für $s < \lambda$ und monoton fallend für $s > \lambda$. Für einen Gradienten mit dem Betrag λ erhält man somit den maximalen Fluss.
In der Weikert Diffusivität wurden die Konstanten c und m oben so gewählt, dass sie ebenfalls diese Eigenschaften aufweist.
Damit dies auch die Black-Sapiro Diffusivität tut, wurde λ mit dem Faktor $\sqrt{5}$ multipliziert, welcher in der originalen Formulierung fehlt.
Beide Setzungen gehen auf Mr*á*zek[9] zurück.

Normalerweise macht man g nicht direkt von $|\nabla u|$ abhängig. Das Rauschen würde g zu stark variieren lassen und die Diffusion damit zu stark beeinflussen. Daher definiert man

$$u_\sigma := u * G_\sigma$$

und macht g von $|\nabla u_\sigma|$ abhängig. Damit erzeugt man ein vorgeglättetes Bild[3], welches zur Bestimmung der Stärke der Diffusion benutzt wird, wodurch auch der Koeffizient g in der Differentialgleichung geglättet wird. Das Rauschen verliert an Einfluss[4].

3.2.7 Nichtlineare Anisotrope Diffusion

Das Stoppen der Diffusion an den Kanten hat zwar den Vorteil diese zu erhalten, jedoch den Nachteil das Rauschen an diesen nicht zu glätten. Es ist aber nicht einzusehen, warum die Diffusion an den Kanten gänzlich gestoppet werden sollte. Es soll ja nur eine Diffusion normal zu den Kanten vermieden werden. Längs dazu soll Diffusion dagegen möglich sein, ist wegen der Rauschglättung sogar wünschenswert.

Beschränken wir uns für die Darstellung vorrübergehend auf den Fall eines zweidimensionalen Bildes.
Um die Glättung entlang von Kanten zu ermöglichen, nehmen wir eine weitere Modifikation des Fick'schen Gesetzes vor. Der Fluss ϕ soll nun nicht mehr parallel zum Gradienten erfolgen. Um dies zu erreichen, wird er mit einer vom Bild abhängigen Matrix $D(u)$ multipliziert:

$$\phi = D(u) \cdot \nabla u$$

Die resultierende Differentialgleichung lautet

$$\partial_t u = \nabla \cdot (D(u) \cdot \nabla u) \tag{3.6}$$

[3] das Bild selbst wird dabei aber nicht verändert!

[4] Die Einführung von u_σ hat auch theoretische Gründe. Nach Mrázek[9] ist das Problem der Perona-Malik-Diffusion schlecht gestellt, da es sensibel auf die Anfangsbedingungen reagiert, jedoch erhält man durch die Verwendung von u_σ ein korrekt gestelltes Problem.

Die Matrix D heißt *Diffusionstensor*. Sie soll zwei normierte, orthogonale Eigenvektoren v_1, v_2 mit

$$v_1 \parallel \nabla u_\sigma,\ v_2 \perp \nabla u_\sigma \tag{3.7}$$

und zugehörige Eigenwerte $\lambda_1,\ \lambda_2$ besitzen. Dann nämlich folgt mit

$$\nabla u = \alpha_1 v_1 + \alpha_2 v_2$$

für den Fluss:

$$\phi = D \cdot (\alpha_1 v_1 + \alpha_2 v_2) = D\alpha_1 v_1 + D\alpha_2 v_2 = \lambda_1 \alpha_1 v_1 + \lambda_2 \alpha_2 v_2$$

Der Fluss ϕ ist also in die Richtungen von v_1 und v_2 in seiner Stärke über die Eigenwerte λ_1 und λ_2 steuerbar.
Da D von u abhängig ist, ist Gleichung (3.6) weiterhin nichlinear. Da außerdem der Fluss nicht mehr parallel zum Gradienten erfolgt, heißt die Diffusion nun *anisotrop*.

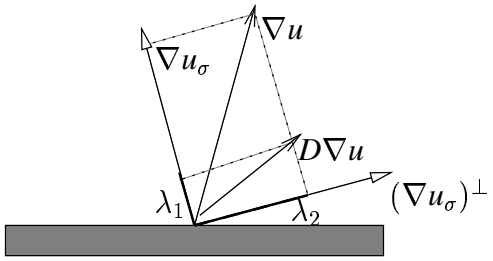

Abbildung 3.5: Wirkung des Diffusionstensors

Den Diffusionstensor kann man konstruieren, indem man die Eigenvektoren v_1 und v_2 zu einer unitären Matrix B zusammenfasst und

$$D = B \cdot \mathrm{diag}(\lambda_1, \lambda_2) \cdot B^T \tag{3.8}$$

setzt.
Nun kann man über die Wahl der Werte λ_1 und λ_2 die Stärke der Diffusion in Richtung des Gradienten und senkrecht dazu steuern.
Normalerweise wird man längs einer Kante volle Diffusion zulassen. In Gradientenrichtung kann man z.B. die Funktion $g(.)$, wie sie in der Perona-Malik-Diffusion verwendet wird, benutzen:

$$\lambda_1 = g(|\nabla u_\sigma|)$$
$$\lambda_2 = 1$$

In Weikert[17] wird eine leicht modifizierte Form vorgeschlagen. Dort werden die Tensoren

$$J_0(\nabla u_\sigma) := \nabla u_\sigma \otimes \nabla u_\sigma = \nabla u_\sigma \nabla {u_\sigma}^T \tag{3.9}$$

und

$$J_\rho(\nabla u_\sigma) := G_\rho * (\nabla u_\sigma \otimes \nabla u_\sigma), \quad \rho > 0 \tag{3.10}$$

definiert, wobei G_ρ die nach (3.1) definierte Gaussfunktion ist. J_0 besitzt zwei Eigenvektoren mit den Eigenschaften (3.7). Es sind nämlich

$$\begin{aligned} J_0 \nabla u_\sigma &= \nabla u_\sigma \nabla {u_\sigma}^T \nabla u_\sigma = \nabla u_\sigma \|\nabla u_\sigma\|^2 \qquad \text{und} \\ J_0 \nabla {u_\sigma}^\perp &= \nabla u_\sigma \nabla {u_\sigma}^T \nabla {u_\sigma}^\perp = \nabla u_\sigma \cdot 0 = 0 \end{aligned}$$

Daher könnte man einfach J_0 als Diffusionstensor benutzen, was jedoch nicht getan wird.
Der Tensor $J_\rho = \begin{pmatrix} j_{11} & j_{12} \\ j_{21} & j_{22} \end{pmatrix}$ hat nach Weikert[17] zwei orthogonale Eigenvektoren v_1, v_2 mit

$$v_1 \parallel \begin{pmatrix} \dfrac{2j_{12}}{j_{22} - j_{11} + \sqrt{(j_{11} - j_{22})^2 + 4j_{12}^2}} \end{pmatrix}$$

und wird dort *Strukturtensor* genannt. Diese Eigenvektoren werden nun zur Konstruktion des Difffusionstensors benutzt. Damit werden zur Bestimmung der Diffusionsrichtungen nicht nur die punktlokalen Gegebenheiten, sondern die Struktur innerhalb einer Umgebung ρ des Punktes mit einbezogen.

3.2.8 Randwerte

Die bisher vorgestellten partiellen Differentialgleichungen haben wir immer auf dem ganzen $\mathbb{R}^n$ betrachtet. Auch die Lösung (3.3) der linearen Diffusion galt für den $\mathbb{R}^n$.
Unsere Bilddaten sind jedoch räumlich beschränkt, das heißt wir wollen auf einem Teilgebiet $\Omega \subset \mathbb{R}^3$ rechnen. Dazu müssen wir dem bislang vorliegenden Anfangswertproblem eine Randbedingung hinzufügen, das heißt wir müssen angeben, wie sich die Lösung auf dem Rand $\partial\Omega$ verhalten soll.
Die drei gängigen Randbedingungen sind:

1. Die *Dirichlet*-Bedingung $u = \psi$. Die Werte von u auf dem Rand werden fest vorgegeben. Auf den physikalischen Prozess der Wärmediffusion bezogen würde dies bedeuten, daß der Rand von Ω auf einer bestimmten Temperatur gehalten würde.

2. Die *Neumann*-Bedingung $\nabla u \cdot \vec{n} = \psi$. Die Werte der Ableitung in Richtung der äußeren Normalen werden auf dem Rand fest vorgegeben. Physikalisch entspricht dies einer konstanten Wärmezufuhr oder einem konstanten Wärmeabfluß. Der Spezialfall $\nabla u \cdot \vec{n} = 0$ wird auch als *natürliche Randbedingung* bezeichnet. Physikalisch bedeutet dies eine Isolierung von Ω nach außen.

3. Die *Robin*-Bedingung $\nabla u \cdot \vec{n} + \alpha u = \psi$. Die Werte der Ableitungen in Richtung der äußeren Normalen hängen auf dem Rand von der Lösung u ab. Physikalisch ist das z.B. der Fall, wenn die Umgebung von Ω eine feste Temperatur aufweist und der Fluss über den Rand von der Differenz dieser zu der Temperatur auf dem Rand abhängt.

Da wir nicht wollen, daß Bildinformation über den Rand verloren geht, kann es für uns nur Sinn machen, die natürliche Randbedingung $\nabla u \cdot \vec{n} = 0$ zu wählen. Die anderen Randbedingungen kommen somit nicht in Frage.
Wir wollen allerdings nicht die isotrope, sondern die anisotrope Diffusion verwenden und erhalten deswegen $(D(u) \cdot \nabla u) \cdot \vec{n} = 0$ als natürliche Randbedingung.

Insgesamt haben wir damit das Anfangs-Randwert-Problem

$$\begin{aligned} &\partial_t u = \nabla \cdot (D(u) \cdot \nabla u) && \text{auf } \mathbb{R}^+ \times \Omega \\ &u(x,0) = u_0(x) && \text{auf } \bar{\Omega} \\ &(D(u) \cdot \nabla u) \cdot \vec{n} = 0 && \text{auf } \mathbb{R}^+ \times \partial\Omega \end{aligned} \tag{3.11}$$

erhalten.

3.3 Strukturerfassung durch Trägheitsmomente

3.3.1 Einleitung

Im Fall unserer 3D Aufnahmen von Neuronen ist die Anwendung der gradientengesteuerten nichtlinearen anisotropen Diffusion problematisch. In zwei Dimensionen kann der Gradient als Kriterium zum Erkennen von Kanten und Zerlegung des Flusses in eine „erlaubte“ und eine „verbotene“ Richtung benutzt werden. In drei Dimensionen gibt es aber drei mögliche Diffusionsrichtungen, in die der Fluss zerlegt werden kann.
Bei unseren Aufnahmen haben wir folgende Fälle zu berücksichtigen:

- Zylinderförmige Stränge wie die Dendriten oder das Axon, an denen die Diffusion nur in einer Richtung erfolgen soll
- Flächen wie die Zellkörperoberfläche, an denen die Diffusion nur in zwei Richtungen verlaufen soll
- Innere Bereiche des Neurons wie der Zellkörper oder Bereiche, in denen keine Zellstruktur aufgenommen wurde, in denen die Diffusion in alle drei Richtungen verlaufen soll

Wir nennen diese Strukturen *linear*, *planar* bzw. *isotrop*.
Eine anisotrope Diffusion, die allein den Gradienten als Kriterium zur Bestimmung der Diffusionsrichtung benutzt, wird diesen Fällen nicht gerecht, da sie genau eine Richtung festlegt, in welche die Diffusion gestoppt werden soll. Im Fall linearer Strukturen müssen aber zwei Richtungen ausgeschlossen werden. Der Gradient kann also als Kriterium zur Erkennung der drei obigen Fälle nicht dienen.

3.3.2 Trägheitsmomente physikalischer Körper

Über den aus der Physik stammenden Begriff des Trägheitstensors ist es möglich, die benötigte Strukturinformation zu erhalten. Dazu geben wir die Formeln zur Berechung

- der Masse

$$m_K = \int_K \rho(y)\ dy \tag{3.12}$$

- des Schwerpunktes

$$sp_K = \frac{1}{m_K} \int_K \rho(y) \cdot y\ dy \tag{3.13}$$

- und des Trägheitstensors

$$I_K = \int_K \rho(y)\ \left(|y - sp_K|^2 \mathbb{I} - (y - sp_K) \otimes (y - sp_K)\right)\ dy \tag{3.14}$$

eines Körpers K.
Der Trägheitstensor ist symmetrisch und positiv definit, hat also im Fall von drei Dimensionen drei orthonormale Eigenvektoren zu drei reellen positiven Eigenwerten.
Der Tensor

$$T_K = \int_K \rho(y)\ ((y - sp_K) \otimes (y - sp_K))\ dy \tag{3.15}$$

hat nach Lenzen[7] dieselben Eigenvektoren wie I_K und die zugehörigen Eigenwerte haben die gleiche Differenz. Wir werden daher im folgenden und für die späteren Rechnungen T_K benutzen und diesen *vereinfachter Trägheitstensor* nennen.
Mit $\lambda_i, i = 1, 2, 3$ bezeichnen wir die der Größe nach sortierten Eigenwerte von T_K, so dass $\lambda_1 \geq \lambda_2 \geq \lambda_3$ gilt, und mit $v_i, i = 1, 2, 3$ die zugehörigen Eigenvektoren.

Die Eigenvektoren des Trägheitstensors heißen *Haupträgheitsachsen* und die Eigenwerte *Haupträgheitsmomente*. Physikalisch geben letztere an, wieviel Energie nötig ist, um den Körper K um die Achse, die der zugehörige Eigenvektor erzeugt,

rotieren zu lassen. Für uns interessant ist aber die Strukturinformation, die uns die Eigenwerte und -vektoren über K liefern.

Beispiel

Als Beispiel betrachten wir einen parallel zu den Koordinatenachsen ausgerichteten Quader. Er habe die Kantenlängen a in x-Richtung, b in y-Richtung und c in z-Richtung.

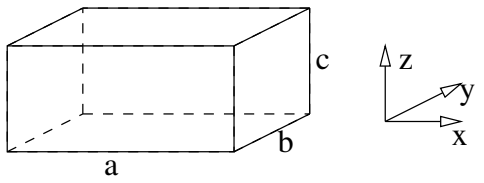

Abbildung 3.6: Quader

Auf diesem Quader definieren wir eine Bildfunktion $\rho(x) \equiv 1$. Damit erhalten wir unabhängig von der relativen Position des Quaders im Raum

$$T_Q = \frac{1}{12}\begin{pmatrix} a^3bc & 0 & 0 \\ 0 & ab^3c & 0 \\ 0 & 0 & abc^3 \end{pmatrix}$$

Die Eigenvektoren von T_Q sind genau die Einheitsvektoren. Sie beschreiben damit die Ausrichtung des Quaders im Raum.
Dominiert eine der drei Längen a, b oder c über die anderen, so dominiert auch der zu dieser Richtung korrespondierende Eigenwert gegenüber den anderen. Genauer gibt das Verhältnis der Eigenwerte von T_K uns die Information, welcher der möglichen Strukturfälle - linear, planar oder isotrop - vorliegt.

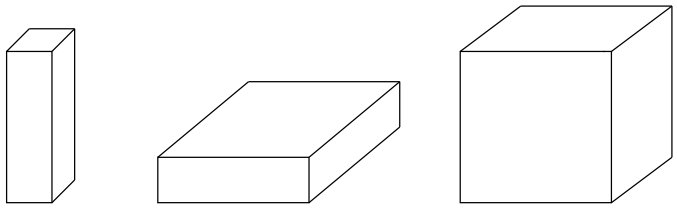

Abbildung 3.7: Beispiel für eine lineare (links) eine planare (mitte) und eine isotrope Struktur (rechts)

Im Fall eines dominierenden Eigenwertes haben wir eine eher lineare, im Fall zweier dominierender Eigenwerte gleicher Größenordnung eine eher planare und im Fall dreier Eigenwerte gleicher Größenordnung isotrope Körper vorliegen.

Um diese Fälle zu charakterisieren, definieren wir Lenzen[7] folgend die Größen

$$c_l := \frac{\lambda_1 - \lambda_2}{\sum_{i=1}^{3} \lambda_i} \tag{3.16}$$

$$c_p := \frac{2\lambda_2 - \lambda_3}{\sum_{i=1}^{3} \lambda_i} \tag{3.17}$$

$$c_i := \frac{3\lambda_3}{\sum_{i=1}^{3} \lambda_i} \tag{3.18}$$

Es gilt $c_l, c_p, c_i \in [0,1]$ und $c_l + c_p + c_i = 1$. Diese Werte können uns eine quantitative Aussage über die Zugehörigkeit eines Körpers zu den drei Strukturen machen. Die Eigenvektoren von T_K geben uns wie im Beispiel des Quaders die Ausrichtung der Struktur im Raum an.

Bezeichnung: Wir werden im folgenden die Eigenvektoren und die Eigenwerte von T_K unter dem Begriff *Trägheitsmomente* zusammenfassen.

3.3.3 Anwendung auf Bilddaten

Um nun die nötigen Diffusionsrichtungen in einem Bildpunkt festzustellen, können wir uns der Informationen, die uns der Trägheitstensor liefert, bedienen. Wir legen ein Volumen V festzulegender Größe um einen Bildpunkt x und interpretieren den so spezifizierten Ausschnitt als physikalischen Körper.

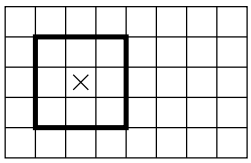

Abbildung 3.8: Bildausschnitt

Unter der Annahme, dass wir nur ganze Voxel und keine Anteile davon in die Volumina $V(x)$ aufnehmen, erhalten wir, da ja die Bildfuntion als konstant auf den einzelnen Voxeln angenommen wurde, aus den Formeln (3.12), (3.13) und (3.15) die folgenden Versionen für

- die Masse

$$m_V(x) = \sum_{y \in V(x)} u(y) \tag{3.19}$$

- den Schwerpunkt

$$sp_V(x) = \frac{1}{m_v(x)} \sum_{y \in V(x)} u(y) \cdot y \tag{3.20}$$

- den vereinfachten Trägheitstensor

$$T_V(x) = \sum_{y \in V(x)} u(y) \cdot ((y - sp_V(x)) \otimes (y - sp_V(x))) \tag{3.21}$$

Dabei ist u der Bildvektor.
Diese Formeln gelten natürlich nur, wenn das Voxelvolumen, wie wir es ja angenommen haben, für alle Voxel gleich 1 ist. Ansonsten müssten die Werte $u(y)$ noch mit dem jeweiligen Voxelvolumen multipliziert werden.

3.3.4 Steuerung der Diffusion

Wie in Abschnitt 3.2.7 dargestellt, können wir nun die in einem Bildpunkt gewonnenen Strukturinformationen zur Steuerung der Diffusion in diesem Bildpunkt einsetzen. Wir schreiben die normierten orthogonalen Eigenvektoren des vereinfachten Trägheitstensors $T_V(x)$ spaltenweise in eine Matrix B und setzen

$$D = B \cdot \mathrm{diag}(t_1, t_2, t_3) \cdot B^T \tag{3.22}$$

mit festzulegenden Einträgen t_1, t_2, t_3, welche die Stärke der Diffusion steuern. Bei deren Wahl ist man nicht von vornherein festgelegt.
Eine Überlegung ist es, für die t_i, feste skalare Größen zu setzen. Wählt man beispielsweise $t_1 = 1, t_2 = t_3 = 0$, so erhält man in jedem Bildpunkt eine Diffusion in nur eine Richtung, nämlich in Richtung des Eigenvektors v_1 von $T_V(x)$. Diese wollen wir auch als *Hauptdiffusionsrichtung* bezeichnen.
Aus der Arbeitsgruppe von Professor Rumpf an der Universität Duisburg stammt der Vorschlag,

$$\begin{aligned} t_1 &= 1 \\ t_2 &= g(c_l) \\ t_3 &= g(1 - c_i) \end{aligned} \tag{3.23}$$

zu setzen. Dabei steht $g(.)$ für eine der Funktionen aus Abschnitt 3.2.6. In die Hauptdiffusionsrichtung erfolgt demnach die Diffsuion in voller Stärke. In die zweite Richtung erfolgt sie stark, falls c_l klein, also die Struktur eher planar oder isotrop angelegt ist, und schwach, falls c_l groß, also die Struktur eher linear angelegt ist. Entsprechend ist t_3 nur dann groß, wenn die Struktur eher isotrop angelegt ist.

3.4 Diskretisierung

3.4.1 Finite Volumen Diskretisierung

Die Lösung des Anfangs-Randwertproblems

$$\begin{aligned} \partial_t u &= \nabla \cdot (D(u) \cdot \nabla u) && \text{auf } \mathbb{R}^+ \times \Omega \\ u(x,0) &= u_0(x) && \text{auf } \bar{\Omega} \\ (D(u) \cdot \nabla u) \cdot \vec{n} &= 0 && \text{auf } \mathbb{R}^+ \times \partial\Omega \end{aligned} \tag{3.24}$$

kann von einem Computer nur in einer endlichen Zahl von Punkten des Gebietes Ω und der Zeitachse berechnet werden. Wir müssen also eine solche Auswahl treffen und einen Algorithmus zur Berechnung von u in den gewählten Punkten entwickeln.
Dieser Vorgang heißt *Diskretisierung* des Problems.

Auf der Zeitachse wählt man eine Anzahl von Punkten mit festen Abständen τ.
Die Wahl der räumlichen Punkte hängt von Ω und dem behandelten Problem ab, wichtig ist aber, dass auch der Rand $\partial\Omega$ ausreichend berücksichtigt wird.
Wir wollen die Menge der räumlichen Punkte G nennen und die Anzahl der Punkte mit $|G|$ bezeichnen.

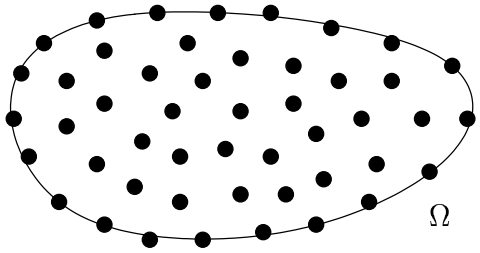

Abbildung 3.9: Gebiet mit ausgewählten Punkten

Den Zeitterm behandelt man bei der Diskretisierung mit Methoden für gewöhnliche Differentialgleichungen. Wir werden eine einfache Vorwärtsdifferenz verwenden. Für andere Methoden sei auf Schwarz[12] verwiesen.
Zur Behandlung des Ortsterms werden wir das Finite Volumen Verfahren verwenden. Alternativen dazu sind das Finite Differenzen Verfahren und das Finite Elemente Verfahren (siehe Schwarz[12]).
Zur Beschreibung des Finite Volumen Verfahrens müssen wir zunächst den Begriff des Gitters erläutern. Ausgehend von der Menge G der räumlichen Punkte, in denen wir die Lösung u berechnen wollen, verbinden wir diese so, daß eine Zerlegung von Ω in Dreiecks- und Viereckselemente für den Fall eines zweidimensionalen Gebietes, in Tetraeder und Hexaeder für den Fall eines dreidimensionalen Gebietes, entsteht. Eine solche Zerlegung wird *Triangulierung* genannt. Die Menge G zusammen mit den Verbindungen zwischen den Punkten, welche wir im folgenden

Knoten nennen wollen, heißt *Gitter*. Besitzen die Gitterknoten eine feste Anordnung, so nennen wir das Gitter strukturiert, sonst unstrukturiert.

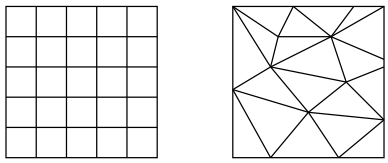

Abbildung 3.10: Beispiel für ein strukturiertes und ein unstrukturiertes Gitter

In unserem Fall ist die Auswahl des Gitters nicht schwierig. Aufgrund der ohnehin schon diskreten Vorgabe des Bildvektors wählen wir die Voxelmittelpunkte als Knoten und verbinden diese zu einem strukturierten Hexedergitter.

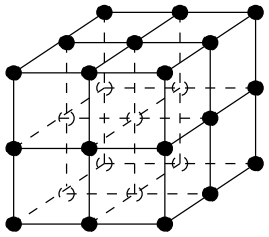

Abbildung 3.11: Gewähltes Gitter

Das Finite Volumen Verfahren konstruiert nun sogenannte *Kontrollvolumina* V_i zu jedem Punkt $x_i \in G$, für die

- $x_i \in V_i$
- $\bigcup_i V_i = \Omega$
- $V_i \cap V_j =$ Lebesque'sche Nullmenge, falls $i \neq j$

gelten. Das entspricht der Konstruktion eines dualen Gitters. Verbindet man dabei die Massenschwerpunkte eines Elementes mit den jeweiligen Kantenmittelpunkten (den Flächenschwerpunkten in 3D), so nennt man das Verfahren *barizentrische Finite Volumen*. Dieses Verfahren wollen wir verwenden. Für unser Gitter entsprechen die Kontollvolumina für innere Knoten dann genau den Bildvoxeln. Nur für Knoten, die auf dem Rand des Bildes liegen, sind sie entsprechend kleiner.
Die Schnittmenge $V_i \cap e$ eines Kontrollvolumens mit einem Element heißt mit der englischen Bezeichnung *subcontrol volume*.

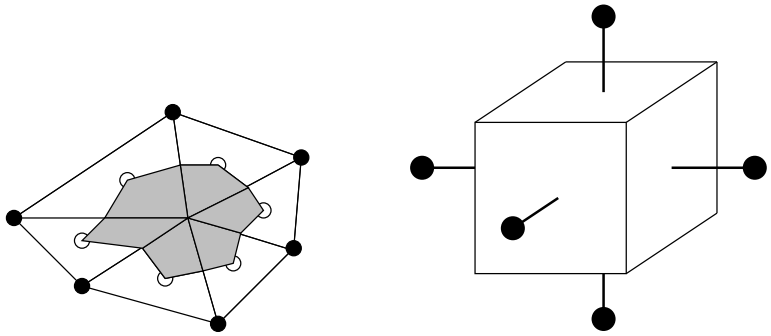

Abbildung 3.12: Kontrollvolumen in einem unstrukturierten Gitter (links) und für unser Gitter (rechts)

Als nächstes integrieren wir die Differentialgleichung über die Kontrollvolumina V_i:

$$\int_{V_i} \partial_t u \, dx = \int_{V_i} \nabla \cdot (D(u)\nabla u) \; dx \quad \forall V_i \tag{3.25}$$

Das Prinzip des Finite Volumen Verfahrens liegt nun darin, das Volumenintegral auf der rechten Seite in ein Randintegral umzuschreiben:

$$\int_{V_i} \nabla \cdot (D(u)\nabla u) \; dx = \int_{\partial V_i} (D(u)\nabla u) \cdot \vec{n} \, ds \quad \forall V_i \tag{3.26}$$

Dabei ist $\vec{n}$ die äußere Normale an ∂V_i.
Das haben wir schon bei der Herleitung der Diffusionsgleichung in Abschnitt 3.2.4 gesehen. Die zeitliche Veränderung einer physikalischen Konzentration innerhalb eines Gebietes ist gleich dem Fluss über den Rand.
Nun wollen wir u ja nur in den Knoten des Gitters berechnen. Dazu approximieren wir u durch eine Linearkombination von Funktionen:

$$u(x) = \sum_{j=1}^{|G|} \alpha_j \varphi_j(x) \tag{3.27}$$

Die Funktionen φ_j heißen Ansatzfunktionen oder Formfunktionen. Für sie soll gelten:

$$\varphi_j(x_i) = \delta_{ij}$$

Dann nämlich ist $u(x_i) = \alpha_i$ und die α_i sind eine diskrete Repräsentation von u. Außerdem wählt man die φ_j so, dass ihr Träger genau die Elemente sind, die den Knoten x_j als Eckpunkt haben.

Ersetzen wir u in (3.26) durch die Linearkombination (3.27), erhalten wir

$$\int_{\partial V_i} (D(u)\nabla u) \cdot \vec{n}\, ds \approx \int_{\partial V_i} (D(u)\nabla(\sum_j \alpha_j \varphi_j)) \cdot \vec{n}\, ds$$
$$= \sum_j \alpha_j \int_{\partial V_i} (D(u)\nabla \varphi_j) \cdot \vec{n}\, ds \quad \forall V_i \quad (3.28)$$

Als nächstes behandeln wir den Zeitterm der Gleichung (3.25). Wir approximieren die Zeitableitung durch eine einfache Vorwärtsdifferenz:

$$\int_{V_i} \partial_t u\, dx \approx \int_{V_i} \frac{u^{t+1} - u^t}{\tau}\, dx \quad \forall V_i$$

Dabei ist τ eine zu wählende Zeitschrittweite. Des Weiteren ersetzen wir die Integrale über V_i durch die Werte von u in den x_i :

$$\int_{V_i} \frac{u^{t+1} - u^t}{\tau}\, dx \approx \frac{\alpha_i^{t+1} - \alpha_i^t}{\tau} \cdot |V_i| \quad \forall V_i$$

Jetzt setzen wir die Gleichung wieder zusammen. Nachdem u nun in der Zeit diskretisiert wurde, müssen wir in (3.28) den α_j und u einen bestimmten Zeitpunkt zuweisen. Insbesondere ist damit D von festen Werten abhängig und somit linearisiert. Wählen wir t, also den alten Zeitpunkt, resultiert

$$\alpha_i^{t+1} = \alpha_i^t + \frac{\tau}{|V_i|} \left(\sum_j \alpha_j^t \int_{\partial V_i} \left(D(u^t)\nabla \varphi_j\right) \cdot \vec{n}\, ds \right) \quad \forall V_i \quad (3.29)$$

Mit Hilfe dieses Rechenschemas lassen sich die Werte α_i^{t+1} von u in den Knoten x_i zum Zeitpunkt $t+1$ explizit aus den Werten α_i^t zum Zeitpunkt t berechnen. Das Schema heißt deswegen auch *explizit*.
Wählen wir die α_j in (3.28) dagegen zum Zeitpunkt $t+1$, resultiert

$$\alpha_i^{t+1} = \alpha_i^t + \frac{\tau}{|V_i|} \left(\sum_j \alpha_j^{t+1} \int_{\partial V_i} \left(D(u^t)\nabla \varphi_j\right) \cdot \vec{n}\, ds \right) \quad \forall V_i \quad (3.30)$$

Diese Schema heißt *semiimplizit*, da zur Berechung der neuen Werte diese bereits mitberücksichtigt werden müssen.
Wählt man auch noch u in (3.28) zum Zeitpunkt $t+1$, nennt man das Schema *vollimplizit*.
Bei allen drei Schemata handelt es sich um ein System von $|G|$ Gleichungen. Dieses ist nichtlinear für den vollimpliziten und linear für den expliziten und den semiimpliziten Fall. Mit

$$a_{ij} = \frac{1}{|V_i|} \left(\int_{\partial V_i} (D(u)\nabla \varphi_j) \cdot \vec{n}\, ds \right) \quad (3.31)$$

schreibt man (3.29) als

$$\alpha^{t+1} = (\mathbb{I} + \tau A)\alpha^t \tag{3.32}$$

und (3.30) als

$$(\mathbb{I} - \tau A)\alpha^{t+1} = \alpha^t \tag{3.33}$$

Dabei nennen wir einen Eintrag $(\mathbb{I} \pm \tau A)_{ij}$ *Kopplung* der Knoten i und j.

Wir werden das semiimplizite Schema verwenden[5]. Dazu haben wir also in jedem Zeitschritt ein lineares Gleichungssystem mit $|G|$ Gleichungen und Unbekannten zu lösen. Auffallend dabei ist, daß aufgrund der Wahl der Ansatzfunktionen φ_j ein Matrixeintrag a_{ij} nur dann von Null verschieden ist, falls die Knoten i und j im Gitter benachbart sind. Bis auf eine feste Anzahl pro Zeile sind daher alle Einträge Null. Dies werden wir bei der Lösung der Gleichungssysteme ausnutzen.

Zur Berechnung der Matrixeinträge a_{ij} ist die Berechnung eines Oberflächenintegrals nötig. Um dies numerisch tun zu können, wird es durch eine Summe approximiert:

$$\begin{aligned} a_{ij} = \frac{1}{|V_i|}\left(\int_{\partial V_i} (D(u)\nabla\varphi_j(x)) \cdot \vec{n}\, ds\right) \\ \approx \frac{1}{|V_i|}\sum_k (D(u)\nabla\varphi_j(\mathrm{ip}_k)) \cdot \vec{n}_k \cdot |S_k| \end{aligned} \tag{3.34}$$

Dabei sind

- ip_k eine Anzahl von Stützstellen auf ∂V_i, die sogenannten Integrationspunkte
- $\vec{n}_k$ die äußere Normale an ∂V_i in ip_k und
- $|S_k|$ die Länge bzw. Fläche, des Teilstücks, auf dem ip_k liegt.

Üblicherweise wählt man pro Strecke bzw. Fläche, die die einzelnen subcontrol volumes des Kontrollvolumens begrenzen, genau einen Integrationspunkt, welchen man in den Mittelpunkt der Strecke bzw. Fläche setzt.
In unserem Gitter ist $|S_k| = 0.25$ für alle Integrationspunkte, da wir die einzelnen Voxel mit einer Kantenlänge von 1 in alle drei Raumdimensionen angenommen haben.

Offen ist nun noch die Berechnung der Ableitung der Ansatzfunktionen $\nabla\varphi_j(\mathrm{ip}_k)$ in den Integrationspunkten sowie die konkrete Wahl der Funktionen.

[5]Dies hat vor allem Stabilitätsgründe, zu denen wir auf Strauss[14] oder Grossmann/Ross[5] verweisen.

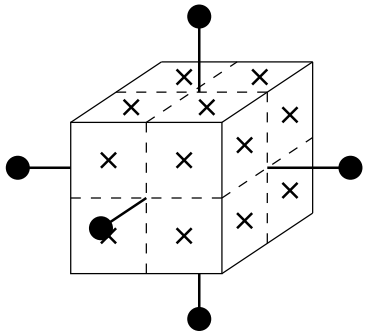

Abbildung 3.13: Integrationspunkte auf dem Kontrollvolumen

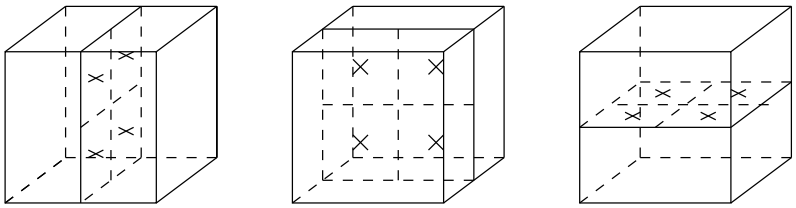

Abbildung 3.14: Integrationspunkte im Element

Man benutzt generell keine globalen Ansatzfunktionen. Diese kann man sich nämlich auch aus elementlokalen Ansatzfunktionen, die nur auf jeweils einem Element definiert sind, zusammengesetzt vorstellen.
Zur Konstruktion dieser elementlokalen Ansatzfunktionen definiert man ein Referenzelement mit festen lokalen Ansatzfunktionen, welche wir mit Φ_i bezeichnen. Über eine Transformationsfunktion $T(e)$ verbindet man das Referenzelement nun mit den Elementen e des Gitters. Dann bekommt man für die Ansatzfunktionen die Beziehung

$$\varphi_j(T_e(\xi,\eta,\vartheta)) = \Phi_{\mathrm{loc}(j,e)}(\xi,\eta,\vartheta) \tag{3.35}$$

Dabei bezeichnen ξ, η, ϑ die Koordinaten im Referenzraum und $\mathrm{loc}(j,e)$ ist eine Funktion, die zu einem Element e und einer globalen Knotennummer j die Nummer der zu j auf e gehörenden lokalen Ansatzfunktion liefert. Natürlich muss bei der Wahl der Φ_i die Bedingung (3.27) erfüllt bleiben.
Der Wert einer Ansatzfunktion an einem Integrationspunkt kann nun aus der Kenntnis der Koordinaten des Integrationspunktes im Referenzraum bestimmt werden.
Zur konkreten Form der Transformationsfunktionen für verschiedene Referenzelemente verweisen wir auf Schwarz[13].
Als Hexaederreferenzelement wählt man einen Würfel der Kantenlänge 1. Im Fall unseres Gitters ist die Transformation dadurch die Identität, wenn die Koordinatenachsen der beiden Räume gleich ausgerichtet sind. Wir werden daher bei der

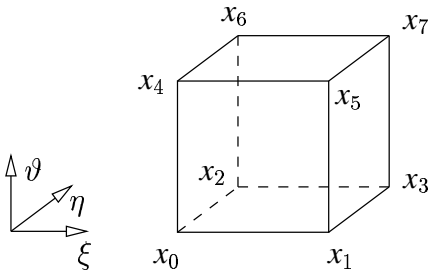

Abbildung 3.15: Referenzwürfel

Implementierung auf die richtige Nummerierung der Knoten achten müssen. Man beachte, dass auch dieser Vorteil nur deswegen gilt, weil wir die Abstände der Knoten in die einzelnen Raumrichtungen mit 1 angenommen haben.
Für die Ableitung der Ansatzfunktionen in den Integrationspunkten bekommen wir in unserem Fall nun:

$$\nabla\varphi_j(\mathrm{ip}(x,y,z)) = \nabla\Phi_{\mathrm{loc}(j,e)}(\mathrm{ip}(\xi,\eta,\vartheta))$$

Als Ansatzfunktionen im Referenzelement wählen wir schließlich die trilinearen Funktionen

$$\begin{aligned}
\Phi_0(\xi,\eta,\vartheta) &= (1-\xi)(1-\eta)(1-\vartheta)\\
\Phi_1(\xi,\eta,\vartheta) &= \xi(1-\eta)(1-\vartheta)\\
\Phi_2(\xi,\eta,\vartheta) &= (1-\xi)\eta(1-\vartheta)\\
\Phi_3(\xi,\eta,\vartheta) &= \xi\eta(1-\vartheta)\\
\Phi_4(\xi,\eta,\vartheta) &= (1-\xi)(1-\eta)\vartheta\\
\Phi_5(\xi,\eta,\vartheta) &= \xi(1-\eta)\vartheta\\
\Phi_6(\xi,\eta,\vartheta) &= (1-\xi)\eta\vartheta\\
\Phi_7(\xi,\eta,\vartheta) &= \xi\eta\vartheta
\end{aligned}$$

Jetzt müssen wir uns noch um die Realisierung der natürlichen Randbedingung kümmern. Dazu muss $D(u)\nabla u = 0$ in denjenigen Integrationspunkten, welche auf dem Rand von Ω liegen, gelten. Diese Forderung ist erfüllt, falls $\nabla u = 0$ und damit $\nabla\phi_j = 0\ \forall j$ in den Integrationspunkten gilt. Dazu lässt man bei der Berechnung der Matrixeinträge

$$a_{ij} = \frac{1}{|V_i|}\sum_k (D(u)\nabla\varphi_j(\mathrm{ip}_k))\cdot\vec{n}_k\cdot|S_k|$$

einfach die Beiträge der auf dem Rand liegenden Integrationspunkte weg.

Zuletzt betrachten wir die entstehenden Kopplungen in einem inneren Knoten i für den stationären Fall mit $D = \mathrm{diag}(1,1,1)$, also bei linearer Diffusion. Wir erhalten (auf vier Nachkommastellen gerundet)

$$\begin{bmatrix} 0.0469 & 0.1563 & 0.0469 \\ 0.1563 & 0.1875 & 0.1563 \\ 0.0469 & 0.1563 & 0.0469 \end{bmatrix}$$

$$\begin{bmatrix} 0.1563 & 0.1875 & 0.1563 \\ 0.1875 & -3.375 & 0.1875 \\ 0.1563 & 0.1875 & 0.1563 \end{bmatrix}$$

$$\begin{bmatrix} 0.0469 & 0.1563 & 0.0469 \\ 0.1563 & 0.1875 & 0.1563 \\ 0.0469 & 0.1563 & 0.0469 \end{bmatrix}$$

Dieses Schema soll das Gitter, auf dem die Kopplungen stattfinden, nachbilden. Der mittlere Eintrag entspricht der Kopplung a_{ii} des Knotens mit sich selbst, die waagerechte Richtung entspricht der x-Richtung, die senkrechte der y-Richtung. Die drei z-Ebenen sind von oben nach unten hintereinander angeordnet.
Die Kopplungen eines Knotens heißen auch *Stern*.

3.4.2 Numerische Diffusion

Wie im vorherigen Abschnitt beschrieben, liegen die Integrationspunkte normalerweise im Mittelpunkt der Oberflächen der subcontrol volumes. Dies hat im Fall der anisotropen Diffusion einen unerwünschten numerischen Effekt, den wir anhand eines zweidimensionalen Beispiels aufzeigen wollen. In drei Dimensionen gilt Analoges.

Beispiel

Betrachten wir ein zweidimensionales quadratisches Element.

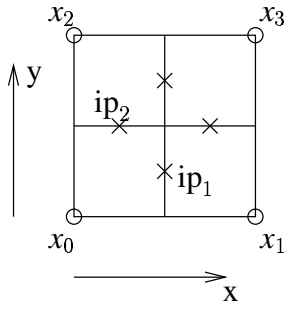

Abbildung 3.16: Das Element des Beispiels

Zum Knoten x_0 des Elementes gehören die Integrationspunkte

$$\mathrm{ip}_1 = \begin{pmatrix} 0.5 \\ 0.25 \end{pmatrix} \text{ und } \mathrm{ip}_2 = \begin{pmatrix} 0.25 \\ 0.5 \end{pmatrix}.$$

Der Diffusionstensor habe die Form

$$D := \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix},$$

d.h. es soll nur Diffusion in die y-Richtung stattfinden. Wir benutzen bilineare Ansatzfunktionen.
Wir betrachten die Kopplung der Punkte x_0 und x_1.

Mit $\varphi_1(\xi, \eta) = \xi(1 - \eta)$ und $\nabla\varphi_1(\xi, \eta) = (1 - \eta, -\xi)$ ist im ersten Integrationspunkt

$$(D \cdot \nabla\varphi_1(\mathrm{ip}_1)) \cdot \vec{n_1} = \left(D \cdot \begin{pmatrix} 0.75 \\ -0.5 \end{pmatrix}\right) \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} = (0, -0.5) \cdot \begin{pmatrix} 1 \\ 0 \end{pmatrix} = 0.$$

Der Beitrag wird durch den Diffusionstensor wie gewünscht Null.
Der entsprechende Beitrag von Integrationspunkt ip_2 ist

$$(D \cdot \nabla\varphi_1(\mathrm{ip}_2)) \cdot \vec{n_2} = \left(D \cdot \begin{pmatrix} 0.5 \\ -0.25 \end{pmatrix}\right) \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} = (0, -0.25) \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} = -0.25 \neq 0.$$

Hier entsteht ein Nicht-Null Beitrag zur Kopplung der beiden Knoten! Das liegt daran, dass zwar der Fluss korrekt in die gewünschte Diffusionsrichtung gelenkt wird, jedoch die äußere Normale in diesem Integrationspunkt nicht senkrecht dazu steht.

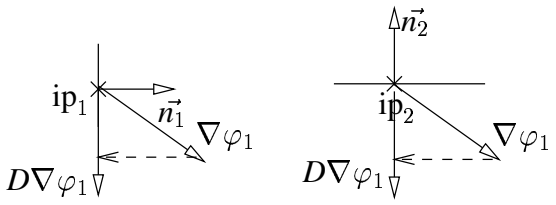

Abbildung 3.17: Schematische Entstehung der Kopplungen

Die beiden Knoten koppeln auch im in y-Richtung gesehen darunterliegenden Element miteinander. Wegen der Symmetrie der Ansatzfunktionen und der symmetrischen Lage der Integrationspunkte im Referenzelement können sich die Kopplungen jedoch nicht zu Null aufheben, was man durch Nachrechnen auch sofort bestätigt.

Insgesamt sieht der Stern im analogen dreidimensionalen Fall mit $D = \mathrm{diag}(0, 1, 0)$ für einen inneren Knoten (gerundet) so aus:

$$\begin{bmatrix} 0.0156 & 0.0938 & 0.0156 \\ -0.031 & 0.1875 & -0.031 \\ 0.0156 & 0.0938 & 0.0156 \end{bmatrix}$$
$$\begin{bmatrix} 0.0938 & 0.5625 & 0.0938 \\ -0.188 & -1.125 & -0.188 \\ 0.0938 & 0.5625 & 0.0938 \end{bmatrix}$$
$$\begin{bmatrix} 0.0156 & 0.0938 & 0.0156 \\ -0.031 & 0.1875 & -0.031 \\ 0.0156 & 0.0938 & 0.0156 \end{bmatrix}$$

Diese unerfreulichen Kopplungen kann man aber beseitigen, indem man die Lage der Integrationspunkte anders wählt.

Wir verschieben die Integrationspunkte auf der Oberfläche der subcontrol volumes in den gemeinsamen Schnittpunkt dieser mit den daran angrenzenden Elementen. Dies ist in den Abbildungen 3.18 und 3.19 gezeigt.

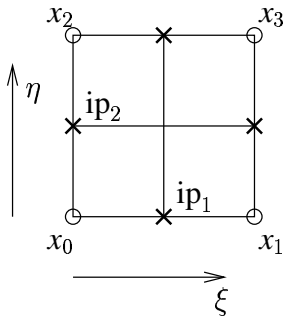

Abbildung 3.18: Verschobene Integrationspunkte im Referenzquadrat

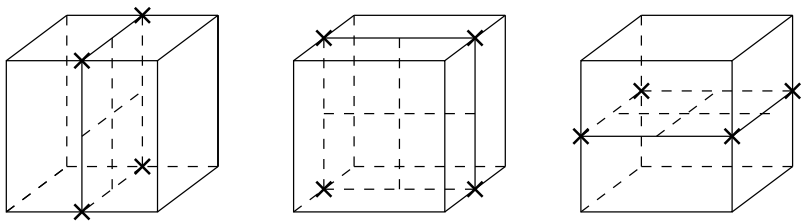

Abbildung 3.19: Verschobene Integrationspunkte im Referenzwürfel

In unserem Beispiel hat ip_2 nun die lokalen Koordinaten $(0, 0.5)$. Damit wird

$$(D \cdot \nabla\varphi_1(\mathrm{ip}_2)) \cdot \vec{n_1} = \left(D \cdot \begin{pmatrix} 0.5 \\ 0 \end{pmatrix}\right) \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \end{pmatrix} = 0$$

und die Knoten x_0 und x_1 sind somit entkoppelt. Der analoge 3D-Stern hat die Form

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$
$$\begin{bmatrix} 0 & 1 & 0 \\ 0 & -2 & 0 \\ 0 & 1 & 0 \end{bmatrix}$$
$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Übrigens führt diese Wahl der Lage der Integrationspunkte im Fall der isotropen linearen Diffusion auf den Stern, der auch bei der Finite-Differenzen Diskretisierung entsteht.

Einen Diffusionstensor wie im obigen Beispiel erhält man, falls eine lineare, in eine der Koordinatenachsen ausgerichtete Struktur vorliegt. Dies wird in unseren Bilddaten jedoch lediglich eine zufällige Ausnahme sein. Im Allgemeinen werden die Strukturen *nicht* in Koordinatenrichtung verlaufen.
Wir modifizieren unser obiges Beispiel daher dahingehend, dass wir als Diffusionstensor

$$D := \begin{pmatrix} 0.5 & 0.5 \\ 0.5 & 0.5 \end{pmatrix}$$

setzen. Dieser hat die Eigenvektoren

$$v_1 = \begin{pmatrix} 1 \\ 1 \end{pmatrix}, \quad v_2 = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

mit 1 und 0 als zugehörigen Eigenwerten. Die Diffusion soll also nur diagonal von links unten nach rechts oben oder umgekehrt verlaufen, was einer linearen, diagonal verlaufenden Struktur entspricht.
Mit den gleichen Bezeichnungen wie oben betrachten wir die Kopplung der beiden Knoten x_0 und x_2.

Unter Benutzung der im obigen Sinne verschobenen Integrationspunkte erhalten wir mit $\varphi_2(\xi,\eta) = (1-\xi)\cdot\eta$ und $\nabla\varphi_2(\xi,\eta) = (-\eta, 1-\xi)$ für die Kopplung von x_0 und x_2 in ip_1

$$(D\cdot\nabla\varphi_2(\mathrm{ip}_1))\cdot\vec{n_1} = \left(D\cdot\begin{pmatrix}0\\0.5\end{pmatrix}\right)\cdot\begin{pmatrix}1\\0\end{pmatrix} = \begin{pmatrix}0.25\\0.25\end{pmatrix}\cdot\begin{pmatrix}1\\0\end{pmatrix} = 0.25$$

und in ip_2

$$(D\cdot\nabla\varphi_2(\mathrm{ip}_2))\cdot\vec{n_2} = \left(D\cdot\begin{pmatrix}-0.5\\1\end{pmatrix}\right)\cdot\begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}0.25\\0.25\end{pmatrix}\cdot\begin{pmatrix}0\\1\end{pmatrix} = 0.25$$

Damit entsteht eine Nicht-Null Kopplung, die wiederum auch nicht durch die Integrationspunkte im Nachbarelement, über die x_0 und x_2 noch koppeln, aufgehoben wird.
Bei Verwendung der „gewöhnlichen“, nicht verschobenen Integrationspunkte ergeben sich für ip_1 der Wert 0.125, für ip_2 der Wert 0.25.
Diese Art der numerischen Diffusion ist nicht durch Wahl der Lage der Integrationspunkte in den Griff zu bekommen. Durch den Diffusionstensor werden die Gradienten auf v_1 projeziert. Da v_1 aber diagonal liegt, die Normalen in den Integrationspunkten aber stets in Koordinatenrichtung zeigen, kann das Skalarprodukt zwischen ihnen und v_1 nie Null werden:

$$\alpha\begin{pmatrix}1\\1\end{pmatrix}\cdot\vec{n} \neq 0 \quad \forall\alpha\in\mathbb{R}\setminus\{0\},\ \forall\vec{n}\in\left\{\begin{pmatrix}\pm1\\0\end{pmatrix},\begin{pmatrix}0\\\pm1\end{pmatrix}\right\}$$

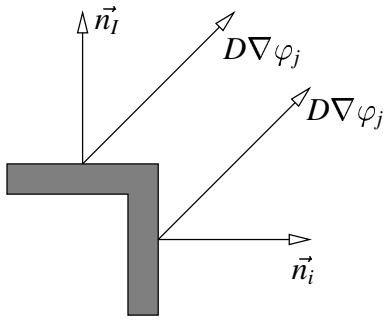

Abbildung 3.20: Entstehung der Kopplungen

Weil die Strukturen und damit die Eigenvektoren des Diffusionstensors im Fall unserer Neuronenaufnahmen meistens eben nicht parallel zu einer der Koordinatenachsen liegen, wird die Verwendung der verschobenen Integrationspunkte somit keinen Vorteil bringen.

Nun betrachten wir noch die Kopplungen zwischen den Knoten x_1 und x_2. Der Diffusionstensor soll beibehalten werden.

Unter Verwendung der verschobenen Integrationspunkte bekommen wir in ip_1

$$(D\cdot\nabla\varphi_2(\mathrm{ip}_1))\cdot\vec{n_1} = \left(D\cdot\begin{pmatrix}0\\0.5\end{pmatrix}\right)\cdot\begin{pmatrix}-1\\0\end{pmatrix} = \begin{pmatrix}0.25\\0.25\end{pmatrix}\cdot\begin{pmatrix}-1\\0\end{pmatrix} = -0.25$$

und in ip_2

$$(D\cdot\nabla\varphi_2(\mathrm{ip}_2))\cdot\vec{n_2} = \left(D\cdot\begin{pmatrix}-0.5\\0\end{pmatrix}\right)\cdot\begin{pmatrix}0\\1\end{pmatrix} = \begin{pmatrix}-0.25\\-0.25\end{pmatrix}\cdot\begin{pmatrix}0\\1\end{pmatrix} = -0.25$$

Unter Verwendung der gewöhnlichen Integrationspunkte erhielten wir entsprechend für ip_1 und ip_2 jeweils den Wert -0.125.
Die Knoten aber koppeln nur in diesen beiden Integrationspunkten. Damit erhalten wir eine negative Kopplung, welche einer Rückwärtsdiffusion entspricht!

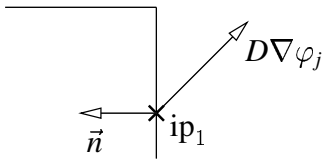

Abbildung 3.21: Entstehung der negativen Kopplungen

Geometrisch interpretiert entstehen die negativen Beiträge dadurch, daß der durch den Diffusionstensor auf die Diffusionsrichtung projezierte Gradient mit der Normalen einen Winkel von mehr als $90°$ bildet (siehe Abbildung 3.21).
Im analogen dreidimensionalen Fall, bei dem die Diffusion diagonal wie im obigen Beispiel auf einer Ebene verlaufen soll, erhält man den Stern

$$\begin{bmatrix}0 & 0 & 0\\0 & 0 & 0\\0 & 0 & 0\end{bmatrix}$$
$$\begin{bmatrix}-0.25 & 0.5 & 0.25\\0.5 & -2 & 0.5\\0.25 & 0.5 & -0.25\end{bmatrix}$$
$$\begin{bmatrix}0 & 0 & 0\\0 & 0 & 0\\0 & 0 & 0\end{bmatrix}$$

Aus theoretischer Sicht ist eine reine Rückwärtsdiffusion problematisch, da sie ein nicht korrekt gestelltes Problem darstellt. Wir haben jedoch keine reine Rückwärtsdiffusion, sondern nur einige rückwärts gewandte Kopplungen vorliegen. Es wäre interessant zu untersuchen, inwiefern sich diese auf die Korrektheit der Problemstellung auswirken.

Wir vermuten allerdings, dass die negativen Kopplungen in unseren Sternen kein Problem darstellen, da man an den Ergebnissen der Anwendungen in Kapitel 5 den glättenden Charakter des Verfahrens deutlich beobachten kann.
Da die negativen Kopplungen gerade normal zur Strukturrichtung liegen, erwarten wir durch sie sogar Vorteile bei der Erhaltung der Strukturkanten.

3.5 Löser für die entstehenden Gleichungssysteme

3.5.1 Notwendigkeit iterativer Löser

Durch die Diskretisierung wurde das Problem, die partielle Differentialgleichung zu lösen, auf das Lösen eines linearen Gleichungssystems zurückgeführt. Die Anzahl der Gleichungen und Unbekannten entspricht dabei der Anzahl der Knoten des Gitters, für uns also der Anzahl der Bildvoxel. Für den Fall eines vergleichsweise kleinen Bildwürfels von 65^3 Voxeln sind das bereits 274625 Unbekannte.

Für derart große Probleme erweist sich der Algorithmus der Gauß'schen Elimination als ungeeignet. Der Speicherbedarf für 274625^2 double-Werte beträgt bei acht Byte pro double bereits gut 600 GigaByte. An die Speicherung der entstehenden Matrizen mit $256 \times 256 \times 200$ Zeilen und Spalten für den Fall der Bildkubi, wie sie eigentlich bei der Mikroskopie entstehen, ist bei einem nötigen Speicherbedarf von weit über 1000 TeraByte nicht zu denken.
Vor allem aber seiner Komplexität von $\mathcal{O}(n^3)$ wegen ist das Gauß'sche Eliminationsverfahren aus Effizienzgründen nicht praktikabel, wenn man auch für große Probleme in angemessener Zeit eine Lösung berechnen will.

Bei der Betrachtung der Diskretisierungsmatrix stellt man fest, dass dessen Einträge a_{ij} nur dann von Null verschieden sind, falls die Indizees i und j zu im Gitter benachbarten Knoten gehören. Im Fall unserer Finite Volumen Diskretisierung gibt es damit pro Matrixzeile nur 27 von Null verschiedene Einträge. Wir haben es mit einer dünn besetzten Matrix zu tun. Eine $n \times n$ Matrix heißt *dünn besetzt*, falls $\#\{a_{ij} | a_{ij} \neq 0\} \in \mathcal{O}(n)$.
Offensichtlich ist es sinnvoll, nur die von Null verschiedenen Matrixeinträge zu speichern. Würde man dann allerdings das Gauß'sche Eliminationsverfahren auf eine solche Matrix anwenden, ginge die Dünnbesetztheit verloren, da zusätzliche Einträge erzeugt würden.

Es gibt jedoch eine anderen Klasse von Lösern, die sogenannten *iterativen Löser*, die die spezielle Matrixstruktur ausnutzen und die Dünnbesetztheit erhalten.

3.5.2 Prinzip iterativer Löser

Zur Berechnung der Lösung eines linearen Gleichungssystems $Ax = b$ konstruieren iterative Löser ausgehend von einem Startwert x^0 eine Folge x^i durch eine Iterationsvorschrift $x^{i+1} = \phi(x^i)$. Ein solches Verfahren heißt *linear*, falls ϕ linear

ist. Es heißt *konsistent*, falls für alle Fixpunkte $\bar{x}$ von ϕ zur rechten Seite b das Gleichungssystem $A\bar{x} = b$ erfüllt ist. Es heißt *konvergent*, falls für alle rechten Seiten b ein vom Startwert x^0 unabhängiger Grenzwert $x^* = \lim_{i\to\infty} x^i$ existiert. Für den Fall, dass es konvergiert, konvergiert ein konsistentes Verfahren also immer gegen die Lösung x von $Ax = b$.

Ein lineares konsistentes Verfahren ist durch

$$x^{i+1} = x^i - K^{-1}(Ax^i - b) \tag{3.36}$$

gegeben. Dabei ist K^{-1} eine sogenannte *angenährte Inverse* von A. Das ist eine aus A abgeleitete, aber einfacher, nämlich in $\mathcal{O}(n)$, zu invertierende Matrix. Die Berechnung der Iterierten x^{i+1} aus x^i ist allein mittels der Nicht-Null-Einträge der Matrizen A und K ausführbar. Für $K = A$ erhielte man die Lösung in einem Schritt. Das will man aber nicht, da das ja genau wieder die Durchführung des Gauß'schen Algorithmus erfordern würde.
Mit der Zerlegung $A = L + D + R$, wobei L der linke untere Teil von A, R der rechte obere Teil von A und D die Diagonale von A ist, erhalten wir als Beispiele für iterative Verfahren für

- $K = \omega\mathbb{I}$ das Richardson-Verfahren
- $K = D$ das Jacobi-Verfahren
- $K = L + D$ das Gauß-Seidel-Verfahren

Eine weitere Variante stellt die sogenannte ILU-Zerlegung dar. Dies ist eine LU-Zerlegung, die nicht vollständig (engl.: *incomplete*) durchgeführt wird, und auf die Zerlegung $A = LU - N$ führt, wobei N eine Restmatrix ist. Das ILU-Verfahren erhält man, indem man in (3.36) $K = LU$ setzt.
Zu Details der ILU-Zerlegung sowie zur Konvergenz der obigen Verfahren verweisen wir auf Hackbusch[6].

Eine andere Form der iterativen Verfahren stellen die sogenannten Gradientenverfahren dar. Sie beruhen auf der Tatsache, dass das Lösen des linearen Gleichungssystems $Ax = b$ äquivalent zur Minimierungsaufgabe

$$F(x) := \frac{1}{2}\langle Ax, x\rangle - \langle b, x\rangle \longrightarrow \min$$

ist. Zur Lösung der Minimierungsaufgabe und damit des linearen Gleichungssystems geht man iterativ vor, indem man in jedem Schritt eine eindimensionale Optimierungsaufgabe

$$f(\lambda) \to \min$$
$$f(\lambda) := F(x^i + \lambda p^i)$$

mit einer von der Iterierten x^i abhängigen *Suchrichtung* p^i löst und die Lösung als neue Iterierte x^{i+1} nimmt.
Die verschiedenen Gradientenverfahren entstehen im Wesentlichen aus der unterschiedlichen Bestimmung der Suchrichtungen p^i, in welcher aber der Gradient von F eine Rolle spielt, woher auch die Namensgebung der Verfahren stammt.
Die entstehenden Iterationsverfahren sind übrigens nicht mehr linear.
Zu allem Weiteren über Gradientenverfahren verweisen wir auf Hackbusch[6].

Als Abbruchkriterium für die Iterationsverfahren dient die Defektreduzierung. Der *Defekt* d^i der Iterierten x^i ist definiert als

$$d^i := Ax^i - b$$

Die Iteration wird abgebrochen, wenn $d^i < \epsilon d^0$ ist, wobei ϵ fest vorgegeben ist. Die Zahl

$$\bar{\varrho} = \left(\frac{\|d^i\|}{\|d^0\|} \right)^{\frac{1}{i}}$$

heißt mittlere Konvergenzrate (der ersten i Iterationen).

Mit der Definition des *Fehlers* $e^i := x^i - x$ kann man das Gleichungssystem $Ax = b$ auch als $Ae^i = d^i$ schreiben.

3.5.3 Mehrgitterverfahren

Das Problem der bislang vorgestellten iterativen Verfahren, die auf der Iterationsvorschrift (3.36) beruhen, ist deren langsame Konvergenz. Mit der Gitterweite $h = \frac{1}{n-1}$, wobei n die Anzahl der Gitterknoten in jede Raumrichtung angibt, liegt beispielsweise für den Fall der Poisson Gleichung $\Delta u = f$ die Konvergenzrate des Jacobi- und des Gauss-Seidel-Verfahrens in $\mathcal{O}(1-h^d)$. Für große Gitter sind damit sehr viele Iterationen bis zur gewünschten Defektreduzierung auszuführen.
Die schlechte Konvergenz beruht auf der Tatsache, dass die Verfahren zur Berechnung der neuen Iterierten eine Mittellung zwischen den Fehlerwerten in benachbarten Knoten ausführen. Der Fehler, der ja auf Null reduziert werden soll, wird somit lediglich geglättet, wird aber, wenn er erst einmal glatt ist, nur noch sehr langsam durch die Mittellung abgebaut. Daher werden die Verfahren auch *Glätter* genannt.
Abbildung 3.22 zeigt diese Wirkung von Glättern auf den Fehler.
Die Idee der Mehrgitterverfahren beruht darauf, nach einigen Glättungsschritten auf ein vergröbertes Gitter überzugehen. Auf dem vergröberten Gitter greift die lokale Nachbarschaft zwischen den Punkten weiter und der Fehler wird durch die Mittelung besser reduziert. Die Konvergenzrate liegt wegen der größeren Gitterweite h weniger nah bei 1.
Nach der Glättung auf dem vergröberten Gitter kann man auch dieses weiter vergröbern, bis man ein Gitter erreicht, auf dem das lineare Gleichungssystem direkt

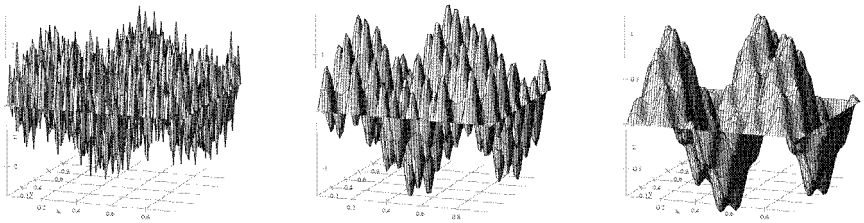

Abbildung 3.22: Wirkung von Glättern auf einen Initialfehler (links) nach einer Iteration (mitte) und nach fünf Iterationen (rechts). Entnommen aus Bastian[1].

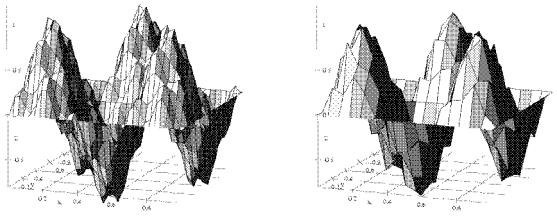

Abbildung 3.23: Der Fehler nach fünf Iterationen aus Abbildung 3.22 auf gröberen Gittern. Entnommen aus Bastian[1].

(oder effizient iterativ) gelöst werden kann. Danach geht man durch stufenweise Verfeinerungen mit Zwischenglättungsschritten wieder auf das ursprüngliche feine Gitter zurück. Dieser Mehrgitteralgorithmus ist in Abbildung 3.24 gezeigt. Der Indizee an den Vektoren und Matrizen entspricht dabei der Gitterstufe, zu der diese jeweils gehören.
Der Transfer eines Vektors auf ein gröberes Gitter heißt *Restriktion*, der Transfer auf ein feineres Gitter *Prolongation*. Den Fall $\gamma = 1$ nennt man *V-Zyklus*, den Fall $\gamma = 2$ *W-Zyklus*, bei welchem noch einmal auf das nächst gröbere Gitter gegangen wird, bevor man zum feineren Gitter zurückkehrt. Die Korrektur $x_l - = v_l$ wird als *Grobgitterkorrektur* bezeichnet. Die Werte ν_1 und ν_2 kennzeichnen die Anzahl der *Vorglättungs-* und *Nachglättungsschritte*.
Ein Durchlauf des Algorithmus 3.24 ist eine Iteration des Mehrgitterverfahrens. Ihr Aufwand liegt (falls γ nicht zu groß gewählt wird, siehe Wittum[16]) in $\mathcal{O}(n)$. Die Anzahl der zur Defektreduzierung nötigen Iterationen liegt in $\mathcal{O}(1)$. Das Verfahren weist somit optimale Komplexität auf.

Allerdings hängt die Konvergenz eines Mehrgitterverfahrens mit seinen zahlreichen Parametern immer auch von der zugrundeliegenden Differentialgleichung, der das lineare Gleichungssystem entspringt, ab. Wir wollen hier in diesem Zusammenhang auf zwei verschiedene Versionen der Transferoperationen eingehen.

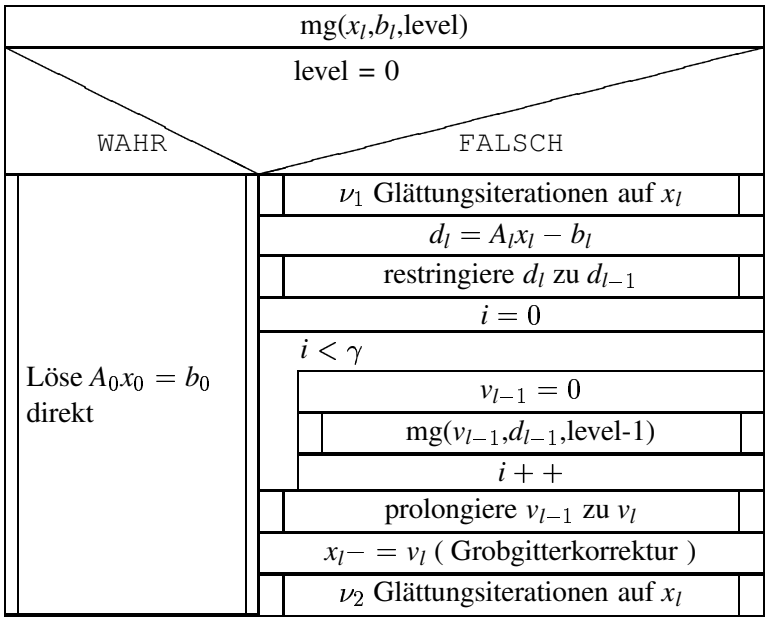

Abbildung 3.24: Mehrgitteralgorithmus

Linearer Transfer

Bei der linearen Prolongation werden die Werte in den Knoten des Feingitters linear durch die Werte in den Knoten des Grobgitters interpoliert[6]. Für den Fall eines strukturierten zweidimensionalen Gitters aus Quadraten, bei dem das Grobgitter durch Herausstreichen jeder zweiten Zeile und Spalte des Gitters entsteht, ist dies in Abbildung 3.25 gezeigt.

Normalerweise stellt man solche Transferoperationen in einer sogenannten *Stern-Schreibweise* dar. Die oben angegebene Prolongation hat darin die Darstellung

$$p = \begin{bmatrix} \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \\ \frac{1}{2} & 1 & \frac{1}{2} \\ \frac{1}{4} & \frac{1}{2} & \frac{1}{4} \end{bmatrix}$$

Die Stern-Schreibweise beschreibt somit die Wirkung der Transferoperation auf dem Gitter.

Die lineare Restriktion, bei der die Prolongation umgekehrt durchgeführt wird, hat in diesem Beispiel entsprechend die Form

$$p = \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix}$$

[6]bzw. bilinear in 2D, trilinear in 3D u.s.w. Wir werden aber immer nur von linearen Transfers sprechen und verweisen darauf, dass aus dem Zusammenhang klar ist, welche Linearität gemeint ist.

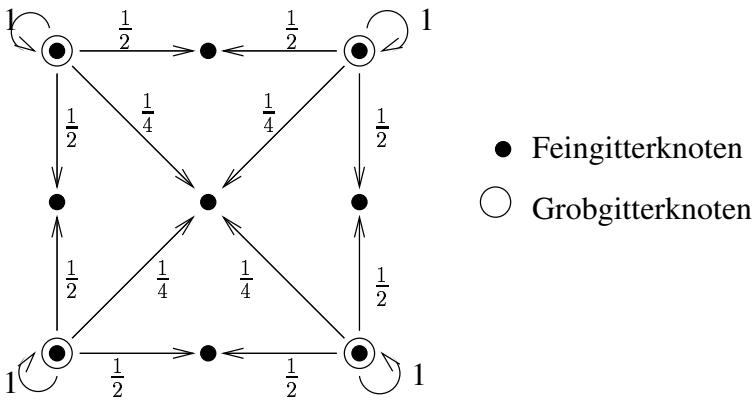

Abbildung 3.25: Lineare Prolongation

Matrixabhängiger Transfer
Wagner[15] beschreibt, daß für die lineare Form des Gittertransfers im Fall von Differentialgleichungen mit unstetigen Koeffizienten die Konvergenzrate des Mehrgitterverfahrens sehr schlecht ist. Diese läßt sich jedoch durch den Gebrauch der sogenannten *matrixabhängigen* Transferoperationen wieder verbessern. Die Gewichte, mit denen die Gitterknoten in die Berechnung eines prolongierten bzw. restringierten Knotens eingehen, entsprechen dabei der Kopplung der Knoten in der Diskretisierungsmatrix. So wird der durch die Differentialgleichung beschriebene Fluss durch die Transferoperationen nachgebildet.
Wir setzten als matrixabhängige Prolongation

$$f_i = (\sum_{j=\mathcal{N}_c(i)} |a_{ij}^f|)^{-1} \cdot \sum_{j=\mathcal{N}_c(i)} |a_{ij}^f| \cdot c_j$$

und als matrixabhängige Restriktion

$$c_i = (\sum_{j=\mathcal{N}_f(i)} |a_{ij}^f|)^{-1} \cdot \sum_{j=\mathcal{N}_f(i)} |a_{ij}^f| \cdot f_j$$

Dabei sind

- c der Grobgittervektor
- f der Feingittervektor
- a_{ij}^f die Diskretisierungsmatrix des Feingitters
- $\mathcal{N}(i)$ die Nachbarknoten von Knoten i im Grob- bzw. Feingitter

Im Fall unserer durch die Trägheitsmomente gesteuerten anisotropen Diffusion haben wir es mit einem varierenden Koeffizienten in der Differentialgleichung zu tun.

Wir werden in Abschnitt 5 testen, wie sich die beiden vorgestellten Transferoperationen auf die Konvergenz des Mehrgitterverfahrens auswirken.

Grobgittermatrix
Offen bleibt die Frage, wie man die Diskretisierungsmatrix auf den groben Gittern, den *Grobgitteroperator*, erhält.
Eine Möglichkeit ist die direkte Berechnung durch Diskretisierung der Differentialgleichung auf dem groben Gitter. Allerdings haben wir eine Differentialgleichung mit einem von der Lösung abhängigen, variierenden Koeffizienten vorliegen und diese Variation kann auf einem groben Gitter nicht aufgelöst werden. Das liegt daran, dass die feinen Strukturen der Neuronen nicht mehr dargestellt werden können. Daher wählen wir einen anderen Ansatz zur Bestimmung der Diskretisierungsmatrix auf den groben Gittern, den sogenannten *Galerkin-Ansatz*. Dabei wird der Grobgitteroperator über den Ansatz

$$A_{l-1} = rA_l p$$

bestimmt.

Weitere Aspekte und eine ausführliche Darstellung des Mehrgitterverfahrens findet man in Hackbusch[6].

4 Implementierung

4.1 Übersicht

Einen wesentlichen Teil dieser Arbeit stellt die Implementierung des im vorherigen Kapitel vorgestellten Filters dar. Dieser wurde im Rahmen des Gesamtprojektes der Nervenzellenrekonstruktion umgesetzt. Die verwendete Programmiersprache ist C++.
Neben Klassen für die benötigten Datenstrukturen enthält das Programmpaket Klassen zur Filterung, Segmentierung und Rekonstruktion.
Wir werden hier nur die zur Filterung nötigten Klassen beschreiben. Diese lassen sich in unterschiedliche Kategorien einteilen, wie es in Tabelle 4.1 dargestellt ist.

Kategorie	Klassen
Hilfsstrukturen	ARRAY VECTOR MULTIARRAY MULTIVECTOR
Zentrale Datenstruktur	VOLUME DATACUBE
Berechnung der Trägheitsmomente	INTEGRATION INTEGRAND Momentcalc MOMENTS
Matrix-Datenstruktur	SPARSEMATRIX LINE
Diskretisierung	FV_3D27
Löser	MULTIGRID
Steuerung des Filters	NLD

Tabelle 4.1: Einteilung der an der Filterung beteiligten Klassen in Kategorien

Eine schematische Übersicht der Beziehung zwischen den einzelnen Klassen ist in Abbildung 4.1 gezeigt.

Die einzelnen Klassen und damit verbundene Implementierungsaspekte werden nun in den folgenden Abschnitten dieses Kapitels beschrieben.

Bemerkung: Bei der Implementierung stütze ich mich zum Teil auf bereits existierende Software. Ich werde es an der jeweiligen Stelle erwähnen, falls eine entsprechende Klasse nicht von mir geschrieben wurde. Klassen ohne einen solchen Hinweis sind grundsätzlich komplett von mir geschrieben worden.

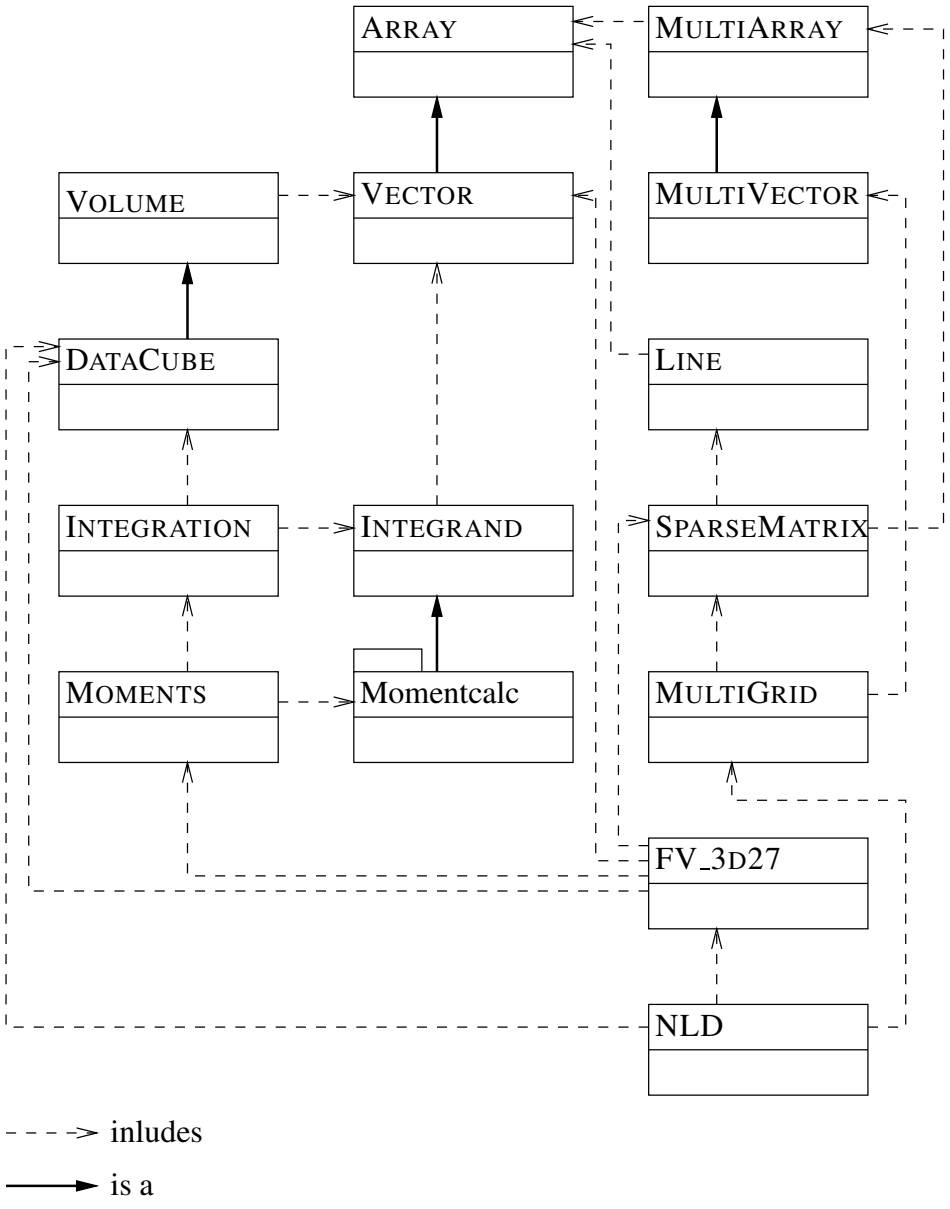

Abbildung 4.1: Übersicht der an der Filterung beteiligten Klassen

4.2 Hilfsstrukturen

Zunächst beschreiben wir einige Hilfsstrukturen, die wir an vielen Stellen benutzen werden. Diese wurden von Christoph Reisinger geschrieben.

4.2.1 Die Klasse ARRAY

Die Klasse ARRAY verwaltet ein Feld variabler Länge von Objekten. Sie ist eine Templateklasse, daher in der Wahl der Objekte flexibel. Die interne Speicherung ist durch einen Zeiger realisiert, Zugriff aber nur über Operatoren möglich. ARRAY verbindet somit die Vorteile der variablen Feldlänge mit der für Debugzwecke vorteilhaften Kontrolle des gültigen Speicherbereichs, die ja bei einfacher Verwendung von Zeigern nicht gegeben ist.
Zu bemerken ist, dass die Nummerierung der Speicherplätze entgegen der üblichen Konvention bei 1 beginnt.

t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}

Abbildung 4.2: Ein Array

4.2.2 Die Klasse VECTOR

VECTOR ist von ARRAY abgeleitet und stellt mathematische Vektoroperationen wie die Berechung der euklidischen Norm und arithmetische Operationen zur Verfügung.

4.2.3 Die Klasse MULTIARRAY

Die Klasse MULTIARRAY ermöglicht die Verwaltung mehrdimensionaler Felder unter Kontrolle des gültigen Speicherbereichs.. Als Speicher steht intern eine Instanz von ARRAY zur Verfügung. Daher ist auch MULTIARRAY eine Templateklasse und in der Wahl der zu speichernden Objekte flexibel.
Die Dimensionsverwaltung geschieht über eine zweite ARRAY Instanz, in der die Größe des Feldes in die einzelnen Dimensionen eingetragen ist. Durch Umrechung in einen globalen Feldindex ist so ein auf Angabe von Koordinaten basierender Zugriff möglich.
Die Zählung der Feldeinträge in die einzelnen Dimensionen beginnt bei 1.

4.2.4 Die Klasse MULTIVECTOR

Von MULTIARRAY abgeleitet, stellt MULTIVECTOR artihmetische Vektoropertationen zur Verfügung.

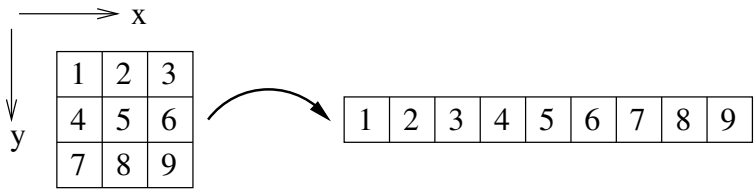

Abbildung 4.3: Beispiel für die Speicherverwaltung im Fall eines zweidimensionalen MULTIARRAY. Die Umrechnungsfunktion von Koordinaten auf globalen Index lautet $I(x, y) = x + s_1 \cdot (y - 1)$, wobei s_1 die Größe des Feldes in die erste Koordinatenrichtung ist.

4.3 Zentrale Datenstruktur

Die im Folgenden beschriebenen Datenstrukturen dienen der zentralen Aufgabe der Verwaltung des Bildvektors. Sie sind ursprünglich für den zwei- und den dreidimensionalen Fall geschrieben worden. Wir werden jedoch immer nur mit dreidimensionalen Bildern arbeiten.
Die beiden Klassen wurden gemeinsam von Philip Broser und mir geschrieben.

4.3.1 Die Klasse VOLUME

Die Klasse VOLUME stellt eine Datenstruktur zur Verwaltung zwei- oder dreidimensionaler Quader zur Verfügung.

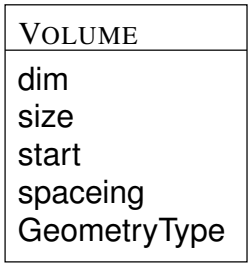

Abbildung 4.4: Die Klasse VOLUME

Wie jedes Bild eine Dimension und Größenangaben in die einzelnen Raumdimensionen hat, besitzt VOLUME ein Attribut dim und einen Vektor size vom Typ VECTOR, in denen die entsprechenden Werte gespeichert werden. Die Größe des Bildes in die einzelnen Raumdimensionen ist dabei die Anzahl der Voxel, die das Bild in die entsprechende Richtung besitzt.
Zusätzlich gibt es ein Attribut spaceing vom Tpy VECTOR, in dem die Abstände der Bildpunkte zueinander in die einzelnen Raumdimensionen eingetragen sind. Wir nehmen diese wie schon erwähnt als 1 an, benutzen daher die tatsächlich in

spaceing eingetragenen Werte nicht. Zur späteren Rekonstruktion aber werden sie benötigt.
Ein weiters Attribut ist ein Vektor start, ebenfalls vom Typ VECTOR, in dem die Koordinaten der linken unteren vorderen Ecke des Quaders gespeichert sind. Dieses wird nötig, wenn wir einzelne Ausschnitte des Bildes, also Teilbilder, festlegen wollen. Für das Gesamtbild ist start der Nullvektor.
Zuletzt hat VOLUME noch ein Attribut GeometryType, in dem angegeben werden kann, ob der Quader ganz normal als Quader, oder aber als Ellipsoid interpretiert werden soll. Mit den bisher gegebenen Attributen ist dies genauso gut möglich. Für die Speicherung unserer Bilddaten macht das keinen Sinn, wir werden dieses Attribut aber bei der Berechnung der Trägheitsmomente im Zusammenhang mit dem Festlegen von Bildausschnitten nutzen. Im Fall des Ellipsoids wird start als der Mittelpunkt des Ellipsoids interpretiert, size als die einzelnen Radien. Damit hat ein Ellipsoid immer Durchmesser mit einer geraden Anzahl von Voxeln, nämlich $2 \cdot$ size.

Die VOLUME Datenstruktur trägt noch keine Bilddaten.

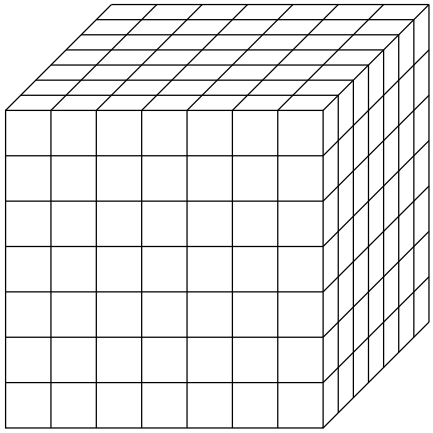

Abbildung 4.5: Ein dreidimensionales Volumen

4.3.2 Die Klasse DATACUBE

Die Klasse DATACUBE erweitert VOLUME durch ein zusätzliches Attribut, welches die Bilddaten speichert. Wir verwenden dazu ein Feld vom Tpy double. Die Bilddaten liegen also hintereinander im Speicher. Die Dimensonsverwaltung geschieht genauso wie in der Klasse MULTIARRAY, allerdings mit dem Unterschied, dass im DATACUBE die Zählung der Einträge in die einzelnen Raumrichtungen mit 0 beginnt. Wie im MULTIARRAY ist durch das Attribut size eine Umrechnung von Koordinaten in einen globalen Feldindex und damit koordinatenweiser Zugriff möglich.

4.4 Berechnung der Trägheitsmomente

Wir wollen - wie es in Abschnitt 3.3.3 beschrieben ist - die Eigenvektoren und Eigenwerte des durch

$$T_V(x) = \sum_{y \in V(x)} u(y) \cdot ((y - sp_V(x)) \otimes (y - sp_V(x))) \tag{4.1}$$

gegebenen vereinfachten Trägheitstensors berechnen. Dies wird in die Schritte

1. Berechnung des Trägheitstensors
2. Berechnung der Eigenwerte und Eigenvektoren

aufgeteilt.
Der erste Schritt verlangt eine Integration[1] über einen Ausschnitt $V(x)$ des Bildes. Um diese unabhängig von der Berechnung der Trägheitsmomente nutzen zu können, wurde sie in einer eigenen Klasse INTEGRATION implementiert.

4.4.1 Die Klasse INTEGRATION

Die Klasse INTEGRATION stellt Methoden zur Integration auf einem Ausschnitt der Datenstruktur DATACUBE, also des Bildes, zur Verfügung.
Der Ausschnitt, auf dem integriert werden soll, ist dabei als Parameter vom Typ VOLUME zu übergeben. Hier zeigt sich der Sinn der Auftrennung der zentralen Datenstruktur für das Bild in Größenangaben etc. und Bilddatenspeicherung; um einfach nur einen Ausschnitt des Bildes zu spezifizieren, ist es nicht nötig irgendwelche Bilddaten mitzuschleppen.
Grundsätzlich wird in INTEGRATION die Formel

$$I_V = \sum_{p \in V} g(u(p), p) \tag{4.2}$$

ausgewertet. Dabei sind V der gewählte Ausschnitt, u das Bild und g eine festzulegende Funktion. Durch die Wahl von g können unterschiedliche Integrationen ausgeführt werden. Dazu besitzt INTEGRATION als Attribut einen Zeiger auf ein Objekt der Klasse INTEGRAND, in dem eine entsprechende Implementation von g vorhanden ist (siehe Abschnitt 4.4.2). Über diesen Zeiger hat INTEGRATION Zugriff auf diese Implementierung.
Man beachte, dass auch hier das Voxelvolumen keine Berücksichtigung findet, also mit 1 angenommen wird. Zur Berücksichtigung der tatächlichen Voxelvolumina ist die Implementierung an dieser Stelle zu ändern.

Wir werden in den folgenden Abschnitten sehen, dass die Integrationsformel (4.2) zur Berechnung des Trägheitstensors nach (4.1) ausreicht.

[1] *Erinnerung:* Dass in Wahrheit nur eine Summe gebildet wird, liegt daran, dass die Bildfunktion als konstant auf den einzelnen Voxeln angenommen wurde.

Offen ist noch, wie das Bild u festgelegt wird. Dafür dient ein Zeiger auf ein Objekt vom Typ DATACUBE, der mittels einer Methode auf die gewünschte Instanz gelenkt werden kann.

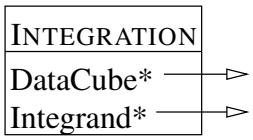

Abbildung 4.6: Die Klasse INTEGRATION

Die Integration ist grundsätzlich auf zwei Arten von Gebieten V möglich, nämlich quaderförmigen und ellipsenförmigen.
Die Klasse VOLUME, von dessen Typ der gewählte Ausschnitt V ja ist, besitzt wie in Abschnitt 4.3.1 erläutert ein Attribut GeometryType, das angibt, ob es als normaler Quader, oder aber als Ellipse interpretiert werden soll. *Der einfacheren Vorstellung halber werden wir im Folgenden in diesem Zusammenhang nur noch die Begriffe Kubus und Kugel statt Quader und Ellipse verwenden. Alle Ausführungen gelten aber mit Ausnahme des Kapitels 5 für die allgemeineren Körper Quader und Ellipse genauso.*
Im Falle eines kubusförmigen Ausschnittes wird die Integration nach dem in Abbildung 4.7 dargestellten Algorithmus ausgeführt.

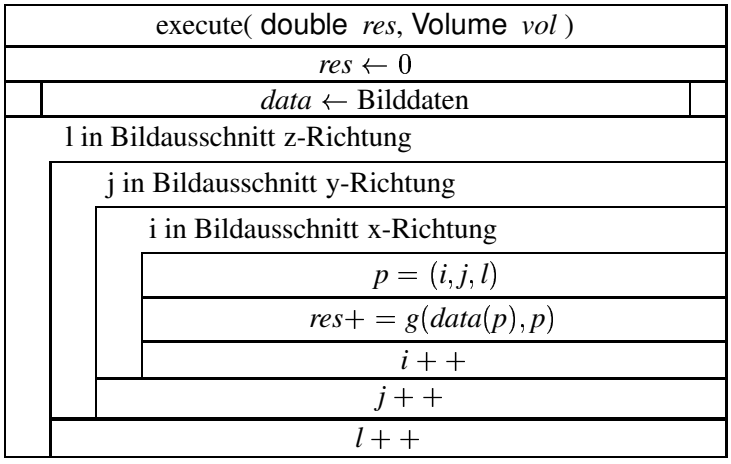

Abbildung 4.7: Integration auf einem kubusförmigen Gebiet

Stellt V dagagen eine Kugel dar, funktioniert das so nicht mehr, da Voxel nur noch teilweise, wenn die Kugeloberfläche sie schneidet, oder gar nicht mehr, falls sie komplett außerhalb der Kugel liegen, berücksichtigt werden dürfen.
Bei der Integration über ein kugelförmiges Gebiet wird daher zunächst ein Kubus

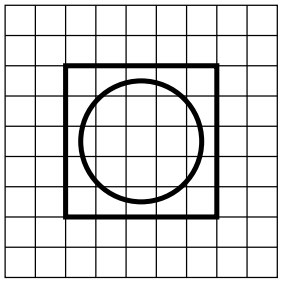

Abbildung 4.8: Kubus- und Kugelförmiger Ausschnitt eines DataCube

mit Seitenabmessungen, die den Durchmessern der Kugel in die einzelnen Raumdimensionen entsprechen, angelegt. In diesen werden dann für jeden Voxel x Gewichte $w(x)$ eingtragen, für welche

$$w(x) = \begin{cases} 1 & \text{Voxel komplett in Kugel enthalten} \\ 0 & \text{Voxel komplett außerhalb der Kugel} \\ a(x) & \text{sonst} \end{cases}$$

gilt, wobei $a(x)$ dem Anteil des Voxels, der innerhalb der Kugel liegt, entspricht.
Um diese Gewichte zu berechnen, benutze ich einen rekursiven Algorithmus, der wie folgt arbeitet:
Um das Gewicht eines bestimmten Voxels zu berechnen, wird überprüft, ob seine Eckpunkte alle innerhalb oder außerhalb des durch die Kugel spezifizierten Gebietes liegen. Trifft einer dieser Fälle ein, wird 1 bzw. 0 zurückgegeben. Anderenfalls wird der Voxel uniform in mehrere Teilvoxel zerlegt. Danach wird der Algorithmus rekursiv auf die Teilvoxel angewandt, bis eine maximale Verfeinerungsstufe erreicht ist. Das Gewicht des Voxels entspricht dann der Summe der Gewichte seiner Teilvoxel. Diejenigen Teilvoxel, die sich bei Erreichen der maximalen Verfeinerungsstufe immer noch teils innerhalb, teils außerhalb der Kugel befinden, erhalten den Wert 0.5 als Gewicht.
Abbildung 4.10 zeigt diese Verfeinerungsschritte und die Gewichte, die die Voxel auf der maximalen Verfeinerungsstufe zugewiesen bekommen. Der Algorithmus ist noch einmal in Abbildung 4.9 dargestellt.
Die Klasse INTEGRATION besitzt zwei Attribute, die die maximale Anzahl der Verfeinerungsschritte und die Anzahl der Teilvoxel, in die ein Voxel bei der Verfeinerung in jede Raumdimension zerlegt wird, angeben. Beim Start des Algorithmus muss der Parameter Volumen mit dem Wert 1 initialisiert werden.
Mittels der berechneten Gewichte führt man nun die Integration gemäß der Formel

$$I_V = \sum_{p \in V} w(p) \cdot g(u(p), p) \tag{4.3}$$

aus.

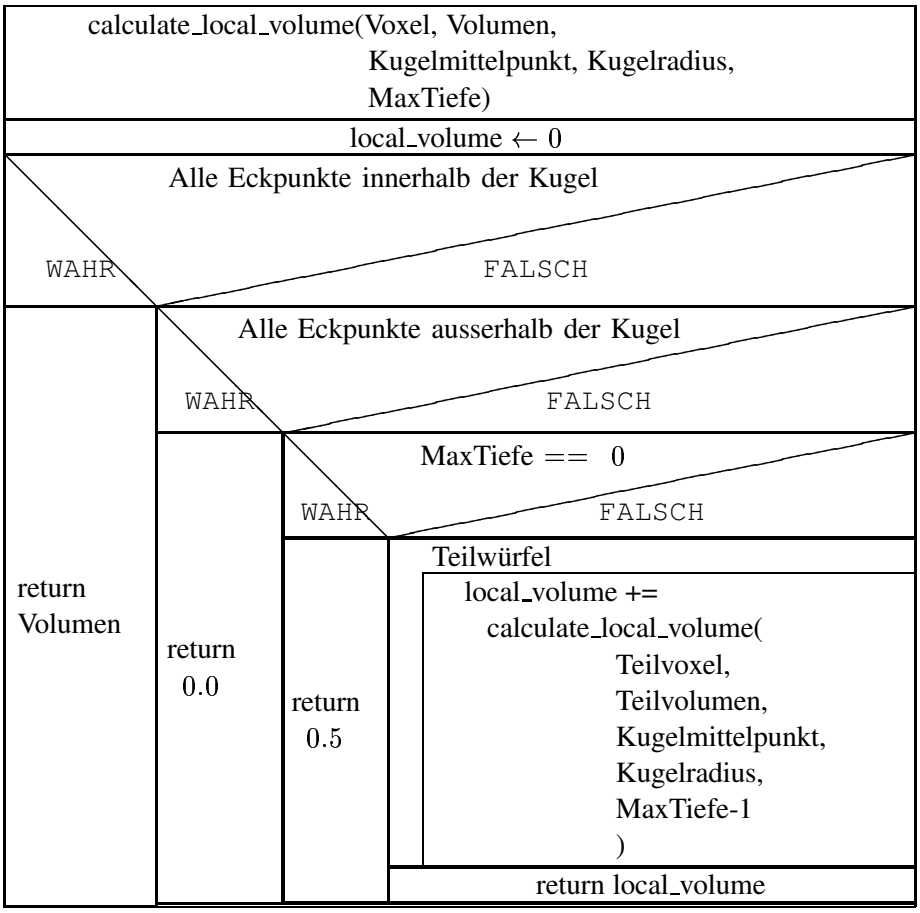

Abbildung 4.9: Algorithmus zur Berechnung des innerhalb eines festgelegten kugelförmigen Bildausschnitts liegenden Anteils eines Voxels

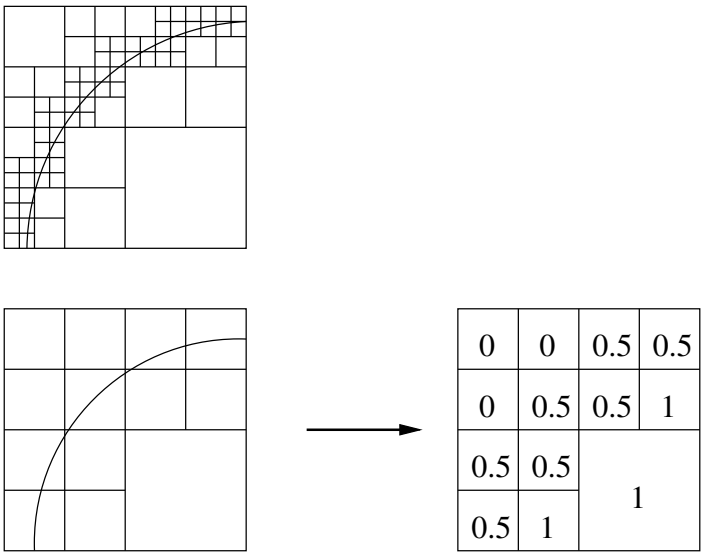

Abbildung 4.10: Adaptive Verfeinerung zur Berechnung der Gewichte bei der Kugelintegration und Zuordnung der Gewichte bei Erreichen der maximalen Verfeinerungsstufe

Der Grund für die Verwendung einer Kugel als Integrationsgebiet liegt in der Hoffnung die lokale Struktur in der Umgebung eines Voxels damit besser erfassen zu können als mit einem Kubus. Die Kugel ist nämlich im Gegensatz zum Kubus rotationsinvariant. Damit hängt die Strukturerkennung nicht von der relativen Ausrichtung der Struktur zu den Koordinatenachsen ab. Ob auf den zu filternden Daten tatsächlich eine unterschiedliche Strukturerkennung stattfindet und sich damit die Filterergebnisse bei Verwendung eines kubusförmigen Integrationsgebietes von denen bei Verwendung eines kugelförmigen unterscheiden, wird erst die Anwendung zeigen.

4.4.2 Die Klasse INTEGRAND

Wie im vorherigen Abschnitt erläutert, geschieht die Wahl der Funktion g aus Formel (4.2) durch Angabe einer Instanz der Klasse INTEGRAND. Allerdings kann INTEGRAND natürlich nur *eine* Implementierung einer Funktion g besitzen. Daher deklarieren wir g virtuell, leiten von INTEGRAND verschiedene Klassen ab, in denen wir g unterschiedlich implementieren, und lassen den INTEGRAND-Zeiger der INTEGRATION-Instanz auf das Objekt der abgeleiteten Klasse zeigen. Dadurch wird die gewünschte Variabilität von g erreicht.

4.4.3 Die Datei Momentcalc

In der Datei Momentcalc sind die von INTEGRAND abgeleiteten Klassen zusammengefasst. Diese implementieren die Funktion g aus (4.2), und zwar je nach

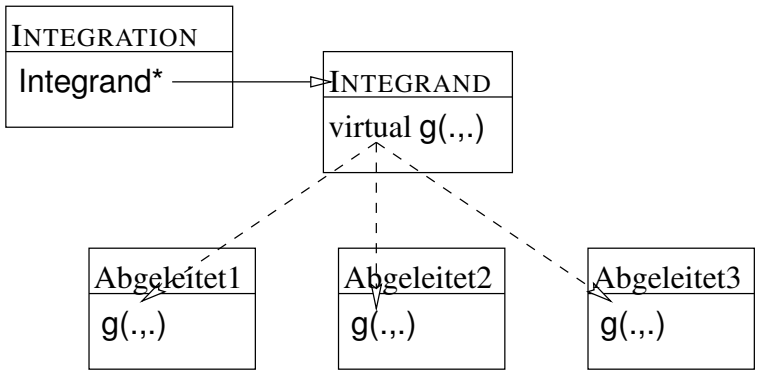

Abbildung 4.11: Konzept der virtuellen Funktion g von INTEGRAND

gewünschter Aufgabe. Im einzelnen stehen zur Verfügung:

- VOLUMEINTEGRAND zur Berechnung des Volumens mit

$$g(u(p), p) = 1$$

- MASSINTEGRAND zur Berechnung der Masse mit

$$g(u(p), p) = u(p)$$

- FOCAL_POINT_X, FOCAL_POINT_Y, FOCAL_POINT_Z zur Berechnung der einzelnen Schwerpunktskoordinaten mit

$$g(u(p), p) = \begin{cases} u(p) \cdot p_x & \text{für FOCAL\_POINT\_X} \\ u(p) \cdot p_y & \text{für FOCAL\_POINT\_Y} \\ u(p) \cdot p_z & \text{für FOCAL\_POINT\_Z} \end{cases}$$

  Dabei sind p_x, p_y und p_z die Koordination des Bildpunktes p.

- INERTIAN_TENSOR, INERTIAN_TENSOR11, INERTIAN_TENSOR12, INERTIAN_TENSOR13, INERTIAN_TENSOR22, INERTIAN_TENSOR23, INERTIAN_TENSOR33 zur Berechnung der Einträge des Trägheitstensors[2]

Eigentlich muss die Funktion g z.B. für INERTIAN_TENSOR12 so aussehen:

$$g(u(p), p) = u(p) \cdot (p_x - \mathrm{sp}_x) \cdot (p_y - \mathrm{sp}_y) \tag{4.4}$$

Dabei ist sp der Schwerpunkt. Dieser muss im Vorhinein berechnet werden und ist für alle Tensoreinträge gleich. Daher leiten wir die einzelnen Klassen zur Berechnung der Einträge des Tensors von einer gemeinsamen Superklasse INERTIAN_TENSOR (die dann wiederum von INTEGRAND abgeleitet ist) ab, in der wir

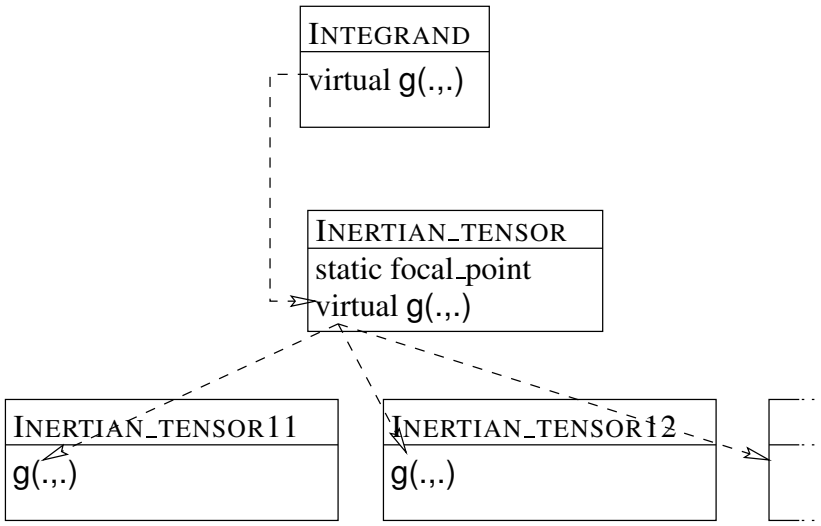

Abbildung 4.12: Zusammenhang der INERTIAN_TENSOR-Klassen

eine Klassenvariable anlegen, in welcher der Schwerpunkt spezifiziert ist. Auf diesen haben die unterschiedlichen Implementierungen von g nun Zugriff.
Es gibt aber noch eine andere Möglichkeit zur Berechnung des Trägheitstensors, die ohne eine solche Klassenvariable auskommt. Dazu nehmen wir eine Umformung der Formel (3.15) zur Berechnung des vereinfachten Trägheitstensors vor:

$$
\begin{aligned}
& (T_V)_{ij}(x) \\
&= \int_V u(y) \cdot (y_i - \mathrm{sp}_i(x)) \cdot (y_j - \mathrm{sp}_j(x))\ dy \\
&= \int_V u(y) \cdot (y_i y_j - y_i \mathrm{sp}_j(x) - y_j \mathrm{sp}_i(x) + \mathrm{sp}_i(x)\mathrm{sp}_j(x))\ dy \\
&= \int_V u(y) y_i y_j\ dy - \mathrm{sp}_j(x) \int_V u(y) y_i\ dy \\
&\quad - \mathrm{sp}_i(x) \int_V u(y) y_j\ dy + \mathrm{sp}_i(x)\mathrm{sp}_j(x) \int_V u(y)\ dy \\
&= \int_V u(y) y_i y_j\ dy - \mathrm{sp}_j(x)\mathrm{sp}_i(x) m_V(x) \\
&\quad - \mathrm{sp}_i(x)\mathrm{sp}_j(x) m_V(x) + \mathrm{sp}_i(x)\mathrm{sp}_j(x) m_V(x) \\
&= \int_V u(y) y_i y_j\ dy - \mathrm{sp}_i(x)\mathrm{sp}_j(x) m_V(x)
\end{aligned}
$$

In der zur Auswertung des Trägheitstensors nötigen Integration ist nun der Schwerpunkt nicht mehr vorhanden.

[2] Wegen der Symmetrie des Tensors reichen diese aus

Wir können daher in den einzelnen INERTIAN_TENSOR-Klassen

$$g(u(p), p) = u(p) \cdot p_i \cdot p_j \,, \quad i, j = x, y, z \tag{4.5}$$

setzen und den Term, in dem die Schwerpunktskoordinaten auftreten, nachträglich subtrahieren.

An dieser Stelle kann man bemerken, daß es nach der Umformung des Trägheitstensors nicht mehr nötig ist, die unterschiedlichen Funktionen g überhaupt innerhalb von Klassen zu verpacken. Ein Funktionszeiger als Attribut von INTEGRATION, auf die jeweils gewünschte Funktion gelenkt, würde genauso gut funktionieren.

4.4.4 Die Klasse MOMENTS

In der Klasse MOMENTS befinden sich die zur Berechnung der Trägheitsmomente aufzurufenden Methoden. Zur Verfügung stehen Methoden zur

- Berechnung des Schwerpunktes,
- Berechnung des Trägheitstensors und
- Berechnung der Trägheitsmomente

jeweils eines Bildausschnittes V. Als Parameter ist den einzelnen Methoden eine Instanz der Klasse VOLUME zu übergeben, um den Bildbereich zu spezifizieren, zu dem die Berechnung ausgeführt soll (vgl. Abschnitt 3.3.3). Das Ergebnis ist bezogen auf den Voxel im Mittelpunkt des Volumens, das heißt, es können nur solche Volumina angegeben werden, deren Größe in jede Raumdimension einer ungeraden Anzahl von Voxeln entspricht. Je nach Methode müssen noch Vektoren zur Aufnahme des Ergebnisses übergeben werden.
Weiter steht eine Methode zur

- Berechnung der Trägheitsmomente zu jedem Punkt des DATACUBE

bereit. Neben einem Volumen müssen hier zwei Felder von Vektoren übergeben werden, deren Länge der Anzahl der Punkte des DATACUBE entspricht. In die in den einzelnen Stellen der Felder stehenden Vektoren werden die Eigenwerte bzw. die Eigenvektoren des Trägheitstensors des zu der Feldstelle gehörenden Bildpunktes geschrieben.

Die Berechnung stützt sich auf die Integration ab. Dazu besitzt MOMENTS ein Attribut der Klasse INTEGRATION. Nach der Integration werden anfallende Restberechnungen durchgeführt, wie Division durch die Masse, Subtraktion des Schwerpunktterms bei den Trägheitstensoreinträgen und die Berechnung der Eigenwerte und Eigenvektoren.

Zur Festlegung des Bildes, auf dem die Trägheitsmomente berechnet werden sollen, gibt es als Attribut einen Zeiger auf ein Objekt vom Typ DATACUBE. Dieses wird intern an das INTEGRATION-Objekt weitergeleitet.

Die einzelnen oben angegebenen Methoden stützen sich aufeinander ab. Die Methode zur Berechnung des Trägheitstensors benutzt die Schwerpunktberechnung, die Methode zur Berechnung der Trägheitsmomente diejenige zur Berechung des Tensors und die zur Berechnung der Trägheitsmomente in allen Punkten des DATACUBE diejenige zur Berechnung der Werte in einem Punkt. Das ist noch einmal in der Abbildung 4.13 dargestellt.

Das Volumen, das bei der Berechnung der Trägheitsmomente in allen Bildpunkten übergeben werden muss, wird vor der intern verwendeten Berechnung der Werte in einem Punkt jeweils neu um diesen zentriert.

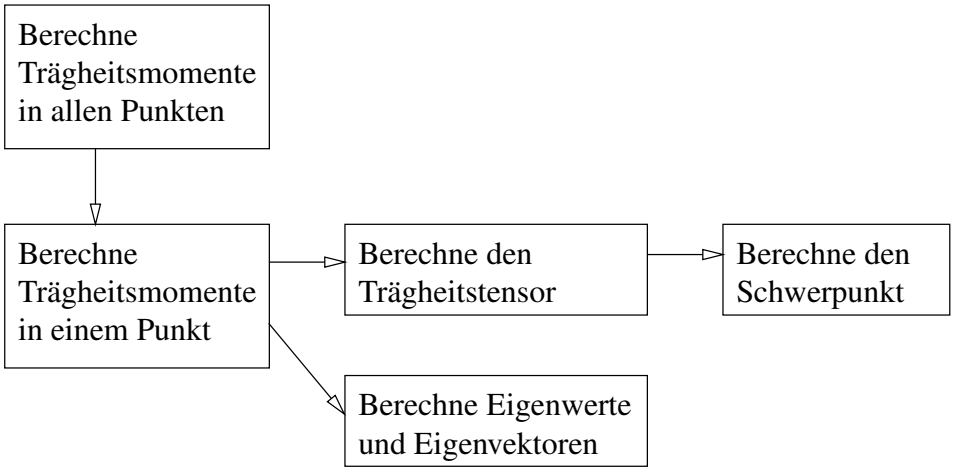

Abbildung 4.13: Abstützung der einzelnen Methoden von MOMENTS

Das Verfahren, nach dem die Eigenwerte und Eigenvektoren des Trägheitstensors berechnet werden, wird im folgenden Abschnitt 4.4.5 erklärt.

Die Berechnung der Trägheitsmomente für sämtliche Punkte eines DATACUBE in der oben dargestellten Weise ist leider sehr ineffizient. Das liegt daran, dass jeder einzelne Bildpunkt dabei sehr oft besucht wird. Wir werden das im übernächsten Abschnitt genauer erklären.
Um die Berechnung schneller ausführen zu können, stellt MOMENTS daher andere Methoden zur Verfügung, die effizienter arbeiten. Diese werden in den Abschnitten 4.4.6 und 4.4.7 erläutert.

4.4.5 Berechnung der Eigenwerte und Eigenvektoren

Zur Berechnung der Eigenwerte und Eigenvektoren der Trägheitstensoren benutzen wir einen auf dem QR-Algorithmus beruhenden Programmcode der Numerical Recipes[11]. Den QR-Algorithmus werden wir hier nur kurz skizzieren. Für genauere Beschreibungen sowie die nicht weiter ausgeführten Punkte verweisen wir auf die Numerical Recipes[11] oder auf Schwarz[12].

Jede quadratische Matrix A lässt sich als Produkt einer orthogonalen Matrix Q und einer rechten oberen Dreiecksmatrix R schreiben:

$$A = QR$$

Wegen $R = Q^T A$ erhält man zu A die orthogonal ähnliche Matrix

$$A' = RQ = Q^T AQ$$

Damit kann man die *QR-Iteration*

$$A_{k+1} = Q_k^T A_k Q_k \tag{4.6}$$

definieren.
Sind nun $|\lambda_1| > |\lambda_2| > \ldots > |\lambda_n|$ die betragsmäßig paarweise verschiedenen Eigenwerte von A, so konvergiert (4.6) gegen eine rechte obere Dreiecksmatrix und es ist

$$\lim_{k\to\infty} a_{ii}^k = \lambda_i, \quad i = 1, \ldots, n$$

Die Konvergenz kann jedoch recht langsam sein, falls die Beträge zweier Eigenwerte nahe beieinander liegen. Es gilt nämlich für die Subdiagonalelemente

$$|a_{i+1,i}^k| \approx \left(\frac{\lambda_{i+1}}{\lambda_i}\right)^k$$

Zur Beschleunigung der Konvergenz verwendet man daher sogenannte *shifts*. Man benutzt die Matrix $A - \sigma\mathbb{I}$, welche die Eigenwerte $\lambda_i - \sigma$ besitzt. Die QR-Zerlegung $A - \sigma\mathbb{I} = QR$ führt dann auf die QR-Iteration

$$A_{k+1} - \sigma_k\mathbb{I} = Q_k^T(A_k - \sigma_k\mathbb{I})Q_k$$

Für die Subdiagonalelemente gilt dann

$$|a_{i+1,i}^k| \approx \left(\frac{\lambda_{i+1} - \sigma_k}{\lambda_i - \sigma_k}\right)^k$$

Statt dieser expliziten verwendet der Code der Numerical Recipes sogenannte *implizite Shifts*. Außerdem benutzt er keine QR-Zerlegung, sondern eine QL-Zerlegung, wobei L eine linke untere Dreiecksmatrix ist. Dafür gelten die obigen Aussagen aber analog.

Der QR-Algorithmus ist besonders effizient für tridiagonale Matrizen. Die QR-Iteration konvergiert dann gegen eine Diagonalmatrix. Daher transformieren wir A mit Hilfe eines zweiten Codes der Numerical Recipes vor dem Start der QR-Iteration auf Tridiagonalgestalt. Dazu wird in den Numerical Recipes das Verfahren von Householder verwendet.

Zur Berechnung der Eigenvektoren setzen wir

$$Q := Q_1 Q_2 \dots Q_m$$

als Produkt aller orthogonalen Matrizen Q_k, die bei der QR-Iteration auftreten. Damit ist für eine Tridiagonalmatrix A

$$D = Q^T A Q$$

und die Eigenwertaufgabe $Av = \lambda v$ wird mit

$$w := Q^T v$$

in

$$Dw = \lambda w$$

überführt.
Damit berechnen sich die Eigenvektoren nach $v_i = Qw_i$, wobei $w_i = e_i$ ist.

4.4.6 Beschleunigung durch Zusammenfassung von Punkten

Wie erwähnt ist die naive Berechnung der Trägheitstensoren auf dem ganzen Bild durch sukzessive Berechnung des Tensors in jedem Bildpunkt sehr ineffizient. Für jeden Punkt x_j wird bei der dafür nötigen Integration nämlich eine Umgebung $V(x_j)$ von Punkten $x_k \in V$ abgearbeitet. Jeder dieser Punkte x_k ist jedoch wiederum in Umgebungen anderer Punkte enthalten und wird so vielfach besucht. Dabei findet bei jedem Besuch eine Addition sowie ein Funktionsaufruf mit eventuellen weiteren artihmetischen Operationen statt.
Es ist daher angebracht, die Anzahl der Besuche eines Punktes zu verringern.

Bei Betrachtung der Umgebungen fällt auf, daß viele davon sich derart überlappen, dass sie nicht nur einen, sondern ganze Bereiche von gemeinsamen Punkten besitzen. Die Idee der Beschleunigung der Berechnung liegt nun darin, einen geeigneten Bereich B von Punkten zusammenzufassen, die nach der Formel (4.2) benötigte Summe für diesen zu bilden und das Ergebnis an diejenigen Punkte, in deren Umgebung B komplett enthalten ist, zu verteilen. Bei dieser Vorgehensweise wird die Summe auf B nur einmal berechnet und nicht entsprechend der Anzahl der Umgebungen, die B enthalten.

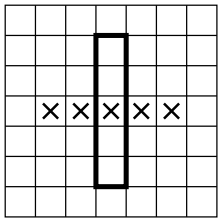

Abbildung 4.14: Bei einer quadratischen Umgebung $V(x_j)$ mit Seitenlänge $s = 5$ ist der markierte Ausschnitt in den Umgebungen aller fünf markierten Voxel komplett enthalten

Wir definieren also eine Menge

$$\mathcal{B} = \{B_i \mid i \in I\}$$

von solchen Bereichen.
Wichtig ist dabei, dass zu sämtlichen Umgebungen $V(x_j)$ eine disjunkte Zerlegung

$$V(x_j) = \bigcup_{i \in J \subset I} B_i, \quad B_{i_1} \cap B_{i_2} = \begin{cases} B_{i_1} = B_{i_2} & i_1 = i_2 \\ \varnothing & i_1 \neq i_2 \end{cases} \quad \forall B_{i_l} \in J$$

in eine Teilmenge dieser Bereiche existiert.
Dann nämlich ist

$$\sum_{x_k \in V(x_j)} g(u(x_k), x_k) = \sum_{i \in J} \sum_{x_k \in B_i} g(u(x_k), x_k) \quad \forall x_j$$

und die Summe über $V(x_j)$ als Addition von Teilsummen über die Bereiche B_i, aus denen $V(x_j)$ besteht, berechenbar. Für jede Umgebung $V(x_j)$ sind die Bereiche B_i, aus denen diese aufgebaut ist, zwar disjunkt, jedoch können sich Bereiche, die zu unterschiedlichen Umgebungen gehören, sehr wohl überlappen.

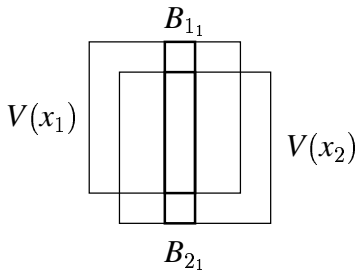

Abbildung 4.15: Überlappende Bereiche

Wir wählen als Bereiche B_i zweidimensionale[3] Scheiben mit Normalenrichtung z, wovon wir jeweils eine um jeden Punkt x_j zentrieren. Die Größe der Scheiben in x-

[3]Damit meinen wir, dass die Scheiben in eine der Raumrichtungen nur einen Voxel breit sind.

und y-Richtung soll die des spezifizierten Inegrationsgebietes, also der Umgebungen $V(x_j)$ sein. In die z-Richtung haben sie damit keinerlei Überlappung und ein $V(x_j)$ besteht aus in z-Richtung nebeneinander angeordneten Scheiben.

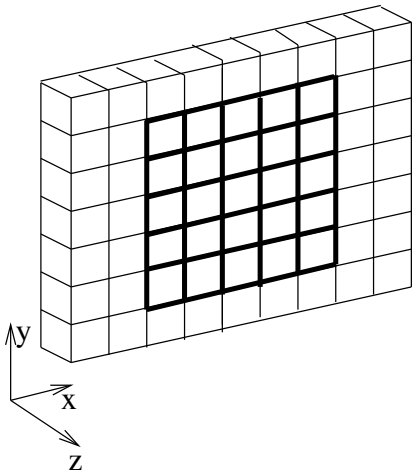

Abbildung 4.16: Scheibe von Punkten

Um die zur Tensorberechung nötigen Integrationen für alle Punkte auf einer Geraden

$$g := \begin{pmatrix} a \\ b \\ z \end{pmatrix}, \; a, b \text{ fest}$$

durchzuführen, ist es jetzt nur noch notwendig, die einzelnen Scheiben in den Punkten auf g in der oben beschriebenen Weise abzuarbeiten.

Der Algorithmus als Ganzes, der aus diesem Vorgehen resultiert, ist in Abbildung 4.17 gezeigt. Dabei wird die Klasse `MomentCube` benutzt. Diese ist von VOLUME abgeleitet und bietet zehn Felder, um die zehn[4] zur Trägheitsmomentenberechnung benötigten Werte in jedem Bildpunkt zu speichern. Die Speicherverwaltung geschieht dabei analog zu der in DATACUBE. Um in das jeweilige Feld, für das die Werte berechnet werden sollen, zu schreiben, wird im Algorithmus ein Zeiger current_mom auf das entsprechende Feld gelenkt und über diesen geschrieben.
Parameter des Algorithmus sind zwei Felder von Vektoren, in die wie bisher die Eigenwerte und Eigenvektoren geschrieben werden.

[4] einen für die Masse, drei für den Schwerpunkt, sechs für den Trägheitstensor

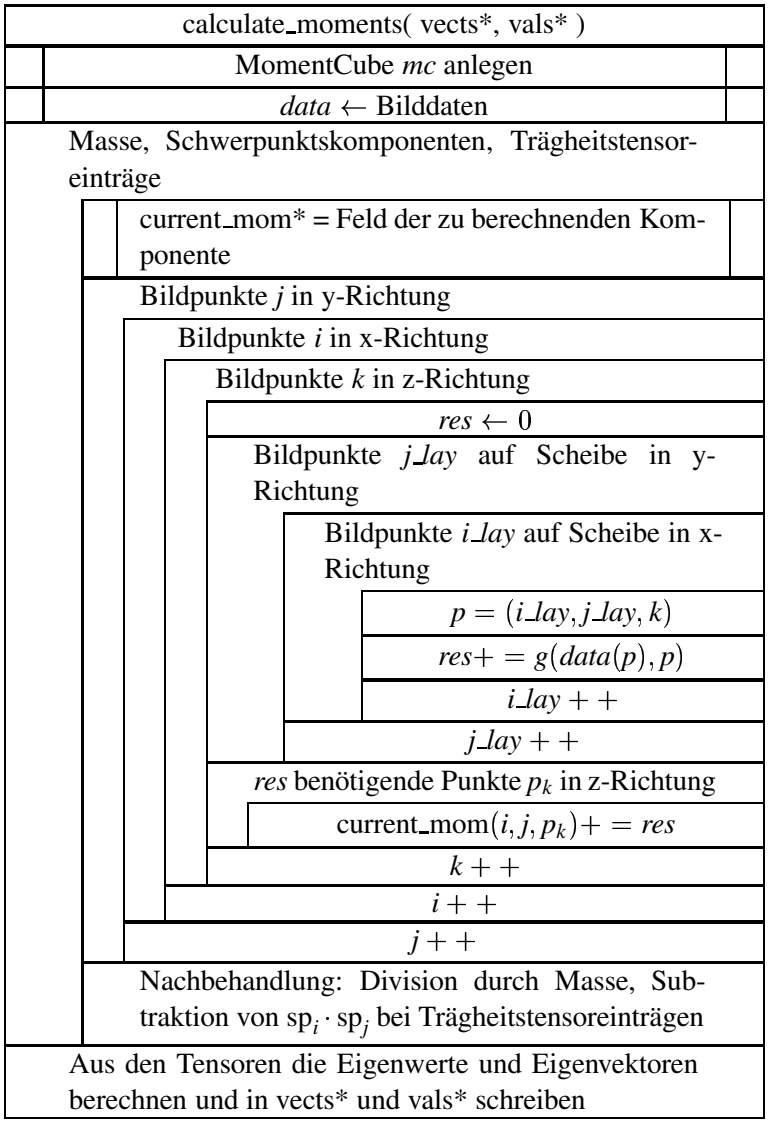

Abbildung 4.17: Algorithmus zur Berechnung der Trägheitsmomente durch Zusammenfassung von Punkten zu Scheiben

Aufwand
Nun wollen wir den Aufwand der neuen Methode für verschiedene Wahlen der Bereiche B_i abschätzen. Dazu gehen wir von einem kubischen Bild mit einer Seitenlänge von n Voxeln und einem kubischen Integrationsgebiet mit einer Seitenlänge von s Voxeln aus und zählen die arithmetischen Operationen, die zur Berechnung des Trägheitstensors in allen Bildpunkten nötig sind.
In jedem besuchten Punkt eines Bereiches B_i sind für die Berechnung der Masse eine Addition, für jede der drei Komponenten des Schwerpunktes eine Multiplikation und eine Addition und für jede der sechs zu berechnenden Tensoreinträge zwei Multiplikationen und eine Addition nötig. Das sind insgesamt 25 Operationen. Bei der Verteilung der 10 Ergebnisse von einem B_i auf die sie benötigenden Punkte sind pro Punkt weitere 10 Additionen notwendig.
Wir betrachten vier Fälle:

1. *Zerlegung der Umgebungen $V(x_j)$ in genau einen Bereich.*

   Dies entspricht dem aus Abschnitt 4.4.4 bekannten naiven Verfahren. Pro Voxel ist die gesamte Umgebung $V(x_j)$ abzuarbeiten. Da die Verteilung entfällt, erhalten wir als Aufwand

   $$A_{\text{naiv}} = n^3 \cdot s^3 \cdot 25$$

2. *Zerlegung der Umgebungen $V(x_j)$ in die einzelnen Voxel.*

   Jeder Voxel entspricht einem eigenen Bereich und muss an s^3 andere Punkte verteilt werden. Diese kleinstmögliche Unterteilung impliziert einen Aufwand von

   $$A_{\text{voxel}} = n^3 \cdot (25 + s^3 \cdot 10)$$

   Obwohl die einzelnen B_i keinerlei Überlapp mehr aufweisen, haben wir kaum etwas gewonnen. Eine Minimierung des Überlapps und der damit verbundenen Besuche jedes Knotens ist also nicht mit einer Minimierung des Gesamtaufwandes gleichbedeutend.

3. *Zerlegung der Umgebungen $V(x_j)$ in Scheiben.*

   Das sind die oben beschriebenen Scheiben. In jedem Bildpunkt müssen zum Aufbau der Scheibe s^2 Punkte ausgewertet werden, das Ergebnis an s Punkte verteilt werden. Wir erhalten

   $$A_{\text{scheiben}} = n^3 \cdot (s^2 \cdot 25 + s \cdot 10)$$

   Dies stellt eine spürbare Beschleunigung im Vergleich zum naiven Verfahren dar. Vor allem aber hat der Aufwand eine geringere Komplexität als der des naiven Verfahrens.

4. *Zerlegung der Umgebungen $V(x_j)$ in Stäbe.*

   Es wird eine Zerlegung in analog zu den Scheiben definierte, eindimensionale Stäbe verwendet. Zum Aufbau eines Stabes müssen s Punkte ausgewertet werden, das Ergebnis pro Stab an s^2 Punkte verteilt werden. Wir bekommen entsprechend

$$A_{\text{stäbe}} = n^3 \cdot (s \cdot 25 + s^2 \cdot 10)$$

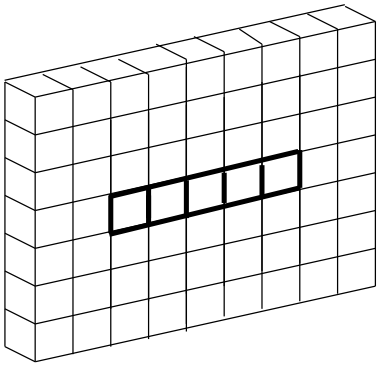

Abbildung 4.18: Stab von Punkten

Die durch die Verwendung der oben beschriebenen Scheiben erzielten Beschleunigungen und die tatsächlich gemessenen Zeiten in Sekunden sind für einige Werte von s in der Tabelle 4.2 zusammengestellt. Es wurde auf einem Bild mit der Seitenlänge $n = 65$ gerechnet.

s	3	5	7	9	11	13
$(25s^3)/(25s^2 + 10s)$	2.65	4.63	6.62	8.62	10.61	12.61
T_{naiv}	6	15	36	75	125	199
T_{scheiben}	3	4	6	10	14	18

Tabelle 4.2: Zur Berechnung des Trägheitstensors auf dem ganzen Bild mit Seitenlänge $n = 65$ gemessene Zeiten in Sekunden für das naive und das beschleunigte Verfahren

Verschiebung der Scheiben

Aus den Aufwandsabschätzungen ist erkennbar, dass die Verwendung von Stäben offenbar noch effizienter wäre, als die Verwendung von Scheiben. Es ist

$$A_{\text{scheiben}} \approx 2 \cdot A_{\text{stäbe}}$$

Der Algorithmus kann aber auch ohne Verwendung von Stäben noch effizienter gestaltet werden.

Der Aufbau der Scheibe in jedem einzelnen Punkt ist nämlich unnötig. Die Scheiben der auf einer z-Ebene benachbarten Punkte haben bis auf die Ränder alle Bildpunkte gemeinsam. So bietet es sich an, das Ergebnis der Scheibe in einem Punkt durch Verschiebung der Scheibe des Nachbarpunktes zu berechnen, und zwar indem man die Beiträge der Bildpunkte auf dem in Verschiebungsrichtung hinten gelegenen Rand von dem Ergebnis dieser Nachbarscheibe subtrahiert und die Beiträge der Bildpunkte, die in Verschiebungsrichtung vor der Nachbarscheibe liegen, addiert.

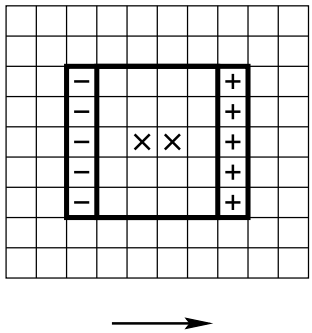

Abbildung 4.19: Verschiebung einer Scheibe von Punkten auf einer Ebene

So muss eine komplette Scheibe nur noch einmal pro z-Ebene aufgebaut werden, in allen anderen Punkten der Ebene wird sie durch sukzessive Verschiebung in x- und y-Richtung gewonnen. Der daraus resultierende Algorithmus ist schematisch in Abbildung 4.21 dargestellt.

Bei der Aufwandsabschätzung gehen wir wieder von einem kubischen Bild mit einer Seitenlänge von n Voxeln sowie einem kubischen Integrationsgebiet mit einer Seitenlänge von s Voxeln aus.
Die Scheibe ist nur noch einmal pro Ebene aufzubauen, also n-mal. In allen übrigen Punkten, das sind demnach $n^3 - n$ Stück, wird sie durch Verschiebung berechnet. Bei einer Verschiebung sind $2s$ Knoten auszuwerten. Das Verteilen der Ergebnisse hat denselben Aufwand wie voher. Daraus resultiert:

$$A_{\text{verschobene\_scheiben}} = (n^3 - n) \cdot (2s \cdot 25 + s \cdot 10) + n \cdot (s^2 \cdot 25 + s \cdot 10)$$

Durch Umformung erhalten wir:

$$A_{\text{verschobene\_scheiben}} = n^3(60s + \frac{1}{n^2}(25s^2 - 50s))$$

Damit können wir den Aufwand mit dem des vorherigen Verfahrens, dem ohne die Verschiebung der Scheiben, vergleichen.
In der Tabelle 4.3 sind die errechneten Beschleunigungen mit den tatsächlich gemessenen Zeiten in Sekunden dargestellt. Bei der Beschleunigungsberechnung wurde der Term $\frac{1}{n^2}(25s^2 - 50s)$ wegen des großen n vernachlässigt.

s		2	4	6	8	10	12	14
$(25s^2 + 10s)/60s$		1.00	1.83	2.67	3.50	4.33	5.17	6.00
n=128	$T_{nicht\_verschoben}$	21	40	67	102	146	192	245
	$T_{verschoben}$	16	22	26	31	37	42	47
n=129	$T_{nicht\_verschoben}$	20	32	51	76	110	170	245
	$T_{verschoben}$	18	36	49	62	75	87	99

Tabelle 4.3: Zur Berechnung des Trägheitstensors in allen Elementen auf einem kubischen Bild mit Seitenlänge n gemessene Zeiten in Sekunden für das beschleunigte Verfahren mit und ohne Verschiebung der Scheiben

Für den Fall $n = 128$ stimmen die errechneten Beschleunigungen recht gut mit den gemessenen überein. Im Fall $n = 129$ dagegen nicht, da beim Übergang von $n = 128$ zu $n = 129$ sowohl das Verfahren ohne Verschiebung der Scheiben schneller, als auch das Verfahren mit Verschiebung der Scheiben deutlich langsamer wird. Vermutlich wirken sich hier speichertechnische Eigenschaften wie die Cache-Verwaltung auf die Laufzeit aus.

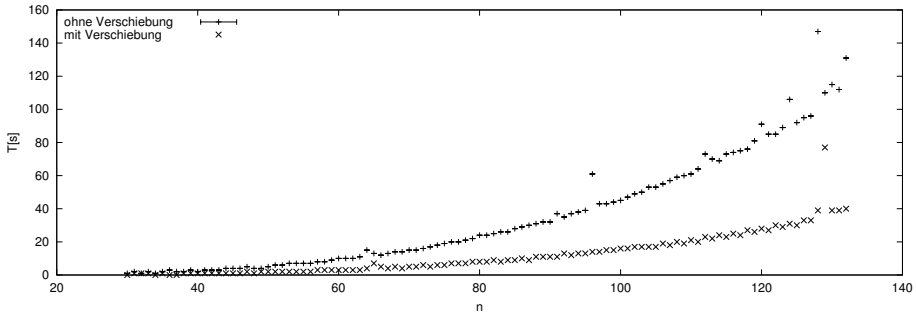

Abbildung 4.20: Zur Trägheitstensorberechnung auf dem ganzen Bild benötigte Zeit in Sekunden für ein kubisches Bild der Größe n^3 für die beschleunigten Verfahren mit und ohne Veschiebung der Scheiben unter Verwendung eines kubischen Integrationsgebietes der Seitenlänge $s = 10$

Um dies zu illustrieren, sind in der Abbildung 4.20 die Laufzeiten der beiden Verfahren für verschiedene Bildgrößen aufgetragen. Dabei wurde ein kubisches Integrationsgebiet mit Seitenlänge $s = 10$ verwendet.
Auffallend sind für das Verfahren ohne Verschiebung der Scheiben die Ausreißer nach oben bei 64 und 128, welches beides Potenzen von 2 sind. Das Verfahren mit Verschiebung der Scheiben hat Ausreißer bei 65 und 129, also Seitenlängen der Form $2^n + 1$.
Dieses Verhalten bietet einen Hinweis darauf, dass tatsächlich die Cache-Verwaltung die Laufzeiten mit beeinflusst.

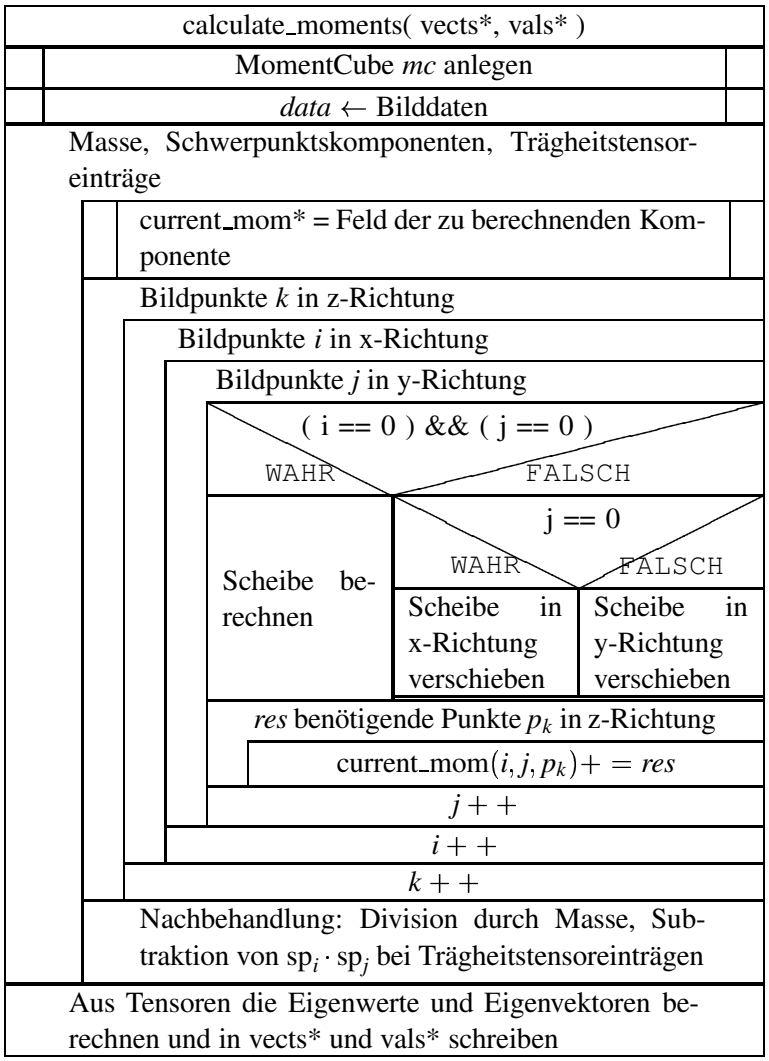

Abbildung 4.21: Algorithmus zur Berechnung der Trägheitsmomente durch Zusammenfassung von Punkten zu Scheiben und Verschiebung der Scheiben

Verschobene Stäbe
Wir betrachten nun noch den Aufwand, den wir bei Verwendung von Stäben statt der Scheiben erhalten würden, wenn wir diese analog nicht in jedem Punkt neu aufbauen, sondern durch Verschiebung berechnen würden.
Der Neuaufbau wäre dann nur in n^2 Punkten - also für eine Bildebene - nötig. Die Verschiebung eines Stabes würde die Auswertung von nur 2 Knoten kosten. Es resultiert

$$\begin{aligned} A_{\text{verschobene\_Stäbe}} &= (n^3 - n^2) \cdot (2 \cdot 25 + 10 \cdot s^2) + n^2 \cdot (s \cdot 25 + 10 \cdot s^2) \\ &= n^3(50 + 10s^2 + \frac{1}{n}(25s - 50)) \end{aligned}$$

Vergleichen wir das mit dem Aufwand ohne Verschiebung, erhalten wir Beschleunigungen von etwa 1.2. Das bedeutet einerseits, dass die Verschiebung der Stäbe gegenüber dem Neuaufbau in jedem Punkt fast keine Beschleunigung bringt. Zum anderen ist damit die Verwendung von Scheiben mit Verschiebung effizienter als die Verwendung von Stäben mit Verschiebung. Das erklären wir so: Die Verwendung von Stäben gegenüber den Scheiben (beide ohne Verschiebung) bringt uns den Faktor 2 an Beschleunigung. Multiplizieren wir diesen mit den 1.2, die uns die Verschiebung der Stäbe noch liefert, erhalten wir insgesamt einen Faktor von 2.4. Die Verschiebung der Scheiben allein bringt aber, wie es aus Tabelle 4.3 ersichtlich ist, weitaus größere Beschleunigungen (wenn man s nicht zu klein wählt).

Fazit
Die Verwendung von Scheiben, die durch Verschiebung bereits bekannter Scheiben berechnet werden, erweist sich als effizienteste Variante der Zusammenfassung von Punkten.

Einschränkung der Anwendbarkeit
An dieser Stelle müssen wir bemerken, dass die hier beschriebene Form der Beschleunigung durch Zusammenfassung von Punkten nur funktioniert, weil die Berechnung auf den Scheiben unabhängig davon verläuft, welchem Knoten das Ergebnis zugeteilt wird. Im Fall der Einträge des Trägheitstensors ist damit nur die Berechnung durch die modifizierte Formel (4.5) möglich, nicht durch die ursprüngliche Formel (4.4), da hier die Integration vom Schwerpunkt und damit vom Knoten, dem das Ergebnis zugeteilt werden soll, abhängig ist.
Nun wird aber auch eine Einschränkung der Anwendbarkeit der Methode deutlich. Als Integrationsgebiet kann man keine Kugel angeben, da die Gewichte, mit denen die Voxel dabei muliplizierte werden, natürlich von dem Knoten, für den die Integration ausgeführt werden soll, abhängen und damit eine unabhängige Integration auf den Scheiben unmöglich ist.

4.4.7 Beschleunigung durch schnelle Fourier-Transformation

Die bei der Berechnung der Trägheitstensoren auftretenden Integrationsformeln entsprechen genau genommen einer Faltung der Bildfunktion mit verschiedenen Gewichtsfunktionen.
Eine Faltung zweier kontinuierlicher Funktionen f und g ist definiert als

$$(f * g)(x) := \int_{\mathbb{R}^n} f(y) \cdot g(x-y)dy$$

Die analog definierte Faltung zweier diskret gegebener Funktionen ist

$$(f * g)(k) := \sum_{j=0}^{n-1} f(j) \cdot g(k-j)$$

wobei n jeweils die Anzahl der Funktionswerte von f und g ist. Dabei rechnen wir in den Argumenten immer modulo n, d.h. die Funktionen f und g sind periodisch fortgesetzt.
Bei der Berechnung der Trägheitstensoren haben wir Summen der Form

$$I_V(x) = \sum_{p \in V(x)} u(p) \cdot k(p) \cdot w(p) \tag{4.7}$$

auszuwerten, wobei $k(p)$ die prinzipiellen drei Möglichkeiten $k(p) = 1$ für die Berechnung der Masse, $k(p) = p_i$ für die Berechung der i-ten Komponente des Schwerpunktes und $k(p) = p_i \cdot p_j$ für die Berechnung des Trägheitstensoreintrages T_{ij} annehmen kann. Die Funktion $w(p)$ gibt die Gewichtung an, mit der die einzelnen Voxel bei Verwendung eines kugelförmigen Integrationsgebietes noch mulitipliziert werden müssen (vergl. Abschnitt 4.4.1). Für den Fall eines kubusförmigen Integrationsgebietes ist $w(p) \equiv 1$.
Nun setzen wir $k(p) = w(p) = 0$ für $p \notin V(x)$. Damit können wir die Summe auch über das ganze Bild Ω laufen lassen und erhalten

$$I_V(x) = \sum_{p \in \Omega} u(p) \cdot k(p) \cdot w(p) \tag{4.8}$$

Die Funktionen $k(p)$ und $w(p)$ fassen wir zu einer Funktion $\widetilde{kw}(p) := k(p) \cdot w(p)$ zusammen. Dann führen wir eine Verschiebung von dessen Funktionswerten durch, indem wir $kw(p) := \widetilde{kw}(x-p)$ setzen. Dann ist nämlich (4.8) als diskrete Faltung

$$I_V(x) = (u * kw)(x) = \sum_{p \in \Omega} u(p) \cdot kw(x-p) \tag{4.9}$$

darstellbar. Dabei sei daran erinnert, dass die Funktionen u und kw über den Rand von Ω hinaus periodisch fortgesetzt sein müssen.

Zu bemerken ist, dass die Funktion *kw* - obwohl es zuerst nicht so aussieht - unabhängig von x, also der Stelle, für die der Trägheitstensor berechnet werden soll, ist.
Betrachten wir dazu als Beispiel die Funktionswerte von *k* bei einem eindimensionalen Integrationsgebiet $V(x)$ der Länge $2n+1$ für die Berechnung der Masse. Es ist dann

$$k(p) = \begin{cases} 1 & p \in [x-n, x+n] = V(x) \\ 0 & p \notin [x-n, x+n] = V(x) \end{cases}$$

Die Funktion *k* ist also von $V(x)$ und damit von x abhängig.
Dagegen ist mit $w(p) = 1$ für $p \in V(x)$ (das ist die Integration auf einem kubusförmigen Gebiet)

$$kw(p) = \begin{cases} 1 & p \in [-n, n] \\ 0 & p \notin [-n, n] \end{cases}$$

von x unabhängig.
Das ist auch bei Verwendung eines kugelförmigen Integrationsgebietes für alle der obigen Varianten von *kw* so.
Dieser Aspekt wird im Folgenden eine entscheidende Rolle spielen. Es sei darauf hingewiesen, dass für den Fall der Einträge des Trägheitstensors die Unabhängigkeit von *kw* von x nur durch die Verwendung der modifizierten Formel (4.5) gegeben ist.

Wie wir gesehen haben, ist es sehr ineffizient, die Faltung (4.9) in jedem Bildpunkt neu zu berechnen. Es gibt jedoch ein bekanntes Hilfsmittel, die Fourier-Transformation, mit deren Hilfe die Faltung zweier Funktionen sehr viel effizienter ausgeführt werden kann.

Die eindimensionale Fourier-Transformation einer diskret gegebenen skalaren Funktion f mit n Funktionswerten ist definiert durch

$$\widehat{f_k} = \sum_{j=0}^{n-1} f_j \cdot e^{2\pi ijk/n}, \quad k = 0, \ldots, n-1 \tag{4.10}$$

Sie ist formal eine Polynominterpolation

$$p(x_k) = \sum_{j=0}^{n-1} f_j \cdot x_k^j, \quad x_k = e^{2\pi ik/n}, \quad k = 0, \ldots, n-1$$

bei der die Stützstellen in regelmäßigen Abständen auf dem komplexen Einheitskreis liegen.
Mit Hilfe dieser Transformation erhalten wir den *diskreten Faltungssatz*, den wir in Anlehnung an die Version für kontinuierliche Funktionen in Forster[3] für den eindimensionalen Fall formulieren und beweisen.

Satz. *Für zwei diskrete skalare Funktionen f und g mit jeweils n Funktionswerten gilt*

$$\widehat{(f * g)}_k = \widehat{f}_k \cdot \widehat{g}_k \tag{4.11}$$

Beweis:

$$\begin{aligned}
\widehat{(f * g)}_k &= \sum_{j=0}^{n-1} (f * g)_j \cdot e^{2\pi ijk/n} \\
&= \sum_{j=0}^{n-1} \sum_{l=0}^{n-1} f_l \cdot g_{j-l} \cdot e^{2\pi ijk/n} \\
&= \sum_{j=0}^{n-1} \sum_{l=0}^{n-1} f_l \cdot g_{j-l} \cdot e^{2\pi ijk/n} \cdot e^{2\pi i(l-l)k/n} \\
&= \sum_{j=0}^{n-1} \sum_{l=0}^{n-1} f_l \cdot g_{j-l} \cdot e^{2\pi i(j-l)k/n} \cdot e^{2\pi ilk/n} \\
&= \sum_{l=0}^{n-1} f_l e^{2\pi ilk/n} \sum_{j=0}^{n-1} g_{j-l} e^{2\pi i(j-l)k/n} \\
&= \sum_{l=0}^{n-1} f_l e^{2\pi ilk/n} \cdot \widehat{g}_k \\
&= \widehat{f}_k \cdot \widehat{g}_k
\end{aligned}$$

□

Mit Hilfe dieses Satzes können wir die Faltungsfunktion $I_V(x)$ berechnen, indem wir die Fourier-Transformierte der Funktionen *u* und *kw* bilden, die transformierten Funktionen $\widehat{u}$ und $\widehat{kw}$ punktweise miteinander multiplizieren und die Produktfunktion mittels der inversen Fourier-Transformation

$$f_k = \frac{1}{n} \sum_{j=0}^{n-1} \widehat{f}_j \cdot e^{-2\pi ijk/n}, \quad k = 0, \ldots, n-1$$

zurücktransformieren. Das Resultat ist genau die Funktion $I_V(x)$.

Hier zeigt sich, wie wichtig es ist, dass *kw* von *x* unabhängig ist. Wäre dies nicht der Fall, müsste man die Fourier-Transformation von *kw* (und die inverse Transformation) für jede Stelle *x* neu durchführen, eine offensichtlich völlig ineffiziente Variante.

Zwar haben wir den Faltungssatz nur für den eindimensionalen Fall bewiesen, jedoch gilt er genauso für höhere Dimensionen. Die Fourier-Transformation für zwei

und mehr Dimensionen ist entsprechend der mehrdimensionalen Polynominterpolation definiert. Aufgrund der Formel

$$p(x_1, x_2) = \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} \alpha_{j_1,j_2} \cdot x_1^{j_1} \cdot x_2^{j_2}$$

für die zweidimensionale Polynominterpolation lautet beispielsweise die zweidimensionale Fouriertransformation

$$\widehat{f}_{k_1,k_2} = \sum_{j_1=0}^{n_1-1} \sum_{j_2=0}^{n_2-1} f_{j_1,j_2} \cdot e^{2\pi i j_1 k_1/n_1} \cdot e^{2\pi i j_2 k_2/n_2}, \quad k_i = 0, \ldots, n_i - 1$$

Schreibt man diese als

$$\widehat{f}_{k_1,k_2} = \sum_{j_1=0}^{n_1-1} e^{2\pi i j_1 k_1/n_1} \sum_{j_2=0}^{n_2-1} f_{j_1,j_2} \cdot e^{2\pi i j_2 k_2/n_2}, \quad k_i = 0, \ldots, n_i - 1$$

so sieht man, dass sie eine Schachtelung zweier Fouriertransformationen, die auf jeweils einem Index ausgeführt werden, darstellt. Dies lässt sich auch auf n Dimensionen verallgemeinern.

Die direkte Auswertung der Formel (4.10) benötigt n^2 komplexe Multiplikationen. Es gibt jedoch auch Algorithmen, die sogenannten *schnellen Fouriertransformationen* (FFT), welche sehr viel effizienter arbeiten und mit $\mathcal{O}(n \log_2 n)$ Operationen auskommen.
Wir stellen dazu den folgenden Algorithmus von Cooley und Tukey vor, wie er in Natterer[10] beschrieben wird. Die Anzahl n der zu berechnenden Funktionswerte bezeichnen wir dabei auch als Länge der Transformation.

Zur Berechnung einer eindimensionalen Fourier-Transformation der Länge $n = 2m$ zerlegt man

$$\widehat{f}_k = \sum_{j=0}^{n-1} f_j \cdot q^{jk}, \; q = e^{2\pi i/n}, \quad k = 0, \ldots, n-1$$

in gerade und ungerade Indizees:

$$\begin{aligned} \widehat{f}_k &= \sum_{l=0}^{m-1} f_{2l} \cdot q^{(2l)k} + \sum_{l=0}^{m-1} f_{2l+1} \cdot q^{(2l+1)k} \\ &= \underbrace{\sum_{l=0}^{m-1} f_{2l} \cdot (q^2)^{lk}}_{=:g_k} + q^k \underbrace{\sum_{l=0}^{m-1} f_{2l+1} \cdot (q^2)^{lk}}_{=:u_k} \end{aligned}$$

Da $q^2 = e^{2\pi i/(n/2)}$ ist, sind dann g_k und u_k jeweils Fourier-Transformationen der Länge $n/2$.
Bezeichnen wir weiter mit A_p die Anzahl der komplexen Additionen und mit M_p die Anzahl der komplexen Multiplikationen, die zur Durchführung einer schnellen Fourier-Transformation der Länge 2^p nötig sind, so erhalten wir

$$M_{p+1} = A_{p+1} = 2A_p + 2^{p+1}$$

Daraus verifiziert man leicht mittels vollständiger Induktion[5]

$$M_p = A_p = p2^p = (\log_2 n) \cdot n$$

Damit ist die eindimensionale Fourier-Transformation der Länge 2^p in $\mathcal{O}(n \log_2 n)$ durchführbar.

Gegenüber der naiven Faltung, bei der die Summe (4.9) für jede Stelle x neu berechnet wird, und welche im eindimensionalen Fall eine Komplexität von $\mathcal{O}(n^2)$ aufweist, ist das eine enorme Beschleunigung. Da wir jedoch bei der Benutzung des naiven Verfahrens die Integration jeweils nur für ein Teilgebiet mit Länge $s < n$ ausführen, haben wir dafür nur eine Komplexität von $\mathcal{O}(n \cdot s)$. Daher wird sich die schnelle Fourier-Transformation vor allem bei der Benutzung großer Integrationsgebiete lohnen, da ihr Aufwand von der Größe s des Integrationsgebietes unabhängig ist. Bei großem n und kleinem s kann das naive Verfahren dagegen sogar schneller sein.

Für die praktische Implementierung der dreidimensionalen FFT gebrauchen wir einen Code der Numerical Recipes[11]. Unter dessen Benutzung benötigen wir für die Berechnung von (4.9) auf einem kubischen Bild der Kantenlänge $n = 65$ genau 10 Sekunden. Im Vergleich mit den Zeiten in Tabelle 4.2 des voherigen Abschnitts, welche wir für das naive Verfahren gemessen haben, stellen wir fest, dass sich das Hilfsmittel der schnellen Fouier-Transformation bereits sehr früh, bei uns ab $s = 5$, lohnt[6].
Im Vergleich mit dem beschleunigten Verfahren des vorherigen Abschnitts schneidet die schnelle Fourier-Transformation dagegen nicht gut ab (siehe Tabelle 4.3). Für den Fall $n = 128$ benötigt sie bei uns 118 Sekunden. Das durch Zusammenfassung von Punkten beschleunigte Verfahren dagegen bleibt selbst für $s = 14$ mit 47 Sekunden noch weit darunter. Wir werden in Kapitel 5 sehen, dass es nicht günstig ist, $s > 14$ zu setzen. Was nützt uns also die FFT?
Die Beschleunigung durch Zusammenfassung von Punkten hat den Nachteil, nur für kubusförmige, nicht dagegen für kugelförmige Integrationsgebiete verwendbar zu sein. Die FFT funktioniert dagegen für beide Varianten. Da sie das naive

[5] Wegen $\widehat{f}_{k+m} = g_{k+m} + q^{k+m}u_{k+m} = g_k + q^k q^{n/2}u_k = g_k - q^k u_k$ kann man aber auch mit $M_{p+1} = 2M_p + 2^p$ und damit $M_p = \frac{1}{2}p2^p$ Multiplikationen auskommen.

[6] Mit s bezeichnen wir wieder die Kantenlänge des kubischen Integrationsgebietes.

Verfahren gerade für große Werte von s massiv beschleunigt, sollten wir sie benutzen, wenn wir die Strukturerkennung auf kugelförmigen Integrationsgebieten durchführen wollen.

Eine grundsätzliche Einschränkung bekommen wir aber auch bei der Verwendung der FFT. Der Algorithmus der Numerical Recipes funktioniert nur, wenn die einzelnen Kantenlängen des Bildes eine Zweierpotenz sind.

4.5 Die Matrix-Datenstruktur

Die Klassen in diesem Abschnitt wurden von Christoph Reisinger geschrieben.

4.5.1 Die Klasse SPARSEMATRIX

Die Klasse SPARSEMATRIX bietet Datenstrukturen und Methoden zur Verwaltung einer (dünn besetzten) Matrix. Wir haben in Abschnitt 3.5.1 gesehen, dass die Speicherung der durch die Diskretisierung entstehenden Matrizen erstens wegen des Speicheraufwandes nicht realisierbar, zweitens wegen der überwiegend aus Nullen bestehenden Einträge auch nicht sinnvoll ist. In der SPARSEMATRIX-Struktur werden daher nur die von Null verschiedenen Einträge gespeichert.

Die Matrix besteht aus zwei geschachtelten Strukturen. Die Klasse SPARSEMATRIX ist von MULTIARRAY abgeleitet; jeder Eintrag in MULTIARRAY repräsentiert eine Zeile der Matrix.
Die einzelnen Zeilen werden durch die Struktur LINE gebildet. Diese wird im folgenden Abschnitt vorgestellt.

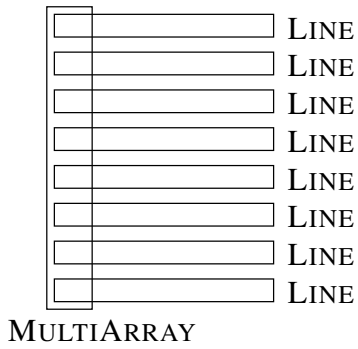

Abbildung 4.22: Aufbau einer SPARSEMATRIX

Die Datenstruktur des MULTIARRAY bietet sich zur Matrixrepräsentation vor allem wegen der Ähnlichkeit zum DATACUBE an. Aus der Diskretisierung geht hervor, dass jede Matrixzeile zu einem Punkt des DATACUBE gehört. Durch die Verwendung des MULTIARRAY funktioniert die Verwaltung der Zeilen in der Matrix

exakt wie die der Punkte im DATACUBE. Einziger Unterschied ist, dass die Nummerierung der Zeilen der Matrix entsprechend der in MULTIARRAY mit 1, die der Punkte im DATACUBE dagegen mit 0 beginnt. Zu einem Punkt n des DATACUBE gehört also die Zeile $n + 1$ der Matrix.
Diese Form der Datenstruktur ist durch ihre MULTIARRAY-Struktur speziell auf den Gebrauch für strukturierte quaderförmige Gitter zugeschnitten. Für unstrukturierte Gitter wäre sie nicht brauchbar.

SPARSEMATRIX hat eine große Funktionalität, die vor allem der Löser benutzt. Für die Diskretisierung benutzen wir bereitgestellte Methoden

- zum Addieren eines Wertes zum Eintrag in der i-ten Zeile und j-ten Spalte
- zum Addieren der Einheitsmatrix
- zum Multiplizieren der Matrix mit einem skalaren Faktor
- zum Multiplizieren einer Zeile i der Matrix mit einem skalaren Faktor
- zum Setzen aller Matrixeinträge auf Null

4.5.2 Die Struktur LINE

Die Struktur LINE verwaltet eine Zeile einer dünn besetzen Matrix. Zur internen Speicherung der Einträge verwendet sie ein Objekt der Klasse ARRAY.
Während die zu einem Eintrag a_{ij} gehörende Matrixzeile i durch die Position der LINE in MULTIARRAY klar ist, muss die zugehörige Spaltennummer j, welche zum Beispiel bei Matrix-Vektor Multiplikationen benötigt wird, zu jedem Eintrag zusätzlich gespeichert werden. Dies geschieht mittels eines zweiten ARRAY-Objekts. Beim Addieren eines Wertes zu einem Eintrag a_{ij} in der LINE wird dieses nach der angegebenen Spaltennummer j durchsucht, um den gewünschten Eintrag zu finden.
Der Diagonaleintrag einer Matrixzeile steht immer an der ersten Position der LINE[7], weitere Festlegungen der Reihenfolge der Einträge sind nicht vorhanden.

t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	⟵ Einträge
10	6	9	5	11	4	14	15	16	⟵ Spaltennummern

Abbildung 4.23: Beispiel einer LINE

[7] Diese Information wird vom Löser benötigt

Alternative Datenverwaltung

Als Alternative zu der hier verwendeten Speicherung der Zeileneinträge, bei der außer beim Diagonaleintrag keine feste Zuordnung zu den Stellen des ARRAY besteht, kann man auch genau eine solche Zuordung vornehmen. Das ist möglich, da die Anzahl der Einträge pro Zeile im Voraus bekannt ist, zum Beispiel sind es bei unserem Finite-Volumen-Stern 27 Stück. Die Zuweisung der Speicherplätze kann entsprechend der Stern-Struktur der Zeileneinträge erfolgen. Für einen Eintrag a_{ij} bestimmt man dazu die relativen Koordinaten des elementlokalen Knotens j zum elementlokalen Knoten i, was in unserem Fall durch

$$
\begin{aligned}
rel_x &= (j\%2) - (i\%2) \\
rel_y &= (j\%4)/2 - (i\%4)/2 \\
rel_z &= j/4 - i/4
\end{aligned}
$$

geschieht, wobei mit ganzzahliger Division ohne Rest gerechnet wird. Die Wirkung der obigen Operationen auf die elementlokalen Knotennummern wird in Abbildung 4.24 gezeigt. Die relativen Koordinaten können wegen der Benutzung der elementlokalen Knotennummern nur die Werte -1, 0 oder 1 annehmen. Es genügt, die elementlokalen Knotennummern zu benutzen, da ohnehin nur Knoten, die gemeinsam einem Element angehören, miteinander koppeln.

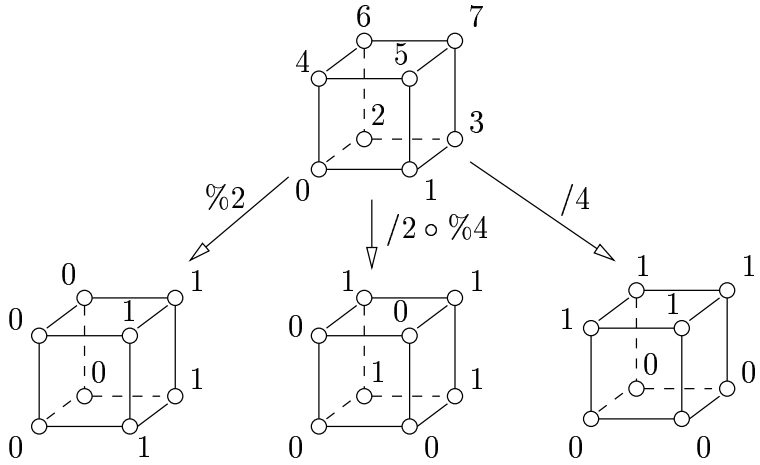

Abbildung 4.24: Berechnung der relativen Koordinaten elementlokaler Knoten

Mittels der relativen Koordinaten kann man die Speicherplatzzuweisung in unserem Fall durch

$$I(rel_x, rel_y, rel_z) = 14 + rel_x + 3rel_y + 9rel_z$$

vornehmen[8], wodurch der Stern lexikographisch auf das ARRAY abgebildet wird. Dieses Vorgehen ist auf andere Dimensionen d verallgemeinerbar. So lautet die Funktion zur Berechnung des Index für höhere Dimensionen

$$I(\overrightarrow{rel}) = \frac{\#\text{Einträge}}{2} + 1 + \sum_{k=0}^{d-1} 3^k rel_k$$

Diese Art der Speicherung hat den Vorteil, das zusätzliche Abspeichen der Spaltennummern unnötig werden zu lassen. Das Durchsuchen des ARRAY nach dem richtigen Eintrag fällt durch die feste Zuweisung weg. Die Spaltennummer eines Eintrags kann aus seiner Zeilennummer und der Kenntnis der Anzahl der Gitterknoten in die einzelnen Raumdimensionen über die relativen Koordinaten berechnet werden. In unserem 3D-Fall durch

$$j = i + rel_x + rel_y \cdot size_x + rel_z \cdot size_x \cdot size_y$$

4.6 Diskretisierung

4.6.1 Die Klasse FV_3D27

Die Klasse FV_3D27 stellt Methoden zur dreidimensionalen zeitabhängigen Finite-Volumen Diskretisierung des Anfang- Randwertproblems

$$\begin{aligned} &\partial_t u = \nabla \cdot (D(u) \cdot \nabla u) && \text{auf } \mathbb{R}^+ \times \Omega \\ &u(x,0) = u_0(x) && \text{auf } \bar{\Omega} \\ &(D(u) \cdot \nabla u) \cdot \vec{n} = 0 && \text{auf } \mathbb{R}^+ \times \partial\Omega \end{aligned} \tag{4.12}$$

zur Beschreibung nichtlinearer anisotroper Diffusion auf einem aus Hexaedern mit Kantenlänge 1 bestehenden Gitter zur Verfügung. Der Diffusionstensor wird dabei durch Trägheitsmomente wie in Abschnitt 3.3.3 beschrieben festgelegt.

[8] Dabei ist wieder berücksichtigt, daß die Nummerierung der Speicherplätze im ARRAY bei 1 beginnt. Beginnt man von 0 an zu nummerieren, muß die Funktion

$$I(rel_x, rel_y, rel_z) = 13 + rel_x + 3rel_y + 9rel_z$$

lauten.

Die Methoden realisieren:

- Raumdiskretisierung durch Assemblierung der Matrix A nach der Formel

$$a_{ij} = \frac{1}{|V_i|} \sum_k \left(D(u)\nabla\varphi_j(\mathrm{ip}_k)\right) \cdot \vec{n}_k \cdot |S_k|$$

- Zeitdiskretisierung durch Setzen von

$$A = \mathbb{I} - \tau A$$

Für die Matrixassemblierung steht als Attribut ein Zeiger auf ein Objekt der Klasse SPARSEMATRIX bereit. Eine Reihe weiterer Attribute, die die stationäre Diskretisierung parametrisieren, können über ebenfalls bereitstehende Methoden gesetzt werden. Diese Attribute sind im einzelnen:

integration_size_x, integration_size_y, integration_size_z: Legen die Größe des Volumens, auf dem die Trägheitsmomente berechnet werden, in die einzelnen Raumrichtungen fest. Dieses Volumen wollen wir auch *Integrationsgebiet* nennen.

geometry_type: Legt die Geometrie des Integrationsgebietes fest. Wählbar sind wie in Abschnitt 4.4.1 beschrieben die Möglichkeiten CUBE, bei der die Trägheitsmomente durch Zusammenfassung von Punkten berechnet werden, und BALL, bei der die Beschleunigung durch FFT Verwendung findet.

ip_flag: Gibt die Lage der Integrationspunkte auf den Oberflächen der subcontrol volums an. Wählbar sind die gewöhnlichen Integrationspunkte durch Setzen auf IP_USUAL und die verschobenen Integrationspunkte (vergleiche Abschnitt 3.4.2) durch Setzen auf IP_BND.

fixed_coeffs: Bestimmt die Wahl der Anisotropiekoeffizienten t_1, t_2, t_3 nach der Formel (3.22). Falls YES gesetzt ist, werden diese auf die Werte der Attribute anicoeff1, anikoeff2, anikoeff3 gesetzt. Sonst werden sie nach den Formeln (3.23) bestimmt.

anicoeff1, anicoeff2, anicoeff3: Dies sind die fixen Werte für die Anisotropiekoeffizienten, falls fixed_coeffs auf YES gesetzt ist; anderenfalls haben diese Attribute keine Auswirkung auf die Diskretisierung.

DependenceType: Gibt die Funktion $g(.)$ in (3.23) an, falls fixed_coeffs auf NO gesetzt ist, anderenfalls hat dieses Attribut keine Auswirkung. Wählbar sind PERONA_MALIK, WEIKERT und BLACK_SAPIRO, welche nach den Formeln in Abschnitt 3.2.6 implementiert sind.

lambda: Parameter der Funktion $g(.)$ in (3.23), Hat nur dann eine Auswirkung, falls fixed_diffusion auf NO gesetzt ist.

4.6.2 Lokalisierung der Trägheitsmomentenberechnung

Bei der Assemblierung der Diskretisierungsmatrix wird der Term $D(u)\nabla\varphi_j(\mathrm{ip}_k)$ ausgewertet. Der Diffusionstensor wirkt also auf den Gradient der Ansatzfunktionen in den Integrationspunkten. Daher müssen auch die Trägheitsmomente, nach denen der Diffusionstensor $D(u)$ ja gebildet wird, in den Integrationspunkten berechnet werden. Dazu haben wir verschiedene Möglichkeiten.

1. Berechnung der Trägheitsmomente direkt in jedem Integrationspunkt

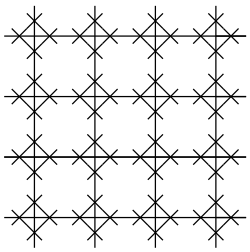

Abbildung 4.25: Duales Gitter aus Kontrollvolumina mit Integrationspunkten

Dies ist der eigentlich gewünschte Ansatz. Benutzt man diesen, gerät man aber in eine Zwickmühle.
Entweder ist nämlich der Speicheraufwand hoch, da pro Element Trägheitsmomente zu zwölf Integrationspunkten gespeichert werden müssen, oder aber man nimmt, falls man dem entgegenhält, dass die Speicherung nicht nötig ist, wenn man die Trägheitsmomente in dem Augenblick berechnet, da man sie benötigt, eine hohe Ineffizienz des Algorithmus in Kauf. Die Beschleunigung durch Benutzung der FFT arbeitet nämlich global, d.h. berechnet alle im DATACUBE benötigten Trägheitsmomente auf einmal und kann daher keine Verwendung finden, wenn man diese nicht komplett speichern will. Die Methode des Zusammenfassens von Punkten funktioniert sogar überhaupt nicht. Denn selbst wenn man ein kubusförmiges Integrationsgebiet benutzt, liegen Voxel nur teilweise im Integrationsgebiet, wenn dieses um den Integrationspunkt zentriert ist (siehe Abbildung 4.26).

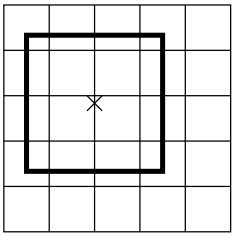

Abbildung 4.26: Um einen Integrationspunkt zentriertes kubisches Integrationsgebiet

Die daher benötigte Gewichtsfunktion, welche den innerhalb des Integrationsgebietes liegenden Anteil eines Voxel angibt, ist von dem Punkt, in dem die Trägheitsmomente berechnet werden sollen, abhängig und zerstört damit die Anwendbarkeit der Methode. Somit bleibt nur das ineffiziente naive Verfahren.

Wegen der dargestellten Nachteile ist die Variante nicht implementiert.

2. Berechnung der Trägheitsmomente in den Gitterknoten und Interpolation in die Integrationspunkte

Zunächst werden die Trägheitsmomente in den Gitterknoten bestimmt. Dann müssen sie aber in die Integrationspunkte „transportiert“ werden. Die einfachste Idee dazu ist, die Eigenwerte und -vektoren linear zu interpolieren.

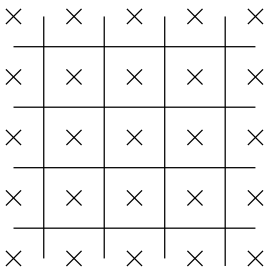

Abbildung 4.27: Duales Gitter aus Kontrollvolumina mit Knotenmarkierungen

Vorteile an dieser Variante sind die Anwendbarkeit der schnellen Methoden zur Berechnung der Trägheitsmomente und/oder der geringere Speicheraufwand. Dennoch hat auch sie gewichtige Nachteile.
Durch die Interpolation der Eigenvektoren geht deren Normierung verloren. Die interpolierten Vektoren müssen daher neu normiert werden, was zusammen mit der Interpolation ziemlich zeitintensiv ist, da diese und die dreifache Vektornormierung für *jeden* Intergrationspunkt durchgeführt werden müssen. Die Gesamtrechenzeit für die Diskretisierung lag in Experimenten um den Faktor fünf(!) über der eines Durchlaufs ohne Interpolation. Außerdem ist die Interpolation wegen der Gefahr der Auslöschung von Vektoren, die wir tatsächlich beobachten konnten, auch numerisch inakzeptabel.

Auch diese Variante ist in der Implementierung nicht verfügbar.

3. Einheitliche Berechung der Trägheitsmomente pro Element bzw. Kontrollvolumen

Bei dieser Variante werden die Trägheitsmomente als konstant auf dem zum Knoten gehörenden Kontrollvolumen angenommen. Sie müssen damit nicht mehr in die Integrationspunkte interpoliert, sondern nur noch in den Gitterknoten berechnet werden. Damit ist auch der Diffusionstensor konstant auf jedem Kontrollvolumen. Die Variante ist die effizienteste der drei vorgestellten und daher als einzige implementiert.
Man kann sie auch als direkte Berechnung der Trägheitsmomente in den Integrationspunkten interpretieren, wenn man das Integrationsgebiet nicht mehr um diese, sondern um den Knoten, zu dem der jeweilige Integrationspunkt gehört, zentriert. Da man in der Wahl des Integrationsgebietes aber ohnehin flexibel ist, stellt das kein Problem dar.

Statt die Trägheitsmomente kontenweise zu berechnen, kann man sie genauso gut elementweise, d.h. für den Mittelpunkt der Elemente des Diskretisierungsgitters, um welchen man dann das Integrationsgebiet zentriert, berechnen und den Diffusionstensor als konstant auf jedem Element annehmen. Dazu muss das Integrationsgebiet eine gerade Anzahl von Voxeln in jede Raumrichtung besitzen.

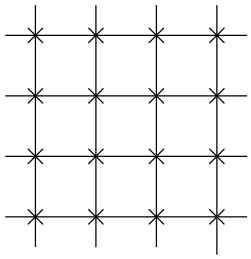

Abbildung 4.28: Duales Gitter aus Kontrollvolumina mit Markierung der Elementmittelpunkte

Die Kugel ist bei uns als VOLUME-Datenstruktur so angelegt, dass sie immer einen Bildpunkt als Mittelpunkt hat. Bei Benutzung eines kugelförmigen Integrationsgebietes werden damit die Trägheitsmomente immer knotenweise berechnet[9]. Bei der Benutzung eines kubusförmigen Integrationsgebietes hat man dagegen prinzipiell beide Möglichkeiten.
Bei der Implementierung der schnellen Methoden zur Berechnung der Trägheitsmomente stehen jedoch nicht immer beide Möglichkeiten zur Verfügung. Da sich die Benutzung der FFT normalerweise nur für kugelförmige Integrationsgebiete lohnt, wurde diese nur für die knotenweise Berechnung implementiert. Die Be-

[9]Würde man die Datenstruktur anders organisieren, fiele diese Einschränkung jedoch weg.

schleunigung durch Zusammenfassung von Punkten dagegen ist nur für den elementzentrierten Fall geschrieben[10].

Anzumerken ist noch, dass man eine symmetrische Diskretisierungsmatrix erhält, falls der Diffusionstensor $D(u)$ als konstant auf jedem Element angenommen wird. Das folgt direkt aus der Symmetrie der Ansatzfunktionen und der symmetrischen Lage der Integrationspunkte innerhalb der Elemente. Nimmt man den Diffusionstensor dagegen als konstant auf jedem Kontrollvolumen an, kann man keine Symmetrie der Diskretisierungsmatrix erwarten.

4.6.3 Interagierende Knoten

Eigentlich besitzt jeder Knoten unseres Gitters pro Element, zu dem er gehört, drei Integrationspunkte (vgl. Kapitel 3.4). Jedoch gehört jeder davon auch noch zu genau einem anderen Knoten des Elements. Diesen anderen Knoten wollen wir *interagierenden Knoten* nennen.
Der Beitrag $(D(u)\nabla\varphi_j(\mathrm{ip}))\cdot\vec{n}\cdot|S|$, den der Integrationspunkt ip zum Eintrag a_{ij} der Diskretisierungsmatrix liefert, unterscheidet sich für die beiden Knoten, zu dem er gehört, nur durch das Vorzeichen der Normalen $\vec{n}$. Es bietet sich also an, den Beitrag nur einmal zu berechnen und den mit -1 multiplizierten Wert dem interagierenden Knoten zuzuteilen.
Damit ist es aber auch nicht mehr notwenig, dass sich der interagierende Knoten den Integrationspunkt ip noch merkt. Die Anzahl der Integrationspunkte, die gespeichert und durchlaufen werden müssen halbiert sich.
In einem zweidimensionalen Referenzelement ist pro Knoten nur noch die Speicherung eines Integrationspunktes nötig. In unserem dreidimensionalen Referenzkubus müssen für die eine Hälfte der Knoten nur zwei, für die andere sogar nur ein Integrationspunkt gespeichert werden. Mit den Nummerierungen aus Abbildung 4.29 nehmen wir in die in der Tabelle 4.4 dargestellten Zuweisungen vor.

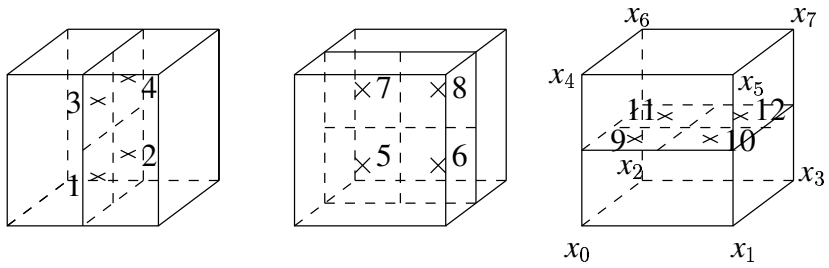

Abbildung 4.29: Nummerierung der Integrationspunkte

[10]Die etwas langsamere Methode ohne Verschiebung der Scheiben ist allerdings auch knotenweise vorhanden.

Knoten	Integrations-punkte	Interagierender Knoten
x_0	ip_1	x_1
	ip_9	x_4
x_1	ip_6	x_3
	ip_{10}	x_5
x_2	ip_5	x_0
	ip_{11}	x_6
x_3	ip_2	x_2
	ip_{12}	x_7
x_4	ip_3	x_5
x_5	ip_8	x_7
x_6	ip_7	x_4
x_7	ip_4	x_6

Tabelle 4.4: Zuweisung der Integrationspunkte mit den zugehörigen interagierenden Knoten zu den einzelnen Knoten des Referenzwürfels

Wir teilen also den elementlokal „unten“ angesiedelten Knoten zwei, den elementlokal „oben“ angesiedelten Knoten einen Integrationspunkt zu.

4.6.4 Algorithmus zur Assemblierung der Diskretisierungsmatrix

Bei der Assemblierung der Matrix laufen wir nicht über die Knoten des Gitters, sondern über die Elemente. Dies hat den Vorteil, die Nummern der Knoten des Elementes nur einmal berechnen zu müssen. Diese erhalten wir bei lexikographischer Nummerierung sowohl der Knoten als auch der Elemente durch

$$
\begin{aligned}
\text{help} &= 1 + e + ((e/(n-1))\%(m-1)) \\
&\quad + (n+m-1) * (e/((n-1)*(m-1))) \\
en[0] &= \text{help} \\
en[1] &= \text{help} + 1 \\
en[2] &= \text{help} + n \\
en[3] &= \text{help} + n + 1 \\
en[4] &= \text{help} + n * m \\
en[5] &= \text{help} + n * m + 1 \\
en[6] &= \text{help} + n * m + n \\
en[7] &= \text{help} + n * m + n + 1
\end{aligned}
$$

Dabei sind *e* die Nummer des Elementes, *n* und *m* die Größe des DATACUBE in x- und y-Richtung und *en* ein Feld, in das die globalen Nummern der Knoten des Elementes *e* entsprechend der lokalen Anordnung eingetragen werden. Das Addieren der Eins auf die Variable help ist nötig, da wir die Aufzählung der Elemente und Knoten bei Null beginnen, die der Matrixzeilen jedoch bei Eins beginnt.
Innerhalb der Elemente durchläuft man nun die lokalen Knoten *i* und für jeden Knoten die ihm zugeteilten Integrationspunkte ip_{i_k} (vgl. Abschnitt 4.6.3). Für jeden Integrationspunkt werden dann die Ansatzfunktionen *j* der lokalen Knoten durchlaufen, der Beitrag $diff := (D \cdot \nabla\varphi_j(\mathrm{ip}_{i_k})) \cdot \vec{n_{i_k}} \cdot 0.25$ zum Matrixeintrag $a_{en[i],en[j]}$ addiert und der Beitrag $-diff$ zum Matrixeintrag $a_{en[intact[i][k]],en[j]}$. Dabei bezeichnen wir mit $intact(i,k)$ den interagierenden Knoten von *i* zum Integrationspunkt *k*.
Die Trägheitsmomente werden vor der Assemblierung für das ganze Bild berechnet.

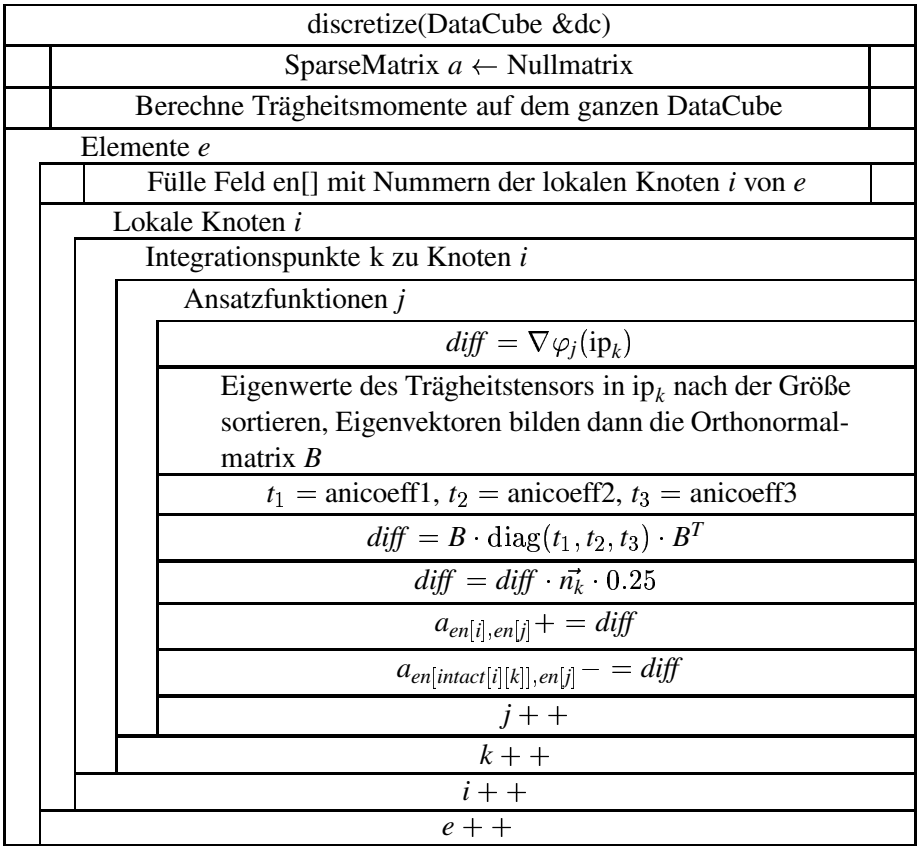

Abbildung 4.30: Diskretisierungsalgorithmus

In Abbildung 4.30 ist der Algorithmus für den Fall fester Anisotropiekoeffizienten dargestellt.

4.7 Löser

Die Klassen in diesem Abschnitt wurden von Christoph Reisinger geschrieben.

4.7.1 Die Klasse MULTIGRID

Zur Lösung der linearen Gleichungssysteme verwenden wir die Klasse MULTIGRID. Diese stellt Datenstrukturen und Methoden zur Lösung von linearen Gleichungssystemen mit Hilfe eines Mehrgitterverfahrens zur Verfügung. Es wird davon ausgegangen, dass das Gleichungssystem aus einem auf einem strukturierten mehrdimensionalen Quadergitter definierten Problem resultiert. Bei der Vergröberung des Gitters wird in jede Dimension jede zweite Gitterreihe entfernt.
Die Grobgittermatrizen werden mittles des Galerkin-Ansatzes berechnet.

Die Funktionalität von MULTIGRID ist auf einige weitere Klassen verteilt. Die Klasse PROLONGATION beherbergt die Transferoperationen, SMOOTHER die Glätter und DIRECTSOLVER die Grobgitterlöser. BASESOLVER fasst gemeinsame Funktionalität iterativer Verfahren zusammen.

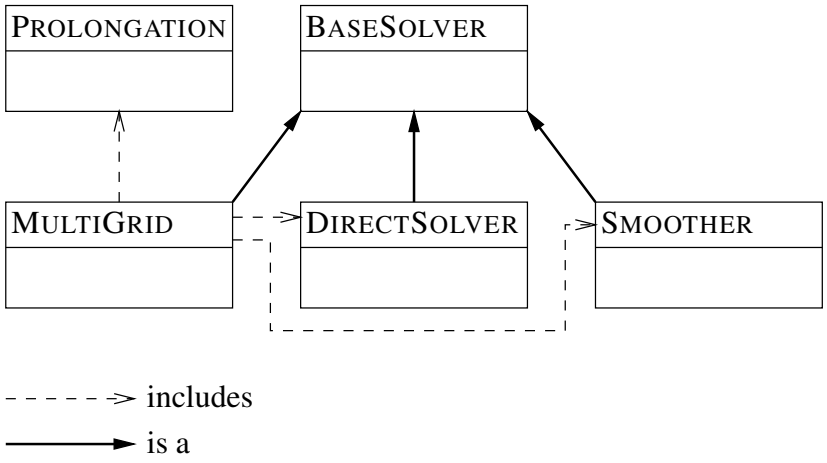

Abbildung 4.31: Zusammenhang der Löser-Klassen

In der Hauptklasse MULTIGRID sind folgende Parameter einstellbar:

- Transferoperationen
- Glätter
- Anzahl der Vorglättungsschritte
- Anzahl der Nachglättungsschritte
- Art des Mehrgitterzyklus (V oder W)

- Grobgitterlöser
- Anzahl der Gitterlevel
- Gewünschte Reduktion des Anfangsdefektes
- Maximale Zyklenanzahl

Insbesondere ist durch Setzen der Anzahl der Gitterlevel auf 1 ein gewöhnliches iteratives Verfahren, der Grobgitterlöser allein, wählbar.

4.8 Steuerung des Filters

4.8.1 Die Klasse NLD

Die Klasse NLD ist die Hauptklasse der Filterung, über die diese gesteuert wird. Als Attribute besitzt NLD zahlreiche Parameter, die die Filterung beeinflussen. Das sind zum einen die Diskretisierungsparameter, die auch die Klasse FV_3D27 besitzt. Von NLD aus werden sie an diese weitergegeben.
Außerdem stehen als Attribute zur Zeitdiskretisierung zur Verfügung:

time_steps: Anzahl der Zeitschritte

tau: Zeitschrittweite

Als Attribute für den Löser stehen bereit:

precision: Gewünschte Reduktion des Anfangsdefektes

levels: Anzahl der Mehrgitterlevel

Die weiteren Attribute des Lösers (Anzahl der Glättungsschritte etc.) sind nicht von außen, sondern nur durch Veränderung der Implementierung von NLD steuerbar.

Schließlich besitzt NLD einen Konstruktor, dem ein Zeiger auf ein Objekt vom Typ DATACUBE übergeben werden muss. So wird das Bild, das gefiltert werden soll, festgelegt. Im Konstruktor werden auch sinnvolle Default-Werte für die einzelnen Attribute festgelegt.

Die Filterung wird durch Aufruf der Methode execute() gestartet.

5 Anwendung und Tests

Wenn in den einzelnen Abschnitten nichts anderes angegeben ist, wählen wir als standartmäßige Filteroptionen

- time_steps = 2
- tau = 2.0
- epsilon = 1e-08
- levels = 1
- integration_size_x = integration_size_y = integration_size_z = 10
- GeometryType = CUBE
- ip_flag = IP_USUAL
- fixed_coeffs = YES
- anicoeff1 = 1, anicoeff2 = 0, anicoeff3 = 0
- DependenceType = PERONA_MALIK
- lambda = 0.1

Die Konvention, mit dem Begriff Kubus ebenfalls den allgemeineren Fall des Quaders einzuschließen, ist in diesem Kapitel aufgehoben. Das Gleiche gilt für die Begriffe Kugel und Ellipse.

5.1 Größe und Geometrie des Integrationsgebietes

Die lokale Strukturerkennung durch Trägheitsmomente steuert die anisotrope Diffusion und ist damit die für die Qualität der Filterung wesentliche Einflussgröße. Es stellt sich die Frage, wie groß dabei das Gebiet, auf dem die lokale Strukturerkennung durchgeführt wird, sein soll. Einsichtig ist, dass dieses weder zu klein, noch zu groß gewählt werden darf, da in beiden Fällen ein korrektes Erkennen der lokalen Struktur nicht erwartet werden kann.
Dazu betrachten wir einen Stab der Breite und Tiefe 3 Voxel und sehr viel größerer Länge, der parallel zur x-Achse ausgerichtet ist, und eine Lücke von drei Voxeln aufweist (siehe Abb. 5.1). Wir wollen feststellen, welche Trägheitsmomente unterschiedliche Integrationsgebietsgrößen liefern und ob die Strukturerkennung trotz der Unterbrechung funktioniert.
Wir wählen kubische Integrationsgebiete, die wir um das Stabende an der Lücke zentrieren. Die Ergebnisse sind in der Tabelle 5.1 zusammengestellt. Die Abkürzung HDR steht dabei für Hauptdiffusionsrichtung.

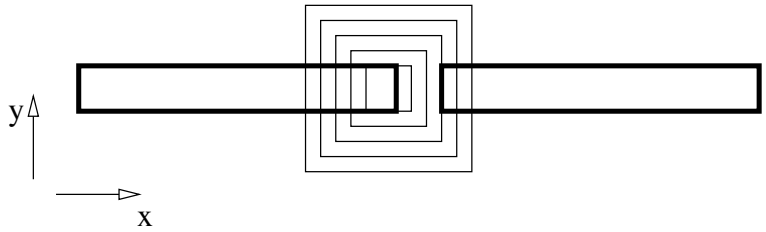

Abbildung 5.1: Schnitt auf z-Ebene durch die Teststruktur mit Integrationsgebieten der Seitenlänge *s*

	$s = 3$	$s = 5$	$s = 7$	$s = 9$	$s = 11$
c_l	0.00	0.00	0.23	0.75	0.84
c_p	0.53	0.00	0.00	0.00	0.00
c_i	0.47	1.00	0.77	0.25	0.16
HDR	y,z	x,y,z	x	x	x

Tabelle 5.1: Ergebnisse der Strukturerkennung aus dem Beispiel in Abbildung 5.1

Wir stellen fest, dass die Seitenlänge des Integrationsgebietes den Durchmesser der zu erkennenden Strukturen überschreiten muss, um die Hauptdiffusionsrichtung wie gewüscht zu erkennen. Dieses Kriterium reicht jedoch zur Strukturerkennung nicht aus. Im Fall von s = 7 wird zwar die Hauptdiffusionsrichtung richtigerweise als x erkannt, die Struktur wegen $c_i = 0.77$ aber als eher isotrop. Selbst für größere Integrationsgebiete wird der Stab nicht als rein linear erkannt. Für die saubere Erkennung breiterer linearer Strukturen sind also sehr große Integrationsgebiete notwendig.
Zu große Integrationsgebiete verfälschen jedoch die lokale Strukturerfassung ebenfalls. Betrachten wir dazu die Abbildung 5.2.

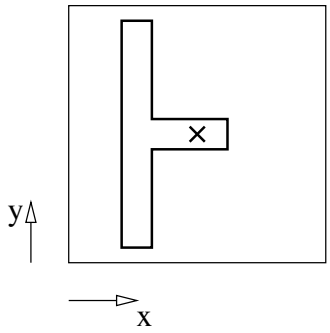

Abbildung 5.2: Zu groß gewähltes Integrationsgebiet

Als Hauptdiffusionsrichtung wird y erkannt, obwohl die Struktur lokal in x-Richtung ausgerichtet ist.

Wir erwarten daher, dass wir für mittlere Integrationsgebietsgrößen die besten Filterergebnisse erhalten.
Erfreulich ist, dass die Lücke die Erkennung der Struktur nicht verhindert.

Wir setzen nun $D = \mathrm{diag}(1, 0, 0)$, d.h. wir lassen Diffusion nur in Richtung der Hauptdiffusionsrichtung zu, und filtern unter Verwendung kubischer Integrationsgebiete verschiedener Größen. In Abbildung 5.3 sind die Ergebnisse für drei ausgewählte Größen gezeigt.

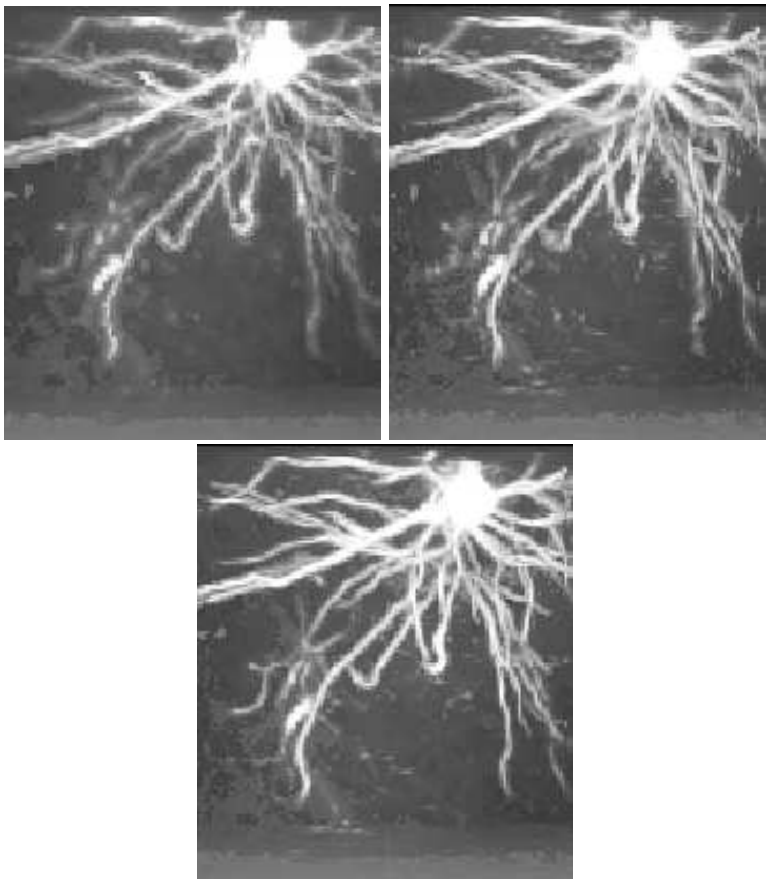

Abbildung 5.3: Ergebnisse der Filterung dargestellt als 3D-Projektion für die Integrationsweiten 2 (oben links), 10 (unten mitte) und 30 (oben rechts). Der Bildausschnitt hat eine Seitenlänge von 129 Voxeln in jede Raumrichtung.

Wie erwartet ist das Ergebnis für mittlere Größen am besten. Diese sollten im Bereich von 8 bis 14 Voxeln liegen. Dennoch liefert die Filterung selbst für sehr große Werte wie 30 (das ist immerhin fast ein Viertel der Bildbreite) oder sehr kleine wie

2, bei der man kaum noch oder falsche Strukturerkennung erwarten kann, verhältnismäßig überraschend gute Ergebnisse.

Als nächstes wollen wir untersuchen, ob sich die Verwendung eines kugelförmigen Integrationsgebietes in der praktischen Anwendung erkennbar auf die Filterergebnisse auswirkt. Zunächst betrachten wir das zweidimensionale Beispiel, das in der Abbildung 5.4 gezeigt ist. Dabei interessieren uns die erkannten Strukturen für die mit v_1, v_2 und v_3 bezeichneten Voxel.

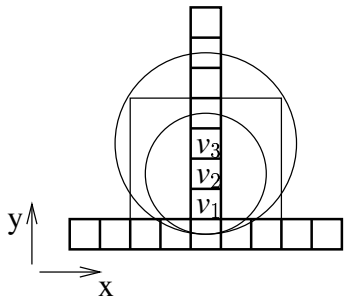

Abbildung 5.4: Beispiel mit verschiedenen Integrationsgebieten

In der Tabelle 5.2 sind die erkannten Strukturen zusammengestellt. Dabei sind in der horizontalen Achse die Voxel, um die das Integrationsgebiet zentriert wurde, sowie dessen Größe angegeben, wobei die erste Zahl die Größe des kubusförmigen, die zweite Zahl die des kugelförmigen Gebietes angibt. Der Übersichtlichkeit halber sind in Abbildung 5.4 nicht alle Integrationsgebiete gezeigt.

	$s = 5$ bzw. 4	s = 7 bzw. 6		
	v_2	v_3	v_2	v_1
Kubus	$c_l = 0.31$ $c_p = 0.69$ HDR = y	$c_l = 0.34$ $c_p = 0.66$ HDR = y	$c_l = 0.13$ $c_p = 0.87$ HDR = y	$c_l = 0.14$ $c_p = 0.86$ HDR = x
Kugel	$c_l = 0.89$ $c_p = 0.11$ HDR = y	$c_l = 0.94$ $c_p = 0.06$ HDR = y	$c_l = 0.52$ $c_p = 0.48$ HDR = y	$c_l = 0.03$ $c_p = 0.97$ HDR = x

Tabelle 5.2: Strukturerkennung für das Beispiel in Abbildung 5.4

Es fällt auf, dass die Kugel bei größerer Entfernung des Mittelpunktes des Integrationsgebietes von der horizontalen Querstruktur die lokal lineare Struktur besser erfasst als der Kubus. Der erkennt die Struktur sogar immer eher als planar. Das muss er aber auch, sieht er schließlich den zur x-Achse parallelen Stab auf voller Integrationsgebietsbreite. Die Kugel dagegen sieht davon nur einen Teil. Daher

scheint es für dieses Beispiel vorteilhaft zu sein, die Kugel als Integrationsgebiet zu verwenden.
Bei Betrachtung der erkannten Hauptdiffusionsrichtung relativiert sich diese Aussage jedoch, da sie durchgehend gleich erkannt wird.

Um eine wirkliche Bewertung vornehmen zu können, vergleichen wir die Ergebnisse der Filterung bei Verwendung eines kugelförmigen und eines kubusförmigen Integrationsgebietes miteinander. Die Diffusion wird wieder nur in Hauptdiffusionsrichtung zugelassen. Die Ergebnisse zeigt die Abbildung 5.5. Da bei der Verwendung des kugelförmigen Integrationsgebietes zur Trägheitsmomentenberechnung mit der FFT gearbeitet wurde, wurde diese zur Vergleichbarkeit auch bei der Verwendung des kubusförmigen Gebietes benutzt und der Bildausschnitt auf eine Seitenlänge von 128 Voxeln reduziert.

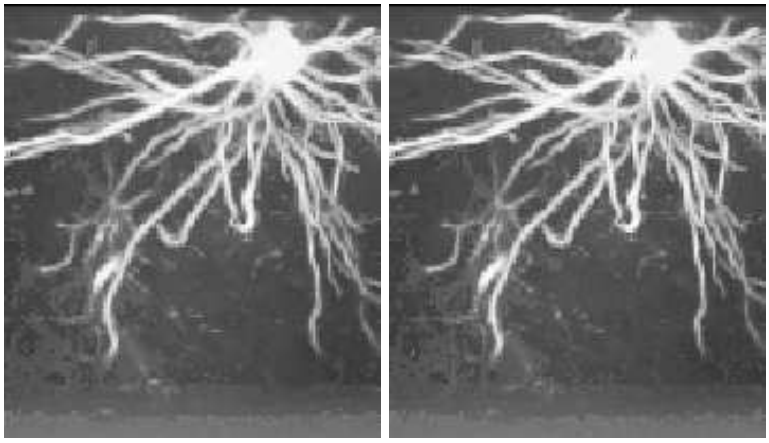

Abbildung 5.5: Ergebnisse der Filterung unter Verwendung eines kubusförmigen (links) und eines kugelförmigen Integrationsgebietes (rechts)

Zwischen den beiden Filterungen ist kein Unterschied erkennbar. Offenbar ist es für unsere Art der Aufnahmen irrelevant, ob ein kubus- oder kugelförmiges Integrationsgebiet verwendet wird, wenn man nur in Hauptdiffusionsrichtung glättet.
An den durchgeführten Filterungen lässt sich allerdings beobachten, dass sich die bei der Fourier-Transformation benutzte periodische Fortsetzung des Bildes über den Rand hinaus offenbar positiv auf die Strukturerkennung am Rand auswirkt. Dazu vergleiche man die Strukturen im unteren Bereich des rechten Randes mit den Ergebnissen, die in der Abbildung 5.3 gezeigt sind.

5.2 Strukturunterscheidung durch variable Anisotropiekoeffizienten

Bislang haben wir die Diffusion nur in die Hauptdiffusionsrichtung zugelassen, haben also prinzipiell lineare Strukturen gesucht. Nun wollen wir die drei Strukturfälle linear, planar und isotrop unterscheiden und entsprechend der lokalen Struk-

tur in unterschiedliche Richtungen glätten. Dazu haben wir in Abschnitt 3.3.4 den Vorschlag gesehen, die Anisotropiekoeffizienten gemäß

$$\begin{aligned} t_1 &= 1.0 \\ t_2 &= g(c_l) \\ t_3 &= g(1 - c_i) \end{aligned} \tag{5.1}$$

zu setzen. Dabei sollte g eine der Funktionen aus Abschnitt 3.2.6 sein, die zur Steuerung der nichtlinearen Diffusion in zwei Dimensionen eingesetzt wird.
Prinzipiell wird also mehr Diffusion zugelassen. Problematisch ist bei diesem Ansatz, dass die einzelnen Strukturen sauber erkannt werden müssen, um diese nicht durch falsche Zuordnung zu verwischen. Das Beispiel des unterbrochenen Stabes im vorherigen Abschnitt hat uns jedoch gezeigt, dass dies schwierig ist, da der Stab auch bei großen Integrationsgebietsweiten nicht als rein linear erkannt wurde. Nach dem Ansatz (5.1) wird ein solcher Stab wohl auch immer eine unerwünschte Querdiffusion erfahren.

Wir testen nun den Ansatz (5.1) und setzten als g die Weikert'sche Funktion mit $\lambda = 0.1$. Alle übrigen Parameter lassen wir gegenüber dem letzten Abschnitt unverändert. In Abbildung 5.6 sind die Ergebnisse gezeigt.

Abbildung 5.6: Filterung mit $D = \mathrm{diag}(\ 1.0,\ g(c_l),\ g(1-c_i)\ ),\ g =$ WEIKERT, $\lambda = 0.1$ für ein kubusförmiges Integrationsgebiet (links) und ein kugelförmiges Integrationsgebiet (rechts)

Diese sind schlecht und kommen dem Fall der linearen Diffusion nahe, obwohl bereits Integrationsgebiete mittlerer Größe, nämlich mit einem Durchmesser von 10 Voxeln benutzt wurden. Auch macht es wieder keinen Unterschied, ob die Strukturerkennung mittels kubus- oder aber kugelförmiger Integrationsgebiete durchgeführt wird. Also verbessert die Kugel offenbar auch die Strukturerkennung nicht signifikant, wo sie schon bei der Erkennung der Hauptdiffusionsrichtung gegenüber dem Kubus keinen Vorteil zeigte.

Die unerwünschte Querdiffusion lässt sich vielleicht durch eine noch kleinere Wahl von λ oder aber durch die Wahl einer anderen Funktion g verringern. Ausschalten kann man sie wegen der oben erwähnten nicht ganz sauber durchführbaren Strukturerkennung jedoch nicht.
Daher erscheint es sinnvoll, auf die Strukturunterscheidung zu verzichten und nur in Hauptdiffusionsrichtung zu glätten. Der Filter sucht somit nur nach linearen Strukturen. Da wir es in unseren Aufnahmen mit Ausnahme des Zellkörpers, der ohnehin deutlich aufgenommen ist, nur mit linearen Strukturen zu tun haben, stellt das kein Problem dar. Die durch diese Art der Filterung begünstigte Schlierenbildung, die auch lineare Strukturen entstehen lässt, wo sich eigentlich nur Rauschen befindet, kann die nachfolgende Rekonstruktion eliminieren.

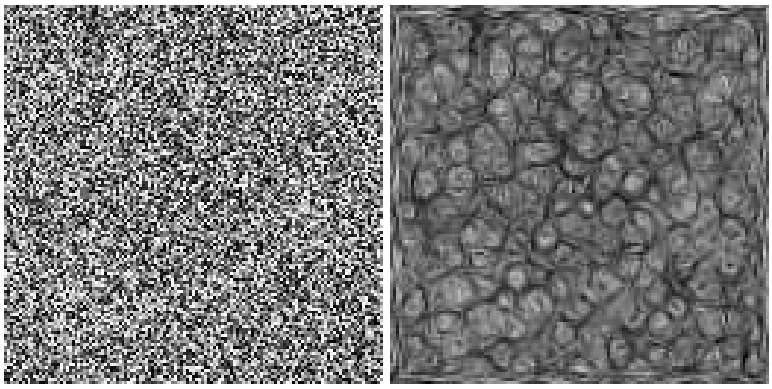

Abbildung 5.7: Schlierenbildung durch Filterung nur in Hauptdiffusionsrichtung. Die am Rand entstehenden Streifen sind auf das Abschneiden des Bildes am Rand bei der Strukturerkennung zurückzuführen, ein weiterer Hinweis darauf, dass das Bild dabei periodisch fortgesetzt werden sollte. Bei Verwendung der Fourier-Transformation bei der Berechnung der Trägheitsmomente treten die Streifen nicht auf.

5.3 Zeitschrittzahl und Zeitschrittweite

Neben der Bestimmung der Diffusionsrichtungen stellt die Festlegung der Zahl der Zeitschritte sowie deren Größe ein zweites Kriterium dar, welches die Qualität der gefilterten Bilder wesentlich beeinflusst. Dabei ist man in der Wahl der Zeitschritte ziemlich frei, geht es schließlich nicht darum, ein physikalisches Phänomen zeitecht zu simulieren, sondern um die Aufbereitung von Bildern. Daher kann auch nur der gewünschte Effekt, den man durch die Filterung erzielen will, ein Kriterium für die Wahl der Zeitschrittzahl und -weite sein.
Ein mit der Filterung verfolgtes Ziel ist das Schließen von Strukturlücken. Um ein Gefühl für die zum Schließen einer solchen Lücke nötigen Zeitschritte zu bekommen, führen wir eine Studie an einem Testbeispiel durch.

Wir definieren ein Bild der Größe 65^3 mit einer Y-förmigen Struktur, die drei Vo-

xel hoch und breit ist und Lücken der Größe drei Voxel aufweist. Voxel, die zur Struktur gehören, erhalten den Anfangswert 1, alle anderen 0. Die Struktur wird orthogonal zur z-Achse in einen Bildkubus gelegt. Abbildung 5.8 zeigt den Schnitt durch die mittlere Ebene.

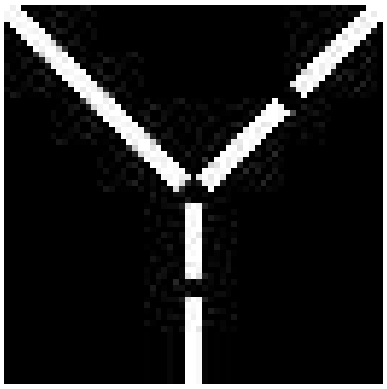

Abbildung 5.8: Testbild

Wir wählen verschiedene Zeitschrittweiten aus und zählen die Anzahl der Zeitschritte, die nötig ist, um die Lücken in der Struktur zu schließen. Dabei gilt eine Lücke als geschlossen, wenn in ihrem mittleren Voxel mindestens der Wert 0.3 erreicht ist. Die Diffusion wird nur in Hauptdiffusionsrichtung zugelassen.
In der Tabelle 5.3 haben wir die Ergebnisse zusammengestellt. Horizontal sind die verschiedenen Zeitschrittweiten, vertikal die drei unterschiedlichen Lücken aufgelistet. Eingetragen ist jeweils die Anzahl der zur Schließung der Lücke nötigen Zeitschritte.

	10.0	2.0	1.0	0.75	0.5	0.3	0.15	0.1	0.05
Gerade Lücke	1	1	2	2	3	5	8	12	24
Mittlere Lücke	1	2	3	4	5	8	16	23	46
Diagonale Lücke	1	1	2	2	3	5	9	13	25

Tabelle 5.3: Zum Schließen der Lücken in Abbildung 5.8 benötigte Zeitschritte

Die Lücken werden alle gut aufgefüllt. Querdiffusion ist kaum vorhanden, was vor allem an der durch die Anisotropie hervorgerufene Rückwärtsdiffusion liegt.
Qualitativ sind die Ergebnisse für das Testbild bei Lückenschließung immer gleich (siehe dazu die Abbildung 5.9), woraus der Schluss gezogen werden könnte, man sollte stets möglichst große und dafür wenige Zeitschritte wählen, um die Rechenzeit gering zu halten.

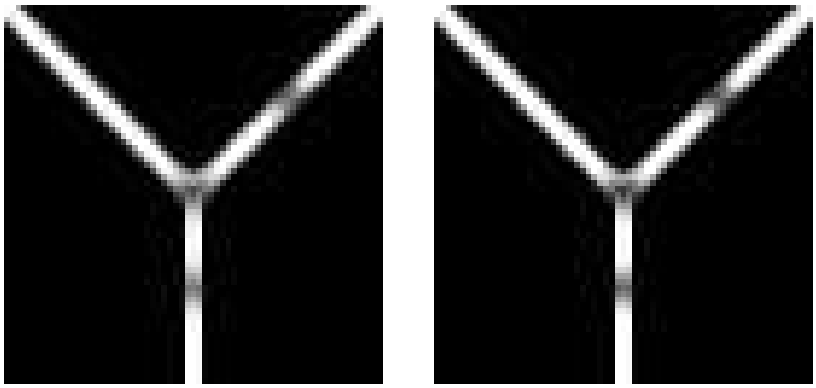

Abbildung 5.9: Ergebnis der nichlinearen anisotropen Diffusion mit $\tau = 2.0$ nach 2 Zeitschritten (links) und $\tau = 0.05$ nach 46 Zeitschritten (rechts)

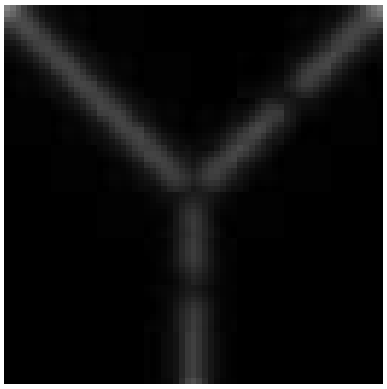

Abbildung 5.10: Lineare Diffusion mit $\tau = 2.0$ nach 2 Zeitschritten. Das Verschwimmen der Struktur ist nicht direkt erkennbar, zeigt sich aber durch den hohen Kontrastverlust.

Fraglich ist, ob das auch für die verrauschten Nervenzellenaufnahmen gilt. Die Strukturerfassung durch die Trägheitsmomente wird auch durch das Rauschen beeinflusst. Sie kann sich im Laufe der Filterung ändern, wenn die Strukturen deutlicher herausgearbeitet werden. Der Diffusionstensor D ist also nicht zeitunabhängig und sollte daher im Prinzip in kleinen Zeitschritten linearisiert werden.

Wir führen zwei Filterungen auf den Neuronenaufnahmen durch. Bei der ersten wählen wir eine Zeitschrittweite von 2.0 bei 2 Zeitschritten, bei der zweiten eine Zeitschrittweite von 0.05 bei 46 Zeitschritten. Die Ergebnisse der Filterungen sind in Abbildung 5.11 zu sehen.

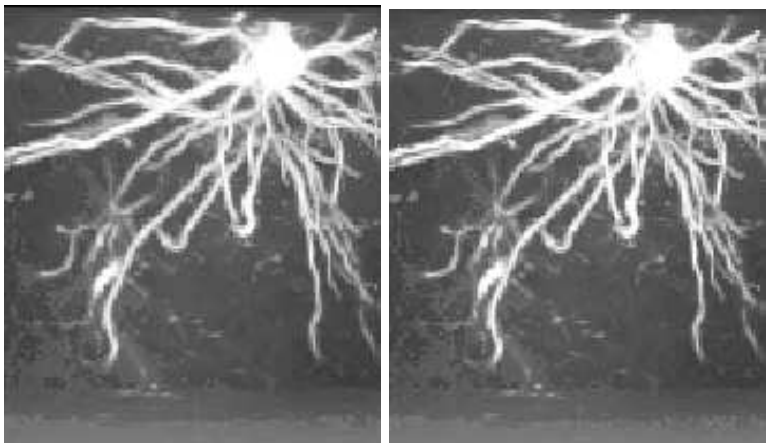

Abbildung 5.11: Filterergebisse mit $\tau = 2.0$ nach 2 Zeitschritten (links) und mit $\tau = 0.05$ nach 46 Zeitschritten (rechts)

Unterschiede zwischen den Ergebnissen sind praktisch nicht festzustellen. Offenbar verändert sich der Diffusionstensor D in der Zeit nur unwesentlich. Große Zeitschritte können also problemlos auf die Neuronenaufnahmen angewandt werden, eine häufige Linearisierung von D ist nicht nötig.

Ungeklärt ist damit aber nach wie vor die Anzahl der Zeitschritte, die man wählen sollte, bzw. die sogenannte *Stoppzeit*. Das ist die Anzahl der Zeitschritte multipliziert mit der Zeitschrittweite. Wir haben durch die obige Filterung auf dem Testbild zwar ein Gefühl für die Anzahl der zur Lückenschließung nötigen Zeitschritte bekommen, aber verfügen noch immer über kein echtes Kriterium, welches uns die Stoppzeit angibt.
Leider muss dies an dieser Stelle auch ungeklärt bleiben. Weikert[17] und Mrázek[9] geben zwar Überlegungen und Kriterien für die Stoppzeit an. Letztlich orientieren sich diese aber an der Vorgabe, ein möglichst „schönes“ Bild zu erhalten. Das kann uns aber als Kriterium nicht reichen.
Wir wollen mit den gefilterten Aufnahmen eine Rekonstruktion von Neuronen durchführen und nicht bloß die gefilterten Bilder betrachten. Damit muss für die

Stoppzeit ein Kriterium gefunden werden, welches eine optimale Rekonstruktion ermöglicht. Eine Möglichkeit bestünde zum Beispiel darin, die Diffusion dann zu stoppen, wenn bei der Rekonstruktion des Dendritenbaumes eine minimale Anzahl von zusammenhängenden Bäumen erkannt wird, da dann offenbar alle Lücken geschlossen sind (welche man durch Diffusion schließen kann). Die Aufgabe, dieses Kriterium zu finden, muss aber Aufgabe der Rekonstruktion sein und wird in dieser Arbeit daher nicht weiter verfolgt.
Zuletzt zeigen wir noch anhand zweier Filterungen, dass sich das Fiterergebnis bei der Wahl einer zu großen Stoppzeit wieder verschlechtert. In der Abbildung 5.12 sind die Ergebnisse für die Stoppzeiten 10 und 100 gezeigt.

Abbildung 5.12: Filterergebnisse mit $\tau = 10$ (links) und mit $\tau = 100$ (rechts) nach jeweils einem Zeitschritt

Deutlich ist im rechten Bild der eintretende Kontrastverlust zu sehen. Es macht also Sinn, nach einem Kriterium zur Festlegung der Stoppzeit zu suchen.

Ausblick: Ideales Ziel dieser Art der Filterung wäre die Konstruktion eines Diffusionstensors, durch dessen Benutzung die zeitabhängige Diffusion gegen ein nichtkonstantes Fixbild konvergiert. Dann könnte man die Diffusionsgleichung stationär lösen und wäre den Parameter der Stoppzeit los.

5.4 Konvergenzverhalten unterschiedlicher Löser

Die Diskretisierung hatte uns auf das Problem geführt, in jedem Zeitschritt ein lineares Gleichungssystem $(\mathbb{I} - \tau A)x^{i+1} = x^i$ zu lösen. In diesem Abschnitt untersuchen wir dazu das Konvergenzverhalten verschiedener iterativer Löser.

Wir betrachten die Tabelle 5.4. Dargestellt sind auf der horizontalen Achse unterschiedliche iterative Löser, insbesondere verschiedene Mehrgittervarianten. Der BiCGStab-Löser ist eine Gradientenverfahren-Variante. Auf der vertikalen Achse

sind verschiedene Zeitschrittweiten, die das lineare Gleichungssystem parametrisieren, gelistet. Gemessen wurde jeweils die Anzahl der für eine Reduzierung des Anfangsdefektes um den Faktor 10^8 nötigen Iterationen, die dafür benötigte Zeit in Sekunden und die mittlere Konvergenzrate $\bar{\varrho}$ (der benötigten Iterationen). Auf die Lösungszeit der Mehrgittervarianten sind jeweils noch 28 Sekunden, die für die Berechnung der Grobgittermatrizen nach dem Galerkin-Ansatz benötigt wurden, hinzuzurechnen.
Es wurde auf einem kubischen Bild mit einer Kantenlänge von 65 Voxeln gerechnet. Zur Strukturerkennung wurde ein kubusförmiges Integrationsgebiet mit einer Kantenlänge von 10 Voxeln benutzt[1]. Damit werden die Trägheitsmomente und die Diffusionsrichtungen elementweise konstant bestimmt und die Matrizen $\mathbb{I} - \tau A$ sind symmetrisch (vergl. Abschnitt 4.6.2). Die Resultate im unsymmetrischen Fall bei konstanten Diffusionsrichtungen pro Kontrollvolumen sind sehr ähnlich. Die Mehrgittervarianten arbeiten mit jeweils zwei Vor- und Nachglättungsschritten.

Zeitvergleich
Die Konvergenzraten des Gauss-Seidel-Verfahrens sind (zunächst unerwartet) weit von 1 entfernt. Die der Mehrgittervarianten liegen trotzdem weit darunter, da diese sehr viel weniger Iterationen benötigen. Da bei ihnen jedoch jede einzelne Iteration sehr viel aufwendiger ist, ist das Gauss-Seidel-Verfahren den Mehrgittervarianten zeitlich überlegen, da man zu den Zeiten der Mehrgittervarianten ja noch 28 Sekunden für die Berechnung der Grobgittermatrizen hinzurechnen muss.
Insgesamt jedoch schneidet von der Effizienz her das BiCGStab-Verfahren am besten ab.

Gauss-Seidel-Verfahren
Wie erklären sich die guten Konvergenzraten des Gauss-Seidel-Verfahrens? Da wir die Lösung des alten Zeitpunktes als Startlösung für die Iteration verwenden, ist die Differenz (der Fehler) $x^i - x$ punktweise gesehen nur an den Stellen betragsmäßig groß, an denen viel Diffusion stattfindet. Dies ist jedoch nur im relativ schmalen Bereich von Kanten im Bild möglich, oder aber in verrauschten Bereichen, in welchen aber das Vorzeichen des Fehlers schnell variiert. Ein solcher Fehler kann natürlich gut durch lokale Mittellung zwischen den Werten in einzelnen Knoten reduziert werden.
Dazu passt auch die bei Vergrößerung der Zeitschrittweite τ zu beobachtende Verschlechterung der Konvergenzraten. Bei größerer Zeitschrittweite findet mehr Diffusion statt und die Bereiche an den Bildkanten, an denen der Fehler groß ist, verbreitern sich, sind somit nicht mehr ganz so gut durch lokale Mittelung zu reduzieren.

[1] Auch die Größe des Integrationsgebietes beeinflusst das Konvergenzverhalten der Löser. Dies erklärt sich daraus, dass durch die Wahl des Integrationsgebietes die Diffusionsrichtungen und damit die Diskretisierungsmatrix beeinflusst wird.

τ		Gauss-Seidel	BiCG-Stab	Linearer Transfer		Matrixabhängiger Transfer	
				GS-Glätter	ILU-Glätter	GS-Glätter	ILU-Glätter
0.2	Zeit[s]	17	9	16	23	16	23
	Iterationen	10	6	3	2	3	2
	$\bar{\varrho}$	0.1317	0.0325	0.0003	3.7 e-7	0.0003	3.7 e-7
0.4	Zeit[s]	19	11	16	23	16	22
	Iterationen	12	8	3	2	3	2
	$\bar{\varrho}$	0.2069	0.0831	0.0017	1.0 e-5	0.0017	1.0 e-5
0.6	Zeit[s]	22	14	22	23	23	23
	Iterationen	14	10	4	2	4	2
	$\bar{\varrho}$	0.2605	0.1337	0.0042	0.0001	0.0075	0.0001
0.8	Zeit[s]	25	15	21	32	21	31
	Iterationen	16	11	4	3	4	3
	$\bar{\varrho}$	0.3055	0.1762	0.0066	0.0002	0.0066	0.0002
1.0	Zeit[s]	28	16	22	31	21	32
	Iterationen	18	12	4	3	4	3
	$\bar{\varrho}$	0.3448	0.2132	0.0092	0.0003	0.0092	0.0003
2.0	Zeit[s]	38	24	32	30	32	31
	Iterationen	25	18	6	3	6	3
	$\bar{\varrho}$	0.4772	0.3464	0.0366	0.0020	0.0367	0.0020
3.0	Zeit[s]	51	29	37	39	37	39
	Iterationen	33	22	7	4	7	4
	$\bar{\varrho}$	0.5672	0.4275	0.0708	0.0079	0.0709	0.0080
4.0	Zeit[s]	61	34	47	keine	48	keine
	Iterationen	40	26	9	Kon-	9	Kon-
	$\bar{\varrho}$	0.6286	0.4837	0.1183	vergenz	0.1181	vergenz
5.0	Zeit[s]	72	38	52	keine	53	keine
	Iterationen	47	29	10	Kon-	10	Kon-
	$\bar{\varrho}$	0.6741	0.5233	0.1573	vergenz	0.1569	vergenz
10.0	Zeit[s]	124	49	103	keine	98	keine
	Iterationen	80	38	18	Kon-	18	Kon-
	$\bar{\varrho}$	0.7938	0.6091	0.3487	vergenz	0.3481	vergenz

Tabelle 5.4: Konvergenzverhalten unterschiedlicher Löser bei einer Defektreduzierung von 10^{-8}.

BiCGStab
Der BiCGStab-Löser verhält sich ähnlich wie das Gauss-Seidel-Verfahren, nur sind die Konvergenzraten etwas besser, wodurch die gesuchte Lösung nach weniger Iterationen erreicht wird. Deswegen und weil eine einzelne Iteration minimal schneller als ein Gauss-Seidel-Schritt ist, braucht es insgesamt weniger Zeit. Auch dieses Verfahren profitiert von der „Nähe“ der Startlösung von der gesuchten Lösung, welche sich mit zunehmendem τ voneinander entfernen.

Mehrgitterverfahren
Die auf den ersten Blick besser erscheinenden Konvergenzraten der Mehrgittervarianten weisen ebenfalls einen Anstieg bei Vergrößerung von τ auf. Dies ist eigentlich eher eine Eigenschaft, die man einem Glätter zurechnet. Daher haben wir den Verdacht, dass nur die Glättungsschritte der Mehrgitteriterationen für die Defektreduzierung verantwortlich sind und die Grobgitterkorrektur praktisch keinen Effekt hat. Dieser erhärtet sich beim Vergleich zwischen den vom Gauss-Seidel-Verfahren und den Mehrgittervariationen mit Gauss-Seidel-Glätter zur Defektreduzierung nötigen Iterationen.
Das Gauss-Seidel-Verfahren benötigt ziemlich genau viermal so viele Iterationen. Da aber bei jeder Mehrgitteriteration auf dem feinsten Gitter vier Gauss-Seidel-Schritte aufgerufen werden (zwei zur Vorglättung und zwei zur Nachglättung), ist die Anzahl der verwendeten Gauss-Seidel-Glättungsschritte bei beiden Verfahren etwa gleich!
Um diese Beobachtung zu bestätigen, setzen wir die Anzahl der Vor- und Nachglättungsschritte auf 1 und führen die Mehrgitteriteration mit Gauss-Seidel-Glättung noch einmal durch. Die Ergebnisse sind in Tabelle 5.5 gezeigt.

	$\tau = 0.2$	$\tau = 0.6$	$\tau = 1.0$	$\tau = 3.0$	$\tau = 5.0$	$\tau = 10.0$
Zeit[s]	22	24	28	50	71	124
Iterationen	5	7	8	14	20	35
$\bar{\varrho}$	0.0166	0.0618	0.0943	0.2649	0.3963	0.5869

Tabelle 5.5: Konvergenzverhalten des 3-Level Mehrgitterverfahrens mit matrixabhängigen Transferoperationen und Gauss-Seidel-Glätter bei nur einem Vor- und Nachglättungsschritt

Die Halbierung der Anzahl der Glättungsschritte verdoppelt etwa die Anzahl der nötigen Mehrgitteriterationen, wodurch sich unsere Beobachtung bestätigt.
Noch weiter festigt sie sich durch die Verwendung verschieden vieler Gitterstufen. Bisher wurde immer mit drei Leveln gerechnet. Verwendet man dagegen zwei oder vier Gitterlevel, so ändern sich die Konvergenzraten nicht. Dies ist die Bestätigung dafür, dass die Grobgitterkorrektur keinen Einfluß auf die Defektreduzierung hat, was etwas enttäuschend und auch überraschend ist, da man zumindest für große Zeitschrittweiten einen Effekt der Grobgitterkorrektur hätte erwarten sollen.

Glätter

Zu beobachten ist bei den Mehrgitterverfahren auch ein Einfluss des verwendeten Glätters. So sind die Konvergenzraten bei Verwendung des ILU-Glätters zunächst besser als bei Verwendung des Gauss-Seidel-Glätters. Bei großen Zeitschrittweiten dagegen bricht die Konvergenz zusammen. Dieses Phänomen lässt sich vermutlich auf die Struktur der Diskretisierungsmatrix und den Galerkin-Ansatz zurückführen. Dazu geben wir zunächst zwei Definitionen.
Gelten für $A \in \mathbb{R}^{I \times I}$

$$\begin{aligned} &a_{ii} > 0, \quad i \in I \\ &a_{ij} \leq 0, \quad i \in I,\ j \in I,\ i \neq j \\ &A \text{ regulär und } A^{-1} \text{ positiv definit} \end{aligned}$$

so heißt A *M-Matrix*.
Weiter heißt A *schwach diagonaldominant*, falls

$$|a_{ii}| \geq \sum_{\substack{j \in I \\ j \neq i}} |a_{ij}|$$

J.W.Ruge und K.Stüben zeigen in McCormick[8], dass eine symmetrische, schwach diagonaldominante M-Matrix unter bestimmten Bedinungen an die Transferoperationen mittels des Galerkin-Ansatzes eine ebensolche Grobgittermatrix erzeugt. Unsere Feingittermatrizen sind jedoch weder M-Matrizen, noch schwach diagonaldominant (vergl. Abschnitt 3.4.2). In den berechneten Grobgittermatrizen wachsen die Einträge betragsmäßig gegenüber den Einträgen in der Feingittermatrix stark an. Außerdem treten Zeilen mit ausschließlich positiven Einträgen auf.
Dieses Verhalten wollen wir an einem eindimensionalen Fall mit $n = 5$ aufzeigen. Wir betrachten die Feingittermatrix

$$A_l = \begin{pmatrix} x & z & & & \\ y & x & z & & \\ & y & x & z & \\ & & y & x & z \\ & & & y & x \end{pmatrix}$$

mit den Transferabbildungen

$$p = \begin{bmatrix} \frac{1}{2} & 1 & \frac{1}{2} \end{bmatrix}, \qquad r = \frac{1}{2} \begin{bmatrix} \frac{1}{2} & 1 & \frac{1}{2} \end{bmatrix}$$

Die mittels des Galerkin-Ansatzes berechnete Grobgittermatrix lautet

$$A_{l-1} = \begin{pmatrix} \frac{5}{8}x + \frac{1}{4}y + \frac{1}{4}z & \frac{1}{8}x + \frac{1}{2}z & 0 \\ \frac{1}{8}x + \frac{1}{2}y & \frac{3}{4}x + \frac{1}{2}y + \frac{1}{2}z & \frac{1}{8}x + \frac{1}{2}z \\ 0 & \frac{1}{8}x + \frac{1}{2}y & \frac{5}{8}x + \frac{1}{4}y + \frac{1}{4}z \end{pmatrix}$$

Nun setzen wir $x = 2$, $y = 1$ und $z = -1$, womit A_l *keine* M-Matrix ist. Es resultiert die Grobgittermatrix

$$A_{l-1} = \begin{pmatrix} 1.25 & -0.25 & 0 \\ 0.75 & 1.5 & -0.25 \\ 0 & 0.75 & 1.25 \end{pmatrix}$$

Berechnen wir nun die Grobgittermatrix mit $x = 1.5$, $y = 0.75$ und $z = -0.25$ noch einmal, was (für innere Zeilen) die Grobgittermatrix des zweiten Levels bedeutet, erhalten wir

$$A'_{l-1} = \begin{pmatrix} 1.0625 & 0.0625 & 0 \\ 0.5625 & 1.375 & 0.0625 \\ 0 & 0.5625 & 1.0625 \end{pmatrix}$$

Die Matrix hat nur noch positive Einträge.
Weiter sind, falls $y + z > \frac{1}{2}x$ ist, die Diagonaleinträge der Grobgittermatrix größer als die Einträge der Feingittermatrix, was sich rekursiv auf die weiteren Level fortsetzt. Das wäre beispielsweise für $x = 2$, $y = 1.5$ und $z = -0.4$ der Fall.
Analog zu diesem eindimensionalen Beispiel sind das Anwachsen und die rein positiven Einträge der Grobgittermatrizen unserer 3D-Diskretisierungen zu verstehen.
Anhand der Konvergenzresultate scheint der Gauss-Seidel-Glätter für große Zeitschrittweiten besser mit der besonderen Matrix-Struktur zurechtzukommen als der ILU-Glätter. Die Vermutung, die Struktur der Grobgittermatrizen könnte dafür verantwortlich sein, entspringt auch aus der Beobachtung, dass einerseits die Konvergenz erst für größere Zeitschrittweiten, bei denen ja die Matrix $\mathbb{I} - \tau A$ des Gleichungssystems durch die stationäre Diskretisierungsmatrix A dominiert wird, zusammenbricht, andererseits bei Verwendung von mehr als drei Gitterleveln schon bei kleineren Zeitschrittweiten keine Konvergenz mehr erreicht wird.

Transfers
Der Vergleich zwischen den Konvergenzresultaten bei der Benutzung von linearen bzw. matrixabhängigen Transferabbildungen zwischen den einzelnen Gitterleveln zeigt keinen Unterschied. Offenbar ist der Diffusionstensor $D(u)$ als Funktion des Bildes glatt genug, um auch mit linearen Transferoperationen gute Konvergenzraten erzielen zu können. Auch die Filterergebnisse zeigen keinen erkennbaren Unterschied (siehe Abbildung 5.13).

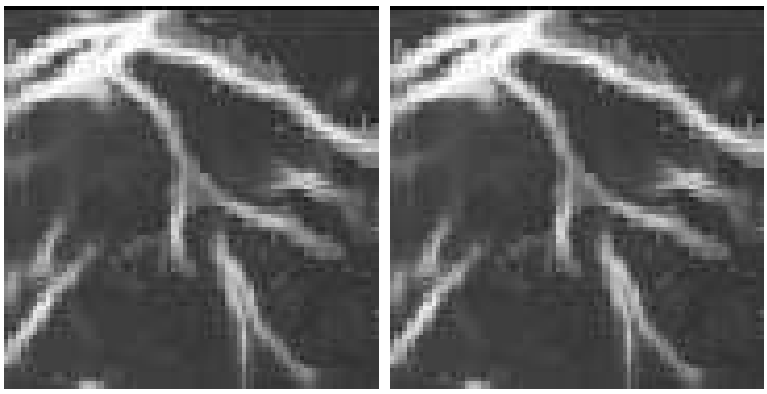

Abbildung 5.13: Verwendung eines 4-Level Mehrgitterverfahrens mit linearen Transferoperationen (links) und matrixabhängigen Transferoperationen (rechts). Das Grobgitter hat nur noch 9 Punkte in jede Raumdimension. Zwischen den Ergebnissen ist kein Unterschied erkennbar.

Bevorzugter Löser
Nach den Resultaten der Tabelle 5.4 sollte man als Löser das BICGStab-Verfahren wählen. Dies erreicht die gewünschte Genauigkeit am schnellsten. Die Konvergenzraten bleiben auch für größere Gitter stabil, wie in der Tabelle 5.6 gezeigt ist.

	$\tau = 0.2$	$\tau = 0.6$	$\tau = 1.0$	$\tau = 3.0$	$\tau = 5.0$	$\tau = 10.0$
Zeit[s]	71	111	131	230	300	408
Iterationen	6	10	12	22	29	40
$\bar{\varrho}$	0.0319	0.1344	0.2148	0.4304	0.5272	0.6201

Tabelle 5.6: Konvergenzverhalten des BiCGStab für ein kubusförmiges Bild mit Kantenlänge $n = 129$

Ausblick: Sollte es wie im vorherigen Abschnitt als wünschenswert angedacht gelingen, die Diffusion stationär zu rechnen, muss die Wahl des Lösers neu überdacht werden. Ein Mehrgitterverfahren kann dann sehr große Vorteile bringen.

5.5 Strukturverbreiterung durch Querdiffusion

Wie wir in Abschnitt 3.4.2 gesehen haben, entsteht quer zur gewünschten Diffusionsrichtung eine numerische Diffusion, die auf die Diskretisierung zurückzuführen ist, und prinzipiell zu einer Verbreiterung der Strukturen führt. Diese ist nicht wünschenswert, vor allem aber für die angestrebte Simulation der Signalausbreitung auf den Neuronen von Nachteil, da die elektrische Signalausbreitung ja durch den Durchmesser der Dendriten bzw. des Axons beeinflusst wird. Somit ist eine realistische Simulation nur möglich, wenn der Durchmesser der Strukturen

bei der Filterung nicht, nur unwesentlich, oder in bekannter und reversibler Form verändert wird.
Die Ergebnisse auf dem Testbild in Abschnitt 5.3 haben jedoch bereits angedeutet, daß die Querdiffusion wegen der gleichzeitigen Rückwärtsdiffusion nur zu einem geringen Kontrastverlust und minimaler Verbreiterung der Strukturen führt.
Dies wollen wir nun anhand eines eindimensionalen Linienplots genauer betrachten.

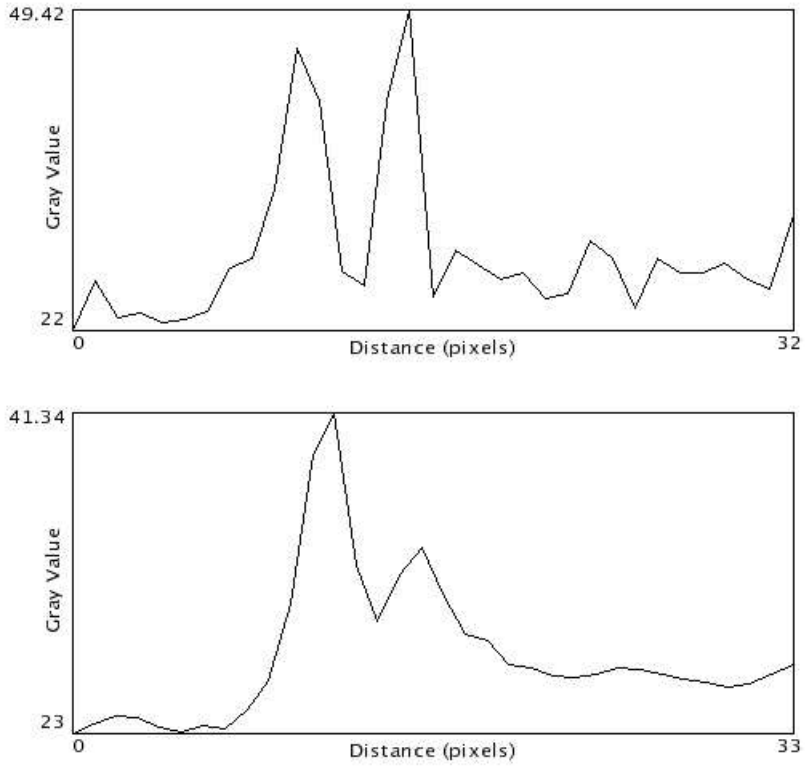

Abbildung 5.14: Linienplots auf einem ungefilterten Bild (oben) und an derselben Stelle auf dem gefilterten Bild (unten)

In Abbildung 5.14 sind zwei eindimensionale Linienplots dargestellt. Diese wurden gewonnen, indem die Bildwerte auf einem durch den Bildkubus gelegten Geradenstück gegen dieses aufgetragen wurden.
Im ungefilterten Bild sind neben dem Hintergrundrauschen zwei markante Spitzen zu sehen, bei denen es sich augenscheinlich um Dendritenstränge handelt. Im gefilterten Bild jedoch ist die eine Spitze durch die Glättung deutlich gedämpft worden. Es handelte sich hierbei also in Wirklichkeit um ein Störsignal. Die Rauschelimination des Filters ist daran sehr gut zu beobachten. Der Dendrit hingegen ist

erhalten geblieben. An diesem ist offenbar nicht nur der Kontrastverlust klein, auch die Breite des Dendriten ist nahezu unverändert.
Dazu sehen wir uns noch einen weiteren Linienplot an, nämlich den in Abbildung 5.15.

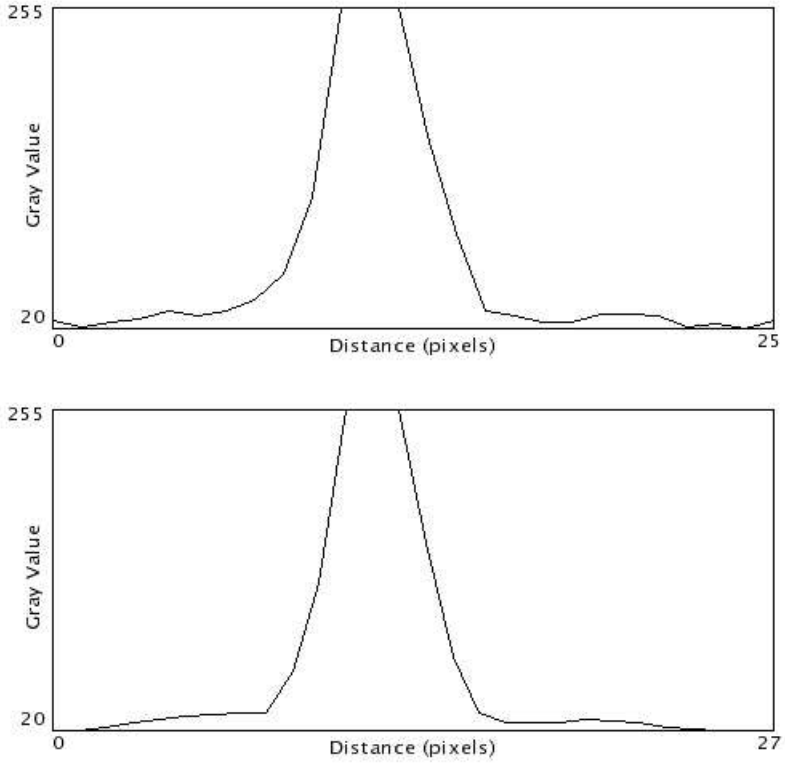

Abbildung 5.15: Linienplots auf einem ungefilterten Bild (oben) und an derselben Stelle auf dem gefilterten Bild (unten)

Augenscheinlich wurde der Durchmesser des Dendriten bei der Filterung praktisch nicht verändert. Lediglich minimale Veränderungen sind zu beobachten. Über die tatsächliche Verbreiterung muss aber letztlich ein festzulegendes Maß entscheiden, welches den Durchmesser der Struktur anhand der Bilddaten quantitativ misst. Das haben wir hier jedoch absichtlich nicht getan, da dies Aufgabe der Rekonstruktion ist. Anhand der hier gezeigten Ergebnisse erscheint es aber praktikabel, nur minimale Veränderungen des Durchmessers der Strukturen bei der Filterung anzunehmen und sie daher als nicht vorhanden zu betrachten.

5.6 Berücksichtigung des Aufnahmerasters

Bei unseren Tests haben wir bislang immer kubus- oder kugelförmige Integrationsgebiete benutzt. Wie in Kapitel 2 erwähnt, wurde das Bild bei der Aufnahme aber nicht in alle drei Raumdimensionen gleich gerastert; die Bildpunkte weisen in z-Richtung einen doppelt so großen Abstand auf wie in die beiden anderen Richtungen. Ein kubusförmiges Integrationsgebiet auf unserem gestauchten Bild entspricht also in Wirklichkeit einem quaderförmigen Integrationsgebiet auf dem natürlichen Original. In Grenzsituationen kann es dadurch zu veränderten Strukturerkennungen kommen.

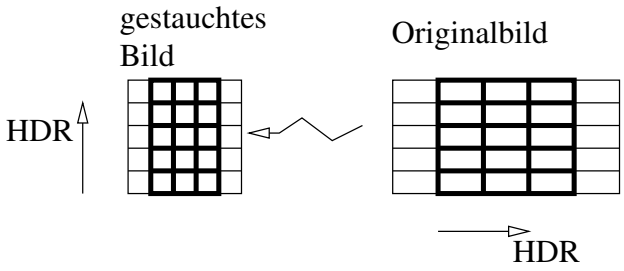

Abbildung 5.16: Veränderung der Strukturerkennung durch Stauchung des Bildes

Will man Integrationsgebiete verwenden, die auf dem Originalbild kubusförmige Gestalt haben, muss man auf unseren gestauchten Bildern quaderförmige Gebiete benutzen, die in z-Richtung nur halb so lang sind wie in die beiden anderen Richtungen.

Abbildung 5.17: Filterergebnisse unter Benutzung eines quaderförmigen Integrationsgebietes mit den Kantenlängen $(16, 16, 8)$

Wir führen die Filterung unter Benutzung eines quaderförmigen Integrationsgebie-

tes mit den Kantenlängen 16 Voxel in x- und y-Richtung und 8 Voxel in z-Richtung durch. Dabei wird wieder nur in Hauptdiffusionsrichtung geglättet.
Das Ergebnis ist nicht besser als jenes, welches wir bei Verwendung eines kubusförmigen Integrationsgebietes erhalten haben. Das konnte man aber auch nicht erwarten. Die Stauchung des Bildes ist nämlich nach wie vor vorhanden und damit auch eine andere Strukturerkennung als auf dem Originalbild. Die Größe des Integrationsgebietes zu verändern reicht nicht aus, das ganze Gitter muss entsprechend des Aufnahmerasters angepasst werden, um die Stauchung rückgängig zu machen. Problematisch an der Verwendung der quaderförmigen Integrationsgebiete ist auch die damit verbundene Bevorzugung einzelner Richtungen, in welche die Hauptdiffusionsrichtung nun viel eher erkannt werden wird. Weiterhin ist es nicht einfach, im Bereich von mittleren Integrationsgebietsgrößen zu bleiben, wenn das Integrationsgebiet in eine Richtung nur halb so lang sein soll wie in die beiden anderen. Das führt automatisch zu schlechterer Strukurerkennung.

Wir raten daher von der Verwendung quaderförmiger Integrationsgebiete ab. Wenn man eine genau zum Originalbild passende Strukturerkennung bekommen möchte, muss man die Voxelgeometrie schon in der Berechnung der Trägheitsmomente berücksichtigen. Allerdings muss man dann auch das anisotrope Gitter bei der Diskretisierung zu Grunde legen, da die Strukturerkennung auf dieses bezogen ist.

Ob ein solches Vorgehen bessere Filterergebnisse hervorbringen kann als die Rechnung auf dem gestauchten Bild, ist fraglich. Für die tatsächliche quantitative Strukturunterscheidung mag es einen Unterschied machen, da die Breite und Dicke von Strukturen auf dem gestauchten Bild anders erkannt wird. Die prinzipielle Einteilung der Strukturen in die drei Fälle linear, planar und isotrop und vor allem die von uns einzig als Filterkriterium benutzte Hauptdiffusionsrichtung wird sich wenn überhaupt nur in wenigen Grenzsituationen ändern.
Ganz davon abgesehen wird sich die Strukturerkennung schon bei einer Veränderung des Aufnahmerasters wieder verändern. Daher ist es eher wünschenswert, ein in alle drei Raumrichtungen gleichmäßiges Aufnahmeraster zu realisieren.

Letztlich rechtfertigen die guten Filterergebisse die Verwendung der gestauchten Bilder.

6 Ausblick auf die Rekonstruktion

Die Filterung der Neuronenaufnahmen geschieht wie schon mehrfach erwähnt nicht, um möglichst „schöne“ Bilder zu erhalten. Das Ziel ist die Rekonstruktion des Neurons und unter diesem Gesichtspunkt ist auch die Filterung zu bewerten.
Daher wollen wir hier zum Abschluss noch die Notwendigkeit des Filters als Vorstufe der Rekonstruktion und seinen Einfluss auf dessen Ergebnisse zeigen.

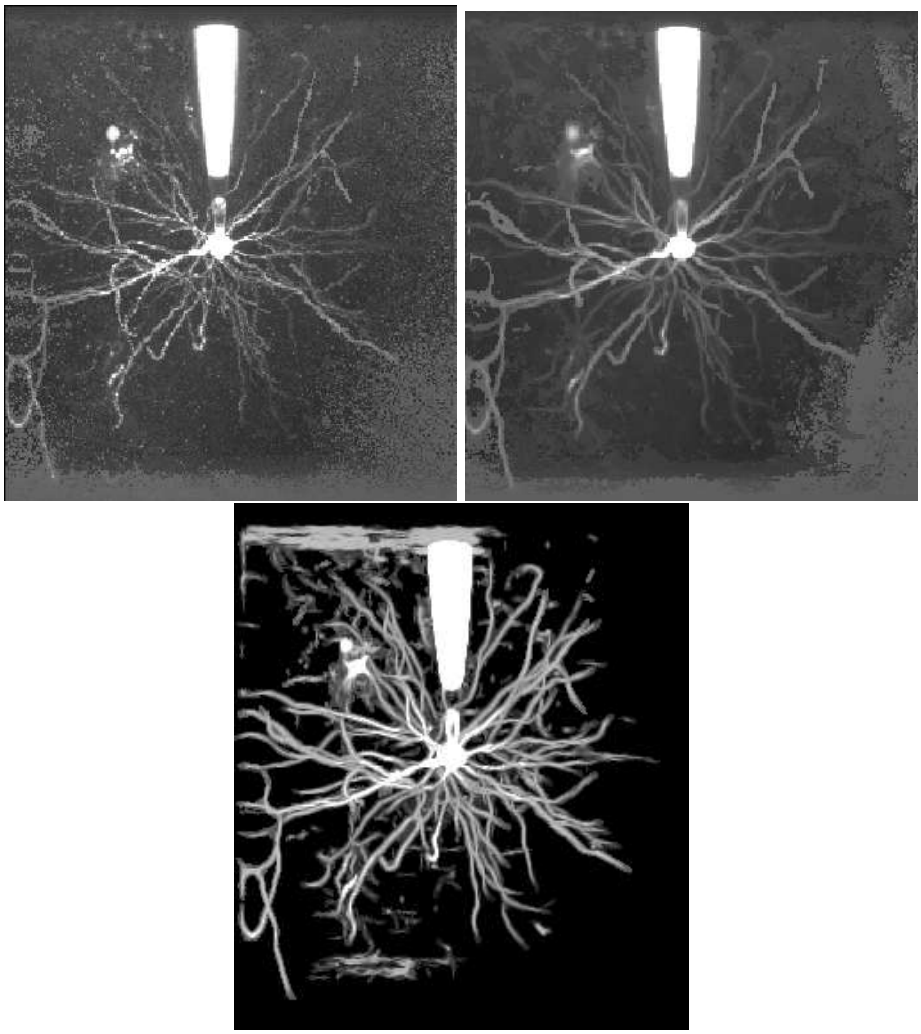

Abbildung 6.1: Ungefilterte (oben links), gefilterte (oben rechts) und nach der Filterung und Segmentierung noch einmal gefilterte Aufnahme (unten)

Wir betrachten die schon bekannte Neuronenaufnahme. Es hat sich bei der Rekonstruktion gezeigt, dass sich eine zweite Filterung des gefilterten und segmentierten

Bildes günstig auf die Rekonstruktionsergebnisse auswirkt. Daher sind in der Abbildung 6.1 die Aufnahme im Originalzustand, nach der Filterung sowie nach einer der Segmentierung nachgeschalteten zweiten Filterung zu sehen.
Deutlich ist die Qualitätsverbesserung durch die Filterung erkennbar. Selbst Strukturen, die im Originalbild mit dem blosen Auge kaum sichtbar waren, werden herausgearbeitet. Auf den Aufnahmne ist noch die Pipette, mit der der Fluoreszenzfarbstoff in die Zelle gebracht wird, sowie Schlieren im oberen und unteren Teil, bei denen es sich um Hirnhautgewebe handelt, vorhanden.

In der Abbildung 6.2 sind nun die Segmentierungen der Bilder aus Abbildung 6.1 gezeigt.

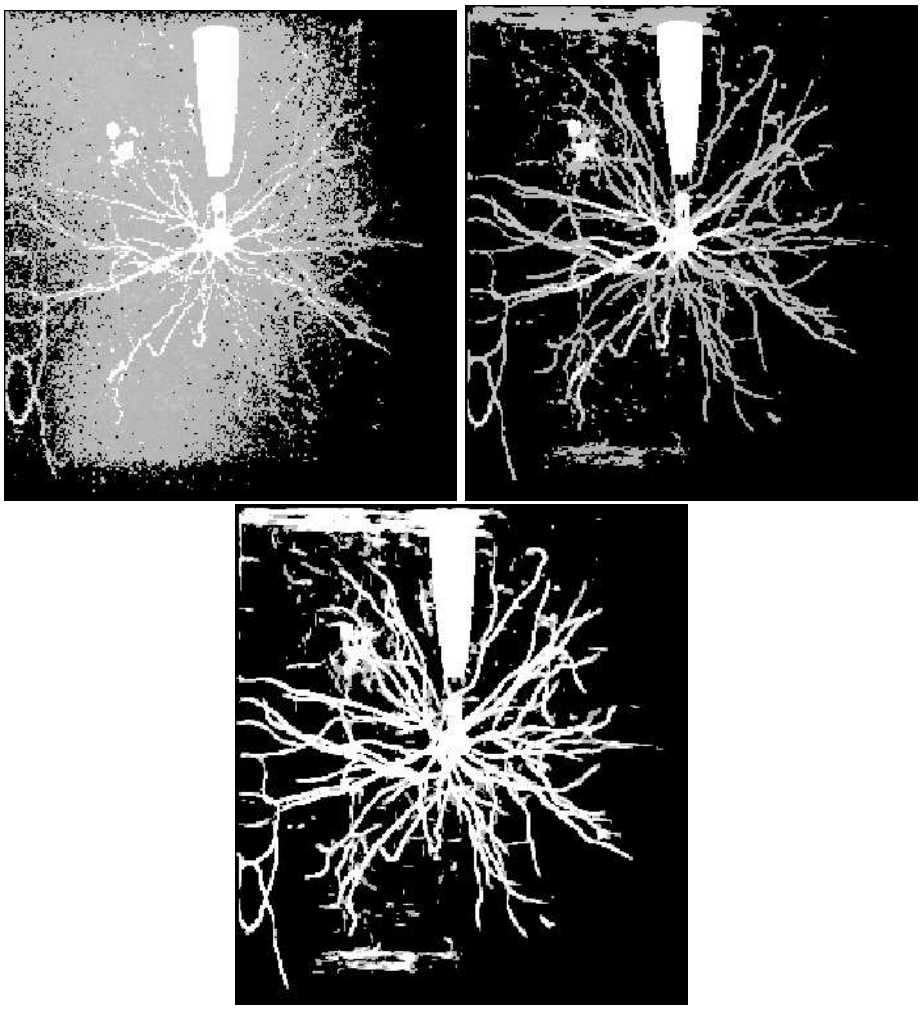

Abbildung 6.2: Segmentierungen der Bilder aus Abbildung 6.1

Zur Segmentierung der Bilder wurden zwei Schwellenwerte angegeben. Liegt der Grauwert eines Voxels über dem größeren Schwellenwert, wird er der Struktur als zugehörig zugeordnet. Diese Punkte sind in den Abbildungen weiß dargestellt. Weiter soll ein Voxel auch dann zur Struktur gehören, wenn sein eigener Grauwert sowie die Grauwerte von mindestens zwei[1] Nachbarvoxeln den kleineren Schwellenwert überschreiten. Diese Punkte sind in den Abbildungen grau dargestellt.
Bereits hier ist erkennbar, dass das ungefilterte Bild keine guten Ergebnisse bei der Rekonstruktion liefern wird, da bei der Segmentierung viel zuviel Struktur erkannt wird.

Auf den segmentierten Bildern wird nun eine von Philip Broser implementierte Rekonstruktion durchgeführt, wobei voher allerdings die Pipette sowie das auf den Bildern sichtbare Hirnhautgewebe manuell entfernt wurde. Die Ergebnisse zeigt Abbildung 6.3. Um übersichtliche Bilder betrachten zu können, werden nur Schnitte gezeigt.

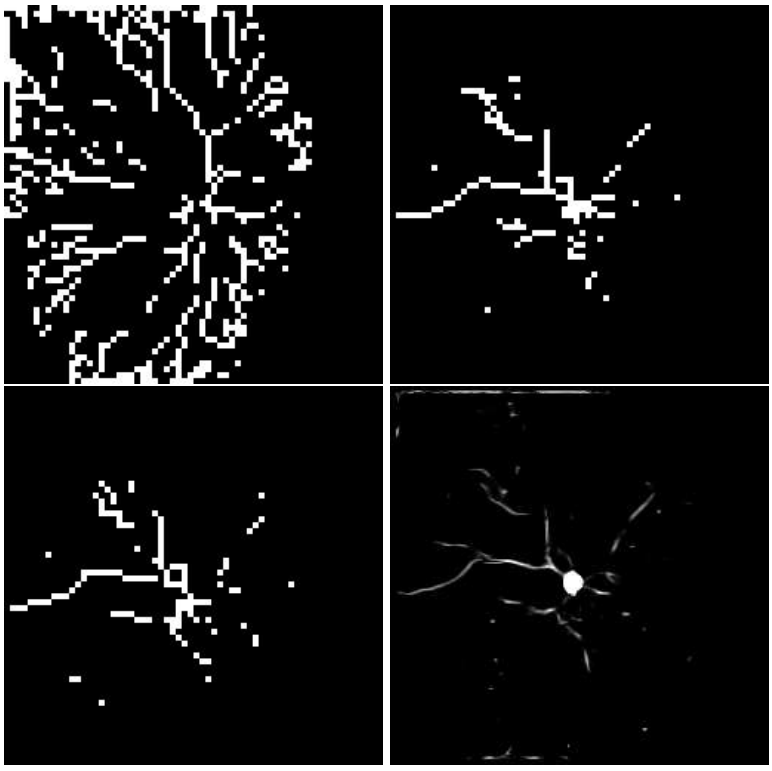

Abbildung 6.3: Schnitt durch die auf der Grundlage der Bilder aus Abbildung 6.2 erstellten Rekonstruktionen. Unten rechts ist zum Vergleich der entsprechende Schnitt durch das zweifach gefilterte Bild gezeigt.

[1] um lineare Strukturen zu erkennen

Wie vorhersehbar, ist das Ergebnis der Rekonstruktion ohne vorherige Filterung unbrauchbar. Nach der Filterung dagegen kommt die rekonstruierte Zelle dem gefilterten Bild schon recht nahe. Wie nahe, kann bei nur augenscheinlicher Betrachtung der Bilder nicht gesagt werden. Dazu muss wiederum die Rekonstruktion selbst ein Kriterium angeben. Dass die zweite Filterung auf dem segmentierten Bild noch eine Verbesserung für das Ergebnis der Rekonstruktion liefert, ist an diesen Bildern nur an den Dedriten im unteren Bildteil zu erkennen.

Abschließende Bemerkungen

Es wurde gezeigt, dass die Filterung der Neuronenaufnahmen durch nichtlineare anisotrope Diffusion sich nicht nur als für die Rekonstruktion unverzichtbare Vorbehandlung erwiesen hat, sondern auch in Verbindung mit der durch die Trägheitsmomente gesteuerten Strukturerkennung sehr gute Ergebnisse liefert.

Leider beansprucht die Filterung als erster Schritt der drei Verarbeitungsschritte Filterung, Segmentierung und eigentliche Rekonstruktion den größten Teil der Rechenzeit, obwohl diese durch die Beschleunigung der Strukturerkennung schon deutlich reduziert worden ist.
Als weiterer Problempunkt erweist sich der hohe Speicheraufwand. Die Filterung von größeren Datensätzen ist daher nicht in einem Stück möglich; es ist nötig, einzelne Teilwürfel aus dem Bild herauszuschneiden, isoliert zu filtern und dann wieder einzusetzen. Da dabei die Voxel in der Nähe des Randes etwas anders behandelt werden, als wenn das Bild als Ganzes gefiltert würde, wird nur der innere Bereich eines Teilwürfels wieder eingesetzt und als gefiltert betrachtet. Auf diese Weise entsteht allerdings ein großer Überlapp zwischen den einzelnen Teilwürfeln, so dass die Rechnung insgesamt länger dauert, als wenn man das Bild als Ganzes filtern würde.

Einen Ausweg könnte die Parallelisierung der Filterung bieten. Diese bietet zum einen die Möglichkeit, eine größere Speicherkapazität zu nutzen und im Idealfall - falls genügend Prozessoren mit entsprechendem Speicher zur Verfügung stehen - das ganze Bild auf einmal zu filtern. Zum anderen würde die Parallelisierung die Rechnung weiter beschleunigen.

Als besonders positiv ist die Erhaltung des Durchmessers der Dendriten bzw. des Axons bei der Filterung hervorzuheben. So wird die für die spätere Simulation der Signalausbreitung auf den Neuronen wichtige Information bewahrt und kann ebenfalls automatisiert ausgewertet werden.

Die in dieser Arbeit verwendete durch Trägheitsmomente gesteuerte nichtlineare anisotrope Diffusion ist nicht auf das vorliegende Problem der Filterung von Neuronenaufnahmen beschränkt. Auch in anderen Fällen, bei denen es um die Herausarbeitung von eindimensionalen Strukturen in verrauschten 2D- oder 3D-Bildern geht, kann sie erfolgreich eingesetzt werden. In Lenzen[7] ist dies bei der 3D-Rekonstruktion von DNA-Strukturen geschehen. Ein anderes Beispiel, bei dem man sich den Einsatz dieses Filters vorstellen könnte, sind Aufnahmen der Spuren von Elementarteilchen in einem Teilchenbeschleuniger.

Wie sich die Simulation der Signalleitung auf der realen Neuronengeometrie gestalten und welche Ergebnisse sie für die medizinische Forschung haben wird, ist zum jetzigen Zeitpunkt kaum absehbar. Mit der automatischen Rekonstruktion der Zelle aus mikroskopischen Daten, die der in dieser Arbeit implementierte Filter ermöglicht, wird jedoch ein wichtiger vorbereitender Schritt dazu getan sein.

Literatur

[1] Peter Bastian. *Paralleles Rechnen*. Vorlesungsskriptum, Universität Heidelberg, 2001.

[2] Neil A. Campbell. *Biologie*. Spektrum, Akademischer Verlag, 2000.

[3] Otto Forster. *Analysis 3*. Vieweg, 1999.

[4] Matthias Fricke. *Parallel-Processing in der Zwei-Photonen-Laser-Raster-Mikroskopie*. Diplomarbeit, Universität Bielefeld, 2000.

[5] Ch.Grossmann/H.-G.Roos. *Numerik partieller Differentialgleichungen*. Teubner, Stuttgart, 1994.

[6] W. Hackbusch. *Iterative Lösung großer schwachbesetzter Gleichungssysteme*. Teubner, Stuttgart, 1993.

[7] Frank Lenzen. *3D-Rekonstruktion von DNA-Strukturen*. Diplomarbeit, Universität Bonn, 2001.

[8] Stephen F. McCormick. *Multigrid Methods*. SIAM, 1987.

[9] Pavel Mrázek. *Nonlinear Diffusion for Image Filtering and Monotonicity Enhancement*. PhD Thesis, Czech Technical University, 2001.

[10] Frank Natterer. *Numerische Mathematik*. Vorlesungsskriptum, Universität Münster, 1996.

[11] *Numerical Recipes in C: The art of scientific computing*. Cambridge University Press, 1988-1992.

[12] H.R.Schwarz. *Numerische Mathematik*. Teubner, Stuttgart, 1997.

[13] H.R.Schwarz. *Methode der finiten Elemente*. Teubner, Stuttgart, 1991.

[14] Walter A. Strauss. *Partielle Differentialgleichungen*. Vieweg, 1995.

[15] Christian Wagner. *Ein algebraisches Multilevelverfahren - Entwicklung und Anwendung auf einen Sanierungsfall*. Habilitationsschrift, Universität Heidelberg, 2000.

[16] Gabriel Wittum. *Mehrgitterverfahren*. Vorlesungsskriptum, Universität Heidelberg, 2000.

[17] Joachim Weikert. *Anisotropic Diffusion in Image Processing*. Teubner, Stuttgart, 1997.

Danksagung

Ich danke Prof. Dr. Gabriel Wittum für die Vergabe dieser interessanten Diplomarbeit. Auch gewann ich durch ihn in einer der schwierigen Phasen des Studiums den Mut zurück, dieses erfolgreich abschließen zu können.
Meinen Eltern möchte ich für die stetige finanzielle, aber vor allem auch für die moralische Unterstützung danken, auf die ich mich jederzeit verlassen konnte.
Philip Broser danke ich für die gute Zusammenarbeit und die Betreuung bei der Diplomarbeit.
Christoph Reisinger danke ich für die Bereitstellung seines Mehrgitterlösers.
Prof. Dr. Martin Rumpf, Tobias Preusser und Marc Droske danke ich für den zweitägigen Aufenthalt am Institut für Angewandte Mathematik der Universität Duisburg.

Besonders danke ich meiner Freundin Julia dafür, dass ich bei ihr immer wieder die nötige Ablenkung und neue Energie fand.