UNIVERSITÄT DORTMUND
FACHBEREICH INFORMATIK

DIPLOMARBEIT zum Thema

Entwicklung eines virtuellen klinischen Kernspintomographen

Christian Schalla, Andreas Trümper

I N T E R N E B E R I C H T E
I N T E R N A L R E P O R T S

Lehrstuhl Informatik 1
Otto-Hahn-Straße 16
44227 Dortmund

Vorwort

Im Mittelpunkt der vorliegenden Diplomarbeit steht ein System, das dazu dient, die Bilderzeugung und –weiterverarbeitung eines klinischen Kernspintomographen zu simulieren.

Die Idee zu dieser Diplomarbeit stammt von PD Dr. med. Dipl.-Phys. Thomas Hackländer vom Klinikum Wuppertal GmbH, mit dem wir schon im Rahmen unserer Projektgruppe ANOMALIA zusammengearbeitet haben. Während sich ANOMALIA mit der Erkennung von pathologischen Strukturen in MRT-Bildern beschäftigte, setzt unser System schon bei der Bilderzeugung an.

Diese Idee entsprang aus dem Wunsch, angehenden Medizinern einen guten Zugang zur recht komplexen Materie eines Kernspintomographen anzubieten. Da nicht jeder Medizinstudent Zugang zu einem klinischen Kernspintomographen hat, war es Ziel dieser Arbeit, Eigenschaften und Verhalten eines solchen Gerätes durch ein Computersystem zu simulieren.

An dieser Stelle möchten wir uns bei unseren Betreuern Prof. Dr. Bernd Reusch und PD Dr. med. Dipl.-Phys. Thomas Hackländern für ihre freundliche Unterstützung danken. Mit ihrer Geduld und Zuversicht haben sie uns erfolgreich ans Ziel geführt. PD Dr. med. Dipl.-Phys. Thomas Hackländer stand uns in zahlreichen Gesprächen zur Wissensakquisition und zur Diskussion und Umsetzung des Konzeptes zur Verfügung.

Darüber hinaus danken wir Dipl.-Inform. Jens Hiltner, der uns mit Rat und Tat zur Seite stand und zahlreiche Hinweise zur Korrektur des schriftlichen Teils unserer Diplomarbeit gab.

Außerdem danken wir Dr. Mertens vom Klinikum Wuppertal GmbH für seine Unterstützung bei der Literatursuche und für seine zahlreichen Anregungen für unsere Diplomarbeit. Frau Uehlendahl und Dr. Kühl, ebenfalls vom Klinikum Wuppertal GmbH, danken wir für ihre Hilfe bei der Bedienung des Kernspintomographen und der Durchführung einiger Probemessungen.

Dr. Kunitz von der Universität Jena möchten wir für seine Anregungen zur Berechnung der Ausgangsdatensätze danken, ebenso wie Prof. Dr. Heinrich Müller für seine Unterstützung auf dem Gebiet der Fouriertransformation.

Das im Rahmen dieser Diplomarbeit entwickelte Softwareprodukt wurde zum Abgabezeitpunkt der Diplomarbeit am Lehrstuhl Informatik 1 hinterlegt, ist aber auch auf der dieser Diplomarbeit beiliegenden CD enthalten.

Diese Arbeit haben wir zusammen erstellt. Dennoch lassen sich die Kapitel wie folgt den Autoren zuweisen.

Andreas Trümper: 3.3, 3.6, 4.3.1, 4.3.2, 5.1.2, 5.2, 5.3, 7.2.1, 7.2.3

Christian Schalla: 2, 3.1, 3.2, 3.4, 3.5, 4.1, 4.2, 4.3.3, 5.1.1, 5.1.3, 5.4, 5.5, 6, 7.1, 7.2.2

Kapitel 1, 3.5.3, 8 und der Anhang wurden gemeinsam erstellt.

Dortmund, im Oktober 1999

Inhalt

1 Einleitung 5
1.1 Problembeschreibung 5
1.2 Aufgabenstellung 5
1.3 Mögliche Anwendungen 5
1.4 Aufbau der Arbeit 6
2 Bildgebende Verfahren in der Medizin 7
3 Grundlagen und Technik der Kernspintomographie 14
3.1 Mikroskopische und makroskopische Magnetisierung, Kernspin, Suszeptibilität und Präzession 14
3.2 Kernspinanregung und Kernspinresonanz 18
3.3 Pulssequenzen 23
3.3.1 Konventionelle Techniken 25
3.3.2 Schnellbildtechniken Teil I (Gradientenechotechniken) 33
3.4 Ortskodierung 41
3.4.1 Fourieranalyse 43
3.4.2 Punkttechnik 44
3.4.3 Linientechnik 45
3.4.4 Schichttechnik 48
3.4.5 Volumentechnik 50
3.4.6 Vielschichttechnik 51
3.5 k-Raum 52
3.5.1 Spin-Echo-Technik 57
3.5.2 Halb-Fourier- und Halb-Echo-Technik 60
3.5.3 Schnellbildtechniken Teil II (Turbo-Spin-Echo) 61
3.6 Artefakte und Störungen der Bilderzeugung 64
4 Konzept des virtuellen MRT 67
4.1 Beschreibung des realen Arbeitsplatzes 67
4.2 Prinzipien des GUI-Entwurfs 70
4.3 Komponenten des Systems 74
4.3.1 Konzept zur Berechnung der Parameterbilder 74
4.3.2 Konzept des Meßwerkzeuges *Virtual MRT* 78
4.3.3 Konzept des Nachbearbeitungswerkzeuges *DICOM-Viewer* 81
5 Umsetzung 85
5.1 Allgemeine Informationen zur Implementierung 85
5.1.1 Wahl der Programmiersprache 85
5.1.2 Arbeitsmittel und Arbeitsumgebung 86
5.1.3 Quelltextkonventionen und Quelltextkommentierung 86
5.2 Berechnung der Parameterbilder mit IMAGEJ 88
5.2.1 *ImagePlus* und *ImageStack* 88
5.2.2 *Plugin* und *PluginFilter* 89
5.2.3 Berechnung der Parameterbilder 90
5.2.4 Nachbearbeitung der Parameterbilder 91
5.3 Der virtuelle Tomograph 92
5.3.1 Das Paket *artefacts* 95
5.3.2 Das Paket *dcm* 95
5.3.3 Die Pakete *fft* und *jigl* 99
5.3.4 Das Paket *sequences* 100
5.3.5 Das Paket *tools* 102
5.3.6 Das Paket *vmrt* 102
5.3.7 Simulation von Artefakten 104
5.4 Das Nachbearbeitungswerkzeug *DICOM-Viewer* 106
5.5 Erweiterungsmöglichkeiten 108
5.5.1 Hinzufügen neuer Pulssequenzen 108
5.5.2 Hinzfügen neuer Artefakt-Simulatoren 114
6 Evaluierung des Projektes 119
6.1 Vergleich mit anderen Systemen 119
6.1.1 MRI IMAGE EXPERT 119
6.2 Das Referenzbild (Kontraststudie) 121
7 Benutzerhandbuch 126
7.1 Installation 126
7.1.1 Hard- und Softwarevoraussetzungen 126

7.1.2 Installation und Programmstart ... 126
7.2 Bedienungsanleitung ... 127
7.2.1 Bedienung des *Virtual MRT* ... 127
7.2.2 Bedienung des *DICOM-Viewers* ... 136
7.2.3 Bedienung der Parameterberechnung und des Zeichenwerkzeuges ... 150
8 Zusammenfassung und Ausblick ... 156
Anhang A: Klassendokumentation ... 157
Klasse AnimationFrame ... 158
Klasse Artefact ... 161
Klasse ArtefactsCombo ... 162
Klasse ArtefactUI ... 163
Klasse DicomViewer ... 164
Klasse FFTTools ... 165
Klasse FileLoader ... 167
Klasse ImagePlus ... 169
Klasse ImageStack ... 175
Klasse KSpaceFrame ... 177
Klasse KSpaceManipulator ... 178
Klasse Motion ... 180
Klasse MotionUI ... 182
Klasse MyPanel (*DICOM-Viewer*) ... 182
Klasse MyPanel (*Virtual MRT*) ... 185
Klasse Pulsesequence ... 187
Klasse PulsesequenceUI ... 189
Klasse SaturationRecovery ... 193
Klasse SaturationRecoveryUI ... 194
Klasse SequenceCombo ... 195
Klasse SeriesLoader ... 196
Klasse SliderPanel ... 196
Klasse ViewerFrame ... 199
Klasse VMRT ... 208
Klasse VMRTFrame ... 208
Anhang B: Literaturverzeichnis ... 216
Anhang C: Abbildungs- und Tabellenverzeichnis ... 219
Anhang E: Index ... 222

1 Einleitung

1.1 Problembeschreibung

Ein klinischer Kernspintomograph dient dazu, krankhafte Veränderungen des Organismus mit einer optimalen Ortsauflösung bei einem optimierten Kontrast zwischen gesundem und krankhaften Gewebe darzustellen. Die Signalintensität der verschiedenen Gewebe wird durch vier physikalische Größen festgelegt: Anzahl der Wasserstoffatome im Abbildungsvolumen (Protonendichte), die Relaxationszeit T_1, die Relaxationszeit T_2 und die Suszeptibilität. Der Beitrag der verschiedenen physikalischen Größen an der Gesamtintensität kann durch Einstellungen von Geräteparametern variiert werden. Diese Einstellungen und deren zeitliche Abfolge wird als Meßsequenz bezeichnet. Für jede Meßsequenz ist ein funktionaler Zusammenhang zwischen Eingangsgrößen und zu erwartender Signalintensität bekannt. Heutzutage sind eine Anzahl verschiedener Meßsequenz-Klassen verfügbar: Saturation-Recovery, Inversion-Recovery, Spin-Echo, Turbo-Spin-Echo und Gradientenecho. Jede dieser Klassen unterteilt sich in Untergruppen, wobei diese jeweils zu unterschiedlichen Kontrastverhältnissen zwischen den Geweben führen.

In der klinischen Anwendung werden solche Messungen ortsaufgelöst durchgeführt. Dabei ist das reale Bildergebnis immer schlechter, als das nach dem funktionalen Zusammenhang zu erwartende. Hierfür gibt es eine Reihe von Gründen: Unzulänglichkeiten der Meßapparatur (z.B. Rauschen), Einfluß der Meßsequenz (z.B. verschlechtert eine Verringerung der Schichtdicke das Signal-zu-Rauschverhältnis, bei lang andauernden Meßsequenzen fällt die Signalintensität ab), physikalische Effekte (z.B. Verzerrungen des lokalen Magnetfeldes durch Luft-Gewebs-Grenzflächen), Einfluß des untersuchten Objektes (z.B. biologische Bewegungen, Pulsationen) und schließlich Nachverarbeitungsfehler der verwendeten Bildrekonstruktionsalgorithmen. Hierdurch ist es vielfach schwierig, das Optimum der Untersuchungsmethode im Vorhinein rein theoretisch – nach Fachbuchwissen – abzuschätzen.

1.2 Aufgabenstellung

Es soll ein Programm entwickelt werden, dessen Benutzeroberfläche sich an realen Kernspintomographen orientiert. Um eine maximale Portabilität zu gewährleisten, soll die Software in JAVA programmiert werden. Der funktionale Zusammenhang der Meßsequenz kann als bekannt angenommen werden. Damit sollte jeweils mindestens eine Untergruppe jeder Meßsequenzklasse implementiert werden. Die Simulation muß sich dabei auf Parameterbilder der vier physikalischen Bildparameter stützen. Diese lassen sich angenähert durch spezielle reale Messungen gewinnen. Es sollte jedoch ein Werkzeug zur Korrektur bzw. Retusche der realen Messungen vorhanden sein. Hierdurch ließen sich auch Erkrankungen didaktisch herausarbeiten. In einem realen Kernspintomographen erfolgt die Datenaufnahme in einem abstrakten zweidimensionalen Raum (k-Raum). Durch eine inverse Fouriertransformation entsteht das sichtbare Bild. Die genannten Verschlechterungen der Bilder können entweder im k-Raum oder bei der Transformation simuliert werden. Die beschriebenen Simulationen sollen sich der Einfachheit halber jeweils nur auf eine Schichtebene auswirken. Eine Erweiterung auf Schichtstapel wäre erst in späteren Versionen denkbar. In der Realität werden fast immer Schichtstapel oder 3D-Volumina gemessen und dann sequentiell ausgedruckt, in einer anderen Ebene geschnitten und nach dem Maximalen-Intensitäts-Projektions-Algorithmus (MIP) nachverarbeitet. Diese grundlegenden Verfahren sollten an real gemessenen Bildstapeln zumindest ansatzweise, z.B. beschränkt auf 90° Projektionen, implementiert werden.

1.3 Mögliche Anwendungen

Das Programm wendet sich an Studenten in der klinischen Ausbildung und Ärzte in der Weiterbildung. Der Anwender soll in die Lage versetzt werden, sich interaktiv mit der komplexen

Materie auseinanderzusetzen. Ohne daß ein direkter Zugang zu einem realen Gerät notwendig wäre, sollen im Sinne eines "was-wäre-wenn" Szenarios Geräteeinstellungen ausgetestet und der speziellen Pathologie angepaßt werden können. Damit können Studenten und Nicht-Radiologen die Funktions- und Arbeitsweise eines kostspieligen und nicht an jedem Krankenhaus verfügbaren Gerätes erfassen und nachvollziehen.

1.4 Aufbau der Arbeit

Kapitel 2 dieser Arbeit gibt zunächst eine Übersicht über die wichtigsten bildgebenden Verfahren in der Medizin. Wir beschreiben dort kurz das Röntgen, die Computertomographie, die Sonographie und die Magnetresonanztomographie.

Da sich unsere Simulation mit der Magnetresonanztomographie befaßt, wird diese in Kapitel 3 ausführlich beschrieben. Nach Erklärung der zu Grunde liegenden Physik werden einige Pulssequenzen vorgestellt und die Bilderzeugung mittels Ortskodierung erklärt. Die im Kapitel Pulssequenzen vorgestellten Formeln sind wesentlich für die Berechnung, die in der Simulation durchgeführt werden. Anschließend wird der k-Raum erklärt. Das Verständnis des k-Raumes ist unter anderem wichtig, um die Entstehung einiger Artefakte zu verstehen. Einige dieser Artefakte, die auch in unserem Computersystem implementiert sind, werden dort kurz beschrieben.

In Kapitel 4 wird ein Konzept zur Simulation eines realen Magnetresonanztomographen vorgestellt. Dabei wird zunächst eine realer Arbeitsplatz beschrieben und anhand dessen die Komponenten der Simulation entwickelt. Dies sind zum einen ein Meßwerkzeug zur Bilderzeugung und zum anderen ein Nachbearbeitungswerkzeug. Zusätzlich benötigen wir ein Werkzeug, das die Berechnung von Parameterbildern aus speziellen Bildserien ermöglicht, auf die das Meßwerkzeug aufbaut.

Kapitel 5 befaßt sich dann mit der Umsetzung des Konzeptes in ein Softwareprodukt. Nach allgemeinen Informationen zur Implementierung wird die Umsetzung der einzelnen Systemkomponenten in die Programmiersprache JAVA beschrieben. Zum Schluß des Kapitels wird auf Erweiterungsmöglichkeiten bezüglich Pulssequenzen und Artefakt-Simulatoren eingegangen.

In Kapitel 6 vergleichen wir unser System mit einem anderen Programm, das eine ähnliche Zielsetzung verfolgt. Darüber hinaus stellen wir einige wichtige Eigenschaften unseres Systems vor und bewerten deren Qualität.

Kapitel 7 beschreibt die Installation und Bedienung der einzelnen Komponenten des Systems.

Abschließend faßt Kaptel 8 die erreichten Ziele der Arbeit zusammen, beschreibt Probleme, die aufgetreten sind und bietet einen Ausblick auf denkbare Erweiterungen.

2 Bildgebende Verfahren in der Medizin

Bildgebende Verfahren sind aus der modernen Medizin nicht mehr wegzudenken. Sie sind zumeist aus der Verschmelzung von medizinischem und technischem Wissen entstanden. Während der Informatiker und/oder Ingenieur sich bei der Einteilung und Einordnung an den dahinter liegenden physikalischen Prinzipien orientiert, stehen für den Mediziner ganz andere Gesichtspunkte im Vordergrund [LEHM97], auf die wir hier allerdings kaum eingehen können.

Aus technischer Sich lassen sich vier bildgebende Verfahren unterscheiden, die heute in der Routinemedizin verstärkt eingesetzt werden. Das sind das klassische Röntgen, die Computertomographie (CT), die Magnetresonanztomographie (MRT) oder Kernspintomographie (KST) und die Sonographie (US = Ultraschall). Darüber hinaus gibt es noch viele weitere Formen der medizinischen Bildgebung, die hier aber nicht erläutert werden sollen.

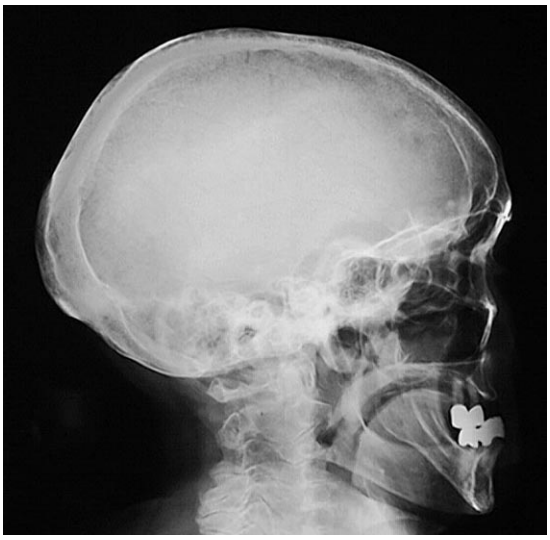

Abbildung 1: Röntgenaufnahme eines Schädels

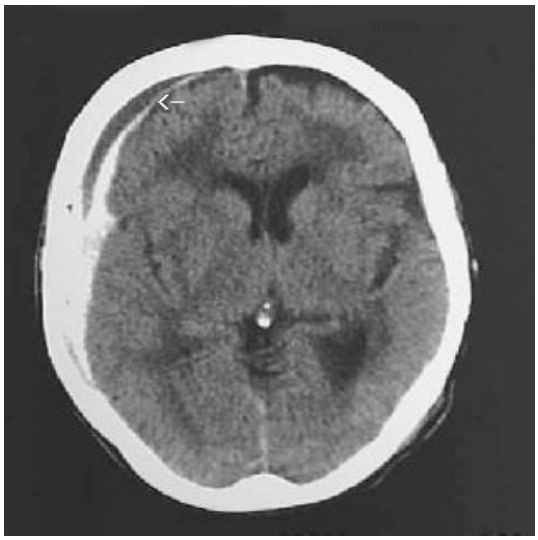

Abbildung 2: CT-Aufnahme eines Schädels

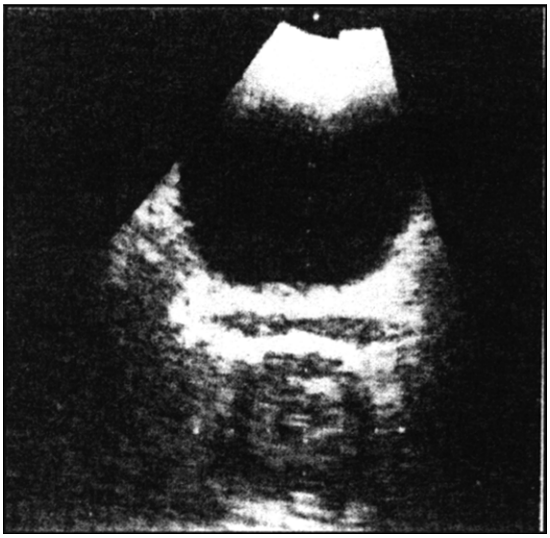

Abbildung 3: Ultraschallaufnahme einer Harnblase [LEHM97]

Abbildung 4: MRT-Aufnahme eines Schädels

Die verschiedenen Verfahren eigenen sich sehr unterschiedlich zur Darstellung verschiedener Gewebe und Funktionen. Das klassische Röntgen ist ein sehr preiswertes Verfahren und eignet sich insbesondere zur Darstellung von Knochen. Es mißt allerdings immer ein Volumen.

In diesem Punkt unterscheiden sich die Tomographieverfahren CT und MRT wesentlich davon. Unter Tomographie versteht man im allgemeinen die Abbildung einer zweidimensionalen Schicht eines dreidimensionalen Objektes, also ein Schnittbildverfahren. Die Computertomographie eignet sich, da sie auch auf Röntgenstrahlen basiert, ebenfalls zur Darstellung von Knochen. Da eine abzubildende Schicht aus einem Volumen ausgewählt werden kann, können mit der CT auch Flächen und Volumen sehr genau gemessen werden. Letzteres gilt auch für die Magnetresonanztomographie. Aufgrund einer völlig anderen Funktionsweise weisen MRT-Bilder einen sehr hohen Weichteilkontrast auf.

Das Ultraschallverfahren eignet sich aufgrund seiner Echtzeitfähigkeit insbesondere zur Darstellung von Organfunktionen. Es ist ein sehr preiswertes und überall verfügbares Verfahren, das allerdings keine hohe Detaildarstellung erlaubt und ebenso wenig exakt reproduzierbar ist. Tabelle 1 faßt die Darstellungsmöglichkeiten der 4 wichtigsten bildgebenden Verfahren zusammen.

	Röntgen	CT	MRT	US
Knochen	+++	+++	+	-
Weichteile	-/+	-	++	+
Gefäße	++	++	++	+
Funktionen	-	-	++	++
Volumina	-	++	++	+

Tabelle 1: Darstellungsmöglichkeiten der 4 wichtigsten bildgebenden Verfahren in der Medizin [LEHM97]

Das klassische **Röntgen** basiert darauf, daß die von einer Röntgenröhre ausgesendete Röntgenstrahlung das abzubildende Objekt durchdringt und dabei je nach Material bzw. Gewebeart unterschiedlich geschwächt wird.

Röntgenstrahlen entstehen, wenn energiereiche Elektronen auf ein Hindernis prallen oder durch sonstige Einflüsse gebremst werden. In der Röntgenröhre werden durch eine Glühkathode Elektronen freigesetzt, die durch eine Hochspannung zur Anode hin beschleunigt werden. Dort treffen sie auf ein sogenanntes Targetmaterial[1], wodurch ein Röntgenspektrum erzeugt wird (Abbildung 5). Es handelt sich dabei um Strahlen mit einer Wellenlänge im Bereich von 10^{-5} nm bis 10 nm [LAUB90].

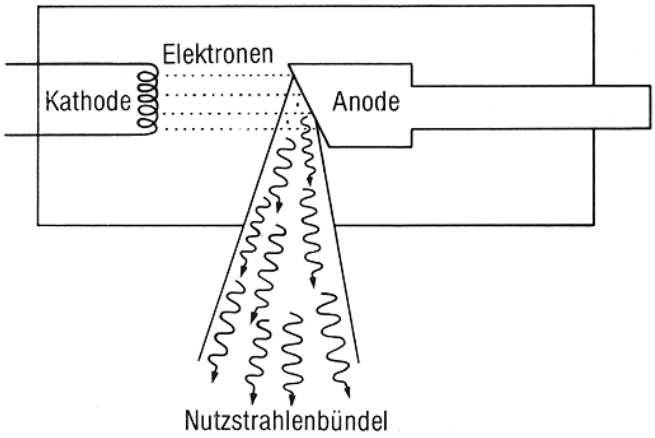

Abbildung 5: Schematische Röntgenröhre [LAUB90]

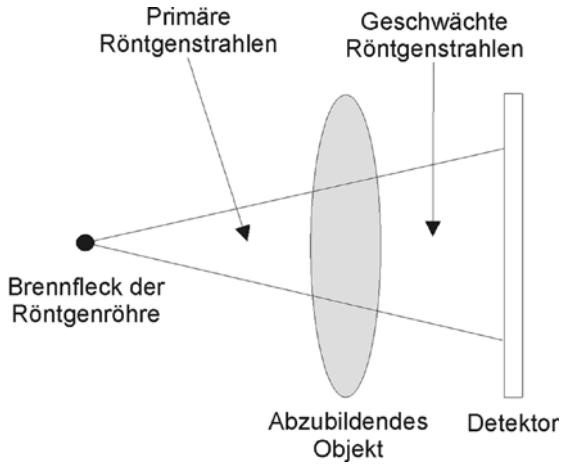

Abbildung 6: Prinzip des Röntgens

Hinter dem durchstrahlten Objekt wird die Intensität der geschwächten Strahlung von Detektoren gemessen (Abbildung 6). Als Detektoren kommen primär Filme, bei der sogenannten Durchleuchtung aber auch Photomultiplier oder digitale Systeme zum Einsatz. Die Filme werden durch das Auftreffen von Röntgenstrahlung geschwärzt. Um die Röntgendosis zu senken verwendet man fluoreszierende

[1] Als Targetmaterial wird in der Regel Wolfram verwendet, da es eine hohe Ordnungszahl (74), einen hohen Schmelzpunkt (3370°C) und eine sehr gute Wärmeleitfähigkeit hat. Letztere ist nötig, da 99% der Energie, die beim Elektronenbeschuß entsteht, in Wärme umgewandelt wird.

Verstärkerfolien, mit denen die Röntgenstrahlen in sichtbares Licht umgewandelt werden. Dieses ist dann zu ca. 95% für die Filmbelichtung verantwortlich ist. Die Röntgenstrahlen selbst tragen dann nur noch ca. 5% zur Belichtung bei. Die Verstärkerfolien liegen dem Film direkt auf [LAUB90], [LEHM97].

In Abhängigkeit von der Röhrenspannung U unterscheidet man harte (U > 100 kV) und weiche (U < 100 kV) Röntgenstrahlung. Je kurzwelliger und energiereicher die Strahlung ist, desto härter ist sie. Weiche Strahlung eignet sich besser zur Darstellung von Knochen, harte Strahlung zur Darstellung von Weichteilen. Bei der Wechselwirkung zwischen weicher Strahlung und Gewebe überwiegt die für den Körper gefährliche Absorption gegenüber der Streuung. Als Abschirmungsmaterial für harte Strahlung kann Blei, für weiche Strahlung Aluminium, Kupfer oder Molybdän verwendet werden.

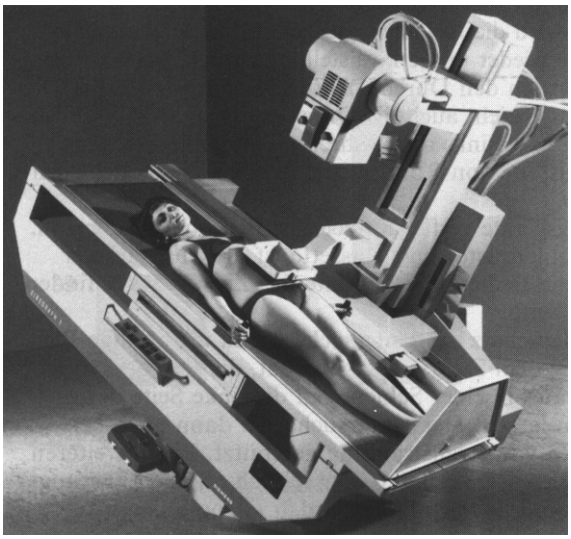

Abbildung 7: Röntgentisch [LAUB90]

Abbildung 8: Computertomograph [LAUB90]

Die rekonstruktive[2] **Computertomographie** ist eine Weiterentwicklung des klassischen Röntgenverfahrens zur Erstellung von Transversaltomogrammen[3]. Der wesentliche Unterschied zum klassischen Röntgen ist der, daß nicht das komplette Volumen des Untersuchungsobjekts im Bild dargestellt wird, sondern nur eine einzige Schicht mit einer endlichen Ausdehnung (Abbildung 9). Somit kommt es nicht zur Überlagerung räumlich getrennter Strukturen. Es können allerdings primär nur senkrecht zur Körperachse liegende Schichten (± 10 Grad) aufgenommen werden. Andere Schichtführungen müssen rechnerisch rekonstruiert werden.

Um ein zweidimensionales Bild zu erhalten, werden einzelne Projektionen einer Objektebene aufgenommen und anschließend mit Hilfe eines Computers zu einer zweidimensionalen Darstellung der Schwächungswerte der ausgewählten Schicht rekonstruiert. Dem zufolge werden die Bilder zunächst auf einem Monitor dargestellt, können jedoch trotzdem auf einen Film übertragen werden.

Beim historischen Computertomographen wurde mit einem ca. 5 mm dicken Röntgenstrahl die ausgewählte Objektebene abgetastet. Danach wurde der Röntgenstrahl um 1° gedreht und die Schicht erneut komplett abgetastet. So verfuhr man in 180 1°-Schritten (Abbildung 10). Aus diesen 180 Intensitätsprofilen rekonstruierte der Computer entweder algebraisch oder mit Hilfe von Fouriertransformationen (Kapitel 3.4.1) das zweidimensionale Schwächungsprofil der selektierten Schicht. Die maximale Auflösung der Bilder war sehr stark durch die Dicke des Röntgenstrahls (5 mm) bestimmt, denn innerhalb eines Elementarquaders kann das Schwächungsverhalten der Röntgenstrahlen nicht weiter differenziert werden [LEHM97].

[2] Das Verfahren wird rekonstruktiv genannt, da das Bild aus vielen verschiedenen Projektionen rekonstruiert wird. Dazu später mehr.
[3] Transversal = axial = zur Körperachse senkrecht.

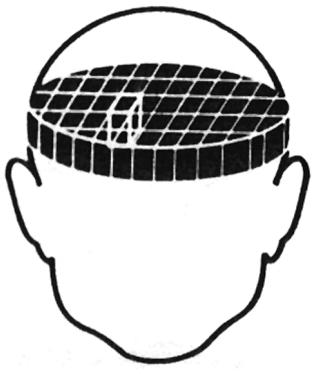

Abbildung 9: Schichteselektion [LEHM97]

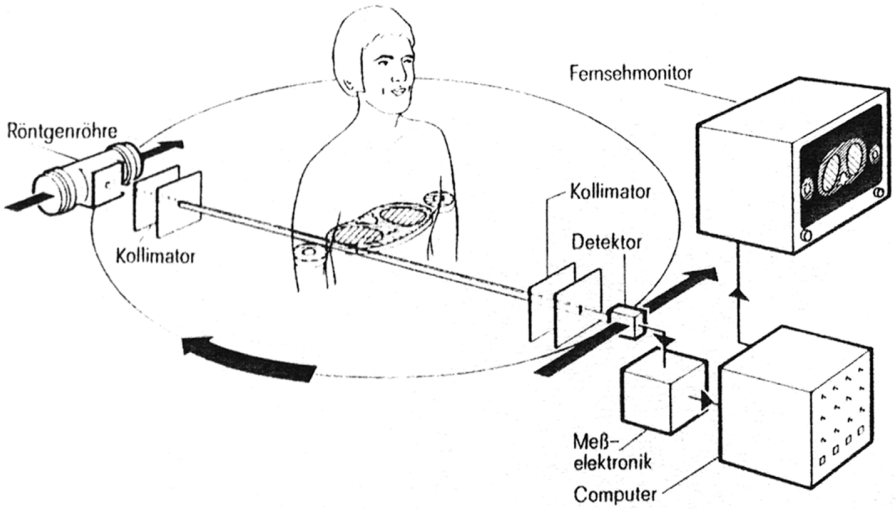

Abbildung 10: Historische Computertomographie [LEHM97]

Mit modernen Computertomographen (Abbildung 11) sind kurze Aufnahmezeiten durch drastisch reduzierten mechanischen Aufwand möglich. Mit einem Röntgenstrahlfächer wird eine große Fläche der Schichtebene simultan erfaßt und das Gerät vollzieht nur noch eine rotatorische Bewegung um den Patienten. Die Aufnahme eines Schichtbildes ist damit in ca. 1 Sekunde möglich [LAUB90]. Nach der Aufnahme eines Schichtbildes wird der Patient automatisch mit dem Patientenlagerungstisch entlang der Längsrichtung weiter in den Tomographen geschoben, so daß ein weiteres Schichtbild aufgenommen werden kann.

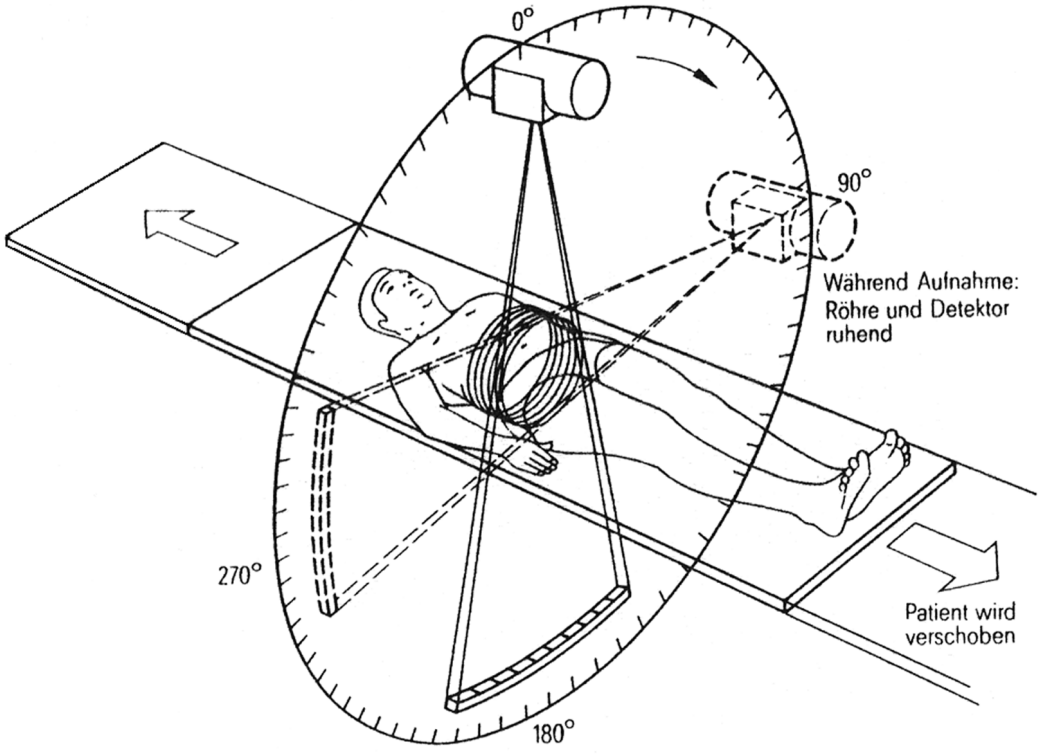

Abbildung 11: Moderne Computertomographie [LEHM97]

Zur bildlichen Darstellung werden den Dichtewerten der verschiedenen Gewebe Grauwerte zugeordnet. Da nur 256 Grauwerte dargestellt werden können – was aufgrund der Wahrnehmungsfähigkeiten des menschlichen Auges auch sinnvoll und ausreichend ist – besteht die Möglichkeit der sogenannten Fensterung. Der Benutzer kann ein Fenster von Dichtewerten angeben, welches dargestellt werden soll. Dieses Fenster wird durch Angabe des Zentrums (Center) und der

Breite (Window) definiert (Abbildung 12). Durch Verschieben des Zentrums erreicht man eine Variation der Helligkeit des Bildes, durch Veränderung der Fensterbreite verändert man den Kontrast. Voraussetzung dafür ist natürlich, daß der ausgewählte Bereich von Dichtewerten auf die maximale Anzahl darstellbarer Grauwerte expandiert wird. Somit können am gleichen CT mit verschiedenen Fenstern unterschiedliche medizinische Fragestellungen beantwortet werden.

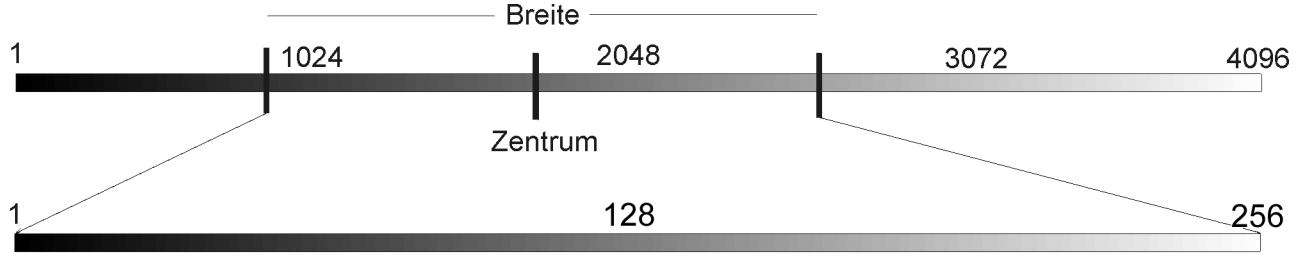

Abbildung 12: Fenstern

Die **Sonographie** basiert auf der Verwendung von Ultraschall (1-15 MHz) und auf dem akustischen Echoeffekt. Ultraschallwellen sind periodische Schwingungen von Materieteilchen, die sich als elastische Wellen räumlich ausbreiten. Die Ausbreitung der Schallwellen ist an Materie gebunden und die Beschaffenheit der Materie spielt für die Ausbreitungsgeschwindigkeit eine entscheidende Rolle.

Die von einem Ultraschallkopf ausgesendeten Schallimpulse werden beim Auftreffen auf akustische Grenzflächen[4] im Gewebe unterschiedlich stark reflektiert. Je größer der Unterschied der Schallwellenwiderstände der verschiedenen Gewebe, desto stärker ist die Amplitude des reflektierten Signals. Dies läßt Rückschlüsse auf die Gewebearten zu. An den Grenzflächen zwischen Weichteilen und Knochen oder Luft ist der Unterschied des Schallwellenwiderstandes sehr groß, so daß die Schallwellen fast vollständig reflektiert werden. Gebiete hinter luftgefüllten Räumen oder Knochen sind somit mit dem Ultraschallverfahren nicht einsehbar. Aus diesem Grund wird während der Untersuchung zwischen Schallkopf und Haut ein Gel aufgetragen. Dieses ermöglicht die luftfreie Ankopplung des Schallkopfes an das Gewebe. Die Entfernung des ausgemessenen Gewebes zum Schallkopf kann aus der Zeitdifferenz zwischen Aussendung der Schallwellen und Empfang des Echos berechnet werden. Der an die Körperoberfläche angelegte Schallkopf funktioniert dabei alternierend als Schallerzeuger und als Schallempfänger.

Zur Umsetzung des gemessenen reflektierten Signals in ein Bild werden verschiedene Verfahren verwendet. Beim sogenannten A-Bildverfahren[5] sendet ein Schallkopf einen kurzen Schallimpuls aus. Die Amplituden des gemessenen Reflektionssignals werden dann im zeitlichen Verlauf aufgetragen (A-Bild Abbildung 13). Das B-Bild zeigt dagegen zweidimensionale Schnittbilder. Dazu wird das A-Bildverfahren mit verschiedenen Ausstrahlungsrichtungen der Schallwellen wiederholt durchgeführt. Jede A-Messung wird in eine Linie von Bildpunkten umgesetzt, indem die Helligkeit des Punktes aus der Amplitude und die Lage aus der Reflektionszeit berechnet wird. Abbildung 13 zeigt dieses Vorgehen für eine Bildlinie. Eine Vielzahl dieser Linien ergeben dann ein zweidimensionales Bild. Allerdings ist dieses nicht rechteckig, sondern dreieckig, da alle Messungen kegelförmig von einem Punkt ausgehen (Abbildung 3). Dieses Verfahren ist so schnell, daß Bildwiederholfrequenzen von 25 Hz erreicht werden, so daß eine Darstellung von Bewegungsabläufen in Echtzeit möglich ist [LEHM97].

[4] Als akustische Grenzfläche bezeichnet man die Grenzfläche zweier Stoffe mit unterschiedlichen Schallwellenwiderständen.
[5] Das A steht für Amplitude.

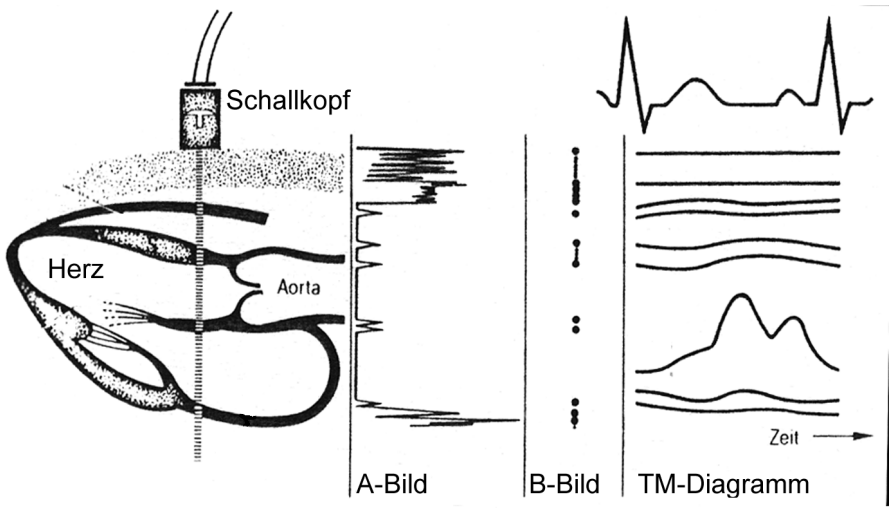

Abbildung 13: Die verschiedenen Darstellungsmodi beim Ultraschallverfahren [LEHM97]

Bei der **Kernspintomographie** macht man sich den Eigendrehimpuls – oder Spin – der Atomkerne mit ungerader Anzahl an Nukleonen[6] zunutze. Die Kerne dieser Atome gleichen rotierenden Kreiseln (Abbildung 14), deren Drehachsen beliebig im Raum orientiert sind. Bringt man sie aber in ein starkes äußeres Magnetfeld, so richten sich die Rotationsachsen parallel oder antiparallel zu diesem Feld aus (Abbildung 15).

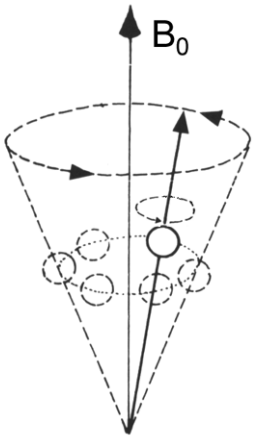

Abbildung 14: Kernspin und Präzession [LAUB90]

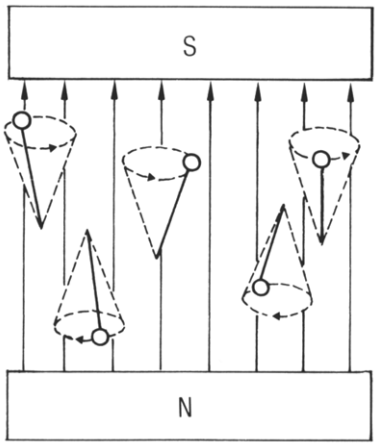

Abbildung 15: Ausrichtung der Kernspins im Magnetfeld [LAUB90]

Bei 37°C und einer magnetischen Flußdichte von 1 Tesla ist die Anzahl der parallel ausgerichteten Atome aus energetischen Gründen entsprechend der BOLTZMANN-Verteilung etwa 7 millionstel mal höher als die der antiparallel ausgerichteten. Da sich jeweils gleiche Anzahlen gegensätzlich ausgerichteter Atome in ihrer Wirkung aufheben, entstehen die Kernspinbilder gerade nur durch diese geringe Zahlendifferenz.

Nach der Ausrichtung durch das äußere Magnetfeld führen die Kerne zusätzlich zu ihrer Eigenrotation eine Kreiselbewegung (Präzession) um die Richtung des Magnetfeldes aus (Abbildung 14). Die Frequenz dieser Präzessionsbewegung wird auch als Lamor- oder Resonanzfrequenz bezeichnet. Grundlegend für die Kernspintomographie ist dabei, daß die Frequenz mit Zunahme der Feldstärke auch linear ansteigt, d.h. für jede Feldstärke gibt es genau eine Lamorfrequenz[7].

[6] Kernteilchen.
[7] Die Proportionalitätskonstante wird gyromagnetisches Verhältnis genannt und ist eine Stoffkonstante. Damit ergibt sich für jede Atomsorte eine andere Lamorfrequenz.

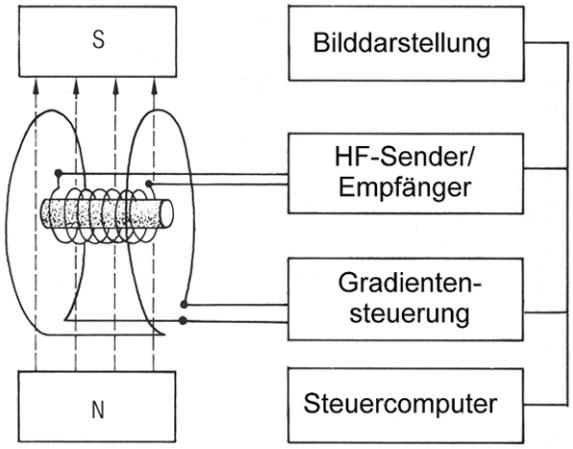

Abbildung 16: Blockschaltbild eines MRT [LAUB90]

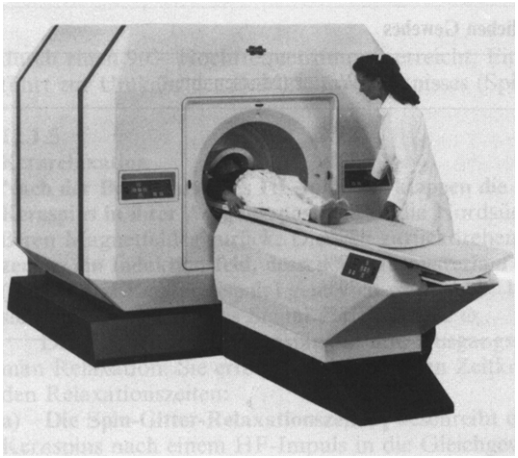

Abbildung 17: MRT-Gerät [LAUB90]

Setzt man die Atome nun zusätzlich einer hochfrequenten elektromagnetischen Strahlung (HF) aus, deren Frequenz genau der Lamorfrequenz entspricht, nehmen sie aus diesem Feld Energie auf. Dadurch wird der Winkel zwischen Magnetfeld und Rotationsachse zunehmend größer, etwa so, als ob man einen trudelnden Kreisel immer wieder anstößt. Nach Abschaltung des HF-Pulses haben die Atome die Tendenz, wieder ihre ursprüngliche Lage einzunehmen und senden dabei ihrerseits eine elektromagnetische Welle der Lamorfrequenz aus. Diese kann über ein Antennensystem empfangen werden und wird schließlich als Kernspin-Resonanz-Signal registriert.

Um nun ein ortsaufgelöstes Bild zu erzeugen, sorgt man durch drei senkrecht zueinander orientierte Elektromagnete (sogenannte Gradienten-Magnete) dafür, daß an jedem Raumpunkt innerhalb des Kernspintomographen ein anderes Magnetfeld herrscht. Damit ist auch die Lamorfrequenz an jedem Punkt unterschiedlich, so daß man durch geeignete Wahl der HF-Frequenz gezielt die Kerne in einem bestimmten Raumpunkt anregen kann.

Durch eine geeignete zeitliche Abfolge der Schaltung der Gradienten (Schichtselektionsgradient, Phasenkodiergradient und Auslesegradient), der HF-Pulse und der Empfängerschaltungen kann so ein beliebig orientiertes Schnittbild des Körpers angefertigt werden. Eine solche Schaltfolge wird auch Pulssequenz genannt.

Da diese Arbeit sich mit der Entwicklung eines virtuellen Kernspintomographen befaßt, folgt eine detaillierte Beschreibung der physikalischen und technischen Grundlagen der Kernspintomographie in Kapitel 3. Darüber hinaus gehende Informationen kann man insbesondere auch [LISS90], [HASH97], [PYKE82] und [LAUB90] entnehmen.

Aus medizinischer Sicht tritt zunächst nicht die zugrundeliegende Technik der Verfahren, sondern andere Aspekte in den Vordergrund. Dazu gehören die Qualität der Darstellung von Organen und Organsystemen, die Detektion und Differenzierung von pathologischen Strukturen, die Belastung des Patienten und die Kosten der Untersuchung.

Diesen Aspekten muß sich die medizinische Informatik als ein dienstleistendes Bindeglied zwischen zwei sehr unterschiedlichen Disziplinen stellen. Forschung und Entwicklung in diesem Bereich müssen sich immer an der medizinischen Relevanz messen lassen und das Verständnis der medizinischen Sichtweise und der Vorgehensweise eines Arztes ist eine wichtige Grundlage dafür. Wir wollen uns deshalb daran orientieren.

3 Grundlagen und Technik der Kernspintomographie

3.1 Mikroskopische und makroskopische Magnetisierung, Kernspin, Suszeptibilität und Präzession

Der menschliche Körper – den es mit der Kernspintomographie zu untersuchen gilt – setzt sich aus einer Vielzahl von Molekülen unterschiedlicher Gruppen zusammen. Die Moleküle sind Atomverbände und Atome bestehen aus einem Atomkern und einer Elektronenhülle. Ein Atomkern wiederum setzt sich aus Protonen und Neutronen zusammen. Alle Atomkerne mit ungerader Protonen- und/oder Neutronenzahl besitzen die Eigenschaft des Kernspins. Dieser ist ein Maß für die Eigenrotation des Atomkerns. Da der Atomkern aus mindestens einem Proton besteht, ist mit der Rotation des Atomkerns ein elektrischer Ringstrom verbunden. Somit erzeugt der rotierende Atomkern auch ein magnetisches Feld, äquivalent zu dem einer stromdurchflossenen Spule oder eines Stabmagneten. Die Stärke und Richtung dieses Feldes werden durch das magnetische Moment μ definiert (Abbildung 18).

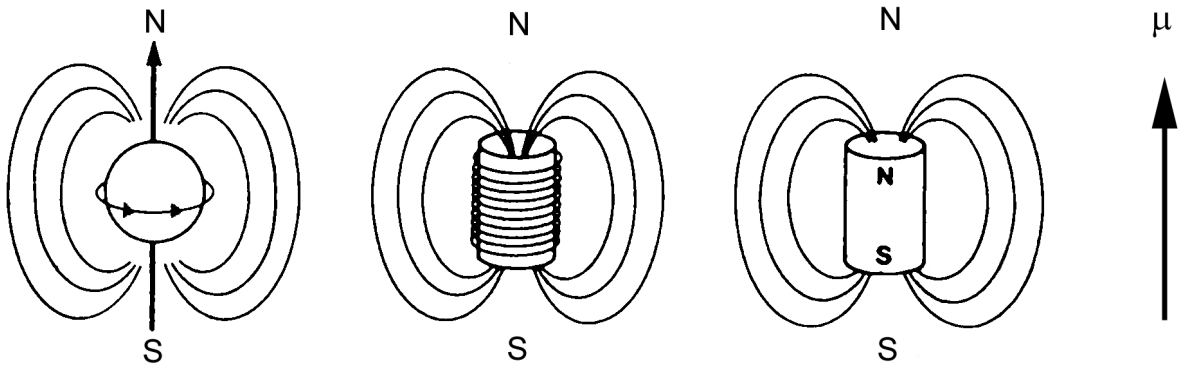

Abbildung 18: Magnetische Dipolfelder [LISS90]

Das magnetische Moment des Neutrons beträgt ca. 2/3 des magnetischen Moments des Protons [LISS90]. Daß das Neutron durch Rotation überhaupt ein magnetisches Moment verursacht ist dadurch bedingt, daß das Neutron wiederum aus positiv und negativ geladenen Quarks zusammengesetzt ist. Diese sind in gleicher Anzahl vorhanden, jedoch umlaufen die positiv geladenen Quarks die Spinachse in einem engeren Radius als die negativen. Daraus resultiert dann wiederum ein Ringstrom. Die Tatsache, daß nur Atomkerne mit einer ungeraden Anzahl von Protonen und/oder Neutronen einen Spin aufweisen ergibt sich dadurch, daß sich die Spins je eines Paares gleichartiger Teilchen (Protonen oder Neutronen) aus energetischen Gründen antiparallel anordenen (Pauli-Ausschließungsprinzip [SIEM92]). Somit ergibt sich der nach außen resultierende Kernspin nur aus dem Spin des ungepaarten Neutrons und/oder Protons.

Die am häufigsten in biologischen Systemen vorkommenden Elemente mit der Eigenschaft des Kernspins sind Wasserstoff (^{1}H) , Stickstoff (^{14}N), Phosphor (^{31}P) und Natrium (^{23}Na)[8] [LISS90]. In der Kernspintomographie verwendet man fast ausschließlich den Wasserstoff, da er das häufigste Element im menschlichen Körper ist und außerdem für die Magnetresonanz am empfindlichsten ist [PYKE82].

Im magnetfeldfreien Raum sind die magnetischen Momente statistisch verteilt in alle Raumrichtungen ausgerichtet, so daß sie sich in ihrer Wirkung kompensieren und sich nach außen eine Nettomagnetisierung[9] $M=0$ ergibt.

[8] Die Zahlen oben links am Elementsymbol stellen die Massenzahlen dar. Sie geben die Anzahl der Nukleonen an. Unter Nukleonen sind die Protonen und Neutronen zusammengefaßt. Sie werden auch als Kernteilchen bezeichnet.

[9] Auch Gesamtmagnetisierung genannt. Sie ergibt sich aus der Summe der magnetischen Momente μ.

Bringt man die Probe in ein (homogenes) Magnetfeld mit der magnetischen Flußdichte B_0 ein, so richten sich die magnetischen Momente entweder parallel oder antiparallel zum äußeren Magnetfeld aus (Abbildung 19). Es ist dann eine nach außen meßbare Magnetisierung vorhanden, die jedoch prinzipiell sehr schwach ist. Eine Begründung für das ungewöhnliche Verhalten der antiparallelen Einstellung kann nur aus der Quantenmechanik heraus gegeben werden und eine Erklärung dieses Sachverhalts würde den Rahmen dieser übersichtsartigen Ausführungen sprengen.

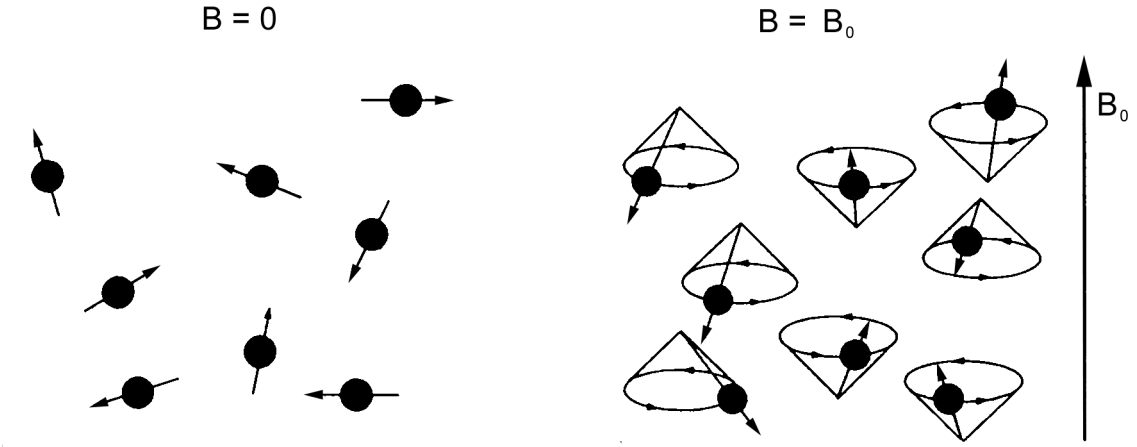

Abbildung 19: Einstellung der Spins im feldfreien Raum und im Magnetfeld B_0 [LISS90]

Mit den beiden Einstellungen sind unterschiedliche Energieniveaus verbunden (Abbildung 21). Dabei ist die parallele Ausrichtung die energetisch günstigere und bevorzugte und der Übergang in das energetisch höhere Niveau erfordert die Zufuhr von Energie der Menge

$$\Delta E = h \cdot \nu = \hbar \cdot \gamma \cdot B \tag{1}$$

mit

- ΔE Energiedifferenz zwischen den Energieniveaus
- h Planck'sches Wirkungsquantum ($6{,}6256 \cdot 10^{-34}$ J·s) und $h = \frac{\hbar}{2\pi}$
- ν Resonanzfrequenz[10] der Spins
- γ Gyromagnetisches Verhältnis (z.B. für das Proton ist $\frac{\gamma}{2\pi} = 42{,}577\,Mhz / T$ [11])
- B Magnetische Flußdichte[12] [LISS90], [HASH97].

Die Anzahl der möglichen Energiezustände ist von der Art des Atomkerns abhängig. Jedes Element hat eine sogenannte Quantenzahl *S*. Beim Wasserstoff ist *S*=½. Sie Anzahl der Energiezustände ergibt sich dann nach der Formel

$$Anzahl\ der\ Energiezustände = 2S + 1 \tag{2}$$

für Wasserstoff zu 2. Diese werden in der Chemie mit -½ und +½ bezeichnet, was andeuten soll, daß eine Gruppe der Kerne sich links herum dreht und die andere Gruppe rechts herum. Mit Hilfe der aus der Physik bekannten Rechte-Hand-Regel ergeben sich dadurch die unterschiedlichen Ausrichtungen des durch die rotierende Ladung erzeugten magnetischen Feldes.

---

[10] Eine Erläuterung des Begriffs Resonanzfrequenz folgt weiter unten.

[11] T ist Einheitenzeichen für die magnetische Flußdichte Tesla.

[12] Die meisten Bücher beschreiben das Symbol *B* als die magnetische Feldstärke. In Wirklichkeit ist jedoch die magnetische Flußdichte gemeint. Diese ist eine vektorielle physikalische Größe zur Beschreibung des Magnetfeldes. Die Magnetische Feldstärke *H* errechnet sich als $H = \frac{B}{\mu_0}$, wobei μ_0 die magnetische Feldkonstante ist.

Die Energiedifferenz zwischen diesen Zuständen liegt in der Größenordnung von 10^{-8} eV, ist jedoch abhängig von der magnetischen Flußdichte B und der Teilchenart (bestimmt durch das gyromagnetische Verhältnis γ). Dennoch sind im Normalfall nicht alle Spins parallel (also energetisch günstig) zum externen Magnetfeld ausgerichtet. Dies trifft nur bei Temperaturen nahe dem absoluten Nullpunkt zu. Die Nettomagnetisierung erreicht dann ihr Maximum. Bei Raumtemperatur ist die thermische Austauschenergie um einige Größenordnungen höher als die Energiedifferenz der beiden Spineinstellungen. Daraus resultiert eine fast identische Besetzung der beiden Niveaus mit kleinem Vorteil zugunsten der energetisch günstigeren – parallelen – Spinausrichtung. Die Besetzung der beiden Energieniveaus ist also temperaturabhängig und wird durch die Boltzmann-Gleichung [LISS90] genau definiert:

$$\frac{n\uparrow}{n\downarrow} = e^{-\Delta E/kT} \tag{3}$$

mit

n	Besetzungszahl des Energieniveaus ($n\uparrow$: antiparallel, $n\downarrow$: parallel)
ΔE	Energiedifferenz zwischen den Energieniveaus
k	Boltzmann-Konstante (k=1,38054·J·K^{-1})
T	Temperatur [K].

Der erwähnte kleine Vorteil zugunsten des energetisch günstigeren Niveaus ist es jedoch allein, der die Nettomagnetisierung in Feldrichtung ergibt und die Information eines Kernspinresonanzexperiments ausmacht. Der Überschuß im niedrigeren Energieniveau beträgt beispielsweise bei einer Temperatur von 37°C (Körpertemperatur) und einem externen Magnetfeld mit der Flußdichte 0,35 T lediglich $2\cdot10^{-6}$, also 2 ppm[13]. Diese auf den ersten Blick kleine Zahl relativiert sich, wenn man bedenkt, daß 1 ml Wasser ungefähr $6{,}691\cdot10^{22}$ Wasserstoffprotonen enthält [HASH97]. Wie aus der Boltzmann-Gleichung ersichtlich ist, läßt sich dieser Überschuß durch Absenken der Temperatur oder durch Erhöhen der externen Magnetfeldstärke (Einsetzen der Definition von ΔE) steigern. Während die erste Möglichkeit aus physiologischen Gründen ausscheidet, erreicht man durch Erhöhen der Magnetfeldstärke ein deutlich größeres meßbares Signal und damit eine bessere Bildqualität. Magnetfelder bis 2 Tesla sind zur Zeit im Einsatz [LISS90].

Obwohl es in einem äußeren Magnetfeld in der Probe mehr Spins im niedrigeren Energieniveau gibt, als außerhalb eines Magnetfeldes, wird der Gesamtenergiegehalt der Probe nicht kleiner, denn das Spinensemble gibt während der Magnetisierung Energie an den umgebenden Atomverband (Gitter) ab.

Die Magnetisierung der Probe dauert eine gewisse Zeit. Das Ansteigen der Magnetisierung folgt einer exponentiellen Wachstumskurve (Abbildung 20). Die Zeitkonstante dieser Wachstumskurve hängt dabei von der Art der Probe und von der Stärke des Magnetfeldes ab. Wenn der Vorgang beendet ist, befindet sich die Probe im energetischen Gleichgewicht. Das ist der Zustand, zu dem ein Spinensemble im Magnetfeld immer hinstrebt.

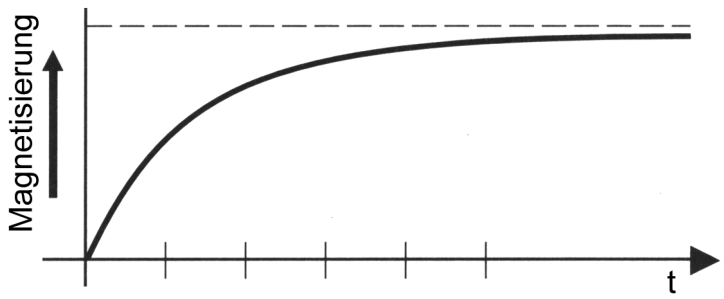

Abbildung 20: Magnetisierungsverlauf einer Probe in einem externen Magnetfeld [HASH97]

[13] ppm = parts per million.

Alle Substanzen werden zu einem gewissen Grad magnetisiert, wenn man sie einem Magnetfeld aussetzt. Der Magnetisierungsgrad, der als **Suszeptibilität** bezeichnet wird, ist jedoch von Substanz zu Substanz verschieden. Man unterscheidet drei Gruppen von Substanzen anhand ihrer Suszeptibilität: diamagnetische, paramagnetische und ferromagnetische.

- Diamagnetische Substanzen haben keine ungepaarten Elektronen in der Elektronenhülle. Sie haben eine leicht negative Suszeptibilität, d.h. wenn sie einem Magnetfeld ausgesetzt werden, bilden sie ein schwaches Magnetfeld entgegen der Hauptfeldrichtung aus. Solche Substanzen sind grundsätzlich nicht magnetisch.

  Die meisten Gewebe im Körper sind diamagnetisch, so zum Beispiel auch Wasser. Das heißt, daß sich eine Nettomagnetisierung entgegen der Richtung des äußeren Magnetfeldes ausbildet.

- Paramagnetische Substanzen haben ungepaarte Hüllenelektronen. Sie werden durch ein magnetisches Feld magnetisiert, verlieren die Magnetisierung jedoch wieder, wenn das Feld abgeschaltet wird. Sie bilden ein magnetisches Feld parallel zur Hauptfeldrichtung aus und haben daher eine leicht positive Suszeptibilität.

  Das Element mit den meisten ungepaarten Elektronen ist Gadolinium. Es ist stark paramagnetisch und wird in der Kernspintomographie als Kontrastmittel eingesetzt.

- Ferromagnetische Substanzen werden von einem Magnetfeld stark magnetisiert und bleiben dies auch nach Abschaltung des Feldes. Sie haben eine große positive Suszeptibilität.

  Die einzigen ferromagnetischen Substanzen sind Eisen, Kobalt und Nickel.

Zum Schluß dieses Kapitels soll noch ein Aspekt präzisiert werden. Und zwar stimmt die Beschreibung der Ausrichtung der Spins parallel bzw. antiparallel zum externen Magnetfeld nur im zeitlichen Mittelwert. Tatsächlich sind alle Spins und damit alle magnetischen Momente in beiden Einstellungsrichtungen in einem Winkel von 54°44‘ zur externen Magnetfeldrichtung ausgerichtet. Zusätzlich zur Eigenrotation rotieren (präzedieren) die Spins mit einer Frequenz ω um die Magnetfeldachse (z-Richtung) (Abbildung 22). Die Präzessionsfrequenz ist durch die Lamor-Beziehung gegeben [LISS90]:

$$\omega = \gamma \cdot B \quad \text{bzw.} \quad \nu = \frac{\gamma \cdot B}{2\pi} \tag{4}$$

mit

ω	Lamor- bzw- Präzessionsfrequenz in Umdrehungen pro Zeiteinheit
ν	Lamor- bzw- Präzessionsfrequenz in Radianten pro Zeiteinheit
γ	Gyromagnetisches Verhältnis
B	Magnetische Flußdichte.

Die Präzessionsfrequenz hängt somit vom Kerntyp und der Stärke des externen Magnetfeldes ab. Bei den in der Kernspintomographie verwendeten Feldstärken ist die Lamorpräzession hochfrequent. Bei 1,0 Tesla beträgt die Frequenz von Protonen etwa 42,577 MHz, liegt also im Bereich von Radiowellen (Tabelle 2, Seite 19) [LISS90].

Das Prinzip der Präzession kann man sich anhand eines Kreisels vorstellen. Wenn man einen (senkrecht) rotierenden Kreisel anstößt (entspricht dem Einschalten des Magnetfeldes), kippt er ein wenig zur Seite, fällt jedoch nicht um. Seine Eigenrotation verhindert dies. Die Drehachse des Kreisels beschreibt von nun an einen Kegel um die Richtung der Schwerkraft [SIEM92].

Im energetischen Gleichgewicht präzedieren alle Spins zwar mit gleicher Geschwindigkeit, jedoch mit unterschiedlicher Phase. Das ist der Grund dafür, daß unter diesen Bedingungen keine Magnetisierungskomponenten in x- oder y-Richtung auftreten. Im statistischen Mittelwert kompensieren sich die Komponenten der Spins in x- und y-Richtung zu Null und es resultiert eine Nettomagnetisierung in Richtung der externen Magnetfeldachse (z-Richtung).

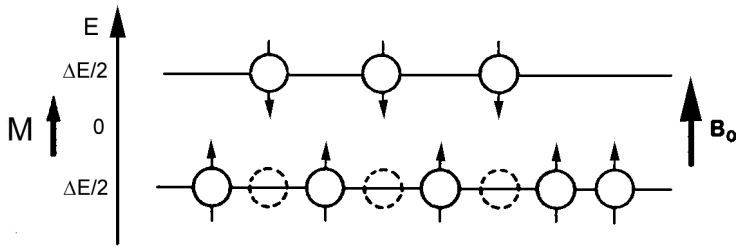

Abbildung 21: Parallele und antiparallele Anordnung der Spins mit unterschiedlichen Energieniveaus [LISS90]

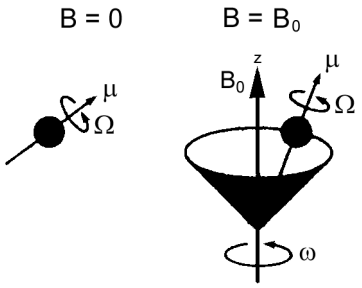

Abbildung 22: Kernspin und Präzession [LISS90]

3.2 Kernspinanregung und Kernspinresonanz

Wir betrachten nun lediglich die für die Nettomagnetisierung verantwortlichen Überschußspins im energetisch günstigeren Niveau. Dies sind je nach externer Magnetfeldstärke lediglich 2-13 ppm. Wie im letzten Kapitel bereits erwähnt, tritt im energetischen Gleichgewicht keine Nettomagnetisierungskomponente in *x*- oder *y*-Richtung auf, da die Spins in unterschiedlicher Phasenlage präzedieren.

Durch Einstrahlung hochfrequenter elektromagnetischer Wellen (HF-Strahlung) in die *xy*-Ebene lassen sich die Spins in Phase bringen[14], d.h. sie präzedieren anschließend phasenkohärent. Für den Energieaustausch ist dabei ausschließlich die magnetische Komponente der elektromagnetischen Strahlung verantwortlich. Wichtig ist, daß die Präzessionsfrequenz und die Frequenz der eingestrahlten elektromagnetischen Strahlen übereinstimmen. Das kann man sich anhand von Stimmgabeln vorstellen: Schlägt man eine Stimmgabel mit der Frequenz f_1 an, so werden die sich ausbreitenden Schallwellen nur andere Stimmgabeln zu Schwingungen veranlassen, die die gleiche Frequenz haben. Man nennt diesen Vorgang Resonanz. Stimmgabeln, die Schallwellen einer Frequenz $f_2 \neq f_1$ erzeugen, werden dagegen nicht angeregt [SIEM92]. Ein Resonanzübergang tritt somit nur bei Übereinstimmung der Anregungsfrequenz mit der Lamorfrequenz auf ($\omega = 2\pi \cdot \nu$) [LISS90]. Die somit definierte Frequenz der HF-Strahlung wird daher auch Resonanzfrequenz genannt.

Man kann sich das Einstrahlen der elektromagnetischen Strahlung auch als Hineinwerfen von kleinen rotierenden Stabmagneten vorstellen. Wir abstrahieren nun aber noch weiter und stellen uns statt vieler kleiner einfach einen großen Stabmagneten vor, der sich in der Probe mit der Präzessionsfrequenz dreht. Unter dieser Voraussetzung werden die Spins ‚mitgezogen', d.h. in die Phasenlage des großen Stabmagneten gebracht (Abbildung 23) [SIEM92].

Zur Vereinfachung der folgenden Betrachtungen nehmen wir nun an, daß wir den Vorgang der Spinbewegung nicht von einem fest stehenden äußeren Standpunkt betrachten, sondern daß sich die Beobachterposition mit den Kernspins mit der Präzessionsfrequenz um die externe Magnetfeldachse dreht. Dadurch fällt die Präzession scheinbar weg und die Betrachtung der Bewegungsvorgänge wird vereinfacht. Vergleichbar ist dies mit der Bewegung eines Karussells. Während ein außenstehender Beobachter eine Überlagerung aus dem Drehen des Karussells um die Karussellachse und aus dem auf- und abbewegen eines Karussellpferdes beobachtet, nimmt ein auf der Karussellplattform stehender Beobachter nur das auf- und abbewegen des Pferdes wahr. Zusätzlich scheint sich die Außenwelt um den Beobachter zu bewegen [SIEM92]. Wir verwenden also zur Vereinfachung einiger Bewegungsabläufe im folgenden ein rotierendes Koordinatensystem.

[14] Der Zustand der absoluten Phasenkohärenz ist nur durch einen 90°-Impuls erreichbar. Darauf wird im folgenden näher eingegangen.

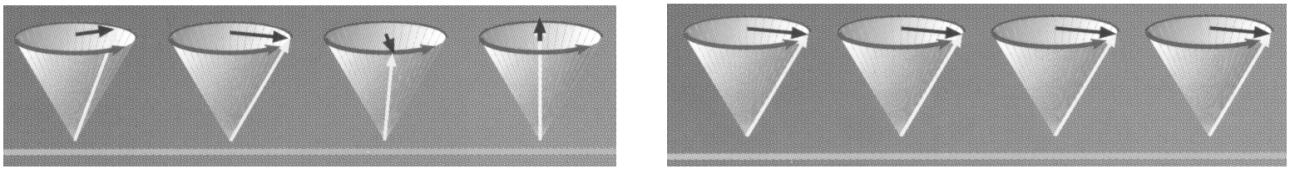

Abbildung 23: Phasenkohärenz der Spins nach Einstrahlung von HF-Strahlung (unten) [SIEM92]

Betrachtet man den Vorgang der HF-Anregung vom rotierenden Koordinatensystem aus, so wächst die im energetischen Gleichgewicht null betragende Magnetisierungskomponente in *xy*-Richtung mit zunehmender Phasenkohärenz der Spins an. Gleichzeitig reduziert sich die M_z-Magnetisierungskomponente dadurch, daß mit der phasenkohärenten Präzession die Bedingung für die Energieabsorption und damit für das Umspringen der Spins in das energetisch höhere Niveau erfüllt ist. Die eingestrahlte Energie entspricht dann nämlich genau der Differenz zwischen den beiden Energieniveaus:

$$E = h \cdot v = \frac{\gamma \cdot h \cdot B}{2\pi} \text{ (Resonanzbedingung)} \qquad (5)$$

mit

- h Planck'sches Wirkungsquantum ($6{,}6256 \cdot 10^{-34}$ J·s)
- v Frequenz der elektromagnetischen Strahlung [s^{-1}]
- γ Gyromagnetisches Verhältnis
- B Magnetische Flußdichte [LISS90].

Schon minimale Abweichungen von dieser Resonanzbedingung verhindern eine Resonanzanregung. Genau diese Tatsache kann aber zur Ortskodierung (Kapitel 3.4) ausgenutzt werden.

Wie vorher schon erwähnt liegt die einzustrahlende Energie im Bereich von 10^{-8} eV. Sie liegt damit im Bereich der KW- und UKW-Energien und um Größenordnungen unter dem Energiegehalt von Röntgenstrahlung (10^7 eV), UV-Strahlung (10^2 eV) und sichtbarem Licht (10^0 eV) (Tabelle 2) [LISS90], [LEHM97].

Frequenz v		Strahlung	Wellenlänge λ	
$3 \cdot 10^{11}$			10^{-3}	
$3 \cdot 10^{10}$		Röntgen und Gammastrahlung (10^{-5}-10 nm)	10^{-2}	nm
$3 \cdot 10^{9}$			10^{-1}	
$3 \cdot 10^{8}$	GHz		1	
$3 \cdot 10^{7}$			10	
$3 \cdot 10^{6}$		Ultraviolette Strahlung (10-400 nm)	100	
$3 \cdot 10^{5}$		Sichtbares Licht (400-700 nm)	1	
$3 \cdot 10^{4}$			10	µm
$3 \cdot 10^{3}$		Infrarotstrahlen (780 nm – 1mm)	100	
300			1	
30		Mikrowellen (10-100 mm)	10	mm
3			100	
300		Fernsehen (ca. 2 m), Ultrakurzwellen (1-10 m)	1	
30	MHz	Kurzwellen (10-80 m)	10	m
3		Mittelwellen (200-600 m)	100	
300			1	
30	kHz		10	
3		Langwellen (> 600m)	100	km
300			10^3	
30	Hz	Schallwellen	10^4	

Tabelle 2: Spektrum elektromagnetischer Wellen [LISS90], [LEHM97]

Alle elektromagnetischen Wellen haben zwei Komponenten: ein elektrisches Feld und ein magnetisches Feld. Beide breiten sich mit Lichtgeschwindigkeit aus, haben die gleiche Frequenz, sind

allerdings um 90° phasenverschoben und stehen senkrecht zueinander. Uns interessiert insbesondere die magnetische Komponente. Die elektrische Komponente ist eher unerwünscht, da sie Wärme erzeugt.

Insgesamt wird durch die HF-Einstrahlung auch der Gesamtmagnetisierungsvektor aus seiner Gleichgewichtslage ausgelenkt und in Richtung der *xy*-Ebene gekippt (Abbildung 24 b). Der Vektor der Gesamtmagnetisierung beschreibt während dieses Vorgangs im ortsfesten Koordinatensystem eine spiralförmige Bahn auf der Kugel, die von dem Gleichgewichtsmagnetisierungsvektor M_0 aufgespannt wird (Abbildung 24 a).

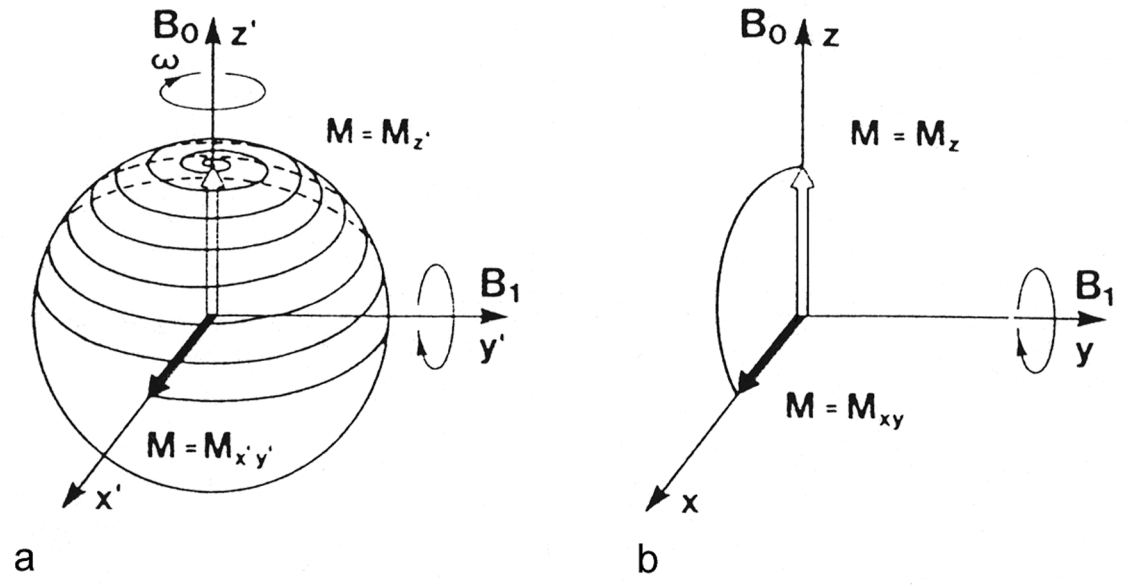

Abbildung 24: Auslenkung des Gesamtmagnetisierungsvektors durch einen 90°-Impuls im ortsfesten Koordinatensystem (a) und im rotierenden Koordinatensystem (b) [LISS90]

Wenn so viele Kernspins in das energetisch höhere Niveau wechseln, daß kein Überschuß mehr im energetisch niedrigerem Niveau besteht, so wird $M_z=0$ und M_{xy} hat den gleichen Betrag wie M_0 im Gleichgewichtszustand. In diesem Fall präzedieren die Spins absolut phasenkohärent und man spricht von einem **90° Impuls**. Die Zeit bis zum Erreichen der 90°-Auslenkung ist von der Stärke und Einwirkungsdauer des von der HF-Strahlung erzeugten Magnetfeldes B_1 abhängig. Oder anders gesagt ist der Grad der Auslenkung allein von der Zeitdauer der Einstrahlung und der Stärke des Magnetfeldes des Hochfrequenzimpulses abhängig:

$$\alpha = \gamma \cdot B_1 \cdot \tau \tag{6}$$

mit

- α Auslenkwinkel (Flipwinkel)
- γ Gyromagnetisches Verhältnis
- B_1 Durch die HF-Strahlung erzeugte magnetische Flußdichte
- τ Einschaltdauer der HF-Strahlung.

Verdoppelt man die Einwirkungsdauer bei gleichbleibender magnetischer Fußdichte B_1 (durch den Hochfrequenzimpuls erzeugt), so erreicht man einen **180°-Impuls**. Die Spins werden dabei um 180° gedreht, das heißt, es treten in diesem Fall keine Quermagnetisierungen in der *xy*-Ebene auf (also auch keine Phasenkohärenz), jedoch wird die Besetzungszahldifferenz der beiden Energieniveaus umgedreht. Bei weiterer Einstrahlungsdauer des Hochfrequenzimpulses nimmt zwar der Auslenkwinkel weiter zu, die Differenz der Nettomagnetisierung zur Gleichgewichtsmagnetisierung wird jedoch wieder kleiner. Also müssen auch wieder mehr Spins in das energetisch niedrigere Niveau wechseln. Das ist dadurch zu begründen, daß die Kernspinresonanz nicht nur eine Resonanzabsorption sondern auch eine Resonanzemission von Hochfrequenzenergie beinhaltet. Die maximal zuführbare

Energie entspricht also der eines 180°-Impulses. Für die Resonanzanregung in der Kernspintomographie werden überwiegend 90°, 180° und HF-Impulse <90° (Kapitel 3.3) verwendet [LISS90]. Kleinwinkelanregungen, also Auslenkungen des Magnetisierungsvektors um Winkel <90°, erreicht man durch Reduzierung der Stärke oder der Dauer der Einstrahlung des HF-Impulses. Sie zeichnen sich dadurch aus, daß der Betrag der Magnetisierung in der *xy*-Ebene kleiner ist, als der des Gleichgewichtmagnetisierungsvektors M_0, nämlich:

$$M_{xy} = M_0 \cdot \sin\alpha \tag{7}$$

mit

M_{xy} Betrag der Magnetisierung in *xy*-Richtung
M_0 Betrag der Gleichgewichtsmagnetisierung
α Auslenkwinkel.

Wir betrachten zur Definition der folgenden Begriffe zunächst einmal die Verhälnisse nach einem 90°-Impuls.

Longitudinale Relaxationszeit T_1

Betrachten wir nun das Spinensemble unmittelbar nach der Resonanzanregung. Nach einem 90°-Impuls präzedieren die Spins mit der Präzessionsfrequenz ω in der *xy*-Ebene. Der rotierende Magnetisierungsvektor induziert in der Empfangsantenne, die ebenfalls in der *xy*-Ebene angeordnet ist, eine Spannung, die der Länge des Magnetisierungsvektors M_{xy} proportional ist. Durch den Übergang ins energetische Gleichgewicht kehren die Spins in einem exponentiellen Zeitverlauf in den Gleichgewichtszustand zurück (Abbildung 25) und verlieren ihre Phasenkohärenz. Dabei emittieren sie die zuvor hineingesteckte Anregungsenergie mit der gleichen Frequenz (Präzessionsfrequenz). M_z nimmt von 0 aus kontinuierlich zu, bis wieder der Gleichgewichtswert erreicht ist. Dieses zeitliche Anwachsen der longitudinalen Magnetisierung wird durch die Relaxationszeit T_1 beschrieben. T_1 ist genau die Zeit, die M_z braucht, um wieder 63,21% ($=1-1/e$) der Gleichgewichtsmagnetisierung M_0 zu erreichen [LISS90]. Tabelle 3 gibt einige T_1-Konstanten von verschiedenen Geweben bei unterschiedlichen Magnetfeldstärken an und Abbildung 27 zeigt den zeitlichen Verlauf der longitudinalen Relaxation verschiedener Gewebe.

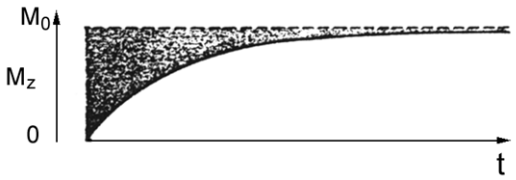

Abbildung 25: T_1-Relaxation nach einem 90°-Impuls [LISS90]

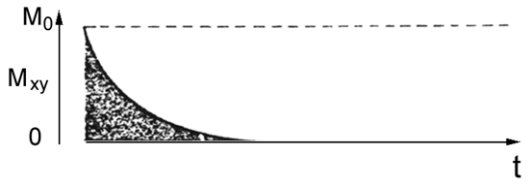

Abbildung 26: T_2-Zerfall nach einem 90°-Impuls [LISS90]

Transversale Relaxationszeit T_2

Unmittelbar nach einer HF-Anregung präzedieren die Spins zwar phasensynchron, jedoch nimmt die Kohärenz im zeitlichen Verlauf ab, da einige Spins sich schneller und einige langsamer bewegen als der Durchschnitt. Die phasensynchrone Bewegung der Spins geht also im Laufe der Zeit verloren, bis die Spins schließlich gleichmäßig verteilt in alle Raumrichtungen zeigen. Dies führt zu einer zeitlich exponentiellen Abnahme der Quermagnetisierung M_{xy}, die durch die transversale Relaxationszeit T_2 gekennzeichnet ist (Abbildung 26). Das ist genau die Zeit, bis 63,21% der Quermagnetisierung verloren gegangen sind.

Da die transversale Relaxation an das Vorhandensein energetisch angeregter Spins gebunden ist, sind die T_2-Zeiten immer kürzer als die T_1-Zeiten. Die transversale Relaxation beeinflußt die longitudinale

Relaxation jedoch nicht, da an das Auseinanderlaufen der Spins kein Energieübertrag gebunden ist. So kommt es schließlich dazu, daß der Betrag (die Länge des Vektors) der Gesamtmagnetisierung während der Relaxationsphase nicht (!) konstant ist. Ist $T_2 << T_1$, so strebt M_{xy} zu einem Zeitpunkt gegen Null, zu dem die longitudinale Relaxation noch kaum begonnen hat (also $M_z \approx 0$).

Der kombinierte Vorgang der longitudinalen und transversalen Relaxation wird **freier Induktionszerfall** (FID = free induction decay) genannt. Wie bereits erwähnt, wird durch die präzedierende M_{xy}-Magnetisierung in der Empfangsspule eine Spannung induziert. Das Signal, das man durch den freien Induktionszerfall erhält, liefert also ausschließlich Informationen über die T_2-Zeit und natürlich über die Protonendichte. Tabelle 3 enthält die T_2-Konstanten einiger Gewebe bei einer Magnetfeldstärke von 1 Tesla. Abbildung 28 zeigt den zeitlichen Verlauf der transversalen Relaxation verschiedener Gewebe.

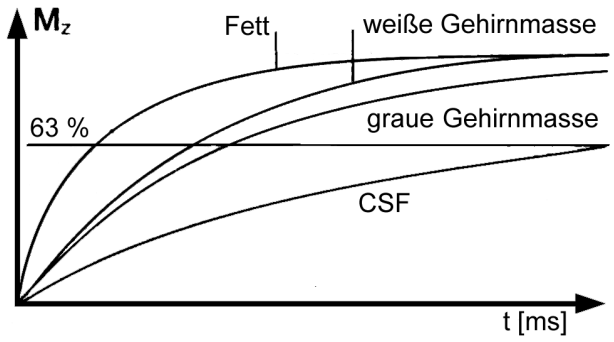

Abbildung 27: Verlauf der longitudinalen Relaxation bei verschiedenen Geweben [SIEM92]

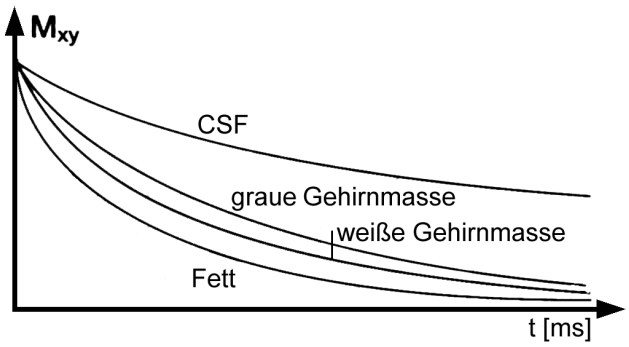

Abbildung 28: Verlauf der transversalen Relaxation bei verschiedenen Geweben [SIEM92]

Transversale Relaxationszeit T_2^*

Der zuvor beschriebene Vorgang der transversalen Relaxation mit der Zeitkonstanten T_2 trifft nur im absolut homogenen Magnetfeld zu. Aufgrund von Magnetfeldinhomogenitäten präzedieren die Spins an unterschiedlichen Orten mit unterschiedlichen Frequenzen, wodurch die Dephasierung verstärkt und die gemessene T_2-Zeit verkürzt wird. Eine Bestimmung der wirklichen, probenspezifischen T_2-Zeit ist mit dem freien Induktionszerfall also nicht möglich. Die durch den freien Induktionszerfall bestimmte T_2-Zeit wird daher gesondert durch T_2^* bezeichnet, wohingegen T_2 den probenspezifischen Wert meint [LISS90]. Ein Verfahren zur Bestimmung der T_2-Zeit wird in Kapitel 4.3.1 erläutert.

Protonendichte

Bleibt nun noch der Zusammenhang zwischen der Protonendichte und der Signalintensität zu klären. Dieser ist jedoch offensichtlich: Je mehr Kerne relaxieren, desto stärker ist das Resonanzsignal. Tabelle 3 zeigt auch die Protonendichte einiger Gewebe.

	T_1 [ms]	T_2 [ms]	Protonendichte [%]
Wasser	2500	2500	100
Fett	200	100	81
CSF	2000	300	100
Graue Gehirnmasse	500	100	69
Weiße Hirnmasse	510	67	61
Ödeme	900	126	86

Tabelle 3: T_1- und T_2-Konstanten (in ms) bei 1 Tesla sowie Protonendichte [Hash97]

Damit sind nun 4 für die Kernspintomographie wichtige intrinsische Gewebeparameter bekannt: Die Suszeptibilität, die longitudinale Relaxationszeit T_1, die transversale Relaxationszeit T_2 und die Protonendichte.

Bevor wir im nächsten Kapitel die Folgen der Hintereinanderschaltung verschiedener Anregungsimpulsen beschreiben, betrachten wir noch einmal kurz die Verhältnisse nach einem 180°-Impuls unter Berücksichtigung der eben gewonnenen Erkenntnisse. Der 180°-Impuls führt zu einer Besetzungsumkehr der Energieniveaus, so daß die *z*-Komonente der Gleichgewichtsmagnetisierung umgedreht wird. Die *xy*-Klomponente ist jedoch zu jeder Zeit Null. Aufgrund der fehlenden Quermagnetisierung M_{xy} ist zunächst kein Resonanzsignal in der *xy*-Ebene nachweisbar. Die Relaxation erfolgt so, daß die *z*-Magnetisierung mit der Relaxationszeit T_1 abnimmt, zum Zeitpunkt t=0,693 T_1 ihren Nulldurchgang erreicht und dann wieder gegen die Gleichgewichtsmagnetisierung M_0 strebt (Abbildung 29).

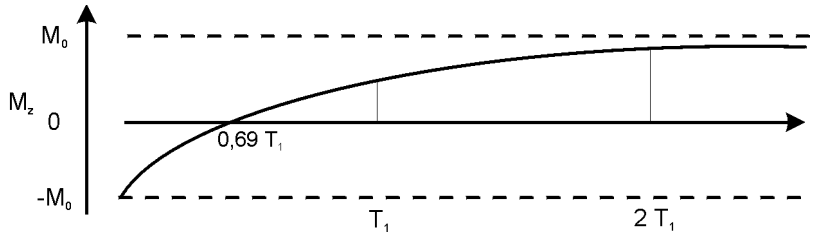

Abbildung 29: T_1-Relaxation nach einem 180°-Impuls

3.3 Pulssequenzen

Nach den einleitenden Worten zu den Grundlagen der Kernspintomographie stellt sich nun die Frage, wie man die im vorangegangenen Kapitel beschriebenen intrinsischen Eigenschaften der einzelnen Gewebetypen (also die beschriebenen Relaxationszeiten T_1 und T_2 sowie die Protonendichte) zur Signalerzeugung ausnutzen kann.

Eine weitere Frage ist, wie man die gemessenen Signale aus dem zuvor beschriebenen Kernspinresonanzexperiment räumlich kodieren und somit zur Bilderzeugung nutzen kann. Im einfachen Kernspinresonanzexperiment würde der eingestrahlte HF-Impuls die Spins im gesamten Körper anregen, somit würden auch sämtliche anregungsfähigen Kernspins zum Gesamtsignal im gleichen Maße beitragen. Dieser Aspekt hat sowohl in der Praxis als auch in der Simulation ebenso wie die Wahl der Pulssequenzen eine große Bedeutung und wird in Kapitel 3.4 und 3.5 behandelt.

Unter dem Begriff Pulssequenz versteht man eine Folge von HF- und Ortskodierungsimpulsen. Wenn man zu dem zuvor beschriebenen FID (Abbildung 30), der nur aus einem einzigen 90°-Impuls besteht, Ortkodierungsimpulse hinzu nimmt, könnte man auch diese Pulsfolge bereits als Pulssequenz bezeichnen. Hier soll die Frage der Ortskodierung aber erst einmal außen vor bleiben. Untersucht man die Signalintensität des FID im Verlauf des Relaxationsprozesses, so ergibt sich folgendes Signalverhalten:

- Zum Startzeitpunkt t_0 hängt die Signalintensität nur von der Spindichte ab, denn der 90°-Impuls klappt den Magnetisierungsvektor M_z mit gleichem Betrag in die *xy*-Ebene um.
- Der weitere Zerfall und damit auch die Signalintensität hängt nur von der Relaxationszeit T_2^* ab.

Die Antenne empfängt nur Signale auf Grund des Zerfalls in der *xy*-Ebene, somit ist klar, daß die T_1-Zeit keinen Einfluß auf die Signalintensität hat.

Die Anordnung der Antennen ist auch für alle anderen Pulssequenzen von Bedeutung. Um überhaupt ein messbares Signal zu erzeugen, ist es notwendig, eine transversale Magnetisierung zu erzeugen. Bei den konventionellen Techniken wird dazu ein 90°-Impuls verwendet. Bei den Gradientenechosequenzen werden aber auch Auslenkwinkel verwendet, die kleiner als 90° sind.

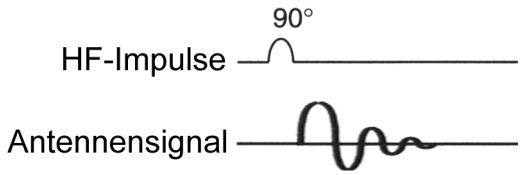

Abbildung 30: Pulsdiagramm einer FID-Sequenz; frei nach [HASH97]

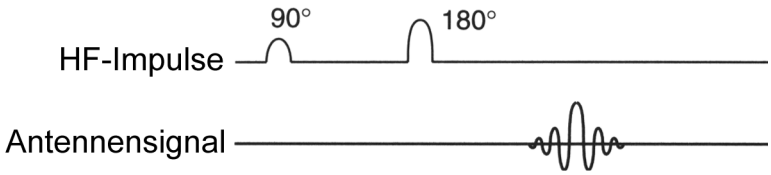

Abbildung 31: Pulsdiagramm einer Spin-Echo-Sequenz [HASH97]

Abbildung 31 zeigt beispielhaft das Diagramm einer Pulssequenz für eine Spin-Echo-Technik (siehe auch Kapitel 3.3.1.3). Hier wird zusätzlich zu dem 90°-Impuls noch ein 180°-Impuls eingestrahlt. Bei dieser Technik wird ein sogenanntes Spinecho erzeugt, das in Abbildung 31 unten angedeutet ist.

Bei komplexeren Pulssequenzen kann man im Gegensatz zum FID den Einfluß der Gewebeeigenschaften (T_1, T_2 und Spindichte) auf die Signalintensität beeinflussen. Allerdings hängt nicht jede Sequenz von allen Gewebeeigenschaften ab. Je nach Pulssequenz und Einstellung der Parameter dieser Sequenz ergeben sich unterschiedliche Gewichtungen der T_1- und T_2-Zeiten sowie der Spindichte. Man spricht in diesem Zusammenhang von T_1-, T_2-, Spindichtebetonung oder –wichtung.

Im ersten Unterkapitel behandeln wir die konventionellen Pulssequenzen. Allen diesen konventionellen Pulssequenzen ist gemeinsam, daß sie aus einer Folge von 180°- und 90°-Impulsen bestehen. Im zweiten Unterkapitel werden wir dann die Schnellbildtechniken beschreiben. Schon seit einiger Zeit existieren Verfahren, die die im Vergleich zu den konventionellen Pulssequenzen sehr langen Akquisitionszeiten reduzieren. Seither haben die Gerätehersteller eine Vielzahl von Sequenzen auf den Markt gebracht, dabei haben gleiche oder ähnliche Sequenzen von Hersteller zu Hersteller unterschiedliche Namen.

Entscheidend für die Simulation ist vor allem, daß eine Reduktion der Akquisitionszeiten auch mit einem Verlust an Qualität und Artefaktentstehung einhergehen kann.

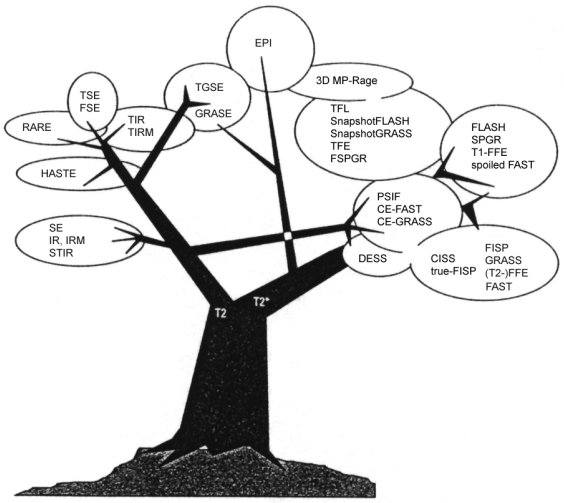

Abbildung 32: Sequenzbaum [SIEM97]

Abbildung 32 zeigt eine Übersicht über die aktuell gängigen Sequenzen für die Magnetresonanz-tomographie. Auf einige dieser Sequenzen wollen wir in den folgenden Kapitel näher eingehen, da sie auch in der Simulation realisiert sind.

Der Sequenzbaum unterteilt sich in zwei Hauptzweige, dabei befinden sich links die konventionellen Sequenzen (u.a. die Spin-Echo-Sequenzen) und rechts die Gradientenechosequenzen.

Die Sequenzen, die wir im ersten Unterkapitel betrachten wollen, befinden sich im unteren linken Blatt des Sequenzbaumes. Es handelt sich dabei um die Spin-Echo-Sequenz (SE) und die Inversion-

Recovery-Sequenz (IR). Zusätzlich wird die Saturation-Recovery-Sequenz (SR) beschrieben. Sie ist klinisch nicht mehr gebräuchlich, weil ein sofortiges Auslesen des dabei auftretenden FID zwar nötig, aber aus technischen Gründen nicht möglich ist. Dadurch ergibt sich auch ein Problem durch Magnetfeldinhomogenitäten, das bei der Spin-Echo-Sequenz vermieden wird. Saturation-Recovery ist hier aber trotzdem zuerst beschrieben, weil man durch deren Beschreibung die anderen Sequenzen leichter versteht. In [HASH97] wird sie als „Sprungbrett“ zum Verständnis der komplexeren Sequenzen bezeichnet. Sie ist leicht umzusetzen und daher auch im Simulationsprogramm verfügbar. Die Blattgruppen gehören jeweils zu ähnlichen Sequenzfamilien. Teilweise unterscheiden sich auch nur die Bezeichnungen, die besonders bei Gradientenechosequenzen bei unterschiedlichen Geräteherstellern sehr heterogen sind.

Beim Betrachten des Sequenzbaumes wird klar, daß das modulare Konzept des virtuellen Kernspintomographen wesentlich ist (siehe dazu auch Kapitel 5.5.1). Sämtliche Sequenzen zu implementieren würde den Rahmen dieser Arbeit sprengen, spätere Versionen jedoch sollten das vorhandene Simulationswerkzeug möglichst einfach um weitere Sequenzen erweitern können. Auch die Entwicklung neuer Sequenzen ist sicherlich noch nicht am Ende angekommen, Mezrich schreibt sogar, daß die Entwicklung neuer Techniken nur durch die Phantasie der Entwickler begrenzt ist [MEZR95], also sollten auch neuentwickelte Sequenzen aufgenommen werden können.

3.3.1 Konventionelle Techniken

Wie bereits erwähnt, bestehen die konventionellen Techniken allesamt aus einer Folge von 90°- und 180°-Impulsen. Alle diese Sequenzen enthalten mindestens einen 90°-Impuls. Ein brauchbares Signal kann nicht durch einen einzelnen 180°-Impuls erzeugt werden, da dieser nur die *z*-Komponente der Gesamtmagnetisierung beeinflußt.

Eine weitere Gemeinsamkeit im Signalverhalten der konventionellen Pulssequenzen ist, daß es mindestens von zwei der Gewebeparameter abhängt. Wir werden in den nächsten Unterkapiteln die konventionellen Pulssequenzen „Saturation-Recovery“ (SR), „Inversion-Recovery“ (IR) und „Spin-Echo“ (SE) beschreiben.

3.3.1.1 Saturation-Recovery

Bei der Pulssequenz Saturation-Recovery handelt es sich um eine Folge von 90°-HF-Impulsen, die im zeitlichen Abstand der Repititionszeit T_R appliziert werden. Ein einzelner dieser 90°-Impulse entspricht wiederum einem FID. Genauso wie beim FID erfolgt also die transversale Relaxation mit dem Zeitfaktor T_2^*, die longitudinale mit dem Zeitfaktor T_1. Das Signal wird direkt nach Applizieren des 90°-Impulses ausgelesen.

Da jetzt aber der Anregungsimpuls mehrfach appliziert wird, wird das Signal beim zweiten und allen folgenden Impulsen schwächer. Das ist dadurch zu erklären, daß der 90°-Impuls die *z*-Komponente der Magnetisierung in die *xy*-Ebene zur Messung klappt, aber abhängig von der Relaxationszeit T_1 und dem Zeitpunkt der Einstrahlung des nächsten 90°-Impulses T_R die *z*-Magnetisierung noch nicht ihren vollen Wert M_0 erreicht hat. Beim Umklappen der *z*-Magnetisierung in die *xy*-Ebene ist bei der Messung aber genau der Anteil der relaxationsfähigen Spins proportional zur Signalintensität, also wird das Signal schwächer.

In Abbildung 33 läßt sich das Gesagte gut veranschaulichen. In Abbildung 33a sieht man die Abfolge der HF-Impulse mit dem Abstand T_R. Abbildung b zeigt die Relaxation in *xy*-Ebene, die wie beim einfachen FID mit der Zeitkonstante T_2^* abläuft. Vereinfacht ist hier aber nicht das empfangene Signal dargestellt, sondern die Amplitude des mit der Zeitkonstante T_2^* abfallenden Signals. Unten in Abbildung 33c sieht man die longitudinale Relaxation. Die T_R-Zeit ist hier mit $T_R=3T_1$ gewählt, also sind zum Zeitpunkt der erneuten Anregung 95% der Spins relaxiert. Nur diese 95% können in der zweiten Anregung zur Signalintensität beitragen, das Signal wird etwas schwächer. Nach dem zweiten Impuls stellt sich dann ein Gleichgewicht ein, der Anteil der nicht relaxierten Spins bleibt konstant.

In Abbildung 34 ist dies nochmals veranschaulicht. In Abbildung 34a sieht man die longitudinale Relaxation mit dem Zeitfaktor T_1. Die Pfeile, die mit *I* und *II* beschriftet sind, deuten einen 90°-Impuls an. In Abbildung 34b wird jeweils nach der Zeit $T_R = T_1$ der nächste HF-Impuls eingestrahlt. Die höchste Amplitude des gemessenen Signals ist damit nur noch 63,21% des vorhergehenden Signals, denn das ist genau die Definition von T_1: T_1 ist die Zeit, nach der 63,21% der Spins relaxiert sind. In Abbildung 34c ist die T_R-Zeit = $3T_1$, was wie schon gesehen dazu führt, daß die maximale Amplitude des zweiten und der folgenden Signale 95% von der maximalen ersten Amplitude beträgt.

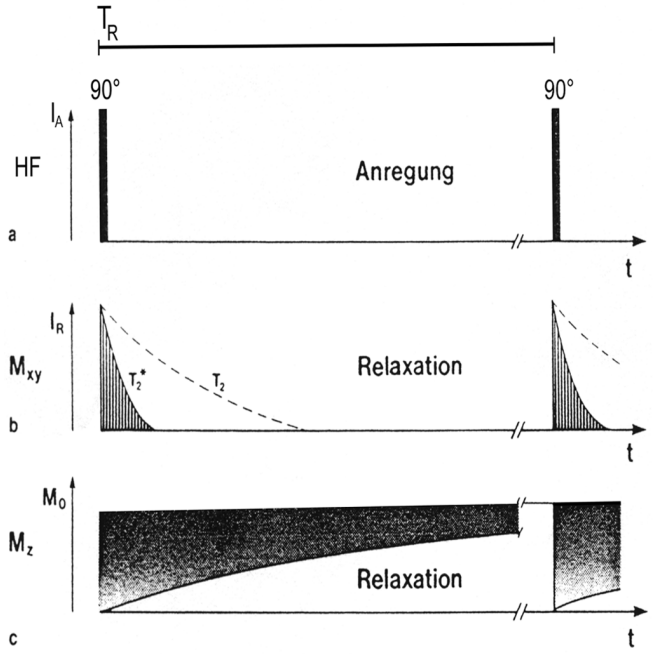

Abbildung 33: Saturation-Recovery-Pulssequenz (SR) [LISS90]

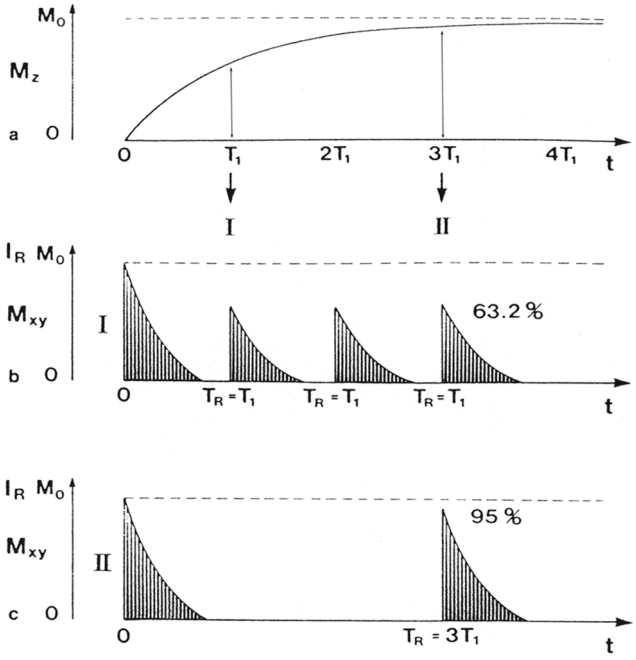

Abbildung 34: Signalintensität beim FID [LISS90]

Während beim reinen FID T_1 und T_2 nicht bestimmbar sind, hängt beim Saturation-Recovery die Signalintensität nicht mehr von T_2^* ab, da direkt zum Zeitpunkt T_R die *z*-Magnetisierung durch den HF-Impuls in die *xy*-Ebene umgeklappt und direkt nach dem 90°-Impuls gemessen wird und damit die Signalintensität von der Protonendichte und T_1 abhängt. An Abbildung 33 und Abbildung 34 wird auch deutlich, daß sich die Signalintensität um so stärker reduziert, je kleiner T_R im Verhältnis zu T_1 ist.

Bei bekannter T_1-Zeit kann man die Signalintensität auch rechnerisch bestimmen[15]:

$$I = N(H) \cdot (1 - e^{-T_R/T_1}) \qquad (8)$$

mit

I	Signalintensität
$N(H)$	Protonendichte
T_R	Repititionszeit
T_1	Spin-Gitter-Relaxationszeit.

Dies ist auch eine der Berechnungen, die neben den Manipulationen des k-Raumes (Kapitel 3.5) eine zentrale Rolle in der Simulation des virtuellen Tomographen spielen wird.

Wenn man die Formel betrachtet wird klar, daß die Signalintensität um so größer ist, je kürzer T_1 wird. Dementsprechend werden Substanzen mit kurzem T_1 hell dargestellt, Substanzen mit langem T_1 dunkel.

[15] Die T_R-Zeit ist ohnehin bekannt, denn T_R ist ein einstellbarer Parameter.

Obwohl die *xy*-Magnetisierung sehr schnell zerfällt, weil sie von T_2^* abhängig ist ($T_2^* << T_1$), ist die Signalintensität nur von der Protonendichte und T_1 abhängig. Das ist damit zu begründen, daß die Messung direkt nach Einstrahlung des zweiten HF-Impulses erfolgt. Wählt man T_R dagegen sehr groß ($T_R >> T_1$) ist die Signalintensität nur noch von der Spindichte abhängig, genau wie beim FID. Man erhielte also ein protonengewichtetes Signal. Wählt man dagegen T_R sehr klein, nimmt der Einfluß von T_1 auf die Signalgebung zu.

In [HASH97] heißt diese Sequenz ohne vollständige Relaxation (d.h. mit kurzen T_R) „partial Saturation“ und mit vollständiger Relaxation „Saturation-Recovery“.

Somit kann man durch Selektion des Parameters T_R entweder ein Protonen- oder ein T_1-gewichtetes Signal erzeugen.

3.3.1.2 Inversion-Recovery

Ebenso wie Saturation-Recovery erlaubt auch diese Pulssequenz T_1-betonte Bilder zu erzeugen. Der Unterschied besteht jedoch in einem zusätzlichen initialen Impuls, der die *z*-Magnetisierung invertiert. Auf die Spinenergieniveaus bezogen heißt das, daß die Besetzungszahl zugunsten der antiparallelen Spins umgekehrt wird. Der Betrag der Magnetisierung bleibt dabei gleich. Nach der sogenannten Inversionszeit T_I erfolgt ein 90°-Impuls. Genau wie der 90°-Impuls beim Saturation-Recovery dient er dazu, die *z*-Magnetisierung in die *xy*-Ebene zu klappen, da nur so ein Signal gemessen werden kann. Die Repititionszeit T_R ist bei Inversion-Recovery durch die Zeitdauer zwischen zwei 180°-Impulsen vorgegeben. Im Unterschied zu Saturation-Recovery tritt nach dem initialen Impuls keine Quermagnetisierung auf, erst beim Auslesen wird durch den 90°-Impuls eine *xy*-Magnetisierung erzeugt.

Direkt nach dem 180°-Impuls relaxiert die *z*-Magnetisierung von $-M_0$ nach M_0. Der Nulldurchgang erfolgt für t=0,69 T_1. Zum Zeitpunkt T_{Inv} wird die *z*-Magnetisierung durch den 90°-Impuls in die *xy*-Ebene umgeklappt. Zur Berechnung der Signalintensität gibt es auch für Inversion-Recovery eine Formel:

$$I = N(H) \cdot (1 - 2e^{-T_{Inv}/T_1}) \tag{9}$$

mit

I	Signalintensität
$N(H)$	Protonendichte
T_{Inv}	Inversionszeit
T_1	Spin-Gitter-Relaxationszeit.

Für T_{Inv} = 0 wird der Ausdruck in Klammern –1, für T_{Inv} = 0,69 T_1 wie gesagt 0 und für $T_{Inv} \rightarrow \infty$ konvergiert er gegen 1.

Somit hängt es von T_{Inv} und von T_1 ab, wie stark die *z*-Magnetisierung zum Zeitpunkt des Ausleseimpulses bereits relaxiert ist. Einsichtig ist auch, daß im Verhältnis zu T_1 nur entweder lange oder kurze Inversionszeiten sinnvoll sind. Für T_{Inv}-Werte die im Bereich von T_1 liegen, bekommt man nur ein kleines Signal und damit ein schlechtes Signal-zu-Rausch-Verhältnis. In Abbildung 35 sieht man in der oberen Abbildung a den Verlauf der Spin-Relaxation. Die Pfeile deuten jeweils den 90°-Impuls zur Messung an. Die Abbildungen b, c und d zeigen dann die Amplitude des gemessenen Signals. In Abbildung c ist die Signalintensität besonders klein, da die z-Magnetisierung nah an ihrem Nulldurchgang ist.

In [HASH97] ist die Signalintensität für die Inversion-Recovery-Sequenz dagegen wie folgt gegeben:

$$I = N(H) \cdot (1 - 2e^{-T_{Inv}/T_1}) \cdot (1 - e^{-T_R/T_1}) \tag{10}$$

zusätzlich mit

T_R	Repititionszeit.

[LISS90] geht von $T_R >> T_1$ aus und in den vom Hersteller vorgegebenen IR-Sequenzen sind die Parameterbereiche in der Regel auch so gewählt, daß diese Formel gilt. Wir wollen den Anwender der Simulation hingegen ohne Vorgaben von Bereichen experimentieren lassen und insofern ist in der Simulation auch ein zu kleiner Wert für T_R möglich, der in der Formel aus [HASH97] mitberücksichtigt wird.

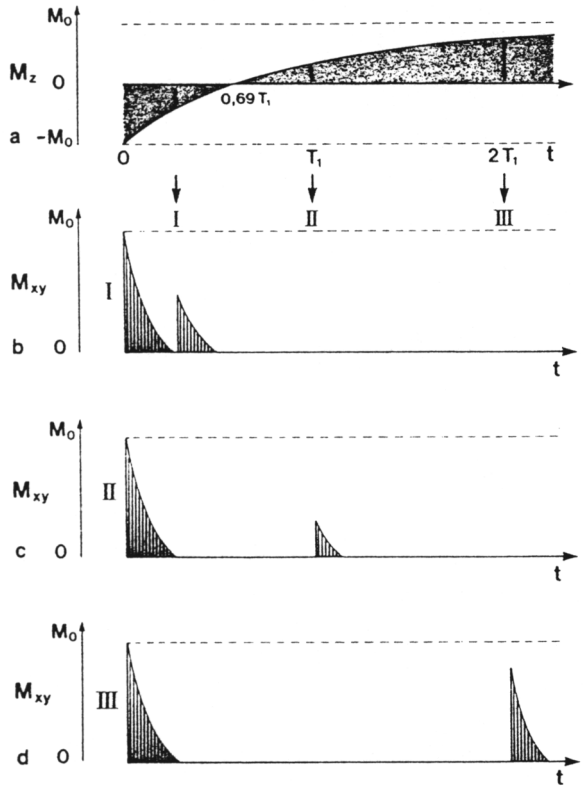

Abbildung 35: Signalintensität der Inversion-Recovery-Sequenz (IR) [LISS90]

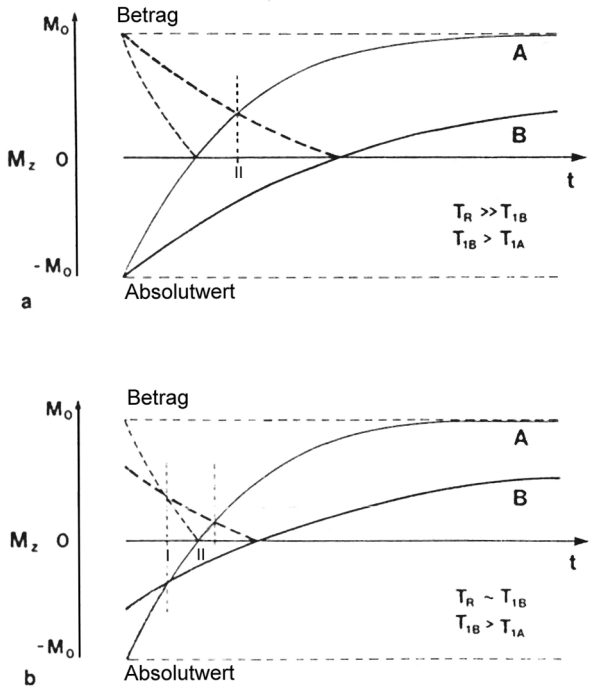

Abbildung 36: Signalintensität mit Absolutwerten und Betrag bei der Inversion-Recovery-Sequenz [LISS90]

Ein Vorteil der Inversion-Recovery-Sequenz gegenüber Saturation-Recovery ist die wegen der doppelten *z*-Magnetisierung erhöhte T_1-Empfindlichkeit. Die Signalintensität wird durch T_2^* nicht beeinflußt. Nachteilig ist, daß hohe Repititionszeiten nötig sind, da M_z sonst nicht vollständig relaxiert. Verwendet man zu kurze Relaxationszeiten, nimmt die Signalintensität ab dem zweiten 180°-Impuls ab. Die langen T_R-Zeiten wiederum führen zu langen Akquisitionszeiten.

Zur Darstellung der Signalintensität als Bild gibt es zwei Möglichkeiten, wobei die eine Möglichkeit nur die Absolutbeträge berücksichtigt, die andere dagegen auch das Vorzeichen der Magnetisierung.

Die Verwendung von Absolutwerten führt dazu, daß die Signalintensität für lange T_{Inv} und kurze T_1 hoch ist. Für kurze T_{Inv} dagegen liefern die Substanzen mit langem T_1 das größere Signal. Es kommt also zu einer Kontrastumkehr, wenn man T_{Inv} in einem bestimmten Intervall verändert. Anschaulicher wird dies in Abbildung 36. Abbildung a zeigt die Signalintensitäten bei vollständiger Relaxation. Dabei sind beide Darstellungsverfahren berücksichtigt. Die gestrichelten Kurven entstehen durch Darstellungsverfahren mit Absolutwerten, die durchgezogenen Kurven berücksichtigen das Vorzeichen mit. Nach dem Nulldurchgang stimmen die Absolutwerte mit den Signalintensitäten überein, dementsprechend liegen auch die Kurven übereinander. Berücksichtigt man die Absolutwerte, so nimmt die Signalintensität für die Probe A zunächst schnell ab und steigt dann auch schnell wieder an. Die Signalintensitäten für die Probe B nehmen langsam ab und steigen nach dem Nulldurchgang langsam wieder an. Am Schnittpunkt, der mit *II* markiert ist, kommt es dann zur Kontrastumkehr. Bei T_{Inv}-Werten die links von *II* liegen, liefert Substanz A das größere Signal, bei längeren T_{Inv}-Werten wird Substanz B heller dargestellt.

In Abbildung b ist die Repititionszeit so gewählt, daß die Spins nicht vollständig relaxieren. Hier tritt beim Darstellungsverfahren mit Absolutwerten sogar zweimal eine Kontrastumkehr auf. Für $T_{Inv} < I$ liefert Substanz A das größere Signal, für T_{Inv}-Werte zwischen *I* und *II* liefert Substanz B das größere Signal und für $T_{Inv} > II$ liefert schließlich wieder Substanz A ein größeres Signal. Für Verfahren, die das Vorzeichen berücksichtigen tritt die Kontrastumkehr entweder schon bei $T_{Inv} = 0$ (Abbildung a) oder aber für sehr kleine T_I auf (Abbildung b) und beeinflußt meist nicht den für klinische Untersuchungen interessanten Bereich von T_{Inv}.

Die hier beschriebene Sequenz ist eine klassische Inversion-Recovery-Sequenz. Meistens erweitert man die Impulsfolge noch um einen weiteren 180°-Impuls, der kurz nach dem 90°-Impuls eingestrahlt wird. Durch diesen zweiten 180°-Impuls erfolgt jetzt eine Invertierung in der *xy*-Ebene. Die beiden 180°-Impulse haben eine grundsätzlich unterschiedliche Funktion. Während der erste die Magnetisierung in *z*-Richtung invertiert, führt der zweite zu einem sogenannten Spin-Echo. Die gleiche Technik benutzen die Spin-Echo-Sequenzen, die im nächsten Kapitel beschrieben werden.

3.3.1.3 Spin-Echo

Die beiden bisher beschriebenen Pulssequenzen wurden in ihrer Signalintensität lediglich von der Spin-Gitter-Relaxationszeit T_1 und der Protonendichte *N(H)* beeinflußt. In der Regel ist jedoch bei pathologischen Veränderungen die Änderung von T_2 bedeutsamer als die Änderung von T_1. Die T_2-Zeit hat daher große diagnostische Bedeutung. Daraus erklärt sich auch, daß die SE-Technik die am häufigsten verwendete Pulssequenz ist.

Die Spin-Echo-Sequenz kann durch geeignete Wahl der Parameter sowohl ein protonengewichtetes Signal, ein T_1-gewichtetes Signal als auch ein T_2-gewichtetes Siganl erzeugen. Bei der Spin-Echo-Sequenz ist die Repititionszeit T_R durch den Abstand zwischen zwei 90°-Impulsen bestimmt. Wie schon besprochen zerfällt beim freien Induktionszerfall (FID) die *xy*-Magnetisierung sehr rasch mit der Zeitkonstanten T_2^*. Da hier unter anderem auch die externen Magnetfeldinhomogenitäten einwirken, wäre eine T_2-Bestimmung nur mit einem perfekt homogenen Magnetfeld möglich, was aber technisch nicht möglich ist. Da aber $T_2^* << T_2$, würde die Signalintensität im wesentlichen von externen Einflüssen abhängen.

Um aber dennoch ein Signal abhängig vom Gewebeparameter T_2 zu erzeugen, bedient man sich des Spin-Echo-Effektes. Das Echo wird dadurch erzeugt, daß man zusätzlich zu einem initialen 90°-Impuls, der die *z*-Magnetisierung in die *xy*-Ebene klappt, einen 180°-Impuls einstrahlt. Das Prinzip des Spin-Echos kann man am besten anhand von Abbildung 37 beschreiben.

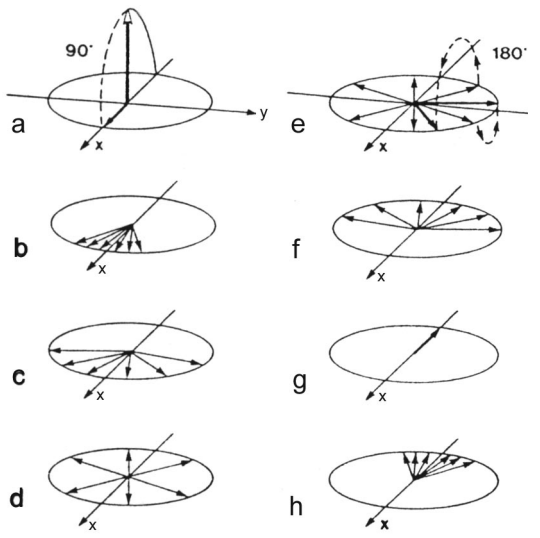

Abbildung 37: Spinrephasierung [LISS90]

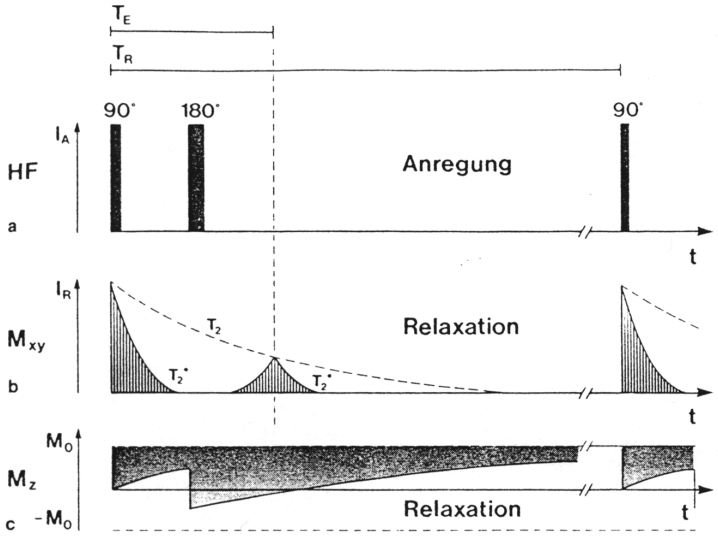

Abbildung 38: Die Spin-Echo-Sequenz [LISS90]

In Abbildung 37a ist das Umklappen der *z*-Magnetisierung in die *xy*-Ebene angedeutet. Die ursprünglich kohärent präzedierenden Spins kreisen auf Grund der lokal leicht unterschiedlichen Feldstärken mit unterschiedlichen Geschwindigkeiten. Diese Inhomogenitäten werden zum einen durch die gegenseitige Beeinflussung, aber vor allem durch die externen Magnetfeldinhomogenitäten verursacht. In den Abbildungen b – d verlieren die Spins immer mehr an Phase, bis die Phasenlagen in d schließlich völlig zufällig verteilt sind. Jetzt ist ein Zustand erreicht, der auch am Ende des FID auftritt. In e wird ein 180°-Impuls eingestrahlt. Dadurch wird die Richtung der Spins umgekehrt, die Präzessionsgeschwindigkeit bleibt aber konstant. Die schneller präzedierenden Spins liegen damit in der Phase hinter den langsam präzedierenden. Die Spins, die ursprünglich einen Phasenvorsprung hatten hinken jetzt also hinterher. In Abbildung f kommt es auf Grund des beschriebenen Verhaltens der Spins schon zu einer partiellen Rephasierung mit anwachsendem M_{xy} . In Abbildung g schließlich sind die durch Magnetfeldinhomogenitäten verursachten Phasenunterschiede kompensiert. Durch die kurzzeitig synchrone Spinbewegung entsteht ein meßbares Signal, das sogenannte Spin-Echo. Nach dem Spin-Echo erfolgt die Dephasierung erneut und M_{xy} zerfällt wieder mit der Zeitkonstanten T_2^* (Abbildung h).

In [HASH97] wird als Analogie zu den Spins eine Gruppe von drei Läufern angeführt, die auf einer Laufbahn im Kreis rennen. Anfangs starten sie am selben Punkt, aber nach einer Zeit τ laufen sie nicht mehr auf derselben Höhe, da der eine schneller läuft. Ein anderer läuft langsamer und fällt zurück. Nun lassen wir die Läufer umdrehen und in die andere Richtung laufen. Die Richtung hat sich geändert, aber die Läufer sind immer noch unterschiedlich schnell. Zum Zeitpunkt 2τ sind sie wieder auf gleicher Höhe. Übertragen auf die Spins bedeutet das, daß sie zu diesem Zeitpunkt wieder in Phase präzedieren.

In Abbildung 38 wird noch einmal die Spin-Echo-Pulssequenz veranschaulicht. In a sieht man die Abfolge der HF-Impulse, in b die transversale Relaxation mit Spin-Echo und in c die longitudinale Relaxation. Bei der Spin-Echo-Pulssequenz gibt es schon zwei einstellbare Parameter, nämlich die Echozeit T_E und die Repititionszeit T_R. Zu bemerken ist, daß der 180°-Impuls zum Zeitpunkt $T_E/2$ eingestrahlt wird. Das Spin-Echo tritt dann erst zum Zeitpunkt T_E auf. Das liegt daran, daß die Dephasierung der Spins ebenso lange dauert wie deren Rephasierung durch den 180°-Impuls. Zudem hat das Echo wegen des transversalen Zerfalls eine viel kleinere Amplitude.

Der Signalzerfall zwischen dem Zeitpunkt t=0 und t=T_E ist wegen der Rephasierung nur von T_2 abhängig und nicht mehr von T_2^*. Die Signalintensität für Spin-Echo berechnet sich als

$$I = N(H) \cdot (e^{-T_E/T_2}) \cdot (1 - e^{-T_R/T_1}) \qquad (11)$$

mit

I	Signalintensität
$N(H)$	Protonendichte
T_R	Repititionszeit
T_E	Echozeit
T_1	Spin-Gitter-Relaxationszeit
T_2	Spin-Spin-Relaxationszeit.

Wenn man die Formel genauer betrachtet, fällt auf, daß der Term in der zweiten Klammer genauso in der Formel für Saturation-Recovery vorkommt. Auf Grund der Relaxation der Spins fließt auch bei der Spin-Echo-Sequenz dieser Term als Faktor ein. Der Aspekt, daß auch T_1 einen Einfluß hat wird im folgenden betrachtet. Je nach Wahl der T_R-Zeit im Verhältnis zu T_1 sind die Spins nicht vollständig relaxiert. Der initiale 90°-Impuls klappt bekanntlich die *z*-Magnetisierung in die *xy*-Ebene um. Genau wie beim Saturation-Recovery sind bei genügend kurzer Repititionszeit noch nicht alle Spins relaxiert. Substanzen mit kurzer T_1-Zeit relaxieren schneller und liefern damit eine höhere Signalintensität, Substanzen mit langer T_1-Zeit relaxieren langsamer und liefern eine geringere Signalintensität. In der Formel wird der Exponent der zweiten *e*-Funktion bei immer kürzerem T_1 kleiner, die *e*-Funktion

konvergiert gegen 1. Wenn man dagegen T_R vergrößert, nimmt der Einfluß von T_1 auf die Bildgebung ab.

Angenommen, zwei Substanzen weisen unterschiedliche T_1 und T_2 auf, dann treten Kreuzungspunkte der Signalintensität auf. Die Spindichte ist bei beiden Substanzen gleich. Das kann man sich in Abbildung 39 veranschaulichen.

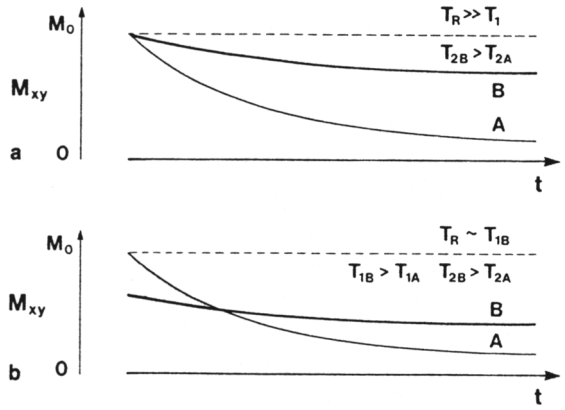

Abbildung 39: SE-Signalintensität in Abhängigkeit von Aufnahme- und Gewebeparametern [LISS90]

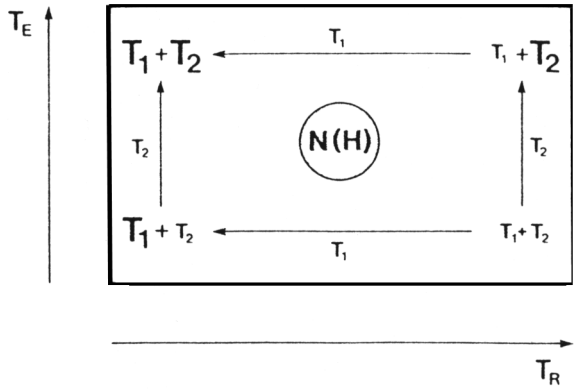

Abbildung 40: Abhängigkeit der Bilderzeugung von den Geräteparameter bei einer Spin-Echo-Sequenz [LISS90]

In Abbildung a sorgt eine lange Repititionszeit für die vollständige Relaxation der Spins. Also ist M_{xy} zum Zeitpunkt des 90°-Impulses für alle Substanzen gleich. Egal wie die T_E-Zeit entlang der *x*-Achse gewählt wird, stets liefert Substanz B mit der längeren T_2-Zeit das größere Signal. Ist dagegen T_R so gewählt, daß keine vollständige Relaxation erfolgt, tritt ein Kreuzungspunkt auf. Die Spins in Substanz A relaxieren in Abbildung b vollständig auf Grund der kürzeren T_1-Zeit, dagegen sind die Spins in Substanz B nicht vollständig relaxiert. Für den initialen T_2-Zerfall tritt somit eine Kontrastumkehr auf, denn für kleine T_E liefert jetzt Substanz A die größere Signalintensität, erhöht man T_E, liefert wieder Substanz B die höhere Signalintensität.

Die Spin-Echo-Sequenz ist also von allen Parametern abhängig (Protonendichte, T_1 und T_2), allerdings kann die Gewichtung dieser Parameter durch die Veränderung der Parameter T_E und T_R beeinflußt werden.

Abbildung 40 zeigt den Einfluß der physikalischen Gewebeparameter in Abhängigkeit der einstellbaren Geräteparameter T_R und T_E. Die Größe der Buchstaben in den vier Ecken veranschaulicht den Einfluß des Parameters auf die Bilderzeugung.

- Eine Spindichtebetonung erreicht man durch lange T_R-Zeiten und kurze T_E-Zeiten. Durch die langen T_R-Zeiten relaxieren die Spins weitgehend, der Einfluß von T_1 auf die Bildgebung nimmt ab. Durch kurze T_E-Zeiten wird der Einfluß von T_2 auf die Bildgebung kleingehalten. Die Signalintensität hängt im wesentlichen nur noch von der Protonendichte ab (untere rechte Ecke).
- Eine T_1-Betonung erreicht man durch kurze T_R-Zeiten und kurze T_E-Zeiten. Durch die kurzen T_R-Zeiten relaxieren die Spins nur unvollständig, der T_1-Einfluß auf die Bildgebung nimmt zu. Die kurzen T_E-Zeiten sorgen dafür, daß T_2 nur einen geringen Einfluß auf die Bildgebung hat. Insgesamt erhält man ein T_1-betontes Signal (untere linke Ecke).
- Eine T_2-Betonung erhält man durch lange T_R-Zeiten und langen T_E-Zeiten (obere rechte Ecke).
- Eine gemischte und in der Medizin nicht gewünschte T_1/T_2-Wichtung erhält man mit kurzen T_R-Zeiten und langen T_E-Zeiten (obere linke Ecke).

Unabhängig von der Wahl von T_R und T_E ist die Signalintensität aber stets von der Protonendichte abhängig.

Zur Erklärung der Bildeigenschaften bei unterschiedlichen Parametereinstellungen ist in [HASH97] ein Beispiel mit dem Signalverhalten von Substanzen, die im Gehirn vorkommen, angegeben. Ihre Eingenschaften sind in Abbildung 41, Abbildung 42 und Tabelle 4 dargestellt.

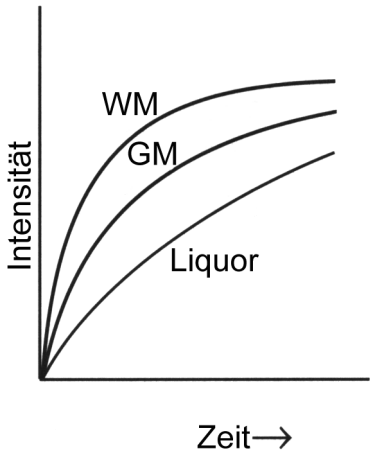

Abbildung 41: T_1-Relaxationskurven einiger Substanzen [HASH97]

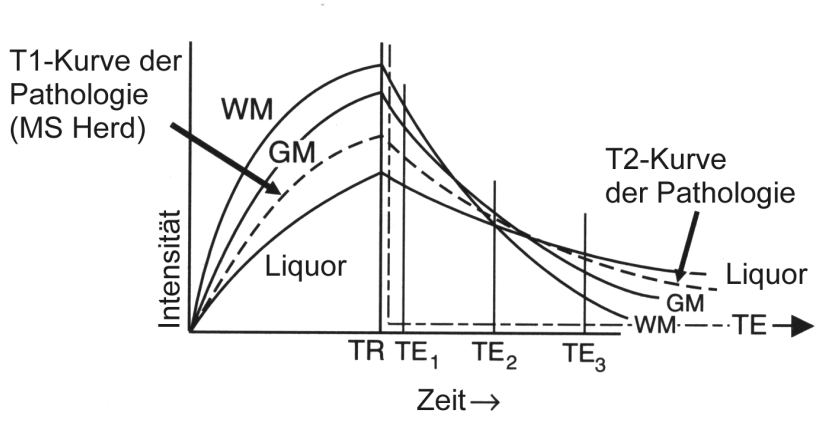

Abbildung 42: T_1-Kurve verschiedener Gewebe im Gehirn. Das eingezeichnete zweite Koordinatensystem zeigt die T_2-Relaxation nach einem 90°-Impuls [HASH97]

	T_1 [ms]	T_2 [ms]	Protonendichte [%]
Weiße Gehirnmasse	510	67	61
Graue Gehirnmasse	760	77	69
Liquor cerebrospinalis	2650	180	100

Tabelle 4: T_1, T_2 und Protonendichte von Gehirngeweben [HASH97]

Abbildung 41 zeigt die T_1-Relaxationskurven der Substanzen. Die weiße Gehirnmasse verhält sich ähnlich wie Fett, denn das enthaltene Myelin hat ähnliche Eigenschaften, die T_1-Relaxation erfolgt schnell, die Signalintensität ist am höchsten. Die graue Gehirnmasse enthält kein Myelin und hat ähnliche T_1-Eigenschaft wie ein typischer Festkörper. Die Gehirnflüssigkeit erscheint dunkel, die T_1-Zeit ist etwas kürzer als die von Wasser. Abbildung 42 zeigt den Verlauf der Signalintensitäten vor und nach dem 90°-Impuls.

In Abbildung 42 ist neben der T_1-Relaxationskurve auch der T_2-Zerfall nach dem 90°-Impuls eingezeichnet. Da $T_R >> T_E$, ist die Darstellung wegen der besseren Übersichtlichkeit nicht maßstabsgetreu, der T_2-Zerfall nach dem 90°-Impuls ist auf einer anderen Zeitachse dargestellt. Die T_2-Relaxationskurve der Gehirnflüssigkeit ist sehr flach, da sie genauso wie Wasser eine lange T_2-Zeit hat. Der Signalzerfall für die weiße Gehirnmasse geht auf Grund der kürzeren T_2-Zeit schneller als für die graue Gehirnmasse. Hier ist zusätzlich noch eine Pathologie eingezeichnet (Multiple Sklerose Herde) und das Signalverhalten für langes T_R und drei verschiedene T_E-Zeiten dargestellt.

- Für sehr kurzes T_E (T_{E1}) wird ein protonengewichtetes Signal erzeugt.
- Für kurzes T_E (T_{E2}) schneiden sich die Signalintensitäten von weißer Gehirnmasse und der Gehirnflüssigkeit. Sie erscheinen auf einem Bild isointens. Die graue Gehirnmasse ist in dieser Aufnahme am hellsten. In dieser Aufnahme ist der Einfluß von T_2 stärker gewichtet.
- Für langes T_E wird eine hohe T_2-Wichtung erzeugt. Auf Grund dessen ist die Gehirnflüssigkeit am hellsten.

Laut [HASH97] kann man die Pathologie am Besten mit der Einstellung T_{E2} vom restlichen Gewebe differenzieren, da Gehirnflüssigkeit und weiße Gehirnmasse isointens sind und die Pathologie heller als beide Substanzen ist.

Bisher haben wir noch nicht den Einfluß der Protonendichte auf den Signalverlauf berücksichtigt. In Tabelle 4 ist die Protonendichte in Prozent angegeben.

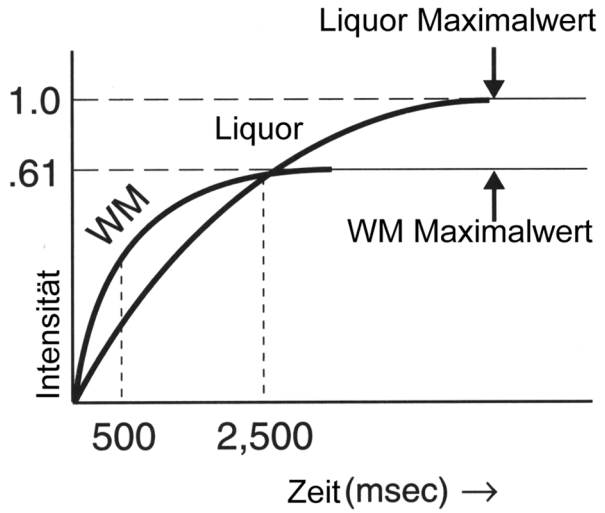

Abbildung 43: Relaxationskurven für Liquor und weiße Gehirnmasse. Hier wurde auch die Protonendichte berücksichtigt [HASH97]

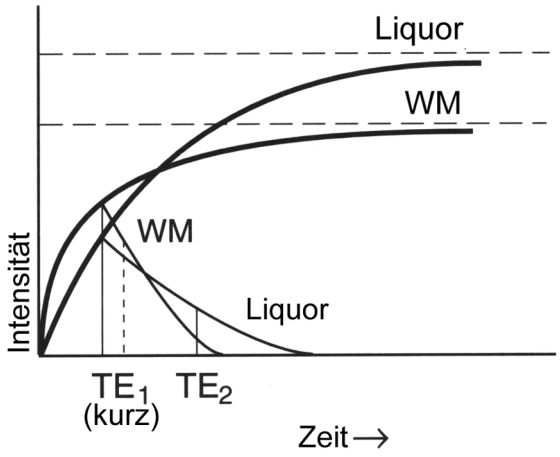

Abbildung 44: Signalintensität für Liquor und weiße Gehirnmasse mit verschiedenen Echozeiten [HASH97]

In Abbildung 43 sieht man den Signalverlauf der longitudinalen Relaxation für die weiße Gehirnmasse und Liquor. Liquor hat auf Grund der höheren Protonendichte einen höhere Maximalwert. An Hand von Abbildung 43 kann man den Zusammenhang zwischen dem Geräteparameter T_R und dem Signalverhalten der beiden Substanzen erklären. Wählt man ein kurzes T_R, hat die weiße Gehirnmasse wegen des kürzeren T_1-Wertes ein größeres Signal als Liquor, für lange T_R hingegen liefert auf Grund der Protonendichte Liquor das größere Signal. Nun kommt es noch auf die Einstellung für T_E an.

Abbildung 44 zeigt das Signalverhalten für kurzes T_R und zwei verschiedene T_E-Werte. Da Liquor ein langes T_2 hat, ist die transversale Relaxationskurve flacher, die Signalintensitäten der beiden Substanzen schneiden sich kurz nach T_{E1}. Für langes T_E (T_{E2}) ergibt sich ein T_2-Kontrast, für kurzes T_E dagegen ein T_1-Kontrast.

Für langes T_R liefert T_{E1} ein protonengewichtetes Bild, für T_{E2} dagegen ein stärker T_2-gewichtetes Bild.

Aus der Spin-Echo-Sequenz sind sowohl T_1 als auch T_2 berechenbar. Zur Berechnung von T_1 sind mehrere Aufnahmen mit unterschiedlichem T_R und konstanten T_E notwendig, zur T_2-Berechnung benutzt man entweder eine Turbo-Spin-Echo-Sequenz (Kapitel 3.5.3) oder variiert T_E bei konstantem T_R. Gute Ergebnisse erreicht man bei der T_1-Messung dann, wenn T_R möglichst stark geändert wird, bei der T_2-Bestimmung muß man die T_E-Zeit stark variieren.

3.3.2 Schnellbildtechniken Teil I (Gradientenechotechniken)

Die bisher beschriebenen konventionellen Techniken haben alle Repititionszeiten im Sekundenbereich. Da bei der konventionellen Bilgebung für jede Bildzeile ein Repititionszyklus benötigt wird, ergeben sich Akquisitionszeiten von mehreren Minuten. Eine Bilderstellung z.B. mit angehaltenem Atem ist damit noch nicht möglich.

Schon seit vielen Jahren versucht man daher auch die Meßzeiten zu reduzieren, um Bewegungsartefakte zu verringern. Die meisten dieser Techniken wie z.B. die von Mansfield entwickelte Echoplanar-Technik erlauben Bilderzeugung im Bereich der T_2-Zeit. Nachteil dabei ist allerdings das im Vergleich zu konventionellen Techniken reduzierte Signal-zu-Rausch-Verhältnis.

Seither wurde eine Vielzahl von Verfahren entwickelt, die alle versuchen, die Akquisitionszeit bei möglichst guter Bildqualität zu reduzieren. Eine Übersicht soll hier nochmals der Sequenzbaum (Abbildung 32, Seite 24) bieten.

Eine Schnellbildtechnik, die auf der Spin-Echo-Sequenz aufbaut und auch in der Simulation implementiert ist die Turbo-Spin-Echo-Sequenz (TSE). Zum Verständnis dieser Sequenz sind weitere Grundlagenkenntnisse nötig, die in den Kapiteln 3.4 und 3.5 erläutert werden. Die Turbo-Spin-Echo-Sequenz selbst ist im Kapitel 3.5.3 beschrieben.

Bisher haben wir nur Sequenzen mit Auslenkwinkeln von 90° und 180° betrachtet. Wenn man bei diesen Sequenzen die Repititionszeit zu stark verkürzt, kann die Magnetisierung nicht genügend relaxieren, wie auch aus Abbildung 45 ersichtlich ist.

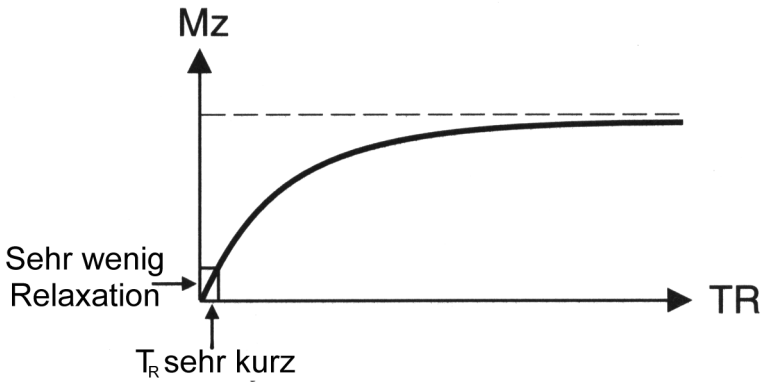

Abbildung 45: Relaxation nach 90°-Impuls mit sehr kurzer Relaxationszeit [HASH97]

Eine genügend große *z*-Komponente der Magnetisierung ist aber Voraussetzung für eine sinnvolle Bilderzeugung. Idee bei den im weiteren beschrieben Gradientenechosequenzen ist daher, die *z*-Magnetisierung durch kleinere Auslenkwinkel ($\alpha < 90°$) weniger stark abzuschwächen. Aus anderen Gründen, die im folgenden Kapitel beschrieben werden, vermeidet man auch den 180°-Refokussierungsimpuls.

Den Gradiententechniken widmen wir das erste Teilkapitel zum Thema Schnellbildtechniken. Eine der grundlegenden Techniken „GE-gespoilt“ wird ausführlich erläutert, andere GE-Sequenzen sollen kurz erwähnt werden. Eine gute Übersicht und weitere Literaturverweise finden sich in [SIEM97], interessant sind auch die Ausführungen dazu in [HASH97].

3.3.2.1 Gradientenechosequenzen

Durch die Verwendung von kleinen Auslenkwinkeln ist es möglich, die T_R-Zeiten zu verkürzen. Ein Problem bei konventionellen SE und IR-Techniken ist, daß nur Repititionszeiten im Bereich der Relaxationszeit T_1 sinnvoll sind. Verkürzt man die T_R-Zeiten bei konventionellen Techniken noch weiter, wird der *z*-Magnetisierungsverktor mit immer kleineren Repititionszeiten immer kleiner, denn die longitudinale Relaxation erfolgt auf Grund der kurzen Repititionszeit nur unzureichend.

Ein kleiner Auslenkwinkel ($\alpha < 90°$) hingegen führt auch schon bei sehr kurzen T_R-Zeiten zu einer großen Gleichgewichts-*z*-Magnetisierung. Im Gegensatz zum 90°-Impuls bei Saturation-Recovery oder Inversion-Recovery wird die longitudinale Magnetisierung nicht auf Null reduziert. Durch den kleinen Auslenkwinkel wird die *z*-Magnetisierung nur geringfügig reduziert und die Wartezeit auf eine große *z*-Magnetisierung verkürzt sich erheblich. Die *z*-Magnetisierung wiederum ist ausschlaggebend für ein genügend großes Signal.

In Abbildung 46 sieht man auf der linken Seite eine konventionelle Spin-Echo-Technik, die einen 180°-Impuls zur Erzeugung eines Spin-Echos verwendet, auf der rechten Seite eine Gradientenechotechnik mit einem Auslenkwinkel unter 90°. Die Abbildung zeigt den Signalvorteil der Gradientenechotechniken bei sehr kurzen T_R-Zeiten. Für größere T_R ist durch die Auslenkung α<90°

die M_{xy}-Komponente des Magnetfeldes zwar kleiner als bei Spin-Echo-Sequenzen, da $M_{xy} = M_z \cdot \sin\alpha$ (obere Abbildung). Bei sehr kurzen Repititionszeiten wird der *z*-Magnetisierungsverktor bei Spin-Echo-Sequenzen sehr klein, bei Gradientenechosequenzen ist M_z dagegen noch so groß, daß der oben beschriebene Signalverlust überkompensiert wird (untere Abbildung).

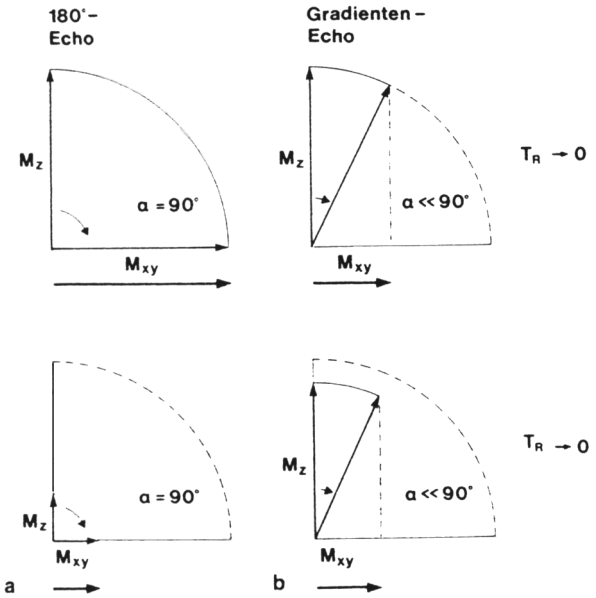

Abbildung 46: Abhängigkeit der Signalintensität vom Auslenkwinkel für sehr kurze Repititionszeiten ($T_R \rightarrow 0$) [LISS90]

Nun könnte man auf die Idee kommen, als Pulssequenzen konventionelle Spin-Echo- und Inversion-Recovery-Sequenzen mit kurzen T_R-Zeiten mit einem kleinen Auslenkwinkel zu kombinieren. Von Nachteil wäre jedoch, daß der 180°-Rephasierungsimpuls nicht nur die Spins in *xy*-Ebene rephasiert, sondern auch die verbleibende z-Magnetisierung invertiert. Dadurch wäre der Zeitgewinn durch die kleinen Auslenkwinkel wieder dahin, denn durch die Invertierung der *z*-Magnetisierung würde die longitudinale Relaxation sich stark verlängern. Die Relaxation würde sogar länger dauern, als die ursprüngliche Relaxation mit 90°-Impuls.

Aus diesem Grund ist es nötig, den Rephasierungsimpuls bei den Gradientenechotechniken zu vermeiden. Die Alternative zum Erzeugen eines Spin-Echos besteht darin, statt des Echos den FID zu vermessen. Leider tritt dieser zu einem ungünstigen Zeitpunkt auf, denn er erfolgt direkt nach der Einstrahlung des 90°-Impulses. Das Abklingen des Impulses und die Gradientenschaltungen brauchen jedoch eine gewisse Zeit, eine Messung direkt zum Beginn des FID ist damit nicht möglich. Daher wird der FID absichtlich dephasiert und zu einem späteren Zeitpunkt wieder rephasiert. Dieser Zeitpunkt ist günstiger für die Messung. In [HASH97] wird daher auch der Name „Gradient-Recall-Echo“ (GRE) für die Gradienten-Echo-Technik verwandt. Um die Spins mittels eines Kompensationsgradienten zu rephasieren, nutzt man wieder die Lamorbeziehung

$$\omega = \gamma \cdot B \tag{12}$$

mit

ω	Resonanzfrequenz
γ	Gyromagnetisches Verhältnis
B	Magnetische Flußdichte.

Die Resonanzfrequenz der Spins ist also linear abhängig von der magnetischen Flußdichte. Durch die Überlagerung mit einem Gradienten präzedieren die Spins leicht unterschiedlich, da die lokale Magnetfeldstärke leicht unterschiedlich ist.

Abbildung 47 zeigt die Gradientenschaltung für die Gradientenechosequenz. Zunächst wird ein negativer Gradient geschaltet um den FID wie beschrieben zu dephasieren, dann ein positiver Gradient mit doppelter Fläche. Die Spins sind also zum Zeitpunkt *2t* wieder in Phase.

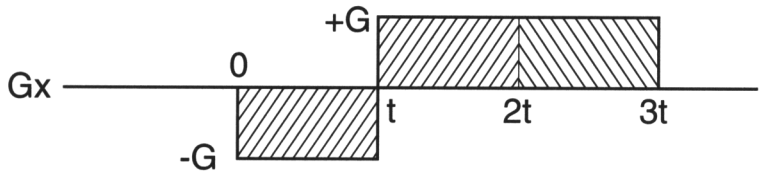

Abbildung 47: Gradientenschaltung für die Gradientenechosequenz

Die Schaltung von Gradientenfeldern kompensiert zwar Dephasierungen der Spins, die durch die Messung entstehen (auch die Ortskodierung verwendet wie schon erwähnt Gradientenfelder), jedoch nicht die Dephasierungen durch Magnetfeldinhomogenitäten der Anlage, durch Suszeptibilität und chemische Verschiebung. Die Suszeptibilität ist ein physikalischer Effekt, der an Gewebsübergängen für lokale Magnetfeldinhomogenitäten sorgt und damit für schnelleren transversalen Zerfall. Unter chemischer Verschiebung versteht man die leicht unterschiedliche Resonanzfrequenzen von Wasserstoffprotonen in Wasser oder Fettbindung. Für nähere Informationen empfiehlt sich [LISS90]. Die Verkürzung der T_2-Zeit errechnet sich insgesamt nach:

$$R_{T_{2^*}} = R_{T_2} + R_M + R_S + R_C \tag{13}$$

mit

M	Magnetfeldinhomogenität	$R_M = \frac{1}{T_M}$
S	Suszeptibilität	$R_S = \frac{1}{T_S}$
C	Chemische Verschiebung	$R_C = \frac{1}{T_C}$
$R_{T_{2^*}}$	Relaxivity	$R_{T_2^*} = \frac{1}{T_2^*}$
R_{T_2}	Kehrwert von T_2	$R_{T_2} = \frac{1}{T_2}$

T_M, T_S und T_C sind Zeitkonstanten, die den Einfluß von Magnetfeldinhomogenität, Suszeptibilität und der chemischen Verschiebung auf T_2^* beschreiben.

Nach einem Rechenbeispiel aus [LISS90] ergibt sich bereits für eine Magnetfeldinhomogenität über einen Voxel von 0,5 ppm (0,00005 %) eine Abnahme einer T_2-Zeit von 100 ms auf T_2^* = 28,6 ms !

In modernen Kernspintomographen beträgt die Inhomogenität des Feldes aber nur ca. 5 ppm über ein Kugelvolumen von 50 cm Durchmesser. Unter Annahme einer linearen Magnetfeldinhomogenität und Verwendung einer 256·256-Matrix ergibt sich eine Magnetfeldinhomogenität von 0,02 ppm/Voxel. Damit reduziert sich die T_2-Zeit nur noch von 100 ms auf T_2^* = 92 ms, also tritt hier nur eine kleine Verfälschung durch die Gerätetechnik an sich auf. Chemische Verschiebung und Suszeptibilität haben hier größeren Einfluß auf die Verkürzung der T_2-Zeit. Dieses Beispiel zeigt aber, das vergleichbare Bilder auf unterschiedlichen Anlagen nur mit sehr hohen Anforderungen an die Technik erreicht werden können.

Der Bildcharakter bei Gradientenechosequenzen ist ebenfalls anders als bei Spin-Echo-Sequenzen. Die Wichtung der erzeugten Bilder ist hier neben der Repititionszeit und der Echozeit auch noch vom

Auslenkwinkel α abhängig. Je nach verwendetem Gradientenechoverfahren hängt die Bilderzeugung entweder vom Quotienten $\frac{T_1}{T_2^*}$, von T_1, von T_2 oder von T_2^* ab.

Die zahlreichen Gradientenechoverfahren lassen sich im Prinzip auf drei grundlegende Verfahren zurückführen. Je nach Herstellerfirma unterscheiden sich jedoch die Akronyme für die einzelnen Verfahren. Tabelle 5 bietet eine Übersicht. Dabei stehen in der ersten Zeile die firmenneutralen Bezeichnungen.

Gerätehersteller	$GE_{gespoilt}$ T_1-betont	$GE_{refokussiert}$ T_1/T_2^*	$GE_{gespiegelt/refokussiert}$ T_2-betont
Diasonics	PFI		
Elscint	FE		
Fonar	Gradienten-Echo		
General Electric		GRASS	
Philips	FFE		
Picker	Partial Saturation Recovery	FAST	CE-FAST
Siemens	FLASH	FISP	PSIF

Tabelle 5: Gradientenverfahren. Akronyme einiger Gerätehersteller [LISS90]

Das gemeinsame dieser drei Techniken ist, daß sie alle Kleinwinkelanregungen verwenden und mit Hilfe von Gradientenschaltungen den 180°-Refokussierungsimpuls ersetzen. Damit ergeben sich auch ähnliche Eigenschaften in Bezug auf die *z*-Magnetisierung. Durch die kurzen T_R-Zeiten verbleibt nach der Messung eine Restmagnetisierung in der *xy*-Ebene. Die Unterschiede der Techniken bestehen darin, wie mit der nach der Anregung und Messung verbleibenden *xy*-Magnetisierung verfahren wird. Bei $GE_{gespoilt}$ wird die *xy*-Magnetisierung durch einen Spoilgradienten verworfen, damit ist nur noch die *z*-Magnetisierung und somit die T_1-Zeit für die Bildgebung relevant (Tabelle 5). Bei $GE_{refokussiert}$ wird die Restmagnetisierung dagegen am Ende der Messung mit einem Refokussierungsgradienten rephasiert und damit für die nächste Anregung maximiert. Aus diesem Grund fließen in diese Sequenz T_1 und T_2^* ein. Bei der Herleitung der Formel wird klar, daß diese Sequenz vom Quotienten T_1/T_2^* abhängig ist. Bilder in Abhängigkeit von nur einem der Parameter sind bei dieser Sequenz nicht möglich.

Während $GE_{gespoilt}$ und $GE_{refokussiert}$ ein Gradientenecho im eigentlichen Sinne erzeugen, nutzt $GE_{gespiegelt/refokussiert}$ die Tatsache, daß jeder HF-Impuls eine Doppelfunktion hat, nämlich HF-Einstrahlung und Spininvertierung. Die Echozeit ist hierbei länger als die Repititionszeit. Mit dieser Sequenz wird also ohne direkten Invertierungsimpuls dennoch ein Spin-Echo erzeugt. Es werden somit T_2-betonte Bilder erzeugt.

Die Signalintensitäten lassen sich für diese Techniken wieder berechnen. Die Formeln für $GE_{gespoilt}$ und $GE_{refokussiert}$ kann man aus (14) ableiten.

$$I \approx N(H) \cdot M_0 \cdot \frac{c \cdot (1-a) \cdot \sin\alpha}{(1-a \cdot \cos\alpha) + b \cdot (\cos\alpha - a)} \qquad (14)$$

mit

I	Signalintensität
$N(H)$	Protonendichte
M_0	Thermische Gleichgewichtsmagnetisierung
a	e^{-T_R/T_1}
b	e^{-T_R/T_2^*}
c	e^{-T_E/T_2^*}
α	Auslenkwinkel

Bei der Sequenz $GE_{gespoilt}$ wird die *xy*-Magnetisierung wie gesagt verworfen. Dieser Zustand träte beim transversalen Zerfall für sehr große T_R auf. Damit vereinfacht sich die Formel für $GE_{gespoilt}$ zu

$$I \approx N(H) \cdot M_0 \cdot \frac{c \cdot (1-a) \cdot \sin\alpha}{(1-a \cdot \cos\alpha)} \tag{15}$$

denn es gilt $b \to 0$ für sehr große T_R. Damit ist die Formel für $GE_{gespoilt}$ von T_R, T_E und α abhängig. Weiter vereinfachen läßt sich die Formel mit Echozeiten $T_E << T_2^*$. Da *c* dann gegen 1 konvergiert lautet die vereinfachte Formel

$$I \approx N(H) \cdot M_0 \cdot \frac{(1-a) \cdot \sin\alpha}{1-a \cdot \cos\alpha} \tag{16}$$

Die Signalintensität wird groß, wenn der Nenner in der Gleichung möglichst klein wird. Also errechnet sich der optimale Auslenkwinkel nach

$$\alpha_{opt} = \arccos e^{-T_R/T_1} \tag{17}$$

Abbildung 48 zeigt die Signalintensität für $GE_{gespoilt}$ in Abhängigkeit vom Auslenkwinkel α, T_R und T_1.

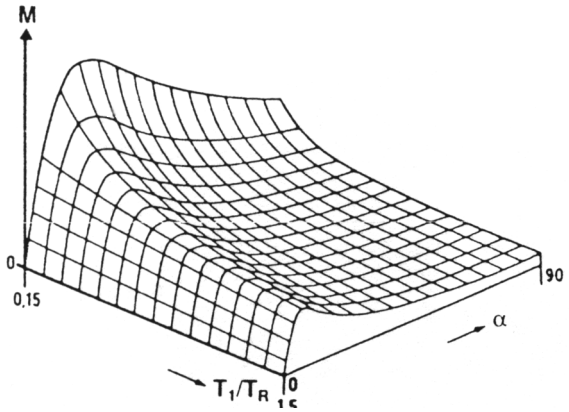

Abbildung 48: Maximale Signalintensität bei gespoilter Gradientenechotechnik [LISS90]

Die größten Signalintensitäten ergeben sich stets für kleine Auslenkwinkel. Bei konstantem Auslenkwinkel nimmt die Signalintensität für längere T_R zu.

Allerdings bedeutet eine hohe Signalintensität nicht unbedingt einen hohen Gewebekontrast, was Abbildung 49 zeigt. Dort ist der Kontrast zwischen weißer Hirnsubstanz und Liquor cerebrospinalis in Abhängigkeit vom Auslenkwinkel dargestellt. Bei $GE_{refokussiert}$ (Abbildung 49a) nimmt der Kontrast von 0° an zu, erreicht bei 90° sein Maximum und fällt bis 180° wieder auf Null ab. Bei $GE_{gespoilt}$ erreicht der Kontrast bei ca. 5° ein relatives Maximum. Über 5° fällt der Kontrast ab, wechselt dann das Vorzeichen und erreicht für 30° ein absolutes Maximum. Über 30° wird der Kontrast wieder kleiner. Der Kontrast bei 30° ist etwa viermal so groß wie der Kontrast bei 5°. Die Signalintensität ist allerdings, wie in Abbildung 49b zu erkennen ist, beim kleineren Winkel wesentlich größer.

Auch die Formel für $GE_{refokussiert}$ läßt sich aus (14) weiter vereinfachen. Für die klinisch relevanten Parametereinstellungen gilt die Forderung $T_R << T_1$ und $T_R << T_2^*$. Da die Quotienten T_R/T_1 und T_R/T_2^* gegen Null konvergieren, ersetzt man *a* und *b* durch die beiden ersten Glieder der Reihenentwicklung für *e*-Funktionen ($e^{-T_R/T_1} = 1 - T_R/T_1, e^{-T_R/T_2} = 1 - T_R/T_2^*$) und erhält

$$I \approx N(H) \cdot M_0 \cdot \frac{\sin\alpha}{1 + T_1 / T_2^* - (T_1 / T_2^* - 1) \cdot \cos\alpha} \tag{18}$$

Der einzige Gewebeparameter, von der die Signalintensität hier abhängt, ist der Auslenkwinkel α, die Echozeit und die Repititionszeit haben keinen Einfluß. Desweiteren hängt die Signalintensität vom Quotienten T_1/T_2^* ab, aber nicht von den Absolutwerten dieser Größen.

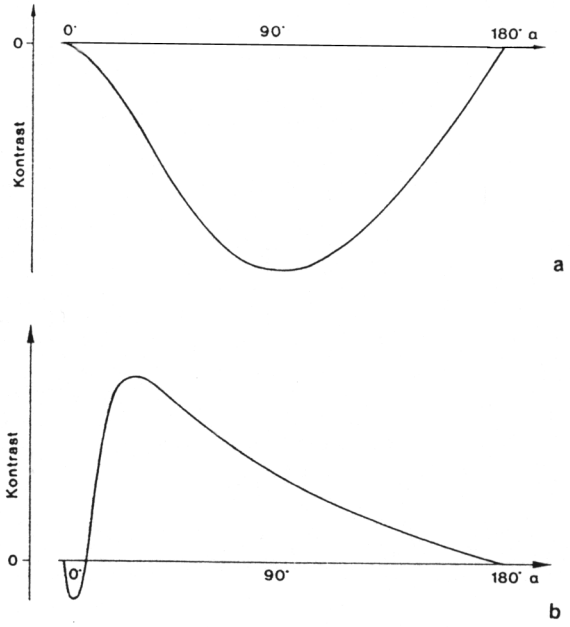

Abbildung 49: Kontrast zwischen weißer Hirnsubstanz und Liquor cerebrospinalis. Oben: Gradientenecho refokussiert, unten Gradientenecho gespoilt [LISS90]

Der Geschwindigkeitsvorteil von Gradientenechosequenzen erklärt sich wie schon erwähnt daraus, daß auf Grund der kleinen Auslenkwinkel schneller eine große *z*-Magnetisierung erreicht wird. Was Tabelle 6 nochmals zeigt. Je kleiner der Auslenkwinkel, desto kürzer ist die benötigte T_R-Zeit für eine große *z*-Magnetisierung.

α [°]	T_1 = 200 ms	T_1 = 500 ms	T_1 = 2000 ms	% der max. Signalintensität
10	50	125	500	17
20	150	375	1500	34
30	340	850	3400	64
40	470	1175	4700	87
50	600	1500	6000	100

Tabelle 6: Repititionszeitbedarf in ms für eine 95%ige Sättigung der *z*-Magnetisierung [LISS90]

Nun muß sich aber erst einmal eine Gleichgewichtsmagnetisierung einstellen. Dafür sind erst mehrere Impulse nötig, bis der Gleichgewichtszustand erreicht ist. Abbildung 50 veranschaulicht dies.

Bei jeder Auslenkung wird die *z*-Magnetisierung um den Faktor $\cos\alpha$ reduziert. Während der Repititionszeit T_R tritt eine Relaxation auf. Nach einigen Impulsen gleichen sich Verlust an *z*-Magnetisierung durch die HF-Impulse und Relaxation aus. Eine Gleichgewichtsmagnetisierung hat sich nun eingestellt. Ist T_R allerdings zu klein, wird die *z*-Magnetisierung durch die Anregungen immer weiter reduziert, während T_R findet kaum Relaxation statt. M_z wird immer kleiner und nähert sich dem Wert 0.

Die Größe der Gleichgewichtsmagnetisierung hängt neben T_R aber auch vom Auslenkwinkel ab, was in Abbildung 51 zu erkennen ist.

Bei größerem Auslenkwinkel α nimmt die Gleichgewichtsmagnetisierung ab (oben die *z*-Magnetisierung für einen kleinen Auslenkwinkel α_1, unten *z*-Magnetisierung für einen größeren Auslenkwinkel α_2). Für längere T_R nimmt die Gleichgewichtsmagnetisierung zu (links kurzes T_R, rechts langes T_R).

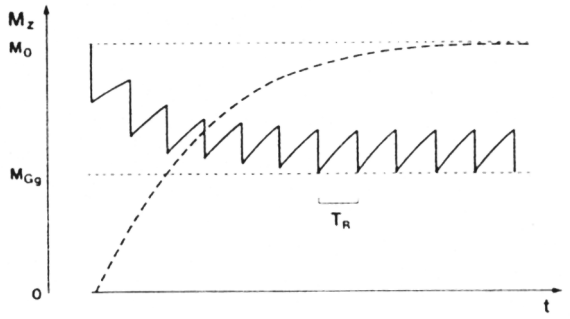

Abbildung 50: Einstellung der Gleichgewichtsmagnetisierung bei Kleinwinkelanregung [LISS90]

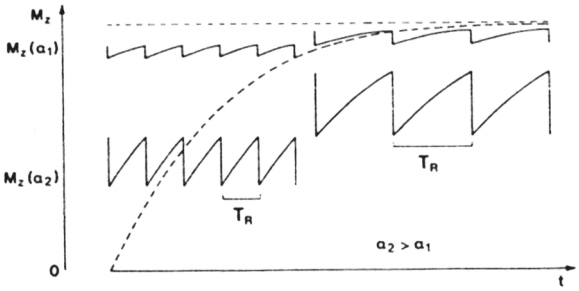

Abbildung 51: Gleichgewichtsmagnetisierung in Abhängigkeit von Auslenkwinkel und Repititionszeit [LISS90]

Da aber die Signalintensität über die Messung möglichst konstant sein muß, benötigt man vor der eigentlichen Messung einige sogenannte Präparationszyklen. Hier gilt, daß lange T_R-Zeiten weniger Präparationszyklen brauchen. Größere Auslenkwinkel benötigen dagegen mehr Präparationszyklen.

Ein Signalvorteil gegenüber einer Spin-Echo-Sequenz tritt nur bei sehr kurzen T_R auf.

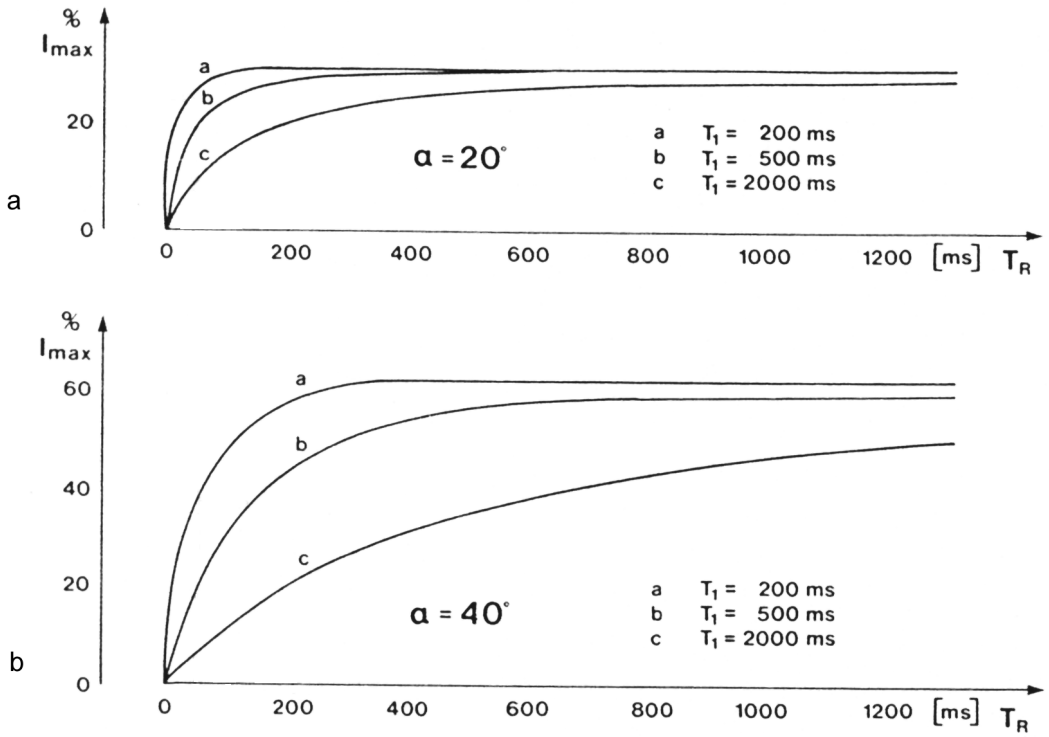

Abbildung 52: Abhängigkeit der Signalintensität einer gespoilten Gradientenechosequenz von Auslenkwinkel und Repititionszeit [LISS90]

Abbildung 52 zeigt die Signalintensität für drei Substanzen (a, b, c) mit verschiedenen T_1-Zeiten in Abhängigkeit von der Repititionszeit und dem Winkel. Die Signalintensität einer Anregung mit einem Auslenkwinkel von $\alpha = 20°$ ist zwar deutlich gegenüber der beim FID maximal möglichen Intensität I_{max} reduziert, jedoch ist das Signal bei kurzen Repititionszeiten stärker als bei einer Spin-Echo-Sequenz. Dieser Vorteil ist jedoch auf kurze T_R beschränkt. Je größer der Auslenkwinkel, desto größer ist der Bereich des Signalvorteils gegenüber Spin-Echo, allerdings ist der Signalgewinn kleiner, wie Tabelle 7 zeigt.

α [°]	T_1 = 200 ms	T_1 = 500 ms	T_1 = 2000 ms	Maximaler Signalgewinn
10	0-35	0-85	0-350	11,4:1
20	0-70	0-180	0-710	5,7:1
30	0-150	0-380	0-1530	2,8:1
40	0-260	0-660	0-2630	1,7:1
50	0-480	0-1220	0-4870	1,2:1
-	-	-	-	1:1

Tabelle 7: Repititionszeitbereich in ms mit Signalvorteil für eine gespoilte Gradientenechotechnik gegenüber Spin-Echo [LISS90]

Ein Aspekt, der auch bei der gespoilten Gradientenechotechnik ins Gewicht fällt, ist der transversale Zerfall mit T_2^*. Zwar ist direkt nach dem HF-Impuls die Signalintensität neben den Geräteparametern nur von T_1 abhängig, da die transversale Magnetisierung der vorhergehenden Messung verworfen wird, zur Echozeit T_E ist aber bereits ein deutlicher Signalzerfall zu erwarten. Daher stellt die Gradientenechotechnik hohe Anforderungen an die Geräte, da schon geringe Inhomogenitäten die Bildgebung merklich verzerren.

Durch den fehlenden 180°-Impuls gegenüber Spin-Echo sind aber viel kürzere Echozeiten möglich, dadurch kann der Einfluß von T_2^* wiederum klein gehalten werden. Dadurch ist eine stärkere T_1-Betonung als bei der Spin-Echo-Technik möglich. Durch die kurzen Echozeiten sind auch mehrere Schichten gleichzeitig akquirierbar (Vielschichttechnik).

Für die Erzeugung parametergewichteter Bilder gelten im wesentlichen für die Gradientenechotechnik die gleichen Grundsätze wie für Spin-Echo. Eine T_1-Gewichtung kann mit einer Verkürzung von T_E und T_R erreicht werden. Eine Protonendichtewichtung wird durch eine Verlängerung von T_R und eine Verkürzung von T_E erzeugt. Eine zunehmende T_2^*-Wichtung ist schließlich durch eine Verlängerung von T_R und T_E möglich. Allerdings muß hier auch noch der Auslenkwinkel α berücksichtigt werden.

Zusammenfassend kann man sagen, daß die Gradiententechniken bei schneller Akquisition Vorteile gegenüber den konventionellen Verfahren bieten. Die Signalintensität ist jedoch nur für kurze Repititionszeiten der Spin-Echo-Technik überlegen, für längere T_R ist Spin-Echo besser geeignet. Das Problem bei den besprochenen gespoilten Gradientenechotechniken ist die Abhängigkeit des transversalen Zerfalls von T_2^*.

Informationen zu weiteren Gradientenechotechniken findet man in [HASH97] oder [SIEM97].

3.4 Ortskodierung

Die bisherigen Betrachtungen bezogen sich auf die gleichzeitige Anregung des gesamten Untersuchungsobjektes. Wir haben somit auch ein Resonanzsignal betrachtet, daß aus dem gesamten Meßobjekt stammte und nicht einem speziellen Ort zugeordnet werden konnte. Die Ortskodierung, d.h. die Zuordnung des gemessenen Resonanzsignals zu bestimmten Bereichen im untersuchten Objekt bzw. im resultierenden Bild des Objektes, ist aber eine der entscheidenden Komponenten in der Kernspintomographie.

Zwar müssen wir die Ortskodierung für unsere Simulationszwecke nicht unbedingt simulieren – denn wir erhalten bereits ein ortskodiertes Bild aus dem Kernspintomographen auf dem wir unsere Simulation aufbauen – allerdings trägt das Verständnis der Ortskodierung erheblich dazu bei, daß auch die in Kapitel 3.5 beschriebenen Modifikationen des k-Raums verstanden werden.

Das Problem der Ortskodierung ist, daß bei den bisherigen Betrachtungen alle Spins mit der gleichen Frequenz präzedieren und somit alle eine Resonanzstrahlung gleicher Frequenz und Wellenlänge aussenden. Es müssen also Möglichkeiten gefunden werden, dem Signal Informationen zu entnehmen – diese können vorher aufgeprägt worden sein – um einzelnen Signalbestandteilen einen Entstehungsort zuzuordnen.

Da die Wellenlänge der eingestrahlten elektromagnetischen Strahlung und damit auch die der Resonanzstrahlung beträchtlich über der Gesamtgröße des Untersuchungsobjektes liegen, können die aus der Resonanzstrahlung gewinnbaren Ortsinformationen nicht zur Ortskodierung verwendet werden. Dazu ein Zahlenbeispiel: Die Wellenlänge von elektromagnetischer Strahlung läßt sich durch

$$\lambda = \frac{c}{\nu} \tag{19}$$

bestimmen, mit

- λ Wellenlänge,
- c Lichtgeschwindigkeit ($3 \cdot 10^8$ m/s),
- ν Frequenz.

In der Kernspintomographie wird zur Resonanzanregung bei einem externen Magnetfeld mit einer Flußdichte von 1,0 Tesla eine Anregungsfrequenz von 42,6 MHz benötigt. Das entspricht einer Wellenlänge von 7,1 m, also beträchtlich mehr als beispielsweise die Größe des menschlichen Kopfes.

Es müssen also andere physikalische Prinzipien zur Ortskodierung angewendet werden. Wie schon in Kapitel 3.2 erklärt, tritt eine Resonanzanregung der Spins nur auf, wenn die Frequenz der eingestrahlten Hochfrequenzstrahlung mit der Präzessionsfrequenz der Spins übereinstimmt. Die Präzessionsfrequenz ist wiederum von der Feldstärke des externen Magnetfeldes abhängig:

$$\nu = \frac{\gamma \cdot B}{2\pi} \tag{20}$$

mit

- ν Lamor- bzw- Präzessionsfrequenz in Radianten pro Zeiteinheit,
- γ Gyromagnetisches Verhältnis,
- B Magnetische Flußdichte.

Im Fall des absolut homogenen Magnetfeldes nehmen bei der Einstrahlung einer HF-Strahlung der Frequenz $\nu_0 = \nu$ alle Spins an der Resonanzanregung teil.

Überlagert man dem homogenen externen Magnetfeld B_0 nun aber ein sich linear (zum Beispiel in Richtung der z-Achse) veränderndes Magnetfeld, so erhält man ein Magnetfeld B, das auf jedem Punkt der z-Achse einen unterschiedlichen Wert annimmt, der jedoch für den entsprechenden Punkt auf der z-Achse eindeutig ist. In Abbildung 53 ist dies dargestellt. Eine Resonanzanregung kann dann nur noch für einen einzigen Ort auf der z-Achse stattfinden, nämlich genau dort, wo $B=B_0$ gilt. Somit läßt sich bei bekanntem Zusammenhang zwischen B und der z-Koordinate eindeutig das empfangene Resonanzsignale dem Ort der Entstehung zuordnen.

Verwendet man nur ein einziges sich linear veränderndes Magnetfeld, so wird aus der Probe eine Ebene herausgeschnitten, in der die Resonanzbedingung erfüllt ist. Verwendet man dagegen zwei entsprechend geschickt geschaltete (zueinander senkrechte) Gradientenfelder, so ist die Resonanzbedingung $B=B_0$ auf einer Geraden in der Probe erfüllt. Beim Übergang zu drei Gradientenfeldern (x-, y-, z-Richtung) kann der Ort der Erfüllung der Resonanzbedingung als Punkt im dreidimensionalen Probenvolumen eindeutig festgelegt werden.

In der Anwendung sind Gradientenfelder üblich, die nur ca. 2 Promille der Stärke des externen Magnetfeldes aufweisen. Bei der Abbildung eines Volumens von 25 cm Durchmesser und einer Bildauflösung von 256^2 Bildpunkten ergibt sich eine Magnetfeldänderung zwischen zwei benachbarten Bildpunkten von $8 \cdot 10^{-6}$ B_0 [LISS90]. Dieses Beispiel zeigt, wie exakt die magnetische Induktion B_0 und die Anregungsfrequenz ν_0 aufeinander abgestimmt werden müssen und wie eng die Resonanzbedingung beschränkt ist.

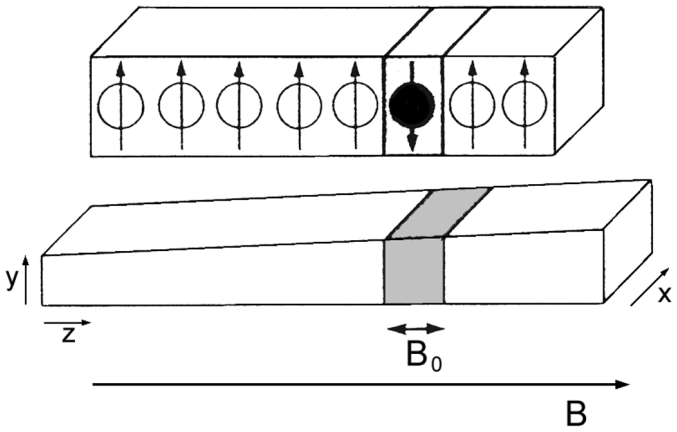

Abbildung 53: Überlagerung eines Gradientenfeldes über das externe Magnetfeld [LISS90]

Insgesamt können 4 Verfahren zur Ortskodierung benutzt werden. Dies sind die Punkt-, Linien-, Schicht- und Volumentechnik. Alle 4 Verfahren sollen in den folgenden Abschnitten im einzelnen erklärt und analysiert werden. Abbildung 54 veranschaulicht die durch die 4 Techniken selektierten Volumina und Tabelle 8 gibt einen Überblick über das Signal-zu-Rausch-Verhältnis (S/R-Verhälnis) und die Akquisitionszeit der 4 Techniken.

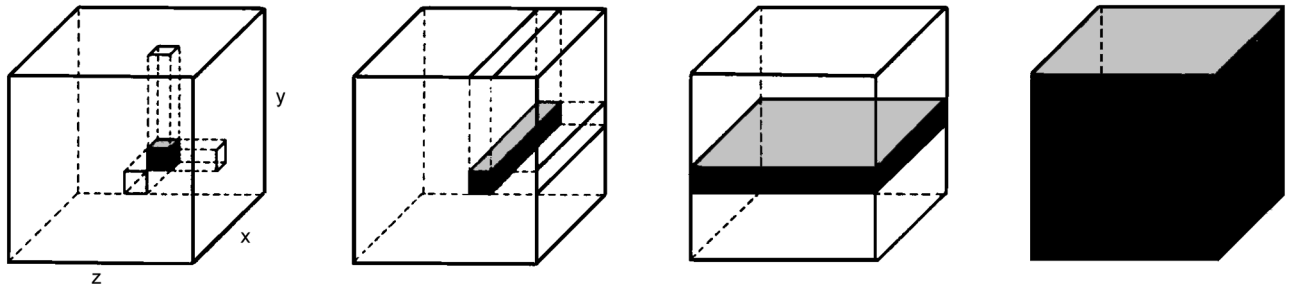

Abbildung 54: Ortskodierungstechniken: Punkt-, Linien-, Schicht- und Volumentechnik (von links) [LISS90]

Bildaufbautechnik	S/R-Verhältnis	Akquisitionszeit/Schicht (n^2 Voxel)	Akquisitionszeit/ Volumen (n^2m Voxel)
Punkt	1	$n^2 \cdot T_R$	$n^2 \cdot m \cdot T_R$
Linien	1	$n \cdot T_R$	$n \cdot m \cdot T_R$
Schicht	$\sqrt{n}$	$n \cdot T_R$	$n \cdot m \cdot T_R$
Volumen	$\sqrt{n \cdot m}$	$n^2 \cdot T_R$	$n \cdot m \cdot T_R$

Tabelle 8: S/R-Verhältnis und Akquisitionszeit für die verschiedenen Bilderzeugungstechniken [LISS90]

3.4.1 Fourieranalyse

Das grundlegende Werkzeug für die Ortskodierung der Signale bei der Kernspintomographie ist die Fourieranalyse. Daher soll diese hier kurz erläutert werden. Der Begriff Fourieranalyse faßt die Fouriertransformation (FT) und die inverse Fouriertransformation (FT^{-1}) zusammen.

Mit Hilfe der FT wird ein Signal über der Zeit in ein Frequenzspektrum zerlegt. Denn oftmals ist es einfacher, im Frequenzbereich statt im Zeit- bzw. Ortsbereich zu arbeiten. Zur Rücktransformation in den Zeitbereich verwendet man dann schließlich die inverse Fouriertransformation.

Nehmen wir also an, wir haben ein Signal $g(t)$ über der Zeit t. Die Fouriertransformation wandelt dieses Signal aus dem Zeitbereich in ein Signal im Frequenzbereich um. Das in den Frequenzbereich transformierte Signal wird mit $G(\omega)$ bezeichnet. Die Transformationsfunktion ist

$$G(\omega) = \int_{-\infty}^{+\infty} g(t) \cdot e^{-i\omega t} dt \tag{21}$$

wobei $\omega=2\pi\nu$ ist und der Term ($e^{-i\omega t}$) einen Vektor repräsentiert, der mit der Winkelfrequenz ω rotiert. Damit erhält man das Spektrum der Frequenzen, die im ursprünglichen Signal *g(t)* vorhanden sind. Wir wollen das anhand zweier einfacher Beispiele veranschaulichen.

Zuerst nehmen wir an, *g(t)* sei gleich $cos(\omega_0 t)$. Wie zu erwarten, liefert die Fouriertransformation dieser Funktion genau einen Peak bei ω_0, denn das ist die einzige Frequenz, die in *g(t)* vorkommt (Abbildung 55). Aufgrund der Symmetrie der FT erhalten wir aber auch einen Peak bei $-\omega_0$, den wir für unsere Betrachtungen aber ignorieren können. Neben der Frequenz repräsentiert der Peak mit seinem Betrag auch die Amplitude der ursprünglichen Kosinus-Funktion.

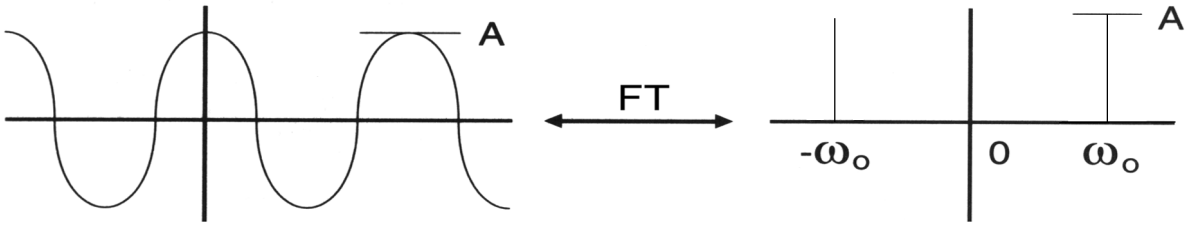

Abbildung 55: Kosinus-Signal (links) und dessen Fouriertransformation (rechts) [HASH97]

Als nächsten schauen wir uns das Signal an, mit dem wir es in der Kernspintomographie meist zu tun haben. Die *sinc* Funktion

$$sinc(\omega_0 t) = \frac{\sin(\omega_0 t)}{\omega_0 t} \tag{22}$$

repräsentiert einen HF-Anregungsimpuls in der Kernspintomographie. Wie aus (Abbildung 56) ersichtlich ist, hat die FT dieser Funktion eine rechteckige Form, was bedeutet, daß die Funktion ein Frequenzspektrum von $-\omega_0$ bis $+\omega_0$ enthält. Dieses Spektrum nennt man auch Bandbreite.

Abbildung 56: Sinc-Funktion (links) und deren Fouriertransformation (rechts) [HASH97]

3.4.2 Punkttechnik

Bei der Punkttechnik verwendet man ein Magnetfeld, so daß die Resonanzbedingung nur in einem kleinen Volumenelement erfüllt ist. Dies kann man zum einen durch Deformation des externen homogenen Magnetfeldes erreichen. Dann muß das zu untersuchende Objekt allerdings durch mechanische Bewegung Punkt für Punkt in dieses kleine Volumenelement bewegt werden. Oder aber man verwendet alternierende (!) Feldgradienten. Dadurch entsteht an allen Orten des zu untersuchenden Objektes ein zeitlich variierendes Magnetfeld. Nur im Schnittpunkt der drei Gradientenfelder ist das Feld zeitlich konstant und erfüllt die Resonanzbedingung.

Beide Möglichkeiten werden allerdings heute nicht mehr verwendet, da der Zeitaufwand für die Akquisition viel zu hoch ist. Da jeder Bildpunkt einzeln abgetastet werden muß, sind n^2 Einzelmessungen notwendig, um eine Schicht zu erfassen.

3.4.3 Linientechnik

Bei der Linientechnik wird eine komplette Bildzeile oder –spalte mit einer Einzelmessung erfaßt. Zur Darstellung eines Bildes mit n^2 Bildelementen sind also nur noch n Einzelmessungen notwendig. Damit reduziert sich der Zeitaufwand gegenüber der Punkttechnik um den Faktor n.

Man erreicht die Auswahl einer Linie aus dem Probenvolumen durch Schaltung zweier zueinander senkrechter Gradientenfelder, z.B. in *y*- und in *z*-Richtung. Dabei kann man entweder wie bei der Punkttechnik erwähnt, zwei alternierende Gradientenfelder verwenden, oder aber man schaltet zwei zeitlich konstante Gradientenfelder. Dabei ist es allerdings wichtig, daß die beiden Felder nacheinander geschaltet werden, denn die gleichzeitige Schaltung hätte eine additive Überlagerung zur Folge, so daß eine quer im Probenvolumen liegende Schicht die Resonanzbedingung erfüllen würde. Abbildung 57 veranschaulicht dies. Links ist die Schichtselektion durch ein geschaltetes Gradientenfeld dargestellt. Es wird eine parallel zur *xz*-Ebene liegende Schicht selektiert. In der Mitte und rechts wird eine quer im Raum liegende Schicht durch gleichzeitige Schaltung von 2 bzw. 3 Gradientenfeldern ausgewählt. Die Lage im Raum kann durch den relativen Beitrag der einzelnen Gradientenfelder beeinflußt werden.

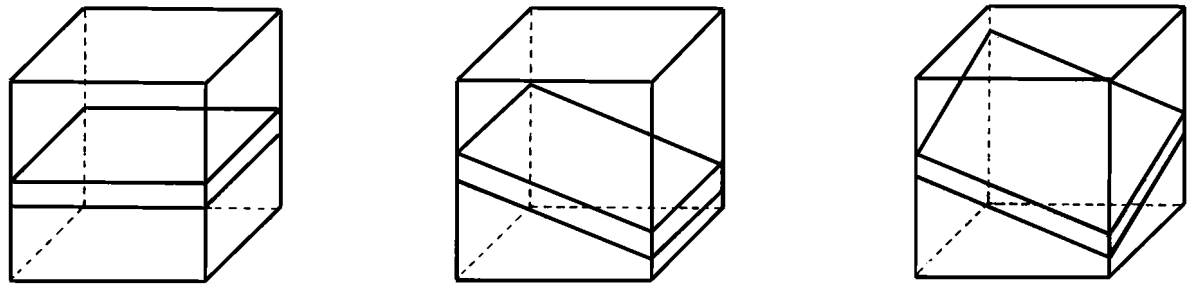

Abbildung 57: Schichtselektion durch das gleichzeitige Schalten von 1, 2 oder 3 Gradienten [LISS90]

Im folgenden soll die Linienselektion anhand der aus Kapitel 3.3.1.3 bekannten Spin-Echo-Technik erläutert werden (Abbildung 58).

Der 90°-HF-Impuls wird bei eingeschaltetem *z*-Gradienten (Schichtselektionsgradient) eingestrahlt. Dadurch werden alle Spins in einer *xy*-Ebene angeregt und um 90° augelenkt. Nach Einstrahlung des 90°-Impulses wird der Schichtselektionsgradient wieder abgeschaltet und der *y*-Gradient angeschaltet. Dieser bewirkt, daß nun alle Spins in einer *xz*-Ebene, die senkrecht zur zuvor ausgewählten Ebene steht, die Resonanzbedingung erfüllen. Durch Einstrahlung des 180°-Impulses werden alle Spins in der nun selektierten Ebene invertiert, also um 180° umgeklappt. Dabei erfahren die Spins, die von dem 90°-Impuls nicht angeregt wurden, eine Umkehr in *z*-Richtung. Die *xy*-Magnetisierungskomponente bleibt dabei 0 ($M_{xy}=0$) und ein T_2-Zerfall dieser Spins ist somit nicht meßbar. Die Spins, die jedoch durch den 90°-Impuls angeregt wurden und deren *xy*-Magnetisierung inzwischen mit der Konstanten T_2^* zerfällt, werden durch den 180°-Impuls rephasiert. Das heißt, die zuvor schneller präzedierenden Spins laufen nun nach der Invertierung den zuvor langsamer laufenden Spins hinterher. Nach genau der gleichen Zeit wie die zwischen 90°-Impuls und 180°-Impuls präzedieren die Spins der selektierten Linie wieder in Phase und erzeugen ein Echo, daß gemessen werden kann.

Die Lamor- und damit auch die Anregungsfrequenz von Wasserstoffprotonen beträgt bei 1 Tesla ca. 42,6 MHz. Nehmen wir der einfachen Rechnung halber einmal an, der Schichtselektionsgradient verläuft von 0,9 Tesla (Fußende) bis 1,1 Tesla (Kopfende). Verwendet man also HF-Impulse von 42,6 Tesla, so wird eine unendlich dünne Schicht in der Mitte des Körpers angeregt. In einer unendlich dünnen Schicht ist aber auch nur eine unendlich kleine Anzahl von Protonen, die ein Signal liefern können, vorhanden. Also muß man eine Schicht mit endlicher Dicke anregen. Dies erreicht man, indem man als HF-Impuls ein Spektrum von Frequenzen verwendet. Nehmen wir an, daß oben beschriebene Gradientenfeld verläuft auf einer Länge von 2 m. Und nehmen wir an, wir wollen eine 2 cm dicke Schicht in der Mitte des Körpers anregen. Dann benötigen wir einen HF-Impuls, der ein

Frequenzspektrum von 42,5574 MHz $(42{,}6\,Mhz\cdot(0{,}9T+(99cm\cdot(1{,}1T-0{,}9T)/200cm))$ bis 42,6426 MHz $(42{,}6\,Mhz\cdot(0{,}9T+(101cm\cdot(1{,}1T-0{,}9T)/200cm))$ enthält. Das abgedeckte Spektrum von 85,2 kHz nennt man Bandbreite. Eine Variation der Schichtdicke erhält man also durch Veränderung der Bandbreite des HF-Impulses oder aber durch Veränderung des Schichtselektionsgradienten.

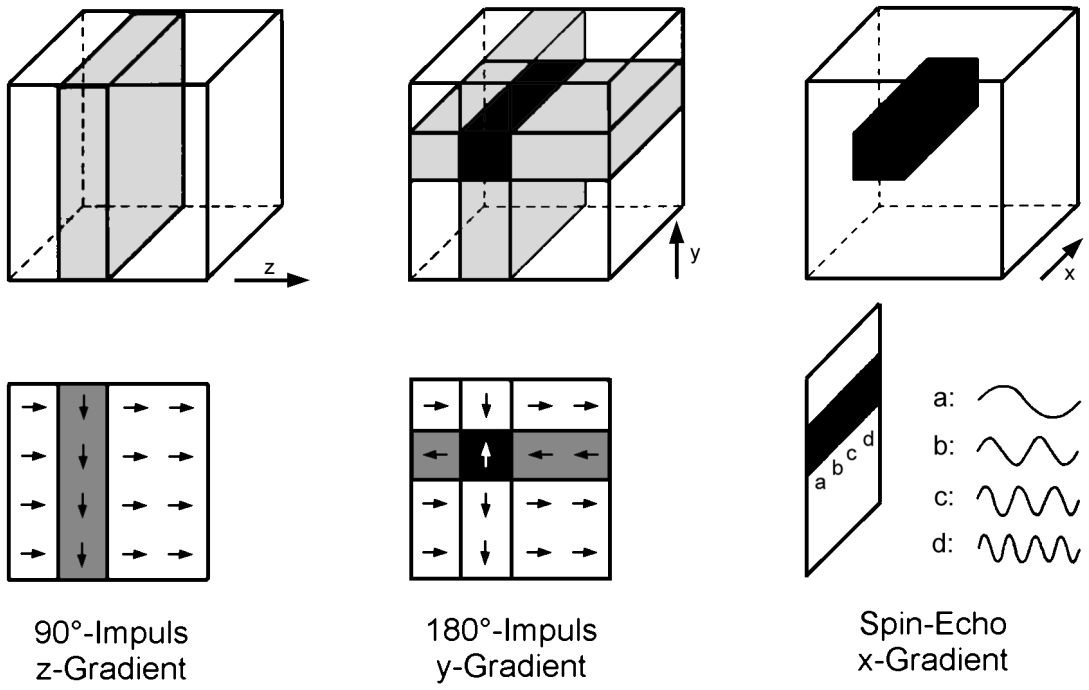

Abbildung 58: Schematische Darstellung der Linientechnik [LISS90]

An dieser Stelle besteht noch das Problem, daß das gemessene Signal aus der gesamten selektierten Linie resultiert. Punkte auf der *x*-Achse werden nicht selektiv angeregt, wodurch auch keine Zuordnung des Signals zu einer bestimmten *x*-Koordinate möglich ist.

Die Kodierung des Signals entlang der *x*-Achse erfolgt mit Hilfe eines weiteren Gradienten, der während der Dauer des Spinechos geschaltet wird. Aufgrund der Schaltung zum Zeitpunkt der Auslesung des Signals wird dieser Gradient Auslesegradient genannt und aufgrund seiner Funktion nennt man ihn auch Frequenzkodiergradient. Er prägt den Spins entlang der *x*-Achse unterschiedliche Präzessionsfrequenzen auf, indem sich die Präzessionsfrequenzen der Spins der lokalen magnetischen Flußdichte anpassen (Abbildung 58 rechts). Somit besteht das als Spinecho empfangene Signal nicht mehr allein aus einer Frequenz, sondern aus einem Frequenzgemisch. Bei bekannter Beziehung zwischen lokaler magnetischer Flußdichte und *x*-Koordinate – und damit bekannter Beziehung zwischen lokaler Präzessionsfrequenz und *x*-Koordinate – läßt sich die Spindichte (Protonendichte) und die T_1- und T_2-Konstante für jede *x*-Koordinate der durch die beiden anderen Gradienten selektierten Linie berechnen.

Das dafür verwendete Verfahren ist die Fourieranalyse (Kapitel 3.4.1). Sie dient der Dekodierung eines Frequenzgemisches sinusförmiger Schwingungen. Gemeint ist die Zerlegung eines Frequenzgemisches in periodische Grundschwingungen, einschließlich der Berechnung deren Amplitude und deren Phasenbeziehung. Die Anzahl der Schwingungen, in die das Signalgemisch zerlegt wird, ergibt sich kanonisch durch die gewünschte Bildauflösung. Bei einer Auflösung von 256^2 Bildpunkten würde man ein Zerlegung des Frequenzgemisches in 256 Komponenten anstreben. Der Frequenzkodiergradient verläuft natürlich kontinuierlich. Das Signalgemisch besteht daher aus unendlich vielen Signalen mit unterschiedlicher Frequenz. Aus diesem Grund ist eine Abbildung in eine endliche Menge von Werten notwendig. Und dies hat wiederum zur Folge, daß die durch den Frequenzkodiergradienten ausgewählten Volumina eine endliche Ausdehnung aufweisen und nicht unendlich klein sind. Eine Veränderung der Größe der selektierten Volumina in Richtung des

Frequenzkodiergradienten kann man demzufolge durch eine Veränderung der Anzahl der Frequenzen erreichen, in die das Signal mittels der Fouriertransformation zerlegt wird.

Wir wollen die Frequenzkodierung noch anhand eines Beispiels veranschaulichen. Dazu betrachten wir Abbildung 59. Hier wurde nur eine Schicht mit Hilfe eines Schichtselektionsgradienten ausgewählt. Die Zahlen in den Quadraten oben links in der Abbildung stellen die Signalamplituden dar, entsprechen also der Protonendichte an der entsprechenden Stelle. Vor der Einschaltung eines Frequenzkodiergradienten empfangen wir aus dem ganzen Volumen ein einheitliches Signal, sagen wir $cos(\omega_0 t)$. Der rechte obere Teil der Abbildung zeigt die Signale, die in jedem Volumenelement erzeugt werden (die wir aber nicht differenzieren können). Wir können nur die Summe dieses Signals messen, das sind $4 \cdot cos(\omega_0 t)$. Nach Einschaltung des Frequenzkodiergradienten (in der Abbildung rechts unten) präzedieren die Spins in der linken Spalte etwas langsamer (bezeichnet mit $cos(\omega_0^- t)$), die in der mittleren Spalte mit unveränderter Geschwindigkeit und die in der rechten Spalte mit leicht erhöhter Geschwindigkeit (bezeichnet mit $cos(\omega_0^+ t)$). Das nun gemessene Signal ist ein Frequenzgemisch, daß mit Hilfe der Fouriertransformation in die einzelnen Frequenzkomponenten zerlegt werden kann. Wir erhalten somit für die linke Spalte eine Signalintensität von $-cos(\omega_0^- t)$, für die mittlere Spalte $3 \cdot cos(\omega_0 t)$ und für die rechte Spalte $2 \cdot cos(\omega_0^+ t)$. Damit ist das Signal zwar schon in Summensignale für die einzelnen Spalten zerlegt, jedoch kann noch keine Auskunft über die Signalverteilung innerhalb einer Spalte gemacht werden. Auf dieses Problem gehen wir im folgenden Kapitel ein.

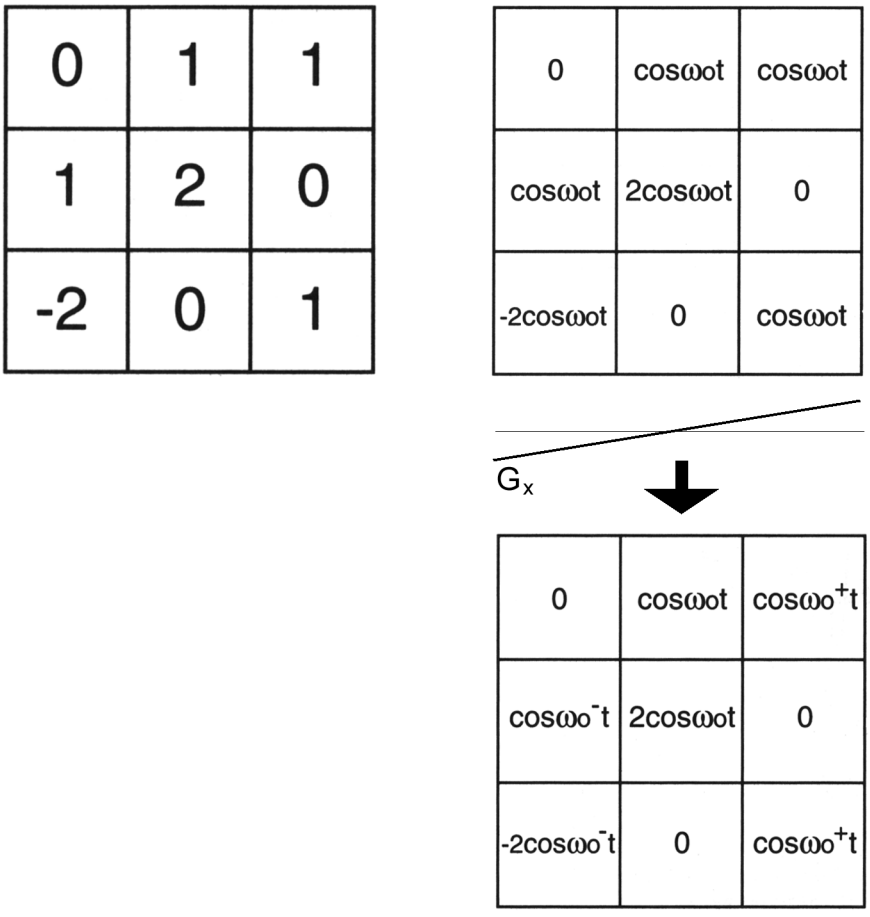

Abbildung 59: Frequenzkodierung [HASH97]

Ebenso wie die Punkttechnik wird auch die Linientechnik heute in der Routinediagnostik nicht mehr angewandt. Das liegt allerdings nicht an der Akquisitionszeit – denn diese entspricht der Größenordnung der im folgenden beschriebenen Schichttechnik und ist um den Faktor *n* kleiner als die der Punkttechnik – sondern am geringen Signal-zu-Rausch-Verhältnis. Dieses ist genauso groß wie bei der Punkttechnik. Das liegt daran, daß die gemessene Signalamplitude der Anzahl der in der gesamten Zeile (*n* Pixel) enthaltenen Spins proportional ist und jedes Bildelement nur einmal gemessen wird.

Dennoch ist das Verständnis der Linientechnik eine wichtige Grundlage für das Verständnis der Schicht- und Volumentechnik.

3.4.4 Schichttechnik

Die Schichttechnik basiert auf der Auswahl einer Schicht durch einen Schichtselektionsgradienten. Wie bei der Linientechnik wird das Signal während der Schaltung eines Frequenzkodiergradienten ausgelesen. Zur Kodierung der dritten Dimension stehen zwei unterschiedliche Verfahren zur Verfügung.

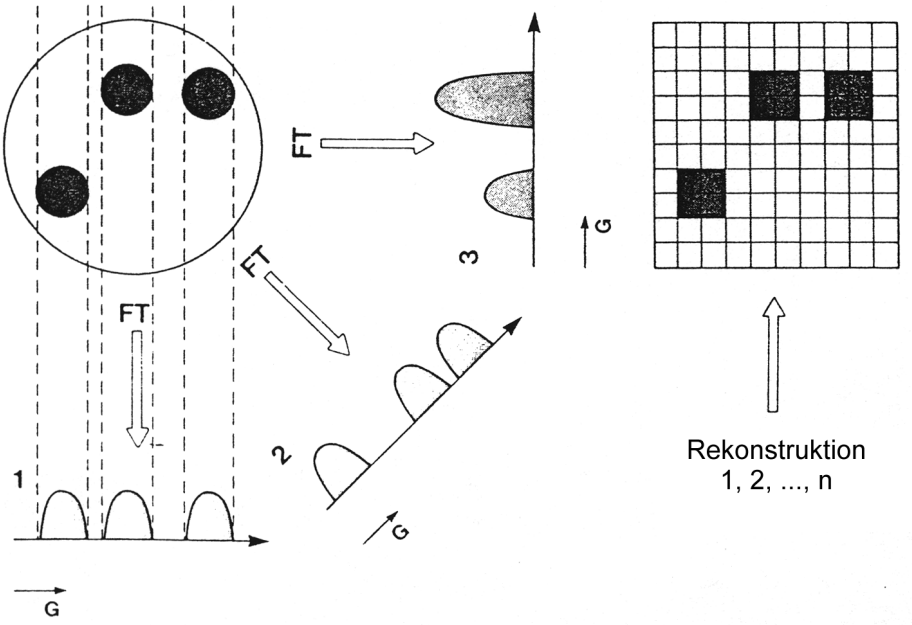

Abbildung 60: Projektions Rekonstruktion [LISS90]

Das erste Verfahren, die sogenannte Projektions-Rekonstruktion, wurde schon kurz im einleitenden Kapitel 2 im Zusammenhang mit der Computertomographie erwähnt. Dazu liest man das Signal der ganzen selektierten Schicht frequenzkodiert aus. Das bedeutet, daß man nach Anwendung einer Fouriertransformation eine eindimensionale Projektion der ganzen Schicht auf eine Linie erhält. Diese Linie verläuft senkrecht zum Schichtselektions- und parallel zum Auslesegradienten. Abbildung 60 zeigt diese Projektion unter (1). Die Idee ist nun, die Projektion *n* mal unter Variation des Projektionswinkels (bzw. des Einstrahlungswinkels des Frequenzkodiergradienten) zu wiederholen. Auch die zweite und dritte Projektion sind in Abbildung 60 dargestellt. Aus diesen *n* Datensätzen können dann schließlich die Signalintensitäten der einzelnen Bildpunkte rekonstruiert werden. Dieses Vorgehen veranschaulicht der rechte Teil von Abbildung 60.

Aufgrund der Rekonstruktion des Bildes aus verschiedenen Projektionen ist eine korrekte Zuordnung der Ortskoordinaten stark von der Homogenität der Magnetfelder abhängig. Inhomogenitäten führen zu kreis- und sternförmigen Artefakten, weshalb dieses Verfahren heute nicht mehr angewendet wird [LISS90].

Das zweite Verfahren verwendet neben dem Schichtselektions- und dem Auslesegradienten ein weiteres Gradientenfeld, den sogenannten Phasenkodiergradienten. Da er die *y*-Koordinaten kodiert, wird er mit G_y bezeichnet. Wir betrachten den Meßvorgang wieder anhand der Spin-Echo-Technik (Abbildung 61). Der 90°-Impuls wird während der Schaltung des Schichtselektionsgradienten (G_z) eingestrahlt. Damit präzedieren alle Spins in der selektierten Schicht phasensynchron mit der Lamor-Frequenz und erfüllen somit die Resonanzbedingung. Nach Abschaltung des Schichtselektionsgradienten wird der Phasenkodiergradient (G_y) in *y*-Richtung eingeschaltet. Dieser bewirkt, daß sich die Präzession der Spins der lokalen magnetischen Flußdichte anpaßt. Nach Abschalten des Phasenkodiergradienten präzedieren alle Spins in der selektierten Schicht wieder mit der gleichen

Geschwindigkeit. Die Phasendifferenz, die zuvor aufgebaut wurde, bleibt dabei erhalten. Sie ist dem Gradientenfeld und damit der *y*-Koordinate proportional und kann durch die Stärke des Gradientenfeldes und dessen Einwirkungsdauer beeinflußt werden. Mißt man das Resonanzsignal (bzw. das Echo) bei eingeschaltetem Frequenzkodiergradienten (G_x), so erhält man wiederum eindimensionale frequenzkodierte Projektionen, die nach einer Fouriertransformation jeweils eine Pixelsäule repräsentieren.

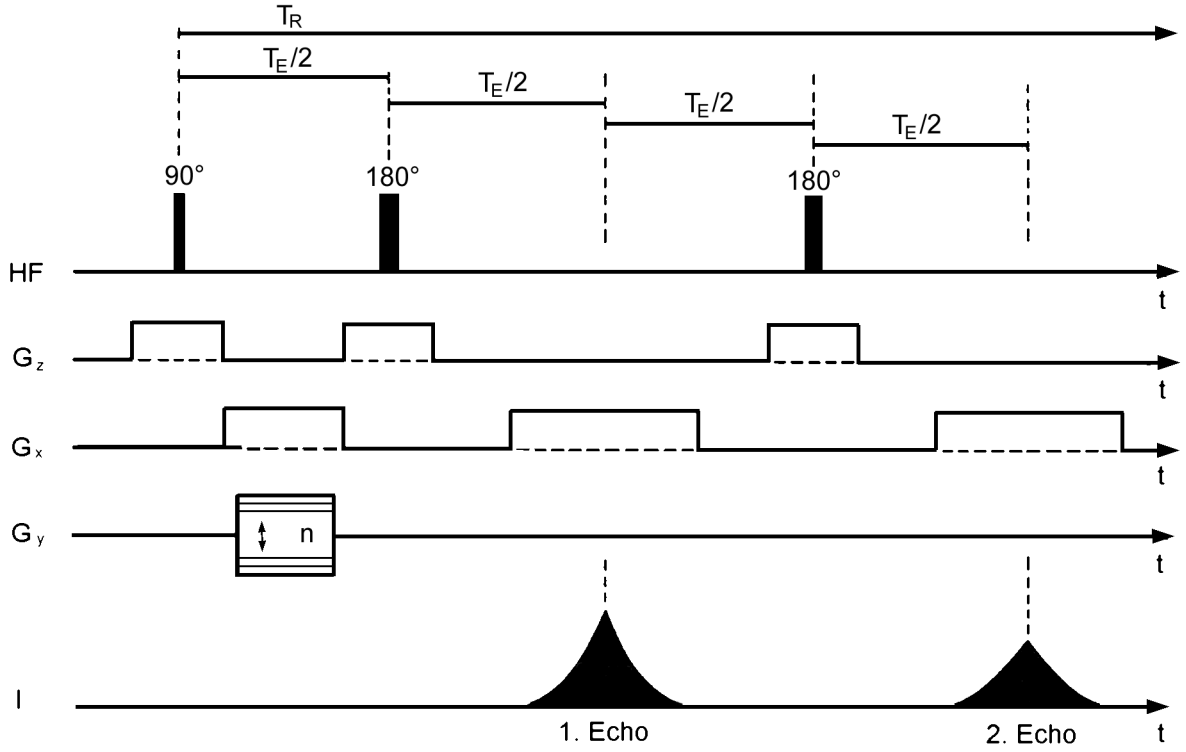

Abbildung 61: Zeitliche Abfolge der HF-Impulse und Gradientenschaltungen am Beispiel der Spin-Echo-Sequenz [HASH97]

Die Signalamplitude für jede Frequenz ist die Vektorsumme der Signalintensitäten der Bildelemente in *y*-Richtung. Aufgrund der *y*-koordinatenabhängigen Phasenverschiebung ist die Vektorsumme ebenfalls phasenkodiert. Bei *n*-maliger Wiederholung der Meßsequenz mit unterschiedlicher Einwirkungsdauer oder Stärke des Phasenkodiergradienten G_y erhält man bei jeder Wiederholung für jede Frequenzkomponente eine unterschiedliche Vektorsumme. Für gewöhnlich wählt man ebenso viele Phasenkodierschritte wie Frequenzkodierschritte, um in *x*- und *y*-Richtung die gleiche räumliche Auflösung zu erhalten. Es liegen dann also *n* Datensätze mit je *n* Meßwerten vor, aus denen die Signalintensitäten der n^2 Bildelemente mit Hilfe einer zweidimensionale Fouriertransformation berechnet werden können.

Um dieses Verfahren noch etwas transparenter zu erklären, soll es anhand eines Bildes mit 2·2 Bildelementen erläutert werden (Abbildung 62). In diesem Fall werden also zwei Messungen mit jeweils unterschiedlichem Phasenkodiergradienten – aber gleichem Schichtselektionsgradienten und Frequenzkodiergradienten – durchgeführt. Bei der ersten Messung verwendet man einfach den kleinstmöglichen Gradienten, also ein in der *y*-Richtung konstantes Feld ($G_y=0$). Die Phasendifferenz in *y*-Richtung ist dann Null. Die Vektorsumme der Signalamplitude beider Frequenzkomponenten ist nach der Fouriertransformation gleich der Summe der Einzelamplituden und damit gleich der Summe der Spindichte. Der zeitliche Signalzerfall durch T_1- und T_2-Relaxation ist dabei nicht berücksichtigt. Aus der ersten Messung resultieren die ersten beiden Gleichungen ((23) und (24)).

Bei der zweiten Messung erhöht man den Phasenkodiergradienten. Man wählt ihn so, daß die Phasendifferenz zwischen den Spins der ersten Spalte und denen der zweiten Spalte 180° beträgt. Damit entspricht die Vektorsumme der Signalamplituden beider Frequenzkomponenten nun der

Differenz der Einzelamplituden und damit der Differenz der Spindichten und es ergeben sich die Gleichungen (25) und (26).

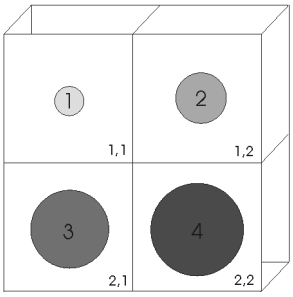

Abbildung 62:Bildrekonstruktion mit der 2-dim. Fouriertransformation [LISS90]

$$I_1 = I_{(1,1)} + I_{(1,2)} = I + 2I = 3I \tag{23}$$

$$I_2 = I_{(2,1)} + I_{(2,2)} = 3I + 4I = 7I \tag{24}$$

$$I_3 = I_{(1,1)} + I_{(1,2)} = I - 2I = -I \tag{25}$$

$$I_4 = I_{(2,1)} + I_{(2,2)} = 3I - 4I = -I \tag{26}$$

Aus der Lösung diese Gleichungssystems (4 Gleichungen mit 4 Unbekannten) ergeben sich die Signalintensitäten $I_{(a,b)}$, die jeweils zu der Spindichte in dem entsprechenden Bildelement proportional sind.

An dieser Stelle ist es angebracht, über den Zeitbedarf einer Schichtaufnahme zu sprechen. Grund für den relativ großen Zeitbedarf sind die Phasenkodierschritte. Wir müssen so viel Phasenkodierschritte durchführen, wie wir Spalten in unserem Ergebnisbild haben wollen. Bei einer Bildauflösung von 256² Bildpunkten, müssen also 256 Phasenkodierschritte durchgeführt werden. Jeder Phasenkodierschritt benötigt ein neues Spin-Echo, also jeweils eine Zeit von T_R. Insgesamt ergibt sich somit ein Zeitbedarf von $n \cdot T_R$ für die Aufnahme einer Schicht. Wie in den Kapiteln 3.3.2 und 3.5.3 beschrieben wird, kann der Zeitaufwand durch Benutzung von Schnellbildsequenzen (z.B. Turbo Spin-Echo) stark reduziert werden.

Die Schichttechnik weist gegenüber der Linientechnik aufgrund des um den Faktor n vergrößerten Meßvolumens ein um den Faktor $\sqrt{n}$ verbessertes Signal-zu-Rausch-Verhältnis auf, wodurch die Bildqualität erheblich verbessert wird. Die minimale Akquisitionszeit kann jedoch nicht verkleinert werden, da beide Techniken eine n-malige Wiederholung des Resonanzexperimentes zur Erfassung einer ganzen Schicht erfordern.

3.4.5 Volumentechnik

Die Volumentechnik ist im gewissen Sinne eine weitere "Steigerung" der Schichttechnik. Zunächst wird das Gesamtvolumen ohne Schichtselektionsgradient angeregt. Für die räumliche Zuordnung des Resonanzsignals werden nacheinander 2 Phasenkodiergradienten (in y- und in z-Richtung) geschaltet. Die Kodierung in x-Richtung erfolgt wie bei den anderen Techniken auch durch einen Auslesegradienten. Das frequenzkodierte Resonanzsignal kann schließlich durch eine dreidimensionale Fouriertransformation in den Ortsbereich übertragen werden.

Die Volumentechnik bietet gegenüber der Schichttechnik den Vorteil eines um den Faktor $\sqrt{m}$ (m ist die Anzahl der Schichten) verbesserten Signal-zu-Rausch-Verhältnisses. Das liegt daran, daß m-mal mehr Spins am Resonanzprozeß teilnehmen. Ein weiterer Vorteil ist die Möglichkeit, quasi beliebig dünne Schichten aufzeichnen zu können. Grund dafür ist, daß bei diesem Verfahren die Auflösung der 3. Dimension durch den Phasenkodiergradienten und dessen Einschaltzeit bestimmt wird und nicht durch die Bandbreite des Schichtselektionsgradienten begrenzt wird. Und da die Einschaltzeit keinen prinzipiellen Einschränkungen unterliegt, sind im Prinzip beliebig dünne Schichten möglich. Die Gesamtakquisitionszeit für ein 3-dimensionales Volumen der Größe $n^2 \cdot m$ beträgt $n^2 \cdot m \cdot T_R$ und ist somit in der Routinediagnostik nur mit Hilfe von Schnellbildtechniken (Kapitel 3.3.2 und 3.5.3) und einer kurzen Repititionszeit T_R in angemessener Zeit möglich. Einen weiteren Vorteil stellt die Möglichkeit dar, im nachhinein Schichten mit beliebiger Raumlage aus den Rohdaten zu berechnen.

3.4.6 Vielschichttechnik

Eine sehr interessante Technik ist die Vielschichttechnik. Sie baut auf dem Gedanken auf, daß alle Pulssequenzen mehr oder weniger lange Zeitperioden aufweisen, während denen weder Gradientenfelder geschaltet sind, noch HF-Impulse eingestrahlt werden. Diese Signalpausen ergeben sich ganz natürlich durch die recht langen T_1-Zeiten verschiedener Gewebe und den dadurch ebenfalls recht langen Repititionszeiten. Der Grundgedanke der Vielschichttechnik ist es nun, während dieser Signalpausen Gradientenfelder und HF-Impulse auf andere Schichten zu schalten. Somit ist eine simultane Aufzeichnung mehrerer Schichten möglich, ohne daß sich dabei die Gesamtakquisitionszeit verlängert.

Zu welchem Zeitpunkt genau die Schichten in den Signalpausen angeregt werden, hängt von der Pulssequenz ab. Bei der Spin-Echo-Technik bietet sich eine konsekutive Anregung der Schichten an. Bei einer Repititionszeit von T_R=800 ms und einer Echozeit von T_E=17 ms ist der Meßvorgang für die zuerst angeregt Schicht innerhalb von weniger als 50 ms abgeschlossen. Die restlichen 750 ms wartet man darauf, daß die T_1-Relaxation fortschreitet und die Spins sich ihrer Gleichgewichtslage nähern. In dieser Zeit kann man jedoch nacheinander 15 weitere Schichten anregen und deren Resonanzsignal messen.

Wählt man für die Inversion-Recovery-Impulsfolge T_R=1600 ms, T_I=400 ms und T_E=17 ms, so dauert die Meßwerterfassung über das Spinecho mit ca. 500 ms wesentlich länger als bei der Spin-Echo-Technik. Durch Impulsverschachtelung können jedoch vor Einstrahlung des 90°-Impulses in die erste Schicht ca. 8 weitere Schichten angeregt werden. Der gleiche Vorgang kann in der zweiten Hälfte des T_R-Intervalls wiederholt werden.

Wie bereits erwähnt, nehmen die Signalpausen mit wachsendem T_R zu. Hieraus folgt, daß bei T_1-betonten Aufnahmen mit kurzem T_R die zu erreichende simultane Schichtanzahl gegenüber T_2-betonen Aufnahmen relativ gering ist. Andererseits muß man aber auch sehen, daß T_2-betone Aufnahmen eine längere T_E-Zeit benötigen, wodurch die theoretisch mögliche höhere Schichtanzahl wieder relativiert wird.

Zu beachten ist auch, daß eine zu kleine Wahl des Schichtabstandes dazu führt, daß die Schichtflanken mehrfach angeregt werden (Abbildung 63b). Das resultiert daraus, daß die Anregungsimpulse (im Frequenzbereich) nicht ideal rechteckig sind, wie dies in Abbildung 63a der Fall ist. In Abbildung 63 (a-c) repräsentiert jede der Frequenzgruppen eine Schicht. Man sieht deutlich die Überschneidung der Schichten in Teil b der Abbildung. Da diese erneute Anregung nicht mit T_R, sondern mit einer viel kürzeren Repititionszeit T_R^* erfolgt, wird die Signalintensität verfälscht. Eine Verringerung dieses Effektes kann man erreichen, indem man nicht direkt benachbarte Schichten nacheinander anregt, sondern zum Beispiel erst die Schichten mit ungerader Nummer und anschließend die Schichten mit gerader Nummer. Oder aber man vergrößert den Schichtabstand (Abbildung 63c), geht dabei allerdings das Risiko ein, Pathologien in den Schichtlücken nicht registrieren zu können. Realisierbare und für die praktische Durchführung sinnvolle Schichtabstände liegen im Bereich von 0 bis 200% der Schichtdicke.

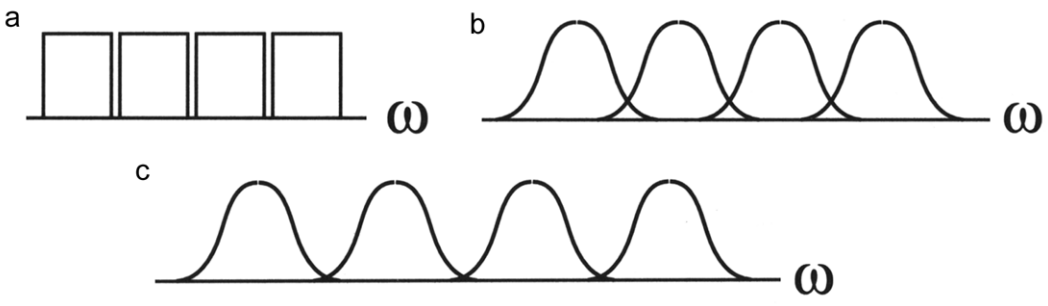

Abbildung 63: Mehrfache Anregung einiger Schichten bei der Vielschichttechnik [HASH97]

3.5 k-Raum

In Kapitel 3.4 und seinen Unterkapiteln sollte eigentlich schon klar geworden sein, daß das gemessene Resonanzsignal nicht direkt das MR-Bild darstellt. Vielmehr wird ein zeitabhängiges frequenz- und phasenkodiertes Schwingungsgemisch gemessen (Abbildung 64). Die komplexe Fouriertransformation dieses Signals ergibt die Verteilung der Quermagnetisierung als Funktion der Frequenz, die Kernresonanzlinie. Der Realteil dieser Funktion wird Absorptionslinie genannt und stellt die Frequenzabhängigkeit der *x*-Komponente des präzedierenden Magnetisierungsvektors dar (Abbildung 65).

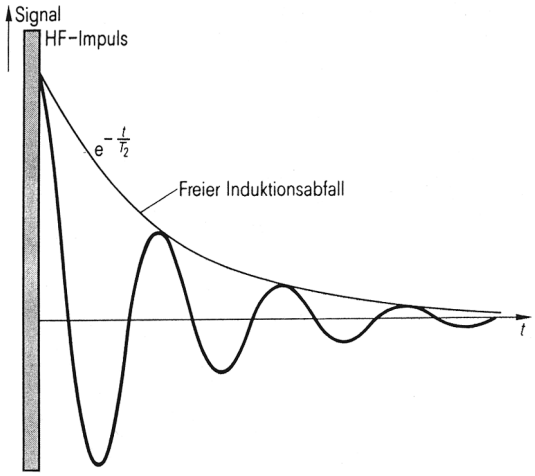

Abbildung 64: Kerninduktionssignal nach einem 90°-Impuls [MORN95]

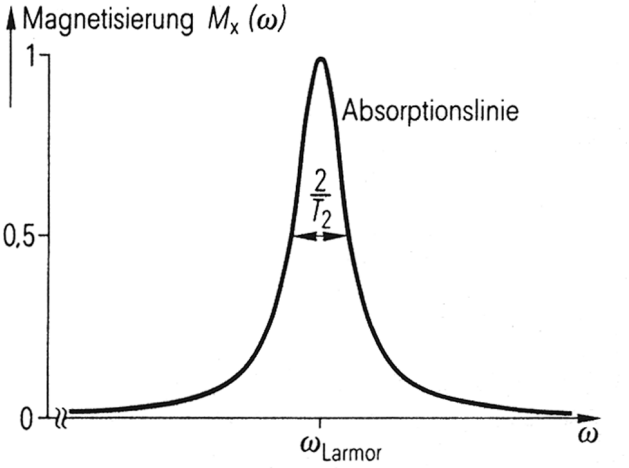

Abbildung 65: Absorptionslinie: Der Realteil der Fouriertransformation des Zeitsignals [MORN95]

Dieses Kapitel soll sich mit dem "Bild" im Fourierraum – dem sogenannten k-Raum – befassen. Es soll aufgezeigt werden, welche Auswirkungen Manipulationen der Daten im k-Raum auf das transformierte Bild im Bildraum haben.

Der k-Raum ist zunächst ein abstrakter Begriff, man kann ihn nicht anfassen und ihn sich nur sehr schlecht vorstellen. Die Visualisierung der Daten im k-Raum (Abbildung 70 und Abbildung 71) gibt noch keinen direkten Eindruck des endgültigen MR-Bildes. Aus diesen Gründen verwenden die folgenden Betrachtungen zur Veranschaulichung die Tatsache, daß die Konzepte und Gleichungen bzgl. des k-Raums denen eines Linsensystems entsprechen, wie es z.B. in Teleskopen, Mikroskopen, Ferngläsern und sogar dem menschlichen Auge vorkommt.

Betrachten wir also zuerst die Erzeugung eines optischen Bildes durch eine Linse (Abbildung 66). Dieses ist ein zweistufiger Prozeß. Zunächst einmal muß das abzubildende Objekt von Licht angestrahlt werden. Das vom Objekt reflektierte und in den Raum gestreute Licht (oder ein Teil davon) wird dann von der Linse ‚aufgesammelt'. Im zweiten Schritt ‚verarbeitet' die Linse das Licht zu einem Bild. Dabei hängen die Bildeigenschaften wie Größe, Auflösung und Kontrast im wesentlichen von den Linseneigenschaften ab. Eine Linse mit größerem Durchmesser erzeugt beispielsweise ein höher aufgelöstes Bild oder anders ausgedrückt, es können kleinere Details abgebildet werden. Die genaue Arbeitsweise der Linse hängt vom Zusammenspiel der Linsenkrümmung und der Geschwindigkeit des Lichtes in der Linse ab, welche geringer ist als die in Luft. Dadurch findet eine Beugung des Lichts an den Grenzflächen zwischen Luft und Linse statt, die schließlich durch die charakteristische Form der Linse zu einer Bündelung des Lichts in einem Punkt in einer Ebene hinter der Linse führt.

Nun wollen wir in den Analogiebetrachtungen noch eine Ebene tiefer gehen. Im achtzehnten Jahrhundert stellte man fest, daß Licht durch ein Gitter ebenfalls gebeugt wird. Aus diesem Grund werden solche Gitter auch Diffraktionsgitter genannt. Der Grad der Beugung hängt dabei von der Frequenz des Lichts und des Gitters ab. In Bezug auf das Gitter ist mit Frequenz die Anzahl der durchsichtigen und undurchsichtigen Elemente pro Millimeter bzw. Längeneinheit gemeint. Man kam schnell zu der Feststellung, daß man eine Linse somit auch durch eine Menge geeignet gewählter

Gitter ersetzen kann (Abbildung 67). Und man konnte sich auch sehr bald vorstellen, daß das Licht selbst aus einer Menge von Schwingungen mit unterschiedlichen Frequenzen zusammengesetzt ist – daher die unterschiedliche Brechung an Gittern mit unterschiedlicher Frequenz. Und mehr noch, diese Schwingungen haben auch eine unterschiedliche räumliche Anordnung.

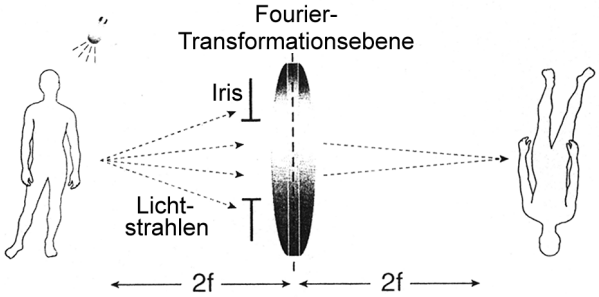

Abbildung 66: Eine Linse mit Fouriertransformationsebene [MEZR95]

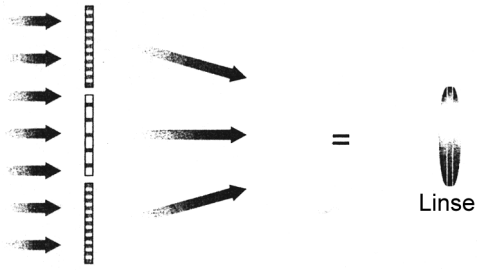

Abbildung 67: Ersetzung einer Linse durch eine Menge von Diffraktionsgittern [MEZR95]

Nun wieder zurück zur Linse. Mit den eben gewonnenen Erkenntnissen ist es klar, daß eine Linse ein Filter ist, der mehr oder weniger der verschiedenen Frequenzen des Lichtes durchläßt. Und je größer die Linse ist, desto mehr Frequenzen werden durchgelassen und desto kleinere Objekte können damit abgebildet werden.

Neben der Filterfunktion muß die Linse die Frequenzen des Lichtes auch konvertieren und zum Bild kombinieren. Und genau diese beiden Funktionen – Filterung und Konvertierung – sind es, die auch von der Fouriertransformation durchgeführt werden. Man kann sich das so vorstellen, daß das Licht, nachdem es von der Linse "aufgefangen" wurde, in einer Ebene mitten in der Linse als "Datensatz" zur Verfügung steht. Auf diesen Daten – wir wollen sie Rohdaten nennen – wird eine Fouriertransformation durchgeführt, um so das Bild in einer anderen Ebene zu erzeugen. Die Ebene inmitten der Linse wird aus diesem Grund auch Fouriertransformationsebene genannt. Man kann sie sich als zweidimensionales Feld vorstellen. Mit jedem Element diese Feldes sind zwei räumliche Frequenzen verbunden, eine in *x*- und eine in *y*-Richtung. Dabei liegen die höheren Frequenzen näher zum Rand und die tieferen näher am Zentrum (Abbildung 68).

Stellen wir nun wieder den Zusammenhang zur Kernspintomographie her. Auch hierbei werden zur Erzeugung eines ortsaufgelösten Bildes zwei Schritte benötigt. Zuerst werden die Spins des abzubildenden Objektes angeregt und in Resonanz gebracht. Die dabei von den Spins ausgesendete elektromagnetische Strahlung wird durch Antennen empfangen. In einem zweiten Schritt wird aus den gemessenen Signalen das MRT-Bild erzeugt. Dieser zweite Schritt wird ebenfalls durch eine Fouriertransformation durchgeführt. Das heißt also, daß es auch in der Kernspintomographie eine abstrakte Ebene geben muß, die die Rohdaten frequenzkodiert enthält. Diese Ebene existiert – sie wird k-Raum genannt – und sie entspricht konzeptionell genau dem eben beschriebenen zweidimensionalen Feld (Abbildung 68).

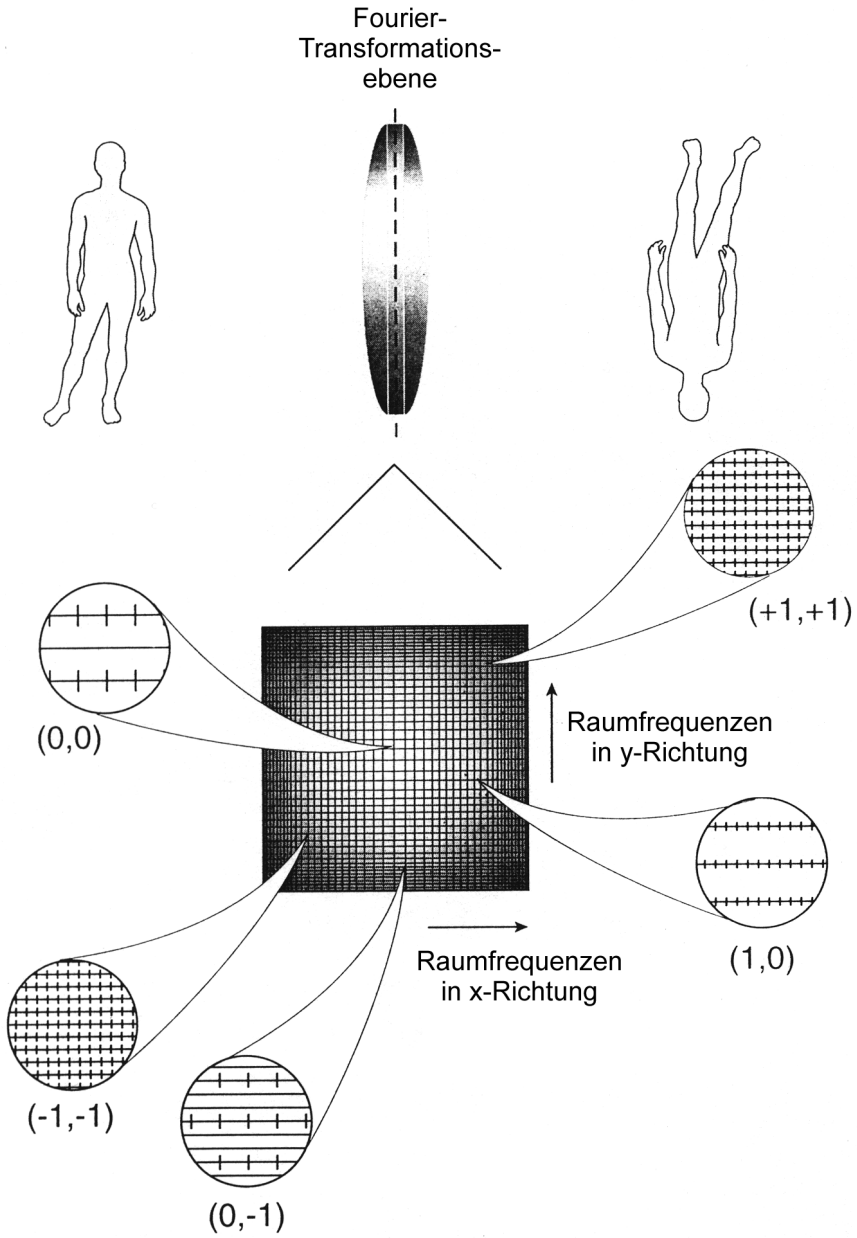

Abbildung 68: Die Fouriertransformationsebene (k-Raum) [MEZR95]

Wir wollen jetzt noch einmal die im letzten Kapitel beschriebene Ortskodierung mit dem k-Raum in Zusammenhang bringen. Während jedes Phasenkodierschrittes wird eine Zeile (oder auch Spalte – das ist Definitionssache) des k-Raums bzw. Datenraums[16] gefüllt. Abbildung 69 zeigt einen k-Raum, der mit 256 Phasenkodierschritten aufgenommen wurde. Da während des Phasenkodierschrittes ohne Phasenkodiergradient das maximale Signal gemessen werden kann, wird dieses Signal meistens in die mittlere Zeile des k-Raums eingefügt. Denn wie wir später noch sehen werden, sind die mittleren Punkte des k-Raums für den Bildkontrast und die Grobstrukturen verantwortlich. Je weiter man den Phasenkodiergradienten erhöht oder vermindert, desto mehr dephasieren die Spins und das gemessene Signal wird schwächer. Daher werden diese Signale weiter außen im k-Raum eingefügt. Es soll aber auch noch einmal betont werden, daß das keine feste Regel ist, sondern lediglich eine Vorgehensweise, die Bilder von guter Qualität liefert. Da jeder der Phasenkodierschritte (im Gegensatz zur Frequenzkodierung in *x*-Richtung) die im Sekundenbereich liegende Zeit T_R benötigt, treten Bewegungsartefakte hauptsächlich in Richtung des Phasenkodiergradienten auf. Genau wie die

[16] Der k-Raum ist die digitalisierte Form des Datenraums. Da es schwierig und unanschaulich ist, das digitalisierte Signal darzustellen, wird in Abbildung 69 das analoge Signal dargestellt.

mittlere Zeile des k-Raums die maximale Signalintensität in Phasenkodierrichtung liefert, so liefert die mittlere Spalte des k-Raums die maximale Signalintensität in Richtung der Frequenzkodierung, da sie dem Zentrum des Echos entspricht.

Der Schichtselektionsgradient hat keinen Einfluß auf einen einzelnen k-Raum. Vielmehr spezifiziert der Schichtselektionsgradient genau einen k-Raum. Oder anders gesagt, jede Schicht hat einen eigenen k-Raum.

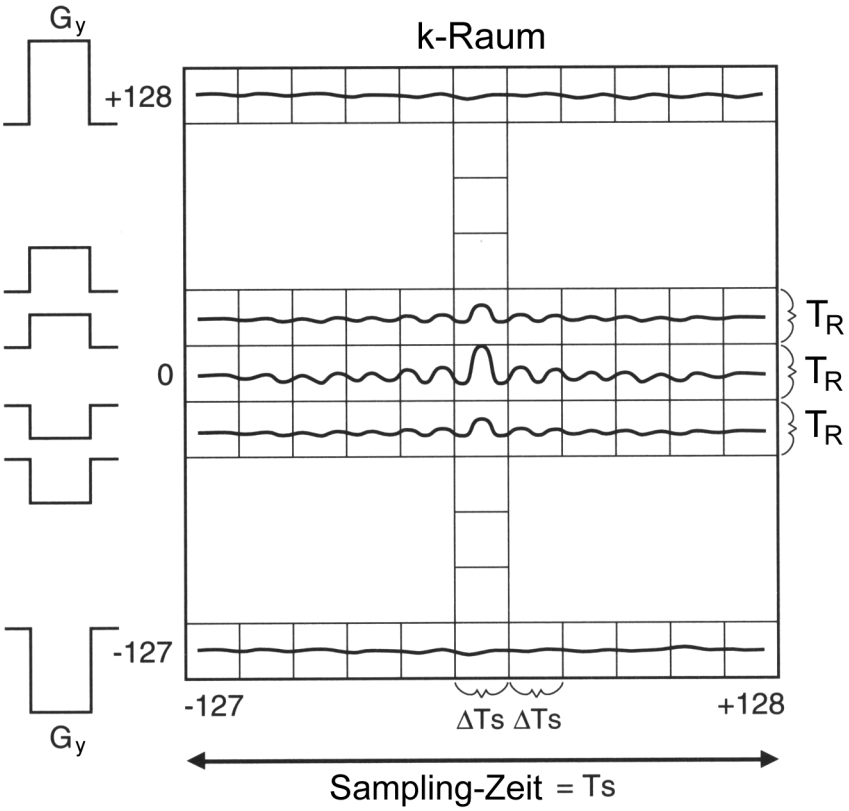

Abbildung 69: k-Raum in analoger Form [HASH97]

Ein Aspekt des k-Raums wurde bisher noch gar nicht angesprochen, nämlich die Unterteilung in einen Real- und einen Imaginärteil. Die Fouriertransformation des Realteils ergibt das Realbild und die Fouriertransformation des Imaginärteils das Imaginärbild. Aus Real- und Imaginärbild erhält man mit Hilfe der Formel

$$m = \sqrt{a^2 + b^2} \tag{27}$$

das Magnitudenbild und mit Hilfe der Formel

$$\theta = b / a \tag{28}$$

das Phasenbild. Dabei ist *a* der Realteil und *b* der Imaginärteil. Uns interessiert in erster Linie das Magnitudenbild. Das Phasenbild wird nur benötigt, wenn Bewegungen eine Rolle spielen. Idealerweise wünschen wir uns ein Realbild mit einem Imaginärteil gleich Null und somit einem Phasenbild gleich Null. In der Realität führen Bewegungsartefakte und Gradientenungenauigkeiten aber immer dazu, daß das Phasenbild nicht Null ist.

Als nächstes sollen einige grundlegende Konzepte der Bildgewinnung aus dem k-Raum bzw. aus der Fouriertransformationsebene erklärt werden. Das wichtigste ist zunächst, daß es keinen direkten Zusammenhang zwischen einem Punkt im k-Raum und einem Punkt im MR-Bild (Bildraum) gibt. Vielmehr leistet jeder Punkt im k-Raum einen gewissen Beitrag zu jedem Punkt im Bildraum. Das kann man sich leicht an einem Beispiel aus der Photographie klarmachen. Staub und Schmutz auf der Linse erzeugen im Bild keine Punkte oder Flecken, sondern führen vielmehr zu einem mehr oder weniger starken Schleier im gesamten Bild. Das gleiche gilt auch für größere Verdeckungen der Linse.

Man denke einfach nur an eine einstellbare Kamerablende. Eine Verkleinerung der Blende hat keinesfalls eine Verkleinerung des sichtbaren Bildausschnitts zur Folge. Vielmehr wird die Schärfe bzw. die Auflösung des Bildes verringert. Ein weiterer Effekt ist natürlich die Verringerung der Helligkeit durch den geringeren Lichteinfall. In der Kernspintomographie kann die Helligkeit zwar im nachhinein manipuliert werden, jedoch ist in jedem Fall eine Verringerung des Signal-zu-Rausch-Verhälnisses zu verzeichnen.

Welchen Einfluß ein Punkt im k-Raum auf die Punkte im Bildraum hat, hängt von seiner Position im k-Raum ab. Punkte am Rand des k-Raums enthalten die Detailinformationen des Bildes und tragen somit zur Schärfe und zu einer höheren Auflösung des Bildraums bei. Dies ist in Abbildung 70 und Abbildung 71 dargestellt. In diesen Abbildungen ist auch die Visualisierung des k-Raums zu sehen. Punkte im mittleren Bereich des k-Raums tragen dagegen eher zum Kontrast des Bildes bei. Auch das ist aus Abbildung 70 und Abbildung 71 ersichtlich. Der Übergang der Bedeutung der Punkte ist fließend. Es gibt also keinen festen Bereich, in dem die Punkte nur zum Kontrast oder nur zur Auflösung beitragen.

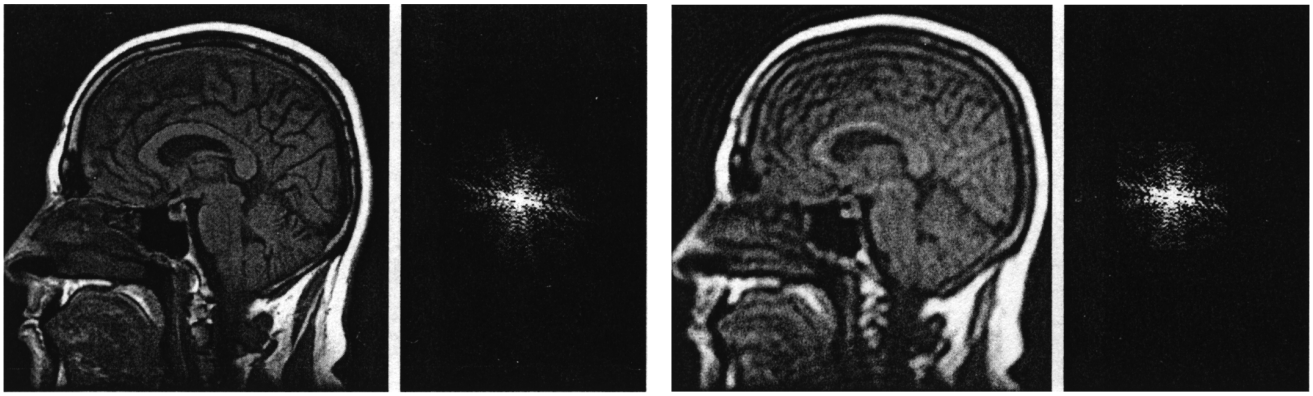

Abbildung 70: k-Raum[17] mit 256*256 Punkten mit dazugehörigem Bild (link) und ein in seiner Größe halbierter k-Raum mit dem dazugehörigen Bild (rechts) [MEZR95]

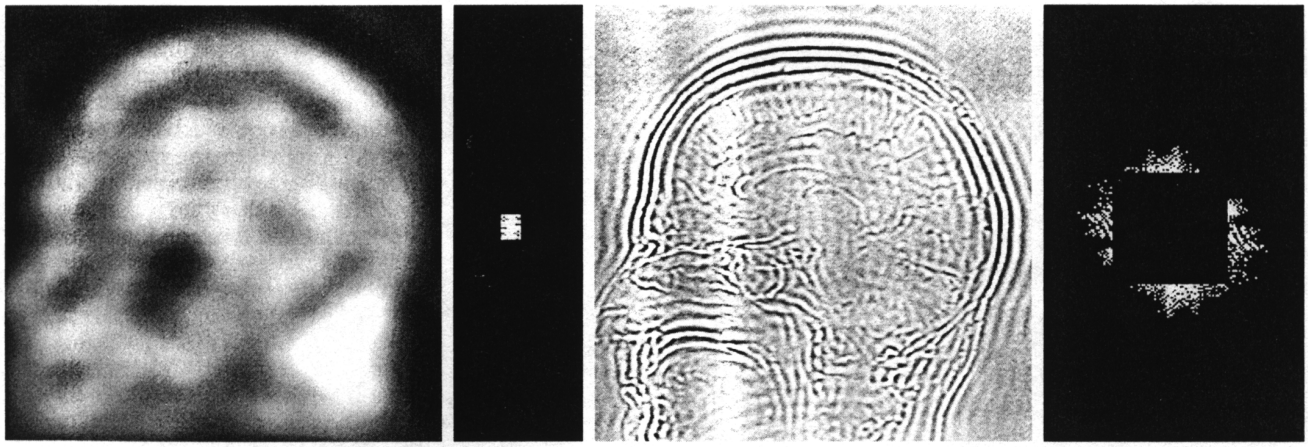

Abbildung 71: k-Raum mit 1/32 seiner ursprünglichen Größe und das dazugehörige Bild (links) und ein k-Raum mit fehlendem Zentrum und das dazugehörige Bild (rechts) [MEZR95]

Die Bildqualität wird also im Endeffekt durch die Größe des k-Raums bestimmt. Ein größerer k-Raum verbessert auch die Bildqualität. Gegen eine übermäßige Vergrößerung des k-Raums sprechen allerdings einige Punkte. Einer davon ist die Akquisitionszeit. Diese muß für den Patienten in einem erträglichen Rahmen gehalten werden. Eine zu lange Akquisitionszeit kann auch dazu führen, daß der

[17] Die dargestellten k-Raum-Bilder sind wie allgemein üblich um die halbe Bildbreite bzw. –höhe nach rechts bzw. unter verschoben worden. Außerdem wurden die Intensitätswerte zur besseren Darstellbarkeit logarithmiert.

Patient sich während der Messung bewegt. Das führt zu Bewegungsartefakten, die sich in der Regel in Unschärfe und Verschmierungen des Bildes äußern. Ein zweiter Punkt ist der, daß eine Vergrößerung der Zeilen- und/oder Spaltenzahl des k-Raums gleichzeitig zu einer Verkleinerung der einzelnen Bildpunkte und der gemessenen Volumenelemente führt. Damit nimmt aber auch die Anzahl der resonanzfähigen Spins pro Volumeneinheit ab, worunter auch die Signalintensität und das Signal-zu-Rausch-Verhältnis leiden. Eine wichtige Faustformel sollte also lauten, so viel k-Raum so schnell wie möglich zu füllen. Qualität und Geschwindigkeit müssen hierbei abgewogen werden.

Welche Punkte des k-Raums in welcher Reihenfolge gefüllt werden, wird durch die **Pulssequenz** bestimmt. Die Während der Pulssequenz angelegten Gradientenstärken bestimmen die Plazierung der Daten im k-Raum. Durch Veränderung der Gradientenstärken können also Auflösung und Kontrast des Bildes beeinflußt werden (analog zur Veränderung des Irisdurchmessers in der Photographie). Dabei beeinflussen allerdings nur der Frequenzkodier- und der Phasenkodiergradient die Bildqualität. Der Schichtselektionsgradient wählt nur die abzubildende Schicht und deren Dicke und Orientierung aus dem Objekt aus. In Abbildung 72 werden die Daten, die mit großem Phasenkodiergradienten akquiriert werden, in die oberen Zeilen des k-Raums eingefügt. Daten, die mit einem Phasenkodiergradienten nahe Null erfaßt werden, landen dagegen in den mittleren Zeilen des k-Raums und Daten die mit negativen Phasenkodiergradienten akquiriert werden, werden am unteren Rand des k-Raum eingefügt. Entsprechendes gilt für die Stärke des Frequenzkodiergradienten und die Spalten des k-Raums. Somit besteht eine direkte Beziehung zwischen den Punkten im k-Raum und der Gradientenstärke während der Akquisition.

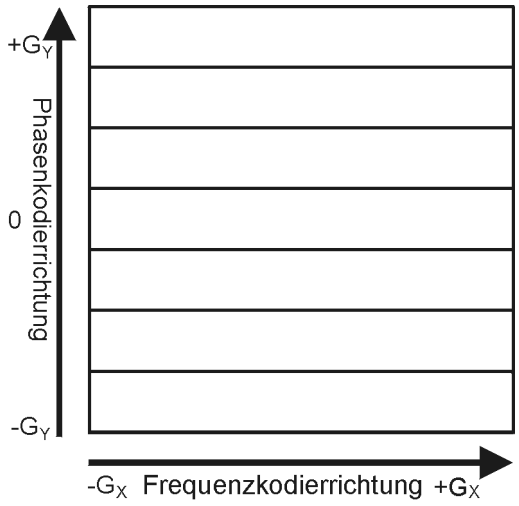

Abbildung 72: Füllen des k-Raums [MEZR95]

Im Folgenden soll anhand einiger Pulssequenzen gezeigt werden, wie genau es möglich ist, die Bildakquisitionszeit zu verkürzen oder die Bildqualität zu verbessern. Es sollen jeweils die Vor- und Nachteile dieser Techniken erläutert werden und auf Möglichkeiten zur k-Raum-Manipulation eingegangen werden. Es handelt sich bei den folgenden Kapiteln um eine Fortsetzung und Ergänzung des Pulssequenz-Kapitels 3.3.

3.5.1 Spin-Echo-Technik

Bei der Standard-Spin-Echo-Technik wird eine Zeile des k-Raums pro 90°-Impuls gefüllt. Die Länge der Zeile ist dabei proportional zur maximalen Stärke und zur Einschaltdauer des Frequenzkodiergradienten G_x. Die Position der Zeile im k-Raum wird dagegen durch den Phasenkodiergradienten G_y bestimmt. Abbildung 73 veranschaulicht dies. Zeilen, die mit einem starken Phasenkodiergradienten akquiriert werden, werden weiter oben im k-Raum plaziert, Zeilen, die mit einem schwachen Gradienten akquiriert werden, landen in der Mitte und Zeilen, die mit negativem Phasenkodiergradienten erfaßt werden, werden im unteren Bereich des k-Raums positioniert.

Die Gesamtakquisitionszeit ergibt sich damit zu $n \cdot T_R$, wobei n die Anzahl der Phasenkodierschritte ist. Bei einer Repititionszeit von 2000 ms und 256 Phasenkodierschritten ergibt sich also eine Akquisitionszeit von 8,5 Minuten [MEZR95]. Um diese Zeit zu verkürzen, kann man die Anzahl der Phasenkodierschritte verringern (z.B. 64 statt 256). Dabei gibt es 2 Möglichkeiten den k-Raum zu füllen.

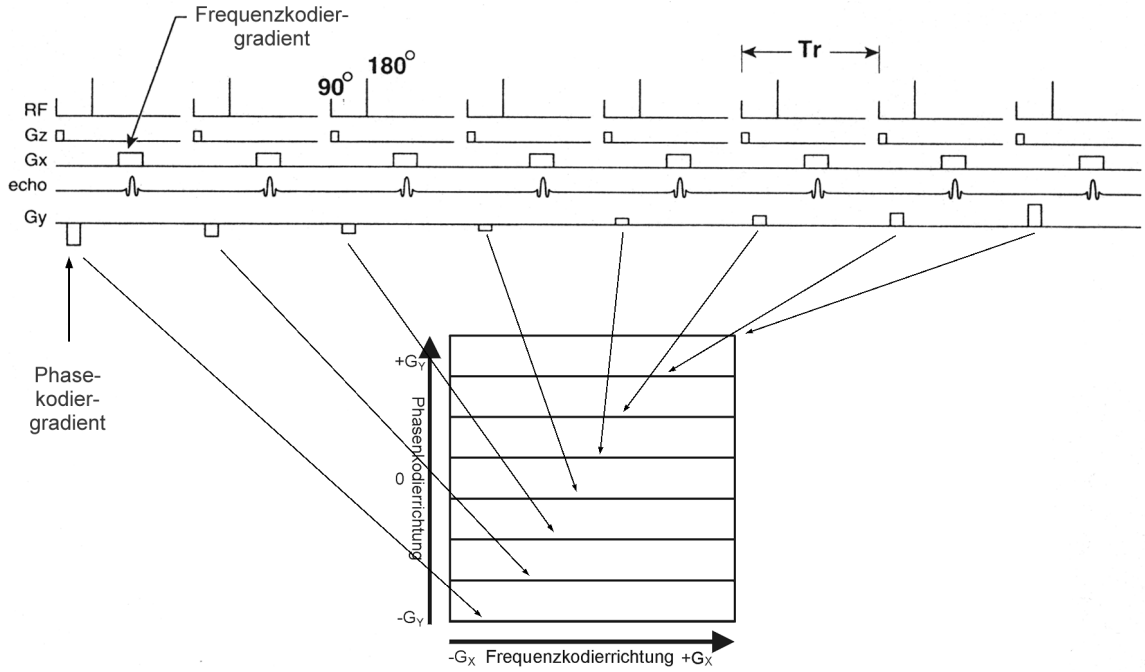

Abbildung 73: Füllen des k-Raums mit der Spin-Echo-Technik [MEZR95]

Die erste Möglichkeit läßt den Abstand zwischen den Zeilen des k-Raums konstant. Dies geschieht durch Aufrechterhalten der ursprünglichen Feldstärkenerhöhung bei den einzelnen Phasenkodierschritten. Da in diesem Fall nur ¼ der Zeilen aufgenommen werden, verringert sich auch die Höhe des k-Raums auf ¼ seiner ursprünglichen Höhe. Und weil die Größe des k-Raums die Auflösung des Bildes bestimmt, verringert sich auch diese entlang der Phasenkodierrichtung. Da die Größe des Bildausschnitts jedoch erhalten bleibt, kommt es dazu, daß das Bild unschärfer wird und verschmierter aussieht. Dies ist darauf zurückzuführen, daß jede Zeile im Bild um den Faktor 4 dicker wird.

Um die Konsistenz mit der Rastervorstellung des k-Raums zu gewährleisten, darf man sich den Abstand zwischen zwei Spalten oder Zeilen im k-Raum nicht wirklich als einen Abstand zwischen diesen vorstellen, in dem ‚Nichts' ist, sondern vielmehr als Breite der Zeile bzw. Spalte. Der Abstand gibt also die Entfernung vom Anfang einer Zeile bis zum Anfang der nächsten Zeile an. Dazwischen ist kein Freiraum.

Die zweite Möglichkeit den k-Raum zu füllen läßt die Gesamthöhe des k-Raums konstant. Dies wird durch Erhöhung des Abstandes der Linien im k-Raum entlang der Richtung des Phasenkodiergradienten erreicht. Da der k-Raum seine ursprüngliche Größe beibehält, bleibt auch die Auflösung gleich. Allerdings muß sich dann zwangsläufig die Ausschnittsgröße des Bildes (entlang einer Richtung) um den Faktor 4 verkleinern, wie wir gleich sehen werden.

Um die Reduzierung der Bildgröße besser zu verstehen, schauen wir uns erneut das Analogon aus der Optik an. Normalerweise sind die Daten in der Fouriertransformationsebene kontinuierlich. Eine Segmentierung in diskrete Linien entspricht dem Auferlegen eines Gitters auf die Fouriertransformationsebene. Dieses Gitter streut das aus der Linse ausfallende Licht, so daß mehrere Kopien des Originalbildes entstehen (Abbildung 74). Diese Kopien werden auch Bilder höherer Ordnung genannt. Man kann dieses Phänomen leicht simulieren, indem man sich eine Nylonstrumpfhose über den Kopf zieht und im Dunkeln gegen eine Straßenlaterne schaut. Auch dann wird man mehrere Kopien der Laterne sehen. Dabei ist der Abstand der Kopien höherer Ordnung

umgekehrt proportional zum Abstand der Gitterlinien (bzw. Nylonfasern) [MEZR95]. Je weiter auseinander die Gitterlinien verlaufen, desto näher liegen die Kopien höherer Ordnung. Rücken die Kopien zu nah aneinander, so kommt es zu einer Überlappung. Das gleiche gilt für den k-Raum. Zur Überlappung – oder Einfaltung – kommt es, wenn der gewählte Ausschnittsbereich zu groß, die Phasenkodierschritte zu klein oder der Patient falsch positioniert ist. An der Konsole erscheint dem Benutzer die Überlappung so, als wäre ein Teil des Bildes an einer Seite abgeschnitten und an der anderen angefügt. Dieses Phänomen kann man sich am besten erklären, wenn man sich die Bildebene viel größer denk als der Monitor dies vorgibt. Der Monitor ist also nur ein kleiner Ausschnitt aus der Bildebene. Auf dieser liegen das Originalbild und die Kopien höherer Ordnung. Wenn eine Überlappung sichtbar ist, liegen die Kopien höherer Ordnung nicht außerhalb des Monitorbereichs, sondern sind ebenfalls (zumindest zum Teil) sichtbar (Abbildung 75).

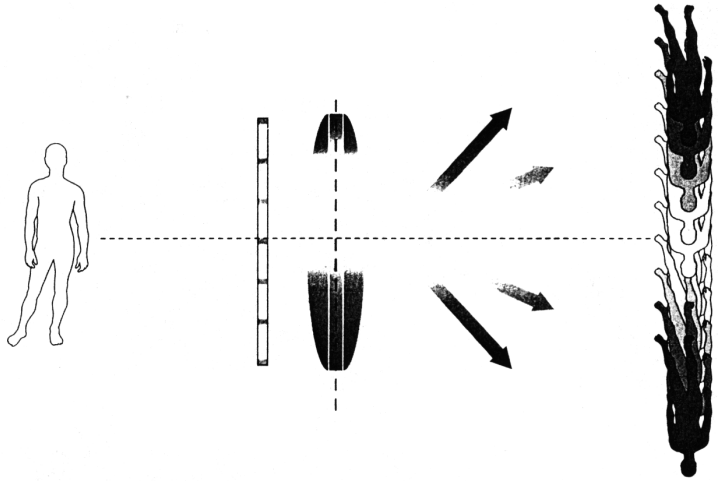

Abbildung 74: Entstehung der Kopien höherer Ordnung [MEZR95]

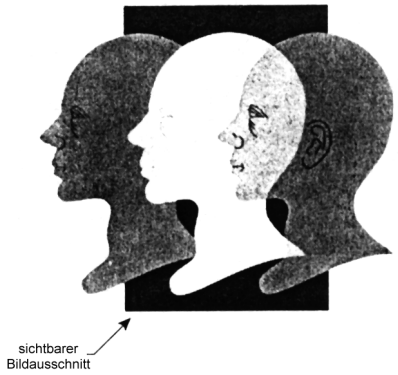

Abbildung 75: Sichtbarer Bildausschnitt [MEZR95]

Was man dagegen machen kann, kann man sich am besten an einem anderen Beispiel vorstellen. Man denke sich eine elastische Linse mit einem in ihrer Oberfläche verankerten Gitter. Durch Auseinanderziehen der Linse erreicht man eine bessere Auflösung des Bildes, muß aber ggf. Überlappungen zwischen dem Originalbild und den Kopien in Kauf nehmen. Durch Zusammendrücken der Linse rücken die Bilder weiter auseinander, die Auflösung des Bildes wird jedoch schlechter. Eine andere Lösung besteht darin, die Linsengröße beizubehalten, jedoch mehr Gitterlinien zu verwenden. In der Kernspintomographie entspricht das der Aufnahme von mehr Linien im k-Raum. Das erhöht jedoch wiederum die Akquisitionszeit. Die folgenden Abschnitte sollen ein Grundverständnis für die Zusammenhänge zwischen der Anzahl der Linien im k-Raum und im Bild und der Auflösung bzw. Größe des Bildes vermitteln.

Grundvoraussetzung ist zunächst einmal, daß der ausgewählte Bildausschnitt (FOV = Field of View) nicht größer ist als der Abstand zwischen den Kopien des Bildes. Das heißt nichts anderes, als daß der Bildausschnitt kleiner werden sollte, wenn der Abstand der Linien im k-Raum größer gewählt wird. Das kann durch folgende Beziehung beschrieben werden, wobei *D* der Abstand der Linien im k-Raum ist.

$$FOV \propto 1/D \tag{29}$$

Dabei wird angenommen, daß die Anzahl der Linien im k-Raum konstant bleibt. Das heißt, wenn mehr Linien akquiriert werden, so wird auch der k-Raum größer. Außerdem gilt folgende Beziehung zwischen dem Linienabstand *d* im Bild und der Größe des k-Raums, die durch die maximale Frequenz k_{max} gekennzeichnet wird:

$$k_{\max} \propto 1/d \tag{30}$$

Wenn also die Anzahl der Zeilen im k-Raum größer wird, wird der Linienabstand (bzw. die Linienbreite) im Bild kleiner und somit verbessert sich die Auflösung. Darüber hinaus kann man den Linienabstand *d* im Bild und den Linienabstand *D* im k-Raum folgendermaßen berechnen (N_i bzw. N_k ist die Anzahl der Zeilen im Bild bzw. im k-Raum):

$$d = \frac{FOV}{N_i} \tag{31}$$

$$D = \frac{k_{\max}}{N_k} \tag{32}$$

Mit diesem Wissen kann man zeigen, daß die Anzahl der Linien im Bild gleich der Anzahl der Linien im k-Raum sein muß, um Überlappungen der Bildkopien zu vermeiden:

$$N_i = N_k \tag{33}$$

Blicken wir nun noch mal auf die beiden Probleme am Anfang zurück. Behält man die Größe des k-Raums bei, vermindert allerdings die Anzahl der Zeilen im k-Raum, so müssen sich proportional auch die Anzahl der Zeilen im Bild und der abgebildete Bereich (FOV) verringern, um die Auflösung beizubehalten. Wird dagegen die Größe des k-Raums und die Anzahl der Linien im k-Raum verkleinert (das ist der Fall, wenn *D* konstant bleibt), so bleibt die Bildgröße (FOV) erhalten, jedoch reduziert sich die Auflösung.

3.5.2 Halb-Fourier- und Halb-Echo-Technik

Bisher haben wir nur Möglichkeiten zur Akquisitionszeitverkürzung kennengelernt, die mit einer Reduzierung der Bildauflösung oder Bildgröße verbunden sind. Es gibt jedoch eine Ausnahme, die durch die Symmetrie der Daten im k-Raum bedingt ist. Dies kann man zum Beispiel in Abbildung 70 erkennen. Betrachten wir zur Veranschaulichung zunächst wieder die Verhältnisse in der Optik. Verdeckt man die Hälfte einer Kameralinse (beispielsweise die linke Hälfte), so wird man feststellen, daß man trotzdem ein vollständiges Bild erhält. Das einzige was auffällt ist, daß die Helligkeit um die Hälfte reduziert ist, da nur die Hälfte der Lichtmenge einfallen kann. Geht man so sorgfältig vor, daß man die Mittellinie der Linse nicht verdeckt, so wird man auch keine Veränderung im Kontrast des Bildes feststellen können. Auch die vertikale Auflösung des Bildes ist unverändert. Die horizontale Auflösung ist allerdings halbiert. Dies sollte nach den Betrachtungen im vorigen Abschnitt klar sein. Verdeckt man nun anstelle der linken Hälfte die rechte, so wird man keine Änderung des Bildes feststellen können. Die Daten in der linken und rechten Hälfte der Fouriertransformationsebene bzw. des k-Raums sind nämlich identisch. Genau genommen liegt eine Punktsymmetrie zum Zentrum vor. Das bedeutet wiederum, daß die Hälfte der Daten im k-Raum redundant sind. Nicht ganz, schließlich führt eine Verdopplung der Zeilen im k-Raum zu einer Verdopplung der Auflösung. Aber was spricht dagegen, während der Datenakquisition nur die Hälfte des k-Raums zu füllen und diese Daten einfach in die andere Hälfte des k-Raums zu kopieren. Genau genommen muß etwas mehr als die Hälfte der Daten akquiriert werden, denn die Aufnahme der kompletten mittleren Zeile ist für eine Erhaltung des Kontrastes unabdingbar. Außerdem gilt die Symmetrie nur theoretisch bzw. nur unter absolut idealen Bedingungen. Schon sehr kleine Magnetfeldinhomogenitäten führen dazu, daß die Symmetrie nicht mehr zu 100 Prozent gegeben ist.

Aber auch dieses Verfahren hat einen kleinen Nachteil. Und zwar ist das die Reduzierung des Signal-zu-Rausch-Verhältnisses um den Faktor $\sqrt{2}$. Dies resultiert daraus, daß nur das halbe Probenvolumen gemessen wird (bzw. nur halb so oft) und neben einer Halbierung der Helligkeit des eigentlichen Signals auch eine Halbierung des Rauschens um den Faktor $\sqrt{2}$ auftritt. Das scheint zunächst recht

unsinnig, hat jedoch mit einem komplizierten statistischen Gesetz (Brownsche Bewegungstheorie) zu tun und sollte hier einfach hingenommen werden.

Kann aufgrund der gewählten Akquisitionstechnik nicht soviel Signal-zu-Rausch-Verhältnis entbehrt werden, so hat man immer noch die Möglichkeit einen Kompromiß einzugehen, indem man beispielsweise ¾ des k-Raums füllt und das übrige Viertel aus den akquirierten Daten rekonstruiert und damit nur einen Verlust des Signal-zu-Rausch-Verhältnisses um 15% erhält.

Neben der Möglichkeit, nur die halbe Anzahl an Zeilen des k-Raums zu akquirieren, gibt es auch die Möglichkeit, alle Zeilen mit nur halber Länge zu akquirieren. Das entspricht einer Verdeckung der oberen oder unteren Hälfte einer Kameralinse. Die **Halb-Echo-Technik** nutzt dieses Verfahren aus. Anstatt das gesamte Echo zu akquirieren, wird nach der Hälfte abgebrochen und die Akquisition für die nächste Zeile gestartet. Auch dieses Verfahren hat eine Verminderung des Signal-zu-Rausch-Verhältnisses um 41% zur Folge. Durch die Reduzierung der T_E-Zeit entstehen jedoch weniger Suszeptibilitätsartefakte (Kapitel 3.6).

3.5.3 Schnellbildtechniken Teil II (Turbo-Spin-Echo)

Der Grundgedanke der Turbo-Spin-Echo-Technik ist der, mit einem 90°-Impuls mehr als eine Linie des k-Raums zu füllen. Die dazu verwendete Pulssequenz weist pro 90°-Impuls mehrere 180°-Impluse auf, die jeweils ein Echo erzeugen. (Abbildung 76).

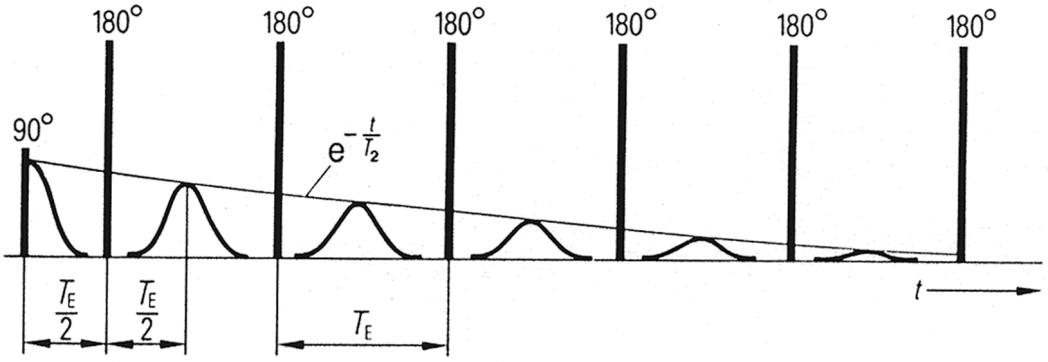

Abbildung 76: Pulsfolge und Signalverlauf einer schnellen Spin-Echo-Technik [MORN95]

Die Formel zur Berechnung der Signalintensität ist

$$I = N(H) \cdot (e^{-n*T_E/T_2}) \cdot (1 - e^{-T_R/T_1}) \qquad (34)$$

mit

I	Signalintensität
$N(H)$	Protonendichte
n	Echonummer
T_R	Repititionszeit
T_E	Echozeit
T_1	Spin-Gitter-Relaxationszeit und
T_2	Spin-Spin-Relaxationszeit.

Der Unterschied zur Formel für die konventionelle Spin-Echo-Sequenz ist der Faktor *n*. Die Signalintensität nimmt exponentiell mit jedem Echo ab.

Durch die Wahl von der Parameter *Echo-Train-Length* (ETL), T_{Eeff} und *Echo-Spacing* (ESP) können die Bildeigenschaften beeinflußt werden. ETL beschreibt die Anzahl der Echos, die innerhalb eines Repititionszyklus erzeugt werden. Eine Turbo-Spin-Echo-Sequenz kann abhängig von der ETL die Bildakquisition erheblich beschleunigen. Ein gutes Beispiel dafür aus [HASH97] geht von folgenden Einstellungen aus:

- T_R = 3000 ms
- N_y (Anzahl Phasenkodierschritte) = 256
- NEX (Anzahl der Anregungen) = 1

Die Dauer einer Akquisition mit einer Spin-Echo-Sequenz beträgt damit

$$3000ms \cdot 256 \cdot 1 = 12{,}8 \text{ Minuten}$$

Mit einer Turbo-Spin-Echo-Sequenz und einer ETL von 8 dauert die Akquisition nur noch

$$\frac{3000ms \cdot 256 \cdot 1}{8} = 1{,}6 \text{ Minuten}$$

ESP beschreibt den Abstand zwischen den einzelnen Echos. Die effektive Echozeit T_{Eeff} ist keine richtige Echozeit im Sinne der Standard-Spin-Echo-Sequenz. Sie ergibt sich vielmehr aus der Anordnung der einzelnen gemessenen Signale im k-Raum.

Bei jedem 180°-Impuls wird der Phasenkodiergradient erhöht, so daß der Inhalt einer anderen Linie des k-Raums gefüllt wird. Im Extremfall könnte man den kompletten k-Raum mit einem einzigen 90°-Impuls gefolgt von entsprechend vielen 180°-Impulsen füllen. Dagegen spricht jedoch der mit der Zeit fortschreitende T_2-Zerfall der angeregten Spins. Dieser führt dazu, daß die Amplitude jeder Linie im k-Raum proportional zur Zeit zwischen Anregung und Messung der Linie kleiner wird. Die Amplitude nimmt exponentiell um den Faktor e^{-T_E/T_2} ab, so daß die Signalamplitude ab einem bestimmten T_E im Signalrauschen verschwindet. Wie man sich leicht vorstellen kann, verändert sich mit den unterschiedlichen Amplituden im k-Raum auch die Bildqualität. Diese hängt entscheidend davon ab, wie der k-Raum gefüllt wird. Füllt man die Linien von einer Seite zur anderen nacheinander, so wird man schon in der Mitte des k-Raums (dort wo der Kontrast des Bildes bestimmt wird) kaum eine Signalamplitude wahrnehmen können. Wir nehmen dabei nun immer noch den (sicherlich ungünstigen) Fall an, daß der ganze k-Raum mit einem 90°-Impuls gefüllt wird. Neben dem sequentiellen Füllen bietet sich aber auch die Möglichkeit, den k-Raum auf eine beliebige andere Art und Weise zu füllen.

Bei der Standard Spin-Echo-Technik kann jede akquirierte Zeile, wann auch immer sie akquiriert wurde, überall im k-Raum plaziert werden, ohne die Bildqualität zu beeinflussen. Da bei den schnellen Spin-Echo-Techniken jedoch jede Zeile eine andere Amplitude aufweist, spielt es sehr wohl eine Rolle, wo jede Zeile im k-Raum plaziert wird.

Nehmen wir noch einmal den Fall an, daß der k-Raum von einer Seite zur anderen gefüllt wird. Verwendet man eine Echozeit von 20 ms für das erste Echo und weitere 20 ms für die anderen Echos und hat man ein Gewebe mit einer T_2-Zeit von 50 ms, so wird die Amplitude im Zentrum des k-Raums um den Faktor 10^{21} kleiner sein, als in der ersten Zeile (vorausgesetzt, es werde 256 Echos gemessen). Für die 128ste Zeile ergibt sich dann eine effektive Echozeit von 128·20 ms = 2560 ms. Es ergibt sich damit eine extrem T_2-gewichtete und kantenbetonte Aufnahme [MEZR95].

Eine Verbesserung des Bildkontrastes ergibt sich, wenn man zuerst die zentralen Linien des k-Raums füllt, von dort aus zunächst zu einer Seite hin weiter füllt und dann an der anderen Seite beginnt, den k-Raum bis zur Mitte zu füllen (Abbildung 77). Somit haben die zentralen Zeilen in etwa die gleiche Signalintensität wie bei der konventionellen Spin-Echo-Technik und der Kontrast entspricht somit ebenfalls dem eines konventionellen Bildes. Allerdings verschlechtert sich die Auflösung. Da die Randzeilen im Rauschen verschwinden, kommt es zu einer effektiven Verkleinerung des k-Raums. Angenommen man hat ein Gewebe mit einer T_2-Zeit von 100 ms und man verwendet eine Echozeit von 20 ms. Dann verschwindet das Signal nach etwa 15 akquirierten Zeilen im Rauschen. Dies ist etwa zum Zeitpunkt $3 \cdot T_2$ (hier 3·100 ms = 300 ms = 15·20 ms) der Fall [MEZR95].

Eine schöne Eigenschaft dieser Technik ist, daß der T_2-Kontrast, der normalerweise über die Echo- und Repititionszeit eingestellt wird, hier durch den Zeitpunkt bestimmt wird, zu dem das Zentrum des

k-Raums gefüllt wird. Sequentielles Füllen des k-Raums vom Zentrum zum Rand ergibt Bilder, die normalerweise mit kurzer T_E- und langer T_R-Zeit erreicht werden. Füllen des k-Raums vom einen Rand zum anderen ergibt Bilder, die einer langen T_E- und einer langen T_R-Zeit entsprechen.

Insbesondere bei der letztgenannten Technik tritt das Problem auf, daß man sehr starke Intensitätsunterschiede im k-Raum hat. In Abbildung 77 ist dieser Sprung beim Übergang von der zentralen Zeile (G_P=0) zur links davon liegenden Zeile (G_P leicht negativ) zu erkennen. Dieser Sprung entspricht in etwa einem dicken Kratzer in einer Linse und hat somit auch die gleichen Artefakte zur Folge: Geisterbilder und Schleier.

Zur Lösung der Probleme bezüglich Auflösung und Artefakten gibt es zwei Ansätze, die aufeinander aufbauen. Die erste Modifikation bezüglich der bisher beschriebenen Techniken füllt mit einem 90°-Impuls nicht den gesamten k-Raum, sondern nur einige Zeilen. Es sind dann natürlich mehrere 90°-Impulse notwendig, um den k-Raum komplett zu füllen. 4 bis 16 180°-Impulse nach jedem 90°-Impuls haben sich als geeignet erwiesen. Bei einer Auflösung von 256 Zeilen sind dann natürlich 16 bis 64 Akquisitionen notwendig, was sich wiederum in der Gesamtakquisitionszeit niederschlägt.

Die Erweiterung dieses Vorgehens besteht darin, den k-Raum nicht sequentiell, sondern symmetrisch zu füllen (Abbildung 78). Das heißt, daß jeweils das Ergebnis der ersten Messung der 16 Akquisitionen nahe dem Zentrum des k-Raums plaziert wird. Alle diese Messungen werden mit T_E=20 ms durchgeführt, so daß man zu diesem Zeitpunkt noch die volle Signalamplitude hat, so daß ein guter Kontrast des Bildes auf jeden Fall gegeben ist. Die Ergebnisse der jeweils zweiten Messung der 16 Akquisitionen werden symmetrisch rechts und links des Zentrums verteilt, so daß auf jeder Seite 8 weitere Zeilen eingefügt werden, die jeweils mit T_E=40 ms aufgenommen werden. So verfährt man weiter, bis schließlich jeweils die äußeren 8 Zeilen (auf beiden Seiten) aus den letzten Messungen der 16 Akquisitionen stammen. Diese Messungen wurden dann jeweils mit T_E=320 ms aufgenommen und weisen somit die niedrigste Signalamplitude auf.

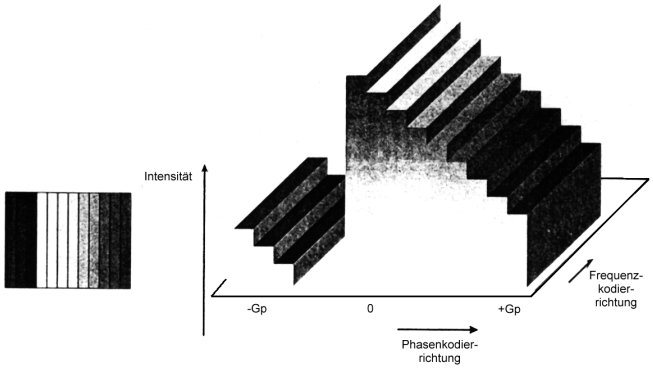

Abbildung 77: 1. Verbesserung der Spin-Echo-Technik durch Füllen des k-Raums vom Zentrum aus [MEZR95]

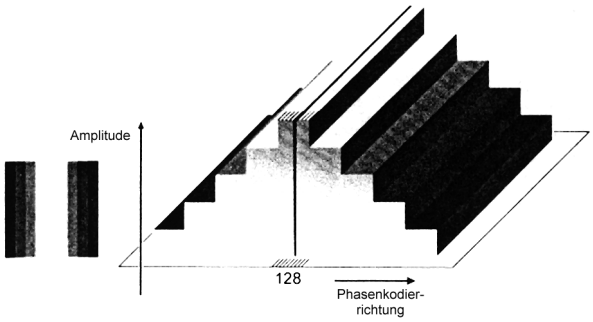

Abbildung 78: Symmetrisches Füllen des k-Raums [MEZR95]

Dies vermeidet extreme Sprünge im Signalverlauf im k-Raum, wie sie in Abbildung 77 zu sehen sind. Auf diese Weise werden die dadurch bedingten Artefakte weitestgehend reduziert. Es wird eine Bildqualität erreicht, die nahe an die Qualität herankommt, die mit der Standard Spin-Echo-Techik erreicht wird, nur ist die Turbo-Spin-Echo-Technik wesentlich schneller.

Einige Einschränkungen muß man aber trotzdem machen: Durch die geringere Intensität der Zeilen am Rand des k-Raums sehen die Bilder weicher aus, haben eine etwas geringere Auflösung und glattere Kanten. Zwar konnten die Intensitätssprünge im k-Raum deutlich verringert werden, kleine Sprünge sind jedoch immer noch vorhanden. Auch das führt zu leichten Artefakten, wie sie zum Beispiel durch kleine Kratzer auf einer Kameralinse verursacht werden. Außerdem kommt es zu einer geringen Reduzierung des Kontrastes, denn schließlich sind nicht nur die mittleren 8 Zeilen des k-Raums für den Bildkontrast verantwortlich. Dadurch können auch Anomalien schlechter sichtbar sein. Diesen Kompromiß zwischen Bildqualität und Akquisitionsgeschwindigkeit sollte man jedoch eingehen können.

3.6 Artefakte und Störungen der Bilderzeugung

Es gibt eine Vielzahl von möglichen Artefakten und Störungen, die bei einer Bildakquisition in der Magnetresonanztomographie auftreten können. Im Rahmen dieser Arbeit wollen wir jedoch nur auf einige wenige kurz eingehen, die entweder in der aktuellen Version unseres Simulationsprogrammes realisiert sind oder in einer späteren implementiert werden sollen. Wir wollen hier nur kurz auf Rauschen, Suszeptibilitätsartefakte, Einfaltung, Bewegungsartefakte und Flußartefakte eingehen. Für weitergehende Informationen kann man z.B. in [LISS90] oder [HASH97] nachlesen.

Rauschen

Jede Messung in der MRT wird durch ein Rauschen gestört, das zum einen durch die thermische Bewegung in der Nachweisspule für die Kernresonanz und zum anderen durch die Brownsche Bewegung der Moleküle im Meßobjekt entsteht [MORN95]. Wichtig für die Stärke des Rauschens und damit für die Qualität des Bildes ist das Signal-zu-Rausch-Verhältnis.

Für das Signal-zu-Rausch-Verhältnis gilt folgende Proportionalitätsbeziehung:

$$SNR \propto V \cdot \sqrt{\frac{N_y \cdot NEX}{BW}} \tag{35}$$

mit

SNR	Signal-zu-Rausch-Verhältnis,
V	Volumen eines Voxels,
N_y	Anzahl Phasenkodierschritte,
NEX	Anzahl der Messung (Number of excitations),
BW	Bandbreite des für die Messung benutzten HF-Impulses.

Die Parameter, die hier einfließen, kann man nur im begrenzten Umfang für ein maximales Signal-zu-Rausch-Verhältnis optimieren, da sie wiederum andere Eigenschaften der Messung beeinflußen. So führt eine kleinere Bandbreite zwar zu einem besseren Signal-zu-Rausch-Verhältnis, erhöht aber auch die Wahrscheinlichkeit von Einfaltungen, auf die wir später in diesem Kapitel noch eingehen wollen.

Eine Erhöhung der Anzahl der Anregungen (NEX) oder der Anzahl der Phasenkodierschritte (N_y) erhöht das Signal-zu-Rausch-Verhältnis ebenfalls, allerdings auch die Akquisitionszeit. Schließlich kann noch das Volumen eines einzelnen Voxels vergrößert werden, um das Signal-zu-Rausch-Verhältnis zu verbessern, allerdings ist eine gröbere Aufnahmematrix auch nicht unbedingt wünschenswert.

Suszeptibilitätsartefakte

Die Gewebeeigenschaft Suszeptibilität, die schon in Kapitel 3.1 beschrieben worden ist, führt bei Verwendung von Gradientenechosequenzen (Kapitel 3.3.2) ebenfalls zur Entstehung von Artefakten. An der Grenzfläche zweier Gewebebestandteile mit unterschiedlicher Suszeptibilität kommt es zu einer Änderung der magnetischen Induktion, die ein lokales Gradientenfeld erzeugt, wie es in Abbildung 79 dargestellt ist.

χ_a ist die Suszeptibilität für das Gewebe auf der linken Seite, χ_b die Suszeptibilität für das Gewebe rechts. Da $\chi_a < \chi_b$ entsteht ein lokaler Gradient G_{xyz}. Wegen der Larmorbeziehung ändert sich auch die Präzessionsfrequenz der Spins, was zur Dephasierung und damit zur Reduktion der transversalen Magnetisierung führt. Das Signal an Übergängen zwischen Substanzen unterschiedlicher Suszeptibilität wird also schwächer. Die stärksten Artefakte sind an Wasser-Luft-Grenzflächen zu erwarten, da Wasser den höchsten Diamagnetismus im Körper aufweist und der Diamagnetismus von Luft dem von materiefreien Raum gleichgesetzt werden kann ($\chi = 0$). Paramagnetische Substanzen weisen hohe Suszeptibilitätsdifferenzen zu allen Diamagnetika auf, daher treten z.B. bei Blutungen im Rahmen der Methämoglobinbildung oder bei Kontrastmittelgabe Suszeptibilitätsartefakte auf.

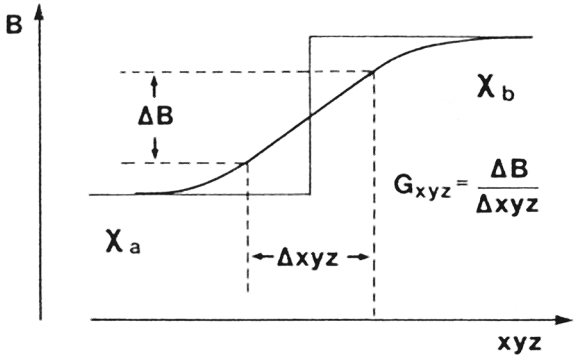

Abbildung 79: Änderung der magnetischen Induktion an Gewebegrenzflächen unterschiedlicher Suszeptibilität [LISS90]

Konventionelle Pulssequenzen wie Spin-Echo und Inversion-Recovery enthalten einen 180°-Rephasierungsimpuls, daher gleichen sie die Dephasierung der Spins durch Suszeptibilitätsdifferenzen aus, bei Gradienten-Echo entfällt dieser jedoch. Daher treten bei Gradientenechotechniken Suszeptibilitätsartefakte auf.

Einfaltung

Das Auftreten dieses Artefaktes hängt mit der Art und Weise zusammen, in der die Signale der Messung ausgelesen und vom Computer weiterverarbeitet werden. Der in Kapitel 3.5 beschriebene k-Raum wird nicht mit dem analogen Signal gefüllt, da der Rechner nur diskrete Werte weiterverarbeiten kann. Aus diesem Grund wird das Signal digitalisiert, indem in äquidistanten Zeitabständen der k-Raum mit dem Wert des Signals gefüllt wird. Diesen Vorgang bezeichnet man als „Sampling“.

Zum Verständnis des Einfaltungsartefaktes wollen wir zunächst noch einmal die Ortskodierung in Erinnerung rufen.

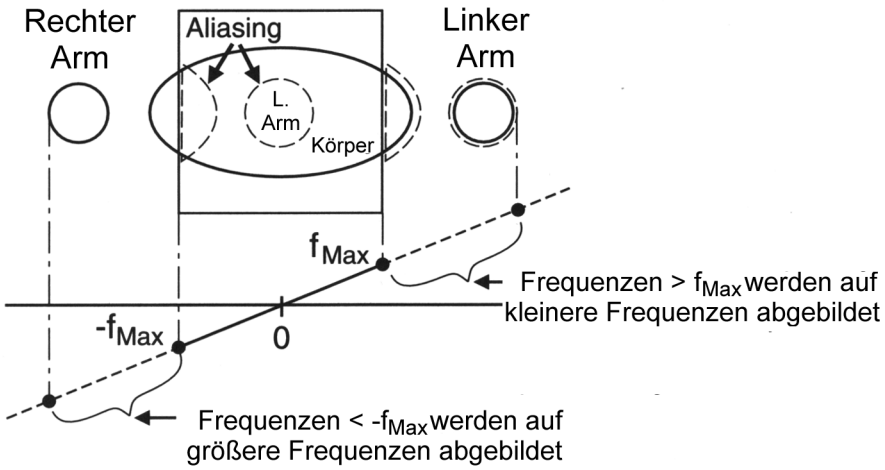

Abbildung 80: Einfaltungen entlang der Frequenzkodierrichtung [HASH97]

Abbildung 80 zeigt am Beispiel einer Aufnahme des Abdomens die Ortskodierung in *x*-Richtung. Diese funktioniert mit Hilfe von Gradienten, die dem eigentlichen Magnetfeld überlagert werden. Auf der rechten Seite hat der Gradient die Frequenz f_{max} auf der anderen Seite $-f_{max}$. Die Frequenz f_{max} hängt von der Bandbreite ab, die wiederum das „Field of View“ (FOV) festlegt. Alle Frequenzen, die kleiner als $-f_{max}$ und größer als f_{max} sind, können nicht korrekt erkannt werden.

Der Gradient ist aber auch außerhalb des FOV geschaltet, damit tragen in Abbildung 80 auch der rechte und linke Arm und ein Teil des Körpers, die in dieser Aufnahme gar nicht im FOV liegen, zum Signal bei. Da die Frequenzen jedoch außerhalb des Intervalls $[-f_{max}, f_{max}]$ liegen, kann die Frequenz

nicht korrekt erkannt werden, es kommt zu einer örtlichen Fehlkodierung, so erscheint u.a. der linke Arm mitten im Bild, obwohl er außerhalb des FOV liegt. Diesen Vorgang nennt man Einfaltung.

Bewegungsartefakte

Bewegungsartefakte werden durch willlkürliche oder unwillkürliche Bewegungen des Patienten (Wechsel der Position, Atmung, Schlucken, Husten usw.) während der Bildakquisition hervorgerufen. Die Artefakte treten nur in Phasenkodierrichtung auf, nicht dagegen in Frequenzkodierrichtung. Das liegt an der Dauer der Frequenz- und Phasenkodierung. Während die Frequenzkodierung nur einige Millisekunden dauert und damit die Bewegungen viel zu langsam sind, um diesen Prozeß zu stören, benötigt ein einzelner Phasenkodierschritt eine Zeit im Sekundenbereich [HASH97].

Die Bewegung während der Bildakquisition bewirkt, daß die Phasenkodierung fehlkodiert wird und führt zu Verschmierungen im erzeugten Bild.

Flußartefakte

Das Signalverhalten von fließendem Blut oder von cerebrospinaler Flüssigkeit (CSF) ist schwer vorherzubestimmen, da es für die MRT von einer Vielzahl von Faktoren wie Geschwindigkeit, der benutzten Pulssequenz, dem Flipwinkel, der Gradientenstärke usw. abhängt. Insbesondere hängt das Signalverhalten beim Blutfluß auch von der Richtung des Flusses und vom Strömungsprofil im Blutgefäß ab. Näheres zu Flußartefakten kann der Leser z.B. in [LISS90] oder in [HASH97] nachlesen.

4 Konzept des virtuellen MRT

In diesem Kapitel und seinen Unterkapiteln soll anhand der Beschreibung eines realen MRT-Arbeitsplatzes ein Konzept zur Umsetzung in eine Simulation entwickelt werden.

4.1 Beschreibung des realen Arbeitsplatzes

Es gibt heutzutage eine Vielzahl von Herstellern von MRT-Geräten und somit auch eine breite Palette verschiedener MRT-Modelle. Trotz der Vielfalt kann man einige wesentliche gemeinsame Komponenten herausarbeiten. Im wesentlichen sind das

- Das Meßsystem (mit dem Magnetsystem, dem Gradientensystem und dem Hochfrequenzsystem),
- Eine Bedienkonsole (mit integriertem Computersystem) und
- Eine Auswertekonsole (ebenfalls mit integriertem Computersystem),

die ein MRT-System ausmachen (Abbildung 81).

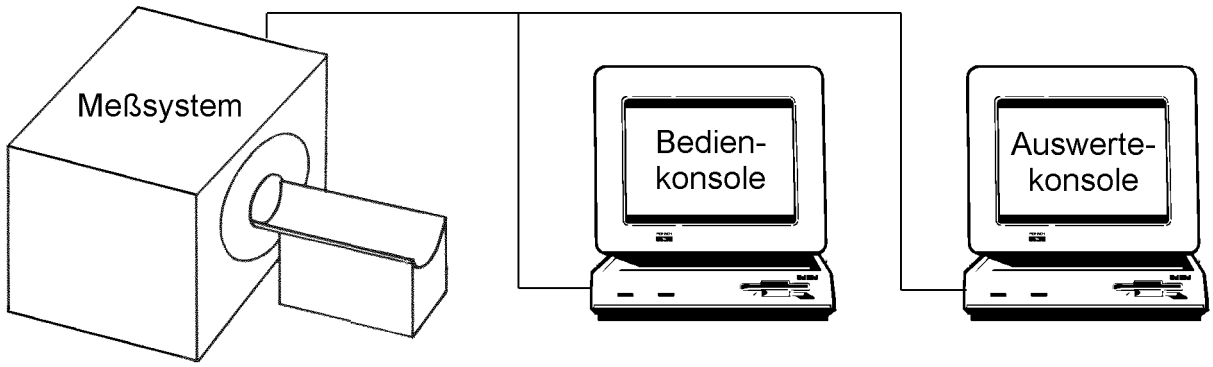

Abbildung 81: Reales MRT-System

Das Meßsystem befindet sich in einem gegen elektromagnetische Strahlung abgeschirmten Raum. Dies hat sowohl den Grund, daß keine äußere Strahlung das Magnetfeld des MRT stören kann, als auch daß die extrem hohe Strahlenemission des MRT nicht in die Umwelt gelangt. Zwar gibt es keine Beweise dafür, daß die Strahlung in dem Maße, wie sie bei der Kernspintomographie eingesetzt wird, für den menschlichen Körper schädlich ist, jedoch werden elektronische Geräte und Systeme sehr wohl durch die Strahlung beeinflußt, zumal sie im UKW-Bereich, also im Radiobereich liegt. Das Meßsystem erscheint nach außen als großer Kunststoffkasten mit einem röhrenförmigen Hohlraum, in den ein Patient zur Untersuchung hineingeschoben werden kann (Abbildung 82). Der hohle Innendurchmesser der Röhre wird von dem Magnetsystem (zur Erzeugung eines möglichst homogenen Grundfeldes), dem Gradientensystem (zur Überlagerung des Grundfeldes mit Gradientenfeldern zur Ortskodierung) und dem Hochfrequenzsystem (zum Senden der Anregungssignale und Empfangen der Resonanzsignale) umgeben. Aufgrund der starken Ströme, die zum Aufbau des Magnetfeldes verwendet werden und der daraus resultierenden Wärmeentwicklung befindet sich auch ein Kühlsystem im Kunststoffgehäuse der Röhre.

Da wir die Anregung der einzelnen Spins, wie sie in Kapitel 3.2 beschrieben wurde, aufgrund der hohen Anzahl von Protonen (1 ml Wasser enthält bereits $6{,}691 \cdot 10^{22}$ Protonen) nicht für jedes einzelne Proton simulieren können, wollen wir auf das eigentliche Meßsystem nicht weiter eingehen. Wir beschreiben in Kapitel 4.3.1, wie wir die für unsere Simulation benötigten Parameterbilder erhalten.

Die Bedien- und Auswertekonsole wollen wir jedoch sehr wohl simulieren, weshalb wir sie hier auch etwas detaillierter beschreiben wollen.

Von der Bedienkonsole hat der Benutzer direkt die Möglichkeit, daß Meßsystem zu steuern und zu Messungen zu veranlassen. Als Hilfsmittel dafür steht ihm ein Computersystem mit Monitor, Tastatur, Maus und einem Audio- und Video-Überwachungssystem zur Verfügung (Abbildung 83).

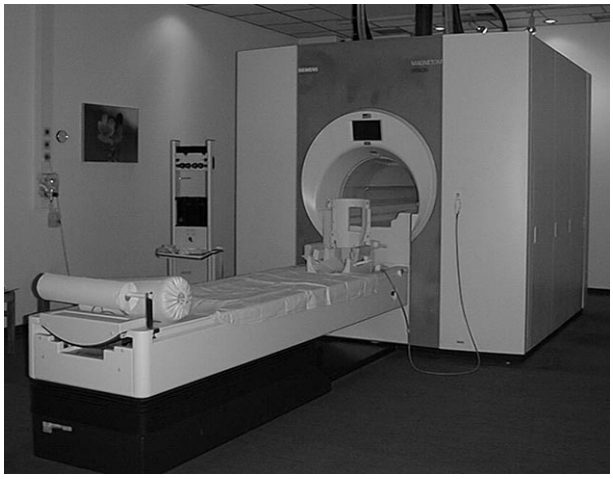

Abbildung 82: MRT-Meßsystem [LAUB90]

Abbildung 83: MRT-Arbeitsplatz

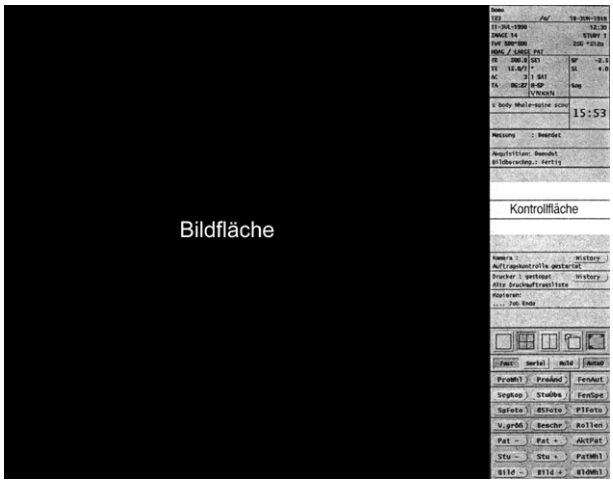

Abbildung 84: Hauptfenster der NUMARIS-Oberfläche

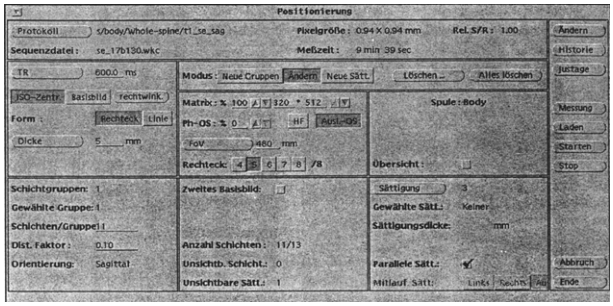

Abbildung 85: Dialogfenster der NUMARIS-Oberfläche

Auf dem Monitor wird dem Benutzer eine grafische Benutzeroberfläche (Arbeitsfläche) angeboten, über die der Tomograph mit Hilfe von Tastatur und Mauseingaben gesteuert werden kann. Ein Beispiel für eine solche Arbeitsfläche ist in Abbildung 84 dargestellt. Diese gliedert sich in eine

- Bildfläche und eine
- Kontrollfläche.

Auf der Bildfläche können maximal 4 Bilder gleichzeitig dargestellt werden, wobei ein Bild als aktiv markiert werden kann. Auf dieses aktive Segment beziehen sich dann alle Einstellungen, die der Benutzer in der Kontrollfläche vornimmt. Die Kontrollfläche unterteilt sich in die Bereiche

- Bildtextfeld,
- Meßtextfeld,
- Meldungsfeld,
- Schalterfeld und
- Funktionsfeld,

wie dies aus Abbildung 84 ersichtlich ist.

Das Bildtextfeld enthält den Standardtext zum Bild im aktiven Segment. Dazu gehören die wichtigsten Paramtereinstellungen, mit denen das Bild aufgezeichnet wurde, sowie das Aufnahmedatum. Das Bildtextfeld kann weder geändert noch abgeschaltet werden.

Im Meßtextfeld erscheint der zur jeweiligen Messung gehörende Text. Dazu zählen insbesondere Statusmeldungen der aktuellen Messung. Während einer physiologisch gesteuerten Messung (z.B. durch den Herzrhythmus) werden in diesem Feld auch die physiologischen Signale (z.B. EKG, Puls) des Patienten dargestellt.

Im Meldungsfeld erscheinen Meldungen von Funktionen, die ohne Funktionsfenster im Hintergrund arbeiten. Das sind z.B. das Übertragen von Bildern an einen Filmbelichter oder einen Papierdrucker oder das Kopieren von Daten auf einen anderen Computer.

Das Schalterfeld enthält häufig benötigte Schalter zur Bilddarstellung. Hierzu gehören insbesondere Schalter zum Wechseln der Anzahl der gleichzeitig darstellbaren Bilder.

Das Funktionsfeld enthält Funktionsknöpfe häufig benutzter Funktionen. Alle Funktionen sind auch über kontextsentive Menüs, die durch Betätigung der rechten Maustaste erscheinen, erreichbar. Das Funktionfeld enthält nur eine Auswahl aller Funktionen und soll eine schnellere Zugriffsmöglichkeit zu diesen Funktionen bieten.

Neben den Bedienelementen auf der Kontrollfläche kann im Arbeitsverlauf eine weitere Kontrollfläche erscheinen, die die untere Hälfte der Bildfläche verdeckt. Eine solche Kontrollfläche erscheint zum Beispiel dann, wenn ein Patient registriert werden soll (d.h. dessen persönliche Daten wie Name, Geburtsdatum, Gewicht, etc. werden dem System mitgeteilt) oder wenn die Parameter einer Pulssequenz eingestellt werden sollen (Abbildung 85).

Die Auswertekonsole stellt sich dem Benutzer in gleicher Weise dar. Der Benutzer kann von dieser Konsole jedoch nicht das Meßsystem steuern, sondern lediglich auf bereits aufgenommene Bilder zugreifen. Aus diesem Grund steht an der Auswertekonsole auch kein Audio- und Video-Überwachungssystem zur Verfügung.

Das Audio- und Video-Überwachungssystem dient insbesondere zur Überwachung des physischen und psychischen Zustands des Patienten. Über einen Melder, den der Patient vor der Untersuchung in die Hand bekommt, kann er ein Signal auslösen, falls er aus irgendwelchen Gründen ernsthafte Probleme hat. Das Personal kann dann mit dem Patienten kommunizieren und ggf. erforderliche Maßnahmen einleiten. Die Überwachungskamera dient dem Benutzer insbesondere auch dazu, Bewegungen des Patienten festzustellen, die die aufgenommenen Bilder unbrauchbar machen können. In unserer Simulation lassen wir das Audio- und Video-Überwachungssystem jedoch erst einmal außen vor und beschränken uns auf die restlichen Komponenten.

In der Regel verläuft die Arbeit am MRT so, daß von der Bedienkonsole einzig und allein die Aufnahmen gemacht werden. Jede Art von Weiterverarbeitung der Bilder findet dann an der Auswertekonsole statt. Theoretisch bestände jedoch auch die Möglichkeit, die Auswertung der Daten bereits an der Bedienkonsole vorzunehmen. Jedoch ist ein Benutzer in der Regel reichlich damit beschäftigt, die richtigen Pulssequenzen für die Messung auszuwählen, deren Parameter geeignet einzustellen und sinnvolle Schichtführungen auszuwählen. Während einer Untersuchung werden nämlich meist verschiedene Messungen durchgeführt. Dabei kann z.B. die Pulssequenz oder die Schichtführung variieren und es können Aufnahmen mit und ohne Kontrastmittel gemacht werden. Während eine Messung läuft, bereitet der Benutzer in der Regel schon die nächste Messung vor, um sie direkt nach dem Ende der vorherigen Messung starten zu können. Somit bleibt dem Benutzer der Bedienkonsole meist keine Zeit, auch noch eine Auswahl und Optimierung (z.B. durch Fensterung) der Bilder zwecks Befundung vorzunehmen.

4.2 Prinzipien des GUI-Entwurfs

Wir haben im vorangegangenen Kapitel schon einige Beispielbilder dafür gesehen, wie sich die grafische Benutzeroberfläche (graphical user interface, GUI) eines realen MRT-Arbeitsplatzes darstellt (Abbildung 84 und Abbildung 85). Wir wollen nun die allgemeinen Prinzipien für den Entwurf einer grafischen Benutzeroberfläche erörtern und dabei erklären und begründen, welche dieser Prinzipien wir davon umsetzen wollen und welche nicht.

Im Mittelpunkt dieser Betrachtungen werden die Begriffe Aufgabenangemessenheit, Selbsterklärungsfähigkeit, Steuerbarkeit, Erwartungskonformität, Fehlerrobustheit, Individualisierbarkeit und Erlernbarkeit stehen. Zunächst müssen jedoch einige grundlegende Begriffe und das allgemeine Vorgehen beim Entwurf einer grafischen Benutzeroberfläche erklärt werden.

Allgemeine Vorgehensweise bei der Softwareentwicklung

Im Zentrum bei der Entwicklung eines Softwareprodukts und insbesondere einer grafischen Benutzeroberfläche steht der Benutzer. Der Benutzer ist laut DIN-Norm eine Person in der Rolle eines Auftraggebers oder eines Auftragnehmers gegenüber einem Rechensystem [HORN98]. In unserem Fall stammte die Idee zur Entwicklung eines virtuellen MRT von Dr. Hackländer, welcher im Bereich Radiologie am Klinikum Wuppertal-Barmen GmbH leitender Oberarzt ist. Er definierte im wesentlichen die Anforderungen an den virtuellen MRT aus medizinischer Sicht. Insbesondere machte er einige Vorgaben, die die grafische Benutzeroberfläche betreffen – dazu im Verlauf dieses Kapitels mehr.

Dem Benutzer gilt es, eine ergonomische Oberfläche zu präsentieren. Das heißt nichts anderes, als daß alle nötigen Voraussetzungen für ein optimales Zusammenwirken von Mensch und Betriebsmitteln (in diesem Fall der Computer mit der darauf installierten Software) geschaffen bzw. verbessert werden sollen. Dabei gilt es, die Hardware des Systems so zu gestalten, daß die Gesundheit des Anwenders nicht beeinträchtigt wird, und die Software an die physischen und psychischen Eigenschaften des Menschen anzupassen.

Die Entwicklung einer grafischen Benutzeroberfläche durchläuft verschiedene Phasen, wobei durchaus Rückschritte von einer Phase zur vorherigen nötig sein können. Die Phasen sind im einzelnen [HORN98]:

- Aufgabenanalyse
- Aufgabenmodellierung
- Dialogspezifikation
- Prototypkonstruktion
- Evaluierung

Bei der Aufgabenanalyse wird das zukünftige Einsatzgebiet der Software untersucht. Nach Festlegung der zukünftigen Anwender geschieht diese Informationsbeschaffung durch Befragung. Insbesondere wird analysiert, wie die von der neuen Software zu lösenden Aufgaben heute ausgeführt werden.

In unserem Fall sind die Anwender unseres Systems insbesondere Medizin-Studenten und MTAs, die an die Arbeit mit einem MRT herangeführt werden sollen, ohne dabei mit einem realen System zu arbeiten, da dies immense Kosten verursachen würde. Weitere Anwendergruppen sind natürlich auch Personen, die im Umgang mit realen Systemen schon geübt sind und den Auszubildenden mit Hilfe des virtuellen MRT den Umgang mit solchen Systemen vermitteln wollen. Denkbar wäre eine Anwendung z.B. auch für Pulssequenzentwickler, also Personen, die versuchen, neue Pulssequenzen zu entwickeln, um beispielsweise Kontraste besser darstellen zu können.

Bei der Aufgabenmodellierung werden die zuvor gesammelten Informationen strukturiert und nach Wichtigkeit sortiert. Eine detaillierte Beschreibung der funktionalen Konzepte der einzelnen Komponenten des Systems folgt in Kapitel 4.3.

Bei der Dialogspezifikation mit der sich dieses Kapitel schwerpunktmäßig befaßt, wird das Systemverhalten und insbesondere das Erscheinungsbild der Benutzerdialoge genau beschrieben. Um ein Arbeiten in die falsche Richtung zu vermeiden ist es sinnvoll, in dieser Phase in engem Kontakt mit dem Auftraggeber zu bleiben.

In der dann folgenden Phase der Prototypkonstruktion wird aufgrund der vorherigen Dialogspezifikation ein erster Prototyp entwickelt, der dem Auftraggeber vorgeführt wird. Detaillierte Informationen dazu befinden sich in Kapitel 5. Es findet dann schließlich die letzte Phase – nämlich die Evaluierung – statt. Zumeist wird es nötig sein, den Prototyp dann noch einmal zu überarbeiten und dem Auftraggeber erneut vorzustellen. Dieser Vorgang wird so lange durchgeführt, bis der Auftraggeber vollkommen zufrieden ist. Kleine Verbesserungen können auch noch während der endgültigen Implementierung des Systems vorgenommen werden. Kapitel 6 enthält einige Informationen zur Evaluierung des fertigen Systems.

Gestaltung einer grafischen Benutzeroberfläche

Im Zentrum der Gestaltung einer grafischen Benutzeroberfläche steht das Fenster (engl.: window). Es ist ein rechteckiges, umrandetes Feld, welches einige Grundfunktionalitäten bietet. Dazu gehört das Verändern der Größe, das Verschieben und das Schließen. Ein Anwendungsfenster sollte im allgemeinen in die folgenden eindeutigen Fensterbereiche unterteilt sein (Abbildung 86):

- Fensterinformationsbereich,
- Befehlsbereich,
- Arbeitsbereich und
- Systemmeldungsbereich (Statusleiste).

Im Fensterinformationsbereich wird der Name (engl.: title) des Fensters dargestellt. Außerdem befinden sich dort einige Schaltflächen (engl.: button), die es ermöglichen, das Fenster zu schließen, zu ikonifizieren und zwischen Vollbildmodus und voreingestellter Größe umzuschalten.

Im Befehlsbereich befindet sich eine Menüleiste. Außerdem kann auch eine Symboleiste vorhanden sein, die die wichtigsten Funktionen durch kleine Schaltflächen erreichbar macht.

Der Arbeitsbereich stellt das Zentrum der Anwendung dar. Daher ist es sehr wichtig, diesen Bereich besonders sorgfältig zu gestalten.

Der Systemmeldungsbereich gibt dem Anwender Informationen über den Systemzustand. Dort findet er Hinweise über die Funktionalität einiger Bedienelemente und wird über den Systemstatus und eventuelle Probleme informiert.

Wir haben uns in Anlehnung an einen realen MRT-Arbeitsplatz (Kapitel 4.1) dazu entschieden, das Konzept noch weiter zu präzisieren. Wir unterteilen den Arbeitsbereich nämlich weiter in

- eine Bildfläche,
- einen ständig sichtbaren Kontrollbereich und
- einen weiteren, ein- und ausblendbaren Kontrollbereich.

In den Hintergrund tritt dagegen die Bedeutung des Befehlsbereichs. Dieses Konzept ist in Abbildung 87 dargestellt. Die Bildfläche soll quadratisch[18] sein und entweder 1 Bild oder 4 Bilder gleichzeitig darstellen können. Der ständig sichtbare Kontrollbereich soll Zugriff auf die wichtigsten

[18] Die Bildfläche hat die feste Größe von 512·512 Bildpunkten.

Systemfunktionen bieten. Der ein- und ausblendbare Kontrollbereich soll dagegen die Einstellungsmöglichkeiten für die Pulssequenzen enthalten (siehe auch Kapitel 5 und 6). Dies entspricht in etwa der in Kapitel 4.1 beschriebenen Gestaltung der Benutzeroberfläche eines realen Arbeitsplatzes. Eine möglichst realitätsnahe Umsetzung ist eines unserer wichtigsten Ziele bei der Oberflächengestaltung. Da die Oberfläche eines realen Systems allerdings sehr speziell, komplex, gewöhnungsbedürftig und anfangs verwirrend ist, versuchen wir auch dem Anwender den Zugang zu unserer Software zu erleichtern, der nur den Umgang mit Standardsoftware gewohnt ist. Wir bewegen uns dabei auf einem zugegebenermaßen sehr dünnen Grad.

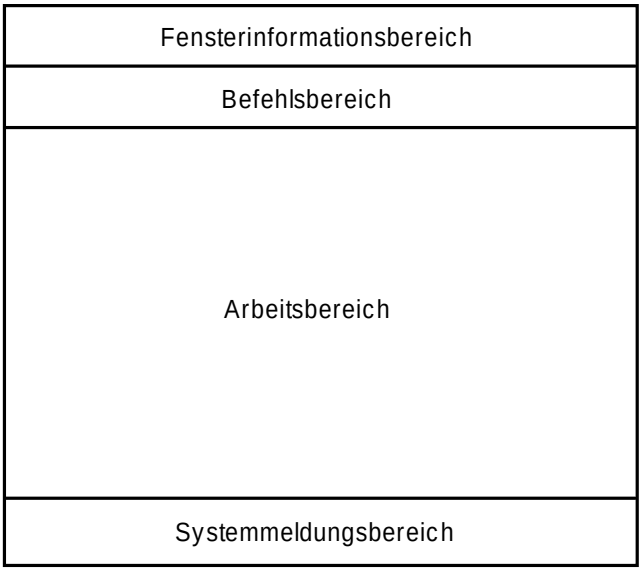

Abbildung 86: Allgemeiner Fensteraufbau

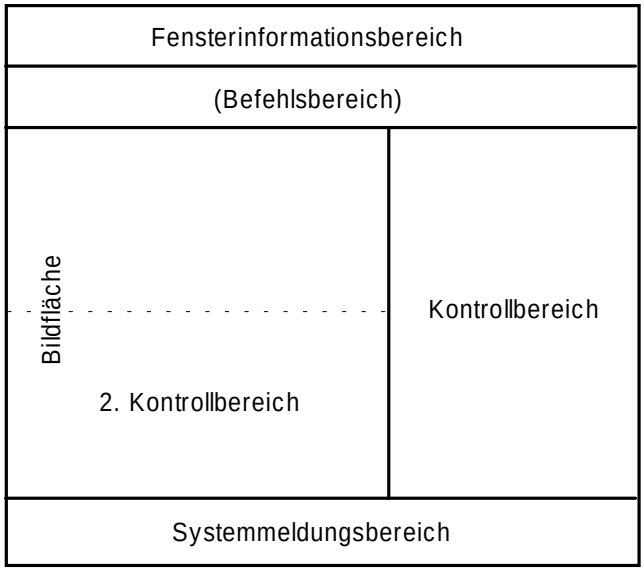

Abbildung 87: Spezieller Fensteraufbau

Die Einzelheiten dieses Konzepts wollen wir anhand der eingangs erwähnten zentralen Begriffen für die Gestaltung einer grafischen Benutzeroberfläche erläutern. Siehe dazu auch [HORN98].

Aufgabenangemessenheit

Aufgabenangemessenheit bedeutet, den Benutzer bei der Erledigung seiner Arbeitsaufgaben zu unterstützen, anstatt ihn mit den Eigenschaften des Systems zu belasten. Dazu beitragen kann die Anpassungsfähigkeit des Systems an wiederkehrende Aufgaben und z.B. die Voreinstellung von sinnvollen Werten für bestimmte Parameter. Diese beiden Eigenschaften wollen auch wir dem Benutzer bieten, zum einen durch die Möglichkeit, wichtige Einstellungen abspeichern zu können und zum anderen dadurch, daß sämtliche Parameter (Fensterung, Simulationszeit, Pulssequenzparameter) sinnvoll voreingestellt sind.

Selbsterklärungsfähigkeit

Mit Selbsterklärungsfähigkeit ist gemeint, daß dem Benutzer auf Verlangen Leistungsumfang und Einsatzzweck des Systems mitgeteilt werden können. Dazu gehören auch aussagekräftige Beschriftungen und Piktogramme der einzelnen Funktionselemente, so daß dem Benutzer unmittelbar verständlich ist, was sich hinter den einzelnen Funktionen verbirgt. Insbesondere ist zu beachten, daß Piktogramme einen höheren Wiedererkennungseffekt haben als Beschriftungen oder gar Abkürzungen.

Im Rahmen unserer Diplomarbeit können wir leider kein komplettes Hilfe-System anbieten. Zum Verständnis des Einsatzzwecks und des Leistungsumfangs des Systems steht dem Benutzer die vorliegende Ausarbeitung zur Verfügung. Zusätzlich soll er durch Systemmeldungen in der Statusleiste und Infofelder (engl.: tooltip) für jede Funktion unterstützt werden.

Steuerbarkeit

Steuerbarkeit bedeutet, daß der Benutzer die Geschwindigkeit der Ausführung steuern können soll und außerdem die Reihenfolge und Menge von Ein- und Ausgaben festlegen können soll.

Diesen Aspekt wollen wir insbesondere bei der Simulation der Aufnahmezeit realisieren. Eine reale MRT-Aufnahme dauert in der Regel einige Minuten, in Ausnahmefällen nur einige Sekunden oder aber auch mehr als eine Stunde. Wir wollen dem Benutzer die Möglichkeit geben, die reale Aufnahmezeit simulieren zu können. Jedoch soll es ihm auch möglich sein, die Berechnungen so schnell wie möglich durchzuführen. Schließlich ist es nicht in jeder Situation wünschenswert, die reale Zeit auf das Ergebnis der Simulation zu warten. Außerdem soll der Benutzer zu jedem Zeitpunkt in der Lage sein, die Simulation abzubrechen und anschließend neu zu starten. Somit behält der Benutzer stets die Kontrolle über das Geschehen.

Durch die sinnvoll voreingestellten Parameterwerte (siehe Aufgabenangemessenheit) soll der Benutzer die Möglichkeit haben, mit so wenig Eingaben wie möglich ein schon recht sinnvolles Ergebnis zu erzielen. Mit fortschreitendem Kenntnisstand des Benutzers kann dieser immer mehr Parameter manuell ändern und die Auswirkungen auf das Ergebnis beurteilen. Auch die Menge der Ausgaben soll steuerbar sein. Das betrifft insbesondere die Bildbeschriftungen, die im ganzen ein- und ausgeblendet werden können. Für spätere Versionen ist es auch denkbar, daß der Benutzer eine Teilmenge aller möglichen Ausgaben zur Darstellung freigeben kann (siehe auch Kapitel 8).

Erwartungskonformität

Die Erwartungskonformität ist eine sowohl wichtige als auch in unserem Fall eine schwierig zu erreichende Anforderung an das System. Erwartungskonformität bedeutet, daß der Dialog den Erwartungen des Benutzers entspricht. Zum einen bezieht sich dies auf die Erwartungen, die der Benutzer aus der Arbeit mit anderen Systemen mitbringt, zum anderen aber auch auf die Erwartungen, die er während der Schulung und im Umgang mit dem System und dem Benutzerhandbuch gebildet hat. Darüber hinaus sollte die Bedienung innerhalb der Anwendung konsistent sein.

Wir wollen zum einen erreichen, daß ein Anwender, der bisher hauptsächlich mit Standardsoftware gearbeitet hat, einen leichten Einstieg in die neue Umgebung findet. Dies wird schon allein dadurch erreicht, daß unser System mit JAVA entwickelt wurde (Kapitel 5.1) und sich im Erscheinungsbild (engl.: look and feel) des jeweiligen Betriebssystems darstellt. Der Benutzer findet also auf jeden Fall ihm vertraut wirkende Bedienelemente vor. Zum anderen soll der Benutzer aber auch an eine reale MRT-Umgebung herangeführt werden. Das heißt, wenn er später wirklich an einem MRT-Arbeitsplatz arbeitet, soll er sich dort schon einigermaßen zurechtfinden können. Das Umgekehrte gilt natürlich für jemanden, der ständig an einem MRT-Arbeitsplatz arbeitet. Solche Personen sollen problemlos in der Lage sein, den virtuellen MRT zu bedienen.

Unterstützt wird die Erwartungskonformität auch durch direkte Rückmeldungen durch das System. So soll z.B. nach dem Start einer Berechnung direkt eine Fortschrittsanzeige gestartet werden. Da einige Berechnungen etwas länger dauern können, erhält der Benutzer so eine Rückmeldung über den Systemstatus.

Fehlerrobustheit

Fehlerrobustheit bedeutet, daß trotz fehlerhafter Eingaben das beabsichtigte Ergebnis mit minimalem Korrekturaufwand erreicht wird. Zu diesem Zweck müssen dem Benutzer die Fehler mitgeteilt und verständlich gemacht werden.

Auch das wollen wir realisieren. Für sämtliche Parameter ist ein erlaubtes Intervall festgelegt. Gibt der Benutzer Werte außerhalb dieses Intervalls ein, so wird der Benutzer durch einen Signalton auf den Fehler aufmerksam gemacht und ihm wird ein sinnvoller Wert im Bereich des erlaubten Intervalls vorgeschlagen. Parameter außerhalb dieses Intervalls sind nicht zulässig.

Ein anderer Aspekt ist die Fehlervermeidung, z.B. durch das Überprüfen, ob alle Änderungen vor dem Schließen der Anwendung gespeichert worden sind. In der aktuellen Implementierung konnten wir allerdings noch kein geeignetes Konzept integrieren, um neu erzeugte und veränderte Bilder mit wenig Aufwand abzuspeichern. Bisher muß jedes Bild einzeln von Hand gespeichert werden. Da aber unter Umständen eine ganze Menge neuer Bilder erzeugt wird, sollte ein Konzept gefunden werden, so daß

der Benutzer mit möglichst wenig Aufwand seine Arbeit sichern kann. Dieses Verhalten kann man für spätere Versionen andenken (siehe auch Kapitel 8).

Individualisierbarkeit

Mit Individualisierbarkeit ist gemeint, daß der Benutzer Anpassungen des Systems an seine Bedürfnisse vornehmen kann. Wir wollen dieses Verhalten dadurch unterstützen, daß wichtige Einstellungen des Systems abgespeichert werden können und somit nach einem Neustart wieder hergestellt werden können.

Erlernbarkeit

Erlernbarkeit bedeutet, daß der Benutzer in seinen Lernstrategien unterstützt wird. Da unser System ein Lehr- und Lernsystem darstellt, stellt die Erlernbarkeit natürlich auch eine zentrale Anforderung dar. In erster Linie wird die Erlernbarkeit dadurch unterstützt, daß alle Funktionen einfach zu erreichen sind und das gesamte System ohne großen Aufwand zu bedienen ist. Im einzelnen tragen natürlich auch die unter den zuvor genannten Punkten erwähnten Eigenschaften erheblich zu einer guten Erlernbarkeit bei.

4.3 Komponenten des Systems

Der reale Arbeitsplatz am MRT besteht wie in Kapitel 4.1 beschrieben aus einer Meßstation und einer Nachbearbeitungsstation. Da die Benutzer des virtuellen MRT einen Eindruck von den realen Gegebenheiten auch am simulierten Meßgerät bekommen sollen, besteht das System des virtuellen MRT ebenfalls aus zwei Komponenten. Das Meßwerkzeug *Virtual MRT* simuliert die Bilderzeugung mit der Möglichkeit, die Parameter für die Pulssequenz beliebig einzustellen. So können angehende Mediziner oder Ärzte in der Fortbildung einen Zugang zu dieser recht komplexen Materie bekommen. Im Gegensatz zum realen MRT sind die Sequenzparameter nicht so restriktiv vorgegeben wie vom Hersteller solcher Geräte. Dies ermöglicht dem Benutzer, frei alle möglichen Parametereinstellungen auszuprobieren.

Mit der zweite Komponente, dem Nachbearbeitungswerkzeug, ist es möglich, einen ganzen Stapel von aufgenommenen Bildern einzuladen und einfache Operationen damit durchzuführen, wie z.B. die Berechnung einer anderen Schnittführung, Abstandsmessung etc.

Bevor wir aber auf diese beiden Komponenten unseres Systems näher eingehen, stellt sich die Frage, auf welchen Daten die Simulation eigentlich basiert. Im einleitenden Kapitel 3 wird die Erzeugung eines Bildes von der Anregung der Spins bis zum fertigen Bild erklärt. Nun enthält aber bereits ein Milliliter Wasser ungefähr $6{,}691 \cdot 10^{22}$ Wasserstoffprotonen. Die Simulation deren Anregung würde die Grenzen jeder Simulation unweigerlich sprengen. Wie soll also die Simulation des MRT funktionieren? Die Antwort auf diese Frage findet man in den Kapiteln über Pulssequenzen (3.3 und 3.5.3). Dort sind einige Formeln angegeben, mit deren Hilfe man für die jeweilige Pulssequenz aus den physikalischen Parametern eine Signalintensität berechnen kann. Die Parameterbilder werden aus realen Messungen berechnet. Sie können mit einem speziellen Zeichenprogramm nachbearbeitet werden. Dieses Zeichenprogramm ermöglicht das gleichzeitige Zeichnen in mehrere Bildebenen. Dabei repräsentieren die Bildebenen die einzelnen Parameter (PD, T_1, T_2, etc.). Dadurch ist es möglich, z.B. Pathologien in die Bilddaten einzufügen, indem man die gewünschten Parameterwerte für die Pathologie einstellt und dann die Konturen in die Parameterbilder einzeichnet.

Der erste Schritt zum Simulationswerkzeug ist also die Berechnung von Parameterbildern, auf die wir im nachfolgenden Kapitel näher eingehen wollen.

4.3.1 Konzept zur Berechnung der Parameterbilder

Die Parameterbilder kann man leider nicht durch eine einfache Messung aus dem MRT gewinnen. Das MRT erzeugt aus den gemessenen Signalen ein Intensitätsbild. Dieses Intensitätsbild wiederum hängt wie in Kapitel 3.3 beschrieben stets von mehreren physikalischen Parametern ab (T_1, T_2 und PD). Der

Einfluß (oder die Wichtung) der physikalischen Parameter wiederum hängt von der Sequenz und der Wahl der jeweiligen Geräteparameter ab (T_E, T_R, T_I, Flip-Winkel etc.).

Durch Messung von ein und derselben Schicht mit unterschiedlichen Parametereinstellungen bekommt man für jeden Voxel eine Intensitätskurve. Aus dieser Intensitätskurve kann man durch numerische Verfahren die physikalischen Größen berechnen.

Nach Rücksprache mit Dr. Hackländer und mit seinem ehemaligen Mitarbeiter Dr. Reichenbach fiel unsere Wahl auf eine Spin-Echo-16-Sequenz zur T_2- und PD-Berechnung und auf eine Turbo-Flash-Sequenz zur T_1-Berechnung. Mittels der Turbo-Flash-Sequenz sollten über ein Levenberg-Marquardt-Verfahren die Nullstellen und damit die T_1-Zeiten bestimmt werden. Da die von uns am MRT gewählte Turbo-Flash-Sequenz nicht die gewünschten Eigenschaften aufwies, verwenden wir zur T_1-Bestimmung eine Spin-Echo-Sequenz und berechnen die Parameterbilder mit einem gewichteten Verfahren der kleinsten Quadrate. Bei der Spin-Echo-16-Sequenz haben wir die gleiche Schicht mit 16 verschiedenen T_E-Zeiten gemessen, T_R blieb dabei konstant. Bei der Spin-Echo-Sequenz haben wir verschiedene Werte für das T_R der Spin-Echo-Sequenz gemessen und T_E konstant eingestellt.

Die Daten liegen nach der Messung im DICOM-Format vor. Aus diesen sollen dann mit Hilfe von geeigneten Approximationsverfahren die Parameterbilder berechnet werden. Dieses Modul muß also folgende Schritte durchführen:

- Import der DICOM-Daten
- Berechnung der Parameterbilder
- Export der berechneten Parameterbilder im DICOM-Format

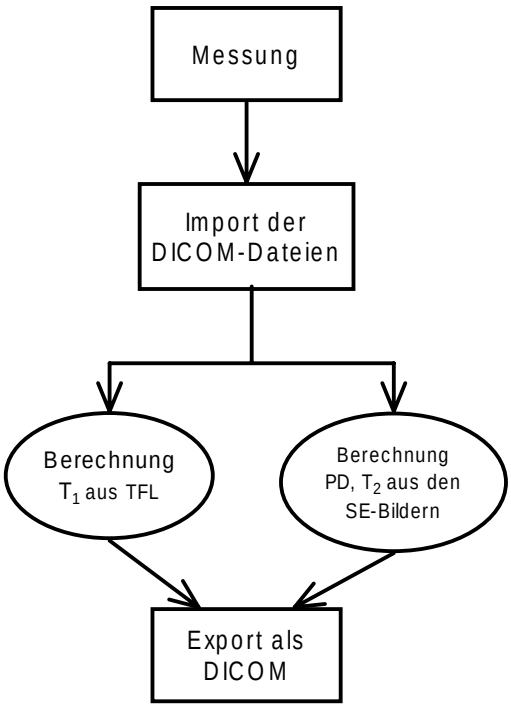

Abbildung 88: Schematische Darstellung des Werkzeuges für Parameterbilder

Auf das DICOM-Format gehen wir im Kapitel 5.3.1 noch näher ein. Es handelt sich um ein allgemeines Format für medizinische Daten, in dem auch die im MRT erzeugten Bilder abgelegt werden (siehe dazu auch [DICOM98]. Zur Berechnung der Parameterbilder aus diesen Messungen existiert eine Vielzahl von Verfahren. Wir verwenden als Algorithmus die gewichtete Methode der kleinsten Quadrate verwendet (siehe dazu z.B. [CONT72]). Dieses Verfahren wollen wir im nächsten Kapitel kurz beschreiben.

4.3.1.1 Die Methode der kleinsten Quadrate

Abbildung 89 zeigt ein Beispiel aus [CONT72]. Die Aufgabe des Verfahrens der kleinsten Quadrate ist die Bestimmung einer Ausgleichsfunktion mit möglichst geringer Abweichung von diesen Meßwerten.

Da die Messungen des MRT durch Rauschen und andere Meßeinflüsse fehlerbehaftet sind, sieht die Aufgabe zur Bestimmung von Parameterwerten nach Logarithmieren der gemessenen Signalintensitäten ähnlich wie in Abbildung 89 aus.

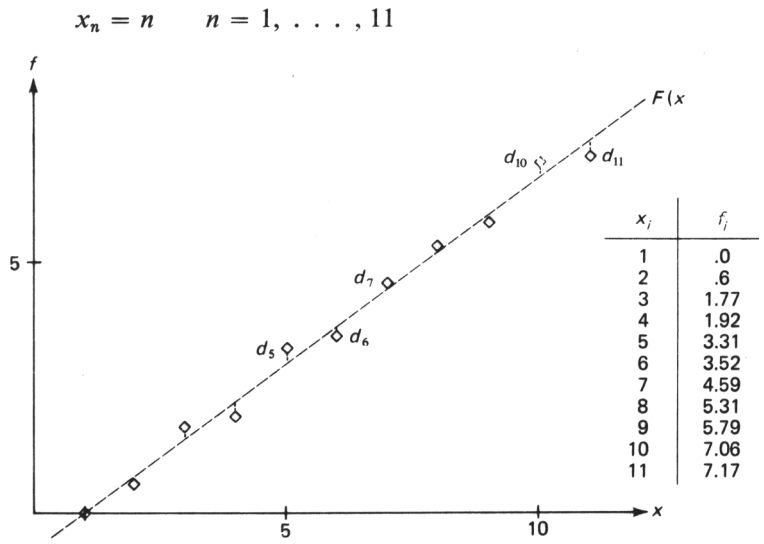

Abbildung 89: Beispiel einer Anwendung des kleinsten-Quadrate-Verfahrens zur Berechnung einer Ausgleichsgeraden [CONT72]

Die Methode der kleinsten Quadrate wählt zur Fehlerabschätzung eine Funktion der Form [CONT72]

$$F(x) = c_1\phi_1(x) + \ldots + c_k\phi_k(x) \tag{36}$$

Die c_i sind dabei Koeffizienten, die $\phi_i(x)$ beliebig gewählte Funktionen. Zur Abschätzung des Fehlers der Ausgleichsfunktion summiert man die Summe der Fehlerquadrate zwischen Ausgleichsfunktion und den Meßwerten.

$$E(c_1,\ldots c_k) = \sum_{n=1}^{N} [\, f_n - F(x_n; c_1,\ldots,c_k)]^2 \tag{37}$$

mit

E	Fehlerfunktion,
$c_1,\ldots,c_k$	Koeffizienten,
N	Anzahl Meßwerte,
f_n	Meßwert,
$F(x_n;c_1,c_k)$	Ausgleichsfunktion, abhängig von dem x-Wert des Meßwertes und der zu bestimmenden Koeffizienten.

Statt der Fehlerwerte werden deren Quadrate verwendet, da sich so die Fehlerbeträge in positiver und negativer Richtung nicht aufheben können. Eine notwendige Bedingung für Extremalstellen der Fehlerfunktion ist eine Nullstelle in der ersten Ableitung. Die erste Ableitung wird für jeden Koeffizienten gebildet. Es entsteht ein System von Gleichungen für $i = (1,\ldots,k)$:

$$\frac{\partial}{\partial c_i} E(c_i,\ldots,c_k) = \sum_{n=1}^{N} \frac{\partial}{\partial c_i} F(x_n; c_1,\ldots,c_k) = -2\sum_{n=1}^{N} [\, f_n - F(x_n; c_1,\ldots,c_k)] \cdot \frac{\partial}{\partial c_i} F(x_n; c_1,\ldots,c_k) \tag{38}$$

Wenn man die Gleichung 0 setzt und $F(x_n;c_1,...c_k)$ aus (36) einsetzt bekommt man eine notwendige Bedingung für ein Minimum, die sogenannten „Normal equations“ für $i = (1,...k)$:

$$-2\sum_{n=1}^{N}[f_n - F(x_n;c_1,\ldots,c_k)]\phi_i(x_n) = 0 \tag{39}$$

Erneutes Einsetzen von (36) ergibt nach einigen Umformungen das System für $i = (1,...k)$:

$$c_1\sum_{n=1}^{N}\phi_i(x_n)\cdot\phi_1(x_n)+\ldots+c_k\sum_{n=1}^{N}\phi_i(x_n)\cdot\phi_k(x_n) = \sum_{n=1}^{N} f_n\phi_i(x_n) \tag{40}$$

Nimmt man jetzt noch Gewichtsfaktoren hinzu, lautet Formel (37):

$$E(c_1,\ldots c_k) = \sum_{n=1}^{N}[f_n - F(x_n;c_1,\ldots,c_k)]^2\,\omega(x_n) \tag{41}$$

mit

E	Fehlerfunktion,
$c_1,...,c_k$	Koeffizienten,
N	Anzahl Meßwerte,
f_n	Meßwert,
$F(x_n;c_1,c_k)$	Ausgleichsfunktion, abhängig von dem x-Wert des Meßwertes und der zu bestimmenden Koeffizienten,
$\omega(x_n)$	Gewichtsfaktoren.

Die Umformungen wie oben auf (41) ergeben dann:

$$c_1\sum_{n=1}^{N}\phi_i(x_n)\cdot\phi_1(x_n)\cdot\omega(x_n)+\ldots+c_k\sum_{n=1}^{N}\phi_i(x_n)\cdot\phi_k(x_n)\cdot\omega(x_n) = \sum_{n=1}^{N} f_n\phi_i(x_n)\cdot\omega(x_n) \tag{42}$$

Dieses Gleichungssystem kann jetzt mit einem Lösungsverfahren, wie z.B. der Gauß-Elimination gelöst werden. Siehe zum Beispiel [CORM94], [SEDG92].

Die Formel für die Spin-Echo-Sequenz ist nach Logarithmieren linear und kann durch ein gewichtetes Verfahren der kleinsten Quadrate berechnet werden.

$$I = N(H)\cdot(e^{-T_E/T_2})\cdot(1-e^{-T_R/T_1}) \tag{43}$$

mit

I	Signalintensität,
$N(H)$	Protonendichte,
T_R	Repititionszeit,
T_E	Echozeit,
T_1	Spin-Gitter-Relaxationszeit,
T_2	Spin-Spin-Relaxationszeit.

Die Repititionszeit ist für die ganze Messung konstant, die Relaxationszeit T_1 ist für jedes Voxel während der Messung ebenfalls konstant, denn es wird immer nur ein und dieselbe Schicht gemessen. Daher setzen wir $k = (1-e^{-T_R/T_1})$. Nach dem Logarithmieren lautet daher die Formel für Spin-Echo:

$$\ln(I) = \ln(N(H)) - \frac{T_E}{T_2} + \ln(k) = (-\frac{1}{T_2})\cdot T_E + [\ln(N(H)) + \ln(k)] \tag{44}$$

Nach dem Berechnen der Koeffizienten mit dem Least-Square-Verfahren kann man die T_2-Zeit als negativen Kehrwert der Steigung der Ausgleichsgeraden berechnen. Die Protonendichte wurde durch Berechnen des y-Achsenabschnitts abgeschätzt.

4.3.2 Konzept des Meßwerkzeuges *Virtual MRT*

Nach der Beschreibung der Rohdatenakquisition wollen wir nun das Konzept des virtuellen Kernspintomographen vorstellen. Zu diesem Zweck soll zunächst einmal das Vorgehen des realen Kernspintomographen veranschaulicht werden.

Wie man in Abbildung 90 erkennen kann, ist es von der Anregung der Wasserstoffprotonen bis zur Bilderzeugung ein langer Weg. Zunächst wird das erzeugte Signal von Benutzereingaben (T_R, T_E, α etc.) (siehe dazu auch Kapitel 3.3) und von Gradientenschaltungen zur Ortskodierung (Kapitel 3.4) beeinflußt (oben links in der Abbildung). Wie schon mehrfach erwähnt, wird aus diesem Signal nicht etwa direkt ein Bild erzeugt. Stattdessen wird zunächst der k-Raum gefüllt, je nach Art der gewählten Pulssequenz benötigt man bis zu N_y Anregungen der Spins dazu.

Der gefüllte k-Raum wird nun einer inversen Fouriertransformation unterzogen, was durch die Ellipse mit der Beschriftung *IFFT* angedeutet ist. Durch diese Transformation entsteht ein fouriertransformiertes Bild mit Realteil und Imaginärteil. Aus diesen Bildanteilen wird schließlich das Intensitätsbild (Magnitudenbild) mittels der Formel $(r^2+i^2)^{1/2}$ berechnet, wobei r den Realteil und i den Imaginärteil darstellen soll.

Das angezeigte Bild besteht aus Graustufen, daher wird das Intensitätsbild zunächst auf ein 12-Bit-Bild konvertiert. Aus diesem 12-Bit-Bild kann der Benutzer ein Fenster auswählen, das durch die Werte *center* und *window* einen Bereich in den 12-Bit-Daten definiert, der letztendlich als 8-Bit-Graustufenbild auf dem Bildschirm dargestellt wird.

Die 12-Bit-Daten können im DICOM-Format gespeichert werden und erhalten neben der Fensterung zahlreiche Informationen über den Patienten, den untersuchenden Arzt, die Aufnahmeparameter usw.

Diese kurze Beschreibung der Funktionsweise eine MRT ist stark vereinfacht, zeigt aber die wesentlichen Schritte zur Bilderzeugung. Abbildung 90 enthält auch die Aspekte des MRT, die durch unser Simulationsprogramm nachgebildet werden sollen. Der Benutzer des Programms soll die Messung durch Wahl der Parameter beeinflussen können. Das aus dem Intensitätsbild berechnete 12-Bit-Bild im DICOM-Format muß der Benutzer der Simulation fenstern können und es ins DICOM-Format exportieren können.

Sehr interessant ist auch die Manipulation des k-Raums, siehe dazu [MEZR95]. In einer idealen Messung träten keine Störungen auf, aber in der Wirklichkeit verzerren sich die Meßwerte durch physikalische Effekte wie z.B. Rauschen oder Suszeptibilität. Auch die Meßapparatur selber (Magnetfeldinhomogenitäten) oder die Wahl der Pulssequenzen (z.B. Turbo-Spin-Echo) beeinflußen die Bildeigenschaften.

Viele dieser Störungen in der Messung können erst im k-Raum simuliert werden. Ziel des virtuellen Tomographen ist es schließlich auch, dem Anwender die komplexe Materie eines Kernspintomographen durch die Simulation näher zu bringen. Allein durch die Anzeige der k-Raum-Matrix zu einem berechneten Bild, kann der doch sehr unanschauliche Begriff des k-Raums dem Benutzer veranschaulicht werden. Für diese Aufgaben muß sich die Simulation daher auch mit der Fouriertransformation und mit der Manipulation des k-Raums befassen.

Da im Falle der Simulation wie schon erwähnt nicht die Signalerzeugung durch die Protonenspins simuliert werden kann, greifen wir jetzt auf die berechneten Parameterbilder zurück, die wir wie in Kapitel 4.3 beschrieben erzeugt haben.

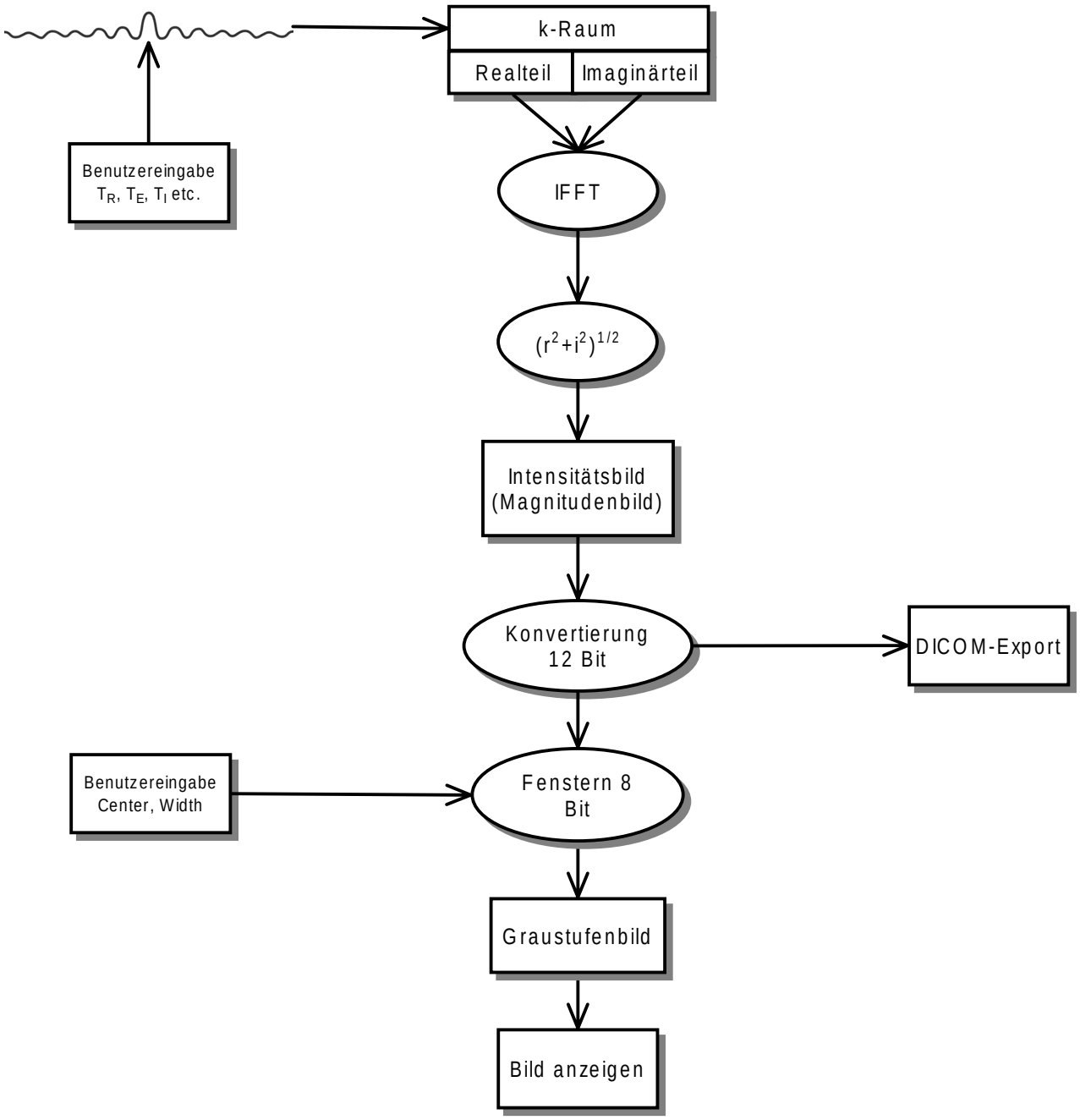

Abbildung 90: Ablaufdiagramm des realen Tomographen

Abbildung 91 soll die Funktionsweise des virtuellen Tomographen beschreiben. Im ersten Schritt werden die Parameterbilder mit einem DICOM-Importfilter eingelesen. Es werden T_1, T_2, Protonendichte (PD), Suszeptibilität (SZ) und Flußgeschwindigkeit (Flow) eingelesen.

Je nach gewählter Sequenz kann der Benutzer Geräteparameter für die Messung angeben. Mit der Rechenvorschrift aus der Formel für die jeweilige Pulssequenz wird aus diesen Parametern und den Parameterbildern ein Intensitätsbild errechnet. Die Effekte, die bei der realen Messung auftreten, können entweder im Bildraum oder im k-Raum simuliert werden. Falls k-Raum-Manipulationen benötigt werden, wird aus dem Intensitätsbild per Fouriertransformation der k-Raum berechnet. Der Realteil entspricht dem Intensitätsbild, der Imaginärteil wird auf 0 gesetzt. Werden keine k-Raum-Manipulationen benötig, so ist auch keine Fouriertransformation nötig und der Zustand A (Abbildung 91) wird eingenommen.

Nach der k-Raum-Berechnung funktioniert die Berechnung des Graustufenbildes genau wie im realen Tomographen (siehe auch Abbildung 92). Aus dem k-Raum werden durch eine inverse

Fouriertransformation Imaginär- und Realteil berechnet, das Intensitätsbild wird dann als Magnitudenbild aus diesen beiden Anteilen errechnet. Das Intensitätsbild wird auf 12-Bit konvertiert und zur Anzeige gefenstert. Der Benutzer wählt mit *center* und *window* einen Ausschnitt aus den 12-Bit-Daten, der auf 8-Bit abgebildet wird. Das 12-Bit-Bild kann zur Weiterbearbeitung mit dem Nachbearbeitungswerkzeug über einen DICOM-Filter exportiert werden. Auch jedes DICOM-Anzeigeprogramm wie z.B. MAGICVIEW kann die mit dem Werkzeug *Virtual MRT* erzeugten Bilder anzeigen. Wurden keine k-Raum-Manipulationen durchgeführt, so wird die Berechnung der inversen Fouriertransformation übersprungen und direkt die Konvertierung auf 12-Bit vorgenommen. Dies entspricht dem Einstiegspunkt von Zustand A in Abbildung 92.

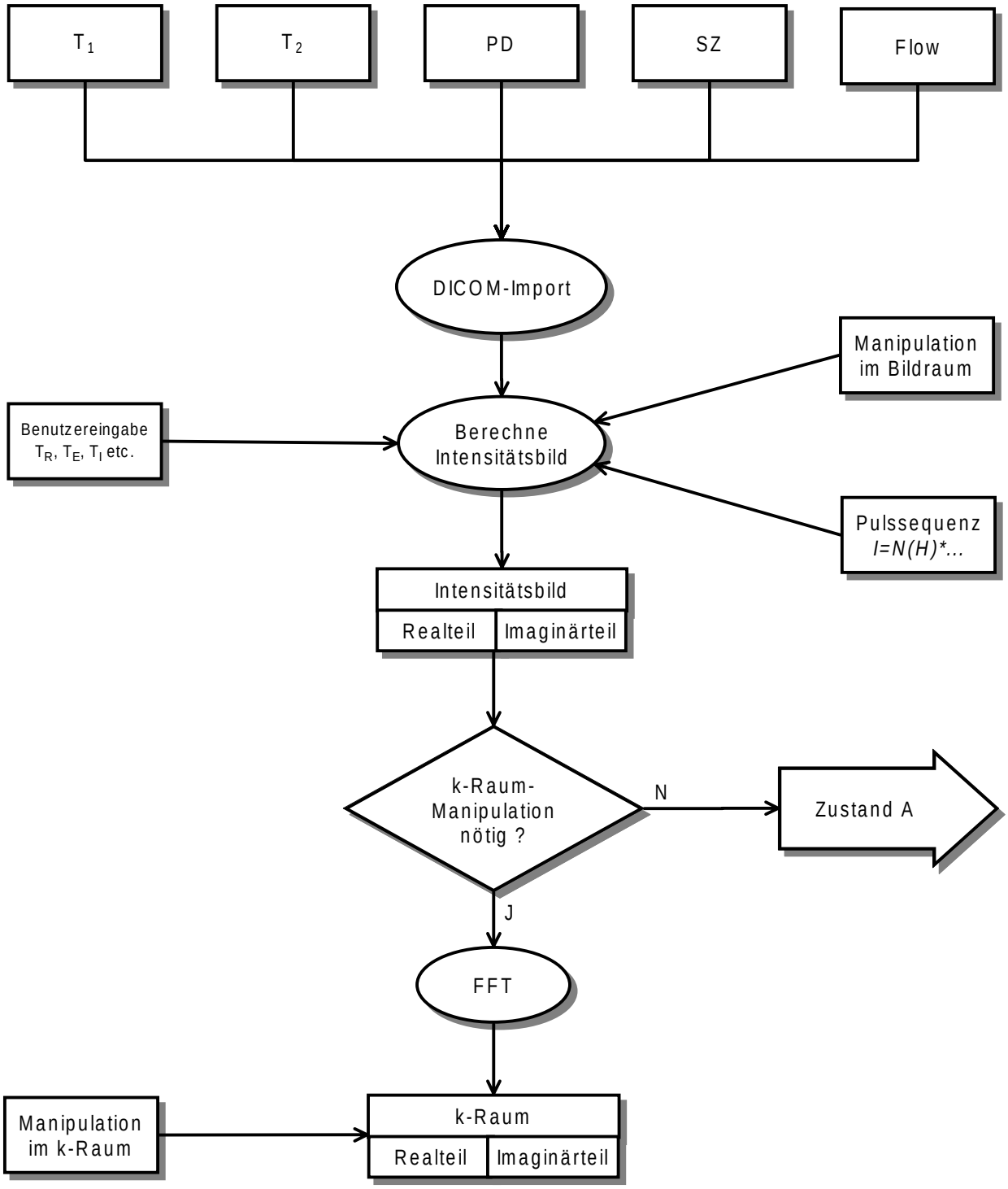

Abbildung 91: Funktion des virtuellen MRT

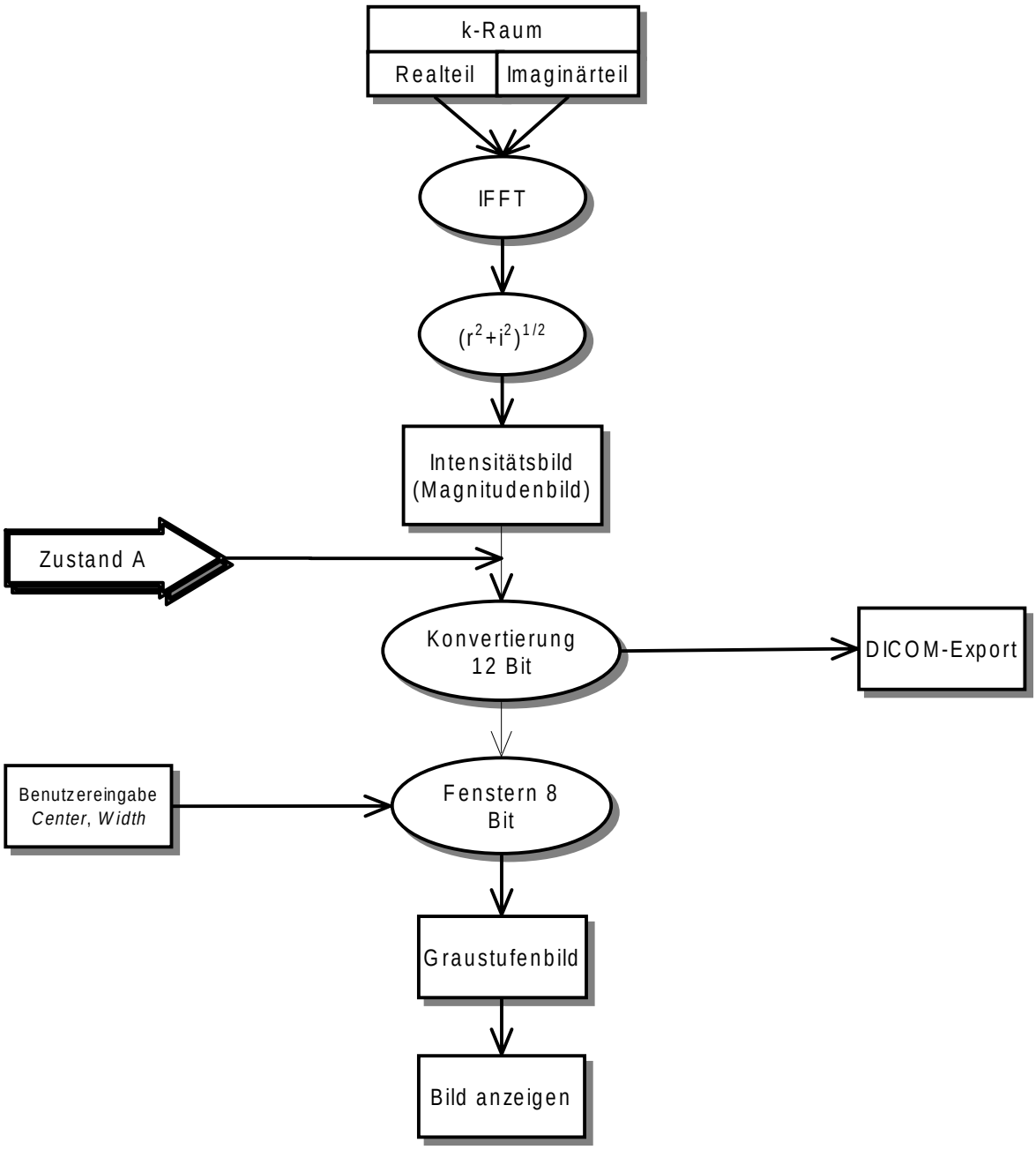

Abbildung 92: Berechnung des Graustufenbildes aus den Daten im k-Raum

4.3.3 Konzept des Nachbearbeitungswerkzeuges *DICOM-Viewer*

Das im letzten Kapitel beschriebene Meßwerkzeug *Virtual MRT* dient der reinen Bilderzeugung und der Speicherung der erzeugten Bilder im DICOM-Format. Es simuliert demnach die Bedienkonsole eines realen MRT, also die Konsole, von der aus das Meßsystem gesteuert und die Bildaufnahmen durchgeführt werden.

Die aufgenommenen Bilddaten werden dann zunächst gespeichert und können schließlich an einer anderen Konsole, der Auswertekonsole, betrachtet und weiterverarbeitet werden (siehe auch Kapitel 4.1). Genau diese Konsole wird durch unser Nachbearbeitungswerkzeug, welches wir *DICOM-Viewer* nennen, repräsentiert. Es wäre ohne weiteres möglich gewesen, beide Komponenten des Systems in einer einzigen Komponente zusammenzufassen, jedoch sollte die Trennung der beiden Konsolen ganz klar hervorgehoben werden.

Der *DICOM-Viewer* soll die Möglichkeit bieten, ein einzelnes DICOM-Bild oder eine ganze Serie von DICOM-Bildern zu laden , diese darzustellen und sie anschließend weiterzuverarbeiten. Bearbeitete und neu erzeugte Bilder sollen dann wieder im DICOM-Format gespeichert werden können. Die Integration des *DICOM-Viewers* in das Gesamtkonzept ist in Abbildung 93 dargestellt. Man sieht, daß er am Ende der Kette unserer Systemkomponenten steht und dennoch eine offene Schnittstelle für die Weiterverarbeitung der Daten bietet.

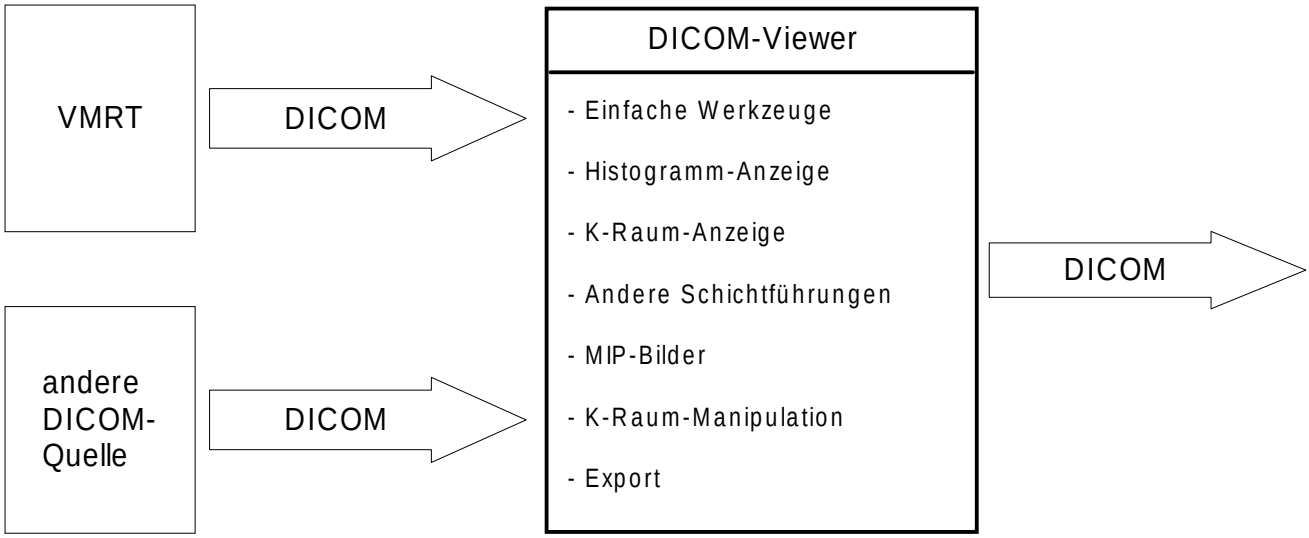

Abbildung 93: Integration des *DICOM-Viewers* in das Gesamtkonzept

Als grundlegende Werkzeuge soll der *DICOM-Viewer* dem Benutzer die Möglichkeit geben, Bildausschnitte mit einer Lupe zu vergrößern und Distanz-, Winkel- und Grauwertmessungen durchzuführen, um beispielsweise Pathologien vermessen und charakterisieren zu können. Desweiteren sollen einzelne Bilder invertiert, gespiegelt und gedreht werden können.

Zudem muß natürlich eine Fensterung der Bilder möglich sein. Das bedeutet nichts anderes, als das ein Bereich der maximal möglichen 4096 Graustufen (12 Bit) auf 256 Graustufen abgebildet wird (Abbildung 94). Die Reduktion auf 256 Graustufen macht durchaus Sinn, da das menschliche Auge nicht in der Lage ist, mehr unterschiedliche Grauwerte zu unterscheiden. Zudem bietet die Auswahl des abzubildenden Fensters die Möglichkeit, die Helligkeit und den Kontrast des Bildes zu variieren.

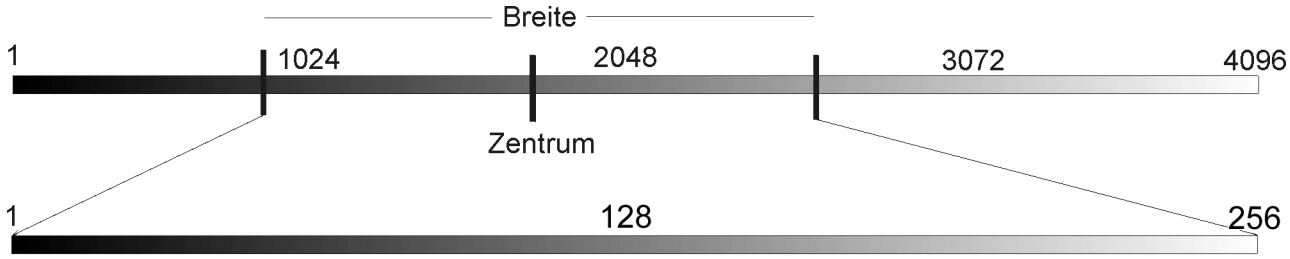

Abbildung 94: Fenstern eines 12-Bit-Bildes auf 8-Bit

Neben diesen einfachen Werkzeugen, sollen auch Werkzeuge zur Verfügung stehen, die eine dreidimensionale Rekonstruktion eines Datensatzes aus mehreren 2-dimensionalen Schichten ermöglichen. Dazu zählt zum einen das Erstellen einer Animation aus den verschiedenen Schichtdatensätzen. Neben der Animation in der Richtung, in der die Schichten vorliegen, soll jedoch auch eine Animation in Richtung der beiden dazu senkrecht stehenden Raumachsen stattfinden können.

Dazu ist es nötig, einzelne Schichten anderer Schnittführungen zu berechnen. Diese sollen letztendlich auch als neue Bilder in den Bildstapel eingefügt werden können. Dabei beschränken wir uns allerdings auf Einzelbilder, da ansonsten der Speicherplatzaufwand extrem hoch wäre. Bei den Projektionen

beschränken wir uns auf die beiden 90°-Projektionen, wie sie in Abbildung 95 und dargestellt sind. Die mit durchgezogener Linie gezeichneten Schichten stellen dabei die Originalschichten dar, die gestrichelt gezeichnete Schicht die neu berechnete. Eine Erweiterung auf andere Projektionsrichtungen erscheint für die Zukunft durchaus sinnvoll, geht jedoch über den Rahmen unserer Arbeit hinaus. In [PG305] wird dieses Thema behandelt.

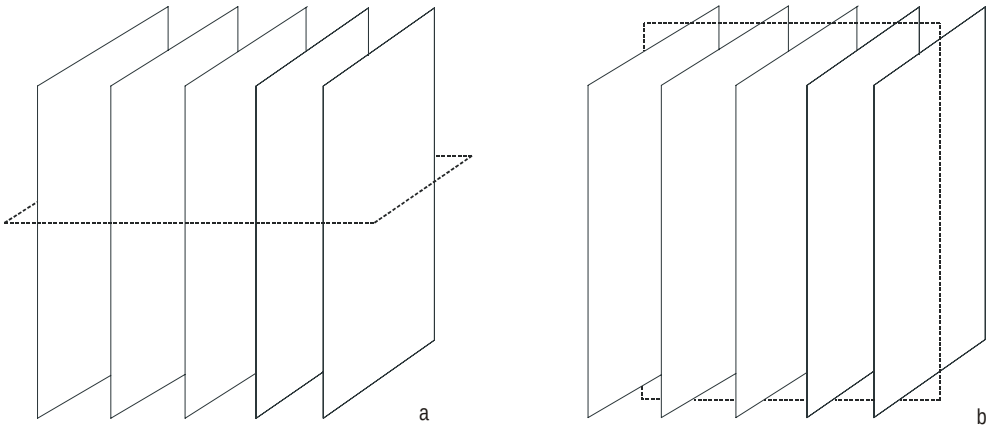

Abbildung 95: 90°-Projektion bei sagitaler Rohdatenorientierung.
a) Rekonstruktionsebene transversal b) Rekonstruktionsebene coronar

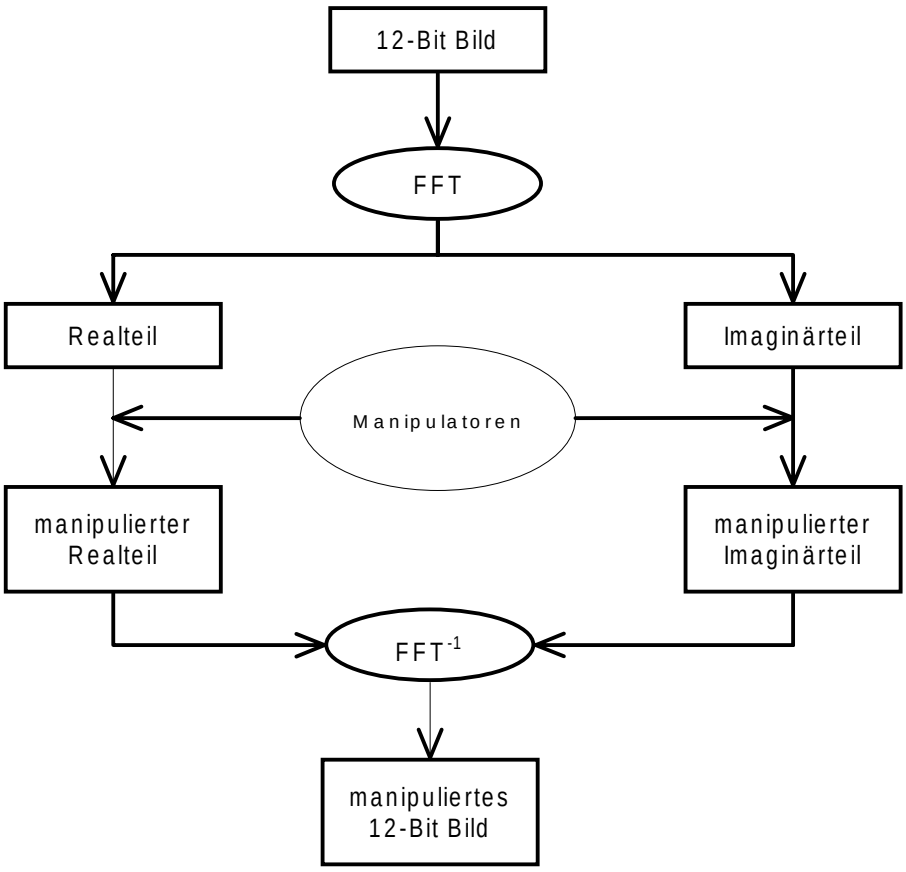

Abbildung 96: Konzept der k-Raum-Manipulation

Eine weitere nützliche Funktion, die allerdings über die Funktionalität des realen Tomographen hinausgeht ist die, sich den k-Raum eines Bildes anzeigen zu lassen (also das Magnitudenbild der Fouriertransformierten) und dort Manipulationen vornehmen zu können. Abbildung 96 zeigt das Konzept der k-Raum-Manipulation. Aus einem 12-Bit Bild werden der Real- und Imaginärteil der Fouriertransformierten berechnet. Auf diesen beiden Komponenten wirken sich die zur Verfügung

stehenden Manipulatoren aus. Nach der Manipulation wird die Rücktransformation berechnet und es steht somit das im k-Raum manipulierte Bild zur Verfügung.

Die Manipulationsmöglichkeiten sind allerdings zunächst beschränkt. Das Löschen von Zeilen bzw. Spalten in einem vorgegebenen Abstand bewirkt die Simulation einer Unterabtastung (engl. Undersampling) in Frequenz- bzw. Phasenkodierrichtung. Das Löschen eines inneren Rechtecks des k-Raums bewirkt den Verlust des Kontrasts in der Rücktransformation, wobei die Detailinformationen hervorgehoben werden. Und das Löschen der Ränder des k-Raums bewirkt genau das Umgekehrte: Die Detailinformationen gehen verloren und die Kontrastinformationen werden hervorgehoben. Diese Funktionalität soll dem Benutzer dazu dienen, ein Gefühl für die in dieser Arbeit beschriebenen k-Raum-Manipulationen (Kapitel 3.5) zu bekommen.

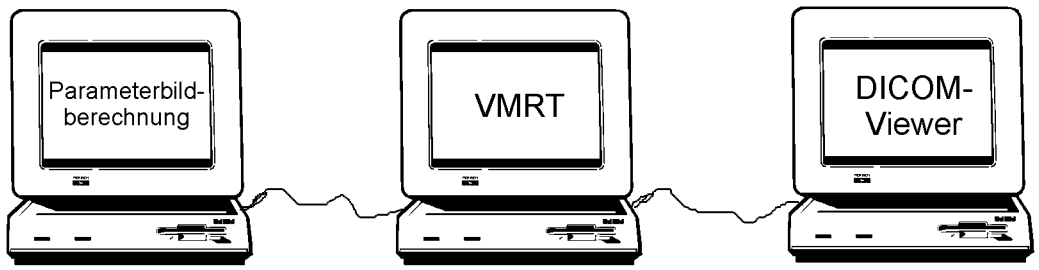

Abbildung 97: Gesamtsystem

Mit den drei nun vom Konzept her komplett beschriebenen Komponenten unseres Systems, steht dem Benutzer ein umfassendes System zur Simulation einer kompletten MRT-Anlage zur Verfügung, wenn man mal davon absieht, daß wir nicht die Spinanregungen der quasi unendlichen Anzahl an Atomen simulieren können. Die strikte Trennung der einzelnen Komponenten läßt sich am besten veranschaulichen, indem man vorsieht, jede Komponente auf einem separaten Rechner zu installieren, so wie die in Abbildung 97 dargestellt ist. Es spricht jedoch auch nichts dagegen, alle Komponenten auf einem Rechner zu installieren.

5 Umsetzung

Nach den Erläuterungen im letzten Kapitel zum Konzept der einzelnen Komponenten des Systems wollen wir nun auf die Umsetzung eingehen. Dabei werden wir nach einigen Erläuterungen zur Wahl der Programmiersprache JAVA die gleiche Reihenfolge beibehalten wie in der Beschreibung im Konzeptkapitel.

Nach einigen allgemeinen Informationen zur Implementierung wird zunächst die Umsetzung der Parameterbildberechnung beschrieben. Zur Realisierung dieser Berechnung wird auf das Werkzeug IMAGEJ zurückgegriffen. IMAGEJ ist eine freie Software, die unter anderem bereits einige grundlegende Zeichenoperationen auf 8, 16 und 32-Bit-Bildern bereitstellt. Das interessante an IMAGEJ ist, daß es sich sehr leicht um eigene Operatoren, die sogenannten *Plugins*, erweitern läßt. In diesem Kapitel werden daher erst die benutzten Schnittstellen von IMAGEJ erläutert, danach wird auf die Umsetzung der Berechnung eingegangen.

Das nächste Kapitel widmet sich der Umsetzung des virtuellen Tomographen (*Virtual MRT*). Anhand des Ablaufdiagramms aus Kapitel 4.3.2 wird die Umsetzung des realen Systems in die Programmiersprache JAVA erklärt.

Im letzten Kapitel wird die Umsetzung des Nachbearbeitungswerkzeugs beschrieben. Zur Implementierung konnten etliche Klassen aus dem *Virtual MRT* wiederverwendet werden.

5.1 Allgemeine Informationen zur Implementierung

5.1.1 Wahl der Programmiersprache

Der Hauptgrund für die Wahl der Programmiersprache JAVA lag im Wunsch von Dr. Hackländer, ein plattformunabhängiges System zu entwickeln. Der Benutzer kann das System *Virtual MRT* unabhängig davon einsetzen, welches Betriebssystem und welche Rechnerkonfiguration er benutzt. Allerdings muß ein genügend schneller Rechner und eine *Virtual Machine*[19] für das benutzte Betriebssystem vorhanden sein.

JAVA erreicht die Plattformunabhängigkeit durch eine Zwischenstufe, den Bytecode. Dieser ist plattformunabhängig und wird von der *Virtual Machine* interpretiert. Diese *Virtual Machines* sind für alle gängigen Plattformen verfügbar, dadurch funktioniert der virtuelle Tomograph automatisch auf allen diesen Plattformen.

JAVA bietet aber noch viele weitere Vorteile einer modernen objektorientierten Programmiersprache. Durch die Objektorientiertheit können wiederverwendbare Module entwickelt werden. Im Vergleich zu C++ sind einige Eigenschaften nicht vorhanden, die dort oft nicht korrekt oder unverständlich verwendet werden. Gerade bei größeren Projekten kann die falsche Verwendung dieser Konstrukte in C++ schnell zu unübersichtlichem Quelltext führen. JAVA verwendet z.B. keine Zeigerarithmetik, sondern nur Referenzen auf Objekte. Durch die starke Typung und den Mechanismus der *Exceptions* wird eine hohe Robustheit erreicht.

Die Ausführungsgeschwindigkeit ist wegen des interpretierten Bytecodes zwar schlechter als für C++-Programme, aber durch JIT-Compiler[20] oder die Option, Maschinencode zu erzeugen, ist der Unterschied zwischen C++ und JAVA in der Ausführungsgeschwindigkeit kaum wahrnehmbar. Für spätere Erweiterungen ist auch die Ausrichtung von JAVA auf die Ausführbarkeit in Netzwerken

[19] Die *Virtual Machine* übersetzt den vom JAVA-Compiler erzeugten Bytecode in Maschinenbefehle, die vom jeweiligen System verstanden wird. Das Programm funktioniert damit auf jedem System, für das eine *Virtual Machine* verfügbar ist.
[20] JIT: Just in Time Compiler. Mit einem JIT werden die Befehle vor der ersten Ausführung in Maschinensprache übersetzt. Da der Bytecode dann nicht bei jedem Durchlauf neu interpretiert wird, ist die Ausführung deutlich schneller.

interessant. Damit wäre es z.B. möglich, die Rohdaten direkt über das Internet in das Programm einzuladen.

Sehr wichtig ist für das Projekt die Möglichkeit, über die *reflection*-Bibliothek dynamisch Klassen zu instantiieren. Dies wird zur Implementierung der Pulssequenzen stark genutzt. *Virtual MRT* soll um weitere Pulssequenzen erweiterbar sein, daher sind die verfügbaren Berechnungsklassen erst zur Laufzeit bekannt und werden über eine externe *Property*-Datei definiert. Mehr Informationen über die Realisierung der Pulssequenzen und zur Implementierung weiterer Pulssequenzen befinden sich in Kapitel 5.5.1.

5.1.2 Arbeitsmittel und Arbeitsumgebung

Zur Umsetzung des Konzepts in ein Softwareprodukt haben wir uns für die Programmiersprache JAVA entschieden (siehe dazu auch Kapitel 5.1). Als Entwicklungswerkzeug für unsere Programme nutzten wir anfangs den JBUILDER 2.01 von INPRISE (BORLAND). Dies hatte gegenüber dem JAVA Developer Kit (JDK) von SUN den Vorteil, daß ein Quelltexteditor mit Syntaxhervorhebung und ein Werkzeug zur Gestaltung der Benutzeroberflächen zur Verfügung stand. Durch ein auf der Homepage von INPRISE zur Verfügung stehendes Update war es uns möglich, dennoch den JDK 1.2 Compiler zu benutzen und auch die original *Swing*-Klassen von SUN zu verwenden. Dies hatte wiederum den Vorteil, das die *Swing*-Bibliothek von INPRISE nicht mit unserem Produkt ausgeliefert werden muß, sondern das System auf jeder Plattform mit dem JDK 1.2 von SUN lauffähig ist. Zum Ende der Entwicklung stiegen wir ohne Probleme auf die Version 3 des JBUILDERS um. Diese Version ist laut Herstellerangaben 100%ig zum JDK 1.2 von SUN kompatibel.

Einige Experimente mit dem JDK 1.1.8 von IBM führten zu Fehlern in, Zusammenhang mit der Fouriertransformation. Wir raten daher dringends von dessen Benutzung ab, obwohl sich damit eine enorme Geschwindigkeitssteigerung erreichen läßt.

Die schriftliche Arbeit, die Sie nun in der Hand halten, wurde mit Hilfe der Textverarbeitung WORD97 von MICROSOFT erstellt. Dazu wurden die beiden Patches SR-1 und SR-2 installiert. Auf der dieser Arbeit beiliegenden CD steht die Ausarbeitung als PDF-Datei zur Verfügung. Diese wurde über den Umweg einer Postscript-Datei mit Hilfe des Programms Exchange 3.01 von Adobe erstellt.

Die Quelltextkommentierung (siehe auch Kapitel 5.1.3) erfolgte so, daß mit Hilfe des im JDK 1.2 von SUN enthaltenen Werkzeugs JAVADOC eine HTML-Quelltextbeschreibung erstellt werden konnte. Diese befindet sich ebenfalls auf der beiliegenden CD im Verzeichnis *JavaDoc*.

5.1.3 Quelltextkonventionen und Quelltextkommentierung

Um den Quelltext gut lesbar und übersichtlich zu gestalten, haben wir einige Konventionen für die Namensgebung, Groß- und Kleinschreibung und Kommentierung festgelegt.

Namenskonvention und Formatierung

Bezeichner werden in Englisch benannt, alle Kommentare werden in Deutsch verfaßt. Einrückungen werden jeweils im Abstand von 2 Leerzeichen vorgenommen. Nach jedem Konstruktor und jeder Methode folgt mindestens eine Leerzeile. Umlaute und Sonderzeichen werden im Quelltext grundsätzlich nicht verwendet[21].

Klassen

Klassennamen beginnen immer mit einem Großbuchstaben. Ist der Klassenname aus mehreren Wörtern zusammengesetzt, beginnt jedes dieser Wörter mit einem Großbuchstaben. Nach der

[21] Da die Klassenbeschreibungen im Anhang direkt aus dem Quelltext generiert wurden, enthalten auch diese keine Umlaute und Sonderzeichen.

Klassendefinition erfolgt die Definition der Klassen- und Instanzvariablen. Danach werden die Konstruktoren definiert, dann folgen die Methoden. Im folgenden ist ein Beispiel für eine Klassendefinition dargestellt.

```
class MyClass extends MyOtherClass {
  // Klassen- und Instanzvariablen
  Object variable1;

  // Konstruktoren
  MyClass() {}

  // Methoden
  public void method1()
  {
    ..
  }
}
```

Methoden

Methodennamen beginnen mit einem Kleinbuchstaben. Alle folgenden Wörter im Methodennamen beginnen mit einem Großbuchstaben. Methoden, die Zugriff auf eine Klassen- oder Instanzvariable ermöglichen, beginnen mit *get/set*.

Klassen- und Instanzvariablen

Klassen- und Instanzvariablen beginnen mit einem Kleinbuchstaben. Alle folgenden Wörter in deren Namen beginnen mit einem Großbuchstaben. Konstanten (also als *final* definiert) werden komplett in Großbuchstaben geschrieben.

Kommentare

Besonders großen Wert haben wir auf die Kommentierung unseres Quelltextes gelegt. Jede Klasse, jeder Konstruktor, jede Methode und jede Klassen- und Instanzvariable ist so kommentiert, daß mit Hilfe des im JDK 1.2 von SUN enthaltenen Werkzeugs JAVADOC automatisch ein HTML-Dokument generiert werden kann, welches alle wichtigen Implementierungsinformationen enthält. Jeder dieser Kommentare beginnt mit /** und endet mit */. Mehrzeilige Kommentare werden wie im folgenden dargestellt realisiert.

```
/**
 * Dieses Instanzvariable ist eigentlich gar nicht wichtig.
 * Es kommt in diesem Fall viel mehr auf den Kommentar an.
 */
int variable1;
```

Bei Konstruktoren und Methoden ist neben der Funktionsbeschreibung auch ein Kommentar für jeden Parameter und den Rückgabewert angegeben. Das sieht dann in etwa folgendermaßen aus:

```
/**
 * Diese Methode hat keine besondere Funktionaliaet. Sie nimmt zwar
 * 2 Parameter entgegen und liefert einen davon zurück, manipuliert
 * diesen jedoch nicht.
 * @param a Der erste Parameter.
 * @param b Der zweite Parameter.
 * @return Der Rueckgabewert.
```

```
 */
public int myMethod(int a, int b)
{
  return a;
}
```

Neben den Kommentare, die von JAVADOC ausgewertet werden, verwenden wir weitere Kommentare, die komplizierte Stellen im Quelltext erläutern sollen. Diese werden ausschließlich durch // eingeleitet und gelten bis zum Ende der Zeile. Um mehrzeilige Kommentare zu erzeugen, muß // jeder dieser Zeilen vorangestellt werden, wie das folgende Beispiel zeigt.

```
...
// Hier wird die Geschwindigkeit mit der Zeit multipliziert,
// um so die zurueckgelegte Strecke zu erhalten.
int distance = speed * time;
...
```

5.2 Berechnung der Parameterbilder mit IMAGEJ

Die Realisierung der Parameterbildberechnung erfolgte wie schon erwähnt mit IMAGEJ. IMAGEJ kann leicht um eigene Operatoren erweitert werden, die dann sämtliche Klassen und Operatoren von IMAGEJ benutzen können. In IMAGEJ existieren bereits Import- und Exportfunktionen für gängige Grafikdateiformate wie *.jpg*, *.gif*, und *.tif*. Insbesondere war schon eine Importfunktion für DICOM-Daten von Dr. Hackländer implementiert worden. Da diese zunächst auf einer fremden JAVA-Bibliothek basierte, wurde sie von uns später durch eine Minimalimplementierung des DICOM-Standards ersetzt [DICOM98]. Auf diese Bibliothek wollen wir aber später im Rahmen der Umsetzung des virtuellen Tomographen eingehen, da sie primär für diesen entwickelt wurde und dort zum Import der Parameterbilder und zum Export der in der Simulation errechneten Bilder eingesetzt wird.

5.2.1 *ImagePlus* und *ImageStack*

Die Klasse *ImagePlus* ist eine erweiterte Klasse für Bilder, die neben Bildern mit 8-Bit Pixeldaten auch Bilder mit 16-Bit vorzeichenbehafteten Pixeldaten, 32-Bit Fließkomma und 32-Bit RGB unterstützt. *ImageStack* ermöglicht den Zugriff auf einen ganzen Stapel von Bildern. Dieser Stapel kann dynamisch erweitert werden.

Für die Parameterbildberechnung werden lediglich 16-Bit vorzeichenbehaftete Bilder benötigt. Auch der Importfilter für DICOM-Dateien erzeugt stets eine Instanz von *ImageStack*, in der die eigentlichen Pixeldaten gespeichert werden.

ImagePlus bietet die Möglichkeit, Attribute als Eigenschaften (in einer Instanz der Klasse *Properties*) zu speichern. *ImagePlus* verwendet dazu die Klasse *Properties*, die Attribute in einer *Hashtable* speichert. Auch der DICOM-Importfilter benutzt diese Tabelle, um dort die DICOM-Informationen abzulegen. Die Klasse zur Parameterbildberechnung greift daher über

```
Properties prop = imp.getProperties();
String[] sl = null;
if (prop != null)
{
  sl = (String[]) prop.get("DICOM");
} else {
  // Keine Dicom-Informationen gefunden, Berechnung abbrechen.
  return;
}
```

auf die DICOM-Informationen zu. Die DICOM-Daten liegen jetzt als Feld vom Typ *String* vor und können auf bestimmte Einträge durchsucht werden.

In *ImageStack* ist ein Attribut *stack* implementiert, das die Pixeldaten als Feld von Objekten von Typ *Object* enthält. Die allgemeine Definition als *Object[]* ermöglicht die Speicherung der obengenannten verschiedenen Bildtypen. Der Zugriff auf die Pixeldaten wird in der Parameterberechnung über

```
imp = WindowManager.getCurrentImage();
is  = imp.getStack();
```

erreicht, auf die einzelnen Schichten im Stapel kann dann über die Methode *getPixels(i)* in der Klasse *ImageStack* zugegriffen werden.

```
private short[][] fillpixdata(int start, int end)
{
  short pix[][] = new short[end-start+1][];
  for (int i=start; i<=end; i++)
  {
    pix[i-start] = (short[]) is.getPixels(i);
  }
  return pix;
}
```

Dabei ist *i* die Nummer der Schicht. Mit *start* und *end* werden die Schichten definiert, die für die Berechnung verwendet werden. Dadurch erspart man sich einige Indexrechnungen, da zuvor über die obige Methode die Vorgaben für T_1- und T_2-Berechnung gefüllt werden.

Am Ende der Berechnung wird noch die Methode *addSlice()* der Klasse *ImageStack* aufgerufen, um die berechneten Parameterbilder in einen *ImageStack* einzufügen. Folgende Programmzeile fügt das Ergebnis der T_1-Berechnung in den *ImageStack* ein:

```
stack.addSlice("T1-Bild", (Object) result[0]);
```

Dieser *ImageStack* wird dann wieder mit dem DICOM-Export-Filter gespeichert.

5.2.2 *Plugin* und *PluginFilter*

Die Schnittstellen *Plugin* und *PluginFilter* bieten die Möglichkeit, das Programm IMAGEJ um eigene Operatoren zu erweitern.

Um die Schnittstelle *Plugin* zu implementieren, muß lediglich die Methode *run* vorhanden sein. Bei unserer Klasse zur Parameterbildberechnung sieht das so aus:

```
public class RawDataCalculator implements PlugIn
{
  ...
  public void run (String arg)
  ...
}
```

Das ist die Methode, die von IMAGEJ aufgerufen wird, wenn der Menüeintrag ausgewählt wird. Die Eintragung in die Menüstruktur erfolgt über die Datei *ij.properties*. Hier sind unter anderem die *Plugins* definiert:

```
plug-in11="DICOMexport",DICOM_export
plug-in12="DICOMimport",DICOM_import
plug-in13="Rohdatenmatrizen berechnen",RawDataCalculator
plug-in14="Stack retuschieren",stackpainter.StackPainter
```

Mit diesen Zeilen werden DICOM-Export und –Import, Parameterbildberechnung und Nachbearbeitungswerkzeug als *Plugin* konfiguriert. Optional können auch Parameter angegeben werden, die IMAGEJ der Methode *run* in *arg* übergibt. In obigem Fall ist *arg* leer.

Für die Schnittstelle *PluginFilter* muß noch eine weitere Methode namens *setup* implementiert werden. Diese wird einmal aufgerufen, danach wird *run* für jede Schicht eines Bilderstapels

aufgerufen. Die Parameterbildberechnung muß auf alle Schichten **gleichzeitig** zugreifen können, daher wurde sie als *Plugin* implementiert.

5.2.3 Berechnung der Parameterbilder

Die Parameterberechnung ist als *Plugin* für IMAGEJ implementiert. Die dazu nötigen Klassen wurden im JAVA-Paket[22] *rawdatacalculator* zusammengefaßt. Die Hauptklasse zur Berechnung ist *RawDataCalculator*. In der Klasse *LeastSquare* findet die eigentliche Berechnung der physikalischen Parameter mit der Methode der kleinsten Quadrate statt. Die Klassen *ImportDialog* und *ExportDialog* implementieren zwei einfache Dialoge. *ImportDialog* ermöglicht die Einstellung der Bildnummern, die zur Berechnung der T_1 bzw. der T_2 und PD-Werte benutzt werden. Außerdem müssen in diesem Dialog Einstellungen für die minimale Signalintensität und die Standardabweichung gemacht werden. Bei Berechnungen mit sehr kleinen Intensitätswerten oder bei Meßreihen mit fast konstanter Signalintensität kämen hier ohne diese Filter unsinnige Werte heraus. *ExportDialog* fragt vom Benutzer einen Namen für die Dateien der Rohdatenmatrizen ab.

Im *Plugin* wird zunächst zum Einlesen der Messung der DICOM-Importfilter aufgerufen. Nach dem Einlesen liegen die Daten als *ImageStack* vor, und die Daten werden wie in Kapitel 5.2.1 beschrieben ausgelesen. Über die Methoden *calculateT1* und *calculateT2* wird dann die eigentliche Berechnung durchgeführt. In *calculateT1* und *calculateT2* sind Schleifen über alle Bildpunkte der Meßreihe implementiert, die die Daten der Messung zur Berechnung aufbereiten. Die Meßreihe liegt als Bildserie vor, aber für die Berechnung werden die Meßpunkte für jeden Bildpunkt für die gewählten Schichten hintereinander benötigt. Für jeden Bildpunkt wird ein zweidimensionales Feld mit dem variablen Parameter als *x*-Wert und dem gemessenen Wert als *y*-Wert gefüllt. Bei der T_1-Messung wurde bei der Messung T_R variiert, bei der T_2-Messung T_E. Dafür sind folgende Programmzeilen nötig (hier für T_2):

```
for (int x=0; x < 256; x++)
{
  for (int y = 0; y < 256; y++)
  {
    int coord = y*256+x;
    ...
    samples[slice][0] = (iTE[slice]);
    samples[slice][1] = Math.log(pix[slice][coord]);
    weights[slice]    = (double) Math.log(pix[slice][coord]);
    ...
  }
}
```

In *samples* werden die Meßpunkte gesetzt, in *weights* werden die Gewichte für die Meßpunkte gesetzt. Als erster Versuch wurde hier der dekadische Logarithmus des Meßwertes benutzt. Die eigentliche Berechnung findet in einer Instanz der Klasse *LeastSquare* statt:

```
ls.setSample(samples);
ls.setWeights(weights);
ls.normalizeWeights();
ls.calculate();
```

Zuerst werden die Meßpunkte gesetzt, dann werden die Gewichte gesetzt und normalisiert. *Calculate* erstellt aus den Punkten und Gewichten die Matrix und löst diese über eine Gaußelimination. Die Lösungen werden dann mit *getSolution* ausgelesen. Aus dem Ergebnis werden dann T_2 und PD berechnet und in das Ergebnisfeld *result* eingetragen.

[22] JAVA-Pakete (engl. package) stellen eine Möglichkeit zur Verfügung, zusammengehörende oder in Beziehung stehende Klassen und Schnittstellen zu einer Einheit zusammenzufassen.

Das Ergebnisfeld *result* wird schließlich in einen *ImageStack* umgewandelt und nach Abfrage der Dateinamen über den *ExportDialog* als DICOM-Dateien gespeichert. Zusätzlich wird eine Datei mit der Endung *idx*[23] erzeugt. Diese Datei benötigt das Simulationsprogramm, um die Rohdatenmatrizen[24] einzulesen. Sie hat folgenden Aufbau:

```
T1FileName=Kontraststudie.T1
T2FileName=Kontraststudie.T2
PDFileName=Kontraststudie.PD
SZFileName=Kontraststudie.SZ
FlowFileName=Kontraststudie.FL
```

Für jede der 5 Rohdatenmatrizen ist ein Eintrag vorhanden. Die Zeichenkette vor dem Gleichheitszeichen ist für jede der 5 Rohdatenmatrizen fest definiert. Mit Hilfe dieser Zeichenkette findet im virtuellen Tomographen die korrekte Zuordnung der einzelnen Rofdatenmatrizen statt. Nach dem Gleichheitszeichen folgt der Dateiname, der die entsprechende Rohdatenmatrix enthält. Der Dateiname kann prinzipiell frei gewählt werden, ist jedoch initial durch den Namen der *idx*-Datei und eine passende Endung für die jeweilige Rohdatenmatrix benannt.

5.2.4 Nachbearbeitung der Parameterbilder

Nach dem Berechnen der Rohdatenmatrizen können diese noch mit einem einfachen Zeichenprogramm bearbeitet werden. Auch das Zeichenprogramm wurde als Plugin für IMAGEJ im Paket *stackpainter* realisiert. Dieses Paket enthält die Klassen *StackPainter*, *StackPainterFrame*, *StackPainterCanvas*, *GlassPane*, *DrawTypes* und *RawDataValue*. In der Methode *run* der Klasse *StackPainter* wird lediglich eine Instanz von *StackPainterFrame* erzeugt und das Fenster sichtbar gemacht. Die eigentliche Interaktion findet in den Klassen *StackPainterFrame*, *StackPainterCanvas* und *GlassPane* statt.

Die Oberfläche ist ähnlich wie das Simulationsprogramm *Virtual MRT* aufgebaut und zeigt links vier Parameterbilder (T_1, T_2, PD und wahlweise Spinfluß oder Suszeptibilität), rechts ist ein Kontrollbereich. In dem Kontrollbereich kann das gewünschte Zeichenwerkzeug gewählt werden (Linie, Kreis, Rechteck, Freihand). Über Schieberegler und die Textfelder können die Werte eingestellt werden, mit denen in die einzelnen Parameterbilder gezeichnet wird. Außerdem können die Bilder selektiert werden, in die überhaupt gezeichnet wird. So kann in ein oder mehrere Parameterbilder gezeichnet werden. Das entscheidende an diesem Zeichenwerkzeug ist, das in mehrere Ebenen der 16-Bit-Parameterwerte **gleichzeitig** gezeichnet werden kann. Diese Interaktion wird in *StackPainterFrame* implementiert.

Das Einzeichnen in die Bilder und das Anzeigen erfolgt in den Klassen *StackPainterCanvas* und *GlassPane*. Die *GlassPane* wird in *Swing* als oberste Schicht in die z-Reihenfolge eingefügt [GEAR99]. Während der Benutzer die Zeichenoperationen durchführt, werden die Elemente in roter Farbe in die *GlassPane* eingezeichnet. Zum Beispiel wird beim Zeichnen einer Linie mit dem ersten Mausklick der Startpunkt bestimmt, durch Ziehen der Maus bei gedrückter Taste wird der Endpunkt festgelegt. Während der Benutzer die Maus zieht, kann er sehen, wie die gezeichnete Linie beim Loslassen der Taste eingezeichnet würde.

Das Einzeichnen der eigentlichen Grafikobjekte in alle gewählten Schichten erfolgt dann in der Klasse *StackPainterCanvas*. Da JAVA keine Operationen auf 16-Bit-Bildern anbietet, wird zunächst in ein 8-Bit-Bild gezeichnet und anschließend die Pixel in alle gewählten Bilder übertragen.

Als Quelltextbeispiel die Methode zum Zeichnen der Linie:

```
public void drawLine()
{
  Graphics gr = magicPlane.getGraphics();
```

[23] .idx steht für Index

[24] Die Begriffe Rohdatenmatrix und Parameterbild werden hier synonym verwendet.

```
      gr.setColor(Color.black);
      gr.fillRect(0,0,256,256);
      gr.setColor(Color.white);
      gr.drawLine(lastClickX, lastClickY, lastReleaseX, lastReleaseY);
      int start = (lastClickY*256+lastClickX-1);
      int end   = (lastReleaseY*256+lastReleaseX-1);
      paint2Images(0, 256*256-1);
    }
```

MagicPlane ist eine Klassenvariable, in der die Zeichenoperation ausgeführt werden. Sie ist als *AWTImage* definiert. In der Methode *paint2Images* wird dann auf die 16-Bit-Daten zugegriffen:

```
    public void paint2Images(int start, int end)
      ...
      for (int i=start; i < end; i++)
      {
        if ((pixels[i] & 0xff) > 1)
        {
          for (int x = 0; x < 5; x++)
          {
            if (rw.DrawImages[x])
            {
              images[x][i] = (short) rw.RawValues[x];
            }
          }
        }
      }
      convertImages();
      repaint();
    }
```

Über *convertImages()* werden die Bilder gefenstert und über *repaint()* neu dargestellt.

Die verbleibenden Klassen *DrawTypes* und *RawDataValue* sind lediglich Hilfsklassen. Während *DrawTypes* dazu dient, die einzelnen Grafikoperationen als statische Variable zu definieren[25] dient eine Instanz der Klasse *RawDataValue* zur Übergabe der Zeichenparameter.

5.3 Der virtuelle Tomograph

An Hand der Ablaufschemata des virtuellen Tomographen kann das Projekt für die Umsetzung gut in JAVA-Pakete aufgeteilt werden, die inhaltlich zusammengehören. Das Projekt *Virtual MRT* teilt sich in folgende Pakete auf:

- *artefacts*: Klassen zur Simulation von Artefakten und Störeinflüssen auf die Messung. Ebenso wie für die Pulssequenzen existiert für jede Artefaktsimulation eine Interaktionsklasse und eine Berechnungsklasse.
- *dcm*: Klassen zum Import und Export von DICOM-Dateien. Allerdings implementiert diese Bibliothek nur eine minimale Unterstützung für DICOM.
- *fft*: Berechnung, Ansicht und Manipulation des k-Raumes.
- *jigl*: „Java Imaging and Graphics Library“. Freie Bibliothek der Brigham Young University, die für die Berechnung der Fouriertransformation und der inversen Fouriertransformation benutzt wird.
- *sequences*: Klassen für die Eingabeinteraktion und Berechnung der Pulssequenzen. Für jede Pulssequenz gibt es einer Interaktions- und eine Berechnungsklasse.

[25] Eine Abfrage wie `if (drawType == DrawTypes.DT_LINE)` ist sicherlich leichter verständlich als die direkte Benutzung der Ganzzahlenwerte, mit denen die Zeichentypen definiert sind.

- *tools*: Einige Hilfsfunktionen, wie z.B. *StringToInteger*, häufig verwendete zusammengesetzte Bedienelemente etc.
- *vmrt*: Hier wird die komplette Benutzerinteraktion abgewickelt: Wahl der Pulssequenz, Eingabe der Parameter, Starten der Berechnung, Fenstern und Anzeige der Bilder etc.

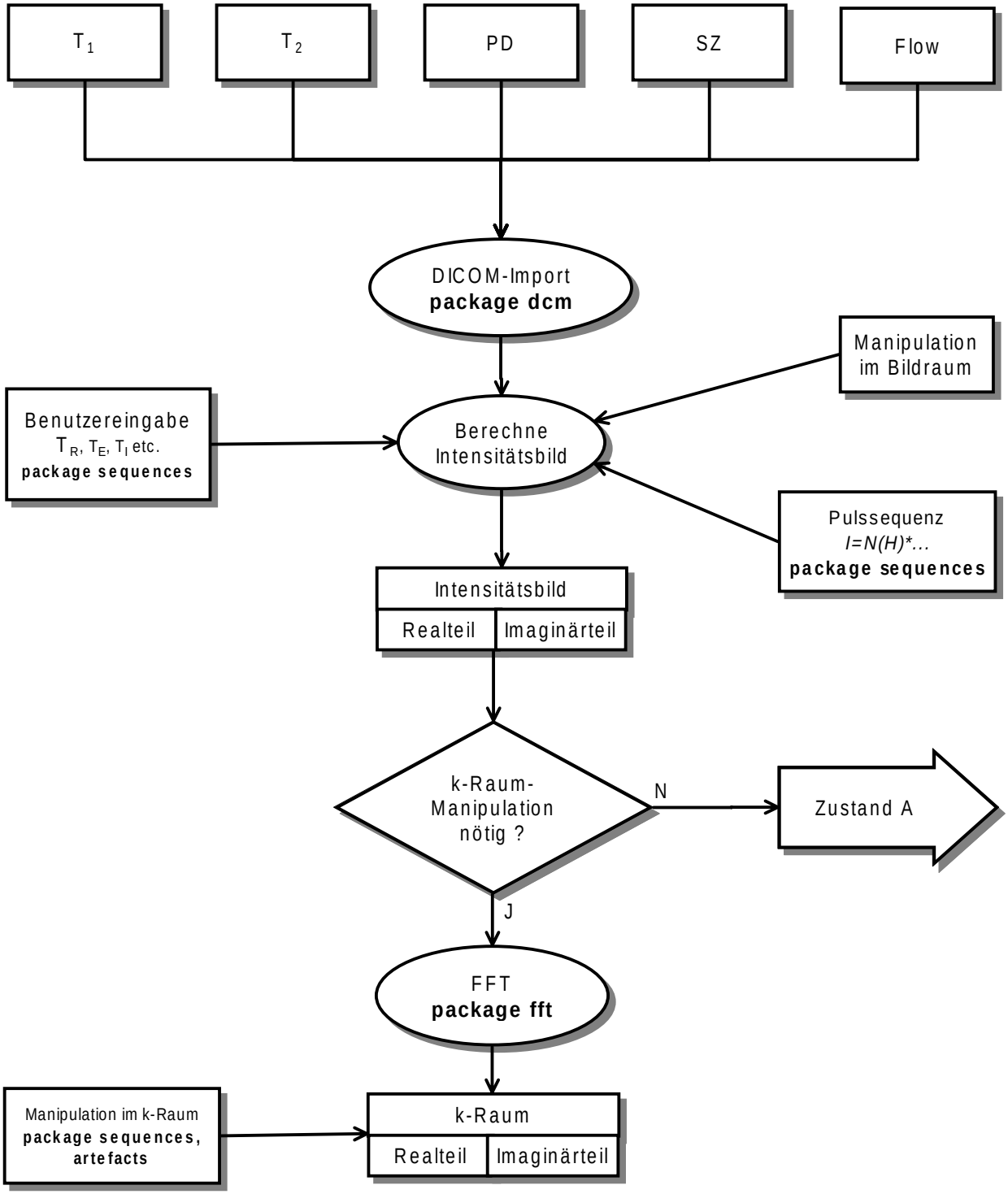

Abbildung 98: Zuordnung der JAVA-Pakete zu den einzelnen Schritten der Simulation

Abbildung 98 zeigt nochmals den Ablauf der Simulation bis zum Füllen des k-Raumes. Hier sind den einzelnen Komponenten der des Graphen die Pakete zugeordnet, die die entsprechende Aufgabe übernehmen. Im ersten Schritt werden die Parameterbilder eingelesen. Der DICOM-Import wird komplett im Paket *dcm* durchgeführt. Im Paket selber sind noch einige Unterpakete definiert, wobei für den virtuellen Tomographen hauptsächlich das Paket *dcm.dicom* interessant ist. Die

Benutzerinteraktion zur Auswahl der Sequenz geschieht in *vmrt*, die Eingaben und die Berechnung werden dann von den jeweiligen Klassen im Paket *sequences* durchgeführt. Aus dem berechneten Intensitätsbild wird dann über das *jigl*-Paket der fouriertransformierte k-Raum berechnet, *fft* ermöglicht die Anzeige des k-Raumes und des Phasenbildes. Manipulationen im k-Raum um Effekte zu simulieren sind entweder in die Sequenz eingebaut oder werden durch Klassen im Paket *artefacts* realisiert.

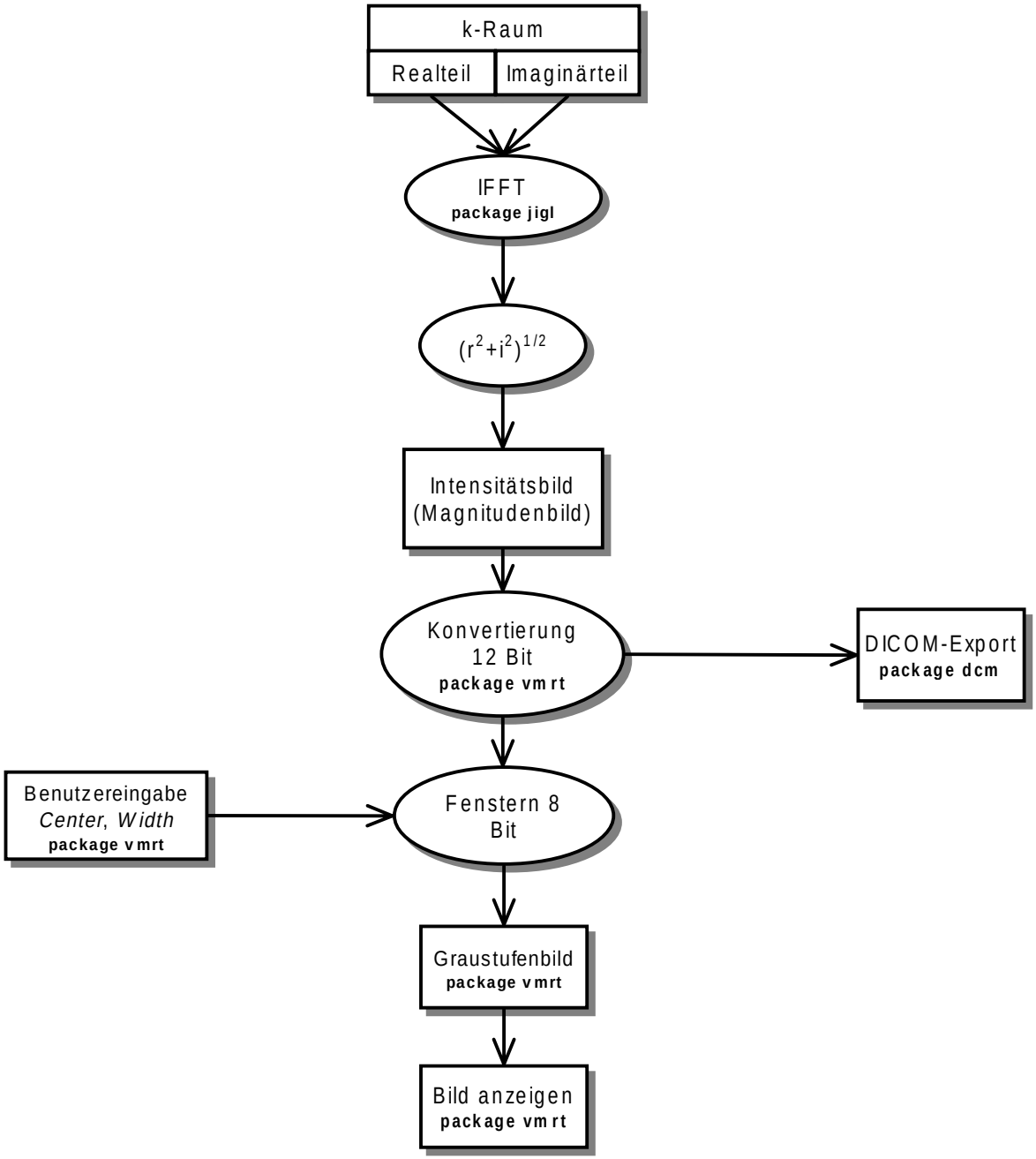

Abbildung 99: Rücktransformation: Zuordnung der JAVA-Pakete zu den einzelnen Schritten der Simulation.

Nach den k-Raum-Manipulationen wird durch eine inverse Fouriertransformation wieder mit dem Paket *jigl* ein Intensitätsbild berechnet (Abbildung 99). Die Konvertierung auf 12 Bit, Fensterung und Anzeige des Bildes werden über *vmrt* abgewickelt, der DICOM-Export erfolgt wieder über das Paket *dcm*. Da am *dcm*-Paket noch Änderungen durchgeführt werden, wurde zum Einsatz in *Virtual MRT* die Version auf den 20.06.1999 eingefroren.

In den folgenden Kapitel wollen wir die einzelnen Pakete näher beschreiben.

5.3.1 Das Paket *artefacts*

Das Paket *artefacts* implementiert die Simulation von Artefakten direkt im Anschluß an oder während der Berechnung eines neuen Bildes durch eine Pulssequenz. Die Klassen dieses Pakets stellen dafür sowohl die Bedienelemente und die Methoden zur Benutzerinteraktion, als auch die Berechnungsmethoden für die Artefakte zur Verfügung. Abbildung 100 zeigt das Klassendiagramm des Paketes *artefacts* inklusive der Beziehung zum Hauptpaket *vmrt* (Kapitel 5.3.6) und zum Paket *sequences* (Kapitel 5.3.4).

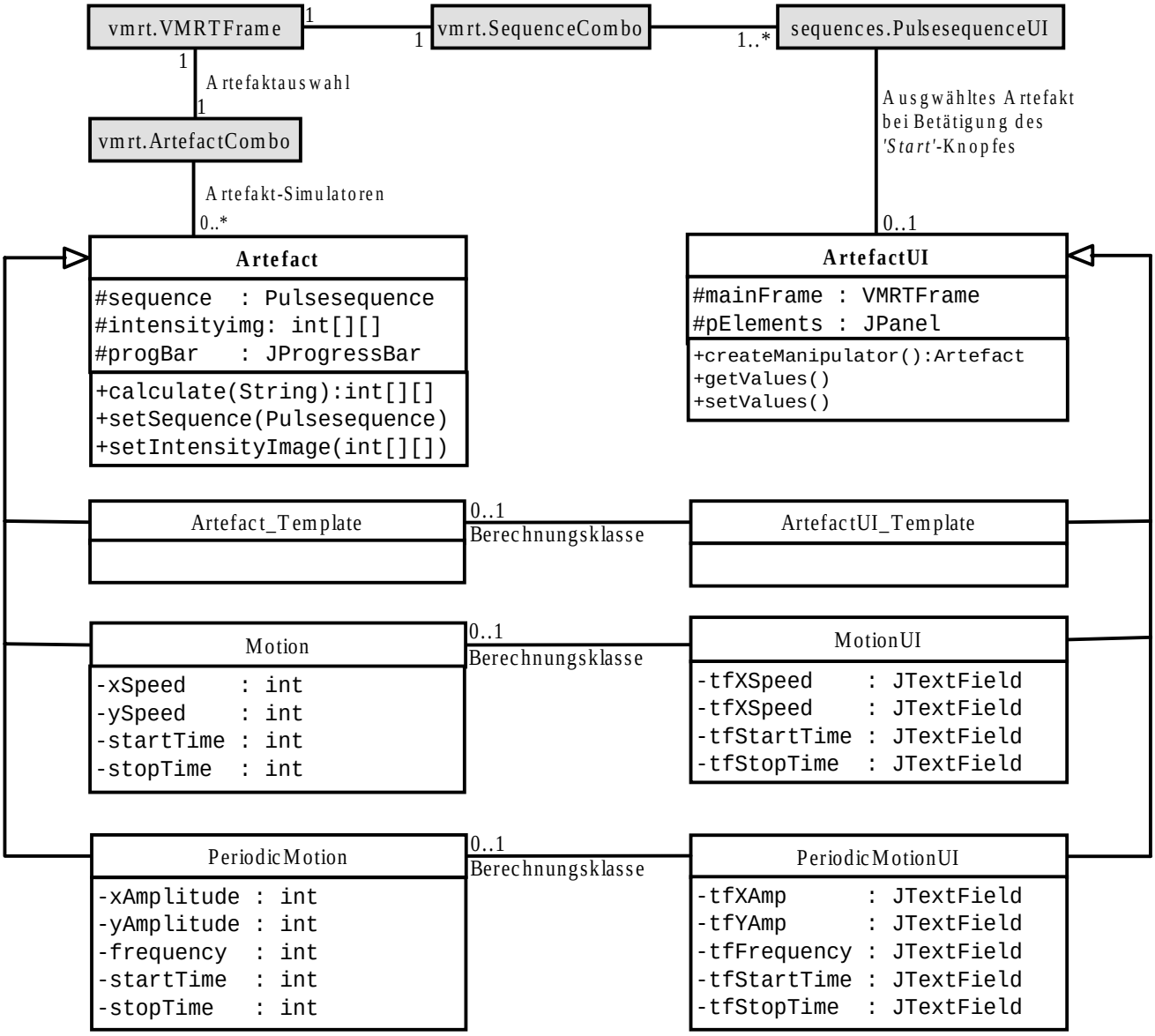

Abbildung 100: Klassendiagramm für das Paket artefacts

Dieses Klassendiagramm sowie die folgenden wurden nach dem UML-Standard erstellt (siehe dazu auch [OEST98]). Über den Standard hinaus haben wir festgelegt, daß Klassen, die nicht zum beschriebenen Paket gehören, grau hinterlegt werden.

Die Klasse *VMRTFrame* stellt das Hauptfenster des Meßwerkzeugs *Virtual MRT* dar und ist für die Benutzerinteraktion verantwortlich. *VMRTFrame* enthält genau eine Instanz der Klasse *ArtefactCombo*. Mittels dieser eine Auswahlbox repräsentierenden Klasse kann der Benutzer einen Artefakt-Simulator auswählen, der bei der Bildberechnung durch die ausgewählte Pulssequenz verwendet werden soll.

Nach dem Starten der Bildberechnung durch die Pulssequenz besorgt sich die Oberflächenklasse der Pulssequenz die Oberflächenklasse des aktuell ausgewählten Artefakt-Simulators. Durch Aufruf der Methode *createManipulator* der Oberklasse *ArtefactUI* wird eine Instanz der Berechnungsklasse des ausgewählten Artefakt-Simulators erzeugt und an die aufrufende Pulssequenz zurückgeliefert. Somit steht eine Referenz auf diese Berechnungsklasse während der gesamten Lebensdauer der Pulssequenzberechnungsklasse zur Verfügung. Dieser kann mittels der Methode *setIntensityImage* das berechnete Intensitätsbild übergeben werden. Zusätzlich kann durch Aufruf der Methode *setSequence* auch eine Referenz auf die aktuelle Pulssequenz übergeben werden. Über diese Referenz ist es der Berechnungsklasse des Artefakt-Simulators dann möglich, sequenzspezifische Parameter auszulesen.

Entscheidend ist die Rolle der Methode *calculate*, die in der Oberklasse *Artefact* bereits abstrakt definiert ist, also von jeder davon abgeleiteten Klasse implementiert werden muß. Der Methode wird eine Zeichenkette übergeben, die der Berechnungsklasse mitteilt, welche ihrer Methoden die richtige Rechenvorschrift für die aktuelle Pulssequenz enthält. Die Pulssequenzklasse kennt diesen Methodennamen, da der Artefakt-Simulator-Entwickler jeden einzelnen Artefakt-Simululator für jede einzelne Pulssequenz durch Angabe dieses Methodennamens in der *sequences_de_DE.properties* Datei freischalten muß. In der *calculate*-Methode muß also die übergebene Zeichenkette ausgewertet werden und die entsprechende Methode aufgerufen werden.

Innerhalb der dann aufgerufenen Methode kann das in der Instanzvariablen *intensityimg* zur Verfügung stehende Intensitätsbild kopiert und beliebig manipuliert werden. Wenn der Entwickler für die Berechnung der Manipulationen noch einige Sequenzparameter benötigt, kann er sich diese über die in der Instanzvariablen *sequence* gespeicherte Referenz auf die aufrufende Pulssequenz besorgen. Da diese Referenz allerdings vom Typ *Pulsesequence* ist, kennt sie zunächst auch nur die Methoden dieser abstrakten Oberklasse. Um auf die Methoden oder Instanzvariablen einer speziellen Pulssequenz zugreifen zu können, muß die *sequence*-Referenz zunächst einer Typumwandlung unterzogen werden. Mittels

```
sequence.getClass().getName()
```

kann der Name der Sequenzklasse erfragt werden und dann mittels Fallunterscheidung oder mit Hilfe des *reflection*-Pakets eine Typumwandlung stattfinden.

Das manipulierte Intensitätsbild muß von der speziellen Berechnungsmethode zunächst an die *calculate*-Methode weitergereicht und schließlich von dieser an die aufrufende Pulssequenz übergeben werden. Die Pulssequenz führt dann die restlichen Schritte aus, die zur Darstellung des neuen Bildes notwendig sind.

Durch die in der Oberklasse *ArtefactUI* abstrakt definierten Methoden *getValues* und *setValues* ist es möglich, die Einstellungen der Bedienelemente eines speziellen Artefakt-Simulators zu sichern und somit wieder herzustellen, wenn dieses Artefakt wieder ausgewählt wird. Die beiden Methoden sind abstrakt definiert, da sie beim Auswählen eines anderen Artefakt-Simulators automatisch von der *ArtefactCombo*-Klasse aufgerufen werden. Der Entwickler ist also gezwungen, diese beiden Methoden in seiner von *ArtefactUI* abgeleitete Klasse zu implementieren. In der Methode *setValues* müssen die Einstellungen der Bedienelemente des Artefakts in Instanzvariablen gesichert werden. Die Methode *getValues* hingegen muß die Werte dieser Instanzvariablen als Einstellungen für die Bedienelemente setzen.

Die beiden in Abbildung 100 dargestellten Klassen *Artefact_Template* und *ArtefactUI_Template* dienen als Vorlage zur Implementierung neuer Artefakt-Simulatoren. Sie enthalten das Grundgerüst der beiden zu implementierenden Klassen und enthalten an den entscheidenden Stellen Kommentare, was wie anzupassen ist. Eine genaue Beschreibung über des Hinzufügen eines neuen Artefakt-Simulators ist in Kapitel 5.5.2 ausführlich dargestellt.

5.3.2 Das Paket *dcm*

Um die Klassen dieses Paketes zu verstehen, ist es nötig, erst einmal einen kurzen Blick auf das DICOM-Format zu werfen. Die vollständige Beschreibung dieses Formates würde hier zu weit gehen, für weitergehende Informationen empfiehlt sich [DICOM98]. Wesentlich an dem DICOM-Format ist, daß es plattformunabhängig ist und mit speziellen Markierungen (engl. tags) arbeitet. Ein *tag* ist wie in Abbildung 101 oben dargestellt aufgebaut. Es wird über Gruppennummer und Elementnummer identifiziert, danach folgt die Länge in weiteren 4 Byte. In *Inhalt* ist die eigentliche Information gespeichert. Diese Reihenfolge ist die sogenannte implizite Transfersyntax. Bei der zweiten Transfersyntax, der expliziten Transfersyntax wird zusätzlich noch der Typ der Information übertragen, bei der ersten wird er durch Gruppen- und Elementnummer implizit definiert.

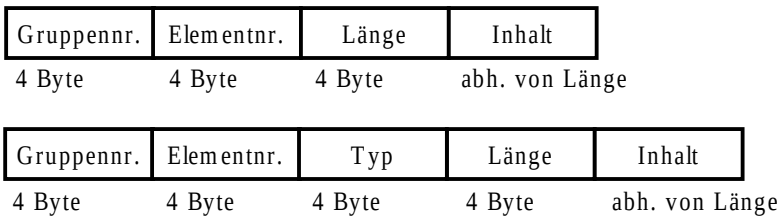

Abbildung 101: Implizite und explizite Transfersyntax

Um numerische Informationen systemunabhängig übertragen zu können, werden diese byteweise übertragen. Dabei werden Zahlen über zwei verschiedene Formate übertragen: *little endian und big endian*. Diese Zahlenformate legen die Übertragungsreihenfolge fest. Abbildung 102 zeigt beispielhaft die Übertragungsreihenfolge für eine 32-Bit-Zahl. Für das Format *little endian* wird das niederwertigste Byte zuerst übertragen, bei *big endian* zuerst das hochwertigste Byte. In der Version der Bibliothek, die hier verwendet wird, ist lediglich die implizite Transfersyntax mit *little endian* implementiert. Dies ist gleichzeitig der Standardfall, der laut DICOM-Spezifikation implementiert sein **muß**, andere **können** unterstützt werden.

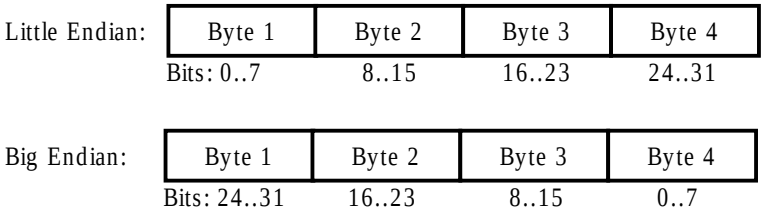

Abbildung 102: Übertragungsreihenfolge für Zahlenformate. Beispiel: 32-Bit Zahl

Abbildung 103 zeigt eine Übersicht über die wichtigsten Klassen in der Biblitohek. Zur besseren Übersichtlichkeit sind nur die wichtigsten Attribute und Methoden dargestellt. Nach dem Einlesen und Interpretieren wird ein DICOM-Objekt als Instanz der Klasse *DicomDataObject* repräsentiert. Das wichtigste Attribut ist die Referenz *grouptable* vom Typ *Hashtable*, in der alle Gruppeneinträge als Instanzen der Klasse *DcmElements* gespeichert sind. *DcmElements* ist eine Klasse, die *Hashtable* lediglich um ein Attribut *length* erweitert. In dieser Tabelle werden schließlich die DICOM-Tags als Instanz vom Typ *DcmValue* abgelegt.

Bevor die Informationen aber in dieser Struktur abgelegt werden können, müssen die Dateien zunächst auf *Stream*-Ebene eingelesen werden. Prinzipiell ist die Quelle dieses *Streams*[26] beliebig, es könnte z.B. auch eine URL[27] als Quelle angegeben werden. In der Implementierung die im *Virtual MRT* verwendet wird, ist als Quelle allerdings nur eine lokale Datei zulässig. Dieser Dateizugriff wird in *DcmFileBuffer* implementiert. In den Vektoren *inVector* und *outVector* werden über Instanzen der

[26] Abstrakter Ein/Ausgabetyp. Für Streams ist es unwesentlich, ob ihre Quelle z.B. auf der lokalen Festplatte, im Internet oder einfach im lokalen Speicher liegt.
[27] URL - Uniform Resource Locator, Mechanismus um eine Datei oder Webseite im Internet festzulegen.

Klasse *DcmIOBlockBuffer* die eigentlichen Rohdaten gespeichert, der Zugriff auf das Dateisystem erfolgt z.B. über die Methode *readBin*.

Das Interpretieren der Rohdaten (*DcmFileBuffer)* in ein *DicomDataObject* und umgekehrt erfolgt über zwei statische Methoden der Klasse *DcmInterpreter*. Hier werden die DICOM-Tags aus den Rohdaten gelesen und entsprechende Einträge in die Gruppen- und Elementtabellen vorgenommen.

Die Schnittstelle zu anderen Anwendungen (z.B. *Virtual MRT*) ist in der Klasse *DcmFileManager* implementiert. Hier wird das Laden und Speichern von DICOM-Dateien aus der Anwendung heraus aufgerufen. Die Methode *loadDcmDataObject* steuert dann den gesamten Ablauf vom Einlesen der Rohdaten aus der lokalen Datei bis zum Erzeugen eines *DicomDataObject* und dessen Rückgabe.

Zum Erkennen der Typen und zum einfacheren Zugriff auf die DICOM-Informationen wird in einer Instanz der Klasse *DcmDED* eine Tabelle erzeugt, die Objekte vom Typ *DcmDataDictionaryElement* enthält. Über das Attribute *type* kann für eine gegebenes DICOM-Tag (bestimmt durch Gruppen- und Elementnummer) der Typ bestimmt werden, was für die Implementierung der impliziten Transfersyntax wichtig ist.

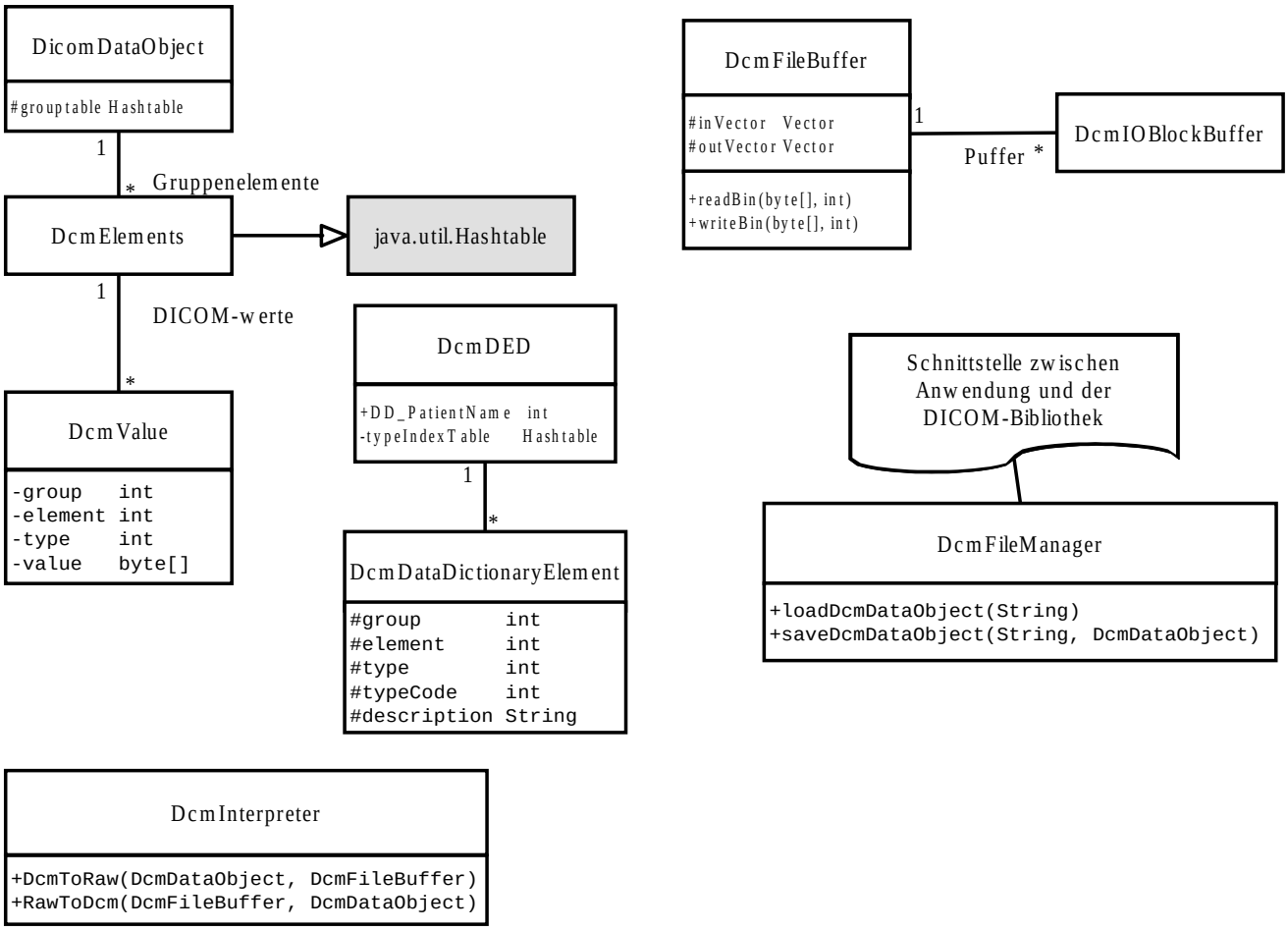

Abbildung 103: Übersicht über die wichtigsten Klassen des DICOM-Paketes

Desweiteren wird für jedes DICOM-Tag eine statische Klassenvariable in *DcmDED* definiert. Die DICOM-Tags haben sprechende Namen wie z.B. *DD_PatientName*, wie in Abbildung 103 exemplarisch dargestellt ist. In der Tabelle sind diese Konstanten unter *typeCode* abgelegt. *DicomDataObject* stellt Methoden zur Verfügung, die mit dieser Konstante eine DICOM-Information auslesen können. Dabei wird zunächst die Element- und Gruppennummer des gewünschten DICOM-Tags bestimmt und anschließend mit Hilfe dieser Nummern die eigentliche Information ausgelesen.

5.3.3 Die Pakete *fft* und *jigl*

Das Paket *fft* enthält die Klassen *kSpaceCanvas, kSpaceFrame, kSpaceManipulator, kSpaceManipulatorCanvas* und *FFTTools.* Die Klassen *kSpaceCanvas, kSpaceFrame* implementieren die Anzeige des k-Raumes mit Möglichkeit, Realteil, Imaginärteil, Phasenbild und Magnitudenbild anzuzeigen. *kSpaceManipulator* und *kSpaceManipulatorCanvas* implementieren einen Dialog, mit dessen Hilfe der Benutzer interaktiv Manipulationen im k-Raum vornehmen kann und deren Auswirkungen auf den Bildraum beobachten kann. Da diese Klassen für den *DICOM-Viewer* benötigt werden, sind sie detailliert in Kapitel 5.4 beschrieben. Die Klasse *FFTTools* stellt schließlich noch einige statische Hilfsmethoden im Zusammenhang mit der Fouriertransformation zur Verfügung.

Das Paket *jigl* ist wie gesagt eine freie Bibliothek der Brigham Young University. Die Bibliothek wird wie im folgenden Quelltextbeispiel verwendet, um eine schnelle Fouriertransformation durchzuführen:

```
// Konvertieren des 12-Bit Bildes nach float
float[][] myFloatImage = new float[size][size];
for (int x=0; x<size; x++) {
  for (int y=0; y<size; y++) {
    myFloatImage[x][y] = (float)(DcmImg.pixel16[(size*y)+x]);
  }
}

// Erzeugen eines RealGrayImage-Objektes, mit dem die FFT
// durchgefuehrt wird
jigl.image.RealGrayImage rgImg =
  new jigl.image.RealGrayImage(myFloatImage);

// Durchfuehren der FFT
jigl.image.ComplexImage cImg =
jigl.image.utils.FFT.doFFT(rgImg, true);
```

Vor dem Berechnen der FFT[28] wird das 12-Bit-Intensitätsbild, das in *DcmImg* gespeichert ist, in ein zweidimensionales Feld vom Typ *float* übertragen. Mit diesem Feld wird dann eine neue Instanz vom Typ *jigl.image.RealGrayImage* erzeugt und mit dem Aufruf *jigl.image.utils.FFT.doFFT(rgImg, true)* durchgeführt.

Der boolesche Wert als zweiter Parameter bestimmt, ob eine FFT oder eine inverse FFT durchgeführt wird: Ist der Parameter **true,** wird eine Fouriertransformation durchgeführt, ist er dagegen **false**, wird eine inverse Fouriertransformation durchgeführt. Das Fouriertransformierte Bild kann also mit

```
refftComImg = jigl.image.utils.FFT.doFFT(finalFFTImg,false)
```

wieder zurücktransformiert werden. Anschließend wird aus dem Ergebnis ein Magnitudbild berechnet:

```
for (int x=0; x<256; x++)
{
  for (int y=0; y<256; y++)
  {
    IntensityMatrix[x][y]=(int)
      (Math.sqrt(Math.pow(refftComImg.getReal(x,y),2) +
      Math.pow(refftComImg.getImag(x,y),2) ) );
  } // for x
} // for y
```

[28] FFT: „Fast Fourier Transform“ - Diskretes Verfahren zur schnelle Berechnung einer Fouriertransformation. Zur Berechnung muß die Größe des Bildes eine Potenz von zwei sein. Für unsere Anwendung wird die Fouriertransformation für ein 256x256-Bild berechnet.

5.3.4 Das Paket *sequences*

Dieses Paket stellt zum einen die Benutzerinteraktion für die Eingabe der Sequenzparameter zur Verfügung, zum anderen erfolgt hier die Berechnung der Intensitätsbilder mit den Formeln für die jeweilige Sequenz, die wir in Kapitel 3.3 vorgestellt haben.

Abbildung 104 zeigt das Klassendiagramm des Paketes *sequences* und die Beziehung zum Hauptpaket *vmrt*. *VMRTFrame* als Hauptinteraktionsklasse enthält eine Instanz der Klasse *SequenceCombo*, die für die Auswahl der Pulssequenz benutzt wird. *SequenceCombo* enthält einen Vektor von Referenzen auf die UI-Klassen aller verfügbaren Pulssequenzen.

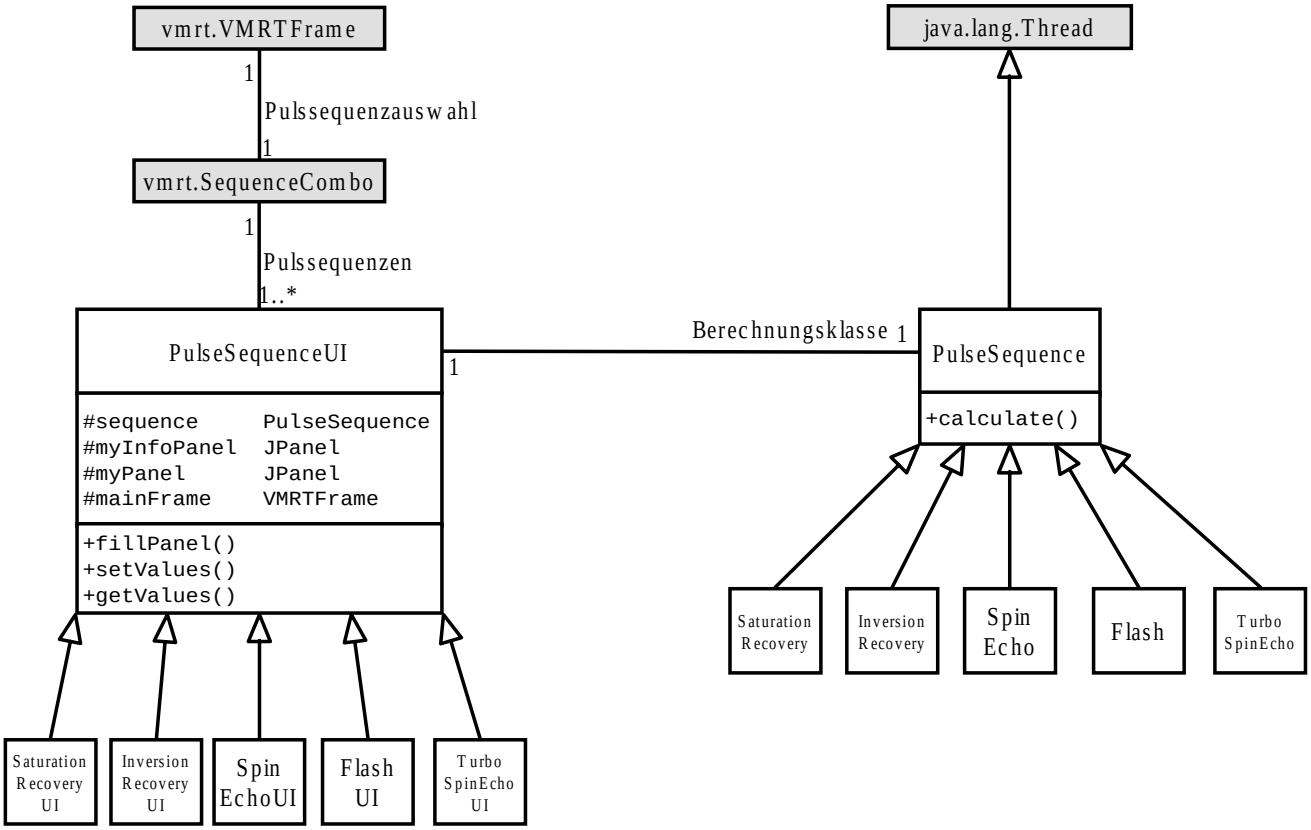

Abbildung 104: Klassendiagramm für das Paket sequences

Jede Pulssequenz wird durch eine UI-Klasse[29] und eine Berechnungsklasse implementiert. In der Oberklasse der Benutzerinteraktionen für Pulssequenzen *PulsesequenceUI* sind alle Eigenschaften der Interaktion implementiert, die nicht sequenzspezifisch sind. Den Instanzen der Klasse *PulsesequenceUI* werden zwei Bereiche (Instanzen der Klasse *JPanel*) übergeben, in denen die *PulsesequenceUI*-Klasse die Elemente zur Einstellung von Sequenzparametern einfügt und Informationen zur Pulssequenz darstellt. Abbildung 105 zeigt den unteren Teil des Hauptfensters der Anwendung. Auf der linken Seite können die Parameter eingestellt werden, auf der rechten Seite sieht man Informationen über die Pulssequenz. Über die Knöpfe ‚*Start*' und ‚*Stop*' kann die Berechnung der Sequenz angestossen und jederzeit gestoppt werden. Dieses Verhalten wird komplett in der Oberklasse *PulsesequenceUI* implementiert. Sequenzspezifisch sind die Bedienelemente T_R-Zeit, T_E-Zeit und Flipwinkel ganz links. Folgende Bedienelemente sind nicht sequenzspezifisch und werden daher schon in der Oberklasse *PulsesequenceUI* eingefügt und verwaltet:

- Wahl der Spule
- Bildmatrix
- FoV (Field of View)

[29] UI für „User-Interface".

- Rechteck (Verhältnis von Ausschnittsbreite zu Ausschnittshöhe)
- Schichtdicke
- NEX (Anzahl der Anregungen)
- Phasen-OS (Phasen-Oversampling)
- Frequenz-OS (Frequenz-Oversampling)

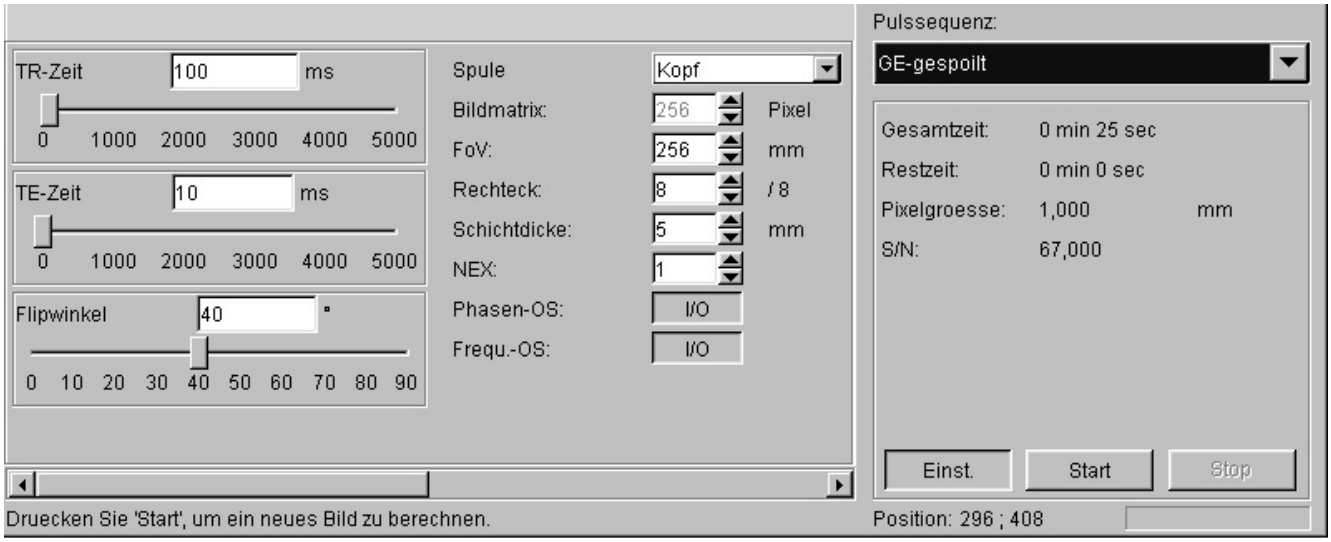

Abbildung 105: Eingabe von Parameterwerten für eine Pulssequenz

Diese nicht sequenzspezifischen Bedienelemente werden in der Methode *fillPanel* erzeugt und dargestellt, die von *SequenceCombo* aufgerufen wird, wenn sich die ausgewählte Pulssequenz geändert hat. Diese Methode ist in den Unterklassen der Benutzerinteraktion überdefiniert, die dort ihre sequenzspezifischen Bedienelemente einfügen und anschließend die Methode der Oberklasse über *super.fillPanel()* aufrufen.

Das wichtigste Attribut von *PulsesequenceUI* ist *sequence*, worin die Referenz auf die zugehörige Instanz der Berechnungsklasse gespeichert ist. Die abstrakten Methoden *setValues* und *getValues* müssen in den Unterklassen implementiert werden. In diesen Methoden wird die Übergabe der Parameter an die Berechnungsklasse der Pulssequenz realisiert. Für die gewählte Pulssequenz in Abbildung 105 müssen zum Beispiel Repititionszeit (T_R), Echozeit (T_E) und der Flipwinkel übergeben werden.

Auch die allgemeinen Eigenschaften der Berechnungsklassen wurden in einer gemeinsamen Oberklasse realisiert, die *Pulsesequence* heißt. Die wichtigste Methode ist *calculate* die in den Unterklassen überdefiniert wird. Aber auch die Methode *calculate* der Oberklasse *Pulsesequence* führt schon einige Operationen aus, so z.B. die Konvertierung des Intensitätsbildes in ein 12-Bit Bild. Außerdem werden einige Einflüsse auf die Bildqualität simuliert, die sich auf den Bildraum auswirken. Über die Methode *addNoiseToPDMatrix* wird zum Beispiel das Rauschen simuliert. Die Simulation der Einflüsse auf die Bildqualität und der Artefakte werden in Kapitel 5.3.7 näher erklärt.

Das Wesentliche einer spezifischen Pulssequenz ist in der Methode *calculate* realisiert. Hier sind die Formeln zur Berechnung der Signalintensität, die in Kapitel 3.3 beschrieben worden sind, in ein JAVA-Programm umgesetzt. Desweiteren wird der Fortschrittsbalken gesetzt und die reale Meßzeit simuliert. Um die reale Meßzeit zu simulieren, wird die Berechnung für eine bestimmte Zeit unterbrochen.

Die Simulation der Meßzeit ist auch ein Grund dafür, warum *Pulsesequence* als Unterklasse von *java.lang.Thread* implementiert ist. Mit einem Aufruf von *currentThread.sleep(int milis)* kann die Berechnung für *milis* Millisekunden unterbrochen werden und damit die reale Dauer einer Messung simuliert werden. Der Wert für die Zeitdauer der simulierten Messung wird von 0-100 Prozent eingestellt. Bei 0 Prozent Simulationszeitfaktor wird die Berechnung ohne Verzögerung durchgeführt,

bei 100 Prozent dauert die Messung in etwa solange wie im realen Tomographen. Gerade bei Schnellbildtechniken kann es aber vorkommen, daß die Berechnung in der Simulation aufgrund aufwendiger Berechnungen länger dauert als in der Realität. So muß zum Beispiel bei der Simulation einer Turbo-Spin-Echo-Sequenz mehrfach eine Fouriertransformation berechnet werden, da die k-Raumzeilen mit unterschiedlichen Echos gefüllt werden. Für jede Echozeit wird eine Fouriertransformation berechnet. In der Realität ist diese Messung durch das Füllen mehrerer Zeilen im k-Raum sehr schnell, in der Simulation dagegen ist die Berechnung der Fouriertransformationen sehr zeitintensiv. Je größer die *ETL* wird, desto kürzer wird die reale Meßzeit, allerdings wird die Simulationszeit immer länger. Ein *Thread* kann außerdem jederzeit durch *stop* unterbrochen werden. Dadurch kann der Benutzer die Berechnung durch Betätigen des Stopknopfes abbrechen.

5.3.5 Das Paket *tools*

Neben einigen Hilfsfunktionen, die als statische Methoden in der Klasse *tools* zusammengefaßt sind, enthält dieses Paket einige häufig gebrauchte grafische Bedienelemente.

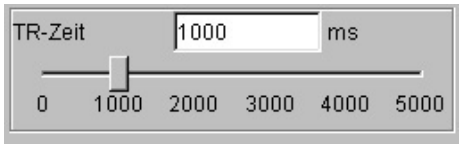

Abbildung 106: Ein Bedienelement vom Typ *SliderPanel* zur Einstellung der Repitionszeit

Das in Abbildung 106 dargestellte Bedienelement ist eine Instanz der Klasse *SliderPanel*. Mit deren Hilfe kann der Benutzer entweder in dem Textfeld einen Wert für die Repititionszeit eingeben oder diesen über den Schiebebalken einstellen. Die Werte des Textfeldes und des Schiebebalkens werden synchronisiert, bei Eingabe in das Textfeld werden die Werte auf Gültigkeit geprüft. Da dieses Verhalten nicht nur für die Repitionszeit benötigt wird, ist es in der Klasse *SliderPanel* zusammengefaßt worden.

Die Klasse *IntegerDocument* erweitert *PlainDocument* und läßt nur Eingaben von Ganzzahlen zu. Eine Instanz dieser Klasse wird über einen Aufruf von *setDocument* für ein Textfeld (z.B. für das Textfeld im *SliderPanel*) als Dokumenttyp festgelegt (siehe dazu [GEAR99], S. 1470 ff.).

Weitere häufig benötigte grafische Bedienelemente sind in den Klassen *LabelTFLabelPanel* und *ThreeLabelPanel* implementiert.

Abbildung 107: Ein Bedienelement vom Typ LabelTFLabelPanel

Das in Abbildung 107 gezeigte Bedienelement *LabelTFLabelPanel* wird z.B. von jeder Pulssequenz zur Einstellung von Parametern wie Anzahl der Anregungen oder Schichtdicke verwendet.

ThreeLabelPanel enthält drei Textanzeigefelder. Instanzen dieser Klasse werden zur Anzeige der Sequenzeigenschaften benutzt (Gesamtzeit, Restzeit usw.).

5.3.6 Das Paket *vmrt*

Im Paket vmrt wird die gesamte Ablaufsteuerung vom Einlesen der Parameterbilder über die Erzeugung der Bilder bis zum Anzeigen und der DICOM-Export der Bilder abgewickelt.

Abbildung 108 zeigt das Klassendiagramm dieses Paketes, wobei zur besseren Übersichtlichkeit nur die wichtigsten Attribute und Methoden angegeben sind.

Die Klasse *VMRT* ist lediglich die Startklasse. Die *main*-Methode instantiiert im wesentlichen ein Objekt vom Typ *VMRT*, im Konstruktor wird wiederum ein Objekt vom Typ *VMRTFrame* instantiiert und sichtbar gemacht.

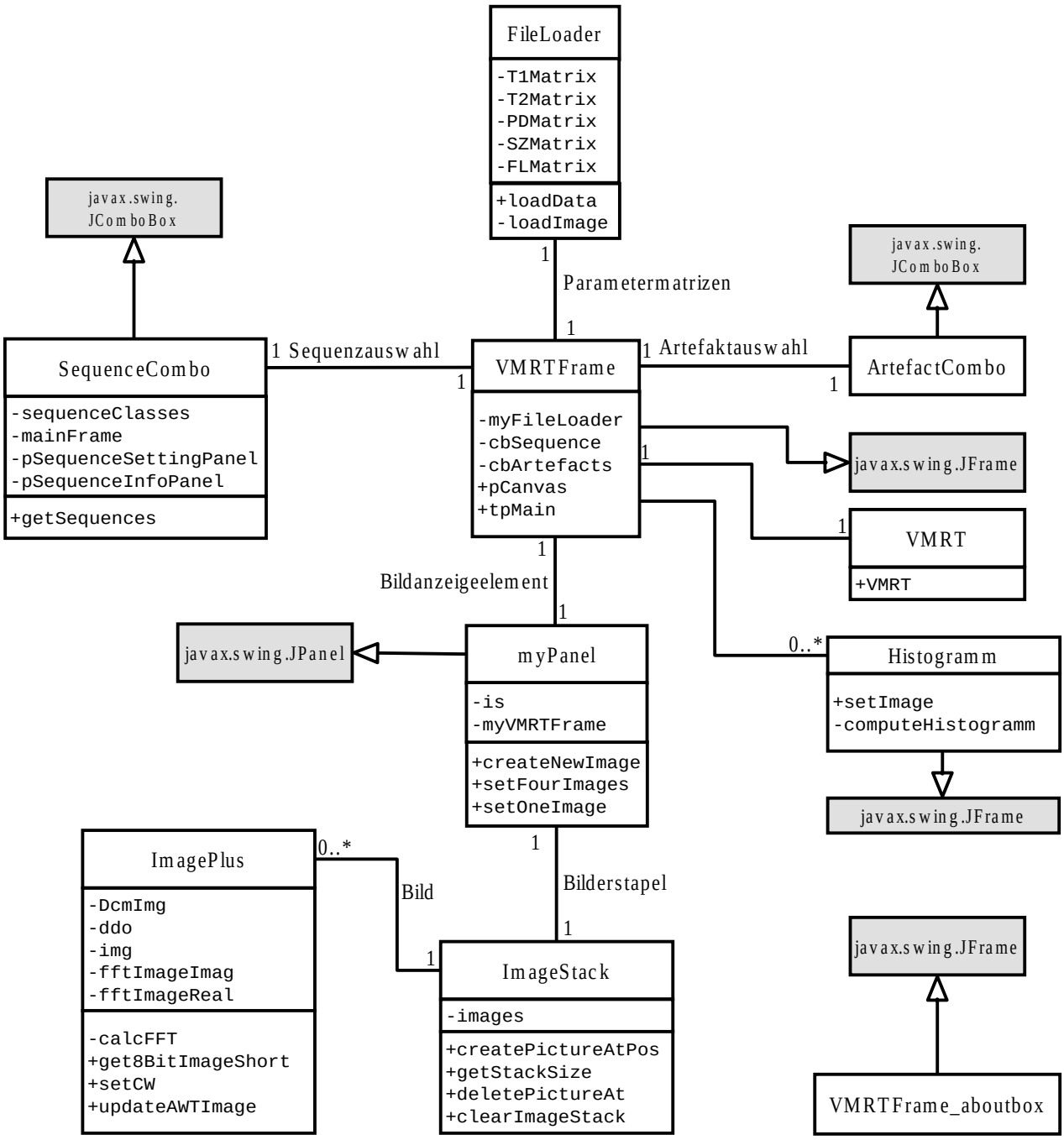

Abbildung 108: Klassendiagramm des Paketes vmrt

VMRTFrame ist die zentrale Klasse des Paketes und des Programmes *Virtual MRT*. Das Klassenattribut *myFileLoader* speichert eine Instanz der Klasse *FileLoader*, die das Einlesen und den Zugriff auf die Parameterbilder regelt. So enthält das Attribut *T1Matrix* die Rohdatenwerte für die T_1-Werte für jeden Bildpunkt. Die anderen Attribute sind mit den entsprechenden weiteren Rohdatenwerten gefüllt (T_2, Protonendichte, Suszeptibilität, Spinfluß). Die Methode *loadData* lädt und interpretiert die Indexdatei und lädt durch Aufrufe der Methode *loadImage* die einzelnen Rohdatenbilder ein, die im DICOM-Format vorliegen. Desweiteren existieren Zugriffsmethoden für jede der Rohdatenmatrizen.

Das Konfigurieren der Pulssequenzen und die Auswahl und Anzeige der Pulssequenz sind in *SequenceCombo* (vgl. dazu 5.3.4) implementiert. Beim Initialisieren von *VMRTFrame* wird die Methode *getSequences* aufgerufen, die die Datei *sequences.properties* einliest und interpretiert. Von jeder UI-Klasse der Sequenzen wird eine Instanz erzeugt und im Vektor *sequenceClasses* gespeichert.

Die Attribute *pSequenceSettingPanel* und *pSequenceInfoPanel* speichern Referenzen auf die Bereiche zur Darstellung der Bedien- und Informationselemente. Dabei stellt die UI-Klasse der Sequenz ihre Bedienelemente in *pSequenceSettingPanel* dar, in *pSequenceInfoPanel* werden Informationen über die Sequenz angezeigt (Restzeit, Signal-zu-Rausch-Verhältnis usw.). Nach der Auswahl einer neuen Sequenz wird der Inhalt von *pSequenceSettingPanel* gelöscht und die Elemente der neuen Sequenz dargestellt.

Über die Artefaktauswahl mit *ArtefactCombo* werden die Einflüsse auf die Messung simuliert, die nicht durch die Messung erzeugt werden. Die Einflüsse Rauschen und Faltung werden schon in den Sequenzen selbst simuliert. In *ArtefactCombo* können zusätzlich weitere Artefakte gewählt werden, dabei existiert wiederum für jedes Artefakt eine UI-Klasse und eine Berechnungsklasse. In der Auswahlbox kann zur Zeit entweder kein Artefakt oder die Simulation von Bewegungsartefakten ausgewählt werden. Weitere Artefakte können über die Datei *artefacts.properties* konfiguriert werden (vgl. Kapitel 5.5.2).

Die Anzeige und das Verwalten der erzeugten Bilder wird über eine Instanz der Klasse *myPanel* gesteuert. Das Attribut *is* enthält eine Referenz auf ein Objekt vom Typ *ImageStack*. *ImageStack* enthält in *images* einen Vektor von Bildern als Instanzen vom Typ *ImagePlus*. *ImagePlus* enthält das erzeugte Bild in drei Ausprägungen:

- *DcmImg* ist eine Referenz auf ein Objekt vom Typ *DcmImage* aus dem DICOM-Paket, die u.a. 12-Bit-Daten des erzeugten Bildes enthält.
- *ddo* ist eine Referenz auf ein Objekt vom Typ *DcmDataObject*. Sie enthält zusätzliche DICOM-Informationen, denn die erzeugten Bilder können auch als DICOM-Datei exportiert werden. Dieses *DcmDataObject* enthält allerdings kein DICOM-Tag (siehe Kapitel 5.3.1) für die Pixeldaten, da diese separat vorliegen.
- *img* ist eine Referenz auf ein Objekt vom Typ *java.awt.image*. Das ist das gefensterte 8-Bit Bild, welches letztendlich dargestellt wird. Dieses *Image*-Objekt wird vor allem aus Performancegründen in einem Attribut gespeichert. Es wird auch im *DICOM-Viewer* benutzt, denn für die Animation eines Bilderstapels wäre die ständige Neuberechnung des 8-Bit-Bildes zu langsam.

Mit den Methoden *getFFTImageImag* und *getFFTImageReal* kann man den Imaginär-, bzw. den Realteil des fouriertransformierten Bildes auslesen. Damit dieser nicht jedesmal neu berechnet wird, sind die Ergebnisse der Fouriertransformation in den Attributen *fftImageImag* bzw. *fftImageReal* gespeichert. Wenn eine Neuberechnung notwendig ist, findet diese mittels *calcFFT* statt. *Get8BitImageShort* liefert das 8-Bit-Bild als Feld vom Typ *short[][]* zurück. Die Fensterung des 12-Bit-Bildes wird mit *setCW* festgelegt. Hat sich die Fensterung des Bildes geändert, oder wurde ein neues Bild an dieser Stelle erzeugt, kann mit *updateAWTImage* ein neues 8-Bit-Bild berechnet werden.

Der Stapel *ImageStack* enthält Zugriffsmethoden auf die einzelnen Bilder vom Typ *ImagePlus* wie *createPictureAtPos* um ein Bild einzufügen, *deletePictureAtPos* um ein Bild zu löschen und *getPictureAtPos* um ein Bild auszulesen usw. Die Methode *clearImageStack* löscht den ganzen Stapel und *getStackSize* liefert die aktuelle Größe des Stapels zurück.

Die Ereignisse aller weiteren Bedienelemente, die in *VMRTFrame* eingefügt werden, werden ebenfalls in dieser Klasse verwaltet.

5.3.7 Simulation von Artefakten

Die Artefakte, die im Kapitel 3.6 beschrieben wurden, werden teilweise schon im Paket *sequences* implementiert, sofern sie bei jeder Messung auftreten, andere Artefakte (aktuell nur Bewegungsartefakte) werden im Paket *artefacts* implementiert.

Rauschen

Die Simulation des normalverteilten Rauschens wird in der Oberklasse der Pulsesequenzen *PulseSequence* implementiert. Der Benutzer kann im *Virtual MRT* auch zwischen einer Kopfspule und einer Körperspule wählen, wobei die Körperspule ein stärkeres Grundrauschen erzeugt. Da *PulseSequence* eine Unterklasse von Thread ist, wird in der Methode *run* vor dem Aufruf der Berechnungsfunktion *calculate* zum Simulieren von Rauschen die Methode *addNoiseToPDMatrix* aufgerufen. Folgender Programmtext simuliert das Rauschen:

```
int coil = myUIClass.getCoil();
Random myRandomizer = new Random();
double noise = 1.0 / myUIClass.getSNRatio();

for (int x=0; x<PDMatrix[0].length; x++)
{
  for (int y=0; y<PDMatrix.length; y++)
  {
    int basicnoise = 0;

    switch (coil)
    {
      case 0: { basicnoise = 100; break;}
      case 1: { basicnoise = 5000; break;}
    }

    int val = PDMatrix[x][y];
    NoisyPDMatrix[x][y] = val + (int)Math.abs(((val*noise)*
      myRandomizer.nextGaussian())) +
     (int)Math.abs(basicnoise*myRandomizer.nextGaussian());
  }
}
```

Zur Simulation des Rauschens wird die originale Protonendichtematrix *PDMatrix* mit einer gaußschen Zufallsverteilungsfunktion überlagert. Zum einen wird der gesamten Protonendichtematrix abhängig von der gewählten Meßspule ein Grundrauschen überlagert, das in *basicnoise* berechnet wird. Zum anderen wird abhängig vom Signal-zu-Rausch-Verhältnis die Protonendichtematrix zusätzlich mit einem Rauschen überlagert. Die Berechnungsfunktion für die spezielle Pulssequenz arbeitet anschließend auf der Protonendichtematrix mit simuliertem Rauschen *NoisyPDMatrix* und erzeugt somit ein verrauschtes Bild.

Suszeptibilitätsartefakte

Suszeptibilitätsartefakte werden bisher nicht simuliert, allerdings wird zusätzlich zu den Parametermatrizen für T_1-Zeiten, T_2-Zeiten und Protonendichte eine Matrix für Suszeptibilitätswerte eingelesen. Spätere Versionen des *Virtual MRT* können diese Parametermatrix zur Simulation von Suszeptibilitätsartefakten nutzen.

Einfaltung

Da wir mit einer festen Parametermatrix von 256*256 Punkten arbeiten, können Einfaltungen nur auftreten, wenn das *FOV* kleiner eingestellt wird. Die Simulation erfolgt im Bildbereich in der Methode *simulateXAliasing* in der Klasse *PulseSequence*. Die Einfaltung wird nach dem Berechnen der Intensitätsmatrix erzeugt, falls das Frequenzoversampling ausgeschaltet ist. Die Methode berechnet dazu zu jedem Punkt im *FOV*, welche Bildanteile von außerhalb des *FOV* liegenden Punkten auf diesen Punkt eingefaltet werden und überlagert die Intensitäten durch Addition.

Bewegungsartefakte

Zur Simulation von Bewegungsartefakten haben wir vereinfachend angenommen, daß sich der gesamte Bildbereich in einer einfachen Translation oder in einer periodischen Bewegung verschiebt.

Parameter wie Bewegungsrichtung und Geschwindigkeit oder bei der periodischen Bewegung Frequenz können vom Benutzer über die Karteikarte *‚Artefakte‘* eingestellt werden.

Berechnung und Benutzerinteraktion sind ebenso wie für die Pulssequenzen in UI-Klasse und Berechnungsklasse aufgeteilt. Die Bewegungsartefakte werden in der Klasse *Motion* berechnet. Da sich eine Bewegung während der Aufnahme je nach Pulssequenz unterschiedlich auf das erzeugte Bild auswirken kann, wird in der Methode *calculate* zunächst eine Fallunterscheidung nach der Pulssequenz gemacht. Zur Berechnung von Bewegungsartefakten für die Sequenzen Saturation-Recovery, Inversion-Recovery, Spin-Echo, Gradienten-Echo werden zu 16 verschiedenen Zeitpunkten die Fouriertransformationen des Intensitätsbildes berechnet. Abhängig von Zeitpunkt, Art und Geschwindigkeit der Bewegung ist das Intensitätsbild unterschiedlich verschoben. Aus den 16 fouriertransformierten Intensitätsbildern werden jeweils 16 Zeilen für den k-Raum des Ergebnisbildes ausgewählt.

Diese 16*16 k-Raum-Zeilen ergeben zusammengenommen den k-Raum des Bildes mit simulierten Bewegungsartefakten. Dieser wird am Ende noch zurücktransformiert, um wieder ein Intensitätsbild zu erzeugen.

Flußartefakte

Ebenso wie die Suszeptibilitätsartefakte sind die Flußartefakte zur Zeit nicht implementiert, aber es wird bereits eine Parametermatrix für Flußgeschwindigkeiten eingelesen. Eine spätere Version des *Virtual MRT* kann diese Parametermatrix dann nutzen, um Flußartefakte zu simulieren.

5.4 Das Nachbearbeitungswerkzeug *DICOM-Viewer*

Das Paket *dicomviewer* erfüllt für das Nachbearbeitungswerkzeug *DICOM-Viewer* eine ähnliche Aufgabe wie das Paket *vmrt* für das Meßwerkzeug *Virtual MRT*. Es verwaltet die gesamte Ablaufsteuerung des *DICOM-Viewers* vom Einlesen eines DICOM-Einzelbildes oder einer Bildeserie, über das Nachver- und -bearbeiten bis hin zum Export der Bilder im DICOM-Format.

Abbildung 109 zeigt das Klassendiagramm des *dicomviewer*-Paketes, wobei auch hier wieder nur die wichtigsten Attribute und Methoden angegeben sind und die paketfremden Klassen grau hinterlegt dargestellt sind. Von diesen sind in der Abbildung sehr viele vorhanden, was nicht nur andeutet, daß das Projekt in verschiedene Pakete unterteilt wurde, sondern auch die Wiederverwendung des Quelltextes zeigt. Insbesondere im Bereich der Bilderverwaltung arbeiten das Meßwerkzeug *Virtual MRT* und der *DICOM-Viewer* sehr ähnlich. Daher werden die Klassen *ImageStack* und *ImagePlus* unverändert für beide Programme genutzt.

Die Klasse *DicomViewer* ist die Startklasse des Nachbearbeitungswerkzeugs. Die *main*-Methode instantiiert ein Objekt vom Typ *DicomViewer* und in dessen Konstruktor wird eine Instanz der Klasse *ViewerFrame* erzeugt und dargestellt.

ViewerFrame stellt die zentrale Klasse dar. Sie enthält alle Bedienelemente zur Steuerung des *DICOM-Viewer* Hauptfensters. Die Methode *load* lädt direkt ein einzelnes DICOM-Bild aus der übergebenen Datei. Die Methode *loadSeries* instantiiert zum Laden einer Bildserie hingegen zuerst ein Objekt vom Typ *SeriesLoader*. Dazu wird dem Objekt der Verweis auf eine Datei mitgeteilt, die ein Bild der zu ladenden Serie enthält. Anhand dieses Bildes werden alle Bilder ermittelt, die zur gleichen Serie gehören und sich im gleichen Verzeichnis befinden. Sobald dies geschehen ist, werden alle Bilder der Serie geladen. Dabei wird auf Methoden des Pakets *dcm.dicom* zurückgegriffen (siehe Kapitel 5.3.1). Da die Klasse *SeriesLoader* die Klasse *Thread* erweitert, kann dem Benutzer während des Ladens eine Rückmeldung über den Fortschritt mitgeteilt werden.

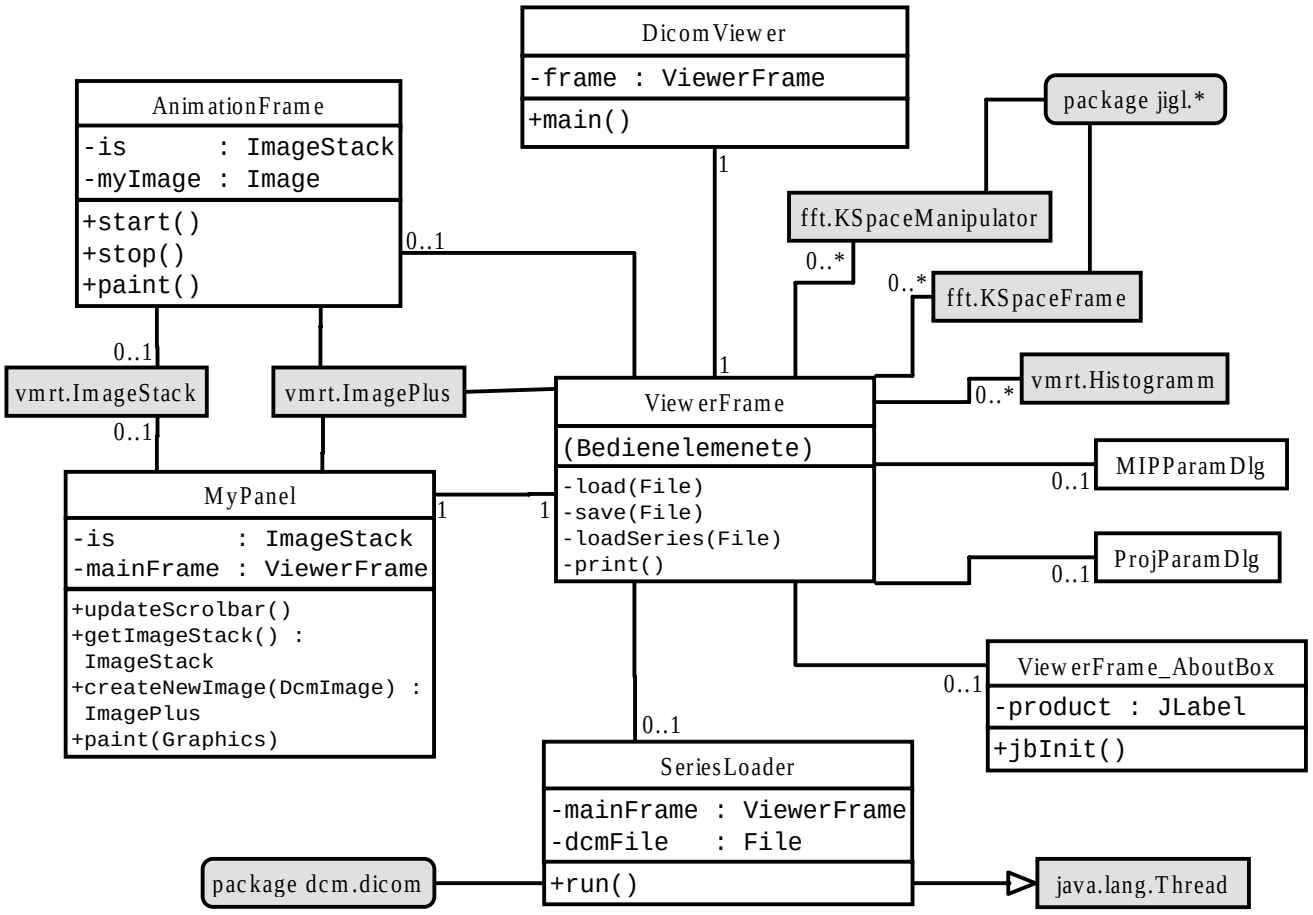

Abbildung 109: Klassendiagramm für das Paket *dicomviewer*

Nachdem ein oder mehrere Bilder geladen wurden, stehen dem Benutzer zahlreiche Möglichkeiten zu deren Ver- und Bearbeitung zur Verfügung. Einfache Werkzeuge, wie Lupe, Abstandsmessung, Winkelmessung, Spiegeln, Drehen und Invertieren werden direkt von *ViewerFrame* und der darin enthaltenen Instanz der Klasse *MyPanel* zur Verfügung gestellt und verwaltet. Diese beiden Objekte kommunizieren über gegenseitige Rückreferenzen. Während in *ViewerFrame* die Ereignisbehandlung (des Mauszeigers auf der Zeichenfläche) und die notwendigen Berechnungen stattfinden, ist *MyPanel* für das Neuzeichnen der dargestellten Bilder verantwortlich, wann immer dies nötig ist.

MyPanel verwaltet eine Instanz der Klasse *ImageStack*, die aus dem Paket *vmrt* wiederverwendet wird. *ImageStack* wiederum enthält in der Instanzvariablen *images* einen Vektor von Bildern als Instanzen von *ImagePlus*. *ImagePlus* schließlich enthält das 12-Bit Bild, das gefensterte 8-Bit Bild und zusätzliche DICOM-Informationen.

Höherwertige Werkzeuge werden durch eigene Klassen repräsentiert. So kann mittels der Klasse *Histogramm* des Pakets *vmrt* das Histogramm des aktiven Bildes angezeigt werden. Die Klasse kapselt das dafür notwendige Darstellungsfenster und die Berechnungsfunktionen.

Gleiches gilt für die Klassen *KSpaceFrame* und *KSpaceManipulator* des Pakets *fft*. *KSpaceFrame* ermöglicht das Anzeigen der Fouriertransformation des selektierten Bildes. Dabei kann zwischen Real- und Imaginärteil und zwischen Phasen- und Magnitudenbild umgeschaltet werden. Die Klasse *KSpaceManipulator* ermöglicht das Anzeigen und Manipulieren des Magnitudenbildes der Fouriertransformation des selektierten Bildes. Anschließend kann eine Rücktransformation durchgeführt und das Ergebnis betrachtet werden. Beide gerade beschriebenen Klassen verwenden zur Berechnung der Fouriertransformation das externe Paket *jigl* (siehe Kapitel 5.3.3).

Die Klasse *AnimationFrame* kapselt die Bedienelemente und Berechnungsfunktionen zur Erzeugung einer Animation aus den geladenen Schichtbildern. Dazu wird der Klasse die Referenz auf den

aktuellen Bildstapel (*ImageStack*) übergeben. Da die Klasse einen *Thread* implementiert findet die Hauptaktion in der *run*-Methode statt. Dort werden die Einstellungen der Bedienelemente ausgewertet und aufgrund dessen entschieden, welches Bild als nächstes dargestellt werden muß. Dieses Bild wird in der Instanzvariablen *myImage* gespeichert und von der *paint*-Methode auf der Zeichenfläche dargestellt. Die Methoden *start* und *stop* werden aufgerufen, wenn die entsprechenden Knöpfe im Animationsfenster gedrückt werden. Sie erzeugen und starten bzw. stoppen und zerstören das *Thread*-Objekt, in dem die Animation stattfindet.

Die Klassen *MIPParamDlg* und *ProjParamDlg* implementieren einfache Dialogfenster zur Parameterabfrage bei der Berechnung eines MIP-Bildes oder einer 90°-Projektion. Nach Betätigung des ‚*OK*'-Knopfes des entsprechenden Fensters werden die Parameterwerte ausgelesen und die entsprechende Methode von *ViewerFrame* (*calcMIPImage* oder *calcProjImage*) mit diesen Parametern aufgerufen.

Das Dialogfenster *ProjParamDlg* bietet unter anderem eine Möglichkeit, eine Interpolation ein- bzw. auszuschalten. Die Interpolation erfolgt dabei nur entlang einer Linie entsprechend folgenden Queltextes, ohne das die Intensitäten der Nachbarlinie berücksichtigt werden:

```
slicenum = fromSlice;

for (int x=0; x<size; x++)
{
  if ( x >= ((int)(slicenum*step)) )
  {
    slicenum++;
  }

  double factor1 = ((step-(x-((slicenum-1)*step)))/step);
  double factor2 = ((x-((slicenum-1)*step))/step);

  // Berechnen des x-Wertes, wo der naechste neue Grauwert zur
  // Verfuegung steht
  int nextx = (int)((slicenum)*step);

  if (nextx > size-1) nextx = size-1;

  for (int y=0; y<size; y++)
  {
    // Berechnen des interpolierten Grauwertes an der akt. Position
    short hlp = (short)Math.abs((factor1*projImg[(y*size) + x    ] ) +
                                (factor2*projImg[(y*size) + nextx] ));
    projImg[(size*y)+x] = hlp;
  } // for y
} // for x
```

5.5 Erweiterungsmöglichkeiten

In den folgenden Unterkapiteln sollen einige Möglichkeiten beschrieben werden, unser System zu erweitern, ohne die von uns erstellten Quelltextdateien zu verändern. Wir haben zwei Schnittstellen implementiert, die dies ermöglichen. Die eine davon ermöglicht das Hinzufügen neuer Pulssequenzen, die andere das Hinzufügen neuer Artefakt-Simulatoren.

5.5.1 Hinzufügen neuer Pulssequenzen

Das Hinzufügen neuer Pulssequenzen zum Meßwerkzeug *Virtual MRT* ist besonders einfach gestaltet. Es sollte einem erfahrenen JAVA-Programmierer ohne Probleme möglich sein, die notwendigen Klassen zu erstellen und die erforderlichen Einträge vorzunehmen.

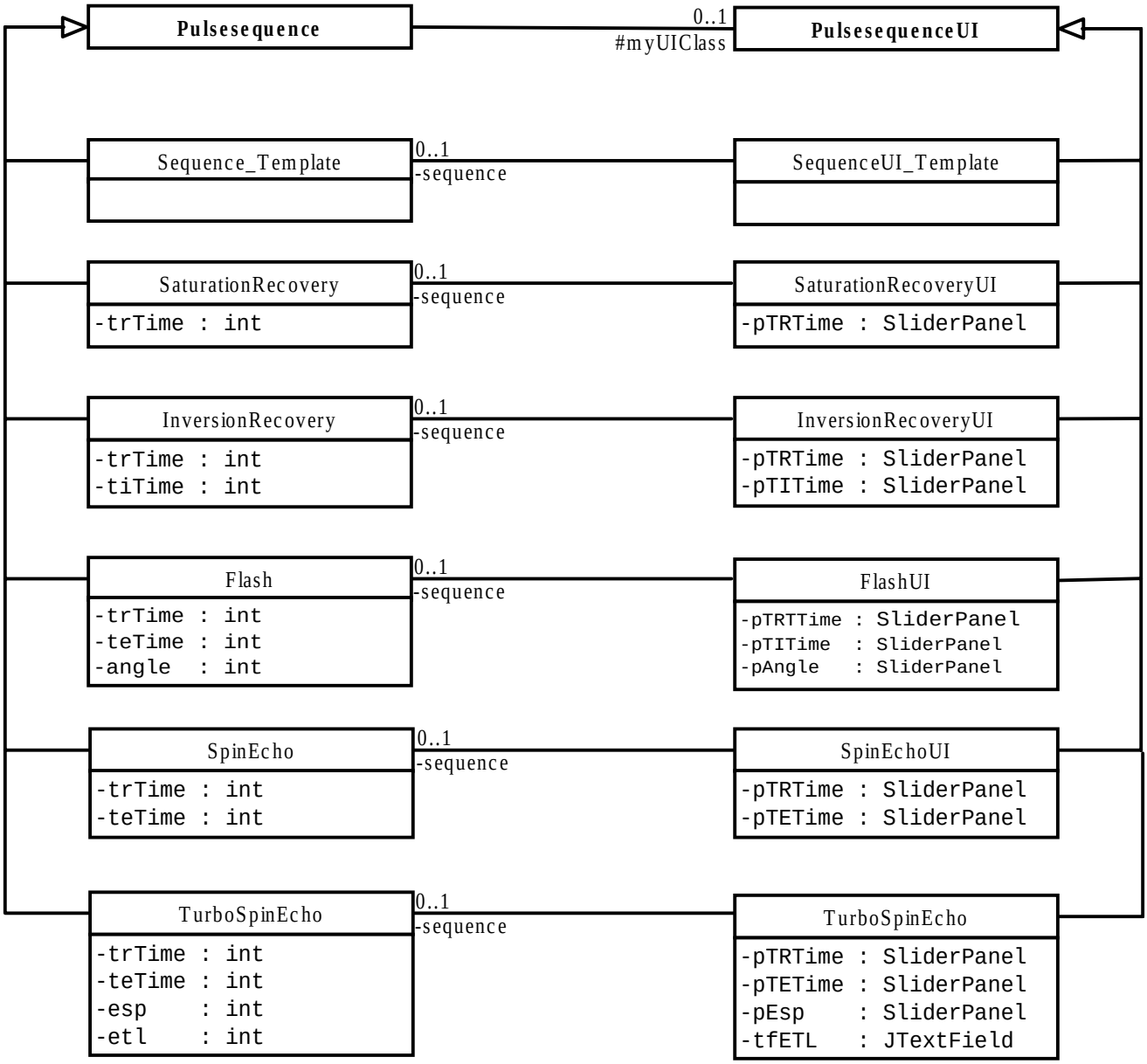

Abbildung 110: Klassendiagramm der Pulssequenz-Klassen

Zum Erstellen einer Pulssequenz ist es nötig, 2 neue JAVA-Klassen zu implementieren. Eine davon repräsentiert die Oberflächenelemente der Pulssequenz, wie zum Beispiel eine Einstellungsmöglichkeit für die Repititionszeit, die andere Klasse implementiert die Berechnungsvorschriften der Pulssequenz und ist das eigentliche Arbeitspferd der Pulssequenz. Nachdem die beiden Klassen erstellt wurden, ist es noch nötig, die neue Sequenz am *Virtual MRT* anzumelden. Dazu ist ein Eintrag in der *Property*-Datei *sequences.properties* notwendig.

Wir wollen das Erstellen einer neuen Pulssequenz anhand der schon vorhandenen SaturationRecovery-Sequenz exemplarisch erarbeiten.

Abbildung 110 zeigt alle in der vorliegenden Version des *Virtual MRT* vorhandenen Pulsequenzklassen, einschließlich der zu erstellenden Klassen *SaturationRecovery* (Berechnunungsklasse) und *SaturationRecoveryUI* (Oberflächenklasse). Aus der Abbildung geht auch hervor, daß alle Berechnungklassen von der abstrakten Oberklasse *Pulsesequence* erben und alle Oberflächenklassen von der abstrakten Oberklasse *PulsesequenceUI*. Außerdem erkennt man, daß jede Berechnungsklasse automatisch eine Referenz auf die dazugehörige Oberflächenklasse enthält, welche zum Auslesen der Parameterwerte aus den Oberflächenelementen benötigt wird. Des weiteren ist auch ersichtlich, daß jede Oberflächenklasse eine Referenz auf die Berechnungsklasse enthalten muß, da

beim Betätigen des ‚*Start*'-Knopfes der Pulssequenz eine neue Instanz der Berechnungsklasse angelegt werden muß und deren Berechnungsmethode (*calculate*) aufgerufen werden muß. Die Referenz der Oberflächenklasse auf die Berechnungsklasse kann allerdings noch nicht in der Oberklasse *Pulsesequence* erzeugt werden, da die Berechnungsklasse in einem *Thread* läuft und bei jedem Betätigen des ‚*Start*'-Knopfes der Pulssequenz eine neue Instanz angelegt werden muß.

Neben den im *Virtual MRT* vorhandenen Pulssequenzen sind in Abbildung 110 auch zwei Klassen mit den Namen *Sequence_Template* und *SequenceUI_Template* zu sehen. Dieses sind zwei Klassen, die als Vorlage zur Erstellung einer neuen Pulssequenz dienen. Sie enthalten das Gerüst für jede Oberflächen- und Berechnungsklasse. Zum Erstellen der beiden Klassen einer neuen Pulssequenz ist es lediglich nötig, die beiden Vorlagenklassen umzubenennen und einige Ergänzungen darin vorzunehmen, so daß die entsprechenden Oberflächenelemente erzeugt und dargestellt werden und die richtigen Berechnungen durchgeführt werden. An welcher Stelle welche Ergänzungen vorgenommen werden müssen, ist in den Vorlagenklassen dokumentiert. Wir wollen daher nicht die kompletten Klassen abdrucken, sondern lediglich einige wichtige Punkte herausstellen.

Erstellen einer neuen GUI-Klasse

Beim Erstellen einer neuen Pulssequenz sollte man sich zunächst Gedanken über die notwendigen und optionalen Parameter der Sequenz machen. Diese Parameter sollten durch Oberflächenelemente, die ein Veränderung der Parameter zulassen, repräsentiert werden. Als Zeichenbereich für diese Oberflächenelemente steht dem Entwickler ein Rahmen mit einer Breite von 512 und einer Höhe von 256 Pixeln zur Verfügung. In diesem Rahmen befinden sich allerdings schon einige Bedienelemente, die für jede Pulssequenz sinnvoll sind und daher von der Oberklasse *PulsesequenceUI* geerbt werden (Abbildung 111). Und zwar sind das Bedienelemente zur Auswahl der Spule, der Bildmatrixgröße, des Aufnahmebereichs, der Schichtdicke, der Anzahl der Anregungen und zum Ein- bzw. Ausschalten des Phasen- bzw. Frequenzoversamplings.

Abbildung 111: Allgemeiner Rahmen für die GUI-Elemente einer Pulssequenz

Für die nun zu entwickelnde Saturation-Recovery-Sequenz ist ein weiteres Bedienelement zur Einstellung der Repititionszeit sinnvoll. Wir wollen dafür einen Schieberegler mit Textfeld und Beschriftung anlegen, so das der Rahmen schließlich so aussieht, wie in Abbildung 112 dargestellt.

Man kann nun diese 3 Oberflächenelemente getrennt anlegen, allerdings enthält das von uns bereitgestellte Paket *tools* bereits eine Klasse namens *SliderPanel*, die die drei Bedienelemente in einem Rahmen vereinigt enthält und automatisch das Textfeld mit dem Schieberegler koordiniert. Das bisher erarbeite Klassengerüst besteht also aus der Klassendefinition, der Definition einer Referenz auf die entsprechende Berechnungsklasse, der Definition des *SliderPanles*, der Definition eines voreingestellten Wertes für die Repititionszeit und einem Standardkonstruktor, der nichts leistet.

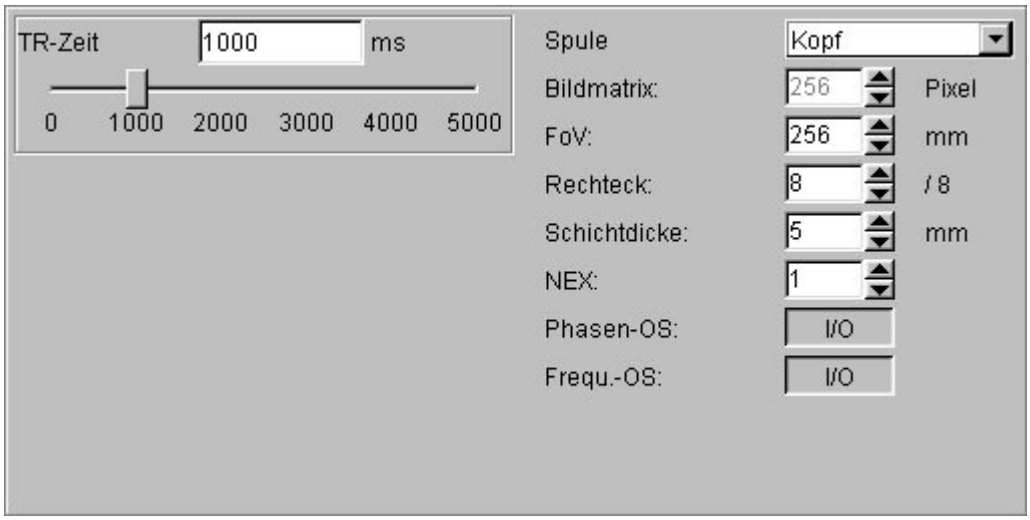

Abbildung 112: Rahmen für die GUI-Elemente der SR-Sequenz

Das Grundgerüst der Oberflächenklasse ist im folgenden dargestellt.

```
public class SaturationRecoveryUI extends PulsesequenceUI {

  private SliderPanel ptrPanel;
  private SaturationRecovery sequence;
  private int trTime = 1000;

  public SaturationRecoveryUI() {}
}
```

Als nächstes erfolgt das Erstellen und Darstellen des *SliderPanels* in der *jbInit*-Methode. Diese sieht dann in etwas vereinfachter Form etwa folgendermaßen aus:

```
void jbInit() throws Exception
{
  // Erzeugen eines neuen Slider-Panels
  ptrPanel = new SliderPanel("TR-Zeit", 0, 5000, "ms");
  // Einrichten der Position und Größe des Slider-Panels
  ptrPanel.setBounds(new Rectangle(5, 5, 250, 70));
  // Wertebereich des Textfeldes beschränken
  ptrPanel.setTextRange(0,9999);

  // Einrichten einer Methode, die die Berechnungszeit
  // ermittelt
  ptrPanel.getTextFieldReference().addCaretListener(
    new CaretListener()  {
      public void caretUpdate(CaretEvent e) {
        updateTotalTime();
      }
  });

  // Darstellen des Slider-Panels
  myPanel.add(ptrPanel);
}
```

Die darin enthaltene Methodenaufruf *updateTotalTime()* ermittelt anhand der aktuell eingestellten Parameterwerte die Zeit, die die Berechnung eines neuen MRT-Bildes benötigen würde. Das Ergebnis wird schließlich durch die Methode *displayTotalTime* im Kontrollbereich des *Virtual MRT* dargestellt. Der Entwickler der neuen Pulssequenz muß lediglich die Formel zur Berechnung der Zeit eingeben. Diese kann von Sequenz zu Sequenz variieren, da unter Umständen unterschiedliche Parameter einfließen. Um die Darstellung der Zeit muß der Entwickler sich nicht kümmern, da diese Aufgabe von der Oberklasse übernommen wird.

```
void updateTotalTime()
{
  long totaltime = 256*ptrPanel.getValue()*pNEX.getValue());
  displayTotalTime(totaltime);
}
```

Schließlich muß noch die Methode *pbStart_actionPerformed* der Oberklasse erweitert werden. Die Methode wird aufgerufen, wenn der Startknopf der Pulssequenz gedrückt wird. In diesem Fall muß eine neue Instanz der entsprechenden Berechnungsklasse erzeugt werden. Dieser müssen die Parameterwerte mitgeteilt werden und schließlich muß die Berechnung eines Intensitätsbildes angestoßen werden.

```
void pbStart_actionPerformed(ActionEvent e)
{
  // Weiterleiten des Aufrufs an die Oberklasse
  super.pbStart_actionPerformed(e);

  // Erzeugen einer neuen SaturationRecoveryInstanz
  sequence = new SaturationRecovery();
  sequence.setUI(this);
  // Auslesen und übergeben der Parameterwerte
  sequence.setTRTime(ptrPanel.getValue());
  // Erzeugen eines Intensitätsbildes mit der aktuellen Sequenz
  mainFrame.createIntensityImage(sequence);
}
```

Erstellen einer neuen Berechnungsklasse

Neben der Oberflächenklasse muß für jede Pulssequenz auch eine Berechnungsklasse erstellt werden. Diese berechnet aus den Rohdatenmatrizen und den eingestellten Parameterwerten eine Intensitätsmatrix und schließlich ein 12-Bit Bild. Jede Berechnungsklasse erbt von der Klasse *Pulsesequence* einige grundlegende Funktionalität. Dazu gehört zum Beispiel die Umwandlung des Intensitätsbildes in ein 12-Bit Bild. Das Grundgerüst der Saturation-Recovery-Berechnungsklasse ist im folgenden dargestellt.

```
public class SaturationRecovery extends Pulsesequence {

  public SaturationRecovery() {
  }

}
```

Die zentrale Methode jeder Berechnungsklasse ist die Methode *calculate*, daher muß sie in jeder Berechnungsklasse vorhanden sein. In ihr findet die Berechnung des Intensitätsbildes aus den Rohdatenmatrizen statt. Außerdem wird die Fortschrittsanzeige weiter gesetzt und ein ggf. eingestellter Simulationszeitfaktor berücksichtigt. Die *calculate*-Methode der Saturation-Recovery-Sequenz stellt sich folgendermaßen dar:

```
public void calculate()
{
  // Durchlaufen der Rohdatenmatrizen in x-Richtung
  for (int x = 0; x < PDMatrix.length; x++)
  {
    // Weitersetzen der Fortschrittsanzeige
    progressBar.setValue(x);
    try
    {
      // Berechnen und darstellen der Restzeit der Berechnung
      int nex = myUIClass.pNEX.getValue();
```

```
        long remainingtime=trTime*(256-x)*nex;
        myUIClass.displayRemainingTime(remainingtime);

        // Beruecksichtigen eines Simulationszeitfaktors
        sleep((long)((((double)(iTimeFactor))/100)*trTime));
      } catch (Exception e) {}

      // Durchlaufen der Rohdatenmatrizen in y-Richtung
      for (int y = 0; y < PDMatrix[0].length; y++)
      {
        // Berechnen des Intesitätswerts an der Stelle (x;y)
        IntensityMatrix[x][y] = (int) ((NoisyPDMatrix[x][y]) *
                                (1 - Math.exp((double)(-trTime) /
                                (double)(T1Matrix[x][y]))));
      } // for y
    } // for x


    // In super.calculate wird die Fortschrittsanzeige
    // zurückgesetzt und dem Hauptfenster mitgeteilt, daß die
    // Berechnung fertig ist.
    super.calculate();
  }
```

Neben der *calculate*-Methode sollte jede Berechnungsklasse auch eine Methode haben, die einem *ImagePlus*-Objekt die Parameter mitteilen kann, mit dem die Berechnung durchgeführt wurde. Nach Erzeugung des Intensitätsbildes in der *calculate*-Methode wird daraus nämlich ein 12-Bit Bild berechnet. Dieses wird in einem Objekt der Klasse *ImagePlus* mit einigen zusätzlichen Attributen gespeichert. Zu diesen Attributen gehören unter anderem die Fensterungswerte aber auch die Sequenzparameterwerte. Werden diese Werte im *ImagePlus*-Objekt gesetzt, kann daraus eine DICOM-Datei erzeugt werden, die alle wichtigen Bildparameter enthält. Im folgenden ist die Methode *addSequenceParameterToImage* zum Setzen der Parameterwerte abgedruckt:

```
protected void addSequenceParameterToImage()
{
  ResultIP.setTR((double)trTime);
  ResultIP.setSequence("SR");
}
```

Anmelden der neuen Pulssequenz

Zum Schluß muß die neue Pulssequenz noch am *Virtual MRT* angemeldet werden. Dazu muß die Datei *sequences.properties* angepaßt werden. Diese Datei befindet sich im gleichen Verzeichnis wie auch die Klassendatei *VMRTFrame.class*. Für jede Pulssequenz müssen in der *Property*-Datei zwei Zeilen eingefügt werden. Eine Zeile enthält den Namen der Berechnungsklasse der Pulssequenz und die andere Zeile enthält den in der Anwendung darzustellenden Namen für die Sequenz. Im folgenden ist der Inhalt der aktuellen *Property*-Datei dargestellt.

```
Sequence_Class_1 = "SaturationRecovery"
Sequence_Name_1  = "Saturation-Recovery"
Sequence_Class_2 = "InversionRecovery"
Sequence_Name_2  = "Inversion-Recovery"
Sequence_Class_3 = "SpinEcho"
Sequence_Name_3  = "Spin-Echo"
Sequence_Class_4 = "Flash"
Sequence_Name_4  = "GE-gespoilt"
Sequence_Class_5 = "TurboSpinEcho"
Sequence_Name_5  = "Turbo-Spin-Echo"
```

Nachdem die eben beschriebenen Schritte durchgeführt wurden, erscheint die neue Pulssequenz beim nächsten Start des *Virtual MRT* in der Auswahlbox der Pulssequenzen und kann zur Berechnung eines MRT-Bildes verwendet werden.

5.5.2 Hinzfügen neuer Artefakt-Simulatoren

Nachdem im vorhergehenden Kapitel das Erstellen neuer Pulssequenzen erklärt wurde, wird hier nun das Erstellen neuer Artefakt-Simulatoren erläutert. Das Vorgehen dabei ist im Grunde ähnlich, jedoch etwas komplexer, da die Artefakt-Simulation mit den Pulssequenzen zusammenarbeiten muß.

Auch zum Erzeugen eines neuen Artefakt-Simulators müssen zwei neue Klassen erstellt werden. Wiederum eine Oberflächenklasse und eine Berechnungsklasse. Die Berechnungsklasse muß zusammen mit einem Namen für den Manipulator in die Datei *artefacts.properties* eingetragen und dadurch am *Virtual MRT* angemeldet werden. Darüber hinaus muß der Artefakt-Simulator durch einen oder mehrere Einträge in die Datei *sequences.properties* für jede Pulssequenz einzeln freigeschaltet werden.

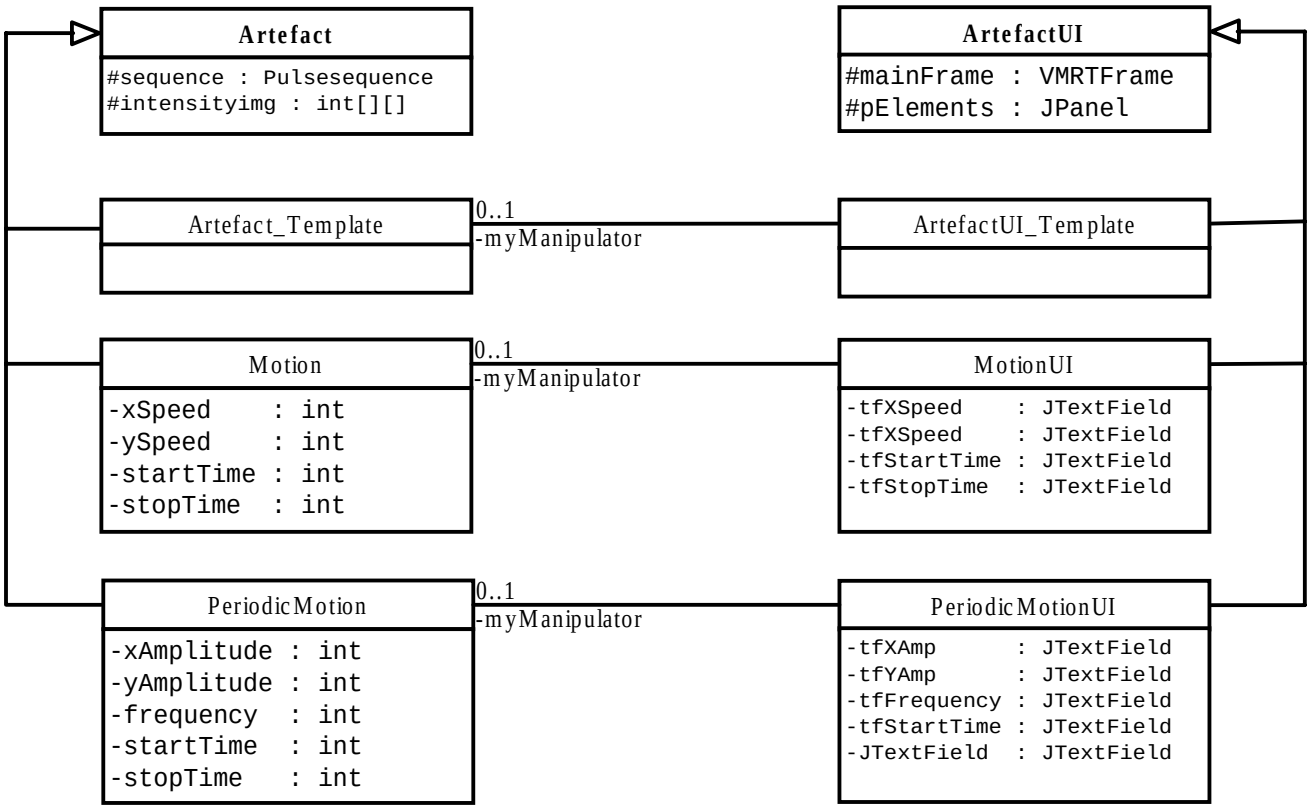

Abbildung 113: Klassendiagramm der Artefakt-Klassen

Abbildung 113 zeigt zunächst wieder alle bisher im Paket *artefacts* vorhandenen Klassen. Jede Berechnungsklasse eines speziellen Artefakt-Simulators erbt von der abstrakten Oberklasse *Artefact*. Dadurch besitzt jede Berechnungsklasse automatisch eine Referenz auf die Pulssequenzklasse, von der aus sie erzeugt und aufgerufen wurde und darüber hinaus ist bereits das Intensitätsbild, das durch die Pulssequenzberechnung erzeugt wird in jeder Artefakt-Klasse enthalten. Jede spezielle Oberflächenklasse erbt von der abstrakten Oberklasse *ArtefactUI*. Dadurch steht automatisch eine Referenz auf den Rahmen zur Darstellung der Bedienelemente und eine Referenz auf das Hauptfenster des *Virtual MRT* zur Verfügung. Letztere kann beispielsweise dazu benutzt werden, um die Fortschrittsanzeige zu aktualisieren.

Wir wollen die Erstellung eines neuen Artefakt-Simulators nun anhand des bereits vorhandenen Translations-Artefakts vorführen. Als Grundlage verwenden wir die beiden Vorlagenklassen *Artefact_Template* als Berechnungsklasse und *ArtefactUI_Template* als Oberflächenklasse.

Erstellen einer neuen Oberflächenklasse

Als ersten Schritt bei der Erstellung eines neuen Artefakt-Simulators sollte man sich Gedanken über die notwendigen Bedienelemente machen. Dabei muß man schon recht sorgfältig vorgehen, da zur Darstellung der Bedienelemenete nur ein Bereich von ca. 255*125 Pixel zur Verfügung steht, nämlich der Bereich in der Karteikarte *‚Artefakte‘*, der unterhalb der Auswahlbox frei ist. Abbildung 114 zeigt die Karteikarte mit den Bedienelementen zur Konfiguration des Translations-Artefakts. Anstatt einzelner Beschriftungen und Textfelder verwenden wir die im Paket *Tools* enthaltene Klasse *LabelTFLabelPanel*, die zwei Beschriftungen und ein Textfeld kapselt. Darüber hinaus bietet die Klasse die Möglichkeit, den Wert des Textfeldes mittels der rechts daneben befindlichen Knöpfe zu verändern.

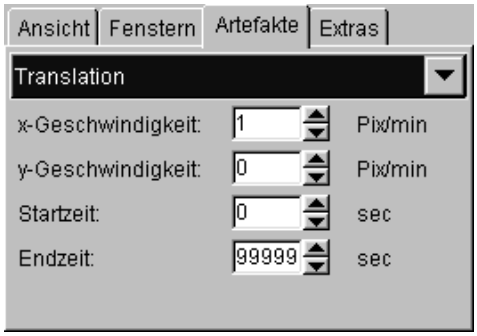

Abbildung 114: Artefakt-Karteikarte des *Virtual MRT*

Der Rumpf der Oberflächenklasse für den Translations-Simulator sieht dann in gekürzter Form folgendermaßen aus:

```
public class MotionUI extends ArtefactUI {

  // Referenz auf die Berechnungsklasse.
  Motion myManipulator;

  // Bedienelemente für die Bewegungsgeschwindigkeit in x-Richtung.
  private LabelTFLabelPanel pXSpeed  =
          new LabelTFLabelPanel("x-Geschwindigkeit:","1","Pix/min");

  // Die restlichen Bedienelemente.

  ...


  public MotionUI() {}

}
```

Als nächstes müssen die Bedienelemente in der *jbInit*-Methode eingerichtet und dem Darstellungsbereich hinzugefügt werden:

```
public void jbInit()
{
  pXSpeed.setBounds(5, 0, 255, 20);
  myPanel.add(pXSpeed);

  ...

}
```

MyPanel ist eine Referenz auf den Darstellungsbereich der Oberflächenelmente, die bereits in der Oberklasse gesetzt wird, so daß der Benutzer sich darum nicht mehr kümmern muß.

Die letzte entscheidende Methode für die Oberflächenklasse ist *createManipulator*. Die Methode wird von einer Pulssequenz aus aufgerufen. Sie erzeugt eine neue Instanz der Berechnungsklasse des Artefakt-Simulators und liefert diese an die aufrufende Pulssequenz zurück, so daß diese wiederum die

Berechnung der Simulation anstoßen kann. Darüber hinaus werden in der Methode *createManipulator* die Parameterwerte der Bedienelemente ausgelesen und der Berechnungsklasse mitgeteilt, wie das am folgenden Ausschnitt des Quelltextes ersichtlich wird.

```
public Artefact createManipulator()
{
  // Erzeugen einer neuen Instanz der Berechnungsklasse
  myManipulator = new Motion();

  // Bedienelemente auslesen und an Berechnungsklasse uebergeben
  myManipulator.setXSpeed(pXSpeed.getValue());

  ...

  return myManipulator;

}
```

Erstellen einer neuen Berechnungsklasse

Nachdem die Oberflächenklasse fertiggestellt ist, muß der Entwickler die Berechnungsvorschrift für die Artefakt-Simulation implementieren. Im Fall des Translations-Artefakts sieht das Gerüst der Berechnungsklasse folgendermaßen aus:

```
public class Motion extends Artefact {

  // Bewegungsgeschwindigkeit in x-Richtung.
  private int xSpeed;

  ...

  // Repititionszeit der Pulssequenz.
  private int trTime;


  public Motion() {}

}
```

Die Berechnungsklasse *Motion* erweitert die abstrakte Oberklasse *Artefact*. Für jeden Parameterwert der Bedienelemente wird eine Klassenvariable angelegt. Diese wird mittels entsprechender Methoden (die hier nicht angegeben sind) gesetzt, sobald von der Oberflächenklasse aus eine neue Instanz der Berechnungsklasse erzeugt wird. Darüber hinaus gibt es Klassenvariablen für wichtige Parameter der Pulssequenz, die zur Berechnung des Artefakts benötigt werden. In diesem Fall ist das die Repititionszeit. Die Sequenzparameter können nur in der folgenden Methode *calculate* ausgelesen und in den Klassenvariablen gespeichert werden, da hier der Name der aufrufenden Pulssequenz ausgelsen und eine entsprechend typumgewandelte Pulssequenz erzeugt wird. Darüber hinaus wird anhand des von der aufrufenden Pulssequenz übergebenen Parameters entschieden, welche Methode aufgerufen werden muß, um die korrekte Berechnungsvorschrift durchzuführen. Nach erfolgreicher Durchführung der Berechnungsfunktion wird das dabei erzeugte manipulierte Intensitätsbild zurückgeliefert. Woher eine Pulssequenz weiß, welche Berechnungsmethode die richtige für sie ist, wird etwas weiter unten im Abschnitt „Freischalten eines Artefakt-Simulators für eine Pulssequenz“ erläutert.

```
public int[][] calculate(String method)
{
  // Bestimmen des Namens des aufrufenden Pulssequenz
  String seqName = sequence.getClass().getName();
  seqName = seqName.substring(seqName.lastIndexOf('.')+1);

  // Casten der Sequenz und Auslesen der benoetigten Parameter
  if (seqName.compareTo("SaturationRecovery") == 0)
  {
```

```
      SaturationRecovery srSequence = (SaturationRecovery)sequence;
      trTime = srSequence.getTRTime();
    } else ...

    // Initialisieren des Rueckgabebildes
    int[][] result = null;

    // Entscheiden, welche Methode aufgerufen werden muss.
    if (method.compareTo("Motion_Std") == 0)
    {
      result = motion_Std();
    }

    return result;

  }
```

Nun wurde in dem oben stehenden Quelltextausschnitt die Methode *motion_Std* aufgerufen. Diese enthält die Standardberechnungsvorschrift für das Translations-Artefakt. Wir wollen den dafür nötigen Algorithmus hier nicht angeben. Wichtig zu wissen ist nur das innerhalb dieser Methode das Intensitätsbild, das von der Pulssequenz berechnet wurde, als *intensityimg* zur Verfügung steht. Davon können Kopien erstellt und diese beliebig manipuliert werden. Außerdem können natürlich die in den Klassenvariablen gespeicherten Parameterwerte des Artefakt-Simulators und der Pulssequenz verwendet werden.

```
public int[][] motion_Std()
{
  // Initialisieren des Ergebnisbildes
  int[][] res = new int[256][256];

  // An dieser Stelle finden die Manipulationen des Intensitätsbildes
  // statt.

  // Zurueckliefern des manipulierten Ergebnisbildes
  return res;
}
```

Natürlich muß die Methode nicht *motion_Std* heißen. Der Name ist vom Aufruf in der *calculate*-Methode und letztendlich von den Einträgen in der Datei *sequences.properties* abhängig. Dazu jedoch später noch mehr.

Anmelden eines Artefakt-Simulators am *Virtual MRT*

Genau wie die Pulssequenzen müssen auch die Artefakt-Simulatoren am *Virtual MRT* angemeldet werden, so daß sie in der Auswahlbox in der Karteikarte ‚*Artefakte*' erscheinen. Dazu sind für jeden Artefakt-Simulator zwei Einträge in die Datei *artefacts.properties* nötig. Einer davon (*Manipulator_Class_X*) definiert den Namen der Berechnungsklasse des Simulators, der andere (*Manipulator_Name_X*) legt eine Beschreibung des Simulators fest, die letztendlich in der Auswahlbox erscheint. Die aktuelle *artefacts.properties*-Datei meldet die beiden Artefakt-Simulatoren *Motion* (Translation) und *PeriodicMotion* (Periodische Bewegung) am *Virtual MRT* an:

```
Manipulator_Class_1 = "Motion"
Manipulator_Name_1  = "Translation"
Manipulator_Class_2 = "PeriodicMotion"
Manipulator_Name_2  = "Period. Bewegung"
```

Freischalten eines Artefakt-Simulators für eine Pulssequenz

Es ist nicht ausreichend, die Artefakt-Simulatoren am *Virtual MRT* anzumelden, wie dies für die Pulssequenzen der Fall ist. Vielmehr muß jeder Artefakt-Simulator für jede Pulssequenz einzeln

freigeschaltet werden. Dies geschieht durch einige Einträge in die Datei *sequences.properties*. Dabei haben die Einträge das folgende Format:

```
<Name der Artefakt-Berechnungsklasse>_<Nummer der Sequenzklasse> =
            „<Methodenname für die Artefaktberechnung>“
```

Die aktuelle *sequences.properties*-Datei sieht dann folgedermaßen aus:

```
Sequence_Class_1="SaturationRecovery"
Sequence_Name_1 ="Saturation-Recovery"
Motion_1="Motion_Std"
PeriodicMotion_1="Motion_Std"

Sequence_Class_2="InversionRecovery"
Sequence_Name_2 ="Inversion-Recovery"
Motion_2="Motion_Std"
PeriodicMotion_2="Motion_Std"

Sequence_Class_3="SpinEcho"
Sequence_Name_3 ="Spin-Echo"
Motion_3="Motion_Std"
PeriodicMotion_3="Motion_Std"

Sequence_Class_4="Flash"
Sequence_Name_4 ="GE-gespoilt"
Motion_4="Motion_Std"
PeriodicMotion_4="Motion_Std"

Sequence_Class_5="TurboSpinEcho"
Sequence_Name_5 ="Turbo-Spin-Echo"
Motion_5="Motion_TSE"
PeriodicMotion_5="Motion_TSE"
```

Durch diese Vorgehensweise ist es möglich, für unterschiedliche Pulssequenzen unterschiedliche Methoden in der Artefakt-Berechnungsklasse aufzurufen. Das ist wichtig, da die Artefaktberechnung von den Parametern der Pulssequenz abhängig sein kann. Ist dies jedoch mal nicht der Fall, so kann die gleiche Methode wiederverwendet werden.

6 Evaluierung des Projektes

Inhalt dieses Kapitels soll es sein, unser Projekt zu bewerten und von anderen Systemen abzugrenzen. Dazu vergleichen wir es zunächst mit einem Programm, daß eine ähnliche Zielsetzung verfolgt wie unser System. Desweiteren werden wir anhand einiger Beispiele die Leistungsfähigkeit unseres Systems herausstellen.

6.1 Vergleich mit anderen Systemen

Das einzige uns bekannte System, daß eine im weitesten Sinne vergleichbare Funktionalität aufweist, wie das von uns entwickelte System, ist das von der schweizerischen Firma MCS S.A. entwickelte Programm MRI IMAGE EXPERT, das uns in der Version 1.0 vorlag. Wir wollen dieses System im folgenden Kapitel kurz vorstellen und mit unserem System vergleichen.

6.1.1 MRI IMAGE EXPERT

Die Zielsetzung von MRI IMAGE EXPERT ist es, die Bildgebung eines Magnetresonanztomographen zu simulieren. Dabei kommt es den Entwicklern offensichtlich nicht auf eine realitätsnahe Gestaltung der Benutzeroberfläche an. Das Hauptfenster von MRI IMAGE EXPERT ist in Abbildung 115 dargestellt.

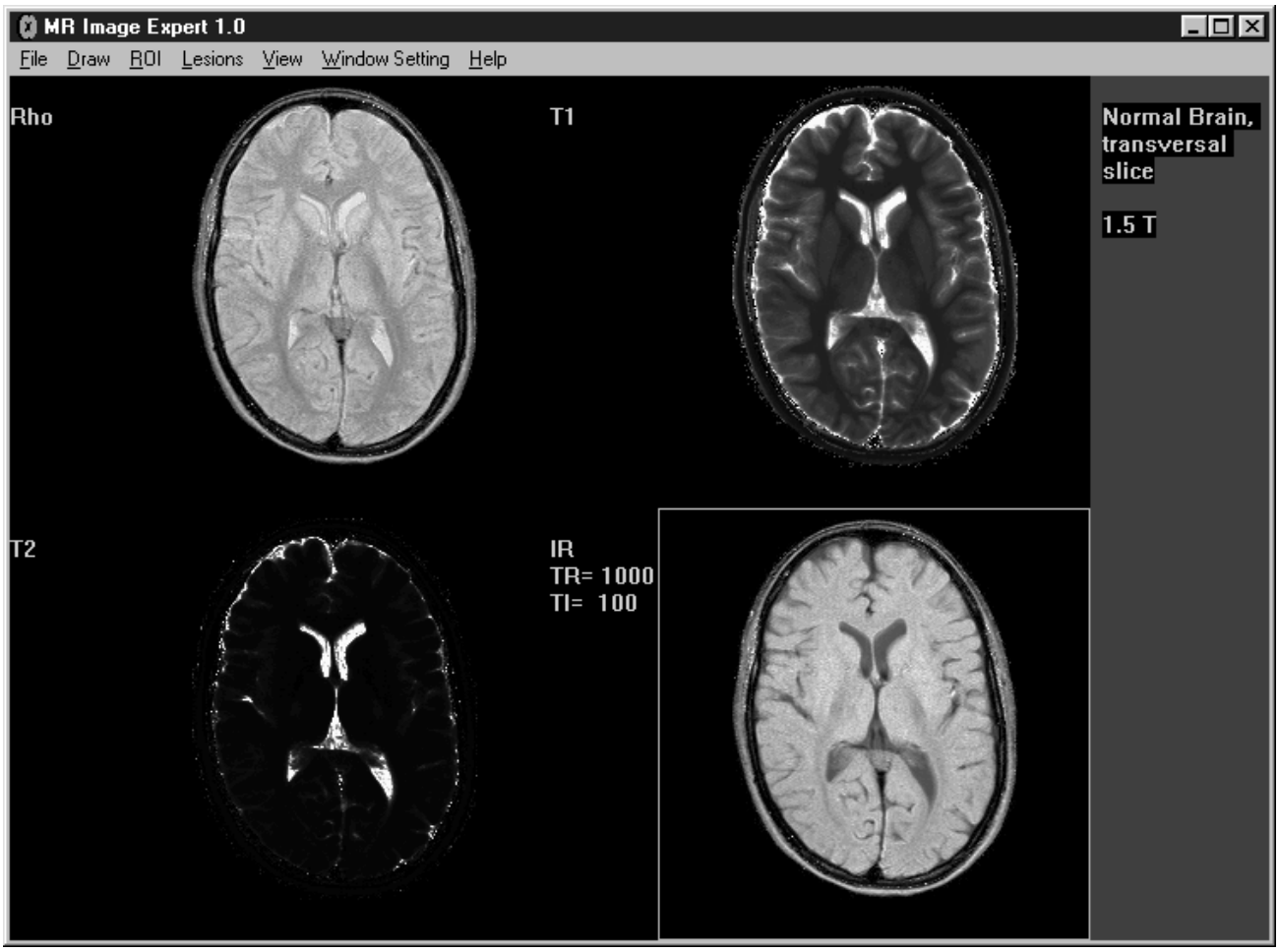

Abbildung 115: MRI IMAGE EXPERT 1.0

Zur Berechnung eines MRT-Bildes muß der Benutzer zunächst, ähnlich wie bei unseren System, einen Rohdatensatz laden. Dabei sind dem Benutzer allerdings ca. 20 Rohdatensätze fest vorgegeben (Abbildung 116). Es gibt keine Möglichkeit, selbst einen neuen Datensatz zu erstellen und diesen zu

Laden. Als nächstes wählt der Benutzer eine Pulssequenz aus, auf deren Grundlage ein neues MRT-Bild berechnet werden soll. Zunächst müssen dazu noch die Parameter für die Pulssequenz eingestellt werden (Abbildung 118). Die Parameter sind schon so voreingestellt, daß sie ein durchaus sinnvolles Bild liefern.

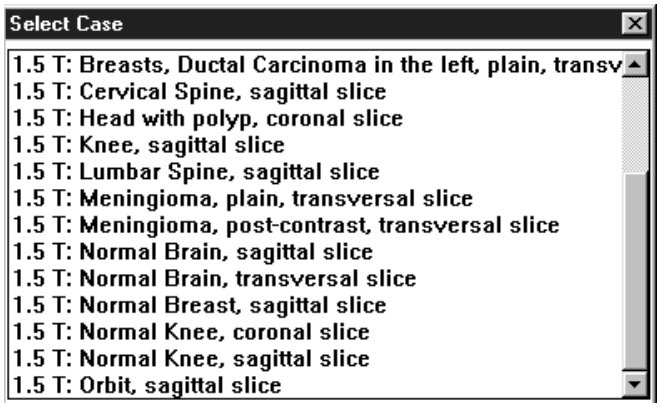

Abbildung 116: Rohdatensätze

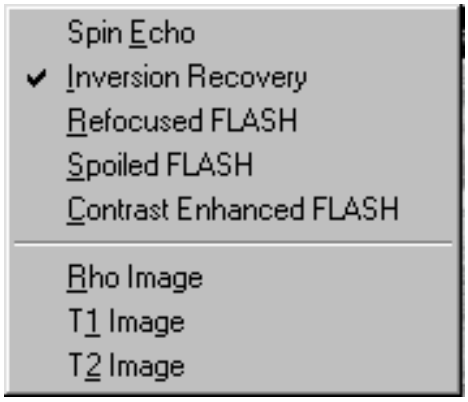

Abbildung 117: Pulssequenzen

Die Berechnung des Bildes erfolgt dann offensichtlich aufgrund der schon aus Kapitel 3.3 bekannten Formeln aus den 3 Rohdatenmatrizen (für die T_1-, T_2- und Protonendichtewert), die übrigens mit MRI IMAGE EXPERT ebenfalls dargestellt werden können (Abbildung 115).

Außer den spezifischen Parametern der eingestellten Pulssequenz können keine weiteren Einstellungen vorgenommen werden. Somit wird offensichtlich ein ideales MRT-Bild berechnet, wie es in Wirklichkeit kaum vorkommen wird. Wünschenswert wäre wenigstens die Simulation von Bildrauschen, da dieses in jedem Fall auftritt.

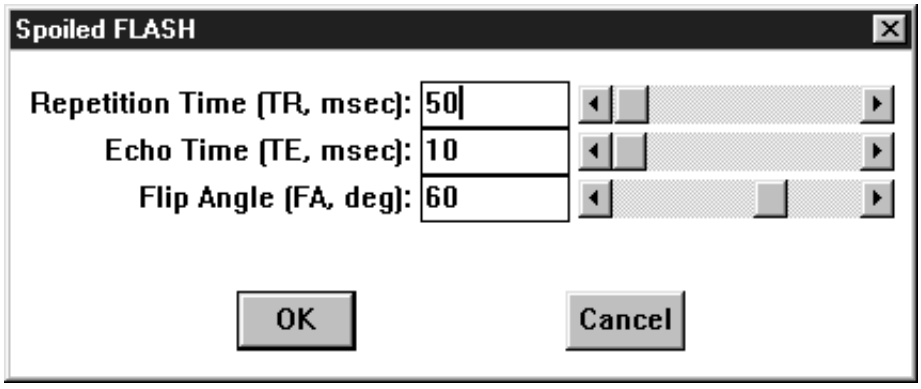

Abbildung 118: Sequenzparameterdialog

Auch von der Benutzeroberfläche zeigt sich MRI Image Expert etwas spartanisch. Es ist lediglich möglich, 4 Bilder zu berechnen. Jedes weitere Bild überschreibt eines der älteren Bilder. Darüber hinaus ist die Benutzeroberfläche nicht gerade ansprechend gestaltet und simuliert nicht im geringsten die Oberfläche einer realen Bedienkonsole. Aber wahrscheinlich war das einzige Ziel dieses Systems, dem Benutzer ein ‚Spielzeug' in die Hand zu geben, mit dem er sich anschauen kann, welche Einflüsse die Sequenzparameter auf eine MRT-Aufnahme haben.

Wir wollen in den folgenden Kapiteln einige Vorzüge unseres Systems präsentieren. Dabei kommt es uns zunächst auf die wirklich realitätsnahe Simulation der Bildgebung an. Auf die Gestaltung der Benutzeroberfläche haben wir schon in Kapitel 4.2 und 5 hingewiesen und werden darauf noch einmal in Kapitel 7.2 eingehen.

6.2 Das Referenzbild (Kontraststudie)

Das Meßwerkzeug *Virtual MRT* bietet die Möglichkeit, einen Referenzdatensatz zu laden und daraus MRT-Bilder zu berechnen. Diese Funktion ist auch dann sehr nützlich, wenn sonst kein Rohdatensatz zur Verfügung steht. Der Referenzdatensatz simuliert 6 verschiedene Substanzen, die im menschlichen Körper vorkommen. Abbildung 119 zeigt den Aufbau des Datensatzes und die Belegung der einzelnen Substanzen mit den T_1-, T_2- und PD-Werten. Anhand dieses Datensatzes kann man sehr gut einige spezielle Eigenschaften der Simulation veranschaulichen.

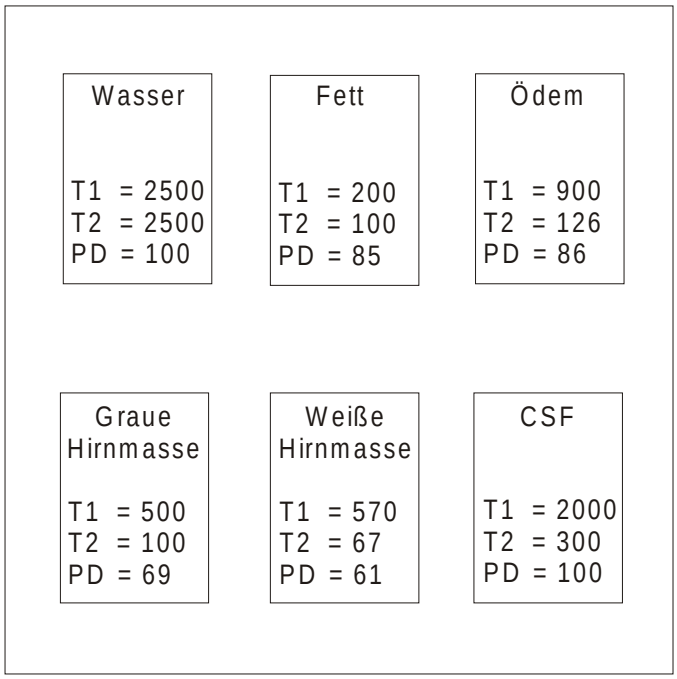

Abbildung 119: Referenzdatensatz. Relaxationszeiten in ms, Protonendichte in Prozent.

Unterdrückung von Substanzen mit einer bestimmten T_1-Zeit mit der IR-Sequenz

Als erstes soll die grundsätzlich richtige Funktionalität anhand der Inversion-Recovery-Sequenz (IR) gezeigt werden. Wie schon in Kapitel 3.3.1.2 erklärt wurde, ist es mit der IR-Sequenz möglich, eine Gruppe von Substanzen mit dem gleichen T_1-Wert zu unterdrücken, das heiß, dafür zu sorgen, daß diese Substanzen kein Signal liefern. Dieses Phänomen ist für die IR-Sequenz spezifisch und ist auf den 180 Grad Inversionspuls vor dem 90 Grad Puls zurückzuführen.

Durch den Inversionspuls wird die Besetzungszahl der beiden Energieniveaus umgekehrt, indem alle Spins um 180° an der *z*-Achse gedreht werden. Eine Auslenkung in Richtung der *xy*-Ebene tritt dabei nicht auf, weshalb die T_2-Zeit keinen Einfluß auf die Bildeigenschaften bei Veränderung der Inversionszeit hat. Nach dem Inversionspuls relaxieren die Spins mit Zeitkonstanten T_1 und erreichen nach $0{,}69 \cdot T_1$ den Nulldurchgang, bevor sie wieder in ihre Ursprungslage zurückkehren.

Der Nulldurchgang von Wasser ist auf Bild 3 (oben rechts) in Abbildung 120 dargestellt. Wasser hat eine T_1-Zeit von 2500 ms, also muß der Nulldurchgang bei $0{,}69 \cdot 2500$ ms=1725 ms stattfinden. Man sieht deutlich, daß das Wasser bei dieser Inversionszeit tatsächlich kein Signal liefert. In Bild 2 (unten links) ist eine kleinere Inversionszeit (1000 ms) und in Bild 4 (unten rechts) eine größere Inversionszeit (2500 ms) gewählt worden. In beiden Fällen liefert das Wasser ein Signal. Im ersten Fall ist das Signal eigentlich negativ, allerdings werden hier die Absolutbeträge dargestellt, was ein gängiges Verfahren ist.

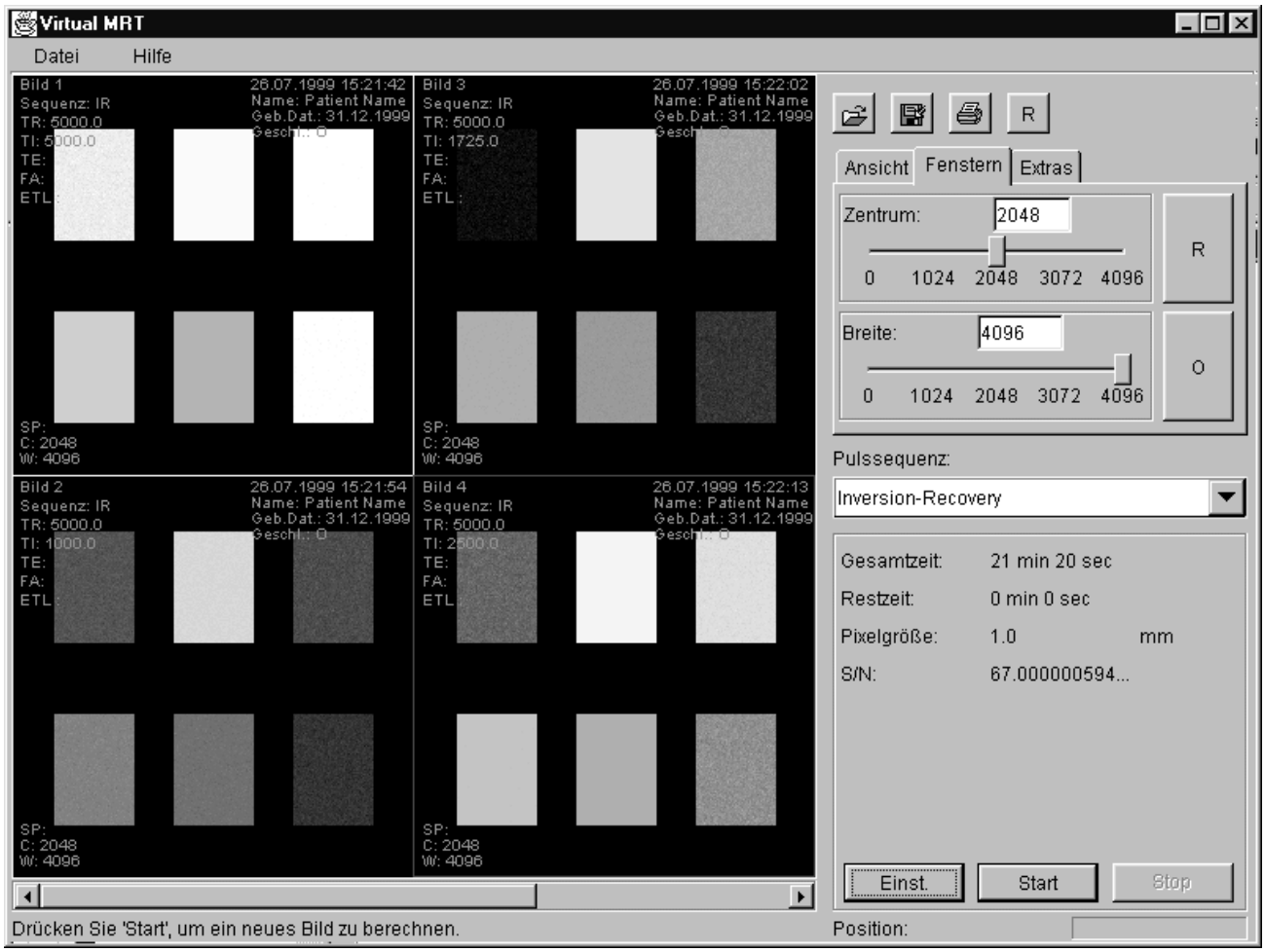

Abbildung 120: Kontraststudie für Wasser

Bildrauschen

Als nächstes soll das Bildrauschen simuliert werden. Dabei ist grundsätzlich zwischen zwei Arten von Rauschen zu unterscheiden. Die erste Form ist ein Grundrauschen. Dieses wird insbesondere von Magnetfeldinhomogenitäten und Ungenauigkeiten im analogen Teil der Signalverarbeitung hervorgerufen. Stark beeinflußt wird das Grundrauschen durch die Wahl der Meßspule. Bei Verwendung einer kleinen Spule (z.B. Kopf- oder Oberflächenspule) ist das Rauschen kleiner als bei der Wahl einer großen Spule wie der Körperspule. Dieser Effekt ist in Abbildung 121 dargestellt. Das linke Bild wurde mit Kopfspule als Einstellung erzeugt, das rechte mit der Körperspule. Man sieht deutlich, daß das rechte Bild auch im eigentlich signalfreien (schwarzen) Bereich ein deutliches Rauschen aufweist. Aber auch in signalreichen Bereichen ist das Rauschen verstärkt. Dieser Aspekt fällt unter den Begriff Signal-zu-Rausch-Verhältnis (S/R-Verhältnis). Das S/R-Verhältnis wird zudem noch durch die Wahl der Voxelgröße (voxel = volume element) beeinflußt. Je kleiner die Voxel gewählt werden, desto stärker wird das Rauschen. Das ist darauf zurückzuführen, daß pro Volumenelement weniger Protonen zur Verfügung stehen, die zum Resonanzsignal beitragen können.

Als Faustformel gilt, daß ein Volumenelement mit $5 \cdot 10^{18}$ Protonen bei Zimmertemperatur und einem Magnetfeld von 1 Tesla ein S/R-Verhältnis von ca. 1 ergibt. Aus diesem Wert wollen wir das S/R-Verhältnis von 1 mm³ Wasser bestimmen. Zunächst gilt, das 1 mol eines Stoffes $6{,}022 \cdot 10^{23}$ Teilchen des Stoffes enthält. Da ein Wassermolekül eine relative Atommasse von ca. 18 hat, enthält ein Liter Wasser $6{,}022 \cdot 10^{23}$ / 18 = $3{,}345 \cdot 10^{22}$ Wassermoleküle. Und da pro Wassermolekül 2 Protonen vorhanden sind, enthält der Liter Wasser $6{,}691 \cdot 10^{22}$ Protonen. Auf einen Milliliter umgerechnet entspricht das $6{,}691 \cdot 10^{19}$ Protonen. Zum Schluß muß dieser Wert noch durch die anfangs erwähnten $5 \cdot 10^{18}$ geteilt werden und man erhält dann ein S/R-Verhältnis von 1 ml Wasser von 13,38.

Die Größe der Voxel kann man durch die Wahl des aufzunehmenden Bereichs (field of view = FOV) und die Schichtdicke beeinflussen. Das rechte Bild in Abbildung 121 ist mit einer Schichtdicke von 1 mm berechnet worden. Das linke Bild dagegen mit einer Schichtdicke von 5 mm. Es sei aber nochmals betont, daß die Wahl der Schichtdicke keinen Einfluß auf das Rauschen im eigentlich signalfreien (schwarzen) Bereich hat.

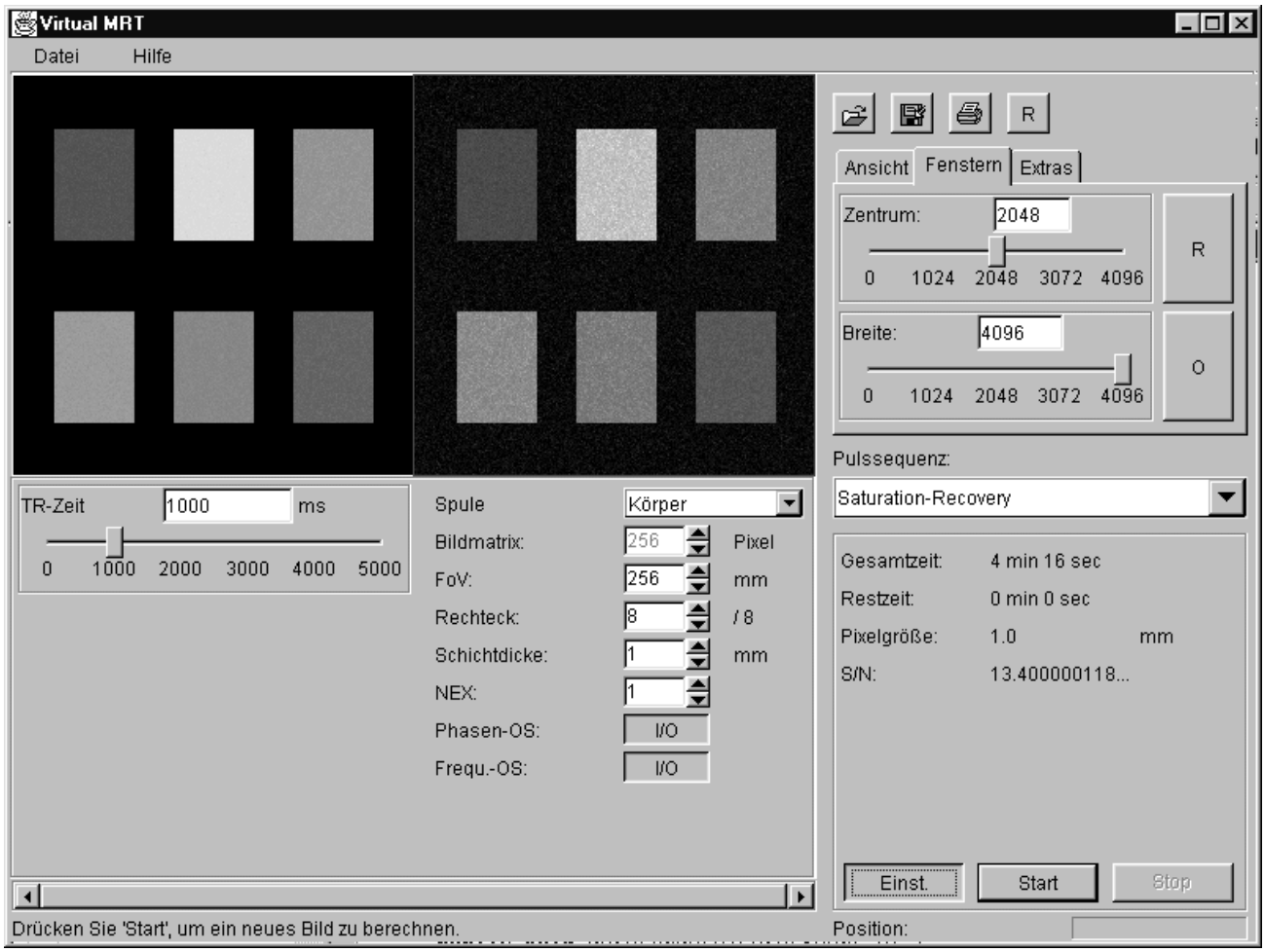

Abbildung 121: Simulation des Bildrauschens. Das linke Bild wurde mit der Kopfspule berechnet, das rechte mit der Körperspule und weist daher deutlich mehr Rauschen auf.

k-Raum-Manipulation mit der Turbo-Spin-Echo-Sequenz

Nun soll noch ein Aspekt der k-Raum-Manipulation erläutert werden. Dazu betrachten wir die Turbo-Spin-Echo-Sequenz. Wie schon in Kapitel 3.5.3 erläutert, werden bei dieser Sequenz pro Repititionszyklus mehrere Zeilen des k-Raums gleichzeitig gefüllt. Dies erreicht man, indem man pro Repititionszyklus nacheinander mehrere 180°-Impulse schaltet, so daß auch mehrere Echos meßbar werden. Allerdings sinkt mit jedem weiteren Echo auch die Signalintensität. Die Echos aus einem Repititionszyklus werden in unserer Turbo-Spin-Echo-Implementierung so verteilt, das die signalstärksten Echos in die Mitte des k-Raums aufgenommen werden, wohingegen die signalschwächeren Echos weiter am Rand des k-Raums verwendet werden. Dadurch kommt es zu einer Treppenbildung im k-Raum, die auch Artefakte nach sich zieht. Dies ist deutlich auf dem rechten Bild in Abbildung 122 zu erkennen. Dieses Bild wurde mit 8 Echos pro Repititionszyklus aufgenommen. Neben den Artefakten fällt auf, daß das Bild auch im allgemeinen deutlich unschärfer ist, als das linke Bild, welches mit nur einem Echo pro Repititionszyklus aufgenommen wurde. Das ist darauf zurückzuführen, daß der Rand des k-Raums mit den signalschwächeren Echos gefüllt ist. Und da der Rand des k-Raums die Detailinformationen des Bildes enthält, erscheint das Bild deutlich verschmiert.

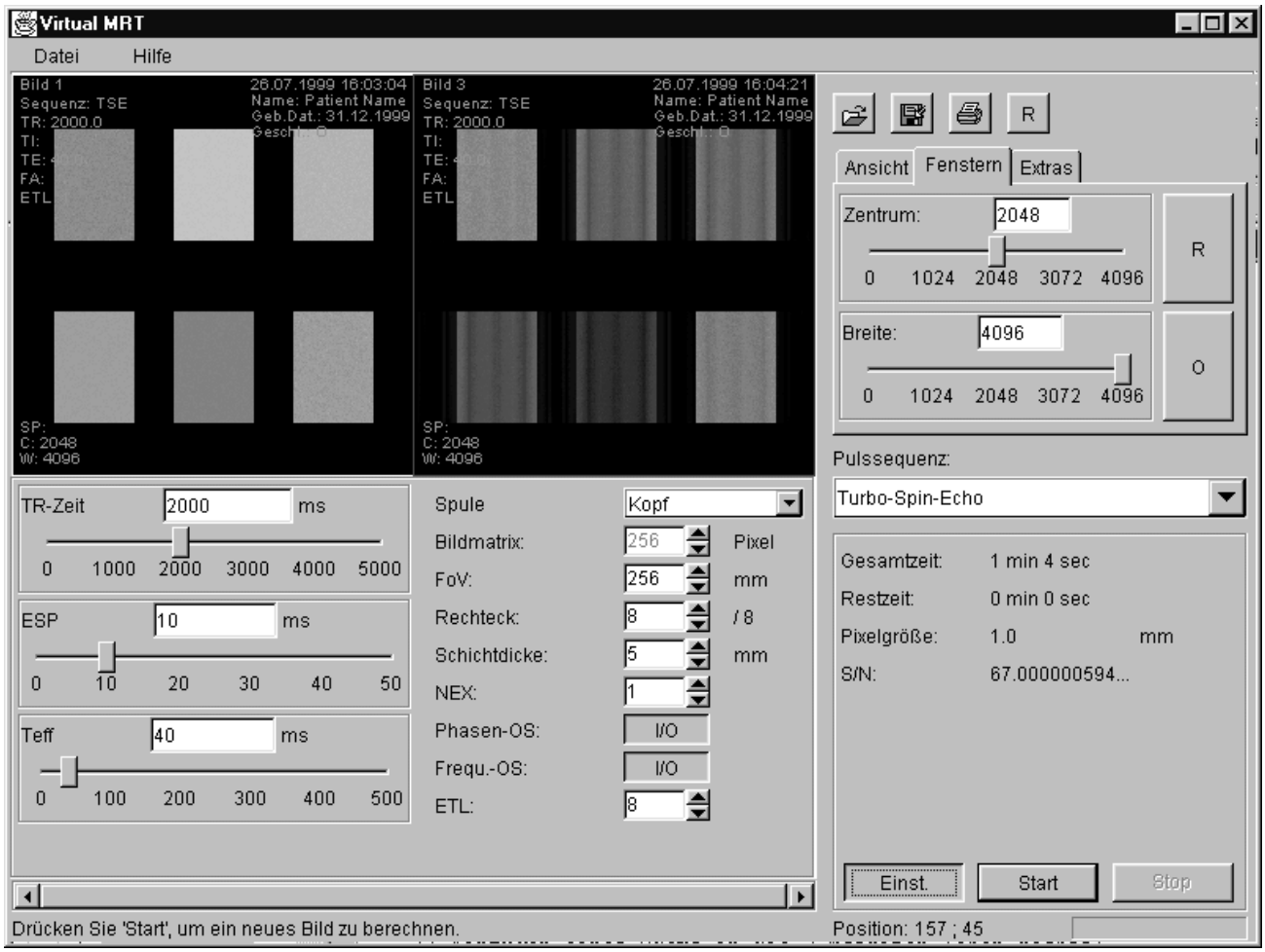

Abbildung 122: Simulation der Turbo-Spin-Echo-Sequenz. Das linke Bild ist mit einem Echo pro Repititionszyklus aufgenommen, das rechte mit 8 Echos. Es ist deutlich unschärfer und weist Artefakte auf.

Simulation von Bewegungsartefakten

Zum Schluß des Evaluierungskapitels soll noch die Möglichkeit zur Simulation von Bewegungsartefakten vorgestellt werden. Dazu verwenden wir anstatt des Referenzdatensatzes einen Rohdatensatz eines Kopfes. Der Karteikarte *‚Artefakte‘* in Abbildung 123 kann man entnehmen, daß während der Messung eine translatorische Bewegung entlang der x-Richtung mit einer Geschwindigkeit von einem Pixel pro Minute stattfindet und über die gesamte Aufnahmezeit von 4 Minuten und 16 Sekunden andauert. Insgesamt findet also während der Messung eine Verschiebung um 4,16 Pixel statt. Da die Aufnahme des k-Raum zeilenweise geschieht und die Aquisition jeder Zeile einen Repititionszyklus, also im Fall von Abbildung 123 1000 ms benötigt, liegt der Aufnahme jeder k-Raum Zeile ein etwas anderes (nämlich leicht verschobenes) Bild zugrunde. Dadurch kommt es zu den typischen Artefakten, die man im rechten Kopf von Abbildung 123 sehr gut erkennen kann.

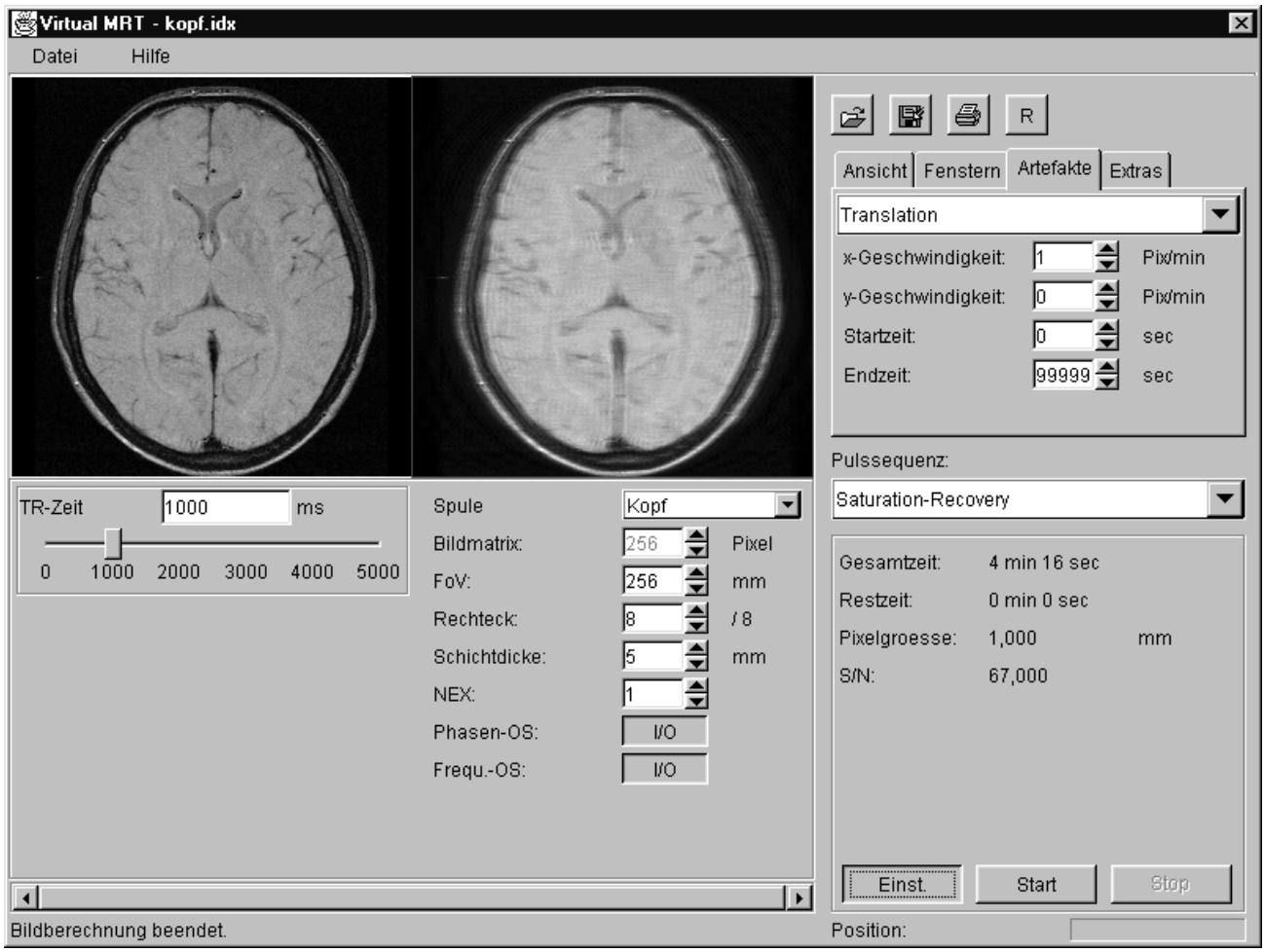

Abbildung 123: Simulation eines Bewegungsartefakts

7 Benutzerhandbuch

7.1 Installation

7.1.1 Hard- und Softwarevoraussetzungen

Da es sich beim virtuellen Tomographen um ein Bildverarbeitungssystem handelt, sollte klar sein, daß die Ansprüche an die Hardware recht hoch sind. Zwar ist unser System auch auf einem Rechner mit Pentium Prozessor lauffähig, wir empfehlen jedoch dringendst einen Pentium II mit mindestens 200 MHz und 64 MB Hauptspeicher. Sehr gut läuft das System auf einem Pentium II mit 450 MHz und 256 MB Hauptspeicher.

Um die beiliegende CD verwenden zu können, muß an den Rechner ein CD-Rom-Laufwerk angeschlossen sein. Sollen auch die Daten von der CD geladen werden, empfehlen wir ein Laufwerk mit mindestens 20-facher Geschwindigkeit.

Das notwendige Betriebsystem ist nur insofern einzuschränken, als daß eine JAVA-VM[30] dafür vorhanden sein muß. Wir haben das System unter WINDOWS95, WINDOWS98, WINDOWSNT 4.0 und SOLARIS 5.1 erfolgreich getestet.

7.1.2 Installation und Programmstart

Für einen ersten Start des virtuellen Tomographen unter einem MICROSOFT-WINDOWS Betriebssystem ist es zunächst nicht erforderlich, einen Installationsvorgang durchzuführen. Beide Teilprogramme – also das Meßwerkzeug *Virtual MRT* und das Nachbearbeitungswerkzeug *DICOM-Viewer* – können direkt von der beiliegenden CD gestartet werden.

Dazu legen Sie die CD in Ihr CD-Rom-Laufwerk ein, öffnen den WINDOWS-Explorer oder den Arbeitsplatz und wählen das Laufwerk aus, in dem sich die CD befindet. Wechseln Sie dann ins Verzeichnis *Virtual MRT* und führen Sie einen Doppelklick auf dem Symbol *VMRT* aus, um das Meßwerkzeug zu starten, auf dem Symbol *DICOMViewer*, um das Nachbearbeitungswerkzeug zu starten oder auf dem Symbol *ImageJ*, um IMAGEJ zu starten, mit dessen Hilfe Rohdatenmatrizen erzeugt und bearbeitet werden können. Eine detaillierte Bedienungsanleitung für diese Programme finden Sie in Kapitel 7.2.

Ein Testdatensatz für das Meßwerkzeug befindet sich im Verzeichnis *Daten VMRT*/*Kontraststudie*. Wählen sie zum Laden des Datensatzes im Dateiauswahldialog des Meßwerkzeuges die Datei *Kontraststudie.idx* aus. Ein weiterer Testdatensatz befindet sich im Verzeichnis *Datem VMRT/Kopf*.

Testdaten für das Nachverarbeitungswerkzeug befinden sich in den Verzeichnissen *Daten DICOM-Viewer/Kopf1* und *Daten DICOM-Viewer/Kopf2*. Das Verzeichnis *Kopf1* enthält einen Datensatz eines Kopfes mit 19 Schichten in guter Qualität. *Kopf2* enthält dagegen einen Datensatz eines Kopfes mit 200 Schichten in mäßiger Qualität, bzw. mit niedrigem Signal-zu-Rausch-Verhältnis.

Möchten Sie das System auf einem MICROSOFT-WINDOWS System auf Ihrer Festplatte installieren, um es von dort jederzeit starten zu können, so müssen sie einfach nur das Verzeichnis *Virtual MRT* auf das gewünschte Laufwerk kopieren. Da sich die JAVA-Laufzeitumgebung im gleichen Verzeichnis befindet, brauchen Sie sich um nichts weiter zu kümmern. Sie können eine Verknüpfung der beiden Symbole *VMRT* und *DICOMViewer* im Verzeichnis *VirtualMRT* auf Ihren Desktop oder in einen beliebigen anderen Verzeichnis erstellen und sich somit einen schnellen Zugriff darauf ermöglichen.

Auf einem anderen System als einem MICROSOFT-WINDOWS System können Sie den virtuellen Tomographen nicht direkt mittels der vorhandenen Verknüpfungen auf der CD starten. Sie müssen

[30] VM = Virtual Machine

sich zunächst eine JAVA-VM für Ihr System besorgen und mit deren Hilfe die Programme starten. Das genaue Vorgehen dabei ist vom Betriebsystem und der gewählten VM abhängig und kann daher von uns nicht für jedes System beschrieben werden. Zur Erleichterung stehen im Verzeichnis *Virtual MRT/JARS jar*-Dateien mir den Namen *vmrt.jar* und *dv.jar* zur Verfügung. Dieses sind Archive, die jeweils alle nötigen Dateien für die beiden Anwendungen *Virtual MRT* und *DICOM-Viewer* enthalten und von jeder VM gestartet werden können.

7.2 Bedienungsanleitung

7.2.1 Bedienung des *Virtual MRT*

Zum Programmstart genügt unter Windows ein Doppelklick auf das bei der Installation erzeugte Programmsymbol. Unter anderen Betriebssystemen kann der Aufruf auf andere Weise geschehen, darauf wollen wir hier aber nicht näher eingehen.

Wurde das Simulationswerkzeug *Virtual MRT* gestartet, funktioniert die Bedienung des Programms unter allen Betriebssystemen gleich. Die folgende Bedienungsanleitung ist also allgemeingültig. Im folgenden wollen wir einen kompletten Bedienungsablauf beschreiben, der alle Funktionen des *Virtual MRT* erklärt.

Das Hauptfenster in Abbildung 124 ist im wesentlichen in 2 Bereiche aufgeteilt. Auf der linken Seite ist die Bildfläche angeordnet. Mit dem Schiebebalken kann durch einen Bilderstapel navigiert werden. Im rechten Bereich befinden sich die Bedienelemente. In den Karteikarten oben rechts befinden sich Bedienelemente für Ansicht, Fensterung, Artefaktsimulation und Extras. In der Auswahlliste darunter kann eine Pulssequenz ausgewählt werden. In der aktuellen Version stehen Saturation-Recovery, Inversion-Recovery, Spin-Echo, Gradienten-Echo und Turbo-Spin-Echo zur Verfügung. Im unteren Teil des rechten Bereiches kann über den Knopf *‚Einst.‘* das Fenster zur Einstellung der Sequenzparameter eingeblendet werden. Mit *‚Start‘* wird die Berechnung gestartet, durch Betätigen von *‚Stop‘* kann die Berechnung jederzeit gestoppt werden.

Am unteren Rand des Fensters befindet sich ein Informationsbereich, der eine Statuszeile und eine Positionsanzeige umfaßt. In der Statuszeile werden wichtige Programmeldungen und Hinweise zum aktuellen Status ausgegeben werden. In der Positionsanzeige wird die aktuelle Position des Mauszeigers über der Bildfläche angezeigt. Rechts daneben befindet sich eine Fortschrittsanzeige, die beim Berechnen einer Pulssequenz den Fortschritt der Berechnung anzeigt.

Die Menüleiste am oberen Rand des Hauptfensters enthält ein Menü *‚Datei’* und ein Menü *‚Hilfe‘*. Die Menüeinträge in *‚Datei‘* ermöglichen das Laden von Parameterbildern und den Export von in der Simulation erzeugten DICOM-Bildern. Außerdem kann das aktuelle Bild gedruckt und das Programm beendet werden.

Die Option *‚Info‘* im Menü *‚Hilfe‘* zeigt Informationen über Autor und Version des Programms an.

Abgesehen von der Option *‚Info‘* können alle Funktionen auch über die Bedienelemente des Kontrollbereichs ausgewählt werden.

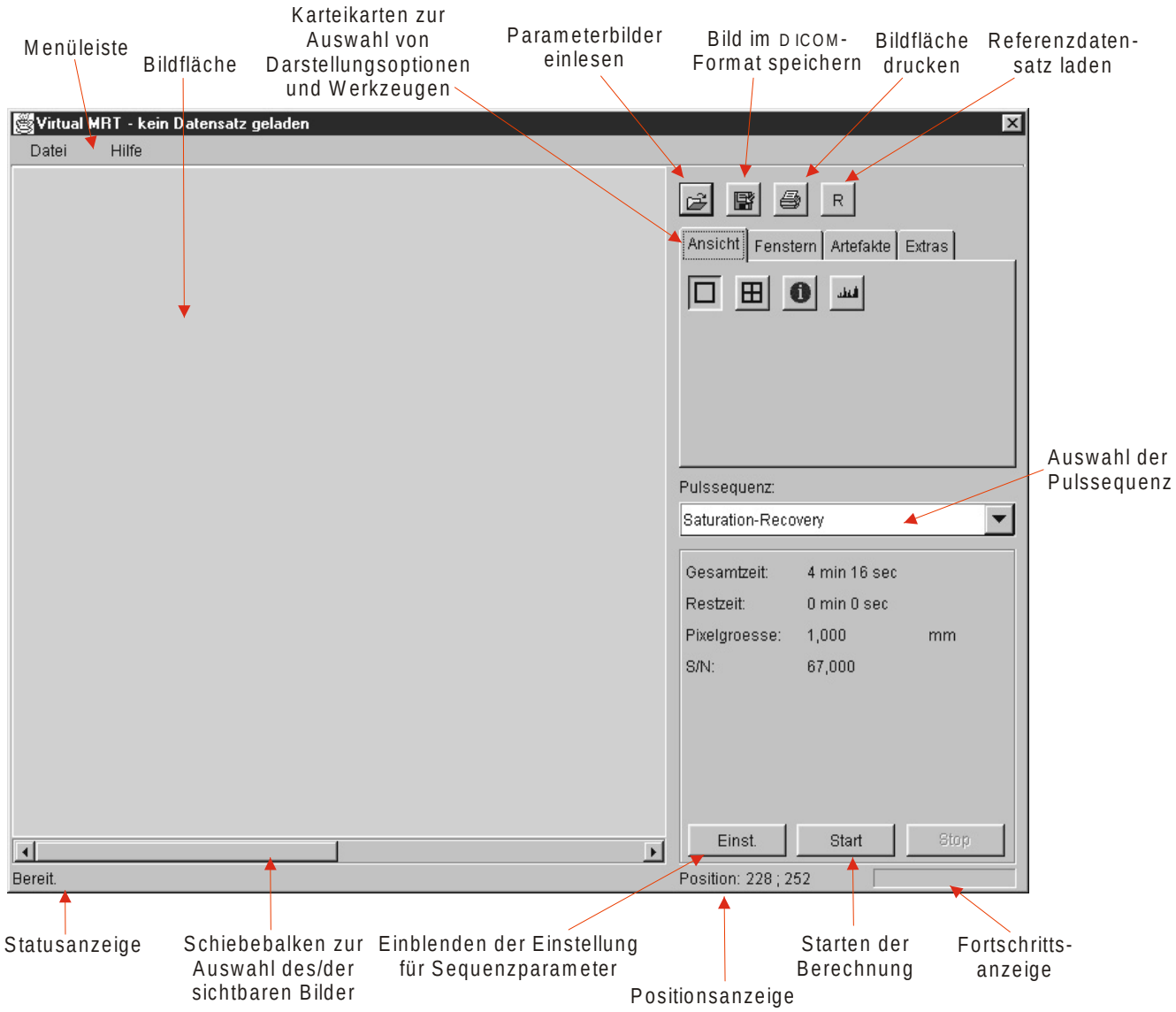

Abbildung 124: Hauptfenster des Simulationswerkzeugs *Virtual MRT* nach dem Programmstart

Allgemeine Funktionen

Als erstes werden wir die Funktion der Bedienelemente oberhalb der Karteikarten erläutern. Über diese Bedienelemente können von links nach rechts die Parameterbilder geladen werden, ein erzeugtes Bild im DICOM-Format exportiert werden, das aktuelle Bild gedruckt und ein Referenzdatensatz geladen werden.

Laden der Parameterbilder

Bevor der Anwender das Programm *Virtual MRT* zur Simulation eines Magnetresonanztomographen einsetzen kann, ist es im ersten Schritt nötig, einen Datensatz mit Parameterbildern einzulesen. Diese Rohdatensätze müssen im DICOM-Format (siehe auch Kapitel 5.3.1) vorliegen, so wie es zum Beispiel im IMAGEJ-*PlugIn* zur Berechnung der Rohdatensätze (siehe auch Kapitel 7.2.3) erzeugt wird.

Zum Laden eines Satzes von Parameterbildern betätigt der Benutzer den -Knopf (siehe Abbildung 135). Daraufhin öffnet sich dann der in Abbildung 136 dargestellte Dateiauswahldialog. Mit Hilfe dieses Dialoges kann der Benutzer durch die Verzeichnisstruktur seines Dateisystems navigieren und die zu ladende Bildserie selektieren. Die Zuordnung der Parameterbilder erfolgt hier über die *idx*-Datei. Durch Betätigung des Knopfes ‚*Öffnen*‘ wird der ausgewählte Parameterdatensatz geladen.

Abbildung 125: Dateiauswahldialog des *Virtual MRT* (hier für MS-Windows)

Laden des Referenzdatensatzes

Mit dem ‚*R*'-Knopf wird der fest im Programm gespeicherte Referenzdatensatz geladen. Dieser Datensatz kann auch verwendet werden, wenn kein Parameterbilddatensatz vorhanden ist. Mit diesem Referenzdatensatz können auch die Beispiele aus Kapitel 6.2 nachvollzogen werden.

Navigieren durch den Bildstapel und Selektieren eines Bildes

Wenn mehr als ein Bild erzeugt wurde, kann man mit Hilfe des horizontalen Schiebebalkens unterhalb der Bildfläche durch den Bildstapel navigieren. Durch einfaches Klicken mit einer beliebigen Maustaste auf dem gerade dargestellten Bild, wird dieses selektiert. Dieser Zustand wird dem Benutzer durch eine rote Umrandung des Bildes angezeigt. Die Selektion eines Bildes ist für einige später beschriebene Funktionen wichtig. Funktionen wie Speichern und Drucken beziehen sich immer auf das aktuell selektierte Bild.

Speichern eines Bildes

Sobald ein Bild erzeugt wurde und auf der Bildfläche dargestellt wird, kann dieses wie eben beschrieben selektiert und anschließend gespeichert werden. Das scheint zu diesem Zeitpunkt noch nicht sehr sinnvoll zu sein, da wir bisher nur Parameterbilder geladen und noch keine Bilder erzeugt haben. Später werden wir jedoch auch beschreiben, wie man aus den Parameterbilder per Simulation Bilder erzeugen kann. Diese Bilder können dann gespeichert werden. Dazu muß der Benutzer den -Knopf betätigen. Der Dialog, der daraufhin erscheint, hat das gleiche Aussehen wie der in Abbildung 125 dargestellte Dateiauswahldialog, allerdings unterscheiden sich Fenstername und Beschriftungen der Knöpfe. Der Benutzer kann das Zielverzeichnis und die Zieldatei auswählen oder aber den Namen einer neu zu erstellenden Datei angeben. Nach Betätigen des Knopfes ‚*Save*' wird das erzeugte Bild im DICOM-META-Format gespeichert und kann mit jedem DICOM-fähigen Programm nachbearbeitet werden.

Drucken des Bildflächeninhalts

Der aktuell sichbare Zeichenflächeninhalt kann auf einem für das System eingerichteten Drucker ausgegeben werden. Zu diesem Zweck muß der Benutzer den -Knopf betätigen. Abbildung 126 zeigt den Dialog für ein MS Windows-System. Der Benutzer kann jetzt den Drucker wählen und den Druckvorgang durch Betätigen des ‚*OK*'-Knopfes starten.

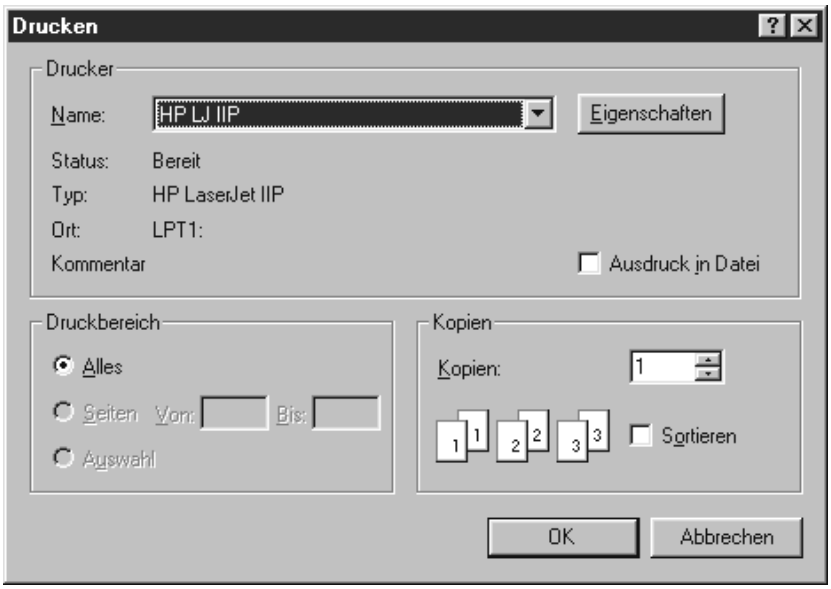

Abbildung 126: MS WINDOWS Druckdialog

Funktionen der Karteikarte ‚*Ansicht*‘

Die Karteikarte ‚*Ansicht*‘ (Abbildung 127) ermöglicht das Umschalten zwischen Einzelbilddarstellung und Darstellung von vier Bildern, Ein- und Ausblenden von Bildinformationen, Histogrammanzeige und k-Raum-Anzeige.

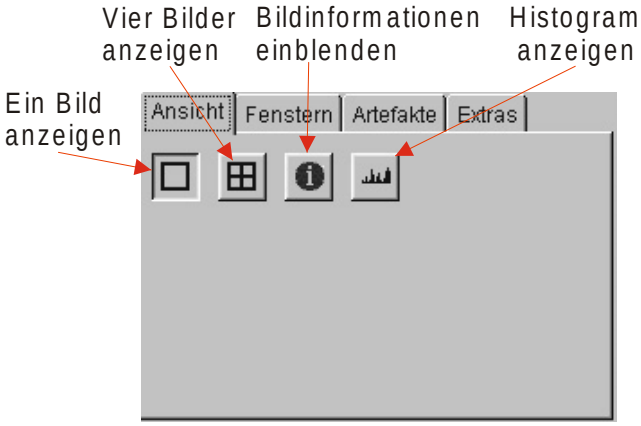

Abbildung 127: Die Karteikarte ‚*Ansicht*‘

1 bzw. 4 Bilder gleichzeitig darstellen

Standardmäßig wird ein Bild auf der Bildfläche dargestellt. Die mit dem *Virtual MRT* erzeugten DICOM-Bilder haben eine Auflösung von 256*256 Punkten, daher werden diese skaliert und auf einer Fläche von 512*512 dargestellt. Durch Betätigung des ⊞-Knopfes können auch vier Bilder gleichzeitig dargestellt werden. Dann wird jedes Bild auf einer Fläche von 256*256 Punkten dargestellt. Abbildung 128 zeigt das Hauptfenster, nachdem die Anzeige auf vier Bilder umgeschaltet wurde.

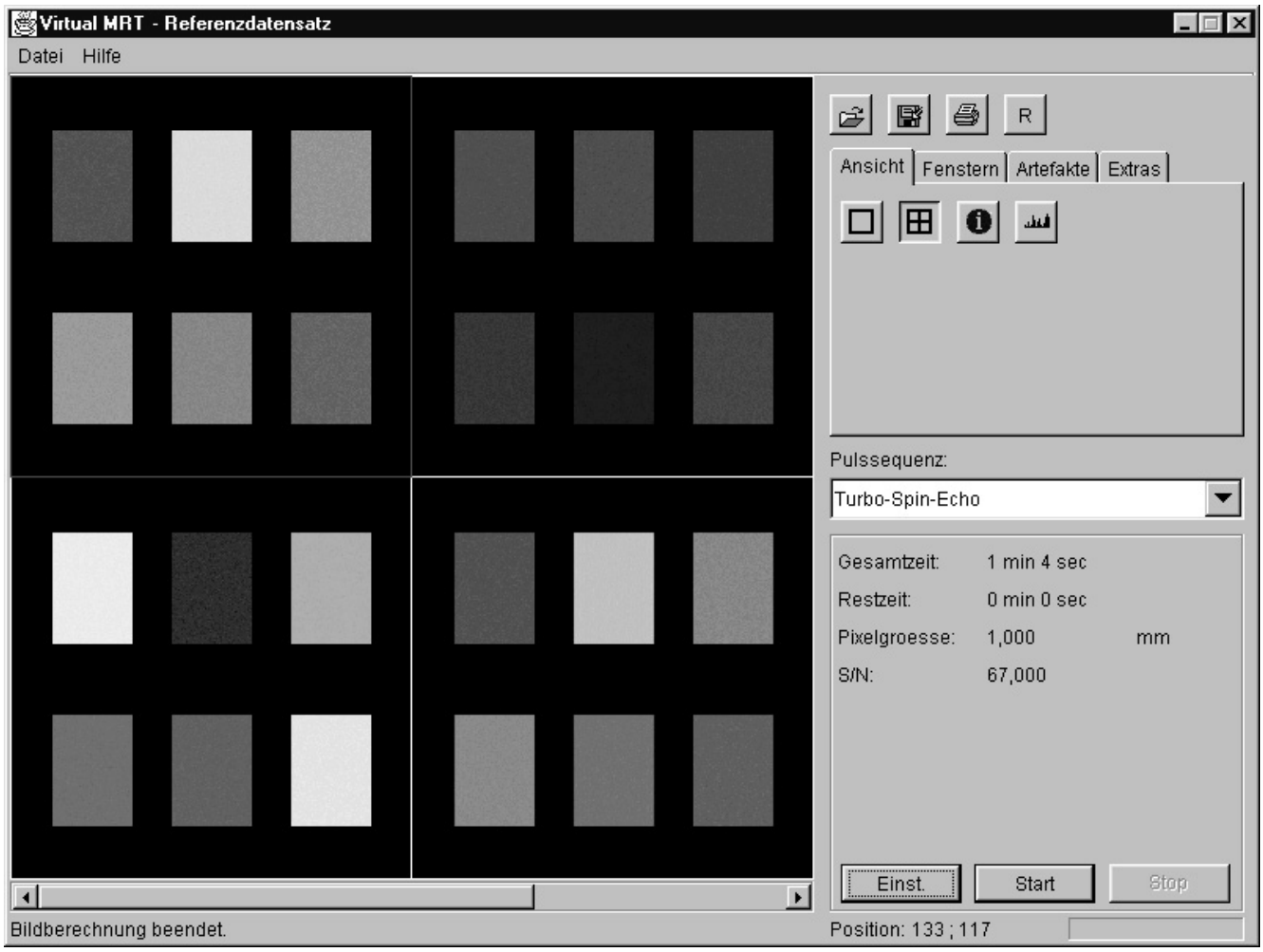

Abbildung 128: Darstellung von vier Bildern im *Virtual MRT*

Bildbeschriftungen ein-/ausblenden

Neben den Bildern können auch zusätzliche Bildinformationen auf der Bildfläche dargestellt werden. Diese Funktion wird durch Drücken des ⓘ-Knopfes ausgewählt. Auf die gleiche Weise ist es möglich, die Bildbeschriftung wieder auszuschalten. Oben links werden die sequenzspezifischen Parameter wie Pulssequenzname (*Sequenz*), Repititionszeit (*TR*), Inversionszeit (*TI*), Echozeit (*TE*), Flipwinkel (*FA*) und die Anzahl der Echos pro Repititionszyklus (*ETL*) dargestellt. Oben rechts wird das Aufnahmedatum des Bildes, der Patientenname (*Name*), sein Geburtsdatum (*Geb.Dat.*) und sein Geschlecht (*Geschl.*) angezeigt. Unten links findet man die Schichtposition (*SP*), die im *Virtual MRT* keine Bedeutung hat und die aktuellen Werte der Fensterung (*C/W*). Der *Virtual MRT* mit eingeblendeten Bildinformation ist in Abbildung 130 dargestellt.

Histogramm anzeigen

Das Histogramm des aktuell selektierten Bildes kann der Benutzer sich durch Betätigen des ▟ Knopfes anzeigen lassen. In einem separaten Fenster wird die Grauwertverteilung dargestellt und kann zu beliebig vielen Bildern gleichzeitig geöffnet werden (Abbildung 129). Daraus ergibt sich eine Vergleichsmöglichkeit für den Benutzer.

Um eine bessere Skalierung für das Histogramm zu bekommen werden Grauwerte, die sehr häufig vorkommen nur als rote Balken dargestellt. Sie werden nicht in voller Höhe dargestellt, da die Balken der restlichen Grauwerte sonst kaum noch zu erkennen wären.

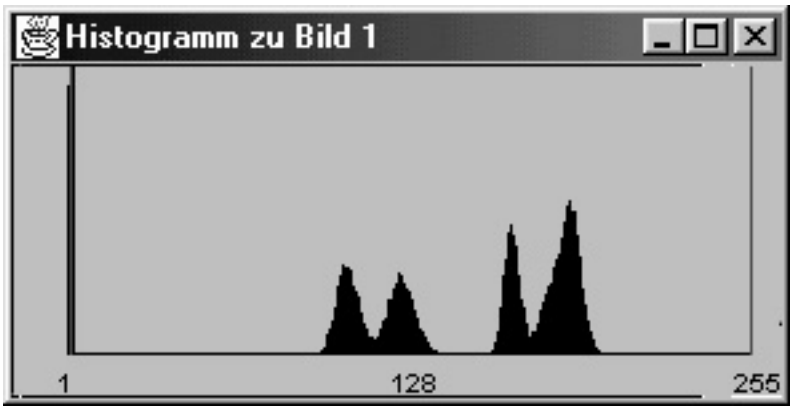

Abbildung 129: Histogrammfenster des *Virtual MRT*

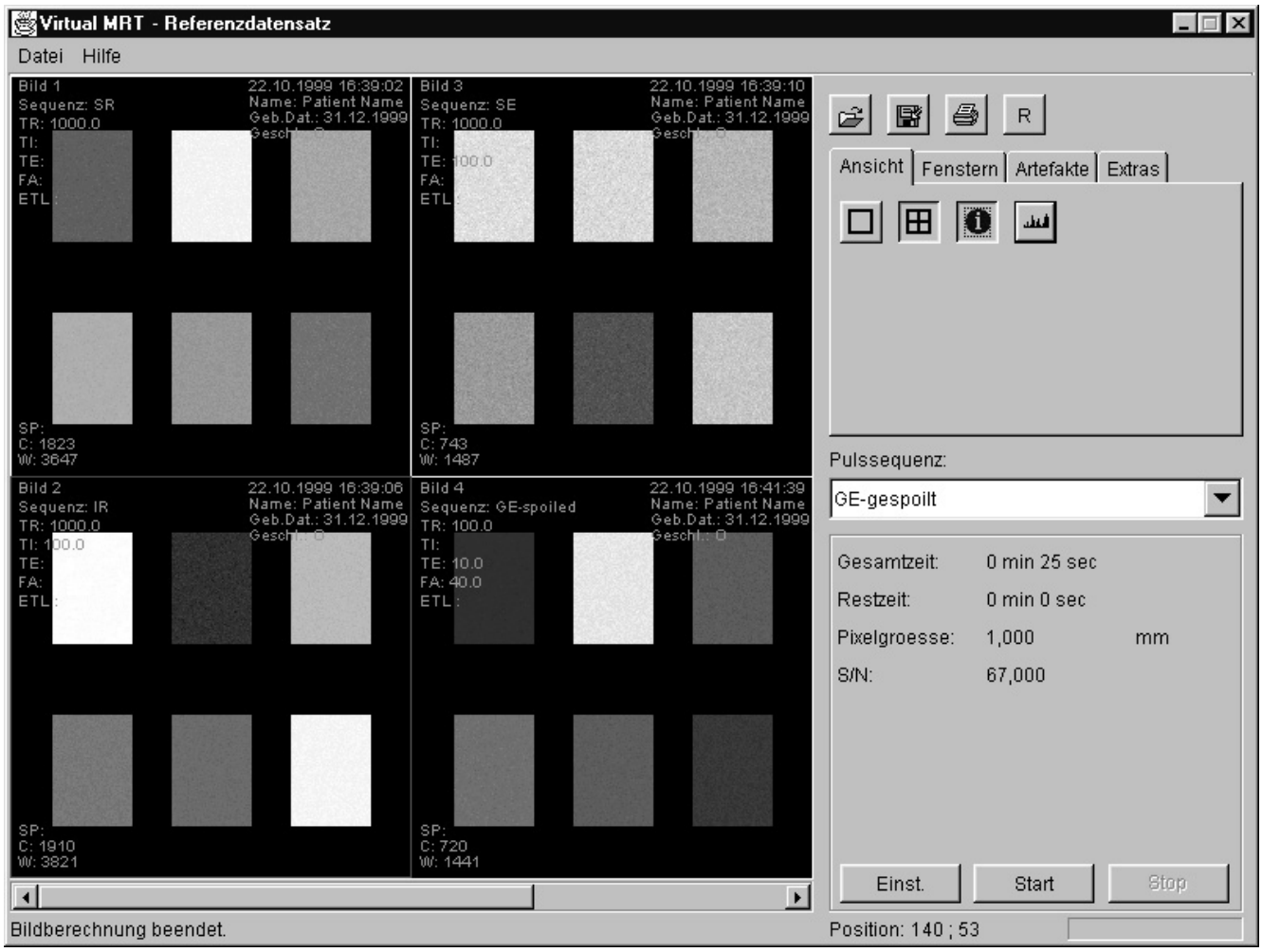

Abbildung 130: Einblendung der Bildinformationen. Man sieht eine Saturation-Recovery-, eine Inversion-Recovery-, eine Spin-Echo- und eine Gradienten-Echo-Sequenz mit den dazugehörigen Parametern.

Funktionen der Karteikarte ‚*Fenstern*‘

Mit der Karteikarte ‚Fenstern‘ kann der Benutzer den Ausschnitt aus den 12-Bit-Daten wählen, der in 8-Bit-Graustufen auf dem Bildschirm angezeigt wird (vgl dazu Kapitel 2 und Abbildung 12).

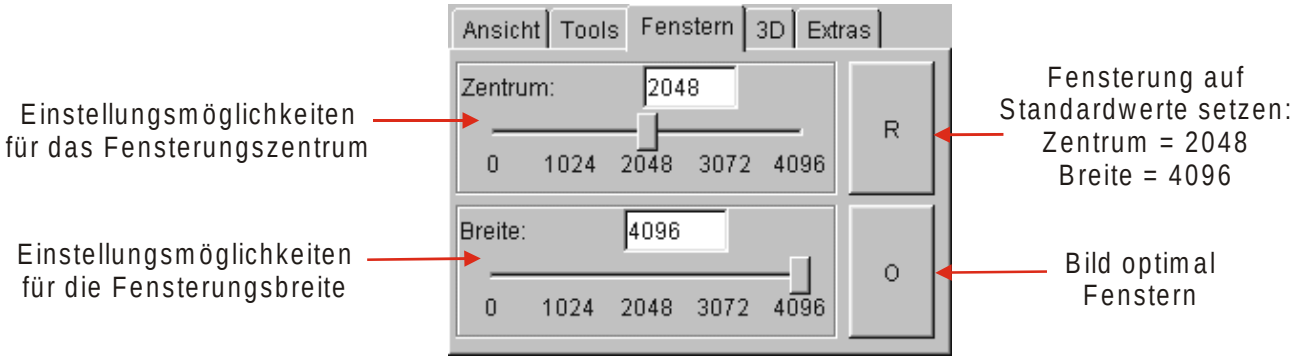

Abbildung 131: Karteikarte ‚*Fenstern*' des *Virtual MRT*

Abbildung 131 zeigt die Karteikarte ‚*Fenstern*'. Die Einstellung kann auf Standardwerte gesetzt werden mit ‚*R*', auf optimale Fensterung eingestellt werden mit ‚*O*' oder vom Benutzer manuell festgelegt werden über die Einstellungen ‚*Zentrum*' und ‚*Breite*'.

Zurücksetzen der Fensterung auf Standardwerte

Mit dem mit ‚*R*' beschrifteten Knopfes wird die Fensterung des selektierten Bildes auf Standardwerte zurückgesetzt. Standardwerte sind für das Zentrum 2048 und für die Breite 4096. Mit dieser Einstellung werden alle möglichen 12-Bit-Werte auch im Graustufenbild dargestellt. Es kann aber sein, daß der Bildkontrast sehr schlecht ist.

Optimale Fensterung

Durch Betätigen des mit ‚*O*' beschriftete Knopfes erfolgt eine optimale Fensterung des selektierten Bildes. Dazu wird zunächst der kleinste Grauwert *min* und größte Grauwerte *max* im Bild bestimmt. Die Breite des Grauwertfensters ergibt sich dann zu *max-min* und das Zentrum zu *min+(max-min)/2*. Somit wird der Bildkontrast optimiert.

Manuelle Fensterung

Mit den Schiebereglern für Zentrum und Breite der Fensterung oder über die zugehörigen Textfelder kann der Benutzer das selektierte Bild manuell fenstern. Entweder kann der Benutzer die linke Maustaste auf einem Schieberegler drücken und mit gedrückter Maustaste verschieben oder direkt Werte in die Textfelder eingeben.

Die Fensterung kann aber noch auf andere Weise geändert werden:

Durch Drücken und gedrückt halten einer beliebigen Maustaste über der Bildfläche wird die Fensterungs-Funktion aktiviert, was durch einen geänderten Mauszeiger verdeutlicht wird. Bewegungen der Maus nach oben oder unten bei gedrückter Taste ändert das Zentrum der Fensterung, Bewegungen der Maus nach rechts oder links variieren die Breite der Fensterung.

Funktionen der Karteikarte ‚*Artefakte* '

Hier kann der Benutzer zusätzliche Störeinflüsse simulieren, die nicht aus der Messung entstehen. Zur Zeit kann hier entweder kein Artefakt oder die Simulation von Bewegungen gewählt werden. Dabei wird zwischen einer einfachen Translation und einer periodischen Bewegung unterschieden. Wird die Simulation einer Translation ausgewählt, erscheinen die Bedienelemente zur Eingabe von Parametern für diese Simulation. Abbildung 132 zeigt die Auswahl des Artefaktes ‚*Translation*'. Die Parameter können durch Eingabe in die Textfelder eingestellt werden.

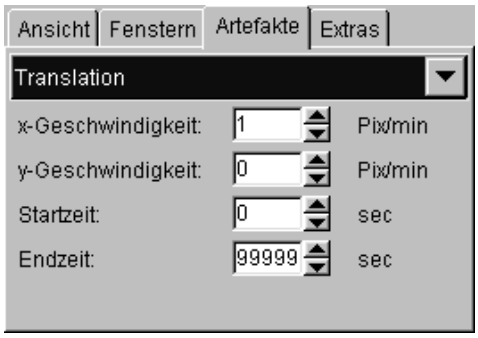

Abbildung 132: Die Karteikarte *‚Artefakte‘*

Funktionen der Karteikarte *‚Extras‘*

Die Karteikarte ‚Extras‘ aus Abbildung 133 bietet die Möglichkeit, die Position des nächsten Bildes festzulegen und den Simulationszeitfaktor einzustellen.

Abbildung 133: Die Karteikarte *‚Extras‘*

Position für ein neues Bild festlegen

Mit Hilfe der Funktion *‚Nächstes Bild positionieren‘* kann der Benutzer festlegen, an welcher Position das nächste berechnete Bild dargestellt werden soll. Mit der Option *‚Selektion‘* wird das berechnete Bild an die aktuell selektierte Stelle eingefügt, ein dort eventuell vorhandenes Bild wird dabei überschrieben. Die Option *‚Freie Stelle‘* fügt dagegen ein neues Bild an der nächsten freien Stelle im Bildstapel ein. Die Standardeinstellung ist *‚Freie Stelle‘*.

Simulationszeitfaktor festlegen

Diese Einstellung soll dem Benutzer einen Eindruck von der realen Meßdauer einer Pulssequenz geben. Die Einstellung ‚Simulationszeitfaktor‘ bestimmt, wie lange die simulierte Messung dauert. Dabei wird der Simulationszeitfaktor in Prozent von der realen Meßdauer angegeben. Eine Einstellung von 50% bedeutet, daß die simulierte Messung halb so lange wie die reale Messung dauert. Eine Einstellung von 0% errechnet das Ergebnis so schnell wie möglich, die Erzeugung des Bildes benötigt dann nur die reine Rechenzeit.

Es kann aber insbesondere bei den schnelle Pulssequenzen vorkommen, daß die Rechenzeit länger ist als die reale Messung. Vor allem bei der Turbo-Spin-Echo, für die bei der Simulation mehrere Fouriertransformationen berechnet werden müssen, ist die Dauer der Simulation unter Umständen länger als die der realen Messung. Die optional wählbaren Artefaktsimulationen verbrauchen zusätzlich Rechenzeit. Für die langsameren Meßsequenzen sollte dieser Faktor nicht zu hoch eingestellt werden, da der Benutzer sonst sehr lange auf das Ergebnis der Berechnung warten muß.

Wahl der Pulssequenz und Einstellen der Meßparameter

Die wichtigsten Einstellungen unseres Simulationswerkzeugs sind die Auswahl und die Eingabe von Parametern der Pulssequenzen.

Diese Einstellungen wollen wir im folgenden genau beschreiben. Die Auswahl der Pulssequenz erfolgt über die Auswahllisten unter dem Text ‚*Pulssequenz*'. Durch Auswahl des Knopfes ‚*Einst.*' wird ähnlich wie im realen Gerät ein Dialog in der unteren Hälfte des Bildbereiches eingeblendet.

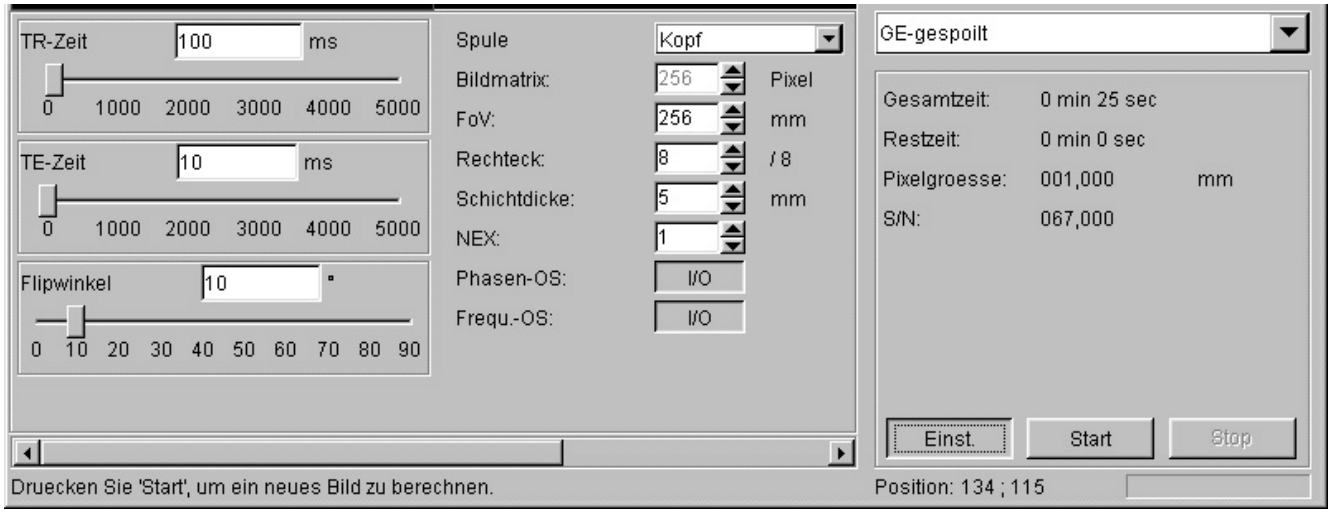

Abbildung 134: Dialog für die Sequenzparameter

Abbildung 134 zeigt exemplarisch die Einstellungen für die Pulssequenz ‚*GE-gespoilt*'. Auf der rechten Seite im Parameterfenster sind die sequenzunspezifischen Einstellungen zu sehen, auf der linken die sequenzspezifischen.

Sequenzunspezifische Parameter festlegen

Folgende sequenzunspezifischen Einstellungen sind für jede Sequenz vorhanden:

- Spule: Hier kann ‚*Kopf*' oder ‚*Körper*' gewählt werden. Wird die größere Körperspule ausgewählt, wird wie im realen MRT das Rauschen stärker.
- Bildmatrix: In der Simulation werden feste Matrizen mit 256 Pixeln Größe verwendet. Dieser Wert ist daher nur informativ und kann nicht verändert werden.
- FoV: Hier wird das ‚*Field of View*' eingestellt, also die Größe des Aufnahmebereichs.
- Rechteck: Verhältnis der Bildbreite zur Bildhöhe.
- Schichtdicke: Legt die Schichtdicke fest. Eine dünnere Schichtdicke führt zu einem schlechteren Signal-zu-Rausch-Verhältnis und damit zu einer schlechteren Bildqualität.
- NEX: ‚Number of excitations' legt fest, wie oft eine Messung durchgeführt wird. Eine Erhöhung dieses Wertes verlängert die Messung erheblich, da die Messung *NEX*-mal durchgeführt wird. Allerdings verbessert sich mit einem höheren Wert das Signal-zu-Rausch-Verhältnis und damit auch die Bildqualität.
- Phasen-OS und Frequenz-OS: Abschalten von ‚*Oversampling*' kann zu Einfaltungsartefakten führen.

Alle Werte können entweder direkt über das Textfeld eingegeben werden oder durch Betätigen der ▼- und ▲-Knöpfe mit der Maus eingestellt werden.

Sequenzspezifische Parameter festlegen

Über die Bedienelemente im linken Teil können die sequenzspezifischen Parameter festgelegt werden. Bei der in Abbildung 134 gezeigten Gradienten-Echo-Sequenz sind dies Repititionszeit, Echozeit und Flipwinkel. Diese Werte können entweder durch Ziehen der Schieberegler mit der Maus oder direkt durch Eingabe von Werten in die Textfelder geändert werden. Folgende sequenzspezifischen Parameter können für die einzelnen Meßsequenzen eingestellt werden:

- Saturation-Recovery: Repitionszeit
- Inversion-Recover: Repitionszeit, Inversionszeit
- Spin-Echo: Repititionszeit, Echozeit
- Gradienten-Echo: Repititionszeit, Echozeit, Flipwinkel
- Turbo-Spin-Echo: Repititionszeit, Echoabstand (ESP), effektive Echozeit und Anzahl Echos (ETL). Bei dieser Sequenz ist der Sequenzspezifische Parameter ETL aus Platzgründen ausnahmsweise unten auf der rechten Seite im Dialog angeordnet.

Starten und Stoppen von Bildberechnungen, Informationsbereich

In Abbildung 134 sieht man neben den Bedienelementen zur Einstellung der Pulssequenzparameter auf der rechten Seite den Informationsbereich für Pulssequenzen. Hier kann durch Betätigung des ‚*Start*'-Knopfes nach Eingabe der Parameter die simulierte Bilderzeugung gestartet werden. Wird aktuell eine Berechnung durchgeführt, ist auch der ‚*Stop*'-Knopf selektierbar, um die Berechnung jederzeit abzubrechen.

Desweiteren werden hier Informationen über die Pulssequenz und den Fortgang der Berechnung angezeigt. Das sind im einzelnen:

- Gesamtdauer: Die Zeit, die für die gewählte Pulssequenz im realen MRT insgesamt nötig wäre.
- Restzeit: Die Restzeit, die für eine reale Messung noch benötigt würde.
- Pixelgroesse: Größe eines einzelnen Pixels im Bild.
- S/N: Das Signal-zu-Rausch-Verhältnis.

7.2.2 Bedienung des *DICOM-Viewers*

Der *DICOM-Viewer* liegt in unserem Gesamtsystem als eigenständiges Programm vor. Es ist zwar durchaus sinnvoll, ihn nach der Erzeugung von DICOM-Bildern mittels des *Virtual MRT* zu verwenden um die erzeugten Bilder nachzuverarbeiten, jedoch können auch beliebige andere Bilder im DICOM-Format geladen, betrachtet und verarbeitet werden. Da der *DICOM-Viewer* also auch unabhängig vom *Virtual MRT* zum Einsatz kommen kann, wollen wir seine Bedienung hier im einzelnen erläutern, obwohl sich einige Überschneidungen mit der Bedienung des *Virtual MRT* ergeben.

Zum Programmstart genügt unter Windows ein Doppelklick auf das bei der Installation erzeugte Programmsymbol. Unter anderen Betriebssystemen kann es möglich sein, daß der Programmstart auf andere Weise geschehen muß, z.B. durch einen Kommandozeilenbefehl. Auf diese Varianten wollen wir hier jedoch nicht eingehen.

Wurde der *DICOM-Viewer* gestartet, gestaltet sich die Bedienung des Programms unter allen Betriebssystemen gleich. Die folgende Bedienungsanleitung ist also allgemein gültig. Wir wollen nun einen kompletten Bedienungsablauf vorführen, der alle Funktionen des *DICOM-Viewers* beinhaltet.

Nach dem Programmstart zeigt sich der *DICOM-Viewer* dem Benutzer durch das in Abbildung 135 dargestellten Hauptfenster.

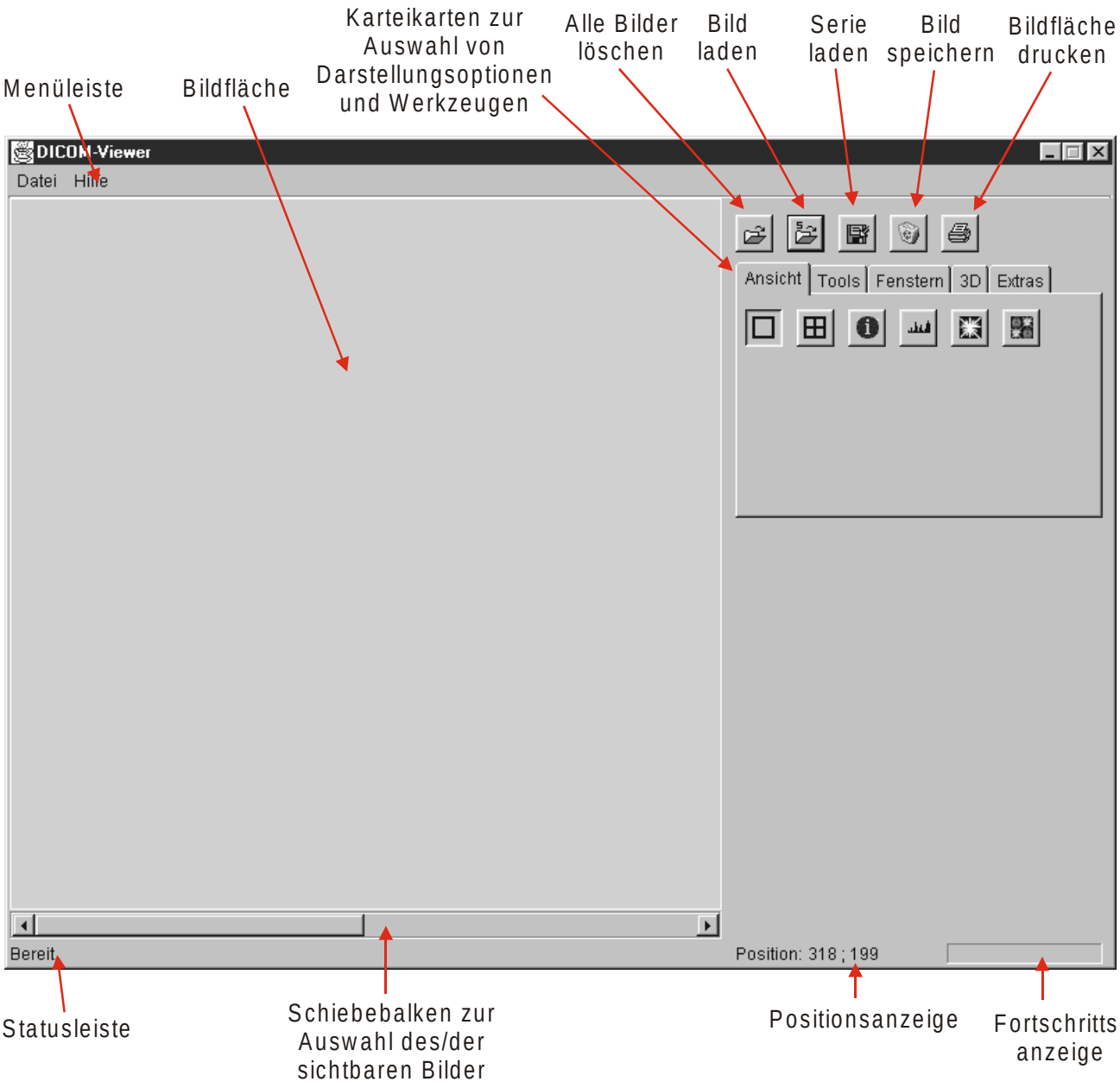

Abbildung 135: Hauptfenster des *DICOM-Viewers* nach dem Programmstart

Das Hauptfenster ist im wesentlichen in 2 Bereiche aufgeteilt. Links befindet sich die Bildfläche mit einem horizontalen Schiebebalken, der das Navigieren durch den aktuellen Bildstapel erlaubt. Die rechte Seite stellt den Kontrollbereich dar. Dieser enthält alle Bedienelemente, die zur Bedienung des *DICOM-Viewers* wichtig sind. Darüber hinaus befindet sich am unteren Rand des Fensters ein Informationsbereich. Dieser umfaßt eine Statuszeile, in der wichtige Programmeldungen und Hinweise ausgegeben werden, eine Positionsanzeige, der die aktuelle Position des Mauszeigers auf der Bildfläche zu entnehmen ist und eine Fortschrittsanzeige, die beim Laden einer Bildserie Informationen über den Fortschritt gibt. Die Menüleiste am oberen Rand des Hauptfensters hat keine große Bedeutung. Über das Dateimenü ist es lediglich möglich, ein Bild zu laden und zu speichern, den Bildflächeninhalt zu drucken und das Programm zu beenden. All diese Funktionen sind aber auch über die Bedienelemente des Kontrollbereichs erreichbar. Wir wollen uns daher im wesentlichen auf deren Beschreibung beschränken.

Allgemeine Funktionen

Zunächst wollen wir die Funktionalität der Knopfreihe oberhalb der Karteikarten im Kontrollbereich erläutern. Darüber hinaus werden wir noch einige allgemeine wichtige Hinweise zur Bedienung geben.

Laden eines einzelnen Bildes

Nach dem Programmstart wird der Anwender zuerst einmal ein Bild laden wollen. Es können ausschließlich Bilder geladen werden, die im DICOM-Format (PLAIN oder META) vorliegen. Der Versuch, eine Datei eines anderen Formats zu laden führt zu einer Fehlermeldung.

Zum Laden eines einzelnen Bildes betätigt der Benutzer den -Knopf (siehe Abbildung 135). Es öffnet sich dann der in Abbildung 136 dargestellte Dateiauswahldialog[31]. Mit Hilfe dieses Dialoges kann der Benutzer durch die Verzeichnisstruktur seines Dateisystems navigieren und das zu ladende Bild selektieren. Durch Betätigen des mit ‚*Öffnen*' beschrifteten Knopfes wird das ausgewählte Bild geladen und auf der Zeichenfläche dargestellt.

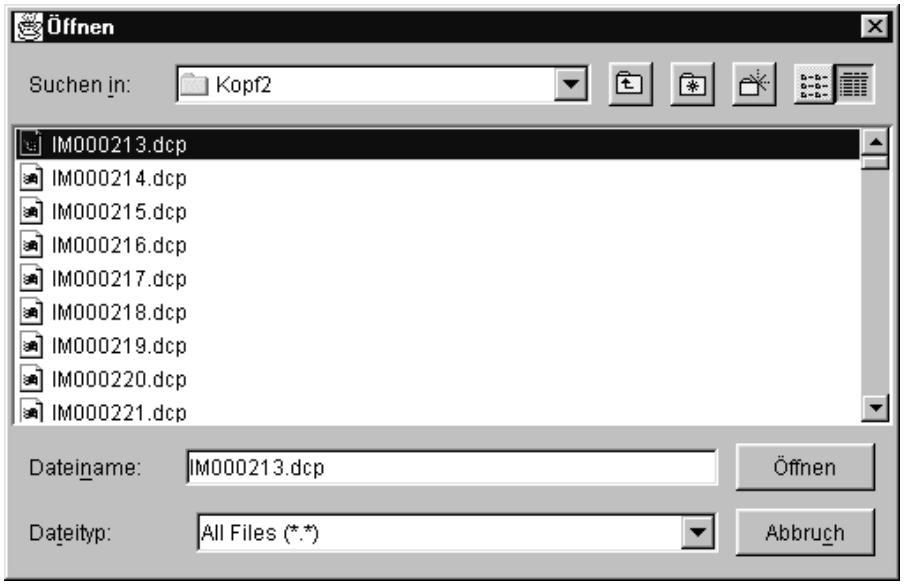

Abbildung 136: Dateiauswahldialog des *DICOM-Viewers*

Laden einer Bildserie

Alternativ zum Laden eines einzelnen Bildes hat der Benutzer die Möglichkeit, eine ganze Serie von DICOM-Bildern zu laden. Der Begriff Serie ist hier so definiert, daß alle Bilder eines Patienten gemeint sind, die sich im gleichen Verzeichnis befinden. Zum Laden einer Bildserie muß der Benutzer den -Knopf betätigen. Das kleine ‚s' in diesem Symbol steht dabei für Serie. Es öffnet sich dann ebenfalls der in Abbildung 136 dargestellte Dateiauswahldialog mit den gleichen Möglichkeiten, wie beim Laden eines Einzelbildes. Der Benutzer kann dann ein beliebiges Bild der zu ladenden Serie selektieren und anschließend den ‚*Öffnen*'-Knopf betätigen. Es werden dann automatisch alle Bilder der Serie geladen und dargestellt, zu der das selektierte Bild gehört und die sich im gleichen Verzeichnis wie das selektierte Bild befinden. Während dieses Vorgangs wird die Fortschrittsanzeige ständig aktualisiert, so daß der Benutzer eine Rückmeldung darüber erhält, wie weit das Laden fortgeschritten ist.

Navigieren durch den Bildstapel und Selektieren eines Bildes

Nachdem mehr als ein Bild geladen wurde, kann man mit Hilfe des horizontalen Schiebebalkens unterhalb der Bildfläche durch den Bildstapel navigieren. Durch einfaches Klicken mit einer beliebigen Maustaste auf dem gerade dargestellten Bild, wird dieses selektiert. Dieser Zustand wird dem Benutzer durch eine rote Umrandung des Bildes angezeigt. Die Selektion eines Bildes ist für

[31] Dieser Dateiauswahldialog erscheint bei Benutzung eines MICROSOFT-WINDOWS-Betriebsystems. Unter anderen Betriebssystemen sieht dieser Dialog eventuell etwas anders aus, weist jedoch die gleiche Funktionalität auf.

einige später beschriebene Funktionen wichtig. So wirkt sich zum Beispiel das Spiegeln, Drehen, Invertieren und Fenstern immer nur auf das selektierte Bild aus.

Speichern eines Bildes

Sobald ein Bild auf der Bildfläche dargestellt wird, kann dieses wie eben beschrieben selektiert und anschließend gespeichert werden. Das scheint zu diesem Zeitpunkt noch nicht sehr sinnvoll zu sein, da wir bisher nur Bilder geladen und nicht verändert oder neu erzeugt haben. Später werden wir jedoch auch beschreiben, wie man die geladenen Bilder bearbeiten und neue Bilder erzeugen kann. Dann ist es sehr wohl sinnvoll, diese Bilder zu speichern. Dazu muß der Benutzer den -Knopf betätigen. Es erscheint dann der in Abbildung 136 dargestellte Dateiauswahldialog mit einem anderen Fensternamen und anderen Beschriftungen der Knöpfe. Der Benutzer kann das Zielverzeichnis und die Zieldatei auswählen oder aber den Namen einer neu zu erstellenden Datei angeben. Nach Betätigung des Knopfes *‚Save‘* wird das selektierte Bild dann im DICOM-Format gespeichert.

Drucken des Bildflächeninhalts

Der aktuell dargestellte Inhalt der Bildfläche kann zu jeder Zeit auf einem auf dem System eingerichteten Drucker ausgedruckt werden. Dazu muß der Benutzer den -Knopf betätigen. Es erscheint dann ein systemabhängiger Druckdialog, wie er in Abbildung 137 für ein MS Windows-System dargestellt ist. Dort kann der Benutzer einen Systemdrucker auswählen und den Druckvorgang durch Betätigen des *‚OK‘*-Knopfes starten.

Abbildung 137: MS Windows Druckdialog

Löschen des aktuellen Bildstapels

Mittels des -Knopfes, werden alle geladenen Bilder von der Bildfläche entfernt und aus dem Speicher gelöscht. Bevor dies geschieht, richtet das System eine Sicherheitsabfrage an den Benutzer.

Funktionen der Karteikarte *‚Ansicht‘*

Wir wenden uns nun den Funktionen der Karteikarte *‚Ansicht‘* zu, die in Abbildung 138 dargestellt ist.

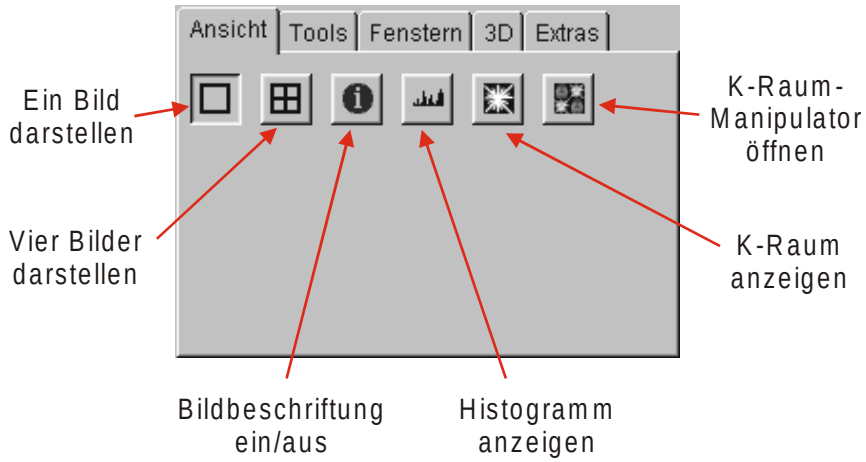

Abbildung 138: Karteikarte *‚Ansicht'* des *DICOM-Viewers*

1 bzw. 4 Bilder gleichzeitig darstellen

Die Darstellung eines Bildes auf der Bildfläche ist die Standardeinstellung. Da die meisten DICOM-Bilder eine Auflösung von 256*256 Punkten haben, werden diese hochskaliert und auf einer Fläche von 512*512 Punkten dargestellt. Es ist aber auch möglich, vier Bilder gleichzeitig darzustellen. Dazu muß der ⊞-Knopf der Karteikarte *‚Ansicht'* gedrückt werden. Es wird dann jedes Bild auf einer Fläche von 256*256 Punkten dargestellt. Bilder mit einer Auflösung von 512*512 Punkten werden dementsprechend herunterskaliert. Abbildung 139 zeigt das Hauptfenster, nachdem eine Bildserie geladen wurde und die Anzeige auf 4 Bilder umgeschaltet wurde.

Bildbeschriftungen ein-/ausblenden

Neben den Bildern können auch zusätzliche Bildinformationen auf der Bildfläche dargestellt werden. Man erreicht dies, indem man den ⓘ-Knopf drückt. Auf die gleiche Weise ist es möglich, die Bildbeschriftung wieder auszuschalten. Neben den *DICOM-Viewer*-internen Bildnummern werden einige DICOM-Informationen eingeblendet. Oben links werden die sequenzspezifischen Parameter wie Pulssequenzname (*Sequenz*), Repititionszeit (*TR*), Inversionszeit (*TI*), Echozeit (*TE*), Flipwinkel (*FA*) und die Anzahl der Echos pro Repititionszyklus (*ETL*) angezeigt. Oben rechts befindet sich das Aufnahmedatum des Bildes, der Patientenname (*Name*), sein Geburtsdatum (*Geb.Dat.*) und sein Geschlecht (*Geschl.*). Unten links findet man schließlich die Schichtposition (*SP*) innerhalb des gesamten Datensatzes, welche von der Wahl des Koordinatensystems während der Aufnahme abhängig ist und die aktuellen Werte der Fensterung (*C/W*).

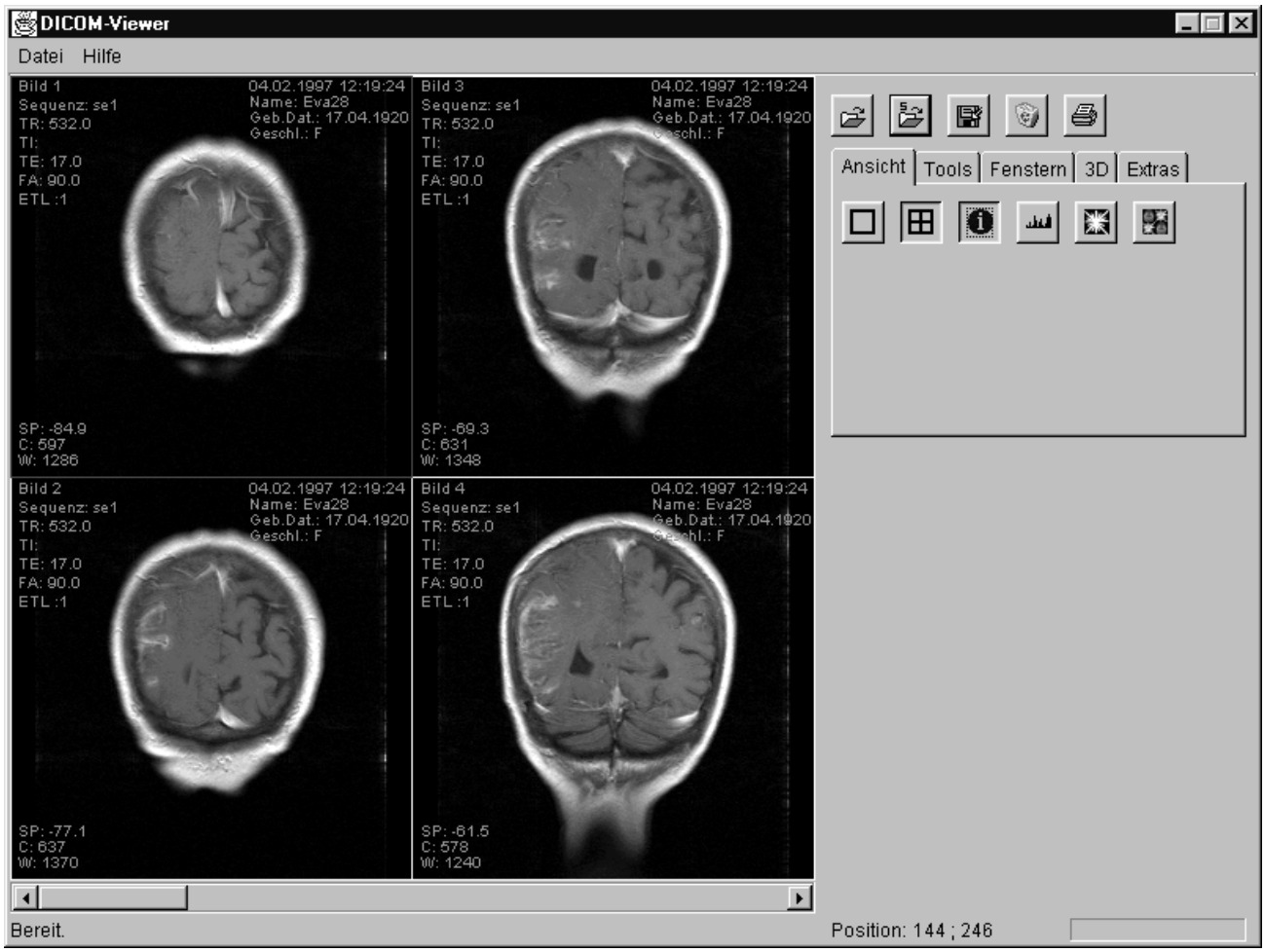

Abbildung 139: *DICOM-Viewer* nachdem eine Bildserie geladen wurde und die Beschriftung eingeschaltet wurde.

Histogramm anzeigen

Mit Hilfe des -Knopfes kann der Benutzer sich das Histogramm, also die Grauwertverteilung, des selektierten Bildes anzeigen lassen. Das Histogramm wird in einem separaten Fenster (Abbildung 140) dargestellt und kann zu beliebig vielen Bildern gleichzeitig geöffnet werden. Daraus ergibt sich eine Vergleichsmöglichkeit für den Benutzer.

Grauwerte, die sehr häufig im Bild vorkommen, werden als rote Balken dargestellt. Dies ist ein Zeichen dafür, daß sie nicht in voller Höhe dargestellt werden, da sonst die Balken der restlichen Grauwerte kaum noch zu erkennen wären.

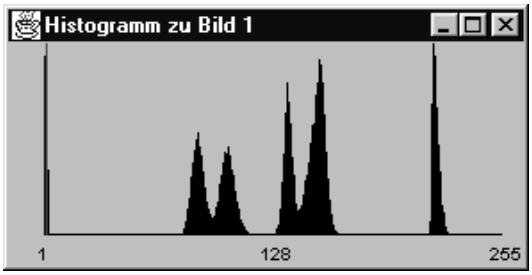

Abbildung 140: Histogrammfenster des *DICOM-Viewers*

k-Raum anzeigen

Durch Betätigen des -Knopfes wird der k-Raum bzw. die Fouriertransformation des selektierten Bildes in einem separaten Fenster (Abbildung 141) dargestellt.

Mit Hilfe der 4 Knöpfe am unteren Rand des Fensters kann der Benutzer zwischen der Anzeige des Magnitudenbildes (‚*Mag*‘), des Phasenbildes (‚*Pha*‘) oder des Real- (‚*Real*‘) oder Imaginärteils (‚*Imag*‘) der Fouriertransformation wählen. Dabei ist jedoch zu beachten, daß die Daten des Magnitudenbildes zur besseren Darstellung logarithmiert werden. Die theoretischen Hintergründe dazu liefert Kapitel 5.4.

Wie das Histogrammfenster kann auch das k-Raum-Fenster für beliebig viele Bilder geöffnet werden und muß mittels des Schließen-Knopfes rechts im Fensterrahmen manuell vom Benutzer geschlossen werden.

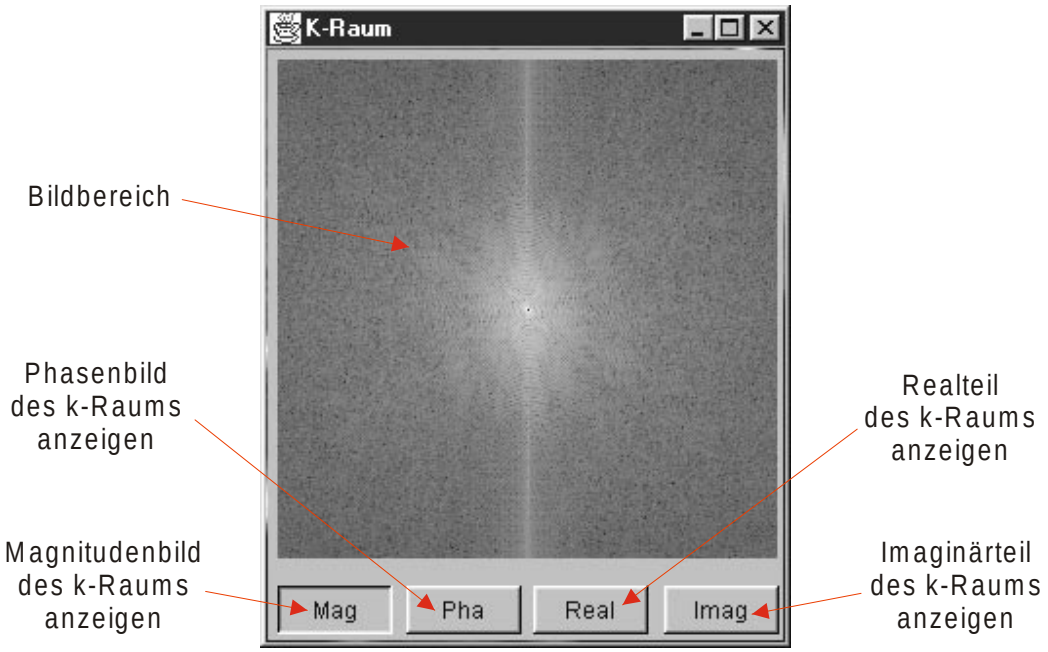

Abbildung 141: k-Raum-Anzeigefenster des *DICOM-Viewers*

k-Raum Manipulator öffnen

Der sogenannte k-Raum-Manipulator bietet Möglichkeiten, mit den in Kapitel 3.5 beschrieben Manipulationen des k-Raums zu experimentieren. Der k-Raum-Manipulator wird durch Betätigen des -Knopfes geöffnet. Es erscheint dann das in Abbildung 142 dargestellte Fenster. Oben links auf der Bildfläche ist das Bild dargestellt, das beim Öffnen des k-Raum-Manipulators selektiert war. Rechts daneben wird das Magnitudenbild (Kapitel 3.5) der Fouriertransformation (k-Raum) des Originalbildes dargestellt. Unten links wird zunächst auch der original k-Raum dargestellt. Allerdings kann dieser mit den im Kontrollbereich zur Verfügung stehenden Bedienelementen manipuliert werden.

Zur Zeit stehen 3 Manipulationsmöglichkeiten zur Verfügung: ‚*Ränder löschen*‘, ‚*Inneres Rechteck löschen*‘ und ‚*Zeilen/Spalten löschen*‘. Die Bedienelemente jedes dieser Operatoren sind jeweils in einem Rahmen gruppiert. Mit Hilfe des Kontrollkästchens vor der Operatorbeschriftung kann ein Manipulator ein- und ausgeschaltet werden. Sobald ein Operator eingeschaltet ist, wirken sich die Einstellungen der restlichen Bedienelemente dieses Operators auf das manipulierte k-Raum-Bild aus. So ist es mit Hilfe des Manipulators ‚*Ränder löschen*‘ möglich, separat einige Zeilen bzw. Spalten des linken, rechten, oberen oder unteren Bildrandes zu löschen, d.h. die Intensitätswerte in diesem Bereich auf Null zu setzen. Wie in Kapitel 3.5 beschrieben, hat diese Manipulation zur Folge, daß die Auflösung des rücktransformierten Bildes kleiner wird, da die Detailinformationen verloren gehen.

Mit Hilfe des Manipulators ‚*Inneres Rechteck löschen*‘ kann die Höhe und Breite eines zentral im k-Raum gelegenen Rechtecks bestimmt werden, daß gelöscht werden soll. Je größer dieses Rechteck gewählt wird, desto kontrastärmer und kantenbetonter wird das rücktransformierte Bild.

Der dritte Manipulator ‚*Zeilen/Spalten löschen*' kann Zeilen bzw. Spalten in einem bestimmten Abstand löschen. Das hat zur Folge, daß die Kopien höherer Ordnung näher zusammenrücken und im rücktransformierten Bild sichtbar werden. Dem Betrachter erscheint dies schließlich als Einfaltung (siehe auch Kapitel 3.5).

Unten rechts auf der Bildfläche kann schließlich das aus dem manipulierten k-Raum zurückberechnete Bild dargestellt werden. Da die Rückberechnung einige Zeit in Anspruch nimmt, wird diese nicht simultan zur Manipulation des k-Raums durchgeführt, sondern muß durch den Benutzer durch Betätigen des Knopfes ‚*Rücktransformation*' angestoßen werden.

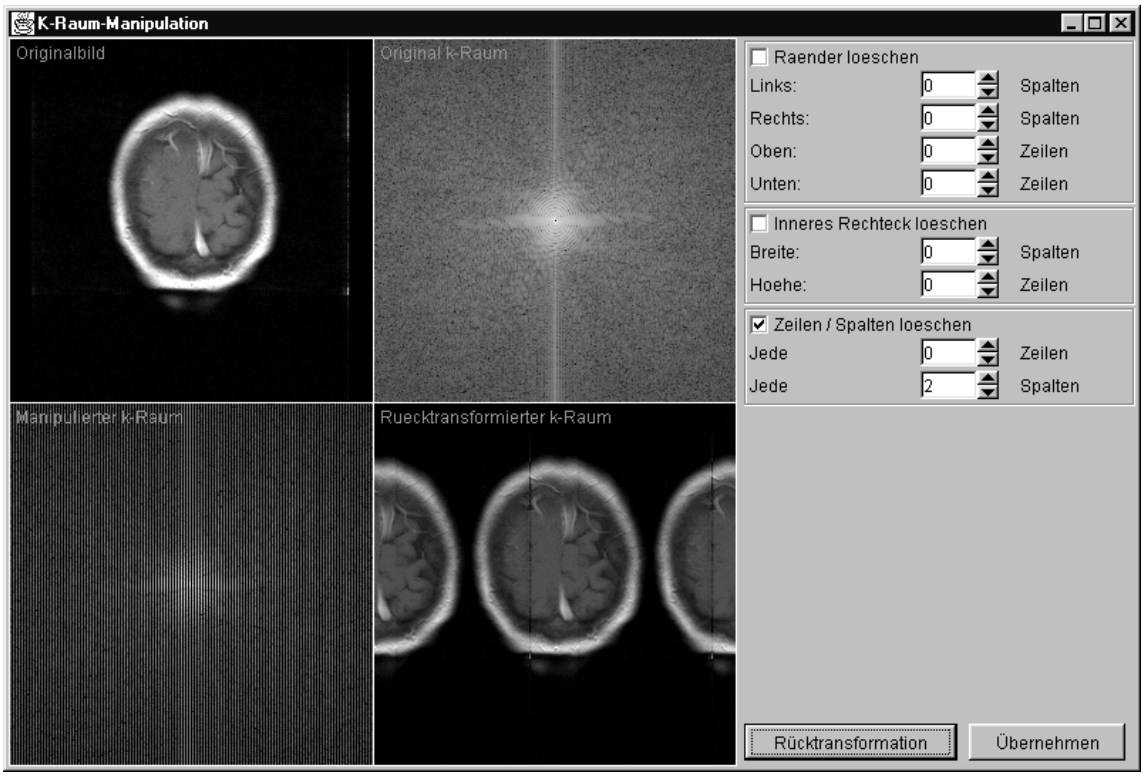

Abbildung 142: k-Raum-Manipulator des *DICOM-Viewers*

Sobald ein rücktransformiertes Bild berechnet wurde, kann dies mit Hilfe der Maus, wie unter dem Punkt ‚*Mauszeiger*' der Kateikarte ‚*Tools*' beschrieben, gefenstert werden. Anschließend kann das Bild mit Hilfe des ‚*Übernehmen*'-Knopfes in den *DICOM-Viewer* übertragen werden. Es wird dort entsprechend der Einstellungen der Karteikarte ‚*Extras*' als neues Bild eingefügt.

Funktionen der Karteikarte ‚*Tools*'

Die Karteikarte ‚*Tools*' (Abbildung 143) stellt einige einfache Meß- und Manipulationswerkzeuge zur Verfügung. Dazu gehört ein einfacher Mauszeiger, der auch zur Fensterung des Bildes benutzt werden kann, eine Lupe, ein Abstandsmesser, ein Winkelmesser, ein Werkzeug zur Grauwerterfassung und Möglichkeiten zum Spiegeln, Drehen, Invertieren und Löschen eines Bildes. Wir bezeichnen die Werkzeuge im folgenden mit den in Abbildung 143 angegebenen Bezeichnungen.

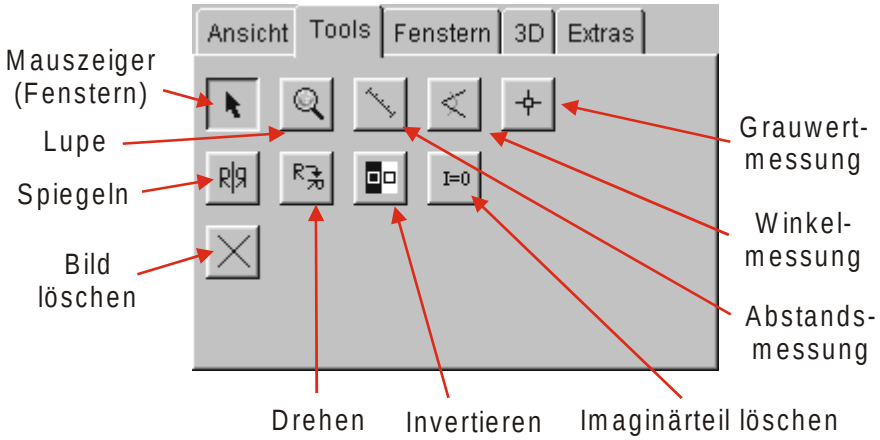

Abbildung 143: Karteikarte ‚*Tools*' des *DICOM-Viewers*

Mauszeiger

Mit Hilfe des ‚*Mauszeiger*'-Werkzeugs kann der Benutzer zum einen genau die Position des Mauszeigers über dem Bild bestimmen. Diese wird immer, wenn sich der Mauszeiger über der Bildfläche befindet, rechts neben der Statuszeile angezeigt. Diese Funktion steht jedoch auch zur Verfügung, wenn die Werkzeuge ‚*Lupe*', ‚*Abstandsmessung*', ‚*Winkelmessung*', oder ‚*Grauwertmessung*' ausgewählt sind.

Zum anderen kann jedoch nur mittels des ‚*Mauszeiger*'-Werkzeugs das selektierte Bild anders gefenstert werden. Durch Drücken und gedrückt halten einer beliebigen Maustaste während sich der Mauszeiger über der Bildfläche befindet wird die Fensterungs-Funktion aktiviert. Dies wird durch einen kontextsensitiven Mauszeiger angezeigt. Durch Bewegen der Maus nach oben oder unten wird das Zentrum der Fensterung variiert, durch Bewegen nach rechts oder links die Breite des ausgewählten Fensters. Die aktuellen Einstellungen für Zentrum und Breite können entweder an der Bildbeschriftung abgelesen oder aber anhand der Einstellungen der Karteikarte ‚*Fenstern*' beobachtet werden.

Lupe

Ist das ‚*Lupe*'-Werkzeug ausgewählt, so kann ein Bereich von 32*32 Bildpunkten unterhalb des Mauszeigers auf einen Bereich von 64*64 Bildpunkten, also um den Faktor 2, vergrößert werden.

Abstandsmessung

Mit Hilfe der ‚*Abstandsmessung*' kann der Benutzer den Abstand zwischen zwei Punkten innerhalb eines Bildes bestimmen. Dazu muß zunächst das Werkzeug ‚*Abstandsmessung*' ausgewählt werden. Dann muß der Benutzer den Mauszeiger an die Anfangsposition der Abstandsmessung bewegen, dort einen Mausknopf drücken und mit gedrückt gehaltenem Mausknopf den Mauszeiger zur Endposition der Abstandsmessung bewegen. Während der Messung wird ständig eine gelbe Verbindungslinie zwischen der Anfangsposition und der aktuellen Position und die Länge dieser Linie eingezeichnet. Nach Loslassen der Maustaste ist die Abstandsmessung abgeschlossen und die endgültige Verbindungslinie und deren Länge wird solange angezeigt, wie sich der Mauszeiger über der Bildfläche befindet. Verläßt der Mauszeiger die Bildfläche, gehen die Informationen der Abstandsmessung unwiderruflich verloren. Sind Maßstabsinformationen in den DICOM-Informationen enthalten, so wird der Abstand in Millimeter angezeigt, ansonsten in Pixel.

Winkelmessung

Die ‚*Winkelmessung*' ermöglicht es dem Benutzer, den Winkel zwischen zwei selbst eingezeichneten Geraden zu bestimmen. Das Vorgehen dazu verläuft ähnlich wie bei der Abstandsmessung. Zunächst muß das Werkzeug ‚*Winkelmessung*' ausgewählt werden. Dann muß der Benutzer den Mauszeiger zum gewünschten Scheitelpunkt der Winkelmessung bewegen und dort eine beliebige Maustaste drücken. Mit gedrückter Maustaste bewegt man nun den Mauszeiger zum Endpunkt der ersten Kathete. Dort muß die Maustaste losgelassen werden. Während dieses Vorgangs wird ständig die aktuelle Kathete eingezeichnet. Durch Bewegen des Mauszeigers ohne gedrückte Maustaste wird nun die zweite Kathete durch den anfangs ausgewählten Scheitelpunkt und der aktuellen Mauszeigerposition bestimmt und gleichzeitig der Winkel zwischen beiden Katheten in Grad angezeigt. Alle eingezeichneten Elemente werden gelöscht, wenn sich der Mauszeiger von der Bildfläche bewegt und gehen damit unwiderruflich verloren.

Grauwertmessung

Ist das Werkzeug ‚*Grauwertmessung*' ausgewählt, so wird, wenn sich der Mauszeiger über einem Bild befindet, der darunter liegende Grauwert in 12-Bit-Darstellung auf gelb hinterlegtem Hintergrund neben dem Mauszeiger angezeigt.

Spiegeln

Durch Betätigen des -Knopfes (‚*Spiegeln*') wird das aktuell selektierte Bild an einer vertikalen Achse gespiegelt.

Drehen

Durch Betätigen des -Knopfes (‚*Drehen*') wird das aktuell selektierte Bild um 90° im Uhrzeigersinn gedreht.

Invertieren

Durch Betätigen des -Knopfes (‚*Invertieren*') werden die Grauwerte des aktuell selektierten Bildes invertiert. Je nach Fensterung des Bildes ist es möglich, daß das Bild nach dieser Operation komplett schwarz oder weiß ist. Da es jedoch nicht sinnvoll ist, automatisch eine neue Fensterung vorzugeben, bleibt es dem Benutzer überlassen, das Bild neu zu fenstern.

Löschen eines Einzelbildes

Durch Betätigen des -Knopfes kann das aktuell selektierte Bild gelöscht werden. Vor dem endgültigen Löschen, stellt das System dem Benutzer eine Sicherheitsabfrage.

Funktionen der Karteikarte ‚*Fenstern*'

Mit Hilfe der Bedienelemente der Karteikarte ‚*Fenstern*' ist es dem Benutzer möglich, das aktuell selektierte Bild anders zu fenstern. Dabei sind ihm eine optimale und eine Standardfensterung vorgegeben. Darüber hinaus gibt es aber auch die Möglichkeit, daß Bild manuell zu fenstern.

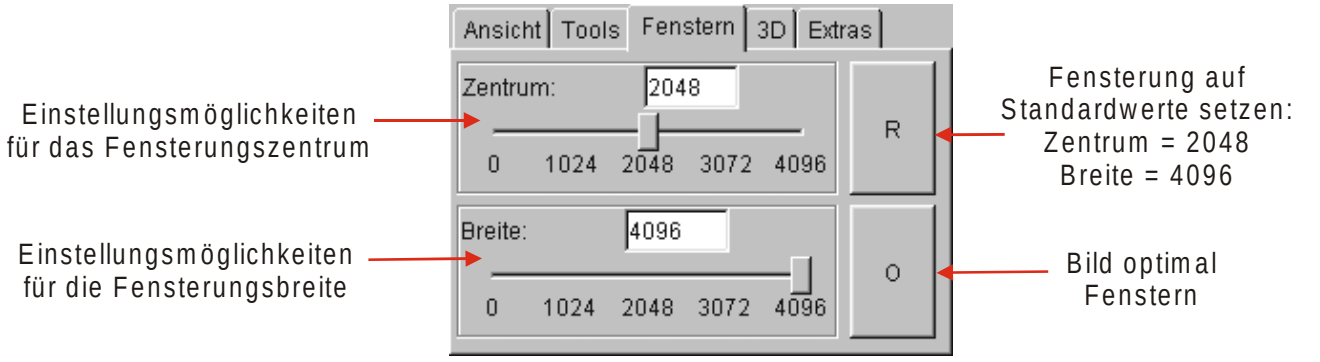

Abbildung 144: Karteikarte ‚*Fenstern*' des *DICOM-Viewers*

Zurücksetzen der Fensterung auf Standardwerten

Mit Hilfe des mit ‚*R*' beschrifteten Knopfes wird die Fensterung des selektierten Bildes auf Standardwerte gesetzt. Für das Zentrum ist der Standardwert 2048, für die Breite 4096. Damit werden alle Grauwerte des Bildes dargestellt. Unter Umständen ist der Bildkontrast jedoch sehr schlecht.

Die Funktion eignet sich insbesondere, wenn ein Bild invertiert werden soll. Da mit dieser Einstellung im Ursprungsbild alle Grauwerte sichtbar sind, sind sie das auch im invertierten Bild. Es kann also nicht der Fall eintreten, daß das Bild nach dem Invertieren komplett schwarz oder weiß ist.

Optimale Fensterung

Der mit ‚*O*' beschriftete Knopf ermöglicht eine optimale Fensterung des selektierten Bildes. Dazu wird zunächst der kleinste Grauwert *min* und größte Grauwerte *max* im Bild bestimmt. Die Breite des Grauwertfensters ergibt sich dann zu *max-min* und das Zentrum zu *min+(max-min)/2*. Die Funktion gewährleistet, daß alle unterschiedlichen Grauwerte des Bildes differenzierbar bleiben und ein optimaler Kontrast über das gesamte Bild erzielt wird. Interessiert jedoch nur ein optimaler Kontrast in einem Teilbereich des Bildes, so muß der Benutzer die Fensterung hierfür manuell einstellen.

Manuelle Fensterung

Mit Hilfe der Schieberegler für das Zentrum und die Breite der Fensterung und der dazugehörigen Textfelder kann der Benutzer das selektierte Bild ganz genau manuell Fenstern. Durch Drücken einer Maustaste auf einem der Schieberegler und Bewegen bei gedrückter Maustaste kann dieser punktgenau verschoben werden. Darüber hinaus gibt es die Möglichkeit, direkt Werte in die Textfelder einzugeben und mit der Eingabetaste zu bestätigen.

Die manuelle Fensterung des Bildes kann außerdem direkt durch Mausbewegung auf der Bildfläche erfolgen, wie dies bereits im Zusammenhang mit dem Werkzeug ‚*Mauszeiger*' der Karteikarte ‚*Tools*' beschrieben wurde.

Funktionen der Karteikarte ‚*3D*'

Die Karteikarte ‚*3D*' bietet einige weiterführende Operationen zur Erstellung einer Animation und zur Berechnung neuer Schichtführungen bzw. eines MIP-Bildes aus dem aktuellen Bildstapel an.

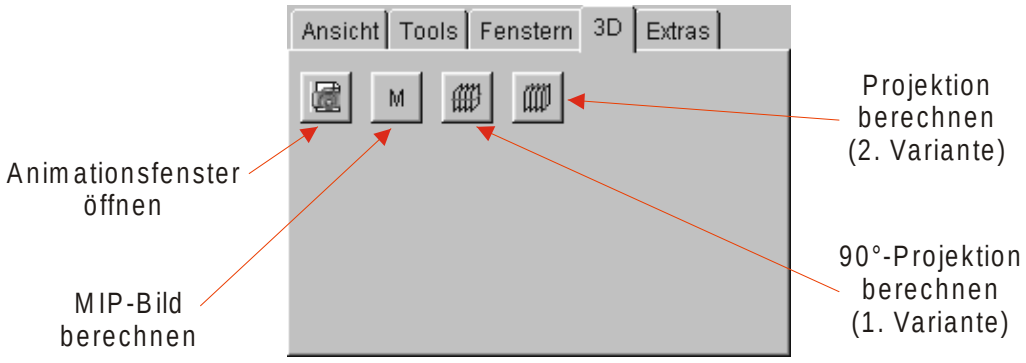

Abbildung 145: Karteikarte ‚*3D*' des *DICOM-Viewers*

Berechnung eines MIP-Bildes

Der mit ‚*M*' beschriftete Knopf dient zur Berechnung eines MIP-Bildes. Das ist ein Bild, daß aus einem Bildstapel berechnet wird, wobei für jeden Bildpunkt die Intensität genommen wird, die an dieser Stelle innerhalb des Bildstapels maximal ist. Diese Funktion eignet sich besonders zur Erstellung von Angiographien. Es ist jedoch zu beachten, daß dazu eine Bildserie nötig ist, die mit einer speziellen Pulssequenz aufgenommen wurde, so daß nur Bereiche mit Blutfluß mit hohen Intensitäten dargestellt werden.

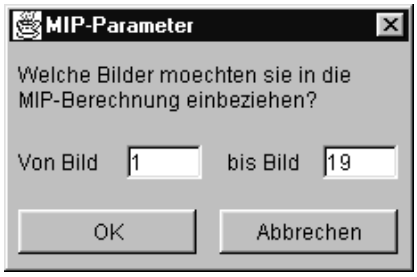

Abbildung 146: Parameterdialog zur Berechnung eines MIP-Bildes

Nach Betätigen des ‚*M*'-Knopfes zur Berechnung des MIP-Bildes erscheint zunächst das in Abbildung 146 dargestellte Dialogfenster. Hier wird der Benutzer aufgefordert, den Bereich der Bilder anzugeben, die in die Berechnung des MIP-Bildes einbezogen werden sollen. Voreingestellt sind alle zur Zeit im Bildstapel befindlichen Bilder. Nach Betätigen des ‚*OK*'-Knopfes wird das MIP-Bild berechnet und dargestellt. An welcher Position das Bild dargestellt wird ist davon abhängig, welche Option in der Karteikarte ‚*Extras*' für die Funktion ‚*Nächstes Bild positionieren*' ausgewählt ist.

Berechnung einer 90°-Projektion

Mit Hilfe der mit und gekennzeichneten Knöpfe kann eine 90°-Projektion, also eine Schicht einer anderen Schichtführung, aus dem aktuellen Bildstapel berechnet werden. Welche Schichtführung das ist, hängt letztendlich schon von der Schichtführung des aktuell geladenen Datensatzes ab. Zur besseren Vorstellung kann der Benutzer sich die Symbole der Funktionsknöpfe ansehen. Die schwarz gezeichneten Schichten stellen die Schichtführung des aktuell geladenen Datensatzes dar, wohingegen die rot gezeichnete Schicht die Lage der neu berechneten Schicht andeutet.

Nach Betätigen eines der beiden Knöpfe zur Berechnung einer der 90°-Projektionen erscheint der in Abbildung 147 dargestellte Dialog zur Eingabe einiger Parameter.

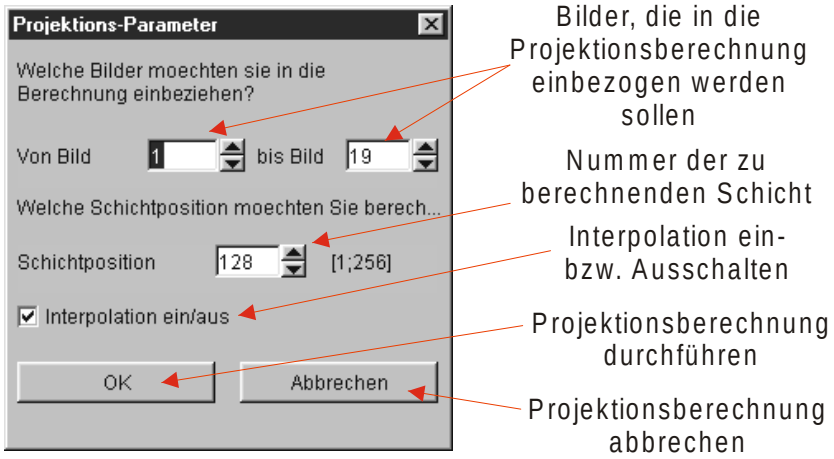

Abbildung 147: Parameterdialog zur Berechnung einer 90°-Projektion

In die beiden oberen Textfelder kann der Benutzer den Bereich der Bilder angeben, die in die Berechnung der 90°-Projektion einbezogen werden sollen. Mit Hilfe des darunter befindlichen Textfeldes kann die Position der neu zu berechnenden Schicht bestimmt werden. Das Intervall der zulässigen Eigaben ist von der Auflösung der Bilder des aktuellen Schichtstapels abhängig. Dabei ist vom Benutzer darauf zu achten, daß alle Bilder die gleiche Auflösung haben und sinnvoller Weise auch zur gleichen Serie gehören. Zusätzlich kann der Benutzer mit Hilfe des unten angeordneten Kontrollkästchens eine Interpolationsfunktion ein- oder ausschalten. Es handelt sich dabei allerdings lediglich um eine lineare Interpolation entlang einer Linie.

Nach Betätigung des ‚*OK*'-Knopfes wird das neue Bild mit Hilfe der eingegebenen Parameter berechnet und dargestellt. An welcher Position das Bild dargestellt wird ist davon abhängig, welche Option in der Karteikarte ‚*Extras*' für die Funktion ‚*Nächstes Bild positionieren*' ausgewählt ist.

Animation

Die Animationsfunktion erlaubt es, die Einzelbildes des aktuellen Bildstapels automatisch nacheinander darzustellen, so daß der Eindruck entsteht, als würde man durch das dargestellte Körperteil „fahren". Das in Abbildung 148 dargestellte Animationsfenster mit seinen zahlreichen Konfigurationsmöglichkeiten erscheint nach Betätigung des mit gekennzeichneten Knopfes.

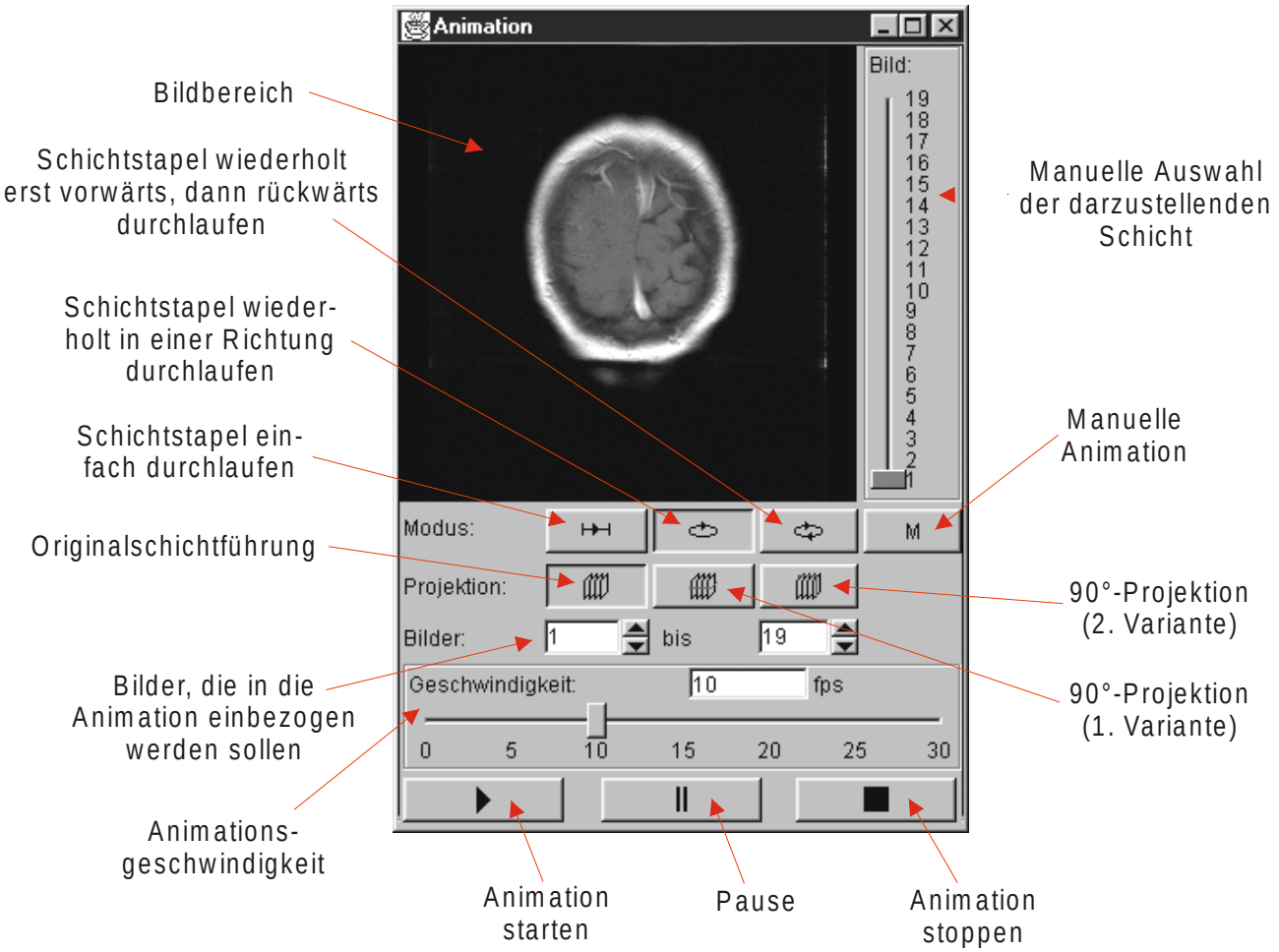

Abbildung 148: Animationsfenster des *DICOM-Viewers*

Möchte der Benutzer nur den kompletten aktuellen Schichtstapel in seiner Originalschichtführung animieren, so muß an den Standardeinstellungen nichts verändert werden. Durch Betätigen des ▶-Knopfes unten links kann die Animation gestartet werden. Sie wird dann fortlaufend vom ersten bis zum letzten Einzelbild durchlaufen. Mit Hilfe des ⏸-Knopfes (unten mittig) kann die Animation jederzeit angehalten und durch erneutes Betätigen des ⏸-Knopfes an der gleichen Stelle wieder fortgesetzt werden. Der ■-Knopf unten rechts beendet schließlich die Animation. Ein Neustart ist dann nur durch Betätigen des ▶-Knopfes möglich und beginnt wieder beim ersten Einzelbild des Bildstapels.

Oberhalb der Knöpfe zum Starten, Anhalten und Stoppen der Animation befindet sich ein Schieberegler mit dem die Animationsgeschwindigkeit eingestellt werden kann. Die Einheit ist in Bildern pro Sekunde (fps = frames per second) angegeben. Durch Betätigen einer Maustaste auf dem Schieberegler und Bewegen des Mauszeigers mit gedrückt gehaltener Maustaste kann der Schieberegler punktgenau bewegt werden. Darüber hinaus besteht die Möglichkeit, direkt einen Wert in das Textfeld oberhalb des Schiebereglers einzugeben und die Eingabe mit der Eingabetaste zu bestätigen. Das Verändern der Animationsgeschwindigkeit ist auch während der laufenden Animation möglich. Es ist zu beachten, daß der angegebene Wert ein Maximalwert ist und stark von der Leistung des Ausführungsrechners abhängt.

Oberhalb der Einstellungsmöglichkeiten für die Animationsgeschwindigkeit befinden sich zwei Texteingabefelder mit der Beschriftung ‚*Bilder*'. Hier kann wie bei der Projektionsberechnung ein Bereich von Bildern angegeben werden, der in die Animation einbezogen werden soll. Standardmäßig sind alle Bilder des aktuellen Stapels eingestellt.

Die drei Knöpfe rechts neben der Beschriftung ‚*Projektion*‘ dienen zur Auswahl der Schichtführung, die bei der Animation verwendet werden soll. Der linke Knopf mit dem Symbol wählt die Originalschichtführung aus, verwendet also die Schichten so, wie sie im Bildstapel vorliegen. Da bei dieser Animationsrichtung keine Schichtumrechnungen nötig sind, kann die Animation mit sehr hohen Geschwindigkeiten durchgeführt werden. Die beiden Knöpfe, die mit bzw. gekennzeichnet sind, wählen als Animationsrichtung eine der beiden 90°-Projektionen, die weiter oben unter dem Stichwort ‚*Berechnung einer 90°-Projektion*‘ beschrieben wurden. Bei der Berechnung der Projektionen ist die Interpolation zwischen den Schichten immer eingeschaltet, da somit ein flüssigerer Verlauf erreicht wird. Da die Berechnung der Projektionen mit erheblichem Zeitaufwand verbunden ist, ist die Ausführungsgeschwindigkeit der Animation recht gering.

Die vier Knöpfe rechts neben der Beschriftung ‚*Modus*‘ dienen zur Auswahl des Animationsmodus. Ist der mit dem -Symbol gekennzeichnete Knopf gedrückt, wird der ausgewählte Schichtstapel nach Betätigung des -Knopfes nur einmal von vorne bis hinten durchlaufen und danach automatisch beendet. Ist hingegen der -Knopf selektiert, wird der Schichtstapel wiederholt vom ersten bis zum letzten vorgewählten Bild durchlaufen. Der Knopf mit dem -Symbol sorgt schließlich dafür, daß der Schichtstapel zuerst vom ersten bis zum letzten vorgewählten Bild durchlaufen wird und anschließend rückwärts vom letzten bis zum ersten. Danach wieder vorwärts und so weiter. Der mit einem ‚*M*‘ beschriftete Knopf ermöglicht schließlich die manuelle Steuerung der Animation mittels des Schiebereglers, der sich rechts neben der Bildfläche befindet. Somit kann jede Schicht manuell angesteuert werden.

Funktionen der Karteikarte ‚*Extras*‘

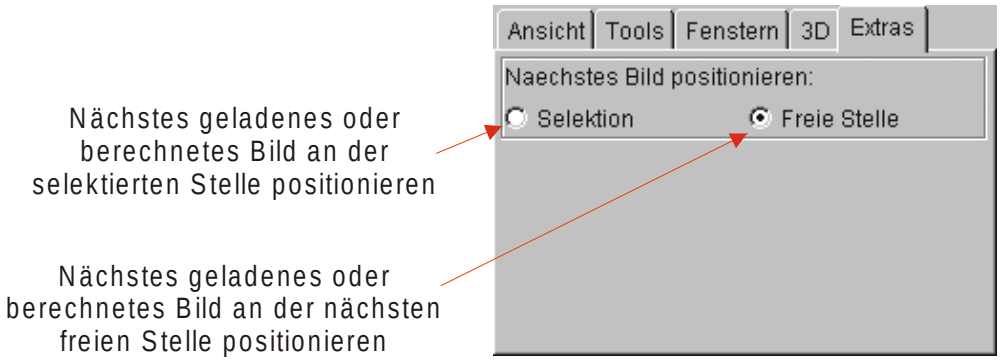

Abbildung 149: Karteikarte ‚Extras‘ des *DICOM-Viewers*

Position für ein neues Bild festlegen

Mit Hilfe der Funktion ‚*Nächstes Bild positionieren*‘ kann der Benutzer festlegen, an welcher Position das nächste geladene oder berechnete Bild dargestellt werden soll. Die Option ‚*Selektion*‘ positioniert das Bild an der selektierten Position und überschreibt dabei gegebenenfalls ein an dieser Stelle zuvor vorhandenes Bild. Die Option ‚*Freie Stelle*‘ fügt ein neues Bild an der nächsten freien Stelle im Bildstapel ein. Dies ist die Standardeinstellung. Dabei werden keine Bilder überschrieben und der Benutzer muß sich nicht selbst darum kümmern, die Position für das nächste Bild festzulegen.

Ein Ausnahmefall tritt ein, wenn eine komplette Bildserie geladen wird. Auch wenn die Option ‚*Selektion*‘ eingeschaltet ist, werden alle Bilder der Serie an den nächsten freien Positionen im Bildstapel eingefügt, damit ein Bild der Serie nicht das zuvor geladene Bild überschreibt.

7.2.3 Bedienung der Parameterberechnung und des Zeichenwerkzeuges

Zum Abschluß unserer Bedienungsanleitungen wollen wir die Werkzeuge zum Berechnen und Bearbeiten neuer Parameterbilder beschreiben. Wie im Umsetzungskapitel beschrieben, sind beide Programme als sogenannte „Plugins“ für das freie Programm IMAGEJ implementiert. Zum Starten von

IMAGEJ genügt unter Windows ein Doppelklick auf das bei der Installation erzeugte Icon. Unter anderen Betriebssystemen kann dies abweichen, darauf wollen wir aber nicht näher eingehen.

Nach dem Start funktioniert das Programm aber unter allen Betriebssystemen gleich. Im folgenden wollen wir beschreiben, wie der Benutzer die Funktionen zur Berechnung und Bearbeitung von Parameterbildern nutzen kann.

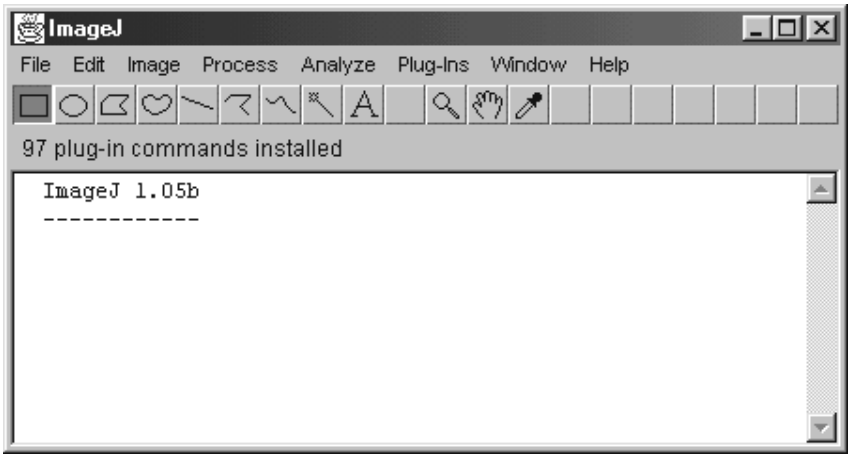

Abbildung 150: Das Hauptfenster von IMAGEJ

Abbildung 150 zeigt das Hauptfenster von IMAGEJ nach dem Start. Für unsere Zwecke ist nur der Menüpunkt *‚Plug-Ins‘* interessant. Dort sind die Module zur Berechnung und Überarbeitung von Parameterbildern implementiert. Zum Berechnen eines Parameterdatensatzes benötigt man einen Datensatz von Parameterbildern aus einer MRT-Aufnahme, dabei muß es sich wie im Konzeptkapitel über die Parameterbildberechnung beschrieben um eine Spin-Echo-16-Sequenz zur T_2 und PD-Berechnung und um eine Spin-Echo-Sequenz zur T_1-Berechnung handeln. Die Sequenz muß so sortiert werden, daß die DICOM-Tags für die Bildnummern fortlaufend von 1 bis *n* sortiert sind, wobei *n* die Anzahl der Bilder ist. Dies kann aber bereits mit dem MRT-Gerät geschehen.

Berechnung der Parameterbilder

Um die Berechnung der Parameterbilder zu starten, muß man aus dem Menü *‚Plug-Ins‘* den Menüpunkt *‚Parameterbilder berechnen‘* aufrufen. Zunächst erscheint ein Dialog zum Import der Ausgangsserie im DICOM-Format wie in Abbildung 151 dargestellt. Zum Einlesen der DICOM-Dateien müssen jetzt die Auswahlkästchen *‚Include DICOM header‘* und *‚Load multiple images‘* aktiviert sein. In das Textfeld mit der Beschriftung „image list“ muß die komplette Serie angegeben werden. Für unser Programm *Virtual MRT* haben wir eine Aufnahme einer Kontrastmittelstudie im realen Tomographen erzeugt. Diese Serie umfaßt 42 Bilder, in diesem Beispiel muß also *„1-42“* in das oben erwähnte Feld eingetragen werden.

Schließlich muß noch über *‚Select a DICOM file‘* eine Datei aus der gewünschten Serie gewählt werden. Zu diesem Zweck erscheint ein Dateiauswahldialog, wie er in Abbildung 152 dargestellt ist. Nach Bestätigung der Dateiauswahl mit *‚OK‘* wird auch der DICOM-Import-Dialog mit *‚OK‘* geschlossen, damit wird das Einlesen der DICOM-Daten gestartet.

Nach einer gewissen Wartezeit[32] werden in einem weiteren Dialog die Bilder aus der Serie für die T_2- und PD-Berechnung und die Bilder aus der Serie zur T_1. Der Dialog ist in Abbildung 153 dargestellt. Oben im Dialog werden die Bilder für die T_1-Berechnung, unten die Bilder für die T_2-Berechnung festgelegt.

[32] Auf eine Pentium-II-System mit 300 Mhz beträgt die Rechenzeit ca. 30 Sekunden.

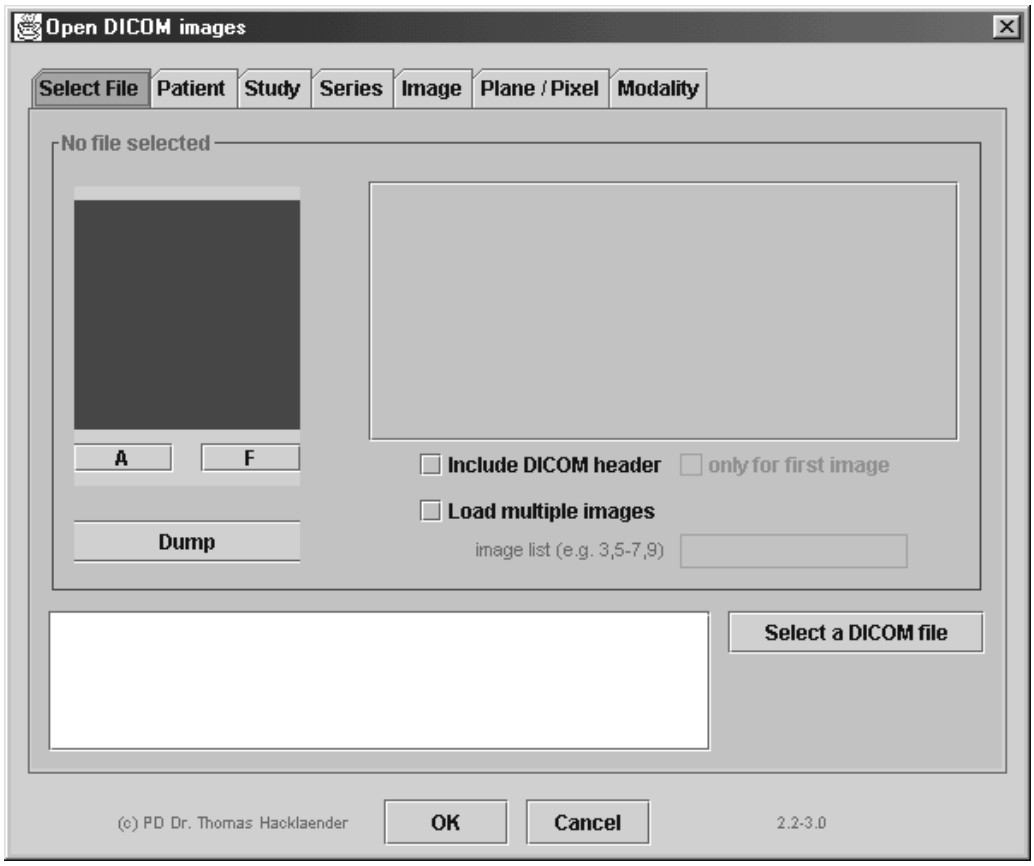

Abbildung 151: Der Importdialog für DICOM-Dateien

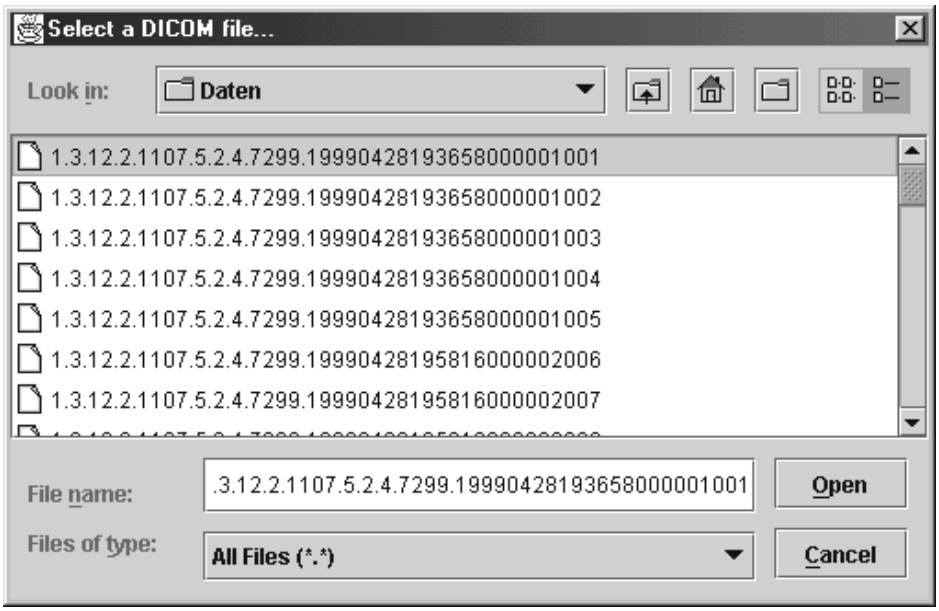

Abbildung 152: Der Dateilauswahldialog

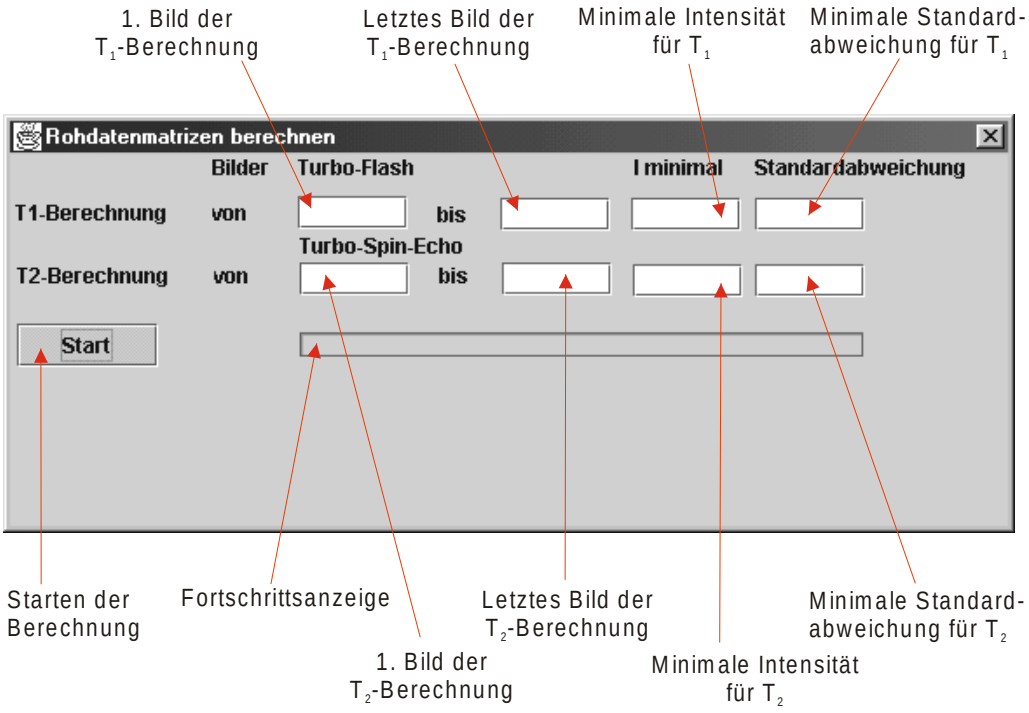

Abbildung 153: Auswahl der Bilder einer Serie zur Parameterbildberechnung

Außerdem wird für beide Serien eine minimale Signalintensität und eine minimale Standardabweichung angegeben. Der minimale Intensitätswert sorgt dafür, daß in Bildbereichen mit sehr niedriger Signalintensität (z.B. Bereiche mit Luft) keine Berechnung durchgeführt wird, da dort keine sinnvollen Werten berechnet werden können. Auch in Bereichen in unserem Beispiel der Kontrastmittelstudie mit fast konstanter Signalintensität kann keine sinnvolle Berechnung durchgeführt werden, dieser Fall wird über den Parameter einer minimalen Standardabweichung abgefangen. Der Benutzer kann mit diesen Einstellungen experimentieren, als gute Werte haben sich 30 für die Signalintensität und 8 für die minimale Standardabweichung herausgestellt. Nach dem diese Einstellungen vorgenommen worden sind, wird die Berechnung durch Betätigen des ‚*Start*'-Knopfes angestossen.

Nachdem die Parameterbilder berechnet sind, können diese als Serie von DICOM-Bildern mit zugehöriger *idx*-Datei exportiert werden. Die *idx*-Datei ist wie im Kapitel 5.2 beschrieben aufgebaut.

Abbildung 154 zeigt den Export-Dialog für die berechneten Parameterbilder. Zum Abspeichern der Parameterbilder muß über den Knopf, der mit ‚...' beschriftet ist, eine Datei ausgewählt werden. Nach Bestätigen des Dateiauswahldialogs werden die übrigen Felder automatisch gefüllt. Der Benutzer kann diese optional ändern, dabei muß jedoch darauf geachtet werden, daß die Dateinamen von oben (T_1) nach unten (Flow) lexikographisch sortiert sind, da sonst die Parameterbildserie zur Nachbearbeitung nicht in der korrekten Reihenfolge erneut eingelesen werden kann. Mit dem Knopf „Speichern" wird die Serie im DICOM-Format gespeichert und eine *idx*-Datei erzeugt. Die so berechneten Parameterbilder können jetzt von dem Programm *Virtual MRT* benutzt werden, um eine Messung zu simulieren.

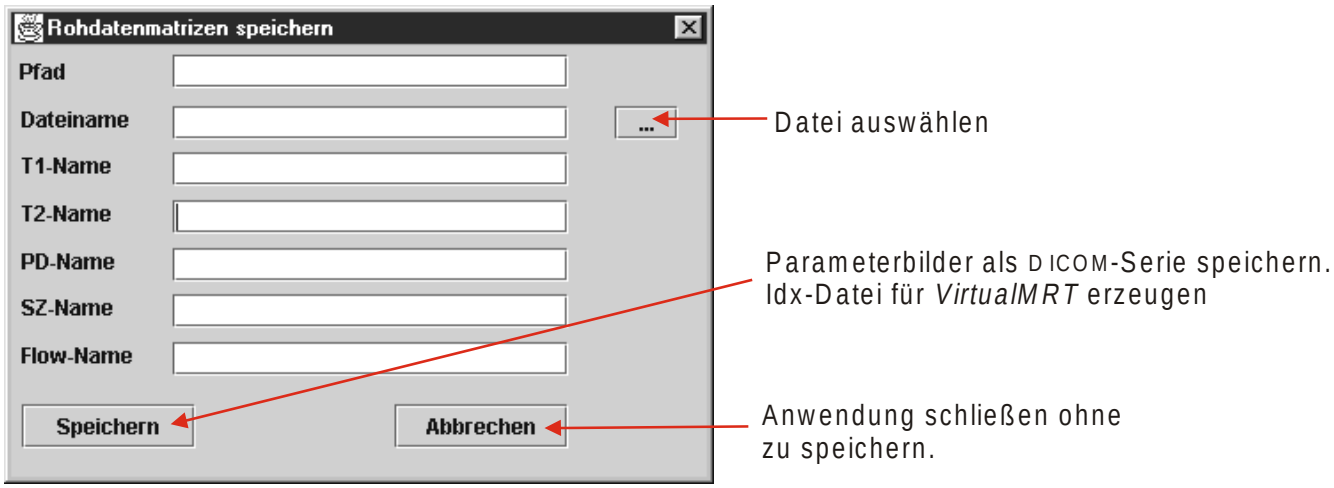

Abbildung 154: Abspeichern der berechneten Parameterbilder

Nachbearbeiten der Parameterbilder

Die Parameterbilder können entweder direkt nach der Berechnung bearbeitet werden, oder man importiert die Parameterbilder über das Plugin „*Dicomimport*“. Bereits im vorherigen Abschnitt haben wir beschrieben, wie dort eine Serie ausgewählt wird, hier muß allerdings das Auswahlkästchen ‚*Include DICOM header*‘ nicht aktiviert werden.

Nach dem Importieren oder Berechnen wählt man jetzt die Option ‚*Stack retuschieren*‘ aus dem ‚*Plugins*‘-Menü. Daraufhin wird ein kleines Malprogramm geöffnet, das sich wie in Abbildung 155 auf dem Bildschirm darstellt. Über die Auswahlknöpfe in der Spalte unter dem -Symbol wird bestimmt, welches Parameterbild unten rechts dargestellt wird. Es kann entweder das Bild für Suszeptibilitätswerte oder aber das Bild für den Spinfluß dargestellt werden.

Wie vom *Virtual MRT* und dem *DICOM-Viewer* gewohnt, werden auch die Parameterbilder links im Anzeigebereich dargestellt. Auf der rechten Seite befinden sich die Bedienelemente zur Steuerung des Retuschierungsprogrammes. In der oberen Reihe von Knöpfen kann die gewünschte Zeichenfunktion ausgewählt werden. Von links nach rechts sind dies Linie, Quadrat, Ellipse und Freihandzeichnung. Durch Betätigen der Umschalttaste während des Auswählens kann auch ein gefülltes Quadrat, eine gefüllte Ellipse und eine gefüllte Freihandzeichnung in die Parameterbilder gezeichnet werden. Zeichnen kann der Benutzer durch Drücken der linken Maustaste und Ziehen der Maus über dem Anzeigebereich.

Der eigentliche Sinn dieses einfachen Werkzeuges ist das Zeichnen in mehrere Bilder des Parameterdatensatzes gleichzeitig. Alle Auswahlkästchen in der Spalte unter dem - Symbol bestimmen, in welche der fünf Parameterbilder gezeichnet wird. Mit der Auswahl, die in Abbildung 155 dargestellt ist, würde die Zeichenoperation auf T_1-, T_2- und dem Protonendichtebild ausgeführt. Dies ist auch die Standardeinstellung.

Durch Betätigen des Knopfes, der mit ‚*U*‘ beschriftet ist, wird die zuletzt ausgeführte Operation rückgängig gemacht.

Zum Beenden des Programmes muß der Benutzer den Knopf in der oberen rechten Ecke des Fensters („*x*“) betätigen.

Es erscheint wieder der aus Abbildung 154 bekannte Dialog, mit dem die Parameterbilder gespeichert werden.

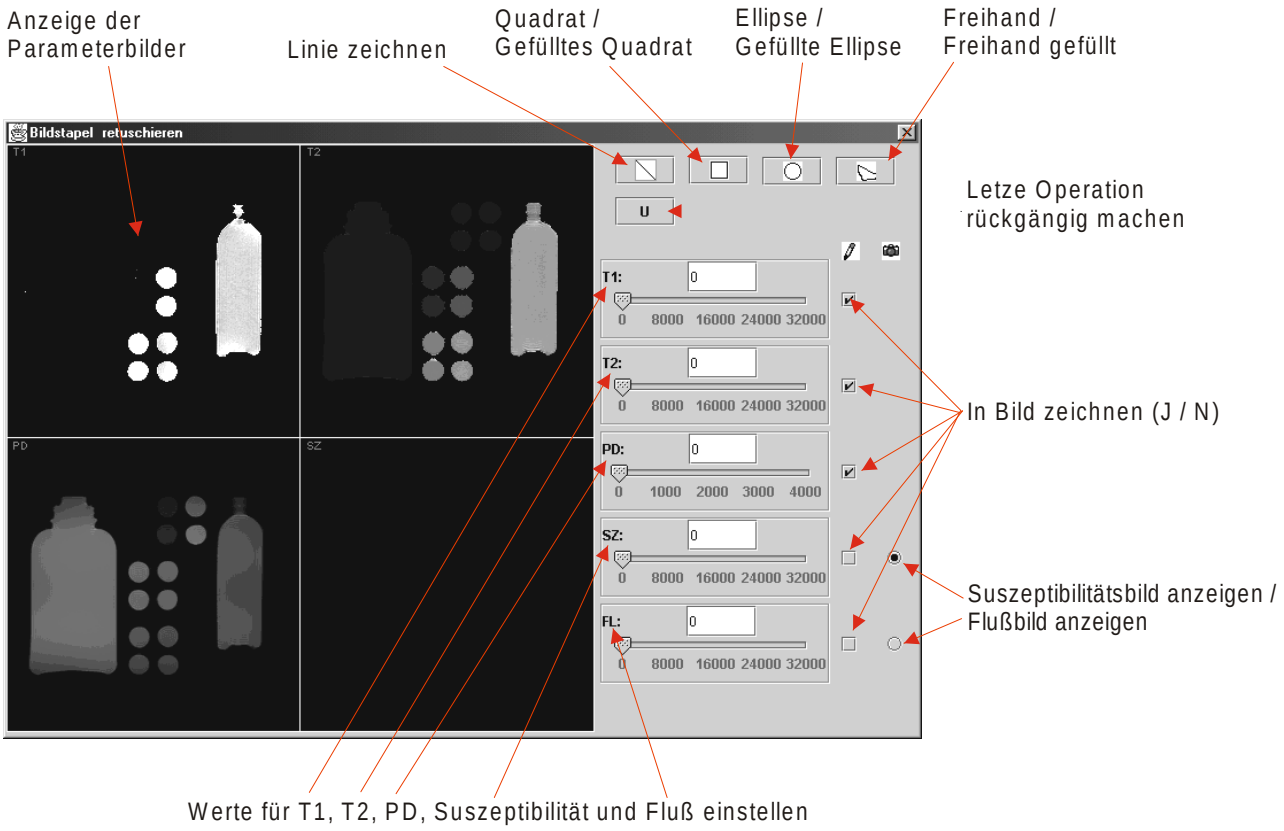

Abbildung 155: Nachbearbeiten der Parameterbilder

8 Zusammenfassung und Ausblick

Der Versuch, ein System zur Simulation der Bildgebung eines Kernspintomographen zu entwickeln, kann als gelungen bezeichnet werden und übertrifft sogar die anfangs von uns für realistisch gehaltenen Ziele.

Wir haben ein Werkzeug geschaffen, das es ermöglicht, aus einer speziellen Kernspinmessung einen Parameterdatensatz zu berechnen, der für die Bildberechnung verwendet werden kann. Dieser kann mittels eines weiteren Werkzeugs so manipuliert werden, daß auch künstliche Pathologien eingezeichnet werden können.

Das Meßwerkzeug *Virtual MRT* ermöglicht es schließlich, mittels der zuvor berechneten Paramterdatensätze und einer auszuwählenden Pulssequenz ein Intensitätsbild zu berechnen. Dieses Intensitätsbild wird in ein 12-Bit Bild konvertiert und in einer gefensterten Variante dargestellt. Die Parameter der gewählten Pulssequenz können wie beim realen Kernspintomographen verändert werden. Unser System läßt dem Benutzer dabei sogar Freiräume, die über die des realen Gerätes hinausgehen. Somit ist es möglich, extreme Einstellungen vorzunehmen und deren Auswirkung auf die Bildgebung zu beobachten.

In die Bildberechnung können darüber hinaus Artefakt-Simulationen einbezogen werden. Zum einen betrifft das das Bildrauschen, zum anderen können komplexe Artefakte wie Einfaltungen oder Bewegungen simuliert werden.

Mittels des Nachverarbeitungswerkzeugs *DICOM-Viewer* kann der Benutzer sich beispielsweise das Histogramm oder den k-Raum eines Bildes anzeigen lassen. Darüber hinaus stehen Werkzeuge zur Verfügung, um einige k-Raum-Manipulationen zu simulieren, eine Animation aus einer Bildserie zu erstellen, Bilder in einer anderen Projektionsrichtung zu berechnen und MIP-Bilder zu erzeugen.

Für die Zukunft ist es denkbar, daß System um neue Pulssequenzen und Artefakte (z.B. Flußartefakte oder Suszeptibilitätsartefakte) zu erweitern. In Kapitel 5.5 ist beschrieben, wie man dabei vorgehen muß.

Denkbar ist auch ein Export der berechneten Bilder in anderen Grafikformaten als dem DICOM-Format, so daß eine einfache Weiterbearbeitung mit Hilfe anderer Grafikprogramme stattfinden kann.

Eine Eigenschaft, die die Realitätsnähe der Simulation unetsrstützen würde, wäre auch, wenn die während einer Messung vom Kernspintomographen verursachten Geräusche simuliert werden könnten. Dies würde allerdings eine genaue Analyse erfordern, in welchem Zusammenhang die Geräusche mit den Pulssequenzen und deren Parametern stehen.

Interessant wäre es sicherlich, unser System um ein Online-Hilfesystem zu erweitern. Das würde unser Werkzeug für den Benutzer als Einstieg in die Materie der Kernspintomographie abrunden. In diesem Rahmen wäre auch die Einbindung von Fotos oder Filmen eines realen Kernspintomographen denkbar. Allerdings hätte dies den Rahmen dieser Arbeit bei weitem gesprengt. Bis zur Realisierung eines Hilfesystems steht dem Benutzer diese schriftliche Ausarbeitung zur Verfügung.

Zu Problemen kam es während der Entwicklung bei der Simulation von Einfaltungen in Phasenkodierrichtung. Hierbei verhielt sich die Simulation nicht, wie die Theorie dies vermuten ließ. Die Umsetzung der in Kapitel 3.5 beschriebenen Konzepte führte in diesem Fall nicht zum Erfolg. Wir vermuten nach Rücksprache mit Prof. Müller, daß das Problem darin begründet liegt, daß wir in der Simulation mit einem diskreten Fourierraum arbeiten, wohingegen die Datenerfassung am realen Gerät in einem kontinuierlichem Fourierraum stattfindet. Dies ist der einzige Unterschied zwischen unserer Simulation und der Realität.

Anhang A: Klassendokumentation

Im folgenden sind die wichtigsten Klassen unseres Projektes mit ihren Klassen- und Instanzvariablen, Konstruktoren und Methoden dokumentiert. Tabelle 9 gibt einen Überblick über alle von uns erstellten Klassen und die Markierung in der ersten Spalte gibt an, ob die entsprechende Klasse in diesem Kapitel dokumentiert ist. In der zweiten Spalte folgt der Klassenname, in der dritten Spalte ist angegeben, in welchem Paket sich die Klasse befindet und die vierte Spalte liefert eine Kurzbeschreibung der Klasse.

I	Klassenname	Paket	Kurzbeschreibung
✓	AnimationFrame	dicomviewer	Animations-Dialog
✓	Artefact	artefacts	Oberklasse für Artefakt-Simulatoren
	Artefact_Template	atefacts	Vorlage für neue Artefakt-Berechnungsklassen
✓	ArtefactsCombo	vmrt	Auswahlbox für die Artefakt-Simulatoren
✓	ArtefactUI	artefacts	Oberklasse für Oberflächenklassen für Artefakte
	ArtefactUI_Template	artefacts	Vorlage für neue Artefakt-Oberflächenklassen
	DcmDataDictionaryElement	dcm.dicom	Wörterbucheintrag für DICOM-Parameter
	DcmDataObject	dcm.dicom	DICOM-Datenobjekt
	DcmDED	dcm.dicom	Wörterbuch mit DICOM-Parametern
	DcmElements	dcm.dicom	Hashtabelle von DICOM-Datenelementen
	DcmEndian	dcm.dicom	Endian-Einstellung des DICOM-Datenobjekts
	DcmFileBuffer	dcm.dicom	Liest und schreibt DICOM-Dateien
	DcmFileManager	dcm.dicom	Liest und schreibt DICOM-Dateien
	DcmInterpreter	dcm.dicom	Wandelt einen Stream in ein DICOM-Datenobejekt
	DcmIOBlockBuffer	dcm.dicom	Blockpuffer für die Dateiein- und ausgabe
	DcmValue	dcm.dicom	Repräsentiert einen DICOM-Wert
✓	DicomViewer	dicomviewer	Startklasse des *DICOM-Viewers*
	ErrorMessage	tools	Fehlerdialog
✓	FFTTools	fft	Einige Funktionen im Zshg. mit der FFT
✓	FileLoader	vmrt	Lädt einen Rohdatensatz
	Flash	sequences	Flash Berechnungsklasse
	FlashUI	sequences	Flash Oberflächenklasse
	Histogramm	vmrt	Histogrammfenster
	IDXFile	tools	Dateifilder für *.idx*-Dateien
✓	ImagePlus	vmrt	Erweiterte Bildklasse (mit Zusatzinformationen)
✓	ImageStack	vmrt	Bildstapel (verwaltet alle Bilder)
	IntegerDocument	tools	Dokument das nur Ganzzahleingaben erlaubt
	InversionRecovery	sequences	Inversion-Recovery Berechnungsklasse
	InversionRecoveryUI	sequences	Inversion-Recovery Oberflächenklasse
	KSpaceCanvas	fft	Zeichenfläche für das k-Raum Anzeigefenster
✓	KSpaceFrame	fft	k-Raum-Anzeigefenster
✓	KSpaceManipulator	fft	k-Raum-Manipulatorfenster
	KSpaceManipulatorCanvas	fft	Zeichenfläche des k-Raum-Manipulators
	LabelTFLabelPanel	tools	2 Beschriftungen und 1 Textfeld im Rahmen
	LabTFLabTDPanel	tools	2 Beschriftungen und 2 Textfelder im Rahmen
	MIPParamDlg	dicomviewer	Parameterdialog zur Berechnung von MIP-Bildern
✓	Motion	artefacts	Translationsartefakt Berechnungsklasse
✓	MotionUI	artefacts	Translationsartefakt Oberflächenklasse
✓	MyPanel (dicomviewer)	dicomviewer	Zeichenfläche des *DICOM-Viewers*

✓	MyPanel (vmrt)	vmrt	Zeichenfläche des virtuellen MRT
	PeriodicMotion	artefacts	Periodische Bewegung Artefakt Berechnungsklasse
	PeriodicMotionUI	artefacts	Periodische Bewegung Artefakt Oberflächenklasse
	ProjParamDlg	dicomviewer	Parameterdialog zur Projektionsberechnung
✓	Pulsesequence	sequences	Oberklasse für Pulssequenzberechnungsklassen
✓	PulsesequenceUI	sequences	Oberklasse für Pulssequenzoberflächenklassen
✓	SaturationRecovery	sequences	Saturation-Recovery Berechnungsklasse
✓	SaturationRecoveryUI	sequences	Saturation-Recovery Oberflächenklasse
	Sequence_Template	sequences	Vorlage für Pulssequenzberechnungsklassen
✓	SequenceCombo	sequences	Auswahlbox für die Pulssequenzen
	SequenceUI_Template	sequences	Vorlage für Pulssequenzoberflächenklassen
✓	SeriesLoader	dicomviewer	Lädt eine DICOM-Bildserie
✓	SliderPanel	tools	Schieberegler mit Textfeld im Rahmen gekapselt
	SpinEcho	sequences	Spin-Echo Berechnungsklasse
	SpinEchoUI	sequences	Spin-Echo Oberflächenklasse
	ThreeLabelPanel	tools	Drei Beschriftungen im Rahmen gekapselt
	TurboSpinEcho	sequences	Turbo-Spin-Echo Berechnungsklasse
	TurboSpinEchoUI	sequences	Turbo-Spin-Echo Oberflächenklasse
✓	ViewerFrame	dicomviewer	*DICOM-Viewer* Hauptfenster
	ViewerFrame_AboutBox	dicomviewer	*DICOM-Viewer* Infofenster
✓	VMRT	vmrt	VMRT Startklasse
✓	VMRTFrame	vmrt	VMRT Hauptfenster
	VMRTFrame_AboutBox	vmrt	VMRT Infofenster

Tabelle 9: Implementierte Klassen

Eine umfassende Dokumentation aller Klassen im HTML-Format befindet sich auf der beiliegenden CD im Verzeichnis *JavaDoc*.

Da im Quelltext keine Umlaute und Sonderzeichen verwendet wurden, kommen diese auch nicht in der nun folgenden Klassendokumentation vor, da diese direkt aus dem Quelltext generiert wurde.

Klasse AnimationFrame

public class **AnimationFrame** extends JFrame implements java.lang.Runnable

Diese Klasse stellt das Fenster zur animierten Darstellung des geladenen Datensatzes dar. Es ermoeglicht, die Einzelbilder manuell oder automatisch nacheinander darzustellen. Dabei ist es auch moeglich, andere Schichtfuehrungen zu waehlen, als die durch den Originaldatensatz vorgegebene. Ausserdem koennen die Animationsgeschwindigkeit und die Bilder, die in die Animation mit einbezogen werden sollen, eingestellt werden. Die Klasse ist in einem Thread implementiert, damit sie die Anwendung nicht blockiert.

Klassen- und Instanzvariablen	
`private ImageIcon`	`iLoop` Icon fuer den Modusknopf 'Animation wiederholt abspielen'.
`private ImageIcon`	`iLoopBack` Icon fuer den Modusknopf 'Animation wiederholt abspielen (vor-zurueck)'.
`private ImageIcon`	`iOneWay` Icon fuer den Modusknopf 'Animation einfach abspielen'.
`private ImageIcon`	`iOrigProj` Icon fuer den Projektionsknopf 'Originalschichten'.
`private ImageIcon`	`iPause`

	Icon fuer den 'Pause'-Knopf.
private ImageIcon	iPlay Icon fuer den 'Play'-Knopf.
private ImageIcon	iProj1 Icon fuer den Projektionsknopf '90°-Projektion 1'.
private ImageIcon	iProj2 Icon fuer den Projektionsknopf '90°-Projektion 2'.
private ImageStack	is Der Stapel mit allen Schichtbildern der Originalschichtfuehrung.
private ImageIcon	iStop Icon fuer den 'Stop'-Knopf.
private JLabel	lAnimModus Beschriftung fuer den Animationsmodus.
private static int	LOOP Modus-Flag fuer wiederholtes Abspielen des Bildstapels.
private static int	LOOPBACK Modus-Flag fuer wiederholtes Abspielen des Bildstapels.
private JLabel	lProjection Beschriftung fuer den Projektionsmodus.
private JLabel	lSliderLabel Beschriftung fuer den Schieberegler zur manuellen Animationssteuerung.
private ViewerFrame	mainFrame Eine Rueckreferenz zum Hauptfenster, von dem aus die Animation aufgerufen wurde.
private static int	MANUAL Modus-Flag fuer die manuelle Steuerung der Animation.
private int	mode Der aktuell eingestellte Animationsmodus.
private java.awt.Image	myImage Das aktuelle Schichtbild, daß dargestellt werden soll.
private java.lang.Thread	myThread Der Thread, in dem die Animation ablaeuft.
private static int	ONE_TIME Modus-Flag fuer das einmalige Abspielen des Bildstapels.
private boolean	paintAll Ein Flag, dass anzeigt, ob auch die Bedienelemente des Fensters neu gezeichnet werden muessen.
private JButton	pbAnimStart Knopf zum Starten der Animation.
private JButton	pbAnimStop Knopf zum Stoppen der Animation.
private SliderPanel	pFps Schieberegler zur Einstellung der Animationsgeschwindigkeit.
private LabTFLabTFPanel	pImages Textfelder zur Einstellung, welche Bilder in die Animation einbezogen werden sollen.
private JPanel	pSliderPanel Rahmen fuer den Schieberegler zur manuellen Steuerung der Animation.
private int	sleep Zeitdauer, die ein Bild eingeblendet bleibt, bis das naechste folgt.
private JSlider	slImage Schieberegler zur manuellen Steuerung der Animation.
private int	startImage Das Bild, mit dem die Animation begonnen wird .
private int	stopImage Das Bild, bei dem die Animation endet.
private JToggleButton	tbAnimMode1 Knopf fuer den Animationsmodus 'Animation einfach abspielen'.
private JToggleButton	tbAnimMode2 Knopf fuer den Animationsmodus 'Animation wiederholt abspielen'.
private JToggleButton	tbAnimMode3 Knopf fuer den Animationsmodus 'Animation wiederholt abspielen (vor-zurueck)'.
private JToggleButton	tbAnimMode4 Knopf fuer den Animationsmodus 'manuelle Animationssteuerung'.

private JToggleButton	tbAnimPause Knopf zum Anhalten der Animation.
private JToggleButton	tbProj1 Knopf fuer den Projektionsmodus 'Originalschichten'.
private JToggleButton	tbProj2 Knopf fuer den Projektionsmodus '90°-Projektion 1'.
private JToggleButton	tbProj3 Knopf fuer den Projektionsmodus '90°-Projektion 2'.

Konstruktoren

`public` **`AnimationFrame`**`(java.awt.Frame fra)`

Standardkonstruktor. Er speichert eine Ruechreferenz auf das Hauptfenster des DICOM-Viewers. Außerdem wird die jbInit-Methode aufgerufen, die die Bedienelemente darstellt.

Parameter:

`fra` - Eine Rueckreferenz auf das aufrufende Fenster.

Methoden

`public void` **`start`**`()`

Die Method erzeugt einen neuen Thread, setzt dessen Prioritaet auf maximal und startet die Animation.

`public void` **`run`**`()`

Die Methode entscheidet aufgrund der aktuellen Einstellungen der Bedienelemente, welches Bild als naechstes dargestellt werden muss. Die Methode besorgt dieses Bild, ruft die repaint-Methode auf und wartet eine gewisse Zeit, bis schliesslich das naechte Bild berechnet und dargestellt wird.

`public void` **`stop`**`()`

Die Methode beendet den aktuellen Thread.

`private void` **`jbInit`**`()`
`throws java.lang.Exception`

Die Methode richtet die Bedienelemente ein und stellt diese dar.

`private void` **`pbAnimStart_actionPerformed`**`()`

Die Methode wird aufgerufen, wenn der Start-Knopf gedrueckt wurde. Sie liest die aktuellen Einstellungen der Bedienelemente aus, speichert sie in Instanzvariablen und sperrt anschliessend einige Bedienelemente. Dann wird die start-Methode aufgerufen und somit die Animation gestartet.

`private void` **`tbAnimPause_actionPerformed`**`()`

Die Methode wird aufgerufen, wenn der Pause-Knopf gedrueckt wurde. Die Methode entscheidet, ob die Pausefunktion nun aktiviert oder deaktiviert ist und versetzt dem entsprechend den Thread in den Schlaf oder erweckt ihn wieder zum Leben.

`private void` **`pbAnimStop_actionPerformed`**`()`

Die Methode wird aufgerufen, wenn der Stop-Knopf gedrueckt wurde. Die Animation wird gestoppt und die Bedienelemente werden wieder in den Bereitschaftszustand versetzt.

`public void` **`paint`**`(java.awt.Graphics g)`

Die Methode zeichnet das aktuell darzustellende Bild in die Zeichenflaeche ein und aktualisiert ggf. auch die Bedienelemente.

Parameter:
g - Der Graphikkontext.

`private void` **`this_componentMoved`**`(java.awt.event.ComponentEvent e)`
Die Methode wird aufgerufen, wenn des Animationsfenster bewegt wurde. Sie setzt dann ein Flag, so dass auch die Bedienelemente neu gezeichnet werden.

Parameter:
e - Das Ereignis, das bei Fensterbewegung ausgeloest wird.

`private void` **`this_windowActivated`**`(java.awt.event.WindowEvent e)`
Die Methode wird aufgerufen, wenn das Animationsfenster aktiviert wurde. Sie setzt dann ein Flag, so dass auch die Bedienelemente neu gezeichnet werden.

Parameter:
e - Das Ereignis, das bei Aktivierung des Fensters ausgeloest wird.

`private void` **`slImage_stateChanged`**`()`
Die Methode wird aufgerufen, wenn sich der Status des Schiebereglers zur manuellen Animationssteuerung aendert. In Abhaengigkeit von der Einstellung des Projektionsmodus wird dann das benoetigte Bild ausgelesen oder neu berechnet.

`private void` **`tbProj1_actionPerformed`**`()`
Die Methode wird aufgerufen, wenn der Projektionsmodus fuer die Originalschichtfuehrung ausgewaehlt wurde. Die Methode setzt nur die Beschriftung des Schiebereglers fuer die manuelle Animationssteuerung neu.

`private void` **`tbProj2_actionPerformed`**`()`
Die Methode wird aufgerufen, wenn der Projektionsmodus fuer die erste 90°-Projektion ausgewaehlt wurde. Die Methode setzt nur die Beschriftung des Schiebereglers fuer die manuelle Animationssteuerung neu.

`private void` **`tbProj3_actionPerformed`**`()`
Die Methode wird aufgerufen, wenn der Projektionsmodus fuer die zweite 90°-Projektion ausgewaehlt wurde. Die Methode setzt nur die Beschriftung des Schiebereglers fuer die manuelle Animationssteuerung neu.

Klasse Artefact

public abstract class **Artefact** extends java.lang.Thread

Diese Klasse ist die abstrakte Oberklasse fuer alle Berechnungsklassen fuer Artefakt-Simulatoren. Sie stellt insbesondere eine Referenz auf die aufrufende Pulssequenz zur Verfuegung und speichert das von der Pulssequenz erzeugte Intensitaetsbild.

Klassen- und Instanzvariablen	
`protected int[][]`	`intensityimg` Das von der aufrufenden Pulssequenz erzeugte Intensitaetsbild, in dem Artefakte simuliert werden sollen.
`protected JProgressBar`	`progressBar` Referenz auf die Fortschrittsanzeige im VMRT-Fenster.
`protected Pulsesequence`	`sequence` Referenz auf die aufrufende Pulssequenz.

Konstruktoren

`public` **`Artefact`**`()`
Standardkonstruktor.

Methoden

`public abstract int[][]` **`calculate`**`(java.lang.String method)`
Diese Methode wird von der aufrufenden Pulssequenz immer zuerst aufgerufen. Der uebergebene Methodenname gibt an, welche Methode die korrekte Berechnungsvorschrift enthaelt. Die Methode weist in dieser Oberklasse allerdings keine Funktionalitaet auf.

Parameter:

`method` - Der Name der Methode, die die korrekte Berechnungsvorschrift fuer das Artefakt enthaelt.Rückgabewert:
Das manipulierte Intensitaetsbild.

`public void` **`setIntensityImage`**`(int[][] intimg)`
Die Methode wird von der aufrufenden Pulssequenz aufgerufen und setzt das von der Pulssequenz berechnete Intensitaetsbild.

Parameter:
`intimg` - Das zu manipulierende Intensitaetsbild.

`public void` **`setSequence`**`(Pulsesequence seq)`
Die Methode wird von der aufrufenden Pulssequenz aufgerufen und setzt eine Referenz auf selbige.

Parameter:
`seq` - Referenz auf die aufrufende Pulssequenz.

`public void` **`setProgressBar`**`(JProgressBar pb)`
Die Methode wird von der aufrufenden Pulssequenz aufgerufen und setzt eine Referenz auf die Fortschrittsanzeige im VMRT-Fenster.

Parameter:
`pb` - Referenz auf die Fortschrittsanzeige im VMRT-Fenster.

Klasse ArtefactsCombo

public class **ArtefactsCombo** extends JComboBox

Die Klasse verwaltet die Auswahlbox, zur Auswahl der Artefakt-Simulatoren. Die Methode *getArtefacts* liest die Artefaktklassen und Artefaktnamen aus der *Property*-Datei aus und erzeugt ueber *Reflections* Instanzen der *ArtefactUI*-Klassen. Auch die Ereignisbehandlung der Auswahlbox findet in dieser Klasse statt. Beim Wechseln des Artefakt-Simulators werden die eingestellten Werte ueber *setValue* in der *ArtefactUI*-Klasse gespeichert und ueber *getValue* wiederhergestellt.

Klassen- und Instanzvariablen

`private java.util.Vector`	`artefactClasses` Vektor zur Aufnahme der Artefakt-Klassen.
`private VMRTFrame`	`mainFrame` Referenz auf das Hauptfenster.
`private int`	`oldindex`
`private JPanel`	`pArtefactUIPanel`

	Referenz auf den Bereich, in dem die GUI-Elemente fuer die Artefaktsimulation dargestellt werden koennen.

Konstruktoren

`public` **`ArtefactsCombo`**`(VMRTFrame frame, JPanel seqSetPanel)`

Der Konstruktor setzt die Referenzen zum Hauptfenster und zum Panel, in dem die GUI-Elemente der Artefaktsimulation dargestellt werden koennen. Ausserdem wird ein leerer Vektor zur Aufnahme der Artefaktsimulatoren angelegt.

Parameter:

`frame` - Rueckreferenz zum Hauptfenster.

`seqSetpanel` - Referenz auf den Darstellungsbereich fuer die Bedienelemente der Artefakt-Simulatoren.

Methoden

`public void` **`getArtefacts`**`()`

Die Methode laedt die Artefaktklassenamen und die Artefaktnamen aus der Property-Datei. Eine Instanz des Artefakts und dessen GUI-Klasse wird ueber den Namen des Artefakts in der Auswahlliste erzeugt. Eine Referenz auf die ArtefactUI-Klasse wird im Vektor artefactClasses gespeichert. Dieses Konzept ermoeglicht es, dem Programm ganz einfach neue Artefakt-Simulatoren hinzuzufuegen. Fuer jedes Artefakt muessen zwei Klassen implementiert werden, eine UI-Klasse und eine Berechnungsklasse, anschliessend muessen Name des Artefakts und Klassenname der Berechnungsklasse in die Property-Datei eingetragen werden.

`private void` **`cb_itemStateChanged`**`(java.awt.event.ItemEvent e)`

Die Methode wird aufgerufen, wenn ein neuer Artefakt-Simulator aus der Auswahlbox ausgewaehlt wird. Der Bereich zur Darstellung der Oberflaechenelemente des Artefakt-Simulators wird dann geloescht und anschliessend mit den Elementen des neuen Simulators neu gefuellt.

Parameter:

`e` - Das Ereignis beim Auswaehlen eines anderen Artefakt-Simulators aus der Auswahlbox.

`public ArtefactUI` **`getSelectedArtefactUI`**`()`

Die Methode liefert eine Referenz auf die UI-Klasse des in der Auswahlbox selektierten Artefakt-Simulators.Rückgabewert:

Referenz auf die UI-Klasse des ausgewählten Artefakts.

Klasse ArtefactUI

public abstract class **ArtefactUI** extends java.lang.Object

Diese Klasse ist die abstrakte Oberklasse fuer alle Oberflaechenklassen fuer Artefakt-Simulatoren. Sie stellt insbesondere eine Referenz auf den Bereich zur Verfuegung, in den die Oberflaechenelemente hineingezeichnet werden koennen.

Klassen- und Instanzvariablen

`private boolean`	`isInitialized` Die Variable merkt sich, ob die Ereignisbehandlungsroutinen der Bedienelemente schon initialisiert wurden.
`VMRTFrame`	`mainFrame` Referenz auf das Hauptfenster.
`protected JPanel`	`myPanel` Referenz auf den Bereich, in dem die Oberflaechenelemente des Artefakt-Simulators dargestellt werden koennen.

Konstruktoren

`public` **`ArtefactUI`**`()`
Standardkonstruktor.

Methoden

`public void` **`setUIPanel`**`(JPanel p)`
Die Methode setzt die Referenz auf den Bereich, in dem die Oberflaechenelemente des Artefakt-Simulators dargestellt werden koennen.

Parameter:
`p` - Referenz auf den Bereich zur Darstellung der Bedienelemente.

`public void` **`setMainFrame`**`(VMRTFrame vmrt)`
Die Methode setzt die Referenz auf das Hauptfenster.

Parameter:
`vmrt` - Die Referenz auf das Hauptfenster.

`public abstract void` **`getValues`**`()`
Die Methode setzt die Werte der Oberflaechenelemente auf die in den Klassenvariablen gespeicherten Werte. Dies wird beim Umschalten zwischen Artefakt-Simulatoren benötigt, um die eingestellten Werte nicht zu verlieren. In dieser Oberklasse hat die Methode allerdings keine Funktionalität.

`public abstract void` **`setValues`**`()`
Die Methode liest die aktuellen Parameterwerte aus den Bedienelementen aus und speichert sie in den entsprechenden Klassenvariablen. In dieser Oberklasse weist die Methode allerdings keine Funktionalitaet auf.

`public void` **`fillPanel`**`()`
Die Methode stellt die Bedienelemente des Artefakt-Simulators im dafuer vorgesehenen Bereich dar.

`void` **`jbInit`**`()`
`throws java.lang.Exception`
Die Methode ist in dieser abstrakten Oberklasse leer. Sie wird in den abgeleiteten Klassen dazu verwendet, um weitere Bedien- und Informationselemente einzurichten.

`public abstract Artefact` **`createManipulator`**`()`
Die Methode erzeugt eine neue Instanz der Berechnungsklasse des Artefakt-Simulators und liefert diese zurueck. In dieser Oberklasse hat die Methode allerdings keine Funktionalitaet.

Klasse DicomViewer

public class **DicomViewer** extends java.lang.Object

Die Klasse ist die Hauptklasse des *DICOM-Viewers*. Sie wird nur zum Starten benoetigt, indem sie eine Instanz von *DicomViewer* erzeugt und deren *init*-Methode aufruft. Dort wird dann eine Instanz von *ViewerFrame* erzeugt, wodurch das *DICOM-Viewer*-Fenster dargestellt wird.

Klassen- und Instanzvariablen

`private ViewerFrame`	`frame`

	Referenz auf die Fensterklasse.
private boolean	packFrame

Konstruktoren

`public` **`DicomViewer`**`()`

Standardkonstruktor. Es wird eine Instanz von ViewerFrame erzeugt. Dadurch wird das Hauptfenster des DICOM-Viewers dargestellt.

Methoden

`public static void` **`main`**`(java.lang.String[] args)`

In dieser Methode wird das Look&Feel gesetzt. Es wird das Systemtypische Look&Feel verwendet. Ausserdem wird eine Instanz dieser Klasse instanziiert und automatisch gestartet.

Parameter:

`args` - Befehlszeilenargumente. Sie werden hier einfach ignoriert.

Klasse FFTTools

public class **FFTTools** extends java.lang.Object

Die Klasse kapselt einige Funktionen, die zur Berechnung der Fouriertransformation eines Bildes nuetzlich sind.

Konstruktoren

`public` **`FFTTools`**`()`

Standardkonstruktor.

Methoden

`private static int[]` **`getShiftedFFTImageSource`**`(float[][] imageReal, float[][] imageImag)`

Die Methode berechnet aus dem gegebenen Real- und Imaginaerteil der FFT eines Bildes das Magnitudenbild der Fouriertransformation.

Parameter:

`imageReal` - Der Realteil der FFT des Ursprungsbildes.
`imageImag` - Der Imaginaerteil der FFT des Ursprungsbildes.

Rückgabewert:

Ein eindimensionales Feld von Ganzzahlen, das zur Erzeugung einer MemoryImageSource benutzt werden kann.

`public static int[]` **`getShiftedFFTImageSource`**`(ImagePlus ip)`

Die Methode berechnet aus einem ImagePlus-Objekt das Magnitudenbild der Fouriertransformation.

Parameter:

`ip` - das Ursprungsbild.

Rückgabewert:

Ein eindimensionales Feld von Ganzzahlen, das zur Erzeugung einer MemoryImageSource benutzt werden kann.

`public static int[]` **`getReshiftedIFFTImageSource`**`(float[][] imgReal, float[][] imgImag)`

Die Methode berechnet die Ruecktransformation eines fouriertransformierten Bildes.

Parameter:
`imgReal` - Der Reateil der Fouriertransformation.
`imgImag` - Der Imaginaerteil der Fouriertransformation.

Rückgabewert:
Ein eindimensionales Feld von Ganzzahlen, das zur Erzeugung einer MemoryImageSource benutzt werden kann.

```
public static float[][] shift(float[][] img,
                int offx,
                int offy)
```

Die Methode verschiebt ein Bild um den angegebenen Betrag nach rechts und nach unten.

Parameter:
`img` - Das zu verschiebende Bild.
`offx` - Der Betrag, um den das Bild nach rechts verschoben werden soll.
`offy` - Der Betrag, um den das Bild nach unten verschoben werden soll.

Rückgabewert:
Das verschobene Bild.

```
public static float[][] magnitude(float[][] re,
                  float[][] im)
```

Die Methode berechnet aus dem Real- und Imaginaerteil eines fouriertransformierten Bildes das Magnitudenbild.

Parameter:
`re` - Der Realteil des fouriertransformierten Bildes.
`im` - Der Imaginaerteil des fouriertransformierten Bildes.

Rückgabewert:
Das Magnitudenbild.

```
public static float[][] calcPhase(float[][] re,
                  float[][] im)
```

Die Methode berechnet aus dem Real- und Imaginaerteil eines fouriertransformierten Bildes das Phasenbild.

Parameter:
`re` - Der Realteil des fouriertransformierten Bildes.
`im` - Der Imaginaerteil des fouriertransformierten Bildes.

Rückgabewert:
Das Phasenbild.

```
public static float[][] scaleMagnitude(float[][] img)
```

Die Methode skaliert die Grauwerte eine Bildes logarithmisch. Dies wird zur Anzeige des Magnitudenbildes einer Fouriertransformation benoetigt.

Parameter:
`img` - Das Bild, dessen Grauwerte log. skaliert werden sollen.

Rückgabewert:
Das Bild, dessen Grauwerte log. skaliert wurden.

```
public static int[][] autoWindow(float[][] img)
```

Die Methode fenstert ein Bild so, dass alle Grauwerte dargestellt werden.

Parameter:
`Das` - zu fensternde Bild.

Rückgabewert:
Das automatisch gefensterte Bild.

`public static int[] convertIntArrayToImage(int[][] source)`

Die Methode konvertiert ein zweidimensionales Feld von Ganzzahlen in ein eindimensionales Feld von Ganzzahlen, aus dem ein Grauwertbild mittels einer MemoryImageSource erstellt werden kann.

Parameter:

`source` - Das zu konvertierende zweidimensionale Feld.

Rückgabewert:

Das Feld, welches zur Erzeugung einer MemoryImageSource dienen kann.

`public static float[][][] getFFT(float[][] myFloatImage)`

Die Methode berechnet die Fouriertransformation eines Bildes und liefert diese als 3-dimensionales Feld zurueck.

Parameter:

`myFloatImage` - Das Bild, dessen FFT berechnet werden soll.

Rückgabewert:

Ein dredimensionales Feld von Ganzzahlen. returnImg[0] ist der Realteil, returnImg[1] der Imaginaerteil. die 2. Komponente des Feldes repraesentiert die x-Koordinate, die 3. Komponente die y-Koordinate.

Klasse FileLoader

public class **FileLoader** extends java.lang.Object

Die Klasse laedt einen Rohdatensatz aus Dateien. Insgesamt werden 6 Dateien verwendet. Die Indexdatei enthaelt die Namen der 5 anderen Dateien. Diese repraesentieren jeweils eine der folgenden Matrizen:

T1: Matrix mit den T_1-Relaxationszeiten
T2: Matrix mit den T_2-Relaxationszeiten
PD: Matrix mit den Protonendichtewerten
SZ: Matrix mit den Suszeptibilitaetswerten
FL: Matrix mit den Flusswerten

Die geladenen Daten werden anschliessend in 5 Matrizen zur Verfuegung gestellt.

Klassen- und Instanzvariablen	
`private DcmDataObject`	`ddo` Das DcmDataObject (enthaelt Bildparameter, jedoch nicht mehr die Pixeldaten)
`private int[][]`	`FLMatrix` Matrix mit den Flusswerten.
`private int[][]`	`PDMatrix` Matrix mit den Protonendichtewerten.
`private int[][]`	`SZMatrix` Matrix mit den Suszeptibilitaetswerten.
`private int[][]`	`T1Matrix` Matrix mit den T1-Werten.
`private int[][]`	`T2Matrix` Matrix mit den T2-Werten.

Konstruktoren

`public FileLoader()`

Standardkonstruktor.

Methoden

`public int[][] **getT1Matrix**()`
Die Methode liefert die Matrix mit den T1-Werten zurueck.

Rückgabewert:
Die Matrix mit den T1-Werten.

`public int[][] **getT2Matrix**()`
Die Methode liefert die Matrix mit den T2-Werten zurueck.

Rückgabewert:
Die Matrix mit den T2-Werten.

`public int[][] **getPDMatrix**()`
Die Methode liefert die Matrix mit den PD-Werten zurueck.

Rückgabewert:
Die Matrix mit den PD-Werten.

`public int[][] **getSZMatrix**()`
Die Methode liefert die Matrix mit den Suszeptibilitaetswerten zurueck.

Rückgabewert:
Die Matrix mit den Suszeptibilitaetswerten.

`public int[][] **getFLMatrix**()`
Die Methode liefert die Matrix mit den Flusswerten zurueck.

Rückgabewert:
Die Matrix mit den Flusswerten.

`public DcmDataObject **getDcmDataObject**()`
Die Methode liefert das DcmDataObject zurueck. Dieses enthaelt allerdings keine Pixel-Informationen mehr. Es dient lediglich dazu, die zusaetzlichen Bildinformationen zu speichern. Somit koennen neu erzeugte Bilder mit den gleichen Informationen versehen und im DICOM-Format abgespeichert werden.

Rückgabewert:
Das DicomDataObject

```
public void loadData(java.lang.String dir,
                     java.lang.String file)
```
Die Methode laedt die Rohdaten aus einer Datei. Zunaechst werden aus der uebergebenen Indexdatei die 5 anderen Dateinamen ausgelesen. Danach werden aus diesen Dateien die Rohdatenmatrizen gelesen.

Parameter:
`dir` - Pfad der Indexdatei.
`file` - Dateiname der Indexdatei.

```
private void loadImage(java.lang.String type,
                       java.lang.String filename,
                       java.lang.String path)
```
Die Methode laedt eines der Rohdatenbilder. Welches Bild geladen wird, wird durch einen Parameter bestimmt.

Parameter:
`type` - Gibt an, welches Rohdatenbild geladen werden soll. Moegliche Werte sind "T1", "T2", "PD", "SZ" und "FL".
`filename` - Der Dateiname des zu ladenden Bildes.

`path` - Das Verzeichnis, in dem sich das zu ladende Rohdatenbild befindet.

`public void` **`setReferenceData`**`()`

Die Methode belegt die Matrizen fuer die T1-, T2- und PD-Werte mit einigen Referenz-Werten. Anhand dieses Referenzbildes, wird die Funktionsweise des virtuellen Tomographen in der schriftlichen Ausarbeitung der Diplomarbeit erlaeutert.

Klasse ImagePlus

public class **ImagePlus** extends java.lang.Object

Die Klasse *ImagePlus* kapselt ein *DcmImage*-Objekt, welches ein 12-Bit DICOM-Bild repraesentiert, inklusive des gefensterten AWT-Bildes, des *DcmDataObjects*, welches alle Bildparameter enthaelt, und einiger anderer Informationen.

Klassen- und Instanzvariablen

`private dcm.DcmImage`	`DcmImg` Das DcmImage-Objekt.
`private DcmDataObject`	`ddo` Das DcmDataObject, welches alle Bildparameter enthaelt.
`private float[][]`	`fftImageImag` Der Imaginaerteil der Fouriertransformation des Bildes.
`private float[][]`	`fftImageReal` Der Realteil der Fouriertransformation des Bildes.
`private boolean`	`fftOK` Markierung die angibt, ob die zur Verfuegung stehende Fouriertransformation das Bildes aktuell ist.
`private java.awt.Image`	`img` Das gefensterte AWT-Bild.
`private int`	`size` Die Zeilen- und Spaltenaufloesung des gespeicherten Bildes.
`private java.lang.String[]`	`textNE` Ein Feld von Texten, die oben rechts im Bild eingeblendet werden koennen.
`private java.lang.String[]`	`textNW` Ein Feld von Texten, die oben links im Bild eingeblendet werden koennen.
`private java.lang.String[]`	`textSE` Ein Feld von Texten, die unten rechts im Bild eingeblendet werden koennen.
`private java.lang.String[]`	`textSW` Ein Feld von Texten, die unten links im Bild eingeblendet werden koennen.

Konstruktoren

`public` **`ImagePlus`**`()`

Standardkonstruktor

`public` **`ImagePlus`**`(dcm.DcmImage di)`

Dieser Konstruktor erzeugt ein ImagePlus-Objekt aus einem DcmImage-Objekt. Dabei wird gleichzeitig die Aufloesung des Bildes ermittelt.

Parameter:

`di` - Das DcmImage-Objeckt, aus dem das ImagePlus-Objekt erzeugt wird.

Methoden

`public java.lang.String[]` **`getTextNW`**`()`

Die Methode liefert ein Feld von Texten, die oben links im Bild eingeblendet werden koennen. Die Auswahl der Texte,

die dort erscheinen, findet an dieser Stelle statt. Die Klasse myPanel, die die Hauptzeichenflaeche repraesentiert, stellt diese Texte ohne weitere Beachtung des Inhalts lediglich dar.

Rückgabewert:
Die Zeichenketten, die oben links im Bild dargestellt werden sollen.

`public java.lang.String[] getTextNE()`

Die Methode liefert ein Feld von Texten, die oben rechts im Bild eingeblendet werden koennen. Die Auswahl der Texte, die dort erscheinen, findet an dieser Stelle statt. Die Klasse myPanel, die die Hauptzeichenflaeche repraesentiert, stellt diese Texte ohne weitere Beachtung des Inhalts lediglich dar.

Rückgabewert:
Die Zeichenketten, die oben rechts im Bild dargestellt werden sollen.

`public java.lang.String[] getTextSW()`

Die Methode liefert ein Feld von Texten, die unten links im Bild eingeblendet werden koennen. Die Auswahl der Texte, die dort erscheinen, findet an dieser Stelle statt. Die Klasse myPanel, die die Hauptzeichenflaeche repraesentiert, stellt diese Texte ohne weitere Beachtung des Inhalts lediglich dar.

Rückgabewert:
Die Zeichenketten, die unten links im Bild dargestellt werden sollen.

`public java.lang.String[] getTextSE()`

Die Methode liefert ein Feld von Texten, die unten rechts im Bild eingeblendet werden koennen. Die Auswahl der Texte, die dort erscheinen, findet an dieser Stelle statt. Die Klasse myPanel, die die Hauptzeichenflaeche repraesentiert, stellt diese Texte ohne weitere Beachtung des Inhalts lediglich dar.

Rückgabewert:
Die Zeichenketten, die unten rechts im Bild dargestellt werden sollen.

`public java.lang.String getSequence()`

Die Methode liefert den Namen der Pulssequenz zurueck, mit der das Bild aufgenommen wurde.

Rückgabewert:
Der Name der Pulssequenz.

`public java.lang.String getTR()`

Die Methode liefert die Repititionszeit zurueck, mit der das Bild aufgenommen wurde.

Rückgabewert:
Die Repititionszeit als Zeichenkette.

`public java.lang.String getTI()`

Die Methode liefert die Inversionszeit zurueck, mit der das Bild aufgenommen wurde. Wurde das Bild mit einer Sequenz aufgenommen, die diesen Parameter nicht beinhaltet, ist der Rueckgabewert leer.

Rückgabewert:
Die Inversionszeit als Zeichenkette oder eine leere Zeichenkette.

`public java.lang.String getTE()`

Die Methode liefert die Echozeit zurueck, mit der das Bild aufgenommen wurde. Wurde das Bild mit einer Sequenz aufgenommen, die diesen Parameter nicht beinhaltet, ist der Rueckgabewert leer.

Rückgabewert:
Die Echozeit als Zeichenkette oder eine leere Zeichenkette.

`public java.lang.String getETL()`

Die Methode liefert die Anzahl der Echos pro Repititionszyklus (Echo Train Length) zurueck, mit der das Bild aufgenommen wurde. Wurde das Bild mit einer Sequenz aufgenommen, die diesen Parameter nicht beinhaltet, ist der Rueckgabewert leer.

Rückgabewert:
Die Anzahl der Echos pro Repititionszyklus als Zeichenkette oder eine leere Zeichenkette.

`public java.lang.String` **`getSliceLocation`**`()`

Die Methode liefert die Position des Bildes (der Schicht) im Gesamtdatensatz zurueck. Die Position haengt von den Einstellungen des Koordinatensystems bei der Aufnahme ab.

Rückgabewert:
Die Schichtposition als Zeichenkette.

`public java.lang.String` **`getFA`**`()`

Die Methode liefert den Flipwinkel zurueck, mit dem das Bild aufgenommen wurde. Wurde das Bild mit einer Sequenz aufgenommen, die diesen Parameter nicht beinhaltet, ist der Rueckgabewert leer.

Rückgabewert:
Der Flipwinkel als Zeichenkette oder eine leere Zeichenkette.

`public java.lang.String` **`getPatientName`**`()`

Die Methode liest den Namen des Patienten, der auf dem Bild dargestellt ist, aus dem DcmDataObject aus und liefert ihn zurueck.

Rückgabewert:
Der Patientenname als Zeichenkette.

`public java.util.Date` **`getPatientBirthdate`**`()`

Die Methode liest das Geburtsdateum des Patienten, der auf dem Bild dargestellt ist, aus dem DcmDataObject aus und liefert es zurueck.

Rückgabewert:
Das Geburtsdatum des Patienten als Datums-Objekt.

`public java.lang.String` **`getPatientSex`**`()`

Die Methode liest das Geschlecht des Patienten, der auf dem Bild dargestellt ist, aus dem DcmDataObject aus und liefert es zurueck.

Rückgabewert:
Das Geschlecht des Patienten als Zeichenkette.

`public double` **`getPixelSpacing`**`()`

Die Methode liest den Pixelabstand des Bildes aus dem DcmDataObject aus und liefert ihn zurueck. Dieser wird benoetigt, um Distanzmessungen durchfuehren zu koennen.

Rückgabewert:
Der Pixelabstand.

`public java.util.Date` **`getDate`**`()`

Die Methode liest das Aufnahmedatum und die Aufnahmezeit des Bildes aus dem DcmDataObject aus und liefert es zurueck.

Rückgabewert:
Das Aufnahmedatum und die Aufnahmezeit des Bildes als Datums-Objekt.

`public int` **`getCenter`**`()`

Die Methode liest das Zentrum der Fensterung aus dem DcmDataObject aus und liefert es zurueck.

Rückgabewert:
Das Zentrum der Fensterung als Ganzzahlenwert.

`public int` **`getWindow`**`()`
Die Methode liest die Breite des Fensterungsbereichs aus dem DcmDataObject aus und liefert sie zurueck.

Rückgabewert:
Die Breite des Fensterungsbereichs als Ganzzahlenwert.

`public void` **`setSequence`**`(java.lang.String s)`
Die Methode setzt den Namen der Aufnahmesequenz im DcmDataObject.

Parameter:
`s` - Der Name der Pulssequenz, mit der das Bild aufgenommen wurde.

`public void` **`setTR`**`(double tr)`
Die Methode setzt die Repititionszeit im DcmDataObject.

Parameter:
`tr` - Die Repititionszeit [ms], mit der das Bild aufgenommen wurde.

`public void` **`setTI`**`(double ti)`
Die Methode setzt die Inversionszeit im DcmDataObject.

Parameter:
`ti` - Die Inversionszeit [ms], mit der das Bild aufgenommen wurde.

`public void` **`setTE`**`(double te)`
Die Methode setzt die Echozeit im DcmDataObject.

Parameter:
`te` - Die Echozeit [ms], mit der das Bild aufgenommen wurde.

`public void` **`setETL`**`(int etl)`
Die Methode setzt die Anzahl der Echos pro Repititionszyklus (Echo Train Length) im DcmDataObject.

Parameter:
`etl` - Die Anzahl der Echos pro Repititionszyklus, mit der das Bild aufgenommen wurde.

`public void` **`setESP`**`(double esp)`
Die Methode setzt den zeitlichen Abstand der Echos im DcmDataObject.

Parameter:
`esp` - Der zeitliche Abstand [ms] der Echos.

`public void` **`setSliceLocation`**`(double sl)`
Die Methodes setzt die Schichtposition im DcmDataObject. Die Position haengt von der Wahl des Koordinatensystems bei der Aufnahme ab.

Parameter:
`sl` - Die Schichtposition des aktuellen Bildes.

`public void` **`setFA`**`(double fa)`

Die Methode setzt den Flipwinkel im DcmDataObject.

Parameter:
`fa` - Der Flipwinkel, mit dem das Bild aufgenommen wurde.

`public void` **`setPatientName`**`(java.lang.String name)`
Die Methode setzt den Patiemtennamen im DcmDataObject.

Parameter:
`name` - Der Name des Patienten, der auf dem Bild dargestellt ist.

`public void` **`setPatientSex`**`(java.lang.String s)`
Die Methode setzt das Geschlecht des Patienten im DcmDataObject.

Parameter:
`s` - Das Geschlecht des Patienten, der auf dem Bild dargestellt ist.

`public void` **`setDDO`**`(DcmDataObject d)`
Die Methode fuegt dem ImagPlus-Objekt ein DcmDataObject hinzu. Darin sind alle Bild- und Patienteninformationen sowie weitere Informationen enthalten. Das DcmDataObject wird benoetigt, um beim Abspeichern des Bildes all diese Informationen ebenfalls abspeichern zu koennen. Wuerde man sich das Objekt nicht merken, wuerden zwischen Speichern und Laden eines Bildes alle darin enthaltenen Informationen verloren gehen. Da das DcmDataObject urspruenglich auch die Pixeldaten enthaelt, diese jedoch schon im DcmImage gespeichert werden, werden die Pixeldaten aus dem DcmDataObject geloescht.

Parameter:
`d` - Das DcmDataObject, das alle wichtigen Bildinformationen enthaelt.

`public DcmDataObject` **`getDDO`**`()`
Die Methode liefert das DcmDataObject mit allen Bildinformationen zurueck.

Rückgabewert:
Das DcmDataObject zum aktuellen Bild.

`public java.awt.Image` **`getAWTImage`**`()`
Die Methode liefert das gefensterte Bild als Java-AWT-Bild zurueck.

Rückgabewert:
Das gefensterte Java-Bild.

`public int` **`getGrayValue`**`(int x,`
`int y)`
Die Methode liefert den Grauwert des 12-Bit Bildes an der angegebenen Position zurueck.

Parameter:
`x` - x-Koordinate des zu ermittelnden Grauwerts.
`y` - y-Koordinate des zu ermittelnden Grauwerts.

Rückgabewert:
Der Grauwert an der Position (x,y).

`public short[][]` **`get8BitImageShort`**`()`
Die Methode liefert das gefensterte AWT-Bild als zweidimensionales Feld von 16-Bit-Zahlen zurueck. Benoetigt wird diese Methode zur Berechnung eines Histogramms. return Das gefensterte AWT-Bild als zweidimensionales Feld von 16-Bit-Zahlen.

```
public java.awt.Image getImagePart(int x,
                   int y,
                   int w,
                   int h)
```

Die Methode liefert den angegebenen Teil des gefensterten (8-Bit) Bildes zurueck. Das Ergebnis wird benoetigt, um einen Ausschnitt des Bildes vergroessern zu koennen (Lupe).

Parameter:

x - Die linke x-Koordinate des Ausschnitts.
y - Die obere y-Koordinate des Ausschnitts.
w - Die Breite des Ausschnitts.
h - Die Hoehe des Ausschnitts.

```
public void mirrorHor()
```

Die Methode spiegelt das 12-Bit Bild. Da dabei keine Bildinformationen verloren gehen, wird die Operation direkt auf dem 12-Bit-Bild im DcmImage-Objekt durchgefuehrt. Nach dem Spiegeln wird auch das 8-Bit Java-AWT-Bild neu berechnet.

```
public void updateAWTImage()
```

Die Methode berechnet das 8-Bit Java-AWT-Bild auf Grundlage des 12-Bit Bildes (im DcmImage-Objekt) und den aktuellen Fensterungseinstellungen neu.

```
public void rotate90CW()
```

Die Methode rotiert das 12-Bit Bild um 90 Grad im Uhrzeigersinn. Da dabei keine Bildinformationen verloren gehen, wird die Operation direkt auf dem 12-Bit-Bild im DcmImage-Objekt durchgefuehrt. Nach dem Rotieren wird auch das 8-Bit Java-AWT-Bild neu berechnet.

```
public int getSize()
```

Die Methode liefert die Zeilen- bzw. Spaltenaufloesung des Bildes zurueck.

Rückgabewert:

Die Aufloesung des Bildes.

```
public void invert()
```

Die Methode invertiert das 12-Bit Bild. Da dabei keine Bildinformationen verloren gehen, wird die Operation direkt auf dem 12-Bit-Bild im DcmImage-Objekt durchgefuehrt. Nach dem Invertieren wird auch das 8-Bit Java-AWT-Bild neu berechnet.

```
public short[] getImage12BitShort()
```

Die Methode liefert das 12-Bit-Bild als Feld von 16-Bit-Zahlen zurueck.

Rückgabewert:

Das Bild als Feld von 16-Bit-Zahlen.

```
public void setCurrentDate()
```

Die Methode setzt das Datum des Aufrufs dieser Methode im DcmDataObject.

```
public void setCW(int c,
          int w)
```

Die Methode aktualiert die Einstellungen fuer die Fensterung im DcmDataObject. Nach der Aktualisierung wird auch das 8-Bit-Bild neu berechnet.

Parameter:

c - Das Zentrum des darzustellenden Fensters.
w - Die Breite des darzustellenden Fensters.

`public void` **`setCenter`**`(int c)`

Die Methode aktualiert die Einstellungen fuer die Fensterung im DcmDataObject. Nach der Aktualisierung wird auch das 8-Bit-Bild neu berechnet.

Parameter:

`c` - Das Zentrum des darzustellenden Fensters.

`public void` **`setWindow`**`(int w)`

Die Methode aktualiert die Einstellungen fuer die Fensterung im DcmDataObject. Nach der Aktualisierung wird auch das 8-Bit-Bild neu berechnet.

Parameter:

`w` - Die Breite des darzustellenden Fensters.

`public int` **`getMinGrayValue12Bit`**`()`

Die Methode liefert den minimalen Grauwert des 12-Bit-Bildes zurueck.

Rückgabewert:

Der minimale Grauwert im 12-Bit Bild.

`public int` **`getMaxGrayValue12Bit`**`()`

Die Methode liefert den maximalen Grauwert des 12-Bit-Bildes zurueck.

Rückgabewert:

Der maximale Grauwert im 12-Bit Bild.

`public void` **`optimalWindowing`**`()`

Die Methode fenstert das aktuelle Bild optimal, d.h. es werden alle Grauwerte, die groesser sind als Null, auf einen Bereich von 256 Grauwerten abgebildet.

`private void` **`calcFFT`**`()`

Die Methode berechnet die Fast-Fouriertransformation des 12-Bit-Bildes. Real- und Imaginarteil werden in Instanzvariablen abgespeichert und stehen damit in Zukunft zur Verfuegung, ohne neu berechnet werden zu muessen. Daneben wird allerdings noch eine Markierung fuer die Aktualitaet der Fouriertransformation mitgefuehrt. Eine Spiegelung oder Invertierung des Bildes macht diese naemlich ungueltig. Da die Berechnung der FFT jedoch einige Zeit benoetigt, ist es nicht sinnvoll, diese staendig neu zu berechnen, obwohl sie garnicht benoetigt wird.

`public float[][]` **`getFFTImageReal`**`()`

Die Methode liefert den Realteil der Fouriertransformation des 12-Bit Bildes zurueck.

Rückgabewert:

Der Realteil der Fouriertransformation.

`public float[][]` **`getFFTImageImag`**`()`

Die Methode liefert den Imaginaerteil der Fouriertransformation des 12-Bit Bildes zurueck.

Rückgabewert:

Der Imaginaerteil der Fouriertransformation.

Klasse ImageStack

public class **ImageStack** extends java.lang.Object

Die Klasse verwaltet einen Stapel von *ImagePlus*-Objekten. Dies sind Bildobjekte, die ein 12-Bit Bild, ein gefenstertes 8-Bit Bild und einige Zusatzinformationen enthalten. Die Klasse *ImageStack* bietet Moeglichkeiten, *ImagePlus*-Objekte zum Stapel hinzuzufuegen und zu loeschen sowie *ImagePlus*-Objekte aus dem Stapel zurueckzuliefern.

Klassen- und Instanzvariablen

private java.util.Vector	`images` Der Vektor enthaelt alle ImagePlus-Objekte.

Konstruktoren

`public` **`ImageStack`**`()`

Standardkonstruktor. Er initialisiert einen Vektor der Groesse 256 mit leeren Objekten.

Methoden

`public int` **`getStackSize`**`()`

Die Methode bestimmt die Anzahl der ImagePlus-Objekt im Stapel und liefert diese Anzahl zurueck.

Rückgabewert:

Die Anzahl der ImagePlus-Objekte im Stapel.

`public int` **`getNextFreePosition`**`()`

Die Methode bestimmt die erste freie Position im Vektor der ImagePlus-Objekte.

Rückgabewert:

Die erste freie Stelle im Stapel.

`public ImagePlus` **`createPictureAtPos`**`(int iposition, dcm.DcmImage di)`

Die Methode erzeugt ein neues ImagePlus-Objekt an der angegebenen Position im Stapel. Dazu wird das uebergebene DcmImage-Objekt verwendet.

Parameter:

`iposition` - Die Position, an der das ImagePlus-Objekt im Stapel angelegt werden soll.
`di` - Das DcmImage-Objekt, aus dem ein ImagePlus-Objekt erzeugt werden soll.

Rückgabewert:

Das neu erzeugte ImagePlus-Objekt.

`public void` **`setPictureAtPos`**`(int iposition, ImagePlus myip)`

Die Methode fuegt ein uebergebenes ImagePlus-Objekt an der angegebenen Stelle in den Stapel ein.

Parameter:

`iposition` - Die Position, an der das ImagePlus-Objekt im Stapel eingefuegt werden soll.
`myip` - Das ImagePlus-Objekt, das in den Stapel eingefuegt werden soll.

`public ImagePlus` **`getPictureAtPos`**`(int iposition)`

Die Methode liefert ein ImagePlus-Objekt aus dem Stapel zurueck. Die Position des zurueckzuliefernden Bildes muss angegeben werden. Ist das Bild nicht vorhanden, wird null zurueckgeliefert.

Parameter:

`iposition` - Die Position des gewuenschten Bildes im Stapel.

Rückgabewert:

Das angeforderte ImagePlus-Objekt.

`public void` **`clearImageStack`**`()`

Die Methode loescht den gesamten Bildstapel. Danach stehen keine ImagePlus-Objekte mehr zur Verfuegung.

`public void` **`deletePictureAtPos`**`(int i)`

Die Methode loescht das Bild an der angegebenen Position des Bildstapels.

Parameter:

`i` - Die Position des Bildes, das geloescht werden soll.

Klasse KSpaceFrame

public class **KSpaceFrame** extends JFrame

Diese Klasse repraesentiert das Fenster zur Darstellung des k-Raums eines Bildes. Anstatt des k-Raums (Magnitudenbild der Fouriertransformation) des selektierten Bildes kann auch das Phasenbild oder der Real- oder Imaginaerteil der Fouriertransformation dargestellt werden.

Klassen- und Instanzvariablen	
`private int`	`FRAME_HEIGHT` Konstante fuer die Hoehe des k-Raum-Fensters.
`private int`	`FRAME_WIDTH` Konstante fuer die Breite des k-Raum-Fensters.
`private java.awt.Image`	`imagImage` Der Imaginaerteil der Fouriertransformation des selektierten Bildes.
`private ImagePlus`	`ip` Das Bild, dessen k-Raum dargestellt werden soll.
`private java.awt.Image`	`magnitudeImage` Das Magnitudenbild (k-Raum) des selektierten Bildes.
`(package private) ButtonGroup`	`myButtonGroup` Gruppe fuer die Knoepfe, die die Auswahl des darzustellenden Bildes ermoeglichen.
`(package private) KSpaceCanvas`	`myCanvas` Referenz auf die Zeichenflaeche, auf der das ausgewaehlte Bild dargestellt werden soll.
`private java.awt.Image`	`phaseImage` Das Phasenbild des selektierten Bildes.
`private java.awt.Image`	`realImage` Der Realteil der Fouriertransformation des selektierten Bildes.
`(package private) JToggleButton`	`tbFFTImag` Der Knopf zur Darstellung des Imaginaerteils der Fouriertransformation.
`(package private) JToggleButton`	`tbFFTReal` Der Knopf zur Darstellung des Realteils der Fouriertransformation.
`(package private) JToggleButton`	`tbMagnitudeImage` Der Knopf zur Darstellung des Magnitudenbildes der Fouriertransformation (k-Raum).
`(package private) JToggleButton`	`tbPhaseImage` Der Knopf zur Darstellung des Phasenbildes.

Konstruktoren

`public` **`KSpaceFrame`**`(ImagePlus imgp)`

Der Konstruktor stellt das k-Raum-Anzeigefenster dar. Gleichzeitig merkt er sich das Bild, fuer den der k-Raum angezeigt werden soll und berechnet zunaecht das Magnitudenbild der Fouriertransformation und stellt dieses dar.

Parameter:

`imgp` - Das Bild, dessen k-Raum angezeigt werden soll.

Methoden

`public java.awt.Image` **`convertIntArrayToImage`**`(int[][] source)`
Die Methode konvertiert ein zweidimensionales Feld von Ganzzahlen in ein Grauwertbild (Java-Image).

Parameter:
`source` - Das in ein Image zu konvertierende zweidim. Feld von Ganzzahlen.

`public void` **`calculateAndDisplayMagnitudeImage`**`()`
Die Methode berechnet das Magnitudenbild der Fouriertransformation des selektierten Bildes. Dabei ist zu beachten, das das angezeigte Bild um die halbe Bildbreite bzw. Hoehe nach rechts bzw. unten verschoben ist und die Grauwerte logarithmisch skaliert wurden. Nur so ist eine kontrastreiche Anzeige moeglich.

`public void` **`calculateAndDisplayPhaseImage`**`()`
Die Methode berechnet das Phasenbild der Fouriertransformation des selektierten Bildes. Dabei ist zu beachten, das das angezeigte Bild um die halbe Bildbreite bzw. Hoehe nach rechts bzw. unten verschoben ist und die Grauwerte logarithmisch skaliert wurden. Nur so ist eine kontrastreiche Anzeige moeglich.

`public void` **`calculateAndDisplayRealImage`**`()`
Die Methode berechnet den Realteil der Fouriertransformation des selektierten Bildes. Dabei ist zu beachten, das das angezeigte Bild um die halbe Bildbreite bzw. Hoehe nach rechts bzw. unten verschoben ist und die Grauwerte logarithmisch skaliert wurden. Nur so ist eine kontrastreiche Anzeige moeglich.

`public void` **`calculateAndDisplayImagImage`**`()`
Die Methode berechnet den Imaginaerteil der Fouriertransformation des selektierten Bildes. Dabei ist zu beachten, das das angezeigte Bild um die halbe Bildbreite bzw. Hoehe nach rechts bzw. unten verschoben ist und die Grauwerte logarithmisch skaliert wurden. Nur so ist eine kontrastreiche Anzeige moeglich.

`private void` **`jbInit`**`()`
`throws java.lang.Exception`
Die Methode baut das k-Raum-Anzeigefenster und seine Elemente auf und stellt sie dar. Zusaetzlich werden ActionListener fuer die Knoepfe angelegt.

Klasse KSpaceManipulator

public class **KSpaceManipulator** extends JFrame

Die Klasse implementiert das Fenster und die Funktionen zur k-Raum-Manipulation. Dazu zaehlen das Loeschen von Zeilen und Spalten am Rand des k-Raums, das Loeschen eines zentral im k-Raum gelegenen Rechtecks und das Loeschen von Zeilen bzw. Spalten in einem bestimmten Abstand.

Klassen- und Instanzvariablen	
(package private) JCheckBox	`cbDeleteLines` Kontrollkästchen zum Ein-/Ausschalten des Manipulators zum Loeschen der Zeilen/Spalten.
(package private) JCheckBox	`cbDeleteMargins` Kontrollkästchen zum Ein-/Ausschalten des Manipulators zum Loeschen der Ränder.
(package private) JCheckBox	`cbDeleteRect` Kontrollkästchen zum Ein-/Ausschalten des Manipulators zum Loeschen des Rechtecks.
(package private) int[]	`intKSpace` Das Magnitudenbild des Original-k-Raums (zur Darstellung).
(package private) int[]	`manipIntKSpace` Das Magnitudenbild des manipulierten k-Raums (zur Darstellung).
(package private)	`manipKSpaceImag`

float[][]	Der manipulierte Imaginaerteil der Fouriertransformation des Originalbildes.
(package private) float[][]	manipKSpaceReal Der manipulierte Realteil der Fouriertransformation des Originalbildes.
(package private) float[][]	origKSpaceImag Der Imaginaerteil der Fouriertransformation des Originalbildes.
(package private) float[][]	origKSpaceReal Der Realteil der Fouriertransformation des Originalbildes.
(package private) JButton	pbCalcImage Knopf, um die Ruecktransformation des manipulierten k-Raums zu berechnen.
(package private) KSpaceManipulatorCanvas	pCanvas Die Zeichenflaeche zur Darstellung der vier Bilder.
(package private) JPanel	pDeleteLines Der Rahmen fuer die Bedienelemente zum Loeschen einzelner Zeilen/Spalten.
(package private) JPanel	pDeleteMargins Der Rahmen fuer die Bedienelemente zum Loeschen der Raender.
(package private) JPanel	pDeleteRect Der Rahmen fuer die Bedienelemente zum Loeschen eines Rechtecks.
(package private) JPanel	pToolbar Der Rahmen für die Bedienelemente zur k-Raum-Manipulation.
(package private) ImagePlus	sourceImg Das ImagePlus-Objekt des Bildes, dessen k-Raum manipuliert werden soll.
(package private) LabelTFLabelPanel	tfDelBottomRows Bedienelemente für die am unteren Rand zu loeschenden Zeilen.
(package private) LabelTFLabelPanel	tfDelColumns Bedienelemente für den Abstand der zu loeschenden Spalten.
(package private) LabelTFLabelPanel	tfDelLeftColumns Bedienelemente für die am linken Rand zu loeschenden Spalten.
(package private) LabelTFLabelPanel	tfDelRectHeight Bedienelemente für die Hoehe des zu loeschenden Rechtecks.
(package private) LabelTFLabelPanel	tfDelRectWidth Bedienelemente für die Breite des zu loeschenden Rechtecks.
(package private) LabelTFLabelPanel	tfDelRightColumns Bedienelemente für die am rechten Rand zu loeschenden Spalten.
(package private) LabelTFLabelPanel	tfDelRows Bedienelemente für den Abstand der zu loeschenden Zeilen.
(package private) LabelTFLabelPanel	tfDelTopRows Bedienelemente für die am oberen Rand zu loeschenden Zeilen.

Konstruktoren

`public` **`KSpaceManipulator`**`(ImagePlus i)`

Der Konstruktor stellt das k-Raum-Manipulatorfenster mit seinen Bedienelementen dar und berechnet die Fouriertransformation des uebergebenen Bildes. Diese und das Originalbild werden auf der Zeichenflaeche dargestellt.

Parameter:

`i` - Das ImagePlus-Objekt des Bildes, dessen k-Raum manipuliert werden soll.

Methoden

`private void` **`jbInit`**`()`
`throws java.lang.Exception`

Die Methode baut das k-Raum-Manipulationsfenster und seine Elemente auf und stellt sie dar. Zusaetzlich werden ActionListener fuer die Knoepfe initialisiert.

`private void` **`deleteLines`**`()`

Die Methode loescht die Zeilen und Spalten in einem vom Benutzer vorgegebenen Abstand. Die Methode wird bei jedem Betaetigen eines Bedienelements des k-Raum-Manipulators aufgerufen, so dass die Aenderungen sofort sichtbar werden.

`private void` **`deleteRect`**`()`

Die Methode loescht ein zentral im k-Raum gelegenes Rechteck einer vom Benutzer vorgegebenen Hoehe und Breite. Die Methode wird bei jedem Betaetigen eines Bedienelements des k-Raum-Manipulators aufgerufen, so dass die Aenderungen sofort sichtbar werden.

`private void` **`deleteMargins`**`()`

Die Methode loescht die Zeilen und Spalten am Rand des k-Raums, so wie der Benutzer dies mit den Bedienelementen eingestellt hat. Die Methode wird bei jedem Betaetigen eines Bedienelements des k-Raum-Manipulators aufgerufen, so dass die Aenderungen sofort sichtbar werden.

`public void` **`updateKSpaceImage`**`()`

Die Methode wird bei jedem Betaetigen eines der Bedienelemente zur k-Raum-Maipulation aufgerufen. Sie berechnet dann sowohl den darzustellenden manipulierten k-Raum neu, als auch den Real- und Imaginaerteil der Fouriertransformation, die beide fuer die Ruecktransformation benoetigt werden.

`void` **`pbCalcImage_actionPerformed`**`(java.awt.event.ActionEvent e)`

Die Methode wird immer dann aufgerufen, wenn der Benutzer den Knopf zur Berechnung der Ruecktransformation des manipulierten k-Raums betaetigt hat. Es werden dann Matrizen des manipulierten Real- und Imaginaerteils der Fouriertransformation erzeugt und mit diesen eine Ruecktransformation vorgenommen. Das ruecktransformaierte Bild wird dann auf der Zeichenflaeche dargestellt.

Parameter:

`e` - Das Ereigenis, das beim Betaetigen des Knopfes zur Berechnung der der Ruecktransformation ausgeloest wird.

Klasse Motion

public class **Motion** extends `Artefact`

Diese Klasse ist die Berechnungsklasse zur Simualation eines Bewegungs-Artefakts. Dabei beschraenkt sich die Bewegung auf eine einfache Translation. Es kann ein Start- und ein Stopzeitpunkt angegeben werden und die Bewegungsgeschwindigkeit fuer *x*- und *y*-Richtung separat festgelegt werden.

Klassen- und Instanzvariablen	
`private int`	`NUMSTEPS` Konstante fuer die Genauigkeit der Berechnung.
`private int`	`startTime` Startzeitpunkt der Translation.
`private int`	`stopTime` Stopzeitpunkt der Translation.
`private int`	`trTime` Repititionszeit der Pulssequenz.
`private int`	`xSpeed` Bewegungsgeschwindigkeit in x-Richtung.
`private int`	`ySpeed` Bewegungsgeschwindigkeit in y-Richtung.

Konstruktoren

`public` **`Motion`**`()`

Standardkonstruktor.

Methoden

`public int[][] calculate(java.lang.String method)`

Diese Methode wird von der aufrufenden Pulssequenz zuerst aufgerufen. Der uebergebene Methodenname gibt an, welche Methode die korrekte Berechnungsvorschrift enthaelt. Vor dem Aufruf dieser Methode werden noch die benoetigten Parameter aus der aufrufenden Pulssequenz ausgelesen.

Parameter:

`method` - Der Name der Methode, die die korrekte Berechnungsvorschrift enthaelt.

`public int[][] motion_Std()`

Die Methode enthaelt die Standard-Berechnungsvorschrift zur Simulation einer translatorischen Bewegung.

`public void setXSpeed(int x)`

Die Methode dient zum Setzen der x-Bewegungsgeschwindigkeit nach dem Auslesen aus dem entsprechenden Bedienelement.

Parameter:

`x` - Bewegungsgeschwindigkeit in x-Richtung.

`public void setYSpeed(int y)`

Die Methode dient zum Setzen der y-Bewegungsgeschwindigkeit nach dem Auslesen aus dem entsprechenden Bedienelement.

Parameter:

`y` - Bewegungsgeschwindigkeit in y-Richtung.

`public void setStartTime(int s)`

Die Methode dient zum Setzen des Startzeitpunkts der Translation nach dem Auslesen aus dem entsprechenden Bedienelement.

Parameter:

`s` - Startzeitpunkt der Translation.

`public void setStopTime(int s)`

Die Methode dient zum Setzen des Stopzeitpunkts der Translation nach dem Auslesen aus dem entsprechenden Bedienelement.

Parameter:

`s` - Stopzeitpunkt der Translation.

```
private int[][] move(int h,
           int v)
```

Die Methode erzeugt eine Kopie des Intensitaetsbildes und verschiebt diese Kopie um den angegebenen Betrag nach rechts und nach unten. Die aus dem Bild geschobenen Pixel erscheinen nicht (!) wieder an der anderen Seite. Darin besteht der Unterschied zur Methode shift der Klasse FFTTools.

Parameter:

`h` - Anzahl der Pixel, um die das Bild horizontal verschoben werden soll.

`v` - Anzahl der Pixel, um die das Bild vertikal verschoben werden soll.

Rückgabewert:

Das verschobene Intensitaetsbild.

Klasse MotionUI

public class **MotionUI** extends `ArtefactUI`

Diese Klasse ist die Oberflaechenklasse zur Simualation eines Bewegungs-Artefakts. Dabei beschraenkt sich die Bewegung auf eine einfache Translation. Es kann ein Start- und ein Stopzeitpunkt angegeben werden und die Bewegungsgeschwindigkeit fuer *x*- und *y*-Richtung separat festgelegt werden.

Klassen- und Instanzvariablen	
(package private) Motion	`myManipulator` Referenz auf die Berechnungsklasse.
private LabelTFLabelPanel	`pStartTime` Bedienelemente für den Startzeitpunkt der Translation.
private LabelTFLabelPanel	`pStopTime` Bedienelemente für den Stopzeitpunkt der Translation.
private LabelTFLabelPanel	`pXSpeed` Bedienelemente für die Bewegungsgeschwindigkeit in x-Richtung.
private LabelTFLabelPanel	`pYSpeed` Bedienelemente für die Bewegungsgeschwindigkeit in y-Richtung.

Konstruktoren

`public` **`MotionUI`**`()`
Standardkonstruktor.

Methoden

`public void` **`jbInit`**`()`
Die Methode richtet die Oberflächenelemente ein und fuegt sie dem Darstellungsbereich (myPanel) hinzu.

`public void` **`setValues`**`()`
Die Methode liest die aktuellen Parameterwerte aus den Bedienelementen aus und speichert sie in den entsprechenden Instanzvariablen.

`public void` **`getValues`**`()`
Die Methode setzt die Werte der Oberflaechenelemente auf die in den Instanzvariablen gespeicherten Werte. Dies wird beim Umschalten zwischen Artefakt-Simulatoren benötigt.

`public Artefact` **`createManipulator`**`()`
Die Methode erzeugt eine neue Instanz der Berechnungsklasse des Artefakt-Simulators und liefert diese zurueck. Ausserdem werden die Einstellungen der Bedienelemente an die Berechnungsklasse uebergeben.

Klasse MyPanel (*DICOM-Viewer*)

public class **MyPanel** extends JScrollPane

Die Klasse implementiert die Zeichenflaeche des *DICOM-Viewers*. Insbesondere wird die *paint*-Methode der *JScrollPane*-Klasse ueberschrieben, um die Bilder, die Bildbeschriftungen und die Werkzeuge darzustellen. Darueber hinaus verwaltet die Klasse eine Referenz auf den gesamten Bildstapel und ist fuer das Hinzufuegen von Bildern zu diesem Stapel verantwortlich.

Klassen- und Instanzvariablen

private int	currentGray aktueller Grauwert unter dem Mauszeiger.
private int	currentX Aktuelle x-Position des Mauszeigers.
private int	currentY Aktuelle y-Position des Mauszeigers.
private int	iActiveImage Nummer des selektierten Bildes.
private int	iNumImages Anzahl der Bilder, die gleichzeitig dargestellt werden (1 oder 4).
private ImageStack	is Verweis auf den Stapel aller Bilder.
private int	lastClickX x-Position des letzten Mausklicks.
private int	lastClickY y-Position des letzten Mausklicks.
private int	lastReleaseX x-Position des letzten MausRelease-Ereignisses.
private int	lastReleaseY y-Position des letzten MausRelease-Ereignisses.
private ViewerFrame	myViewerFrame Rueckreferenz zum Hauptfenster.
private java.awt.Image	zoomImage Der Bildausschnitt, der zusaetzlich gezeichnet werden muss, wenn die Lupen-Funktion eingeschaltet ist.
private int	zoomX x-Position, an der das Zoombild gezeichnet werden soll.
private int	zoomY y-Position, an der das Zoombild gezeichnet werden soll.

Konstruktoren

`public` **`MyPanel`**`()`

Standard-Konstruktor.

`public` **`MyPanel`**`(ViewerFrame backReference)`

Konstruktor mit Rueckreferenz zum Hauptfenster als Parameter.

Parameter:

`backReference` - Rueckreferenz zum Haupfenster. Sie wird benoetigt, um Einstellungen einiger Bedienelemente auszulesen.

Methoden

`public void` **`updateScrollbar`**`()`

Die Methode aktualisiert die Scrollbalkeneinstellungen in Abhaengigkeit davon, ob ein oder vier Bilder gleichzeitig dargestellt werden sollen.

`public void` **`setOneImage`**`()`

Die Methode setzt den internen Parameter, welcher angibt, wie viele Bilder gleichzeitig dargestellt werden, auf 1. Ausserdem wird der Scrollbereich aktualisiert.

`public void` **`setFourImages`**`()`

Die Methode setzt den internen Parameter, welcher angibt, wie viele Bilder gleichzeitig dargestellt werden, auf 4. Ausserdem wird der Scrollbereich aktualisiert

---

`public void` **`setActiveImage`**`(int activeImage)`

Die Methode legt die Nummer des zur Zeit selektierten Bildes fest.

Parameter:

`activeImage` - Die Nummer des selektierten Bildes.

`public void` **`setZoomImage`**`(java.awt.Image zoomI)`

Die Methode legt das Bild fest, das zusaetzlich eingezeichnet werden muss, wenn die Lupen-Funktion eingeschaltet ist.

Parameter:

`zoomI` - Der Bildauschnitt, der zusaetzlich gezeichnet werden muss.

`public void` **`setZoomPos`**`(int x,`
`int y)`

Die Methode legt die x- und y-Position des Zoomfensters fest.

Parameter:

`x` - Die x-Position des Zoomfensters.
`y` - Die y-Position des Zoomfensters.

`public void` **`setLastClickPos`**`(int x,`
`int y)`

Die Methode legt die x- und y-Position des letzten Mausklicks fest.

Parameter:

`x` - Die x-Position des letzten Mausklicks.
`y` - Die y-Position des letzten Mausklicks.

`public void` **`setReleasePos`**`(int x,`
`int y)`

Die Methode legt die x- und y-Position des letzten Mouse-Release Ereignisses fest.

Parameter:

`x` - Die x-Position des letzten Mouse-Release-Ereignisses.
`y` - Die y-Position des letzten Mouse-Release-Ereignisses.

`public void` **`setCurrentPos`**`(int x,`
`int y)`

Die Methode legt die aktuelle x- und y-Position des Mauszeigers fest.

Parameter:

`x` - Die aktuelle x-Position des Mauszeigers.
`y` - Die aktuelle y-Position des Mauszeigers.

`public void` **`setGrayValue`**`(int g)`

Die Methode legt den Grauwert an der aktuellen Mausposition fest. Dieser wird zur Anzeige des Grauwerts bei Einstellung des entsprechenden Werkzeugs benoetigt.

Parameter:

`g` - Der Grauwert an der aktuellen Mausposition.

`public int` **`getActiveImage`**`()`

Die Methode liefert die Nummer des zur Zeit selektierten Bildes.

Rückgabewert:

Die Nummer des selktierten Bildes.

`public ImageStack` **`getImageStack`**`()`

Die Methode liefert den Bildtstapel, also die Menge aller Bilder, die sich zur Zeit im Scrollbereich befinden (auch wenn sie gerade nicht sichtbar sind).

Rückgabewert:

Der Stapel aller zur Zeit verfuegbarer Bilder.

`public ImagePlus` **`createNewImage`**`(dcm.DcmImage di)`

Die Methode erzeugt aus dem uebergebenen DcmImage-Objekt ein neues ImagePlus-Objekt und fuegt dieses in den Bildstapel ein. Zuerst wird die Position bestimmt, an der das Bild angelegt wird. Dann wird das Bild erzeugt und dargestellt und der Scrollbereich aktualisiert.

Parameter:

`di` - Das DcmImag-Objekt, aus dem ein ImagePlus-Objekt erzeugt werden soll.

Rückgabewert:

Das neu erzeugte ImagePlus-Objekt.

`public int` **`createNewImage`**`(ImagePlus myip)`

Die Methode fuegt das uebergebene ImagePlus-Objekt in den Bildstapel ein. Zuerst wird dafuer die Position bestimmt, an der das Bild eingefuegt wird. Dann wird das Bild in den Stapel eingefuegt und der Scrollbereich aktualisiert.

Parameter:

`myip` - Das ImagePlus-Objekt, das in den Stack eingefuegt werden soll.

Rückgabewert:

Die Position, an der das ImagePlus-Objekt eingefuegt wurde.

`private int` **`getPosForNewImage`**`()`

Die Methode bestimmt die Position fuer ein neu einzufuegendes Bild. Die Position ist von den Benutzervorgaben abhaengig. Das Bild kann entweder an der selektierten Position eingefuegt werden oder aber an der naechsten freien Position.

Rückgabewert:

Die Position, an der das naechste Bild eingefuegt werden soll.

`private int` **`getSBOffset`**`()`

Die Methode liefert den aktuellen Offset des Scrollbalkens in Bildern zurueck.

Rückgabewert:

Der aktuelle Offset des Scrollbalkens.

`public void` **`paint`**`(java.awt.Graphics g)`

Die Paint Methode sorgt dafuer, dass alle gewuenschten Elemente auf der Zeichenflaeche dargestellt werden. Die Methode wird immer dann aufgerufen, wenn sich etwas auf der Zeichenflaeche veraendern soll. Dazu zaehlen Aenderungen an den Bilder, an den Bildbeschriftungen oder Darstellungselemente, die durch die Werkzeuge bedingt sind (Lupe, Distanzmessung, Winkelmessung).

Parameter:

`g` - Der Graphikkontext.

Klasse MyPanel (*Virtual MRT*)

public class **MyPanel** extends JScrollPane

Die Klasse implementiert die Zeichenflaeche des virtuellen MRT. Insbesondere wird die *paint*-Methode der *JScrollPane*-Klasse ueberschrieben, um die Bilder und Beschriftungen einzuzeichnen. Darueber hinaus verwaltet diese Klasse den gesamten Bildstapel und ist für das Einfuegen von neuen Bildern in diesen Stapel verantwortlich.

Klassen- und Instanzvariablen

`private int`	`iActiveImage` Nummer des selektierten Bildes.
`private int`	`iNumImages` Anzahl der Bilder, die gleichzeitig dargestellt werden (1 oder 4).
`private ImageStack`	`is` Verweis auf den Stapel aller Bilder, die zur Zeit dargestellt werden.
`(package private) VMRTFrame`	`myVMRTFrame` Rueckreferenz zum Hauptfenster.

Konstruktoren

`public` **`MyPanel`**`()`

Standard-Konstruktor.

`public` **`MyPanel`**`(VMRTFrame backReference)`

Konstruktor mit einer Rueckreferenz zum Hauptfenster als Parameter.

Parameter:

`backReference` - Rueckreferenz zum Haupfenster. Sie wird benoetigt, um Einstellungen einiger Bedienelemente auszulesen.

Methoden

`public void` **`setOneImage`**`()`

Die Methode setzt den internen Parameter, welcher angibt, wie viele Bilder gleichzeitig dargestellt werden, auf 1. Ausserdem wird der Scrollbereich aktualisiert.

`public void` **`setFourImages`**`()`

Die Methode setzt den internen Parameter, welcher angibt, wie viele Bilder gleichzeitig dargestellt werden, auf 4. Ausserdem wird der Scrollbereich aktualisiert.

`public void` **`updateScrollbar`**`()`

Die Methode aktualisiert die Scrollbarlkeneinstellungen in Abhaengigkeit davon, ob ein oder vier Bilder gleichzeitig dargestellt werden sollen.

`private int` **`getSBOffset`**`()`

Die Methode liefert den Offset des Scrollbalkens zurueck.

Rückgabewert:

Der Offset des Scrollbalkens.

`public int` **`getNumImages`**`()`

Die Methode liefert die Anzahl der gleichzeitig dargestellten Bilder zurueck.

Rückgabewert:

Die Anzahl gleichzeitig dargestellter Bilder.

`public void` **`setActiveImage`**`(int activeImage)`

Die Methode legt die Nummer des zur Zeit selektierten Bildes fest.

Parameter:
`activeImage` - Die Nummer des selektierten Bildes.

`public int` **`getActiveImage`**`()`
Die Methode liefert die Nummer des zur Zeit selektierten Bildes zurueck.

Rückgabewert:
Die Nummer des selktierten Bildes.

`public ImageStack` **`getImageStack`**`()`
Die Methode liefert den ImageStack, also die Menge aller Bilder, die sich zur Zeit im Scrollbereich befinden (auch wenn sie gerade nicht sichtbar sind).

Rückgabewert:
Der aktuelle Bildstapel.

`public ImagePlus` **`createNewImage`**`(dcm.DcmImage dcmimg)`
Die Methode erzeugt aus einen DcmImage (12-Bit-Bild) ein ImagePlus-Objekt (ein Bild inklusive Fensterungseinstellungen (Window und Center) und weiterer Informationen). Dieses neue Objekt wird dem Bildstapel hinzugefuegt und an den Benutzer zurueckgeliefert.

Parameter:
`intensityImage` - Das ungefensterte 12-Bit Bild.

Rückgabewert:
Das neu erzeugte ImagePlus-Objekt.

`public void` **`paint`**`(java.awt.Graphics g)`
Die paint-Methode sorgt dafuer, dass alle gewuenschten Elemente auf der Zeichenflaeche dargestellt werden. Dazu gehoeren die Bilder selbst, sowie deren Beschriftungen. Die Methode wird immer dann aufgerufen, wenn sich etwas auf der Zeichenflaeche veraendern soll.

Parameter:
`g` - Der Graphikkontext.

Klasse Pulsesequence

public abstract class **Pulsesequence** extends java.lang.Thread

Diese Klasse ist die Oberklasse fuer alle Pulssequenz-Berechnungsklassen. Sie erweitert die Klasse *Thread*. Somit sind alle Pulssequenzen *Threads*. Das bietet die Moeglichkeit, die Abarbeitung zu stoppen und Fortschrittsanzeigen auszugeben. Die Klasse definiert Methoden zur Berechnung eines Bildes mittels der Pulssequenzformel. Da die Klasse *abstract* ist, muss sie erweitert werden.

Klassen- und Instanzvariablen	
`protected int[][]`	`IntensityMatrix` Ergebnismatrix, enthaelt die Intensitaetswerte.
`protected int`	`iTimeFactor` Simulationszeitfaktor (0-100 Prozent).
`protected VMRTFrame`	`mainFrame` Referenz zum Hauptfenster.
`protected PulsesequenceUI`	`myUIClass` Referenz auf die zugehoerige GUI-Klasse.
`protected int[][]`	`NoisyPDMatrix` Matrix mit den Werten fuer die Protonendichte inkl.
`protected int[][]`	`PDMatrix` Matrix mit den Werten fuer die Protenendichte.

protected JProgressBar	progressBar Referenz zur Fortschrittsanzeige im Hauptfenster.
protected int[][]	Result12Bit Ergebnismatrix, automatisch auf 12-Bit gefenstert.
protected ImagePlus	ResultIP Das Ergebnisbild (12-Bit) als ImagePlus-Objekt.
protected int[][]	T1Matrix Matrix mit den T1-Werten.
protected int[][]	T2Matrix Matrix mit den T2-Werten.

Konstruktoren

`public` **`Pulsesequence`**`()`

Standardkonstruktor. Er setzt die Prioritaet des Thread auf maximal.

Methoden

`public void` **`setMainFrame`**`(VMRTFrame vmrt)`

Die Methode setzt die Referenz zum Hauptfenster. Anschliessend besorgt sich die Methode Referenzen auf die Fortschrittsanzeige und den Simulationszeit- faktor und speichert diese in Instanzvariablen.

Parameter:

`vmrt` - Die Referenz auf das Hauptfenster.

`public void` **`run`**`()`

Die Methode ruft die Rechenfunktion der Pulssequenz auf. Sie wird automatisch aufgerufen, wenn Pulsesequence.start() aufgerufen wird. aufgrund der Thread-Funktionalitaet darf die Rechenfunktion immer nur ueber die start()-Methode aufgerufen werden.

`protected void` **`addSequenceParameterToImage`**`()`

Die Methode fuegt das Berechnungsdatum des Ergebnisbildes zu diesem Bild hinzu. Somit ist diese Information auch vorhanden, wenn das Bild im DICOM-Format abgespeichert wird. In den Unterklassen koennen weitere Informationen (z.B. Seuqnzname) zum Bild hinzugefuegt werden.

`public void` **`setRawData`**`(FileLoader fl)`
`throws java.lang.NullPointerException`

Die Method liest aus dem uebergebenen FileLoader-Objekt die Rohdatenmatrizen aus und legt sie in den lokalen Variablen ab. Ausserdem wird die Ergebnismatrix initialisiert.

Parameter:

`fl` - Das FileLoader-Objekt, dass die auszulesenden Rohdaten enthaelt.

`public void` **`calculate`**`()`

Die Methode stellt die Berechnungsfunktion fuer die Pulssequenz dar. Allerdings muss die eigentliche Funktionalitaet in den Unterklassen definiert werden. Diese Methode stoesst nach Berechnung des Intensitaetsbides in der Unterklasse den potentiellen Artefakt-Simulator an. Zum Ende der Methode wird die Fortschrittsanzeige zurueckgesetzt und das Hauptfenster benachrichtigt, dass die Berechnung abgeschlossen ist.

`public int[][]` **`getIntensityMatrix`**`()`

Die Methode liefert die Ergebnismatrix (Intensitaetsmatrix) zurueck.

Rückgabewert:

Die Ergbnismatrix.

`public short[]` **`getShort12BitMatrix`**`()`

Die Methode liefert das 12-Bit-Bild als eindimensionales Feld von 16-Bit-Zahlen zurueck.

Rückgabewert:
Das 12-Bit-Ergebnisbild.

`public void` **`setUI`**`(PulsesequenceUI ui)`
Die Methode setzt die Referenz auf die zugehoerige UI-Klasse.

Parameter:
`Referenz` - auf die zugehoerige UI-Klasse.

`public void` **`convertIntensityMatrixTo12Bit`**`()`
Die Methode konvertiert die zunaechst berechnete Intensitaetsmatrix in ein 12-Bit Bild. Dazu wird zunaecht ein Histogramm des Intesitaetsbildes berechnet und auf dessen Grundlage werden die Intensitaetswerte auf Grauwerte im Bereich [0;4095] abgebildet.

`public void` **`integrateFOVInIntensityMartrix`**`()`
Die Methode integriert die aktuelle Einstellung des FOV (Field of View) in die Intensitaetsmatrix. Das bedeutet, dass die Randbereich mit Nullen gefuellt werden. Dadurch wird natuerlich das Bild kleiner. In der Realitaet wuerde sich das anders verhalten. Eine Verkleinerung des FOV haette eine Verbessung der Aufloesung zur Folge. Zwar wuerde der dargestellte Bildausschnitt kleiner, jedoch auch die Pixelgroesse. Wuerde man in der Simulation jedoch den verkleinerten Bildausschnitt auf die Ursprungsgroesse vergroessern, haette dies eine Verschlechterung der Aufloesung zur Folge, da die Rohdatenmatrizen leider keine unendlich hohe Aufloesung aufweisen. Also haben wir uns dafuer entschieden, das Bild kleiner darzustellen.

`public void` **`simulateXAliasing`**`()`
Die Methode simuliert Einfaltungen, die Aufgrund eines zu klein gewaehlten FOV entstehen. Ausserhalb des Bildes liegende signalgebende Pixel werden falsch frequenzkodiert und werden somit in den Bildbereich projiziert.

`public void` **`addNoiseToPDMatrix`**`()`
Die Methode fuegt der Rohdatenmatrix mit den Protonendichtewerten ein Rauschen hinzu, dass vom eingestellten Signal-zu-Rausch-Verhaeltnis und der ausgewaehlten Spule abhaengt. Die Spule liefert ein Grundrauschen im gesamten Bild, wohingegen das Signal-zu-Rausch-Verhaeltnis nur ein Rauschen in signalgebenden Bildteilen nach sich zieht.

Klasse PulsesequenceUI

public abstract class **PulsesequenceUI** extends java.lang.Object

Diese Klasse ist eine abstrakte Oberklasse fuer alle Pulssequenz-Oberflaechenklassen. Die Klasse stellt einige Standardmethoden zur Verfuegung, die jede Unterklasse ueberschreiben muss. Das stellt sicher, das Erweiterungen dieser Klasse auf jeden Fall im Hauptfenster dargestellt werden koennen.

Klassen- und Instanzvariablen	
`protected Artefact`	`artefact` Referenz auf einen Artefakt-Simulator.
`(package private) JComboBox`	`cbCoil` Auswahlbox zur Auswahl der Spule.
`private boolean`	`isInitialized` Die Variable merkt sich, ob die Listener der Bedienelemente schon initialisiert wurden.
`protected java.util.Properties`	`kSpaceManipulator` Eigenschaftsfeld für Methodenaufrufe für k-Raummanipulationen.
`(package private) JLabel`	`lCoil` Beschriftung zur Auswahl der Spule.

(package private) JLabel	lFreqOS Beschriftung des Knopfes zum Ein- und Ausschalten des Frequenzoversamplings.
(package private) JLabel	lPhOS Beschriftung des Knopfes zum Ein- und Ausschalten des Phasenoversamplings.
protected VMRTFrame	mainFrame Referenz auf das Hauptfenster.
protected JPanel	myInfoPanel Referenz auf den Bereich, in dem die Informationselemente der Pulssequenz dargestellt werden koennen.
protected JPanel	myPanel Referenz auf den Bereich, in dem die Oberflaechenelemente der Pulssequenz dargestellt werden koennen.
(package private) JButton	pbStart Knopf, um die Berechnung des Intensitsetsbildes zu Starten.
(package private) JButton	pbStop Knopf, um die Berechnung des Intensitaetsbildes zu Stoppen.
(package private) LabelTFLabelPanel	pFOV Elemente zur Einstellung des Field of View (FOV).
(package private) LabelTFLabelPanel	pMatrix Elemente zur Einstellung der Bildmatrixgroesse.
(package private) LabelTFLabelPanel	pNEX Elemente zur Einstellung der Anzahl der Anregungen.
(package private) ThreeLabelPanel	pPixelSize Elemente zur Anzeige der Pixelgroesse.
(package private) LabelTFLabelPanel	pRect Elemente zur Einstellung der Bildproportionen.
(package private) ThreeLabelPanel	pRemainingTime Elemente zur Anzeige der Restberechnungzeit.
(package private) ThreeLabelPanel	pSNRatio Elemente zur Anzeige des Signal-zu-Rausch-Verhaeltnisses.
(package private) LabelTFLabelPanel	pThickness Elemente zur Einstellung der Schichtdicke.
(package private) ThreeLabelPanel	pTotalTime Elemente zur Anzeige der Gesamtberechnungzeit.
protected Pulsesequence	sequence Referenz auf die Pulssequenz-Berechnungsklasse.
(package private) JToggleButton	tbFreqOS Knopf zur Ein- und Ausschalten des Frequenzoversamplings.
(package private) JToggleButton	tbPhOS Knopf zur Ein- und Ausschalten des Phasenoversamplings.
(package private) JToggleButton	tbSettings Knopf zum Einblenden der Parametereinstellungen.

Konstruktoren

`public` **`PulsesequenceUI`**`()`

Standardkonstruktor.

Methoden

`public void` **`setSettingPanel`**`(JPanel sequencePanel)`

Die Methode setzt die Referenz auf den Bereich, in dem die Oberflaechenelemente der Pulssequenz dargestellt werden koennen.

Parameter:

`sequencePanel` - Referenz auf den Bereich zur Darstellung der Einstellungselemente.

`public void` **`setInfoPanel`**`(JPanel sequenceInfoPanel)`

Die Methode setzt die Referenz auf den Bereich, in dem die Informations- elemente der Pulssequenz dargestellt werden koennen.

Parameter:
`sequencePanel` - Referenz auf den Bereich zur Darstellung der Informationselemente.

`public void` **`setMainFrame`**`(VMRTFrame vmrt)`
Die Methode setzt die Referenz auf das Hauptfenster.

Parameter:
`vmrt` - Die Referenz auf das Hauptfenster.

`public abstract void` **`getValues`**`()`
Die Methode setzt die Einstellungen der Bedienelemente der Sequenz auf die in den entsprecheden Klassenvariablen gespeicherten Werte. Die Methode wird beim Umschalten zwischen den verschiedenen Pulssequenzen verwendet, um zuvor gemachte Einstellungen wiederherzustellen. In dieser Oberklasse weist die Methode allerdings keine Funktionalitaet auf.

`public abstract void` **`setValues`**`()`
Die Methode liest die aktuellen Sequenzparameterwerte aus den Bedienelementen aus und speichert sie in den entsprechenden Instanzvariablen. Die Methode wird beim Umschalten zwischen den verschiedenen Pulssequenzen verwendet, um die Einstellungen der Bedienelemente zu sichern und sie bei erneuter Auswahl der Pulssequenz ggf. wieder herstellen zu koennen. In dieser Oberklasse weist die Methode allerdings keine Funktionalitaet auf.

`public void` **`fillPanel`**`()`
Die Methode fuegt einige Bedien- und Informationselemente in die Einstellungs- und Informationsbereiche, die fuer alle Pulssequenzen sinnvoll sind.

`void` **`jbInit`**`()`
`throws java.lang.Exception`
Die Methode ist in dieser abstrakten Oberklasse leer. Sie wird in den abgeleiteten Klassen dazu verwendet, um weitere Bedien- und Informationselemente einzurichten.

`public void` **`enableStartButton`**`(boolean flag)`
Die Methode aktiviert oder deaktiviert den Startknopf. D.h. sie ermoeglicht das Betaetigen des Knopfes oder verhindert dies.

Parameter:
`flag` - True, wenn der Knopf aktiviert werden soll, sonst False.

`public void` **`enableStopButton`**`(boolean flag)`
Die Methode aktiviert oder deaktiviert den Stopknopf. D.h. die ermoeglicht das Betaetigen des Knopfes oder verhindert dies.

Parameter:
`flag` - True, wenn der Knopf aktiviert werden soll, sonst False.

`void` **`pbStart_actionPerformed`**`(java.awt.event.ActionEvent e)`
Die Methode wird aufgerufen, wenn der Startknopf gedrueckt wurde. In diesem Fall wird der Startknopf deaktiviert und der Stoppknopf aktiviert. Darueber hinaus wird eine Instanz eines potentiellen Artefakt-Simulators erzeugt.

Parameter:
`e` - Das Ergeignis, das beim Betaetigen des Startkopfes ausgeloest wird.

`void` **`pbStop_actionPerformed`**`(java.awt.event.ActionEvent e)`

Die Methode wird aufgerufen, wenn der Stoppknopf gedrueckt wurde. In diesem Fall wird der Stoppknopf deaktiviert und der Startknopf aktiviert. Ausserdem wird die Fortschrittsanzeige und die Restzeit auf Null zurueckgesetzt.

Parameter:

`e` - Das Ergeignis, das beim Betaetigen des Stoppknopfes ausgeloest wird.

`void` **`tbSettings_actionPerformed`**`(java.awt.event.ActionEvent e)`

Die Methode wird aufgerufen, wenn der Knopf zur Anzeige des Einstellungsbereichs gedrueckt wurde. Der Einstellungbereich wird dann entweder ein- oder ausgeblendet.

Parameter:

`e` - Das Ergeignis, das beim Betaetigen des Knopfes ausgeloest wird.

`void` **`displayTotalTime`**`(long totaltime)`

Die Methode zeigt die uebergebene Gesamtberechnungszeit im Informationsbereich der Pulssequenz an. Dazu wird die uebergebene Zeit zunaecht in Minuten und Sekunden umgerechnet.

Parameter:

`totaltime` - Die Gesamtberechnungzeit in ms.

`void` **`updateTotalTime`**`()`

Die Methode ist in dieser abstrakten Oberklasse leer. In den abgeleiteten Klassen wird in ihr die Gesamtzeit der Berechnung berechnet. Die Methode muss in dieser Klasse nur vorhanden sein, damit der Entwickler sich nicht mehr um die automatische Aktualisierung der Darstellung der Gesamtzeit kuemmern muss.

`void` **`displayRemainingTime`**`(long remainingtime)`

Die Methode zeigt die uebergebene Restberechnungszeit im Informationsbereich der Pulssequenz an. Dazu wird die uebergebene Zeit zunaecht in Minuten und Sekunden umgerechnet.

Parameter:

`remainingtime` - Die Restberechnungzeit in ms.

`public int` **`getCoil`**`()`

Die Methode liefert den Index der selektierten Aufnahmespule zurueck.

Rückgabewert:

Der Index der selektierten Spule in der Auswahlbox.

`public double` **`getSNRatio`**`()`

die Methode berechnet das aktuelle Signal-zu-Rausch-Verhaeltnis aufgrund der eingestellten Sequenzparameter. /n Anmerkung zum Signal-Rausch-Verhaeltnis: Bei 1 Tesla hat ein Probenvolumen Wasser mit 8*10E18 Protonen ein SR-Verhaeltnis von 1. Da ein ml Wasser 6,691*10E22 Protonen hat, entspricht das einem Volumen von 0,074626865 mm^3 Wasser.

`public void` **`displayPixelSize`**`()`

Die Methode berechnet die aktuelle Pixelgroesse aufgrund der Einstellungen von Matrixgroesse und Field of View. Die Pixelgroesse wird dann im Informationsbereich angezeigt.

`public void` **`displaySNRatio`**`()`

Die Methode berechnet das aktuelle Signal-zu-Rausch-Verhaeltnis und zeigt dieses im Informationsbereich an.

`public void` **`addProperty`**`(java.lang.String className, java.lang.String methodName)`

Diese Methode fügt ein Element in die Eigenschaftsliste kSpaceManipulator ein.

Parameter:

`className` - Name der Berechnungsklasse des Artefakt-Simulators.
`methodName` - Name der Methode, die die korrekte Berechnungsfunktion enthaelt.

`public java.lang.String` **`getArtefactMethod`**`()`

Die Methode liefert den Namen der Methode der ausgeaehlten Artefakt-Klasse zurueck, die die fuer diese Pulssequenz korrekte Berechnungsvorschrift enthaelt.

Rückgabewert:

Der Name der Methode der Artefakt-Klasse, die die korrekte Berechnungsvorschrift enthaelt.

`public Artefact` **`getArtefact`**`()`

Die Methode liefert eine Referenz auf einen ggf. ausgewählten Artefakt-Simulator zurueck.

Rückgabewert:

Eine Referenz auf die Berechnungsklasse des ausgewaehlten Artefakt-Simulators.

Klasse SaturationRecovery

public class **SaturationRecovery** extends `Pulsesequence`

Die Klasse berechnet ein Intensitaetsbild mittels einer Saturation-Recovery-Sequenz. Einziger Sequenzparameter ist die Repititionszeit.

Klassen- und Instanzvariablen

`private int`	`trTime` Repititionszeit.

Konstruktoren

`public` **`SaturationRecovery`**`()`

Standardkonstruktor.

Methoden

`protected void` **`addSequenceParameterToImage`**`()`

Die Methode fuegt die Sequenzparameterwerte dem berechneten Bild hinzu. Dadurch kann eine DICOM-Datei mit allen relevanten Informationen gespeichert werden.

`public void` **`calculate`**`()`

Die Methode berechnet aus den Rohdatenmatrizen und den Sequenzparameterwerten die Intensitaetsmatrix. Ausserdem ist sie fuer die Aktualisierung der Fortschrittsanzeige und das Einhalten des Simulationszeitfaktors verantwortlich.

`public void` **`setTRTime`**`(int tr)`

Die Methode setzt die Instanzvariable fuer die TR-Zeit nach dem Auslesen aus dem entsprechenden Bedienelement.

Parameter:

`tr` - TR-Zeit.

`public int` **`getTRTime`**`()`
Die Methode liefert die aktuell eingestellte Repititionszeit der Pulssequenz zurueck.

Rückgabewert:
TR-Zeit.

Klasse SaturationRecoveryUI

public class **SaturationRecoveryUI** extends `PulsesequenceUI`

Diese Klasse erzeugt die Bedienelemente fuer eine Saturation-Recovery-Sequenz und stellt diese dar.

Klassen- und Instanzvariablen	
`private SliderPanel`	`ptrPanel` Schieberegler fuer die Repititionszeit.
`private int`	`trTime` Voreingestellte TR-Zeit von 1000 ms.

Konstruktoren

`public` **`SaturationRecoveryUI`**`()`
Standardkonstruktor.

Methoden

`public void` **`jbInit`**`()`
`throws java.lang.Exception`
Diese Methode fuegt die sequenzspezifischen Steuerelemente in den Darstellungsbereich ein. Zunaechst wird fillPanel der Oberklasse aufzurufen, um Start- und Abbrechenknopf einzufuegen, die in jeder Sequenz enthalten sind.

`public void` **`setValues`**`()`
Die Methode liest die aktuellen Sequenzparameterwerte aus den Bedienelementen aus und speichert sie in den entsprechenden Instanzvariablen.

`public void` **`getValues`**`()`
Die Methode setzt die Werte der Bedienelemente der Sequenz auf die in den entsprecheden Instanzvariablen gespeicherten Werte.

`void` **`pbStart_actionPerformed`**`(java.awt.event.ActionEvent e)`
Die Methode wird aufgerufen, wenn der Startknopf gedrueckt wurde. Es wird eine neue Instanz der Berechnungsklasse erzeugt. Dieser werden die aktuellen Sequenzparameter mitgeteilt. Dann wird die Berechnung des Intensitaetsbildes angestossen.

Parameter:
`e` - Das Ereignis, das beim Betaetigen des Startknopfes ausgeloest wird.

`void` **`pbStop_actionPerformed`**`(java.awt.event.ActionEvent e)`
Die Methode wird aufgerufen, wenn der Stopknopf gedrueckt wurde. In diesem Fall wird der Thread zur Berechnung des Bildes gestoppt und der Status des Start- und Stopknopfes zurueckgesetzt.

Parameter:

`e` - Das Ereignis, das beim Betaetigen des Stoppknopfes ausgeloest wird.

`void` **`updateTotalTime`**`()`

Die Methode berechnet aus den aktuell eingestellten Sequenzparameterwerten die theoretische Gesamtdauer der Berechnung des Bildes. Diese Zeit wird im Kontrollbereich des Hauptfensters angezeigt.

Klasse SequenceCombo

public class **SequenceCombo** extends JComboBox

Die Klasse verwaltet die Auswahlbox zur Auswahl der Pulssequenz. Die Methode *getSequences* liest die Sequenzklassen und Sequenznamen aus der *Property*-Datei aus und erzeugt ueber *Reflections* Instanzen der *SequenzUI*-Klassen. Auch die Ereignisbehandlung der Auswahlbox findet in dieser Klasse statt. Jetzt werden auch die eingestellten Werte ueber *setValue* in der SequenzUI-Klasse gespeichert und ueber *getValue* wieder gelesen.

Klassen- und Instanzvariablen

`private VMRTFrame`	`mainFrame` Referenz auf das Hauptfenster.
`private int`	`oldindex` Der Index der zuletzt selektierten Sequenz in der Auswahlbox.
`private JPanel`	`pSequenceInfoPanel` Referenz auf das Infofenster fuer die Pulsseqeunzen.
`private JPanel`	`pSequenceSettingPanel` Referenz auf das Einstellungsfenster fuer die Pulsseqeunzen.
`private java.util.Vector`	`sequenceClasses` Vektor zur Aufnahme der Pulssequenz-Klassen.

Konstruktoren

```
public SequenceCombo(VMRTFrame frame,
           JPanel seqSetPanel,
           JPanel seqInfoPanel)
```

Der Konstruktor setzt die Referenzen zum Hauptfenster, zum Einstellungs- und zum Infofenster der Pulssequenzen. Ausserdem wird ein leerer Vektor zur Aufnahme der Pulssequenzen angelegt.

Parameter:

`frame` - Rueckreferenz zum Hauptfenster.
`seqSetpanel` - Referenz auf das Einstellungsfenster fuer die Pulssequenzen.
`seqInfoPanel` - Referenz auf das Infofenster fuer die Pulssequenzen.

Methoden

`public void` **`getSequences`**`()`

Die Methode laedt die Klassennamen der Sequenz-berechnungsklassen und die Sequenznamen aus der Property-Datei. Eine Instanz der Sequenz und deren UI-Klasse wird ueber den Namen der Sequenz in der Auswahlliste erzeugt. Eine Referenz auf die PulssequenzUI-Klasse wird im Vektor sequenceClasses gespeichert. Dieses Konzept ermoeglicht es, dem Programm ganz einfach neue Pulssequenzen hinzuzufuegen. Fuer jede Sequenz muessen zwei Klassen implementiert werden, eine UI-Klasse und eine Berechnungsklasse, anschliessend muessen Name und Klassenname der Berechnungsklasse in die Property-Datei eingetragen werden.

`private void` **`cb_itemStateChanged`**`(java.awt.event.ItemEvent e)`

Die Methode wird aufgerufen, wenn eine neue Pulssequenz aus der Auswahlbox ausgewaehlt wird. Der Bereich zur Darstellung der Oberflaechenelemente der Pulssequenz wird dann geloescht und anschliessend mit den Elementen der

neuen Pulssequenz neu gefuellt.

Parameter:

`e` - Das Ereignis beim Auswaehlen einer anderen Pulssequenz aus der Auswahlbox.

Klasse SeriesLoader

public class **SeriesLoader** extends java.lang.Thread

Die Klasse implementiert eine Routine zum Laden einer Serie von DICOM-Bildern. Damit waehrend des Ladens der *DICOM-Viewer* nicht blockiert wird, ist die Klasse als *Thread* implementiert. Waehrend des Ladens der Bilder wird die Fortschrittsanzeige im *DICOM-Viewer* aktualisiert.

Klassen- und Instanzvariablen

`private java.io.File`	`dcmfile` Der Dateiname (inkl.
`private ViewerFrame`	`mainFrame` Rueckreferenz zum aufrufenden Fenster.
`private JProgressBar`	`progressbar` Eine Referenz auf die Fortschrittsanzeige im DICOM-Viewer.

Konstruktoren

```
public SeriesLoader(ViewerFrame mf,
          java.io.File f)
```

Der Konstruktor setzt die uebergebenen Referenzen auf das aufrufende Fenster, auf eine Datei der zu ladenden Serie und auf die Fortschrittsanzeige im DICOM-Viewer.

Parameter:

`mf` - Eine Rueckreferenz zum aufrufenden Fenster.

`f` - Der Dateiname (inkl. Pfad) eines Bildes der zu ladenden Serie.

Methoden

```
public void run()
```

Die Methode laedt alle Bilder, die zu der Serie gehoeren, zu der auch die eine explitzit angegebene Datei gehoert. Dazu werden alle Dateien im gleichen Verzeichnis darauf ueberprueft, ob deren Patientenname mit dem des einen explizit angegebenen Bildes uebereinstimmen. Ist dies der Fall, gehoert das Bild zur gleichen Serie und wird geladen.

Klasse SliderPanel

public class **SliderPanel** extends JPanel

Diese Klasse stellt einen Rahmen mit Beschriftung, Textfeld und Schieberegler zur Verfuegung. Die Werte von Schieberegler und Textfeld werden synchronisiert. Im Textfeld sind nur ganzzahlige Eingaben in einem bestimmten Wertebereich zulaessig.

Klassen- und Instanzvariablen

`(package private) java.awt.GridBagLayout`	`gridBagLayout1` Layout zur Anordnung der einzelnen Oberflaechenelemente.
`private boolean`	`ignoreStateChange` Wenn dieser Merker wahr ist, ignoriert der Schieberegler das Event "item-State-Changed" Der Schieberegler und das Textfeld werden synchronisiert, allerdings soll es auch moeglich sein, im Textfeld groessere Werte einzugeben als fuer den Slider erlaubt ist.
`private int`	`imajorspacing`

	Grosser Abstand der Ticks.
private int	imaximum Maximalwert fuer den Schieberegler.
private int	iminimum Minimalwert fuer den Schieberegler.
private int	iminorspacing Kleiner Abstand der Ticks.
private int	itmaximum Maximalwert fuer das Textfeld.
private int	itminimum Minimalwert fuer das Textfeld.
private java.lang.String	label Inhalt des Beschriftungsfeldes.
(package private) JLabel	lSliderLabel Beschriftung des Schiebereglers.
(package private) JLabel	lUnitLabel Die Beschriftung fuer ein optionales Einheitenzeichen.
(package private) JSlider	slSlider Der Schieberegler.
private java.lang.String	strtoolTip Tooltip-Text.
(package private) JTextField	tfSliderTextField Das Textfeld.
private java.lang.String	unit Inhalt der Beschriftung fuer die Einheit.

Konstruktoren

`public` **`SliderPanel`**`()`

Der Konstruktor erzeugt die einzelnen Bedienelemente und stellt diese entsprechend dem gewaehlten Layout dar.

```
public SliderPanel(java.lang.String lab,
           int min,
           int max)
```

Der Konstruktor setzt die Beschriftung sowie den Minimal- und Maximalwert des Schiebereglers und des Textfeldes. Die Grenzwerte sind bei Aufruf dieses Konstruktors fuer den Schieberegler und das Textfeld identisch.

Parameter:

`lab` - Beschriftung des Schiebereglers.
`min` - Minimalwert fuer den Schieberegler und das Textfeld.
`max` - Maximalwert fuer den Schieberegler und das Textfeld.

```
public SliderPanel(java.lang.String lab,
           int min,
           int max,
           java.lang.String u)
```

Der Konstruktor setzt die Beschriftung sowei den Minimal- und Maximalwert des Schiebereglers und des Textfeldes. Die Grenzwerte sind bei Aufruf dieses Konstruktors fuer den Schieberegler und das Textfeld identisch. Ausserdem wird die Beschriftung fuer das Einheitenzeichen gesetzt.

Parameter:

`lab` - Beschriftung des Schiebereglers.
`min` - Minimalwert fuer den Schieberegler und das Textfeld.
`max` - Maximalwert fuer den Schieberegler und das Textfeld.
`u` - Beschriftung fuer das Einheitenzeichen.

Methoden

`public void` **`fillPanel`**`()`

Die Methode stellt die Bedienelemente dieser Klasse dar.

`public void` **`setToolTip`**`(java.lang.String tip)`

Die Methode legt den ToolTip-Text fest.

Parameter:

`tip` - Der ToolTip-Text.

`public void` **`setTextRange`**`(int min,`
`int max)`

Die Methode legt das erlaubte Intervall fuer Werte im Textfeld fest. Dieses kann von dem Intervall fuer den Schieberegler getrennt gesetzt werden.

Parameter:

`min` - Kleinster erlaubter Wert fuer das Textfeld.

`max` - Groesster erlaubter Wert fuer das Textfeld.

`public void` **`setTickSpacing`**`(int major,`
`int minor)`

Die Methode legt das kleine und grosse TickSpacing fuer den Schieberegler fest.

Parameter:

`minor` - Das kleine Tickspacing.

`major` - Das grosse Tickspacing.

`void` **`jbInit`**`()`
`throws java.lang.Exception`

Die Methode erzeugt die einzelnen Bedienelemente dieser Klasse und stellt diese dar.

`void` **`slSlider_stateChanged`**`()`

Die Methode wird aufgerufen, wenn sich Zustand des Schieberegler aendert. Dann wird das Textfeld mit dem Schieberegler synchronisiert.

`void` **`tfSliderTextField_focusLost`**`()`

Die Methode ueberprueft die Gueltigkeit des Wertes im Textfeld, wenn dieses den Fokus verliert.

`public int` **`getValue`**`()`

Die Methode liefert den aktuellen Wert des Textfeldes.

Rückgabewert:

Der aktuelle Wert des Textfeldes.

`public void` **`setValue`**`(int ivalue)`

Die Methode setzt den Wert des Schiebereglers. Dies ist noetig, um den Schieberegler mit dem Textfeld zu synchronisieren, wenn in das Textfeld ein Wert eingegeben wurde.

Parameter:

`ivalue` - Der Wert, auf den der Schieberegler gesetzt werden soll.

`public JTextField` **`getTextFieldReference`**`()`

Die Methode liefert eine Referenz auf das Textfeld zurueck. Dadurch koennen von aussen EventListener hinzugefuegt

werden.

Rückgabewert:
Eine Referenz auf das Textfeld.

Klasse ViewerFrame

public class **ViewerFrame** extends JFrame

Die Klasse implementiert das Hauptfenster des DICOM-Viewers mit all seinen Bedienelementen und deren Ereignisbehandlung. Ein Grossteil der Funktionalitaet des DICOM-Viewers steckt ebenfalls in dieser Klasse.

Klassen- und Instanzvariablen	
`private ButtonGroup`	`bgNumOfImages` Knopfgruppe zur Gruppierung der Knoepfe rb1Image und rb4Images.
`private ButtonGroup`	`bgPlaceNextImage` Gruppe fuer die Knoepfe zur Auswahl der Bildplazierung.
`private int`	`curr_x` Aktuelle x-Mausposition auf der Zeichenflaeche.
`private int`	`curr_y` Aktuelle y-Mausposition auf der Zeichenflaeche.
`private java.awt.Cursor`	`DEFAULT_CURSOR` Standard-Mauszeiger.
`private ImageIcon`	`i1Image` Icon fuer den Knopf zur Darstellung eines Einzelbildes.
`private ImageIcon`	`i4Images` Icon fuer den Knopf zur gleichzeitigen Darstellung von 4 Bildern.
`private ImageIcon`	`iAngel` Icon fuer den Knopf zur Winkelmessung.
`private ImageIcon`	`iAnimation` Icon fuer den Knopf zur Anzeige des Animationsfensters.
`private ImageIcon`	`iDelete` Icon fuer den Knopf zum Loeschen des selektierten Bildes.
`private ImageIcon`	`iDistance` Icon fuer den Knopf zur Abstandsmessung.
`private ImageIcon`	`iGrayValue` Icon fuer den Knopf fuer die Grauwertmessungs-Funktion.
`private ImageIcon`	`iHistogram` Icon fuer den Knopf zur Anzeige des Histogramm-Fensters.
`private ImageIcon`	`iInfoText` Icon fuer den Knopf zum Ein- bzw.
`private ImageIcon`	`iInvert` Icon fuer den Knopf zum Invertieren des selektierten Bildes.
`private ImageIcon`	`iKSpace` Icon fuer den Knopf zur Anzeiges des k-Raum-Fensters.
`private ImageIcon`	`iKSpaceMan` Icon fuer den Knopf zur Anzeige des k-Raum-Manipulator-Fensters.
`private ImageIcon`	`iMirror` Icon fuer den Knopf zum Spiegeln des selektierten Bildes.
`private ImageIcon`	`iNew` Icon fuer den Knopf zum Loeschen aller Bilder.
`private ImageIcon`	`iOpen` Icon fuer den Knopf zum Laden eines Einzelbides.
`private ImageIcon`	`iOpenSeries` Icon fuer den Knopf zum Laden einer Bildserie.
`private ImageIcon`	`iPointer` Icon fuer den Knopf zur Auswahl des Standardmauszeigers.
`private ImageIcon`	`iPrint` Icon fuer den Knopf zum Drucken des Zeichenflaecheninhalts.

private ImageIcon	iProjection1 Icon fuer den Knopf zur Projektionsberechnung (1.
private ImageIcon	iProjection2 Icon fuer den Knopf zur Projektionsberechnung (2.
private ImageIcon	iRotate Icon fuer den Knopf der Rotations-Funktion.
private ImageIcon	iSave Icon fuer den Knopf zum Speichern eines Bildes.
private ImageIcon	iZoom Icon fuer den Knopf der Lupe-Funktion.
private int	lastCenter Fensterungszentrum-Einstellungen beim letzte Mausklick.
private int	lastMouseClickX Letze Mausklick-x-Koordinate.
private int	lastMouseClickY Letze Mausklick-y-Koordinate.
private long	lastTime Zeitpunkt des letzten Mausklicks im Hauptfenster.
private int	lastWindow Fensterungsbreite-Einstellungen beim letzte Mausklick.
private JLabel	lNextImageAt Beschriftung fuer die Auswahlelemente fuer die Bildplatzierung.
private JLabel	lPosition Beschriftung fuer die Positionsangabe des Mauszeigers.
private JLabel	lPositionText Ausgabefeld fuer die Positionsangabe des Mauszeigers.
private JMenuBar	menuBar1 Die Menueleiste.
private JMenu	menuFile Das Dateimenue.
private JMenuItem	menuFileExit Dateimenueeintrag 'Beenden'.
private JMenuItem	menuFileOpen Dateimenueeintrag 'Oeffnen'.
private JMenuItem	menuFilePrint Dateimenueeintrag 'Drucken'.
private JMenuItem	menuFileSave Dateimenueeintrag 'Speichern'.
private JMenu	menuHelp Hilfemenue.
private JMenuItem	menuHelpAbout Hilfemenueeintrag 'Ueber...'.
private int	mouseClicks Anzahl der Mausklicks (benoetigt zur Abstands- und Winkelmessung).
private JButton	pbAnimation Knopf zum Oeffnen des Animations-Fensters.
private JButton	pbCalcMIP Knopf zur Berechnung eines MIP-Bildes.
private JButton	pbClearAll Knopf zum Loeschen aller Bilder.
private JButton	pbDelete Knopf zum Loeschen des selektierten Bildes.
private JButton	pbHistogram Knopf zur Darstellung des Histogramms des selektierten Bildes.
private JButton	pbInvert Knopf zum Invertieren des selektierten Bildes.
private JButton	pbKSpace Knopf zur Darstellung des k-Raums des selektierten Bildes.
private JButton	pbKSpaceManip Knopf zum Oeffnen des k-Raum-Manipulators.
private JButton	pbLoad Knopf zum Laden eines neuen DICOM-Bildes.

private JButton	pbLoadSeries Knopf zum Laden einer neuen DICOM-Bildserie.
private JButton	pbMirror Knopf zur Auswahl des Spiegel-Werkzeugs.
private JButton	pbOptWindowing Knopf zur Durchfuehrung einer optimalen Fensterung.
private JButton	pbPrint Knopf zum Drucken des Zeichenflaecheninhalts.
private JButton	pbProjection1 Knopf zur Berechnung einer 90-Grad-Projektion (1.
private JButton	pbProjection2 Knopf zur Berechnung einer 90-Grad-Projektion (2.
private JButton	pbResetWindowing Knopf zum Zuruecksetzen der Fensterung auf Center=2048 und Window=4096.
private JButton	pbRotate Knopf zur Auswahl des Rotations-Werkzeugs.
private JButton	pbSave Knopf zum Speichern eines Bildes.
(package private) MyPanel	pCanvas Zeichenflaeche (Bildflaeche / Canvas).
private SliderPanel	pCenter Schieberegler fuer die Helligkeit (Zentrum des Fensters).
private JPanel	pNextImageAt Panel zur Gruppierung der Elemente, die die Auswahl der Bildplazierung fuer neu erzeugte Bilder ermoeglicht.
private JProgressBar	prbarProgress Fortschrittsanzeige.
private java.util.Properties	preferences Hashtabelle für die Standardeinstellungen
private SliderPanel	pTimeFactor Schieberegler zur Auswahl des Simulationszeitfaktors (0-100%).
private JPanel	pToolbar Rahmen fuer den Werkzeugbereich am rechten Rand des Fensters.
private SliderPanel	pWindow Schieberegler fuer den Kontrast (Breite des Fensters).
(package private) JToggleButton	rb1Image Knopf zur Auswahl der Darstellung von nur einem Bild.
(package private) JToggleButton	rb4Images Knopf zur Auswahl der Darstellung von 4 Bildern gleichzeitig.
(package private) JToggleButton	rbAngel Knopf zur Auswahl des Winkelmesswerkzeugs.
(package private) JToggleButton	rbDistance Knopf zur Auswahl des Messwerkzeugs (Distanz).
(package private) JToggleButton	rbGrayValue Knopf zur Auswahl des Grauwert-Messwerkzeugs.
private JRadioButton	rbNextImageAtFreePlace Knopf zur Erzeugung des naechsten Bildes an der naechsten freien Position.
(package private) JRadioButton	rbNextImageAtSelection Knopf zur Erzeugung des naechsten Bildes an der selektierten Position.
private JToggleButton	rbPointer Knopf zur Auswahl des Standard-Mauszeigers.
(package private) JToggleButton	rbZoom Knopf zur Auswahl der Lupe.
(package private) JScrollBar	sbHorScrollbar Horizontale Bildlaufleiste fuer die Zeichenflaeche.
private JLabel	statusBar Statusleiste.
(package private) JToggleButton	tbImageText Knopf zur Darstellung der Bildbeschriftungen (Nummern, TR, TI, u.a.).
private JPanel	tp3D Karteikarte fuer die 3D-Werkzeuge.

private JPanel	tpExtras Karteikarte fuer die Zusatzfunktionen.
private JPanel	tpLayout Karteikarte fuer alle Layout-Funkionen.
private JTabbedPane	tpMain Hauptrahmen fuer die Karteikarten.
private JPanel	tpTools Karteikarte fuer alle Werkzeuge (Lupe, Distanzmessung, Winkelmessung, Grauwertmessung, Invertieren, Spiegeln, Drehen).
private JPanel	tpWindow Karteikarte fuer die Fensterungs-Funktionen (Helligkeit und Kontrast).
private boolean	unsavedChanges Merker fuer nichtgespeicherte Aenderungen.
private java.awt.Cursor	WINDOW_CURSOR Mauszeiger fuer die Fensterung.
private java.awt.Cursor	ZOOM_CURSOR Mauszeiger fuer die Zoom-Funktion, die Distanzmessung und die Winlemessung.

Konstruktoren

public **ViewerFrame**()
Standardkonstruktor. Er initialisiert die Bedienelemente des DICOM-Viewers.

Methoden

private void **jbInit**()
throws java.lang.Exception
Die Methode richtet die Bedienelemente ein und stellt diese dar.

---

private void **pCanvas_mouseDragged**(java.awt.event.MouseEvent e)
Die Methode implementiert die Fensterung des selektierten Bildes mittels Mausbewegung. Eine Bewegung der Maus in x-Richhtung veraendert das Zentrum, eine Bewegung in y-Richtung die Breite. Ausserdem wird fuer die Distanz- und Winkelmessung die aktuelle Mausposition in der Canvas-Klasse gesetzt.

Parameter:
e - Das Ereignis, das beim Bewegen der Maus mit gedrueckter Maustaste ausgeloest wird.

---

private void **helpAbout_actionPerformed**()
Die Methode wird aufgerufen, wenn die Aktion Hilfe | Info aus dem Menue aufgerufen wurde. Das Informationsfenster wird aufgeblendet.

---

protected void **processWindowEvent**(java.awt.event.WindowEvent e)
Die Method wird aufgerufen, wenn der Schliessen-Knopf oben rechts am Fenster betaetigt wird. Die Methode muss ueberschrieben werden, damit das Programm bei einem System-Close beendet werden kann.

Parameter:
e - Das Ereignis beim Betaetigen des Schliessen-Knopfs.

---

private void **updateCW**()
Die Methode aktualisiert die Center- und Window-Einstellung in der selektierten ImagePlus-Klasse.

---

private void **updateCenter**()
Die Methode aktualisiert die Center-Einstellung in der selektierten ImagePlus-Klasse.

---

private void **updateWindow**()

Die Methode aktualisiert die Window-Einstellung in der selektierten ImagePlus-Klasse.

`private void` **`pbLoadSeries_actionPerformed`**`()`

Die Methode blendet einen Dateiauswahldiaolg auf. Nach Auswahl einer Datei aus der Liste, wird die Bildserie, die zu dem selektierten Bild gehoert geladen und auf der Zeichenflaeche dargestellt.

`private void` **`pbLoad_actionPerformed`**`()`

Die Methode wird aufgerufen, wenn der 'Bild laden'-Knopf gedrueckt wurde oder die Aktion Datei | Oeffnen aufgerufen wurde. Ein Dateidialogfenster wird aufgeblendet, wo der Benutzer das zu oeffnende Bild auswaehlen kann. Der ausgewaehlte Dateipfad wird zurueckgeliefert und anha´nd dessen wird das gewaehlte Bild geladen und dargestellt.

`public void` **`loadDcmImage`**`(java.lang.String strDirectory, java.lang.String strFile)`

Die Methode laedt aufgrund der Angabe des Dateinamens und der Verzeichnisses ein DICOM-Bild. Dabei wird zunaechst ein DcmDataObject erzeugt, daraus dann wiederum ein DcmImage-Objekt und aus diesem dann schliesslich ein ImagePlus-Objekt.

Parameter:

`strDirectory` - Das Verzeichnis, in dem das zu ladende Bild steht.

`private void` **`pbSave_actionPerformed`**`()`

Die Methode wird aufgerufen, wenn der 'Speichern'-Knopf gedrueckt wurde oder die Aktion Datei | Speichern aufgerufen wurde. Ein Dateidialogfenster wird aufgeblendet, wo der Benutzer den Pfad und den Namen der zu speichernden Datei angeben kann. Das selektierte Bild wird dann an der angegebenen Position gespeichert.

`private void` **`pbPrint_actionPerformed`**`()`

Die Methode wird aufgerufen, wenn der 'Drucken'-Knopf gedrueckt wurde oder die Aktion Datei | Drucken aufgerufen wurde. Der Systemabhaengige Druckdialog wird aufgeblendet. Nach dessen Bestaetigung wird der Inhalt der Zeichenflaeche auf dem ausgewaehlten Drucker ausgegeben.

`private void` **`pbDelete_actionPerformed`**`()`

Die Methode wird aufgerufen, wenn der Knopf zum Loeschen des selktierten Bildes gedrueckt wurde. Es wird zunaecht ein Dialog aufgeblendet, der eine Sicherheitsabfrage darstellt und nach dessen Bestaetigung wird das selektierte Bild geloescht.

`private void` **`pbClearAll_actionPerformed`**`()`

Die Methode wird aufgerufen, wenn der Knopf zum Loeschen aller Bilder gedrueckt wurde. Es wird zunaecht ein Dialog aufgeblendet, der eine Sicherheitsabfrage darstellt und nach dessen Bestaetigung werden alle Bilder geloescht.

`private void` **`pCanvas_mouseMoved`**`(java.awt.event.MouseEvent e)`

Die Methode fuehrt verschiedene Funktionen aus, je nachdem welches Werkzeug ausgewaehlt wurde. In jedem Fall wird die Position des Mauszeiger auf der Zeichenflaeche in der Positionsanzeige ausgegeben. Ist ein anderes Werkzeug ausser dem Standard-Mauszeiger aktiviert (z.B. Lupe), so wird der Inhalt der Zeichenflaeche staendig neu gezeichnet. Was genau gezeichnet wird, wird in der paint-Methode der Zeichenflaeche festgelegt.

Parameter:

`e` - Das Ereignis beim Bewegen der Maus ueber die Zeichenflaeche.

`private void` **`pCanvas_mouseExited`**`()`

Die Methode wird aufgerufen, wenn der Mauszeiger die Zeichenflaeche verlaesst. Die Positionsanzeige wird dann geloescht und ein Standardmauszeiger eingestellt. Ausserdem werden einige Ausnahmewerte fuer die Grauwert-, Distanz- und Winkelmessung gesetzt.

`private void` **`rb4Images_actionPerformed`**`()`

Die Methode wird aufgerufen, wenn der Knopf zur gleichzeitigen Anzeige von 4 Bildern gedrueckt wurde. Der Inhalt der Zeichenflaeche wird dann neu aufgebaut.

`private void` **`rb1Image_actionPerformed`**`()`

Die Methode wird aufgerufen, wenn der Knopf zur Anzeige von nur einem Bild gedrueckt wurde. Der Inhalt der Zeichenflaeche wird dann neu aufgebaut.

`private void` **`pCanvas_mousePressed`**`(java.awt.event.MouseEvent e)`

Die Methode wird aufgerufen, wenn eine Maustaste gedrueckt wird, waehrend sich der Mauszeiger ueber der Zeichenflaeche befindet. In diesem Fall werden die Mausklickkoordinaten abgespeichert und die Fensterungswerte des zuletzt selektierten Bildes gesichert.

`private void` **`pCanvas_mouseReleased`**`(java.awt.event.MouseEvent e)`

Die Methode wird aufgerufen, wenn eine Maustaste losgelassen wird, waehrend sich der Mauszeiger ueber der Zeichenflaeche befindet. Die Mauskoordinaten werden dann gesichert. Dies wird fuer die Winkelmessung benoetigt.

`private void` **`pCanvas_mouseClicked`**`(java.awt.event.MouseEvent e)`

Die Methode wird aufgerufen, wenn in der Zeichenflaeche ein Mausknopf gedrueckt wird. Es werden dann je nach ausgewaehltem Werkzeug verschiedene Funktionen durchgefuehrt. In jedem Fall werden die Fensterungseinstellungen des bisher selektierten Bildes gesichert. Beim neu selektierten Bild werden die alten Fenstereinstellungen wiederhergestellt. Anschliessend wird der Inhalt der Zeichenflaeche aktualisiert. Was genau dabei passiert, ist in der paint-Methode der Zeichenflaeche festgelegt.

Parameter:

`e` - Das Ereignis beim Druecken einer Maustaste ueber der Zeichenflaeche.

`private void` **`tbImageText_mouseClicked`**`()`

Die Methode wird aufgerufen, wenn der Knopf zum Ein- bzw. Ausschalten der Bildbeschriftung gedrueckt wird. Der Inhalt der Zeichenflaeche wird dann neu aufgebaut, entweder mit oder ohne Beschriftungen.

`private void` **`fileOpen_actionPerformed`**`()`

Die Methode wird aufgerufen, wenn die Aktion Datei | Oeffnen aus dem Menue aufgerufen wird. Diese Methode wird nur auf die entsprechende Methode des Oeffnen-Knopfes in der Werkzeugleiste umgeleitet.

`private void` **`fileSave_actionPerformed`**`()`

Die Methode wird aufgerufen, wenn die Aktion Datei | Speichern aus dem Menue aufgerufen wird. Diese Methode wird nur auf die entsprechende Methode des Speichern-Knopfes in der Werkzeugleiste umgeleitet.

`private void` **`filePrint_actionPerformed`**`()`

Die Methode wird aufgerufen, wenn die Aktion Datei | Drucken aus dem Menue aufgerufen wird. Diese Methode wird nur auf die entsprechende Methode des Drucken-Knopfes in der Werkzeugleiste umgeleitet.

`private void` **`fileExit_actionPerformed`**`()`

Die Methode wird aufgerufen, wenn die Aktion Datei | Beenden aus dem Menue aufgerufen wird. Ggf. wird eine Abfrage aufgeblendet, ob die neuen Bilder verworfen werden sollen. Nach deren Bestaetigung wird das Programm beendet.

`private void` **`pbMirror_actionPerformed`**`()`

Die Methode wird aufgerufen, wenn der Knopf zum Spiegeln des selektierten Bildes gedrueckt wurde. Es wird dann

die entsprechende Methode fuer das selektierte Bild aufgerufen und der Zeichenflaecheninhalt neu dargestellt.

`private void` **`pbRotate_actionPerformed`**`()`

Die Methode wird aufgerufen, wenn der Knopf zum Rotieren des selektierten Bildes gedrueckt wurde. Es wird dann die entsprechende Methode fuer das selektierte Bild aufgerufen und der Zeichenflaecheninhalt neu dargestellt.

`private void` **`pbInvert_actionPerformed`**`()`

Die Methode wird aufgerufen, wenn der Knopf zum Invertieren des selektierten Bildes gedrueckt wurde. Es wird dann die entsprechende Methode fuer das selektierte Bild aufgerufen und der Zeichenflaecheninhalt neu dargestellt.

`public void` **`pbOptWindowing_actionPerformed`**`()`

Die Methode wird aufgerufen, wenn der Knopf zum optimalen Fenstern des selektierten Bildes gedrueckt wurde. Es wird dann die entsprechende Methode fuer das selektierte Bild aufgerufen und der Zeichenflaecheninhalt neu dargestellt. Ausserdem werden die Einstellungen der Schieberegler fuer die Fensterung angepasst.

`private void` **`pbHistogram_actionPerformed`**`()`

Die Methode wird aufgerufen, wenn der Knopf zur Darstellung des Histogramms zum selektierten Bild gedrueckt wurde. Es wird dann ein Fenster aufgeblendet, in dem das Histogramm des selektierten Bildes zu sehen ist.

`private void` **`pbKSpaceManip_actionPerformed`**`()`

Die Methode wird aufgerufen, wenn der Knopf zur Darstellung des k-Raums-Manipulators fuer das selektierte Bild gedrueckt wurde. Es wird dann ein Fenster aufgeblendet, in dem das Originalbild, der original k-Raum, der manipulierte k-Raum und die Ruecktransformation des manipulierten k-Raums dargestellt werden. Der Benutzer hat die Moeglichkeit, verschiedene Manipulationen am k-Raum vorzunehmen und dann die Ruecktransformationen berechnen zu lassen.

`private void` **`pbKSpace_actionPerformed`**`()`

Die Methode wird aufgerufen, wenn der Knopf zur Darstellung des k-Raums des selektierten Bild gedrueckt wurde. Es wird dann ein Fenster aufgeblendet, in dem der k-Raum (Magnitudenbild) des selektierten Bildes zu sehen ist. Ausserdem kann man sich das Phasenbild sowie den Real- und Imaginaerteil anzeigen lassen.

`private int` **`getImageMouseIsOn`**`(int x,`
`int y)`

Die Methode bestimmt aufgrund der x- und y-Koordinate des Mauszeigers auf der Zeichenflaeche, ueber welchem Bild sich der Mauszeiger gerade befindet. Dabei geht die Einstellung 1 Bild / 4 Bilder und die Bildlaufleistenposition mit in die Berechnung ein.

Parameter:

`x` - Die aktuelle x-Koordinate des Mauszeigers.

`y` - Die aktuelle y-Koordinate des Mauszeigers.

Rückgabewert:

Nummer des Bildes, ueber dem sich der Mauszeiger befindet.

`public JProgressBar` **`getProgressBar`**`()`

Die Methode liefert eine Referenz auf die Fortschrittsanzeige zurueck.

Rückgabewert:

Eine Referenz auf die Fortschrittsanzeige.

`public void` **`setChanges`**`()`

Die Method setzt den Merker fuer nicht gespeicherte Bilder.

`private void` **`resetChanges`**`()`

Die Methode setzt den Merker fuer nicht gespeicherte Bilder zurueck.

`private ImagePlus` **`getSelectedImage`**`()`

Die Methode liefert das ImagePlus-Objekt des zur Zeit selektierten Bildes.

Rückgabewert:

Das ImagePlus-Objekt des selektierten Bildes.

`private void` **`sbHorScrollbar_adjustmentValueChanged`**`()`

Die Methode wird aufgerufen, wenn sich der Zustand des Scrollbalkens veraendert hat. Der Scrollbalken wird dann aktualisiert.

`private void` **`pbAnimation_actionPerformed`**`()`

Die Method wird aufgerufen, wenn der Knopf zur Darstellung des Animationsfenster gedrueckt wurde. Es wird eine neue Instanz des Animationsfensters erzeugt und dargestellt.

`private void` **`pbCalcMIP_actionPerformed`**`()`

Die Methode wird aufgerufen, wenn der Knopf zur Berechnung eines MIP-Bildes gedrueckt wurde. Es wird dann zunaechst ein Dialog aufgeblendet, der abfragt, welche Bilder in die MIP-Berechnung einbezogen werden sollen. Nach dessen Bestaetigung wird ein neues MIP-Bild berechnet, in den Bildstapel eingefuegt und dargestellt.

```
public void calcMIPImage(int start,
             int end)
```

Die Methode berechnet ein MIP-Bild aus den angegebenen Bildern. Es wird ein ImagePlus-Objekt erzeugt, dass die gleichen DICOM-Informationen enthaelt, wie die Bilder, aus denen das MIP-Bild erzeugt wurde.

Parameter:

`start` - Das erste Bild, das in die Berechnung einfliessen soll.
`end` - Das letzte Bild, das in die Berechnung einfliessen soll.

`private void` **`pbProjection1_actionPerformed`**`()`

Die Methode wird aufgerufen, wenn der Knopf zur Berechnung der ersten 90-Grad-Projektion gedrueckt wurde. Es wird dann zunaechst ein Dialog aufgeblendet, der abfragt, welche Bilder in die Berechnung einbezogen werden sollen, welche neue Schicht berechnet werden soll und ob die Schicht interpoliert werden soll. Nach dessen Bestaetigung wird ein neues Bild der gewaehlten Projektion berechnet, in den Bildstapel eingefuegt und dargestellt.

```
public void createNewProj1Image(int start,
                int end,
                int slice,
                boolean interpolate)
```

Die Methode berechnet ein neues Bild der ersten 90-Grad-Projektion, fuegt es in den Bildstapel ein, fenstert es optimal und stellt es dar.

Parameter:

`start` - Das erste Bild, das in die Berechnung einfliessen soll.
`end` - Das letzte Bild, das in die Berechnung einfliessen soll.
`slice` - Die neue Schicht, die berechnet werden soll.
`interpolate` - Schalter fuer Interpolation ein/aus.

```
public ImagePlus calcProjection1(int slice,
                 int fromSlice,
                 int toSlice,
                 boolean interpolate)
```

Die Methode berechnet ein neues Bild der ersten 90-Grad-Projektion. Das Bild wird allerdings nicht in den Bildstapel eingefuegt. Die Methode wird zur Berechnung der Bilder fuer die Animation benoetigt.

Parameter:

`start` - Das erste Bild, das in die Berechnung einfliessen soll.
`end` - Das letzte Bild, das in die Berechnung einfliessen soll.
`slice` - Die neue Schicht, die berechnet werden soll.
`interpolate` - Schalter fuer Interpolation ein/aus.

```
private void pbProjection2_actionPerformed()
```

Die Methode wird aufgerufen, wenn der Knopf zur Berechnung der zweiten 90-Grad-Projektion gedrueckt wurde. Es wird dann zunaechst ein Dialog aufgeblendet, der abfragt, welche Bilder in die Berechnung einbezogen werden sollen, welche neue Schicht berechnet werden soll und ob die Schicht interpoliert werden soll. Nach dessen Bestaetigung wird ein neues Bild der gewaehlten Projektion berechnet, in den Bildstapel eingefuegt und dargestellt.

```
public void createNewProj2Image(int start,
                int end,
                int slice,
                boolean interpolate)
```

Die Methode berechnet ein neues Bild der zweiten 90-Grad-Projektion, fuegt es in den Bildstapel ein, fenstert es optimal und stellt es dar.

Parameter:

`start` - Das erste Bild, das in die Berechnung einfliessen soll.
`end` - Das letzte Bild, das in die Berechnung einfliessen soll.
`slice` - Die neue Schicht, die berechnet werden soll.
`interpolate` - Schalter fuer Interpolation ein/aus.

```
public ImagePlus calcProjection2(int slice,
                 int fromSlice,
                 int toSlice,
                 boolean interpolate)
```

Die Methode berechnet ein neues Bild der ersten 90-Grad-Projektion. Das Bild wird allerdings nicht in den Bildstapel eingefuegt. Die Methode wird zur Berechnung der Bilder fuer die Animation benoetigt.

Parameter:

`start` - Das erste Bild, das in die Berechnung einfliessen soll.
`end` - Das letzte Bild, das in die Berechnung einfliessen soll.
`slice` - Die neue Schicht, die berechnet werden soll.
`interpolate` - Schalter fuer Interpolation ein/aus.

```
private void setStdDirectory(JFileChooser fd)
```

Die Methode setzt das aktuelle Verzeichnis auf den zuletzt gewählten Pfad.

Parameter:

`fd` - Dialog, für den der Pfad gesetzt wird.

```
private void loadPreferences()
```

Die Methode liest einie Voreinstellungen aus der Datei preferences.properties ein.

```
private void savePreferences()
```

Die Methode wird beim Beenden der Anwendung aufgerufen und speichert dann einige Voreinstellungen wie z.B. den Dateipfad, aus dem zuletzt ein Rohdatensatz geladen wurde.

Klasse VMRT

public class **VMRT** extends java.lang.Object

Die Klasse ist die Hauptklasse des virtuellen Tomographen. Sie wird nur zum Starten benoetigt, indem eine Instanz von *VMRT* erzeugt wird. In deren Konstruktor wird dann wiederum eine Instanz von *VMRTFrame* erzeugt, wodurch das Hauptfenster des virtuellen Tomographen dargestellt wird.

Konstruktoren

`public VMRT()`

Standardkonstruktor. Es wird eine Instanz von VMRTFrame erzeugt und diser wird dargestellt, indem die pack-Methode aufgerufen wird.

Methoden

`public static void main(java.lang.String[] args)`

Die Methode erzeugt eine neue Instanz der Klasse VMRT und setzt das Look&Feel. Es wird das Systemtypische Look&Feel verwendet.

Parameter:

`args` - Befehlszeilenargumente. Sie werden hier einfach ignoriert.

Klasse VMRTFrame

public class **VMRTFrame** extends JFrame

Diese Klasse stellt die Oberflaeche fuer das Messwerkzeug des virtuellen Tomographen dar. Es werden alle graphischen Elemente erzeugt, eingerichtet und dargestellt. Ausserdem findet hier die Ereignissteuerung der Bedienelmente statt.

Klassen- und Instanzvariablen	
`private ButtonGroup`	`bgNumOfImages` Knopfgruppe zur Gruppierung der Knoepfe rb1Image und rb4Images.
`private ButtonGroup`	`bgPlaceNextImage` Knopfgruppe fuer die beiden Knoepfe rbNextImageAtSelection und rbNextImageAtFreePlace.
`private ArtefactsCombo`	`cbArtefacts` Auswahlbox zur Auswahl eines Artefakt-Simulators.
`private SequenceCombo`	`cbSequence` Auswahlbox, aus der der Benutzer die zu verwendende Pulssequenz auswaehlen kann.
`private int`	`curr_x` Aktuelle x-Mausposition auf der Zeichenflaeche.
`private int`	`curr_y` Aktuelle y-Mausposition auf der Zeichenflaeche.
`private java.awt.Cursor`	`defaultCursor` Standard-Mauszeiger.
`private ImageIcon`	`i1Image` Icon fuer den Knopf, um 1 Bild darzustellen.
`private ImageIcon`	`i4Images` Icon fuer den Knopf, um 4 Bilder darzustellen.
`private ImageIcon`	`iHistogram` Icon fuer den Knopf zur Darstellung des Histogrammfensters.
`private ImageIcon`	`iInfoText` Icon fuer den Knopf zur Anzeige der Bidbeschriftung.
`private ImageIcon`	`iKSpace` Icon fuer den Knopf zur Darstellung des k-Raum-Fensters.
`private ImageIcon`	`iOpen`

	Icon fuer den Knopf zum Laden eines Rohdatensatzes.
private ImageIcon	iPrint Icon fuer den Knopf zum Drucken des Zeichenflaecheninhalts.
private ImageIcon	iSave Icon fuer den Knopf zum Speichern eines Bildes.
private int	lastCenter Fensterungszentrum-Einstellungen beim letzten Mausklick.
private int	lastMouseClickX Letzte Mausklick-x-Koordinate.
private int	lastMouseClickY Letzte Mausklick-y-Koordinate.
private long	lastTime Zeitpunkt des letzten Mausklicks im Hauptfenster.
private int	lastWindow Fensterungsbreite-Einstellungen beim letzten Mausklick.
private JLabel	lNextImageAt Beschriftung fuer die Bedienelemente, die zur Einstellung der Position des naechsten Bildes dienen.
private JLabel	lPosition Textfeld fuer die Positionsangabe des Mauszeigers, wenn dieser sich ueber dem Canvas befindet.
private JLabel	lPositionText Beschriftungsfeld fuer die Mausposition.
private JLabel	lSequence Beschriftung ueber der Sequenzauswahlbox.
private JMenuBar	menuBar1 Die Menueleiste.
private JMenu	menuFile Das Dateimenue.
private JMenuItem	menuFileExit Dateimenueeintrag 'Beenden'.
private JMenuItem	menuFileOpen Dateimenueeintrag 'Oeffnen'.
private JMenuItem	menuFilePrint Dateimenueeintrag 'Drucken'.
private JMenuItem	menuFileSave Dateimenueeintrag 'Speichern'.
private JMenu	menuHelp Das Hilfemenue.
private JMenuItem	menuHelpAbout Hilfemenueeintrag 'Ueber...'.
private FileLoader	myFileLoader Das FileLoader-Objekt laedt einen Rohdatensatz und stellt die Rohdatenmatrizen zur Verfuegung.
private JPanel	pArtefacts Bereich, in dem die Artefakt-Simulatoren ihre Oberflaechenelemente darstellen koennen.
private JButton	pbHistogram Knopf zur Darstellung des Histogramms des selektierten Bildes.
private JButton	pbKSpace Knopf zur Darstellung des k-Raums des selektierten Bildes.
private JButton	pbLoad Knopf zum Laden eines neuen Rohdatensatzes.
private JButton	pbOptWindowing Knopf zum Berechnen der optimalen Fensterung.
private JButton	pbPrint Knop zum Drucken des Zeichenflaecheninhalts.
private JButton	pbReferenceData Knopf zum Laden des Referenzdatensatzes.
private JButton	pbResetWindowing Knopf zum Zuruecksetzen der Fensterung auf C=2048 und W=4096.
private JButton	pbSave Knopf zum Speichern eines Bildes.

MyPanel	pCanvas Zeichenflaeche (Bildflaeche / Canvas).
private SliderPanel	pCenter Schieberegler fuer die Helligkeit (Center).
private JPanel	pNextImageAt Rahmen zur optischen Gruppierung der Elemente, die zur Einstellung der Position des naechsten Bildes dienen.
private JProgressBar	prbarProgress Fortschrittsanzeige.
private java.util.Properties	preferences Hashtabelle für die Standardeinstellungen
private JPanel	pSequence Darstellungsbereich, in dem die Pulssequenzen ihre eigenen Info-GUI-Elemente darstellen koennen.
private JPanel	pSequenceSettings Darstellungsbereich, in dem die Pulssequenzen ihre Einstellungsmoeglichkeiten darstellen koennen.
private SliderPanel	pTimeFactor Schieberegler zur Auswahl des Simulationszeitfaktors (0-100%).
private JPanel	pToolbar Rahmen fuer den Werkzeugbereich am rechten Rand des Fensters.
private SliderPanel	pWindow Schieberegler fuer den Kontrast (Window).
JToggleButton	rb1Image Knopf zur Auswahl der Darstellung von nur einem Bild.
private JToggleButton	rb4Images Knopf zur Auswahl der Darstellung von 4 Bildern gleichzeitig.
private JRadioButton	rbNextImageAtFreePlace Knopf zur Erzeugung des naechsten Bildes an der naechsten freien Position.
JRadioButton	rbNextImageAtSelection Knopf zur Erzeugung des naechsten Bildes an der selektierten Position.
JScrollBar	sbHorScrollbar Horizontale Bildlaufleiste fuer die Zeichenflaeche.
private JLabel	statusBar Statusleiste.
JToggleButton	tbImageText Knopf zur Darstellung der Bildbeschriftungen (Nummern, TR, TI, u.a.).
private JPanel	tpArtefact Karteikarte fuer die Artefaktsimulation.
private JPanel	tpExtras Karteikarte fuer die Zusatzfunktionen (Simulationszeit, u.a.).
private JPanel	tpLayout Karteikarte fuer alle Layout-Funkionen.
private JTabbedPane	tpMain Hauptrahmen fuer die Karteikarten .
private JPanel	tpWindow Karteikarte fuer die Fensterungs-Funktionen (Helligkeit und Kontrast).
private boolean	unsavedChanges Merker fuer nichtgespeicherte Aenderungen.
private java.awt.Cursor	windowCursor Mauszeiger fuer die Fensterung.

Konstruktoren

public **VMRTFrame**()

Standardkonstruktor. Er initialisiert das Fenster, fuellt die Auswahlbox mit den zur Verfuegung stehenden Pulssequenzen und stellt die Bedienelemente der ersten Pulssequenz dar.

Methoden

`private void` **`jbInit`**`()`
`throws java.lang.Exception`

Die Methode richtet die Bedienelemente ein und stellt diese dar.

`private void` **`pCanvas_mouseDragged`**`(java.awt.event.MouseEvent e)`

Die Methode implementiert die Fensterung des selektierten Bildes mittels Mausbewegung. Eine Bewegung der Maus in x-Richtung veraendert das Zentrum, eine Bewegung in y-Richtung die Breite des Fensters.

Parameter:

`e` - Das Ereignis, das beim Bewegen der Maus ausgeloest wird.

`private void` **`helpAbout_actionPerformed`**`()`

Die Methode wird aufgerufen, wenn die Aktion Hilfe | Info aus dem Menue aufgerufen wurde. Das Informationsfenster wird dann aufgeblendet.

`protected void` **`processWindowEvent`**`(java.awt.event.WindowEvent e)`

Die Method wird aufgerufen, wenn der Schliessen-Knopf oben rechts am Fenster betaetigt wird. Die Methode muss ueberschrieben werden, damit das Programm bei einem System-Close beendet werden kann.

Parameter:

`e` - Das Ereignis beim Betaetigen des Schliessen-Knopfes.

`private void` **`updateCW`**`()`

Die Methode aendert die Fensterungseinstellungen des selektierten Bildes gemaess den Einstellungen der Schieberegler.

`private void` **`updateCenter`**`()`

Die Methode aktualisiert die Center-Einstellung in der selektierten ImagePlus-Klasse.

`private void` **`updateWindow`**`()`

Die Methode aktualisiert die Window-Einstellung in der selektierten imagePlus-Klasse.

`private void` **`setStdDirectory`**`(JFileChooser fd)`

Die Methode setzt das aktuelle Verzeichnis auf den zuletzt gewählten Pfad.

Parameter:

`fd` - Dialog, für den der Pfad gesetzt wird.

`private void` **`pbLoad_actionPerformed`**`()`

Die Methode wird aufgerufen, wenn der 'Oeffnen'-Knopf gedrueckt wurde oder die Aktion Datei | Oeffnen aufgerufen wurde. Ein Dateidialogfenster wird aufgeblendet, wo der Benutzer die zu oeffnende Datei auswaehlen kann. Der ausgewaehlte Dateipfad wird zurueckgeliefert und an ein FileLoader-Objekt uebergeben, das dann die Rohdatenmatrizen laedt.

`private void` **`pbSave_actionPerformed`**`()`

Die Methode wird aufgerufen, wenn der 'Speichern'-Knopf gedrueckt wurde oder die Aktion Datei | Speichern aufgerufen wurde. Ein Dateidialogfenster wird aufgeblendet, wo der Benutzer den Pfad und den Namen der zu speichernden Datei angeben kann. Mit Hilfe des zurueckgelieferten Dateipfades und dem selektierten Bild wird ein DcmSecondCapt-Objekt erzeugt und dessen write-Methode aufgerufen. Dadurch wird das selektierte Bild im DICOM-Format gespeichert.

`private void` **`pbPrint_actionPerformed`**`()`

Die Methode wird aufgerufen, wenn der 'Drucken'-Knopf gedrueckt wurde oder die Aktion Datei | Drucken aufgerufen wurde. Der systemabhaengige Druckdialog wird aufgeblendet. Nach dessen Bestaetigung wird der Inhalt der Zeichenflaeche auf dem ausgewaehlten Drucker ausgegeben.

`private void` **`pbReferenceData_actionPerformed`**`()`

Die Methode wird aufgerufen, wenn der 'Referenzbild'-Knopf gedrueckt wurde. Es wird ein Refenezdatensatz geladen, der die Parameterdaten von 6 verschiedenen Geweben enthaelt.

`private void` **`pCanvas_mouseMoved`**`(java.awt.event.MouseEvent e)`

Die Methode bestimmt waehrend der Bewegung des Mauszeigers ueber der Zeichenflaeche dessen Poition und gibt sie in einem Ausgabefeld aus. Dabei muss zwischen der Darstellung von 1 und 4 Bildern unterschieden werden.

Parameter:

e - Das Ereignis beim Bewegen der Maus ueber die Zeichenflaeche.

`private void` **`pCanvas_mouseExited`**`(java.awt.event.MouseEvent e)`

Die Methode wird aufgerufen, wenn der Mauszeiger die Zeichenflaeche verlaesst. Die Positionsanzeige wird dann geloescht.

Parameter:

e - Das Ereignis beim Verlassen der Zeichenflaeche mit der Maus.

`private void` **`rb4Images_actionPerformed`**`()`

Die Methode wird aufgerufen, wenn der Knopf zur gleichzeitigen Anzeige von 4 Bildern gedrueckt wurde. Der Inhalt der Zeichenflaeche wird dann neu aufgebaut.

`private void` **`rb1Image_actionPerformed`**`()`

Die Methode wird aufgerufen, wenn der Knopf zur Anzeige von nur einem Bild gedrueckt wurde. Der Inhalt der Zeichenflaeche wird dann neu aufgebaut.

`private void` **`pCanvas_mousePressed`**`(java.awt.event.MouseEvent e)`

Die Methode wird aufgerufen, sobald ein Mausknopf auf der Zeichenflaeche gedrueckt wird. Es werden dann die Mauskoordinaten und die aktuellen Fensterungswerte gespeichert. Dies wird benoetigt, um die Fensterung per Mausbewegung durchfuehren zu koennen.

Parameter:

e - Das Ereignis, das beim Druecken eines Mausknopfes ausgeloest wird.

`private void` **`pCanvas_mouseReleased`**`(java.awt.event.MouseEvent e)`

Die Methode wird aufgerufen, sobald eine Maustaste ueber der Zeichenflaeche wieder losgelassen wird. Der Mauszeiger wird dann auf den Standardwert zurueckgesetzt.

Parameter:

e - Das Ereignis, das beim Loslassen einer Maustaste ausgeloest wird.

`private void` **`pCanvas_mouseClicked`**`(java.awt.event.MouseEvent e)`

Die Methode wird aufgerufen, wenn auf der Zeichenflaeche ein Mausklick durchgefuehrt wird. Es werden dann die Fenstereinstellungen des bisher selektierten Bildes gesichert. Beim neu selektierten Bild werden die alten Fenstereinstellungen ausgelesen und wieder dargestellt. Anschliessend wird der Inhalt der Zeichenflaeche aktualisiert.

Parameter:

e - Das Ereignis beim Druecken einer Maustaste auf der Zeichenflaeche.

`private void` **`tbImageText_mouseClicked`**`()`

Die Methode wird aufgerufen, wenn der Knopf zum Einzeichnen der Bildbeschriftung gedrueckt wird. Die Beschriftung wird dann ein- oder ausgeschaltet und der Inhalt der Zeichenflaeche wird neu aufgebaut.

`private void` **`fileOpen_actionPerformed`**`()`

Die Methode wird aufgerufen, wenn die Aktion Datei | Oeffnen aus dem Menue aufgerufen wird. Diese Methode wird nur auf die entsprechende Methode des Oeffnen-Knopfes in der Werkzeugleiste umgeleitet.

`private void` **`fileSave_actionPerformed`**`()`

Die Methode wird aufgerufen, wenn die Aktion Datei | Speichern aus dem Menue aufgerufen wird. Diese Methode wird nur auf die entsprechende Methode des Speichern-Knopfes in der Werkzeugleiste umgeleitet.

`private void` **`filePrint_actionPerformed`**`()`

Die Methode wird aufgerufen, wenn die Aktion Datei | Drucken aus dem Menue aufgerufen wird. Diese Methode wird nur auf die entsprechende Methode des Drucken-Knopfes in der Werkzeugleiste umgeleitet.

`public void` **`fileExit_actionPerformed`**`()`

Die Methode wird aufgerufen, wenn die Aktion Datei | Beenden aus dem Menue aufgerufen wird. Die Anwendung wird beendet.

`private void` **`pbOptWindowing_actionPerformed`**`()`

Die Methode wird aufgerufen, wenn der Knopf zum optimalen Fenstern des selektierten Bildes gedrueckt wurde. Es wird dann die entsprechende Methode fuer das selektierte Bild aufgerufen und der Zeichenflaecheninhalt neu dargestellt. Ausserdem werden die Einstellungen der Schieberegler fuer die Fensterung angepasst.

`private void` **`pbHistogram_actionPerformed`**`()`

Die Methode wird aufgerufen, wenn der Knopf zur Darstellung des Histogramms zum selektierten Bild gedrueckt wurde. Es wird dann ein Fenster aufgeblendet, in dem das Histogramm des selektierten Bildes zu sehen ist.

`private void` **`pbKSpace_actionPerformed`**`()`

Die Methode wird aufgerufen, wenn der Knopf zur Darstellung des K-Raums des selektierten Bild gedrueckt wurde. Es wird dann ein Fenster aufgeblendet, in dem der k-Raum (Magnitudenbild) des selektierten Bildes zu sehen ist. Ausserdem kann man sich das Phasenbild sowie den Real- und Imaginaerteil anzeigen lassen.

`private void` **`sbHorScrollbar_adjustmentValueChanged`**`()`

Die Methode wird aufgerufen, wenn sich der Zustand des Scrollbalkens veraendert hat. Der Scrollbalken wird dann aktualisiert.

`private int` **`getImageMouseIsOn`**`(int x,`
`int y)`

Die Methode bestimmt aufgrund der x- und y-Koordinate des Mauszeigers auf der Zeichenflaeche, ueber welchem Bild sich der Mauszeiger gerade befindet. Dabei geht die Einstellung 1 Bild / 4 Bilder und die Bildlaufleistenposition mit in die Berechnung ein.

Parameter:

`x` - x-Koordinate des Mauszeigers.
`y` - y-Koordinate des Mauszeigers.

Rückgabewert:

Nummer des Bildes, ueber dem sich der Mauszeiger befindet.

`public void` **`createIntensityImage`**`(Pulsesequence sequence)`

Die Methode erzeugt ein Intensitaetsbild aus den Rohdaten mittels der ausgewaehlten Pulssequenz. Dazu werden die Rohdaten zunaechst aus dem FileLoader-Objekt ausgelesen. Die Rohdaten werden dann an die Sequenz uebergeben und diese wird gestartet. Die Sequenz meldet, wenn sie mit der Berechnung fertig ist. Sie ruft dann die Methode drawCreatedIntensityImage auf. Die Methode wird angestossen, wenn der 'Berechnen'-Knopf der Pulssequenz betaetigt wurde.

Parameter:

`sequence` - Die Pulssequenz, die zur Erzeugung des Intensitaetsbildes verwendet werden soll.

`public ImagePlus` **`drawCreatedIntensityImage`**`(Pulsesequence sequence)`

Die Methode stellt ein von einer Pulssequenz erzeugtes Intensitaetsbild auf der Zeichenflaeche dar. Dazu wird das Intensitaetsbild zunaechst aus der Pulssequnz ausgelesen und dann in ein DcmImage-Objekt umgewandet. Damit enthaelt das Bild die geleichen zusaetzlichen Bildinformationen wie die Rohdatenmatrizen. Einige Werte, wie z.B. die Fensterungswerte werden natuerlich entsprechend neu gesetzt. Spaeter ist es somit moeglich, das Bild im DICOM-Format abzuspeichern. Aber zunaecht wird das DcmImage-Objekt erstmal in einem ImagePlus-Objekt gespeichert.

Parameter:

`sequence` - Die Pulssequenz, aus der die Intensitaetsmatrix ausgelesen werden soll.

`public JProgressBar` **`getProgressBar`**`()`

Die Methode liefert eine Referenz auf die Fortschrittsanzeige zurueck.

Rückgabewert:

Eine Referenz auf die Fortschrittsanzeige.

`public int` **`getTimeFactor`**`()`

Die Methode liefert den eingestellten Simulationszeitfaktor zurueck.

Rückgabewert:

Der vom Benutzer eingestellte Simulationszeitfaktor.

`public ImagePlus` **`getSelectedImage`**`()`

Die Methode liefert das ImagePlus-Objekt des zur Zeit selektierten Bildes zurueck.

Rückgabewert:

Das ImagePlus-Objekt des selektierten Bildes.

`public FileLoader` **`getFileLoader`**`()`

Die Methode liefert das FileLoader-Objekt zurueck, in dem die Rohdatenmatrizen und einige zusaetzliche Informationen stehen (wie z.B. Pixelabstand).

Rückgabewert:

Das aktuelle FileLoader-Objekt.

`public void` **`setStatusBar`**`(java.lang.String txt)`

Die Methode setzt den Text der Statusleiste auf den uebergebenen Text.

Parameter:

`txt` - Der Text, der in der Statusleiste angezeigt werden soll.

`public java.lang.String` **`getStatusText`**`()`

Die Methode liefert den Text der Statusleiste zurueck.

Rückgabewert:

Der Text der Statusleiste.

`public void` **`setProgressBar`**`(int val)`

Die Methode setzt den Wert der Fortschrittsanzeige auf den uebergebenen Wert.

Parameter:

`val` - Der Wert, auf den die Fortschrittsanzeige gesetzt werden soll.

`public ArtefactUI` **`getSelectedArtefactUI`**`()`

Die Methode liefert eine Referenz auf die Oberflaechenklasse des aktuell ausgewaehlten Artefakt-Simulators zurueck.

Rückgabewert:

Die Oberflaechenklasse des selektierten Artefakt-Simulators.

`public void` **`loadPreferences`**`()`

Die Methode liest einige Voreinstellungen aus der Datei preferences.properties ein.

`public void savePreferences()`

Die Methode wird beim Beenden der Anwendung aufgerufen und speichert dann einige Voreinstellungen wie z.B. den Dateipfad, aus dem zuletzt ein Rohdatensatz geladen wurde.

Anhang B: Literaturverzeichnis

[BORN96] Born, G.: „Referenzhandbuch Dateiformate – Grafik, Text, Datenbanken, Tabellenkalkulation“, 4. Auflage, Addison-Wesley, Bonn, 1996

[BRON95] Bronstein, I.N / Semendjajew, K.A. / Musiol, G. / Mühling, H.: „Taschenbuch der Mathematik, Verlag Harri Deutsch, Thun und Frankfurt am Main, 1994

[CONT72] Conte / de Boor: „Elementary Numerical Analysis: An Algorithmic Approach“, McGraw-Hill Book Co., New York 1972

[CORM94] Cormen, T. / Leiserson, C. / Rivest, R.: „Introduction to Algorithms“, 13. Auflage, MIT Press, Massachusetts, 1994

[DICOM98] National Electrical Manufacturers Association: „Digital Imaging and Communications in Medizine (DICOM)“, Part 1-14, National Electrical Manufactures Association, Rosslyn (Virginia), 1998

[GEAR99] Geary, D.M.: „Graphic Java 2 – Mastering the JFC“, Volume II, Sun Microsystems Press, 1999

[GENE83] Golub, G.H. / van Loan, C.F.: „Matrix computations“, North Oxford Academy, Oxford, 1983

[GONZ92] Gonzales, R.C. / Woods, R.E.: „Digital Image Processing“, Addison-Wesley, 1992

[GOSL96] Gosling, J. / Joy, B. / Steele, G.: „The Java Language Specification“, Addison-Wesley, 1996

[GOSL97a] Gosling, J. / Yellin, F.: „Das Java API. Band 1: Die Basispakete“, 1. Auflage, Addison-Wesley, Bonn, 1997

[GOSL97b] Gosling, J. / Yellin, F.: „Das Java API. Band 2: Das Window Toolkit und Applets“, 1. Auflage, Addison-Wesley, Bonn, 1997

[HABE95] Haberäcker, P.: „Praxis der digitalen Bildverarbeitung und Mustererkennung“, Hanser-Verlag, München, Wien, 1995

[HACK97] Hackländer, T.: „Kernspintomographische Messung desregionalen cerebralen Blutvolumens nach der Relaxationsmethode“, Habilitationsschrift der Medizinischen Fakultät der Heinrich-Heine-Universität Düsseldorf, 1996.

[HASH97] Hashemi, R.H. / Bradley, W.G.: „MRI – The Basics“, Williams & Wilkins, Baltimore, 1997

[HINS83] Hinshaw / Lent: „An Introduction to NMR Imaging – From the Bloch Equation to the Imaging Equation“, Proc IEEE 71 (Seite 338-350), 1983

[HORN98] Horn, C. / Kerner, I.O.: „Lehr und Übungsbuch Informatik – Band 4: Technische Informatik und Systemgestaltung“, Kapitel „Entwicklung grafischer Benutzeroberflächen“, Seite 239-274, Fachbuchverlag Leipzig im Carl Hanser Verlag, München, 1998

[KLET95] Klette, R. / Zamperoni, P.: „Handbuch der Operatoren für die Bildverarbeitung“, 2. Auflage, Vieweg Verlag, 1995

[KRES88] Krestel, E.: „Bildgebende Systeme für die medizinische Diagnostik“, 2. Auflage, Siemens AG, Hrsg. Heinz Marneburg, 1988

[KÜHN96] Kühnel, R.: „Die Java-Fibel: Programmieren interaktiver Homepages für das World Wide Web“, 1. Auflage, Addison-Wesley, Bonn, 1996

[LAUB90] Laubenberger, T.: „Technik der medizinischen Radiologie“, 5. Überarbeitete Auflage, Dt. Ärzte-Verlag GmbH, Köln, 1990

[LEHM97] Lehmann, T. / Oberschlep, W. / Pelikan, E. / Repges, R.: „Bildverarbeitung für die Medizin“, Springer, Berlin, 1997

[LEMA98] Lemay, L. / Cadenhead, R.: „Java 1.2 programmieren in 21 Tagen“, Markt&Technik Buch- und Software-Verlag, Haar, 1998

[LISS90] Lissner / Seiderer: „Klinische Kernspintomographie“, Enke, Stuttgart, 2. Auflage, 1990

[LYON99] Lyon, D.A.: „Image processing in Java“, Prentice Hall, 1999

[MEZR95] Mezrich, R.: „A Perspective on k-space“, Radiology, Seite 297-315, Mai 1995

[MORN95] Morneburg, H.: „Bildgebende Systeme für die medizinische Diagnostik“, 3. Auflage, Siemens AG, 1995

[NICO97] Nico, N. / Albrecht, R.: „Wissenschaftliche Arbeiten schreiben mit Winword 97“, Addison Wesley, Bonn, 1997

[OEST98] Oestereich, B.: „Objektorientierte Softwareentwicklung – Analyse und Design mit der Unified Modeling Language“, 4., aktualisierte Auflage, Oldenbourg Verlag, München, 1998

[PG305] Projektgruppe 305: „Anomalia-Endbericht – Wissensgesteuertes Bildverarbeitungssystem zur Erkennung pathologischer Strukturen in medizinischen Bilddaten“, Hrsg. Reusch, B. / Fathi, M. / Hiltner, J., Lehrstuhl Informatik 1, Universität Dortmund, 1998

[PHIL95] Philips: "Basic Principles of MR Imaging", Niederlande, 1995

[PRES86] Press, W.H. / Flannery, B.P. / Teukolsky, S.A. / Vetterling, W.T.: „Numerical Recipes – The art of scientific computing“, Cambridge University Press, 1986

[PSCH94] Pschyrembel: „Klinisches Wörterbuch“, 257. Auflage, de Gruyter Verlag, 1994

[PYKE82] Pykett, I.L.: „Kernspintomographie – Röntgenbilder ohne Röntgenstrahlen“, Spektrum der Wissenschaft, Seite 40-54, Juli 1982

[REIC97] Reichenbach, J.R. / Hackländer, Th. / Harth, T. / Hofer, M. / Rasser, M. / Mödder, U.: „^{1}H T_1 and T_2 measurements of the MR imaging contrast agents Gd-DTPA and Gd-DTPA BMA at 1.5 T“ aus: „European Radiology“, S.264-274, Juli 1997

[REIS97] Reiser, M. / Semmler, W.: „Magnetresonanztomographie“, 2. Auflage, Springer Verlag, 1997

[REVE97] Revet, B.: „DICOM Cook Book for Implementations in Modalities“, Philips Medical Systems, Niederlande, 1997

[SEDG92] Sedgewick, R. „Algorithmen in C“, Addison-Wesley, 1992

[SIEM92] Siemens AG: „Magnet, Spins und Resonanzen – Eine Einführung in die Grundlagen der Kernspintomographie“, Erlagen, 1992

[SIEM97] Siemens AG: „Bildgebende Sequenzen in der Kernspintomographie und ihre klinische Anwendung“, Sonderdruck aus electromedia 64/65, 1997

[STAR70] Starmer, F. / Clark, D.O.: „Computer computations of cardiac output using the gamma function“ aus: „Applied Physiology“, 1970

[SUN98a] Sun Microsystems: „Java 3D API Specification“, Palo Alto (USA), 1998

[SUN98b] Sun Microsystems: „Programmer's Guide to the Java 2D API“, Mountain View (USA), September 1998

[SUN99] Sun Microsystems: „Java Look and Feel Design Guidelines“, Sun Microsystems, Palo Alto (Kalifornien), 1999

[WAHL89] Wahl, F.M.: „Digitale Bildsignalverarbeitung“, berichtigter Nachdruck, Springer-Verlag, 1989

[WALS98] Walsch, A. / Fronkowiak, J.: „Die Java Bibel“, 1. Auflage, MITP-Verlag, Bonn, 1998

[WILH99] Wilhelms, G. / Kopp, M.: Java Professionell, MITP-Verlag, Bonn, 1999

[WOOD92] Wood: „Fourier Imaging“ in Stark / Bradley: „Magnetic Resonance Imaging“, Mosby Year Book, 2. Auflage, St. Louise, 1992

Anhang C: Abbildungs- und Tabellenverzeichnis

Abbildung 1: Röntgenaufnahme eines Schädels ... 7
Abbildung 2: CT-Aufnahme eines Schädels ... 7
Abbildung 3: Ultraschallaufnahme einer Harnblase ... 7
Abbildung 4: MRT-Aufnahme eines Schädels ... 7
Abbildung 5: Schematische Röntgenröhre ... 8
Abbildung 6: Prinzip des Röntgens ... 8
Abbildung 7: Röntgentisch ... 9
Abbildung 8: Computertomograph ... 9
Abbildung 9: Schichteselektion ... 10
Abbildung 10: Historische Computertomographie ... 10
Abbildung 11: Moderne Computertomographie ... 10
Abbildung 12: Fenstern ... 11
Abbildung 13: Die verschiedenen Darstellungsmodi beim Ultraschallverfahren ... 12
Abbildung 14: Kernspin und Präzession ... 12
Abbildung 15: Ausrichtung der Kernspins im Magnetfeld ... 12
Abbildung 16: Blockschaltbild eines MRT ... 13
Abbildung 17: MRT-Gerät ... 13
Abbildung 18: Magnetische Dipolfelder ... 14
Abbildung 19: Einstellung der Spins im feldfreien Raum und im Magnetfeld B_0 ... 15
Abbildung 20: Magnetisierungsverlauf einer Probe in einem externen Magnetfeld ... 16
Abbildung 21: Parallele und antiparallele Anordnung der Spins mit unterschiedlichen Energieniveaus ... 18
Abbildung 22: Kernspin und Präzession ... 18
Abbildung 23: Phasenkohärenz der Spins nach Einstrahlung von HF-Strahlung (unten) ... 19
Abbildung 24: Auslenkung des Gesamtmagnetisierungsvektors durch einen 90°-Impuls ... 20
Abbildung 25: T_1-Relaxation nach einem 90°-Impuls ... 21
Abbildung 26: T_2-Zerfall nach einem 90°-Impuls ... 21
Abbildung 27: Verlauf der longitudinalen Relaxation bei verschiedenen Geweben ... 22
Abbildung 28: Verlauf der transversalen Relaxation bei verschiedenen Geweben ... 22
Abbildung 29: T_1-Relaxation nach einem 180°-Impuls ... 23
Abbildung 30: Pulsdiagramm einer FID-Sequenz ... 24
Abbildung 31: Pulsdiagramm einer Spin-Echo-Sequenz ... 24
Abbildung 32: Sequenzbaum ... 24
Abbildung 33: Saturation-Recovery-Pulssequenz (SR) ... 26
Abbildung 34: Signalintensität beim FID ... 26
Abbildung 35: Signalintensität der Inversion-Recovery-Sequenz (IR) ... 28
Abbildung 36: Signalintensität mit Absolutwerten und Betrag bei der Inversion-Recovery-Sequenz ... 28
Abbildung 37: Spinrephasierung ... 29
Abbildung 38: Die Spin-Echo-Sequenz ... 29
Abbildung 39: SE-Signalintensität in Abhängigkeit von Aufnahme- und Gewebeparametern ... 31
Abbildung 40: Abhängigkeit der Bilderzeugung von den Geräteparameter bei einer Spin-Echo-Sequenz ... 31
Abbildung 41: T_1-Relaxationskurven einiger Substanzen ... 32
Abbildung 42: T_1-Kurve verschiedener Gewebe im Gehirn ... 32
Abbildung 43: Relaxationskurven für Liquor und weiße Gehirnmasse ... 33
Abbildung 44: Signalintensität für Liquor und weiße Gehirnmasse mit verschiedenen Echozeiten ... 33
Abbildung 45: Relaxation nach 90°-Impuls mit sehr kurzer Relaxationszeit ... 34
Abbildung 46: Abhängigkeit der Signalintensität vom Auslenkwinkel für sehr kurze Repititionszeiten ($T_R \rightarrow 0$) ... 35
Abbildung 47: Gradientenschaltung für die Gradientenechosequenz ... 36
Abbildung 48: Maximale Signalintensität bei gespoilter Gradientenechotechnik ... 38
Abbildung 49: Kontrast zwischen weißer Hirnsubstanz und Liquor cerebrospinalis ... 39
Abbildung 50: Einstellung der Gleichgewichts-magnetisierung bei Kleinwinkelanregung ... 40
Abbildung 51: Gleichgewichtsmagnetisierung in Abhän-gigkeit von Auslenkwinkel und Repititionszeit ... 40
Abbildung 52: Abhängigkeit der Signalintensität einer gespoilten GE-Sequenz von Auslenkwinkel und Repititionszeit ... 40
Abbildung 53: Überlagerung eines Gradientenfeldes über das externe Magnetfeld ... 43
Abbildung 54: Ortskodierungstechniken: Punkt-, Linien-, Schicht- und Volumentechnik ... 43
Abbildung 55: Kosinus-Signal (links) und dessen Fouriertransformation (rechts) ... 44
Abbildung 56: Sinc-Funktion (links) und deren Fouriertransformation (rechts) ... 44
Abbildung 57: Schichtselektion durch das gleichzeitige Schalten von 1, 2 oder 3 Gradienten ... 45

Abbildung 58: Schematische Darstellung der Linientechnik 46
Abbildung 59: Frequenzkodierung 47
Abbildung 60: Projektions Rekonstruktion 48
Abbildung 61: Zeitliche Abfolge der HF-Impulse und Gradientenschaltungen am Beispiel der Spin-Echo-Sequenz 49
Abbildung 62:Bildrekonstruktion mit der 2-dim. Fouriertransformation 50
Abbildung 63: Mehrfache Anregung einiger Schichten bei der Vielschichttechnik 51
Abbildung 64: Kerninduktionssignal nach einem 90°-Impuls 52
Abbildung 65: Absorptionslinie: Der Realteil der Fouriertransformation des Zeitsignals 52
Abbildung 66: Eine Linse mit Fouriertransformationsebene 53
Abbildung 67: Ersetzung einer Linse durch eine Menge von Diffraktionsgittern 53
Abbildung 68: Die Fouriertransformationsebene (k-Raum) 54
Abbildung 69: k-Raum in analoger Form 55
*Abbildung 70: k-Raum mit 256*256 Punkten* 56
Abbildung 71: k-Raum mit 1/32 seiner ursprünglichen Größe 56
Abbildung 72: Füllen des k-Raums 57
Abbildung 73: Füllen des k-Raums mit der Spin-Echo-Technik 58
Abbildung 74: Entstehung der Kopien höherer Ordnung 59
Abbildung 75: Sichtbarer Bildausschnitt 59
Abbildung 76: Pulsfolge und Signalverlauf einer schnellen Spin-Echo-Technik 61
Abbildung 77: 1. Verbesserung der Spin-Echo-Technik durch Füllen des k-Raums vom Zentrum aus 63
Abbildung 78: Symmetrisches Füllen des k-Raums 63
Abbildung 79: Änderung der magnetischen Induktion an Gewebegrenzflächen unterschiedlicher Suszeptibilität 65
Abbildung 80: Einfaltungen entlang der Frequenzkodierrichtung 65
Abbildung 81: Reales MRT-System 67
Abbildung 82: MRT-Meßsystem 68
Abbildung 83: MRT-Arbeitsplatz 68
Abbildung 84: Hauptfenster der NUMARIS-Oberfläche 68
Abbildung 85: Dialogfenster der NUMARIS-Oberfläche 68
Abbildung 86: Allgemeiner Fensteraufbau 72
Abbildung 87: Spezieller Fensteraufbau 72
Abbildung 88: Schematische Darstellung des Werkzeuges für Parameterbilder 75
Abbildung 89: Beispiel einer Anwendung des kleinsten-Quadrate-Verfahrens zur Berechnung einer Ausgleichsgeraden ..76
Abbildung 90: Ablaufdiagramm des realen Tomographen 79
Abbildung 91: Funktion des virtuellen MRT 80
Abbildung 92: Berechnung des Graustufenbildes aus den Daten im k-Raum 81
Abbildung 93: Integration des DICOM-Viewers in das Gesamtkonzept 82
Abbildung 94: Fenstern eines 12-Bit-Bildes auf 8-Bit 82
Abbildung 95: 90°-Projektion bei sagitaler Rohdatenorientierung 83
Abbildung 96: Konzept der k-Raum-Manipulation 83
Abbildung 97: Gesamtsystem 84
Abbildung 98: Zuordnung der JAVA-Pakete zu den einzelnen Schritten der Simulation 93
Abbildung 99: Rücktransformation: Zuordnung der JAVA-Pakete zu den einzelnen Schritten der Simulation 94
Abbildung 100: Klassendiagramm für das Paket artefacts 95
Abbildung 101: Implizite und explizite Transfersyntax 97
Abbildung 102: Übertragungsreihenfolge für Zahlenformate. Beispiel: 32-Bit Zahl 97
Abbildung 103: Übersicht über die wichtigsten Klassen des DICOM-Paketes 98
Abbildung 104: Klassendiagramm für das Paket sequences 100
Abbildung 105: Eingabe von Parameterwerten für eine Pulssequenz 101
Abbildung 106: Ein Bedienelement vom Typ SliderPanel zur Einstellung der Repitionszeit 102
Abbildung 107: Ein Bedienelement vom Typ LabelTFLabelPanel 102
Abbildung 108: Klassendiagramm des Paketes vmrt 103
Abbildung 109: Klassendiagramm für das Paket dicomviewer 107
Abbildung 110: Klassendiagramm der Pulssequenz-Klassen 109
Abbildung 111: Allgemeiner Rahmen für die GUI-Elemente einer Pulssequenz 110
Abbildung 112: Rahmen für die GUI-Elemente der SR-Sequenz 111
Abbildung 113: Klassendiagramm der Artefakt-Klassen 114
Abbildung 114: Artefakt-Karteikarte des Virtual MRT 115
Abbildung 115: MRI IMAGE EXPERT 1.0 119
Abbildung 116: Rohdatensätze 120
Abbildung 117: Pulssequenzen 120
Abbildung 118: Sequenzparameterdialog 120
Abbildung 119: Referenzdatensatz. Relaxationszeiten in ms, Protonendichte in Prozent. 121
Abbildung 120: Kontraststudie für Wasser 122

Abbildung 121: Simulation des Bildrauschens ... *123*
Abbildung 122: Simulation der Turbo-Spin-Echo-Sequenz. ... *124*
Abbildung 123: Simulation eines Bewegungsartefakts ... *125*
Abbildung 124: Hauptfenster des Simulationswerkzeugs Virtual MRT nach dem Programmstart ... *128*
Abbildung 125: Dateiauswahldialog des Virtual MRT (hier für MS-Windows) ... *129*
Abbildung 126: MS WINDOWS Druckdialog ... *130*
Abbildung 127: Die Karteikarte ‚Ansicht' ... *130*
Abbildung 128: Darstellung von vier Bildern im Virtual MRT ... *131*
Abbildung 129: Histogrammfenster des Virtual MRT ... *132*
Abbildung 130: Einblendung der Bildinformationen. ... *132*
Abbildung 131: Karteikarte ‚Fenstern' des Virtual MRT ... *133*
Abbildung 132: Die Karteikarte ‚Artefakte' ... *134*
Abbildung 133: Die Karteikarte ‚Extras' ... *134*
Abbildung 134: Dialog für die Sequenzparameter ... *135*
Abbildung 135: Hauptfenster des DICOM-Viewers nach dem Programmstart ... *137*
Abbildung 136: Dateiauswahldialog des DICOM-Viewers ... *138*
Abbildung 137: MS Windows Druckdialog ... *139*
Abbildung 138: Karteikarte ‚Ansicht' des DICOM-Viewers ... *140*
Abbildung 139: DICOM-Viewer nachdem eine Bildserie geladen wurde und die Beschriftung eingeschaltet wurde ... *141*
Abbildung 140: Histogrammfenster des DICOM-Viewers ... *141*
Abbildung 141: k-Raum-Anzeigefenster des DICOM-Viewers ... *142*
Abbildung 142: k-Raum-Manipulator des DICOM-Viewers ... *143*
Abbildung 143: Karteikarte ‚Tools' des DICOM-Viewers ... *144*
Abbildung 144: Karteikarte ‚Fenstern' des DICOM-Viewers ... *146*
Abbildung 145: Karteikarte ‚3D' des DICOM-Viewers ... *147*
Abbildung 146: Parameterdialog zur Berechnung eines MIP-Bildes ... *147*
Abbildung 147: Parameterdialog zur Berechnung einer 90°-Projektion ... *148*
Abbildung 148: Animationsfenster des DICOM-Viewers ... *149*
Abbildung 149: Karteikarte ‚Extras' des DICOM-Viewers ... *150*
Abbildung 150: Das Hauptfenster von IMAGEJ ... *151*
Abbildung 151: Der Importdialog für DICOM-Dateien ... *152*
Abbildung 152: Der Dateilauswahldialog ... *152*
Abbildung 153: Auswahl der Bilder einer Serie zur Parameterbildberechnung ... *153*
Abbildung 154: Abspeichern der berechneten Parameterbilder ... *154*
Abbildung 155: Nachbearbeiten der Parameterbilder ... *155*

Tabelle 1: Darstellungsmöglichkeiten der 4 wichtigsten bildgebenden Verfahren in der Medizin ... *8*
Tabelle 2: Spektrum elektromagnetischer Wellen ... *19*
Tabelle 3: T_1- und T_2-Konstanten (in ms) bei 1 Tesla sowie Protonendichte ... *22*
Tabelle 4: T_1, T_2 und Protonendichte von Gehirngeweben ... *32*
Tabelle 5: Gradientenverfahren. Akronyme einiger Gerätehersteller ... *37*
Tabelle 6: Repititionszeitbedarf in ms für eine 95%ige Sättigung der z-Magnetisierung ... *39*
Tabelle 7: Repititionszeitbereich in ms mit Signalvorteil für eine gespoilte Gradientenechotechnik gegenüber Spin-Echo 41
Tabelle 8: S/R-Verhältnis und Akquisitionszeit für die verschiedenen Bilderzeugungstechniken ... *43*
Tabelle 9: Implementierte Klassen ... *158*

Anhang E: Index

1

180°-Impuls 20, 23

9

90° Impuls 20
90°-Impuls 57

A

A-Bildverfahren 11
Absorptionslinie 52
Akquisitionszeit 43, 50, 56, 58, 63
akustische Grenzfläche 11
Anode 8
Artefakt 63
Atomkern 12, 14
Auflösung 56
Ausbreitungsgeschwindigkeit 11
Auslenkwinkel 20
Auslesegradient 13, 46

B

Bandbreite 44
B-Bildverfahren 11
Beugung 52
Bewegungsartefakt 57
Bildausschnitt 59
Bildkontrast 54
Bildqualität 56
Bildwiederholfrequenz 11
Boltzmann-Gleichung 16
BOLTZMANN-Verteilung 12
Breite 11
Brownsche Bewegungstheorie 61

C

Center 10
Computertomographie 8, 9
CT-Aufnahme 7

D

Dephasierung 22
Detektor 8
diamagnetisch 17
Dichtewert 10
Diffraktionsgitter 52

E

Echoeffekt 11
Eigendrehimpuls 12
Eigenrotation 14
Einfaltung 59
elektrisches Feld 19
elektromagnetische Wellen 19
Elektronenhülle 14
Elementarquader 9
Energiedifferenz 16
Energieniveau 15
exponentielle Wachstumskurve 16

F

Fenstern 10
ferromagnetisch 17
Field of View 59
Filterung 53
Flipwinkel 20
Fluoreszenz 8
Flußdichte 12
Fourieranalyse 43, 46
Fouriertransformation 9, 43, 52, 53
FOV 59
free induction decay 22
freier Induktionszerfall 22
Frequenz 52
Frequenzbereich 43
Frequenzkodiergradient 46, 48, 57
Frequenzspektrum 43

G

Gandolinium 17
Geisterbilder 63
Glühkathode 8
Gradientenfeld 42, 44
Gradienten-Magnete 13
Grauwert 10
Grobstrukturen 54
gyromagnetisches Verhältnis 15

H

Halb-Echo-Technik 60
Halb-Fouriertechnik 60
Helligkeit 11
HF-Puls 13

I

Imaginärbild 55
Intensitätsprofil 9
Intensitätsunterschied 63
inverse Fouriertransformation 43

K

kantenbetontes Bild 62
Kernresonanzlinie 52
Kernspin 14
Kernspinanregung 18
Kernspinresonanz 18
Kernspin-Resonanz-Signal 13
Kernspintomographie 12
Kleinwinkelanregungen 21
Kontrast 11, 56
Kontrastmittel 17
Konvertierung 53
Kopien höherer Ordnung 58
Kosinus-Funktion 44
k-Raum 52

L

Lamor-Beziehung ... 17
Lamorfrequenz ... 12, 18
Linienabstand ... 59
Linientechnik ... 45
Linsensystem ... 52
longitudinale Relaxationszeit ... 22
Longitudinale Relaxationszeit ... 21

M

Magnetfeld ... 15
Magnetfeldinhomogenitäten ... 22
magnetische Flußdichte ... 12, 15
magnetischen Flußdichte ... 16
magnetisches Feld ... 19
magnetisches Moment ... 14
Magnetisierungsgrad ... 17
Magnetresonanz ... 14
Magnetresonanztomographie ... 8
Magnitudenbild ... 55
MRT-Aufnahme ... 7

N

Nettomagnetisierung ... 17
Neutron ... 14
Nukleon ... 12

O

Ortsbereich ... 43
Ortskodierung ... 19, 41, 54

P

paramagnetisch ... 17
Patientenlagerungstisch ... 10
Pauli-Ausschließungsprinzip ... 14
Phase ... 17
Phasenbild ... 55
Phasendifferenz ... 49
Phasenkodiergradient ... 13, 48, 57
Phasenkodierschritt ... 50, 54, 58
Phasenkohärenz ... 19
Präzession ... 12, 17
Präzessionsfrequenz ... 17, 42, 46
Projektion ... 9, 48
Projektions-Rekonstruktion ... 48
Projektionswinkel ... 48
Proton ... 14
Protonendichte ... 22
Pulssequenz ... 13, 57
Punkttechnik ... 44

Q

Quantenzahl ... 15
Quark ... 14
Quermagnetisierung ... 21, 52

R

Realbild ... 55
Rechte-Hand-Regel ... 15
Resonanzbedingung ... 42
Resonanzfrequenz ... 12, 18
Ringstrom ... 14
Rohdaten ... 53
Röhrenspannung ... 9
Röntgen ... 7, 8
Röntgenaufnahme ... 7
Röntgenfilm ... 8
Röntgenröhre ... 8
Röntgenspektrum ... 8
rotierendes Koordinatensystem ... 18

S

Schallwellen ... 11
Schallwellenwiderstände ... 11
Schichtbild ... 10
Schichtselektionsgradient ... 13, 45, 48, 55
Schichttechnik ... 48
Schleier ... 63
Schnittbildverfahren ... 8
Schwächungsprofil ... 9
schwärzen ... 8
Signal-zu-Rausch-Verhälnis ... 56
Signal-zu-Rausch-Verhältnis ... 43, 50, 60
Sonographie ... 11
Spin ... 12
Spin-Echo-Technik ... 57
Spule ... 14
Stabmagnet ... 14
Stimmgabel ... 18
Suszeptibilität ... 17, 22
Suszeptibilitätsartefakte ... 61

T

T_1 21
T_2 21
T_2* ... 22
Tomographieverfahren ... 8
transversale Relaxationszeit ... 22
Transversale Relaxationszeit ... 21
Transversaltomogramm ... 9
Tubo-Spin-Echo ... 61

Ü

Überlappung ... 59
Ultraschall ... 11
Ultraschallaufnahme ... 7
Ultraschallkopf ... 11
Ultraschallverfahren ... 8
Unschärfe ... 57

V

Verstärkerfolien ... 9
Vielschichttechnik ... 51
Volumentechnik ... 50

W

Wasserstoff ... 14
Wellenlänge ... 42
Window ... 11

Z

Zeitbereich ... 43
Zentrum ... 10