WSI/GRIS - VCM, Universität Tübingen

High Dynamic Range Windowing

Applied On

Images

Medical Data

Curvilinear Grids

Inhaltsverzeichnis

Benjamin Schnaidt

1 Die Idee

1.1 Windowing

Das so genannte *Windowing* (dt. Fensterung) ist das zentrale Thema dieser Arbeit. Daher möchten wir an dieser Stelle zuerst diesen Begriff einführen.

Ein einfaches Beispiel für Windowing könnte das Folgende sein: Man hat ein Bild mit 256 verschiedenen Grauwerten (Helligkeitswerten) gegeben und möchte es nun aber mit nur 16 Graustufen anzeigen. Dazu müssen 256 verschiedene Werte auf nur 16 Werte reduziert werden.

Abbildung 1: Links mit 256 Graustufen, rechts mit nur 16 Graustufen (Images\Schnaidt\Leaf1, 2.png)

Abbildung 1 zeigt das Ergebnis dieser Reduktion. Man kann leicht erkennen, dass die Qualität des rechten Bildes deutlich geringer ist, es wird nur noch "weniger genau" dargestellt.

Nun werden Sie an dieser Stelle natürlich zurecht fragen, weshalb solch eine Verminderung der Qualität des Bildes auf nur 16 Grauwerte sinnvoll sein sollte. In der Tat ist dies auch kein typisches Beispiel für Windowing. In der Realität wird es beispielsweise bei medizinischen Daten aus der Computertomographie (CT) verwendet. Vom Patient wird mit Hilfe eines CT-Scanners eine Aufnahme erzeugt, die typischerweise aus 4096 Grauwerten besteht. Ein üblicher Monitor eines PCs kann aber nur 256 Grauwerte darstellen. Hier müssen also 4096 Werte auf 256 Werte reduziert werden, prinzipiell auf dieselbe Weise wie im Beispiel oben.

In **Abbildung 2** sieht man zwei Beispiele von CT-Bildern, die mit nur 256 Grauwerten angezeigt werden. Ein Vergleich mit den Originalbildern ist an dieser Stelle natürlich nicht möglich, da Sie dieses Dokument entweder als Ausdruck oder an einem Bildschirm vorliegen haben. Nicht nur übliche Monitore sind auf 256 Grauwerte beschränkt, auch Drucker haben dieses Problem. Meist liefern sie sogar deutlich weniger unterscheidbare Helligkeitsstufen, wenn es sich um keine teuren, professionellen Geräte handelt.

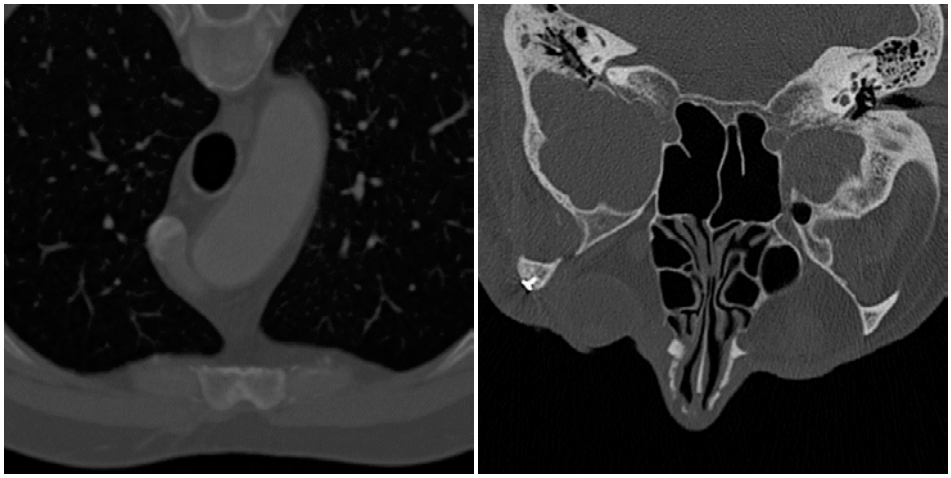

Abbildung 2: Links CT-Bild einer Lunge, rechts eines Kopfes (Images\Schnaidt\CtImage1, 2.png)

Der Grund, weshalb obige Reduktion in der Praxis als Windowing bezeichnet wird, hängt mit dem *Histogramm* eines Datensatzes zusammen. In **Abbildung 3** sehen Sie ein Beispiel für solch ein Histogramm. Nach rechts sind alle möglichen Grauwerte des Bildes abgetragen, beispielsweise von 0 bis 4095 bei CT-Daten. Die Höhe zum jeweiligen Grauwert beschreibt seine Häufigkeit. In diesem Histogramm haben besonders die fast schwarzen Pixel mit einem niedrigen Grauwert eine sehr hohe Häufigkeit, was auch typisch für CT-Daten ist, denn der luftgefüllte Hintergrund in einem CT ist grundsätzlich schwarz.

Abbildung 3: Links Histogramm, rechts Datensatz (Images\Schnaidt\Histogram1, 2.png)

Die Aufgabe des Windowing besteht nun darin, einen Teil - ein "Fenster" - des Histogramms auszuwählen, welches letztendlich dargestellt werden soll. Beispielsweise könnte man von den Grauwerten 0...4095 die Grauwerte 0...511 auswählen. Dieses "Fenster" bzw. Grauwertintervall mit 512 Grauwerten muss dann natürlich genauso wieder auf 256 Werte reduziert werden, wenn es auf dem Bildschirm dargestellt werden soll. Aber da nur 512 (statt 4096) auf 256 Werte reduziert werden müssen, geht in diesem Intervall weniger Genauigkeit verloren.

Abbildung 4: Links Histogramm, rechts Datensatz (Images\Schnaidt\HistogramPart1, 2.png)

In **Abbildung 4** wurde ein beinahe schwarzer Teil des Histogramms als Fenster ausgewählt und genau dieses Fenster wird mit 256 verschiedenen Grauwerten rechts dargestellt. Alle Werte oberhalb des ausgewählten Intervalls werden auf Weiß und alle unterhalb auf Schwarz abgebildet. Im Vergleich zum letzten Bild fällt deutlich auf, dass vorher schwarze Regionen in Wirklichkeit nicht einfach schwarz sind, sondern Details der Lunge und auch ein gewisses Hintergrundrauschen enthalten.

Das Beispiel verdeutlicht, dass das ausgewählte Fenster und die Genauigkeit unseres Bildschirms ganz entscheidend beeinflussen, wie wir den Datensatz wahrnehmen und welche Details wir noch erkennen können. Natürlich beeinflusst dies auch nicht nur die menschliche Wahrnehmung, auch maschinelle Algorithmen können beispielsweise bestimmte Organe nicht mehr identifizieren, wenn die Genauigkeit durch das Windowing zu stark reduziert wurde.

Gleichzeitig stellen sich in der Praxis aber die beiden folgenden Einschränkungen:

- Das ausgewählte Fenster umfasst normalerweise das komplette Histogramm. Im obigen Beispiel möchte man also alle 4096 Grauwerte auf die 256 Grauwerte des Bildschirms abbilden.
- Die Genauigkeit des Bildschirms liegt fast immer bei 256 Grauwerten und ist nicht einfach erweiterbar. Wie bereits erwähnt, stellen Drucker meist sogar noch weniger Grauwerte zur Verfügung.

Nun bleibt aber dennoch eine dritte Möglichkeit, die Qualität des angezeigten Bildes zu verbessern, auf die bisher noch nicht eingegangen wurde: Die Methode bzw. Art der Abbildung von einem großen Wertebereich auf einen kleinen Wertebereich.

In den bisherigen Bildern geschah dies *linear*. Dies ist die einfachste und daher auch am meisten verbreitete Art des Windowing, welche wieder am Besten am Beispiel von CT-Daten zu erklären ist (Eine mathematische Definition der Methoden finden Sie in Kapitel 3 ab Seite 99). 4096 Grauwerte entsprechen 12 Bit Daten, 256 Grauwerte 8 Bit Daten. Das lineare Windowing vernachlässigt die 4 Bits mit der niedrigsten Wertigkeit. Ein Beispiel hierfür sehen Sie in **Abbildung 5**. Die niedrigsten 4 Bits werden einfach verworfen.

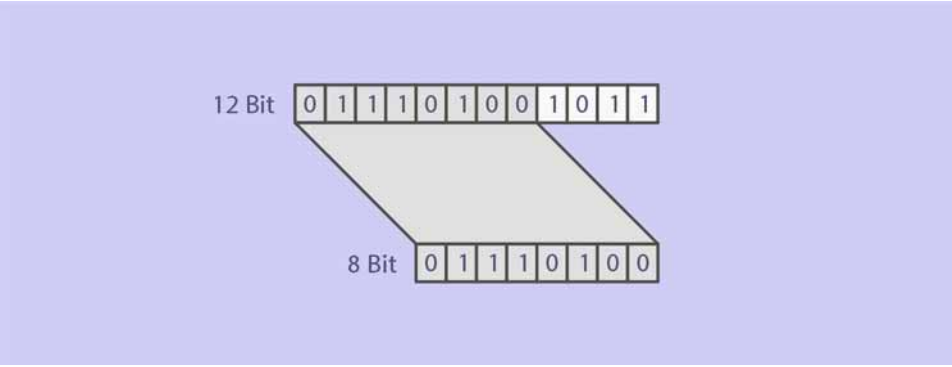

Abbildung 5: Lineares Windowing von 12 Bit auf 8 Bit (Images\Schnaidt\LinearWindowing.png)

Dieses Verfahren ist schnell und einfach. Es stellt sich aber die Frage, was geschieht, wenn die 4 verworfenen Bits wichtige Informationen enthielten. Man könnte sich vorstellen, dass in einem Datensatz zwischen dem Inneren eines Blutgefässes, der Gefäßwand und dem umliegenden Gewebe nur sehr geringe

Unterschiede sind. Reduziert man an dieser Stelle die Genauigkeit, wird man hier keine Unterschiede mehr sehen oder mit Algorithmen erkennen können.

1.2 High Dynamic Range Windowing

Das *High Dynamic Range Windowing* (auch High Dynamic Range Imaging oder Tone Reproduction genannt) versucht die lineare Abbildung durch eine Abbildung zu ersetzen, die an das jeweilige Bild angepasst ist und möglichst viel der Genauigkeit des Originals erhält.

Ziel ist zum einen das Erhalten von Details, aber auch gleichzeitig das Erzeugen eines "natürlichen" Bildes. Letzteres ist bei medizinischen Daten oder Simulationsdaten nicht leicht zu definieren, ist aber bei einer normalen Photographie intuitiv klar: Man möchte das Bild auf dem Monitor möglichst so wahrnehmen, wie es der Photograph mit seinem Auge wahrnahm, als er das Bild erzeugte. Dies gilt insbesondere für die Helligkeit des Bildes.

An dieser Stelle werden auch die Grenzen des High Dynamic Range Windowing deutlich:

- Die Unterscheidung zwischen nicht relevanten und relevanten Details ist algorithmisch schwierig und kann außerdem von den Erwartungen und Zielen des Betrachters abhängen. Letzteres kann ein allgemeiner Algorithmus, wie wir ihn hier vorstellen werden, nicht berücksichtigen.

- Das Bild enthält keine direkten Informationen mehr darüber, wie es aufgenommen wurde oder gar wie der Photograph die Szene sah. Bei medizinischen Daten wäre dies auch gar nicht möglich, da der Patient hierfür einer Cryosektion[1] unterzogen werden müsste. Daher ist ein "natürliches" Bild eine subjektive Wahrnehmung, die von Mensch zu Mensch variieren kann.

Das Ziel dieses Projektes war es, einen Algorithmus für das High Dynamic Range Windowing zu schreiben, der gleichzeitig auf einem breiten Spektrum von Datensätzen gute Ergebnisse liefert und dennoch rechnerisch effizient durchzuführen ist. Der Algorithmus basiert dabei auf einer Methode von Reinhard *et al.* (2002, p.267-276).

Konkret bedeutet dies:

- Zu den verwendeten Daten gehören reale Photographien, Renderings, medizinische Datensätze und außerdem Simulationsdaten mit verschiedenen Gittertypen. Wie Sie sicherlich bereits in der Einführung bemerkt haben, legen wir dabei einen besonderen Schwerpunkt auf die medizinischen Daten, da dieses Projekt im Arbeitsbereich VCM (*Visual Computing for Medicine*) des WSI/GRIS (*Wilhelm Schickard Institut für Informatik, GRaphisch-Interaktive Systeme*) der Universität Tübingen entstanden ist.

- Um die Rechenzeit für den jeweiligen Zweck in sinnvollen Grenzen zu halten, bieten wir verschiedene Qualitätsstufen an, die mit steigender Rechenzeit auch bessere Qualität liefern.

[1] Diese Technik wurde bei den so genannten *Visible Human* Datensätzen angewendet. Hierfür wird der tote Körper mit Flüssigkeit ausgefüllt, tiefgefroren und in dünne Scheiben zersägt. Auf diese Weise können farbechte Photographien des gesamten Körpers erzeugt werden.

- Außerdem vergleichen wir unseren Algorithmus mit anderen Methoden für das High Dynamic Range Windowing, um fundierte Qualitätsaussagen machen zu können.

Dieses Projekt wurde von uns gemeinsam bearbeitet, aber die einzelnen Kapitel unterteilen auch gleichzeitig die Aufgabenbereiche. Am Anfang des jeweiligen Kapitels finden Sie den Namen des Autors, der das betreffende Kapitel verfasst hat. Da die Kapitel grundsätzlich als unabhängige Einheiten betrachtet werden können, kann es dabei auch zu Redundanzen kommen.

Wir freuen uns selbstverständlich über Ihre Kritik und Anregungen,
Benjamin Schnaidt (BSC@SciMacros.de).

Benjamin Schnaidt

2 Das Programm

2.1 Vorwort

Im folgenden Kapitel finden Sie Details zum Programm, das unseren Algorithmus praktisch umsetzt. Das Kapitel richtet sich in erster Linie an Benutzer des Programms und kann als eine Art Bedienungsanleitung betrachtet werden. Es setzt daher keine mathematischen Vorkenntnisse über unseren Algorithmus voraus. Mathematische und auch technische Details finden Sie ab Kapitel 3, Seite 99.

Das Programm selbst ist in englischer Sprache geschrieben, daher verwenden wir auch in dieser Ausarbeitung an passender Stelle die englischen Ausdrücke, um die Bedienung des Programms zu vereinfachen und um Verwechslungen zu vermeiden.

2.2 Einführung

2.2.1 Erste Schritte

Wenn Sie das Programm zum ersten Mal starten, öffnet sich ein Dialog, der Ihnen als Hilfestellung bei der Einführung dienen kann (**Abbildung 6**). Sie können ihn auch im Menü über **Help > First Steps** erreichen.

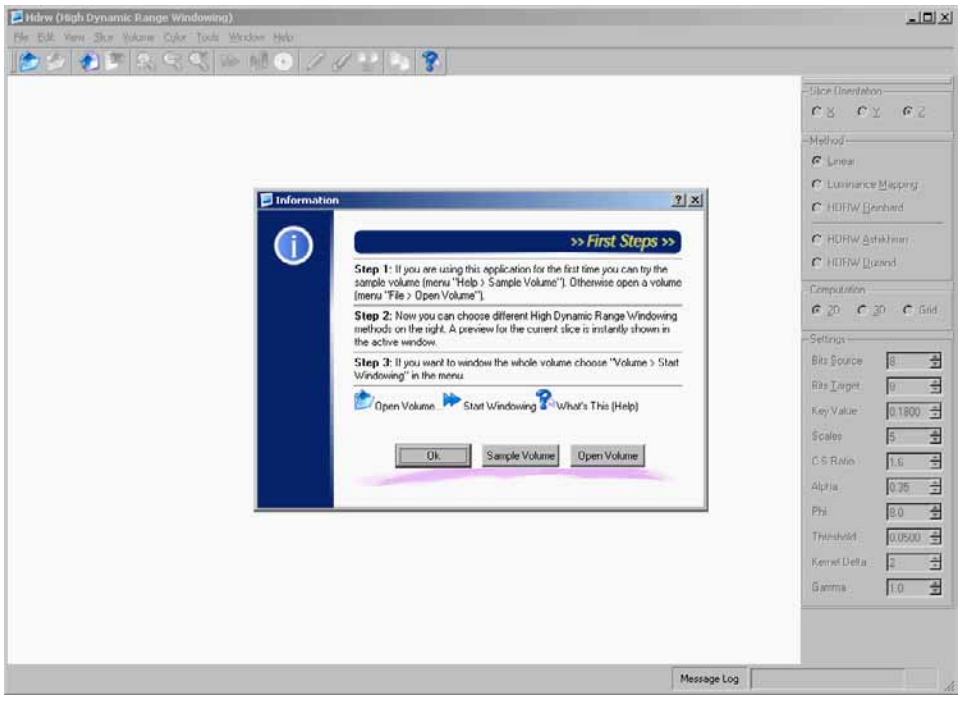

Abbildung 6: Erste Schritte nach dem ersten Laden von *Hdrw* (Images\Schnaidt\FirstSteps.png)

Zuerst benötigt man einen Datensatz, auf dem das Windowing durchgeführt wird. Da unser Programm mit allgemeinen Datensätzen arbeitet, die auch 3-dimensional sein können, werden diese im Programm als *Volume* (dt. Volumen)

bezeichnet. Im Prinzip ist ein Volume einen Stapel von Bildern, der im einfachsten Fall auch nur aus einem Bild bestehen kann.

Sie können den Dialog in **Abbildung 6** dazu nutzen, um ein Volumen zu laden, falls Sie bereits ein Volumen auf Ihrer Festplatten haben (**Open Volume**) oder Sie können auf **Sample Volume** klicken. Dies erzeugt ein Beispielvolumen, mit dem sich die Funktionen des Programms testen lassen.

Abbildung 7 zeigt das Beispielvolumen. Da das Bild fest in das Programm integriert ist und nicht zuviel Speicherplatz verbrauchen darf, handelt es sich eigentlich um ein normales Bild, welches in ein künstliches High Dynamic Range Bild umgerechnet wurde. Es eignet sich aber gut, um die verschiedenen Methoden des Programms zu testen und verhält sich dabei nahezu wie ein echter Datensatz.

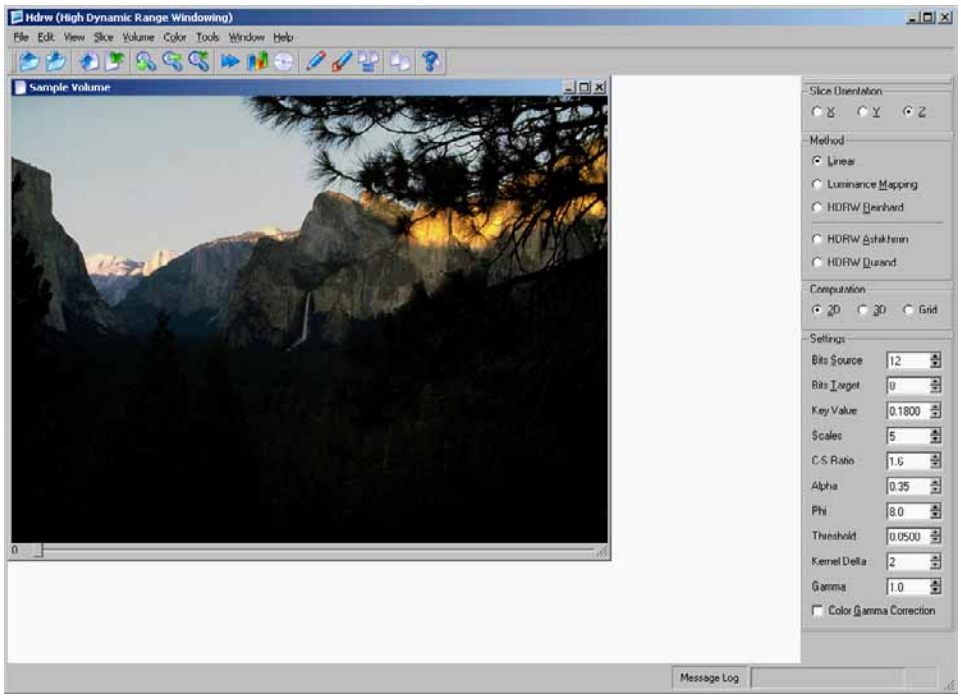

Abbildung 7: Das Beispielvolumen (Images\Schnaidt\SampleVolume.png)

Auf der rechten Seite des Bildschirms finden Sie ein Fenster mit den Optionen. Hiermit lässt sich die Methode des Windowing und außerdem die genauen Einstellungen festlegen. Zu Beginn wird **Linear** Windowing verwendet, welches Sie bereits in Kapitel 1 kennen gelernt haben. Das Beispielvolumen hat ebenfalls 4096 verschiedene Helligkeitswerte (Grauwerte), die in diesem Fall mit Farben kombiniert sind. Dies können Sie an der Einstellung **Bits Source 12** erkennen (12 Bits entspricht 4096 Werten). Die Ausgabe auf dem Bildschirm, das heißt, was sie in der Abbildung sehen, geschieht mit 8 Bit (**Bits Target 8**). Diese beiden Werte müssen Sie im Normalfall nicht ändern.

Sie können nun verschiedene Windowing Methoden testen, beispielsweise **Luminance Mapping** oder **Hdrw Reinhard**. Sobald Sie die entsprechende Einstellung ändern, wird das Bild automatisch aktualisiert.

Unter dem Volumen in **Abbildung 7** sehen Sie einen Schieberegler. Hiermit können Sie zwischen den einzelnen Bildern (Schichten) des Volumens wechseln. Daneben wird die Nummer der aktuellen Schicht angezeigt, im Bild **0**.

2.2.1.1 Windowing des gesamten Volumens

Was Sie im Fenster sehen, ist nur eine *Preview* (dt. Vorschau) für die aktuelle Schicht des Volumens. Das Windowing des gesamten Volumens kann je nach Methode und Größe des Volumens etwas mehr Zeit in Anspruch nehmen. Um dies zu starten, wählen Sie um Menü **Volume > Start Windowing** oder das Icon .

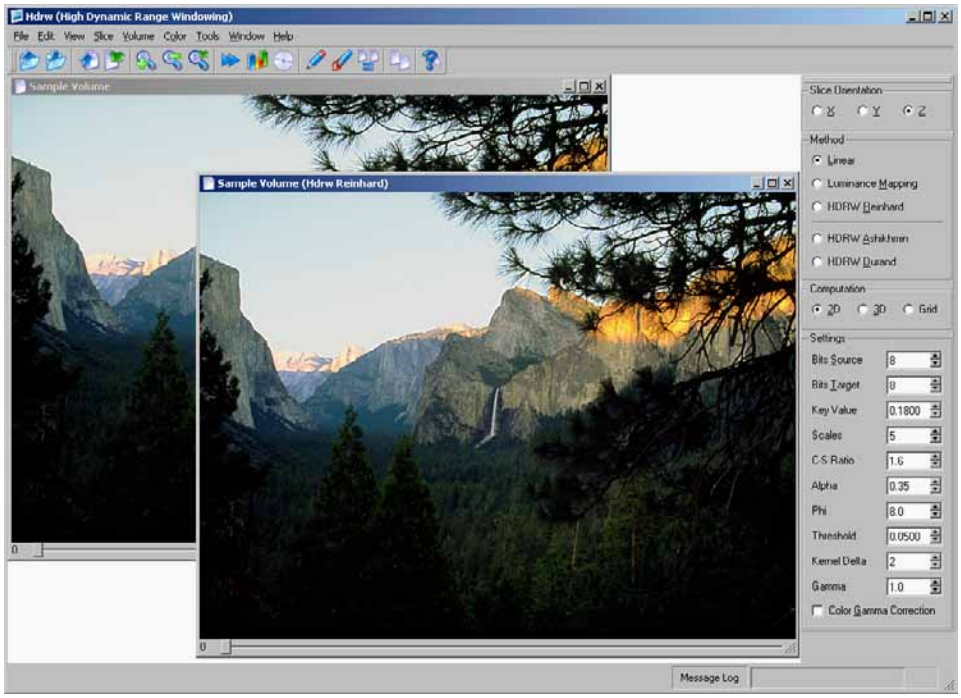

Abbildung 8: Nach dem Windowing (Images\Schnaidt\SampleVolumeHdrwReinhard.png)

Abbildung 8 zeigt das Ergebnis des Windowing mit der Methode **Hdrw Reinhard**. Das Resultat erscheint in einem neuen Fenster und ist nun ein Volumen mit nur noch 8 Bit. Dieses Volumen kann nun zum Beispiel abgespeichert und in einem anderen Programm verwendet werden.

2.2.1.2 Das Hilfesystem

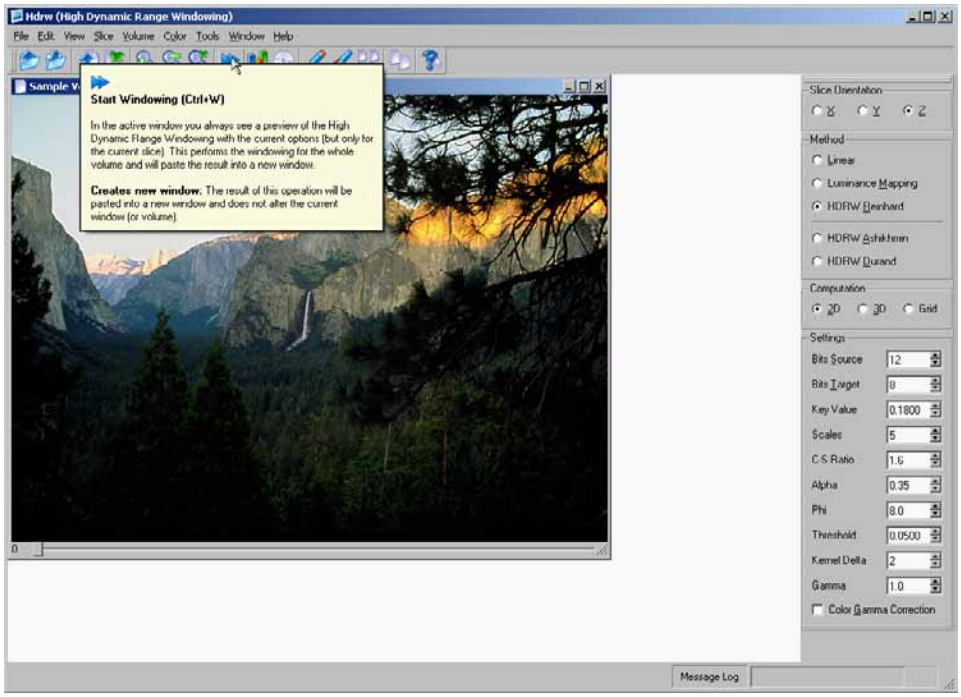

Abbildung 9: What's This Hilfe (Images\Schnaidt\WhatsThis.png)

Im Menü unter **Help > What's This** oder mit dem Icon können Sie auf ein beliebiges anderes Icon oder Interface Element klicken und erhalten dazu direkt einen kurzen Hilfetext, der die Funktion beschreibt (Siehe **Abbildung 9**).

2.2.1.3 Weiterführende Hilfe

Im Rest dieses Kapitels finden Sie eine genauere Beschreibung aller Funktionen und Einstellungen des Programms. Dies ersetzt die übliche Hilfedatei, welche bei den meisten Programmen mit enthalten ist.

Sie können sich an dieser Stelle auch nur die wichtigsten Funktionen durchlesen und dieses Kapitel später bei Bedarf zum Nachschlagen benutzen.

Unterhalb der meisten Funktionen finden Sie eine oder mehrere der folgenden *Anmerkungen*, die Ihnen angeben, wie die Funktion arbeitet:

- **Arbeitet auf dem Volumen:** Diese Funktion arbeitet auf dem Volumen und nicht auf der Vorschau für die aktuelle Schicht, die Sie im aktiven Fenster sehen. Wenn Sie dies nach dem Windowing anwenden wollen, wählen Sie zuerst **Volume > Start Windowing** aus, um das Windowing für das ganze Volumen durchzuführen.

- **Arbeitet auf der Vorschau:** Diese Funktion arbeitet auf der Vorschau für die aktuelle Schicht, die Sie im aktiven Fenster sehen.

- **Öffnet ein neues Fenster:** Das Ergebnis dieser Funktion wird in ein neues Fenster eingefügt und ändert nicht das aktuelle Fenster (bzw. Volumen).

- **Ändert das Volumen:** Diese Funktion ändert das Volumen des aktiven Fensters. Wenn Sie diese Funktion rückgängig machen wollen, speichern Sie das Volumen zuerst.

2.3 Die Funktionen

2.3.1 Menü File

2.3.1.1 File > Open Volume

Diese Funktion öffnet ein Volumen aus einer Datei. Folgende Volumenformate werden von *Hdrw* unterstützt:

- *SLC* Format: Ein typisches Format für medizinische Volumen mit 8 oder 16 Bit *Integern* (dt. Ganzzahlen). Zusätzlich unterstützt *Hdrw* 32 Bit und 64 Bit *Floats* (floating-point numbers, dt. Gleitkommazahlen).
- *SCALAR* Format: Speichert pro Gitterpunkt einen skalaren Wert in 32 Bit Float Genauigkeit. Zu dem Format gehört üblicherweise eine *.grid* oder *.mesh* Datei, welche das Gitter beschreibt, auf dem die skalaren Daten angeordnet sind.
- *VECTOR* Format: Speichert pro Gitterpunkt einen Vektor mit 3 Werten in 32 Bit Float Genauigkeit. Zu dem Format gehört üblicherweise eine *.grid* oder *.mesh* Datei, welche das Gitter beschreibt, auf dem die Vektoren angeordnet sind.

In **Abbildung 10** sehen Sie den Dialog für das Öffnen einer *SLC* Datei.

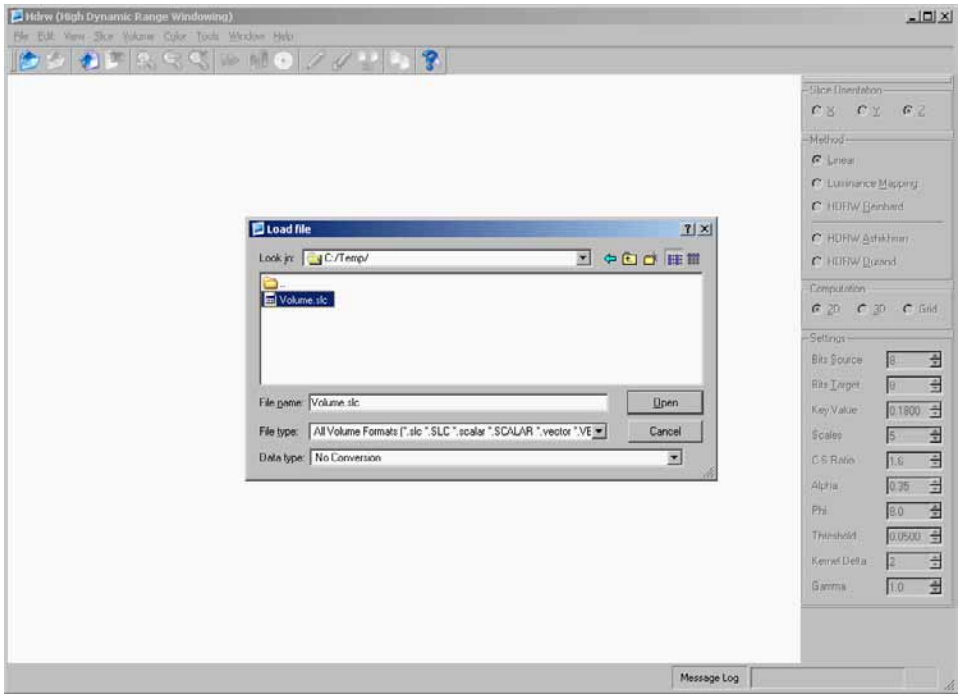

Abbildung 10: File > Open Volume (Images\Schnaidt\FileOpenVolume.png)

Nachdem Sie eine passende Datei ausgewählt haben, können Sie unter **Data Type** zusätzlich den Datentyp einstellen, in den das Volumen beim Laden umgewandelt wird. Im Normalfall können Sie dies bei **No Conversion** belassen (Siehe **Abbildung 10**), wodurch das Volumen mit dem Datentyp geladen wird, der in der Datei selbst benutzt wird (Bei einer *SCALAR* Datei ist dies beispielsweise immer ein 32 Bit Float, wie oben erwähnt wurde). Sie haben aber außerdem folgende Typen zur Auswahl:

- **Integer (8 Bit):** 8 Bit Ganzzahl, d.h. maximal 256 verschiedene Grauwerte
- **Integer (16 Bit):** 16 Bit Ganzzahl, d.h. maximal 2^{16} verschiedene Grauwerte
- **Integer (32 Bit):** 32 Bit Ganzzahl, d.h. maximal 2^{32} verschiedene Grauwerte
- **Floating-Point Number (32 Bit):** 32 Bit Gleitkommazahl
- **Floating-Point Number (64 Bit):** 64 Bit Gleitkommazahl

Von oben nach unten liefern die Datentypen eine höhere Genauigkeit und dadurch eventuell bessere Bildresultate, brauchen aber auch mehr Speicher und Rechenzeit.

Zusätzlich unterstützt *Hdrw* komprimierte Dateien im *GZ* Format, welche an der typischen Endung *.gz* erkennbar sind (Beispielsweise *Volumen.slc.gz*). Dies ist eine unter Unix weit verbreitete Kompression, die auch im *PNG* Bildformat verwendet wird. Sie können jedes beliebige Dateiformat im Programm komprimiert laden und auch wieder komprimiert speichern. Je nach Datei können Sie so bis zu 50% des Speicherplatzes sparen. Besonders bei großen Volumen kann dies Vorteile bringen, ohne die Ladezeiten zu sehr zu beeinflussen (Das Speichern dauert durch die Kompression etwas länger).

2.3.1.2 File > Save Volume As

Hiermit können Sie ein Volumen speichern. Dabei stehen Ihnen dieselben Formate wie beim Laden von Volumen zur Verfügung (Siehe 2.3.1.1, Seite 15) und zusätzlich:

- *SEG* Format: Dies ist ein Format speziell für die Segmentierung. Es speichert alle segmentierten Voxel, d. h. Voxel mit Grauwert 255. Mehr zur Segmentierung finden Sie in 2.3.7.1, Seite 58.

In **Abbildung 11** sehen Sie den Dialog zum Speichern.

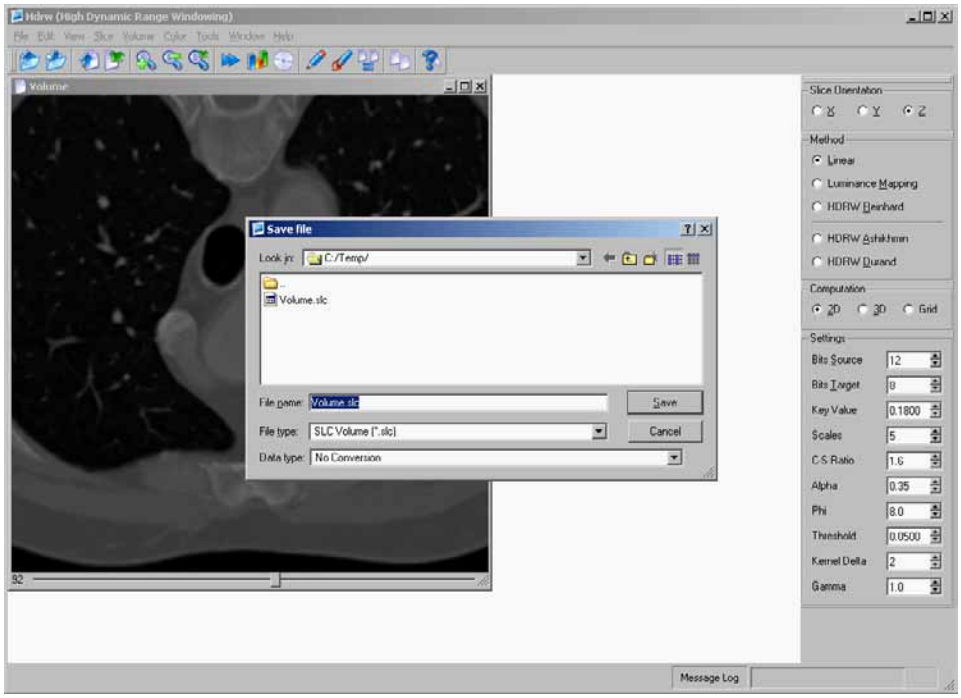

Abbildung 11: File > Save Volume As (Images\Schnaidt\FileSaveVolumeAs.png)

Mit **Data Type** können Sie den Datentyp festlegen, der in der Datei gespeichert wird. Bei den Formaten *SCALAR* und *VECTOR* wird aber immer der Datentyp **Floating-Point Number 32 Bit** verwendet, da die Formate nur hierfür definiert sind.

Wichtige Anmerkungen:

- **Arbeitet auf dem Volumen:** Diese Funktion arbeitet auf dem Volumen und nicht auf der Vorschau für die aktuelle Schicht, die Sie im aktiven Fenster sehen. Wenn Sie dies nach dem Windowing anwenden wollen, wählen Sie zuerst **Volume > Start Windowing** aus, um das Windowing für das ganze Volumen durchzuführen.

Weiterführende Informationen:

- Verfügbare Datentypen (2.3.1.1, Seite 15)
- *GZ* komprimierte Dateien (2.3.1.1, Seite 15)

2.3.1.3 File > Open Images As Volume

Mit dieser Funktion können Sie mehrere Bilder auf einmal laden und gemeinsam zu einem Volumen verschmelzen. Die Schichten des neuen Volumens entsprechen dabei gerade den geladenen Bildern.

Da jedes Volumen eine feste Breite, Höhe und Tiefe hat (wie ein Quader), ist es vorteilhaft, wenn die Bilder alle dieselbe Größe haben. Ansonsten nimmt *Hdrw* die Breite und Höhe des ersten Bildes als Vorgabe und verkleinert bzw. vergrößert die folgenden Bilder dementsprechend.

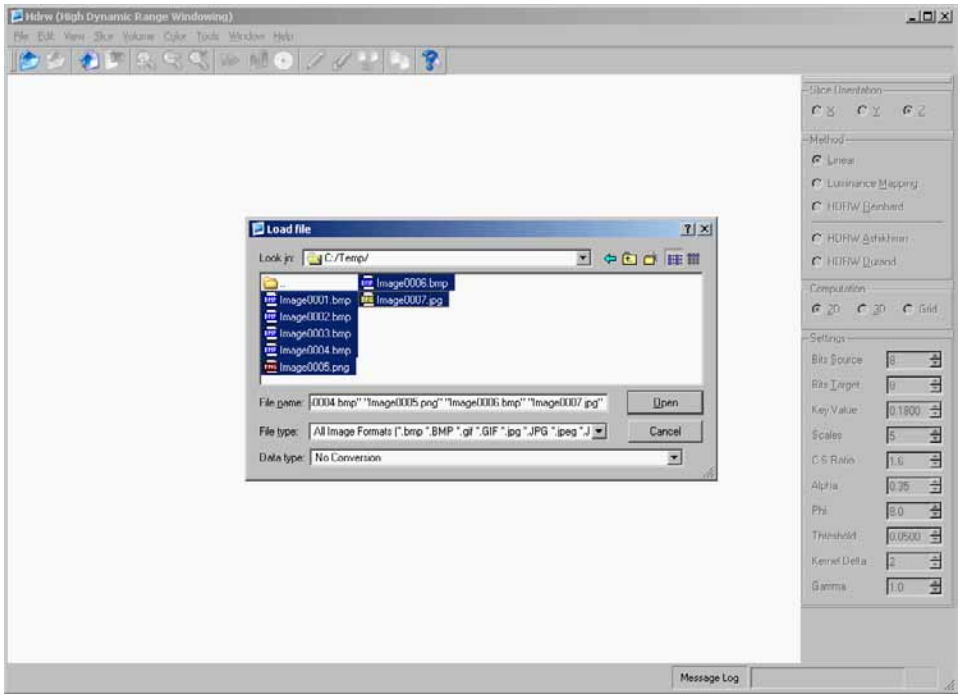

Abbildung 12: File > Open Images As Volume (Images\Schnaidt\FileOpenImagesAsVolume.png)

In **Abbildung 12** werden beispielsweise 7 verschiedene Bilder geladen und zu einem Volumen verbunden (Mit der Shift Taste lassen sich mehrere Dateien gleichzeitig markieren). Dabei sind alle Bildformate verfügbar, die *Trolltech Qt* einlesen kann (Dies ist der Framework, mit dem *Hdrw* erstellt wurde). Verbreitete Formate darunter sind das *JPG* Format, das *PNG* Format, das *BMP* Format und das *GIF* Format.

Da es sich um normale Bilddateien handelt, haben diese maximal nur 256 verschiedene Helligkeitswerte wie Ihr Monitor (Dies entspricht 8 Bit). Farbige Bilder werden üblicherweise im 24 Bit *RGB* Format gespeichert. Dies bedeutet, dass 8 Bit für Rot, 8 Bit für Grün und 8 Bit für Blau verfügbar sind. Obwohl sich damit etwa 16,8 Millionen verschiedene Farben darstellen lassen, entspricht dies ebenfalls nur 256 verschiedenen Helligkeitswerten. Auch die meisten Grafikkarten bzw. Monitore benutzen das 24 Bit *RGB* Format (Oder das 32 Bit *RGB* Format mit einem zusätzlichen 8 Bit Transparenzwert, den *Hdrw* aber nicht benötigt).

Weiterführende Informationen:

- Verfügbare Datentypen (2.3.1.1, Seite 15)
- *GZ* komprimierte Dateien (2.3.1.1, Seite 15)

2.3.1.4 File > Save Volume As Images

Ein Volumen kann auch wieder in Form einzelner Bilder gespeichert werden. Sie können sich dabei das Volumen als Quader vorstellen, der in einzelne Schichten zerlegt und Schicht für Schicht abgespeichert wird.

Dabei stehen Ihnen die Bildformate zur Verfügung, die *Trolltech Qt* schreiben kann (Dies ist der Framework, mit dem *Hdrw* erstellt wurde). Verbreitete Formate darunter sind das *JPG* Format, das *PNG* Format und das *BMP* Format.

In **Abbildung 13** sehen Sie ein Beispiel hierfür. Der angegebene Dateiname wird als Präfix für den Dateinamen benutzt, im Bild **NewImage**. Die Bilder der einzelnen Schichten werden danach von *Hdrw* automatisch nummeriert (NewImage0000, NewImage0001, ...).

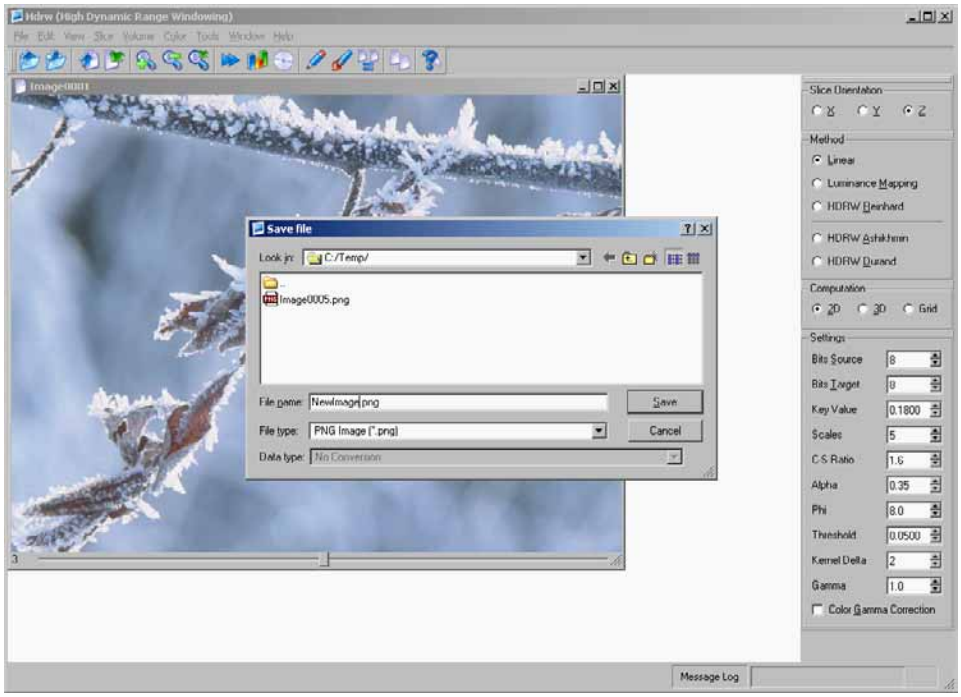

Abbildung 13: File > Save Volume As Images (Images\Schnaidt\FileSaveVolumeAsImages.png)

Da es sich um normale Bilddateien handelt, können diese maximal 256 verschiedene Helligkeitswerte speichern (Dies entspricht 8 Bit). Entweder handelt es sich dabei um 8 Bit Graustufenbilder oder 24 Bit *RGB* Farbbilder, die beide nur 256 verschiedene Helligkeitswerte ermöglichen.
Daher eignet sich diese Funktion am besten für Volumen, auf denen bereits ein Windowing durchgeführt wurde (Siehe 2.2.1.1, Seite 13).

Wichtige Anmerkungen:

- **Arbeitet auf dem Volumen:** Diese Funktion arbeitet auf dem Volumen und nicht auf der Vorschau für die aktuelle Schicht, die Sie im aktiven Fenster sehen. Wenn Sie dies nach dem Windowing anwenden wollen, wählen Sie zuerst **Volume > Start Windowing** aus, um das Windowing für das ganze Volumen durchzuführen.

Weiterführende Informationen:

- 24 Bit *RGB* Format (2.3.1.3, Seite 18)

- *GZ* komprimierte Dateien (2.3.1.1, Seite 15)

2.3.1.5 File > Save Slice As

Mit dieser Funktion lässt sich die aktuelle Schicht des Volumens als Bilddatei speichern. Sie wird dabei so gespeichert, wie sie im aktiven Fenster momentan angezeigt wird. D. h. dies betrifft die Vorschau für die aktuelle Schicht, auf die bereits das Windowing angewendet wurde (Auch wenn dies für das gesamte Volumen noch nicht der Fall war).

Dabei stehen Ihnen die Bildformate zur Verfügung, die *Trolltech Qt* schreiben kann (Dies ist der Framework, mit dem *Hdrw* erstellt wurde). Verbreitete Formate darunter sind das *JPG* Format, das *PNG* Format und das *BMP* Format.

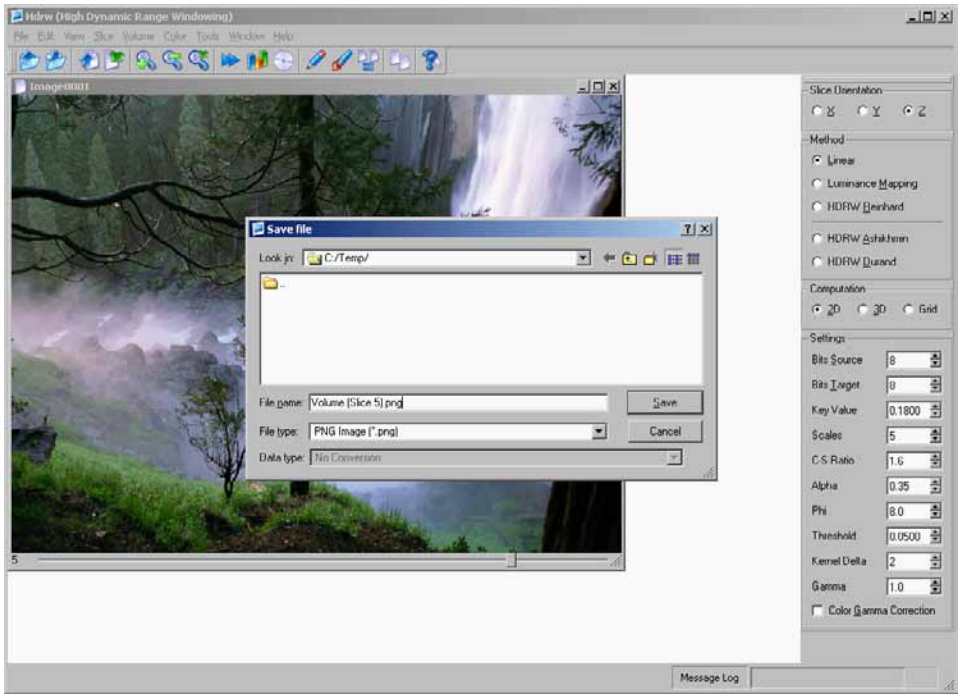

Abbildung 14: File > Save Slice As (Images\Schnaidt\FileSaveSliceAs.png)

In **Abbildung 14** sehen Sie hierfür ein Beispiel. Die aktuelle Schichtnummer wird zum Dateinamen automatisch hinzugefügt.

Wichtige Anmerkungen:

- **Arbeitet auf der Vorschau:** Diese Funktion arbeitet auf der Vorschau für die aktuelle Schicht, die Sie im aktiven Fenster sehen.

Weiterführende Informationen:

- 24 Bit *RGB* Format (2.3.1.3, Seite 18)
- *GZ* komprimierte Dateien (2.3.1.1, Seite 15)

2.3.1.6 File > Reset To Default Options

Hiermit können Sie die Optionen, die Sie in **Abbildung 15** auf der rechten Seite sehen, auf die Standardeinstellungen zurücksetzen. Die Standardeinstellungen liefern bei den meisten Bildern gute Resultate, es kann allerdings sein, dass Sie die Helligkeit mit einem höheren oder niedrigeren **Gamma** Wert anpassen müssen, je nachdem was für eine Methode (**Method**) Sie benutzen.

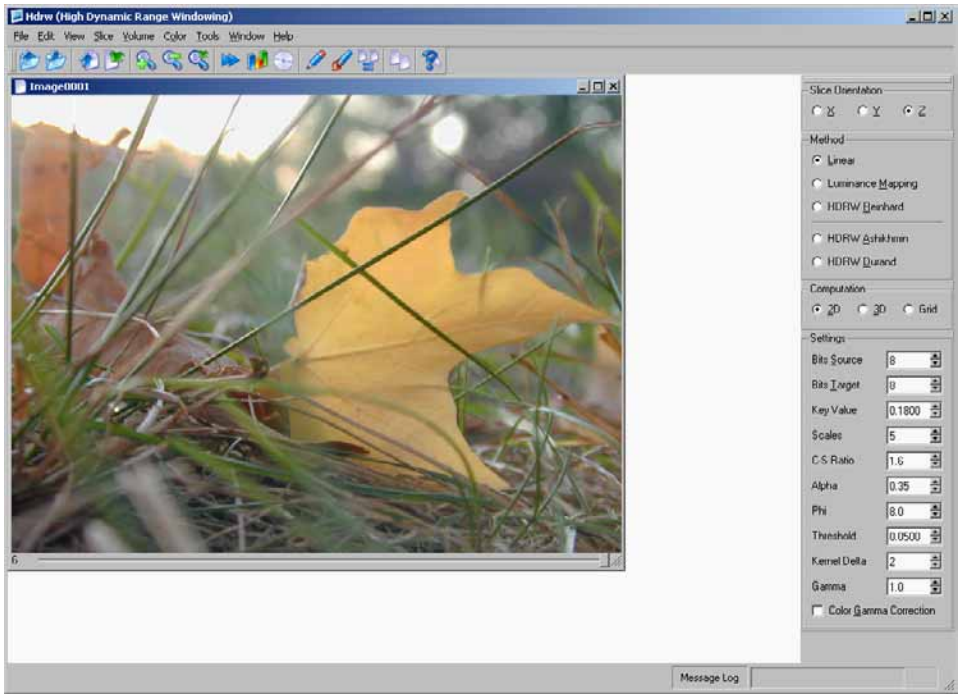

Abbildung 15: File > Reset To Default Options (Images\Schnaidt\FileResetToDefaultOptions.png)

2.3.1.7 File > Open Options File

Um die Optionen (auf der rechten Seite in **Abbildung 16**) aus einer Datei wiederherzustellen, können Sie diese Funktion benutzen. Sie lädt eine Datei im *HDRW* Format (mit der Endung *.hdrw*), die alle gespeicherten Optionen enthält. Mit **File > Save Options File As** können Sie solch eine Datei erstellen.

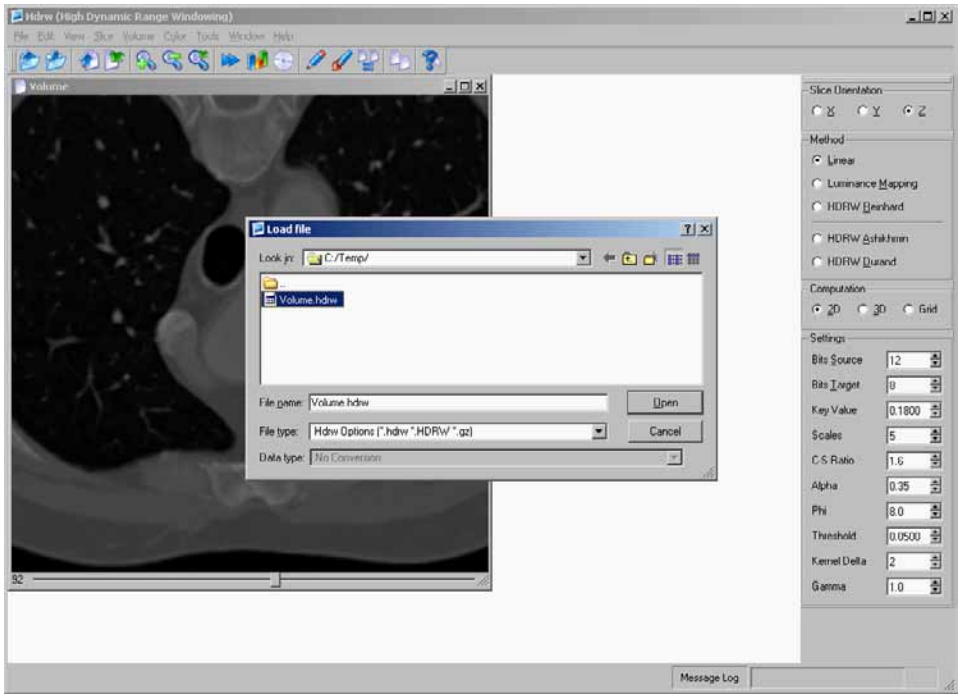

Abbildung 16: File > Open Options File (Images\Schnaidt\FileOpenOptionsFile.png)

Weiterführende Informationen:

- **File > Save Options File As** (2.3.1.8, Seite 24)
- *GZ* komprimierte Dateien (2.3.1.1, Seite 15)

2.3.1.8 File > Save Options File As

Die Optionen für das Windowing (auf der rechten Seite von **Abbildung 17**) lassen sich mit dieser Funktion in einer Datei speichern. Dies ist hilfreich, wenn Sie für ein Volumen die Standardeinstellungen angepasst haben, um ein besseres Resultat zu erhalten - auf diese Weise können Sie die Optionen dann sichern.

Abbildung 17: File > Save Options File As (Images\Schnaidt\FileSaveOptionsFileAs.png)

Wenn Sie die Datei genauso wie das Volumen benennen (Beispielsweise *Volume.hdrw* für *Volume.slc*), wird die Datei beim nächsten Öffnen des Volumens automatisch mitgeladen und die gespeicherten Optionen wiederhergestellt.

Weiterführende Informationen:

- **File > Open Volume** (2.3.1.1, Seite 15)
- **File > Open Options File** (2.3.1.7, Seite 23)
- *GZ* komprimierte Dateien (2.3.1.1, Seite 15)

2.3.1.9 File > Exit

Hiermit beenden Sie *Hdrw* (Siehe **Abbildung 18**). Bitte speichern Sie zuvor alle Volumen und Einstellungen, die Sie später eventuell noch benötigen.

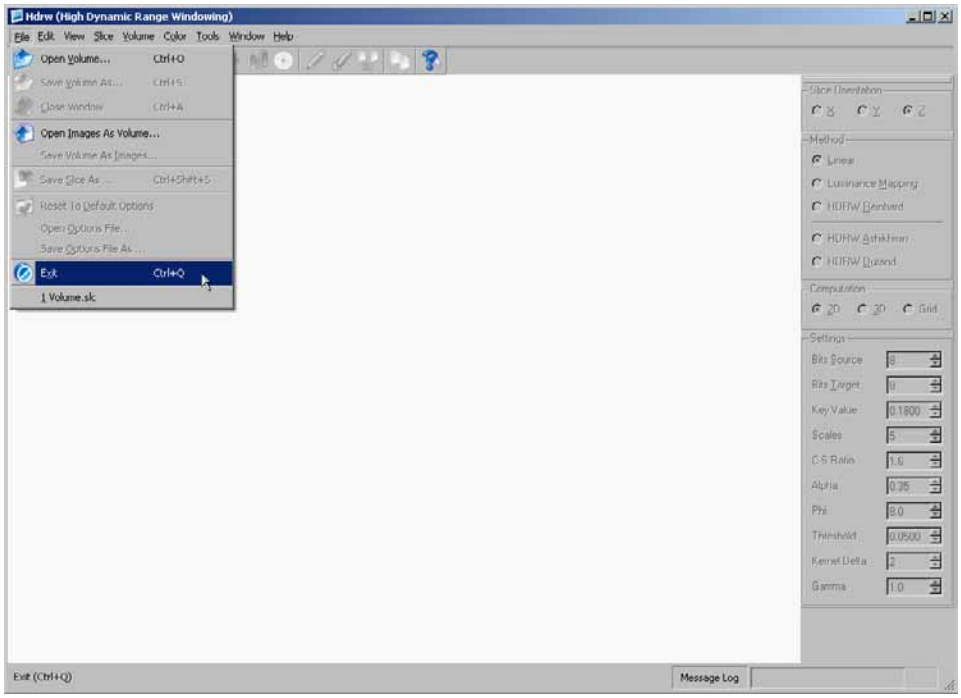

Abbildung 18: File > Edit (Images\Schnaidt\FileExit.png)

2.3.2 Menü Edit

2.3.2.1 Edit > Copy Slice

Diese Funktion kopiert die aktuelle Schicht in die Zwischenablage, wie sie im aktiven Fenster angezeigt wird (D.h. bereits nach dem Windowing, siehe **Abbildung 19**).

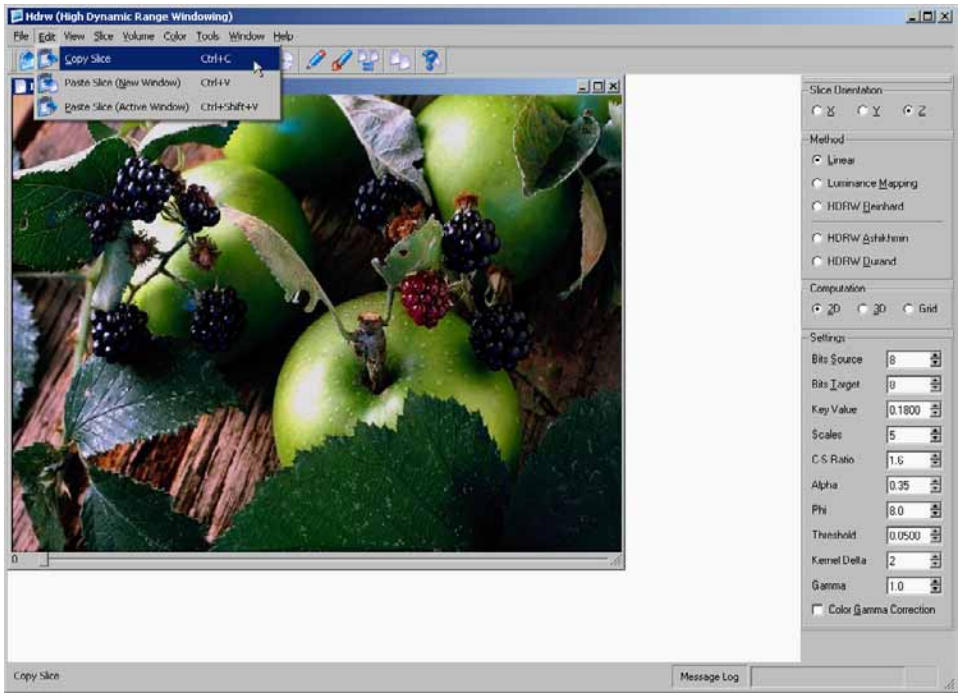

Abbildung 19: Edit > Copy Slice (Images\Schnaidt\EditCopySlice.png)

Da das normale Format für Bilder benutzt wird, können Sie den Inhalt der Zwischenablage danach in jedem gängigen Bildverarbeitungsprogramm (Adobe Photoshop, Corel PHOTO-PAINT, ...) oder Textverarbeitungsprogramm (Microsoft Word, ...) einfügen.

Wichtige Anmerkungen:

- **Arbeitet auf der Vorschau:** Diese Funktion arbeitet auf der Vorschau für die aktuelle Schicht, die Sie im aktiven Fenster sehen.

Weiterführende Informationen:

- 24 Bit *RGB* Format (2.3.1.3, Seite 18)

2.3.2.2 Edit > Paste Slice (New Window)

Dies fügt den Inhalt der Zwischenablage in ein neues Fenster ein (**New Image** in **Abbildung 20**).

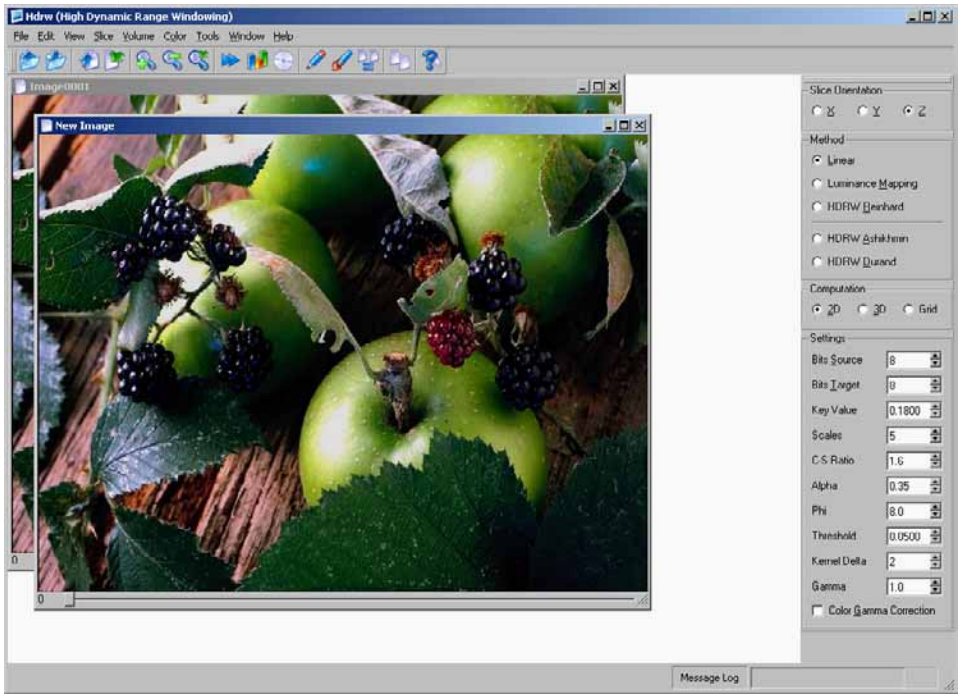

Abbildung 20: Edit > Paste Slice New Window (Images\Schnaidt\EditPasteSliceNewWindow.png)

Da das normale Bildformat der Zwischenablage benutzt wird, können Sie beliebige Bilder aus anderen Programmen hier einfügen.

Da es sich um normale Bilder handelt, können diese maximal 256 verschiedene Helligkeitswerte haben (Dies entspricht 8 Bit). Dies gilt auch für Farbbilder im 24 Bit *RGB* Format.

Weiterführende Informationen:

- 24 Bit *RGB* Format (2.3.1.3, Seite 18)

2.3.2.3 Edit > Paste Slice (Active Window)

Mit dieser Funktion können Sie ein Bild aus der Zwischenablage in das Volumen einfügen. Das Bild wird dabei hinter der aktuellen Schicht platziert (In **Abbildung 21** wurde das Bild hinter Schicht 0 eingefügt und landet damit auf Schicht 1).

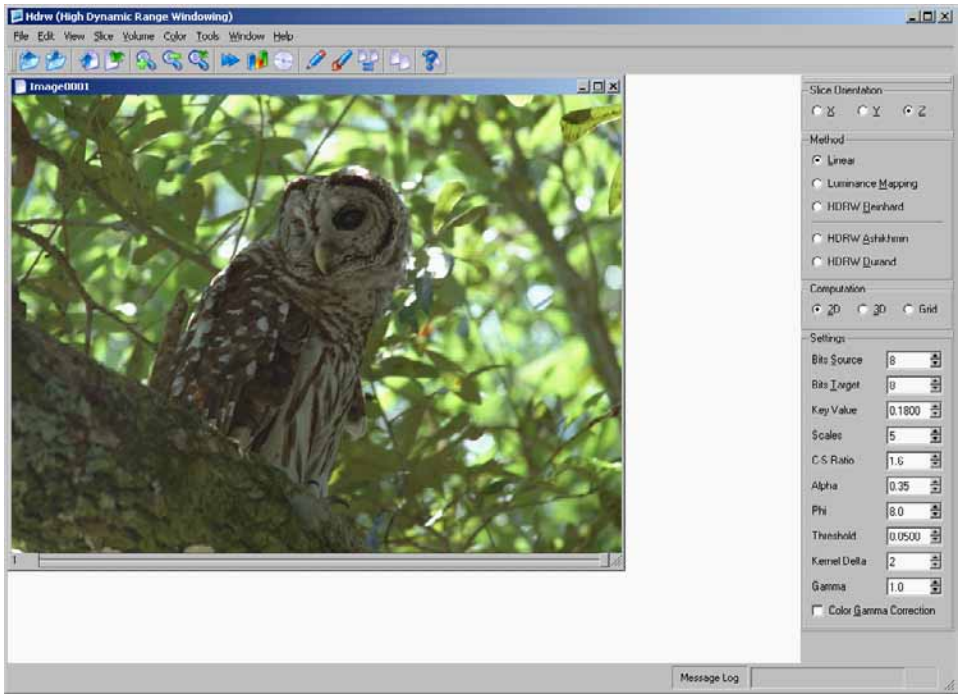

Abbildung 21: Edit > Paste Slice Active Window (Images\Schnaidt\EditPasteSliceActiveWindow.png)

Das Volumen selbst ist wie ein Quader mit einer bestimmten Breite, Höhe und Tiefe. Damit das Bild in das Volumen eingefügt werden kann, hat es am Besten dieselbe Breite und Höhe (Ansonsten wird das Bild automatisch von *Hdrw* verkleinert oder vergrößert).

Da es sich um normale Bilder handelt, können diese maximal 256 verschiedene Helligkeitswerte haben (Dies entspricht 8 Bit). Dies gilt auch für Farbbilder im 24 Bit *RGB* Format.
Es kann dabei passieren, dass eingefügte Bilder in einem Volumen mit beispielsweise 4096 Grauwerten sehr dunkel erscheinen. Dies rührt daher, dass im Vergleich zum Grauwert 4095 (Weiß im Volumen) der Grauwert 255 (Weiß im eingefügten Bild) sehr dunkel ist. In diesem Fall kann es hilfreich sein, zuerst mit **Volume > Start Windowing** das Windowing für das Volumen durchzuführen und dann das Bild einzufügen, dann haben sowohl Volumen als auch das Bild 256 Grauwerte.

Wichtige Anmerkungen:

- **Ändert das Volumen:** Diese Funktion ändert das Volumen des aktiven Fensters. Wenn Sie diese Funktion rückgängig machen wollen, speichern Sie das Volumen zuerst.

Weiterführende Informationen:

- 24 Bit *RGB* Format (2.3.1.3, Seite 18)

2.3.3 Menü View

2.3.3.1 View > Zoom In & Out, To Fit, 25%, 33%, ...

Hiermit können Sie die Größe des aktiven Fensters anpassen. Es stehen Ihnen folgende Funktionen zur Verfügung:

- **Zoom In:** Vergrößert das Bild um 25% (Siehe **Abbildung 22**)
- **Zoom Out:** Verkleinert das Bild um 25% (Siehe **Abbildung 22**)
- **Zoom To Fit:** Maximiert das Fenster (Siehe **Abbildung 23**)
- **Zoom 25%:** Verkleinert das Fenster auf 25% der Originalgröße
- **Zoom 33%:** Verkleinert das Fenster auf 33% der Originalgröße
- **Zoom 50%:** Verkleinert das Fenster auf 50% der Originalgröße
- **Zoom 75%:** Verkleinert das Fenster auf 75% der Originalgröße
- **Zoom 100%:** Setzt das Bild auf seine Originalgröße zurück
- **Zoom 200%:** Vergrößert das Fenster auf 200% der Originalgröße
- **Zoom 300%:** Vergrößert das Fenster auf 300% der Originalgröße
- **Zoom 400%:** Vergrößert das Fenster auf 400% der Originalgröße
- **Zoom 500%:** Vergrößert das Fenster auf 500% der Originalgröße

Es ändert sich dabei nur die Größe des angezeigten Fensters, das Volumen bleibt dabei unverändert. Mit **Volume > Resize** können Sie dagegen die Größe des Volumens permanent ändern.

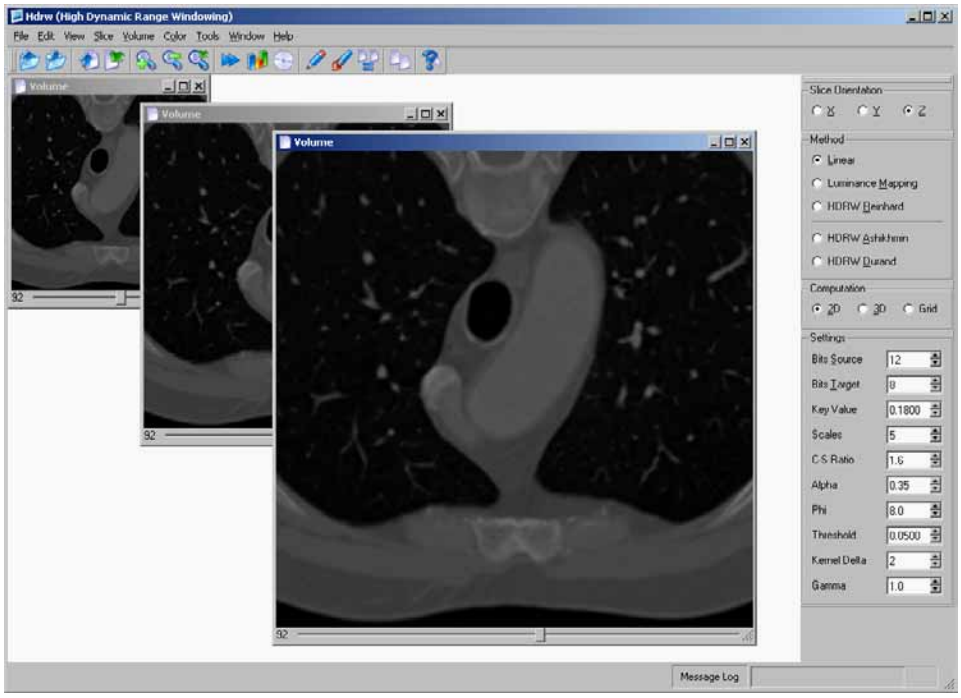

Abbildung 22: View > Zoom (Images\Schnaidt\ViewZoom.png)

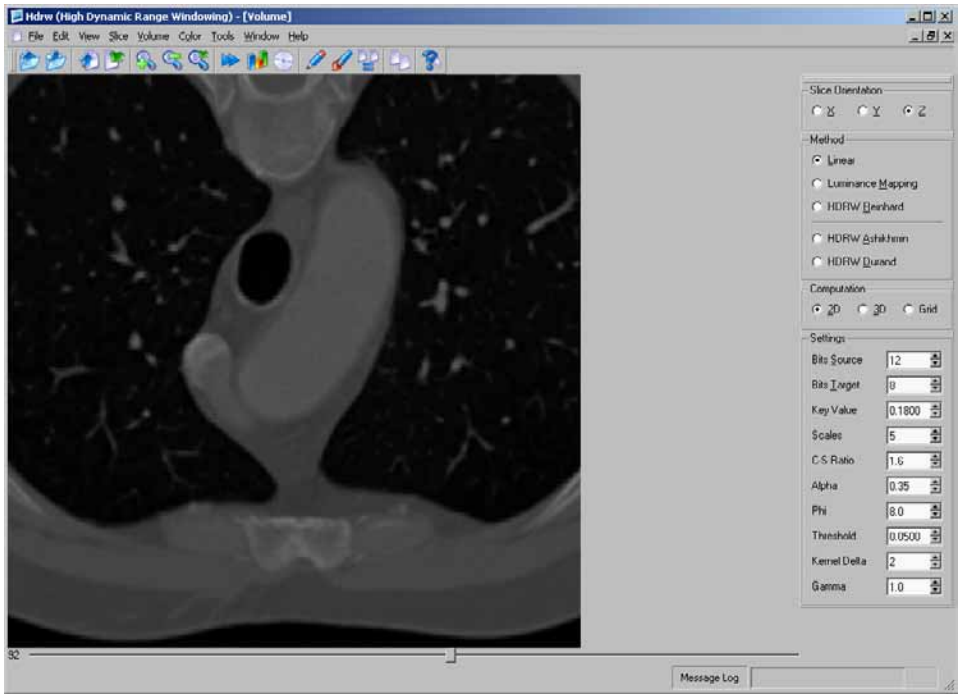

Abbildung 23: View > Zoom To Fit (Images\Schnaidt\ViewZoomToFit.png)

Zum Vergrößern und Verkleinern des Fensters können Sie auch das *Size Grip* in der rechten unteren Ecke des Fensters benutzen (Siehe **Abbildung 22**).

Weiterführende Informationen:

- **Volume > Resize** (2.3.5.3, Seite 44)

2.3.3.2 View > Options

In **Abbildung 24** ist die **Options** Symbolleiste blau markiert. Sie enthält alle wichtigen Einstellungen für das Windowing. Mehr über die Bedeutung der Einstellungen erfahren Sie in Kapitel 2.4, Seite 78. **File > Reset To Default Options** setzt die Optionen auf die Standardeinstellungen zurück. Mit **File > Load Options File** und **File > Save Options File As** können Sie die Optionen laden und speichern.

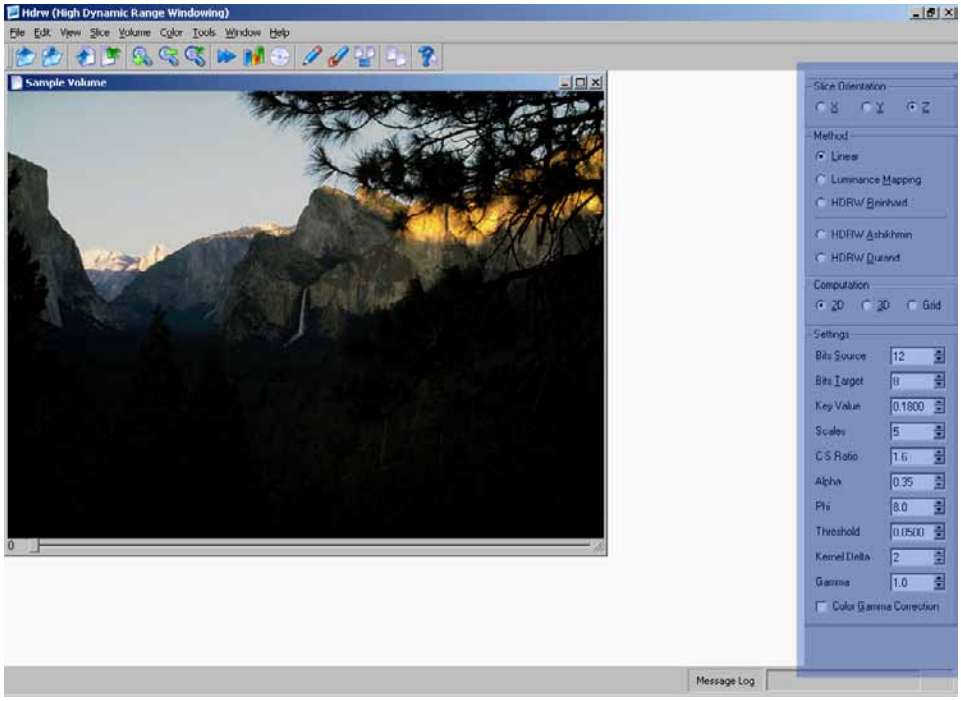

Abbildung 24: View > Options (Images\Schnaidt\ViewOptions.png)

Weiterführende Informationen:

- **File > Reset To Default Options** (2.3.1.6, Seite 22)
- **File > Open Options File** (2.3.1.7, Seite 23)
- **File > Save Options File As** (2.3.1.8, Seite 24)

2.3.3.3 View > Toolbar

Mit der **Toolbar** (Blau markiert in **Abbildung 25**) haben Sie schnellen Zugriff auf die wichtigsten Funktionen von Hdrw. Sie können zuerst auf und dann auf andere Icons in der Toolbar klicken, um den jeweiligen Hilfetext abzurufen.

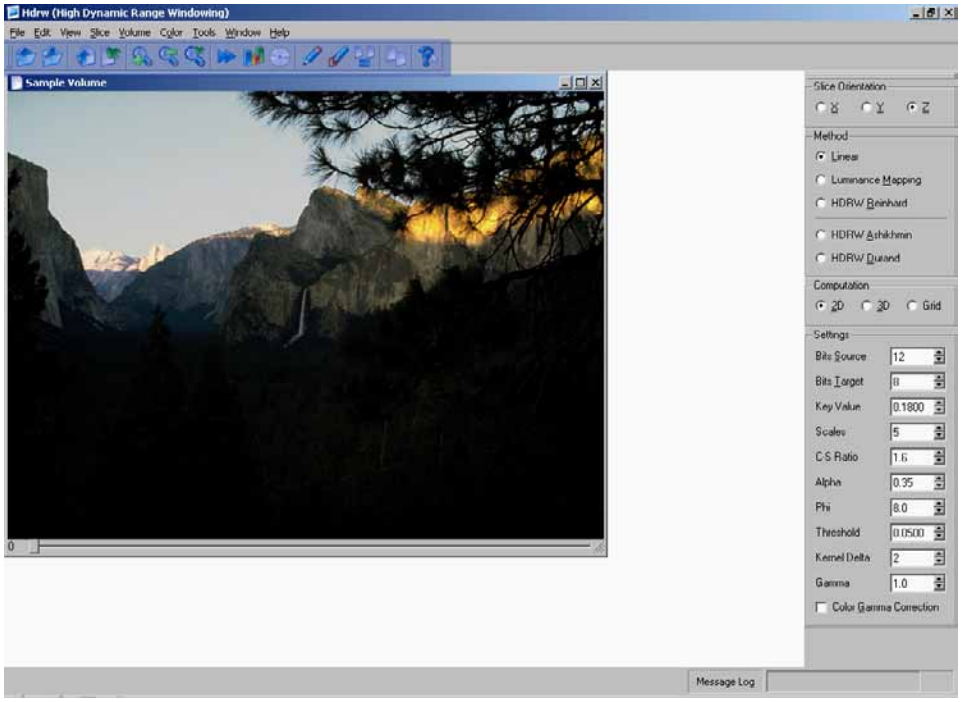

Abbildung 25: View > Toolbar (Images\Schnaidt\ViewToolbar.png)

Weiterführende Informationen:

- **Help > What's This** (2.3.9.3, Seite 76)

2.3.3.4 View > Message Log

In **Abbildung 26** sehen Sie unten das **Message Log** blau markiert. Drücken Sie die Taste **Message Log** in der Statusleiste am unteren Rand des Hdrw Fensters, um das Logbuch zu öffnen.

In ihm werden die letzten Meldungen gespeichert, die in der Statusleiste erscheinen. Hier können Sie nachschlagen, welche Operationen Sie als letztes durchgeführt und auch wie viel Zeit diese Operationen in Anspruch genommen haben.

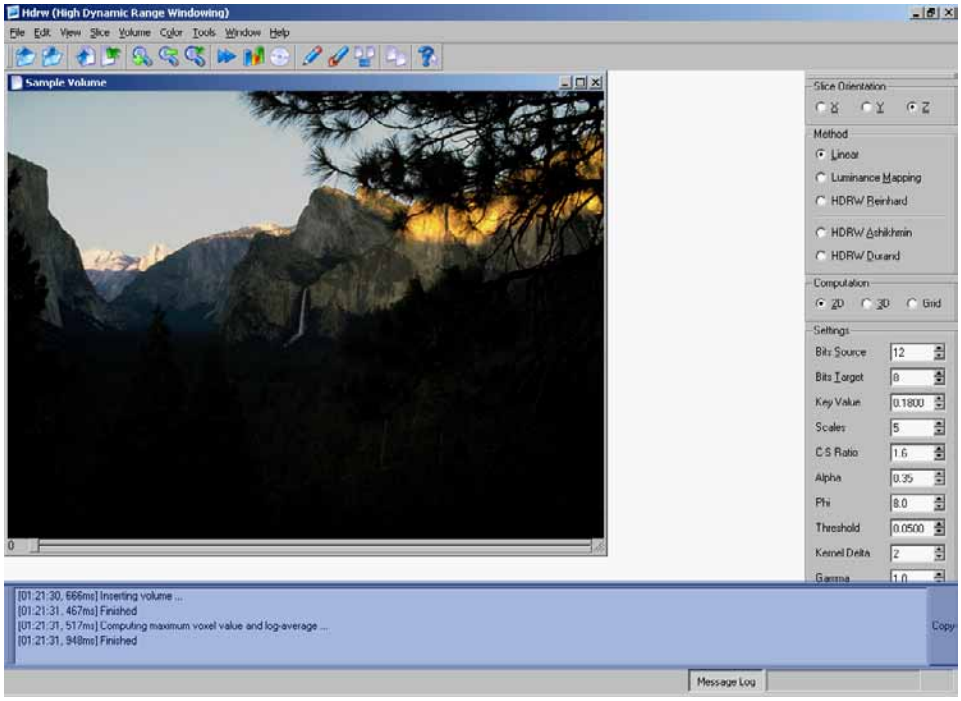

Abbildung 26: View > Message Log (Images\Schnaidt\ViewMessageLog.png)

Die Taste **Copy** (Rechts unten in **Abbildung 26**) kopiert den Inhalt des Logbuchs in die Zwischenablage.

2.3.4 Menü Slice

2.3.4.1 Slice > Add Slices From Volume

Wenn Sie ein anderes Volumen in ein Volumen einfügen möchten, können Sie diese Funktion verwenden.

Stellen Sie als erstes die aktuelle Schicht des ersten Volumens ein. Beginnend hinter dieser Schicht wird das zweite Volumen hinzufügt. Laden Sie dann das zweite Volumen aus einer Datei, wie **Abbildung 27** zeigt.

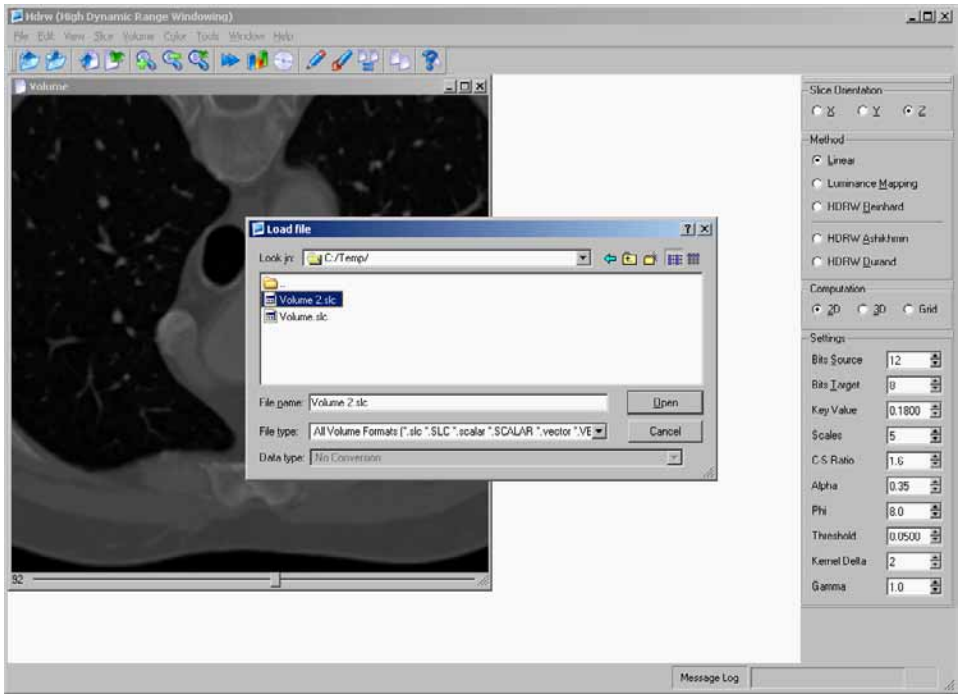

Abbildung 27: Slice > Add Slices From Volume (Images\Schnaidt\SliceAddSlicesFromVolume.png)

Am Besten funktioniert dies, wenn das erste und das zweite Volumen dieselbe Breite und Höhe haben. Ansonsten passt *Hdrw* die Größe des zweiten Volumens automatisch an.

Falls das eingefügte Volumen viel zu dunkel erscheint, haben die Volumen eine unterschiedliche Anzahl von Grauwerten. Wenn beispielsweise das Originalvolumen 4096 Grauwerte hat (Dies entspricht 12 Bit) und das eingefügte Volumen 1024 Grauwerte (10 Bit), dann wird das eingefügte Volumen nun zu dunkel angezeigt, da 1023 (Weiß im eingefügten Volumen) wesentlich geringer ist als 4095 (Weiß im Originalvolumen). In **Abbildung 28** können Sie hierfür ein Beispiel sehen. Erst nachdem **Bits Source** von 12 auf 10 Bit reduziert wurde (wie dies im Bild bereits der Fall), stimmt die Helligkeit des eingefügten Volumens, allerdings ist dann das Originalvolumen zu hell.
Falls Sie beide Volumen bei gleicher Helligkeit zusammenfügen wollen, können Sie das Histogramm des eingefügten Volumen zuvor strecken (Laden Sie dazu das Volumen zuerst separat und wählen Sie im Menü **Volume > Histogram**).

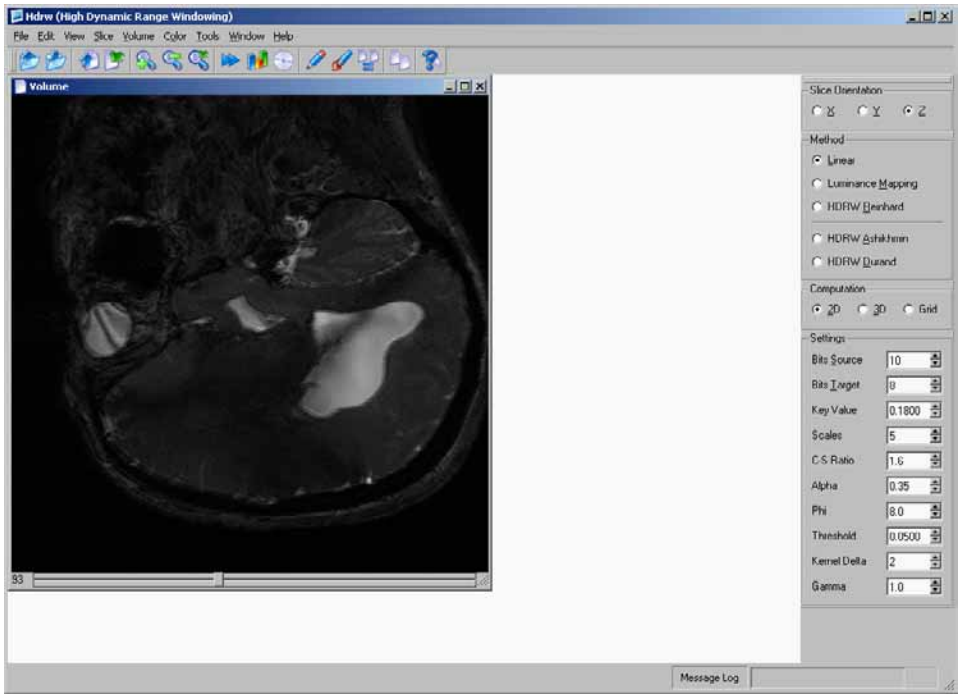

Abbildung 28: Slice > Add Slices From Volume (Images\Schnaidt\SliceAddSlicesFromVolume2.png)

Es stehen dieselben Volumenformate wie beim Öffnen von Volume mit **File > Open Volume** zur Verfügung.

Wichtige Anmerkungen:

- **Ändert das Volumen:** Diese Funktion ändert das Volumen des aktiven Fensters. Wenn Sie diese Funktion rückgängig machen wollen, speichern Sie das Volumen zuerst.

Weiterführende Informationen:

- **File > Open Volume** (2.3.1.1, Seite 15)
- **Volume > Histogram** (2.3.5.2, Seite 41)

2.3.4.2 Slice > Add Slices From Images

Neben einem ganzen Volumen kann man auch einzelne oder mehrere Bilddateien in ein Volumen einfügen. Wählen Sie auch hier zuerst die passende Schicht im geöffneten Volumen. Die Bilder werden dann direkt hinter dieser Schicht eingefügt.

Nachdem Sie diesen Dialog geöffnet haben, können Sie eine oder mehrere Bilddateien direkt auswählen, wie in **Abbildung 29** (Mit der Shift Taste lassen sich mehrere Dateien gleichzeitig markieren).

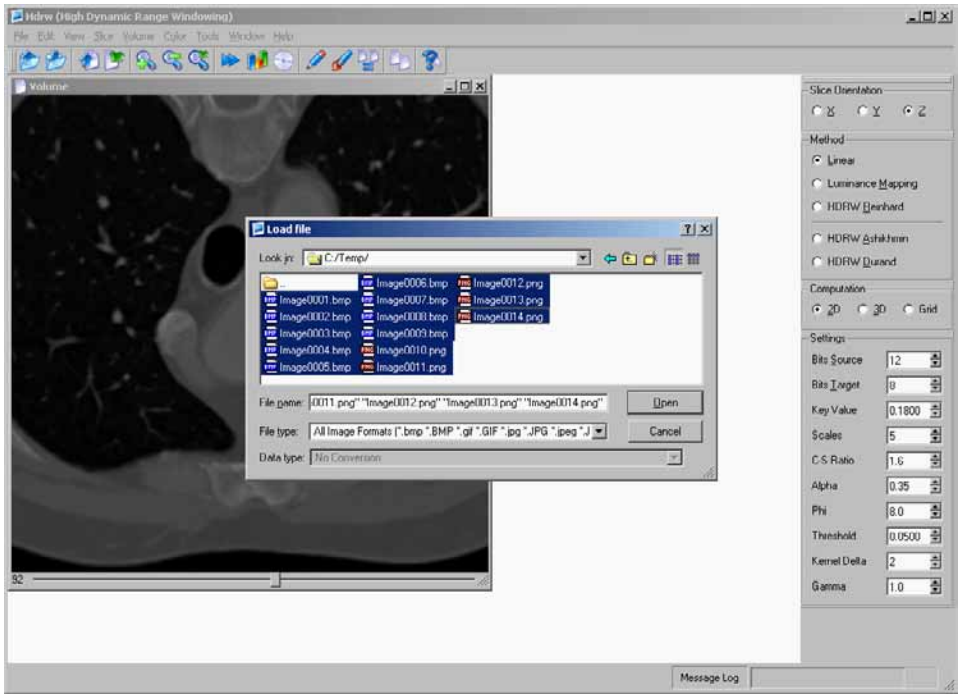

Abbildung 29: Slice > Add Slices From Images (Images\Schnaidt\SliceAddSlicesFromImages.png)

Da es sich um normale Bilder handelt, können diese maximal 256 verschiedene Helligkeitswerte haben (Dies entspricht 8 Bit). Dies gilt auch für Farbbilder im 24 Bit *RGB* Format.
Es kann dabei passieren, dass eingefügte Bilder in einem Volumen mit beispielsweise 4096 Grauwerten sehr dunkel erscheinen. Dies rührt daher, dass im Vergleich zum Grauwert 4095 (Weiß im Volumen) der Grauwert 255 (Weiß im eingefügten Bild) sehr dunkel ist. In diesem Fall kann es hilfreich sein, zuerst mit **Volume > Start Windowing** das Windowing für das Volumen durchzuführen und dann das Bild einzufügen, dann haben sowohl Volumen als auch das Bild 256 Grauwerte.

Es stehen dieselben Dateiformate wie beim Öffnen von Bildern mit **File > Open Images As Volume** zur Verfügung.

Wichtige Anmerkungen:

- **Ändert das Volumen:** Diese Funktion ändert das Volumen des aktiven Fensters. Wenn Sie diese Funktion rückgängig machen wollen, speichern Sie das Volumen zuerst.

Weiterführende Informationen:

- **File > Open Images As Volume** (2.3.1.3, Seite 18)
- 24 Bit *RGB* Format (2.3.1.3, Seite 18)

2.3.4.3 Slice > Move Slices

Mit dieser Funktion lassen sich Schichten des Volumens verschieben. Das Beispiel in **Abbildung 30** verschiebt die Schichten 92 bis 95 an die Schichtposition 110. Wie auch sonst im Programm beginnt dabei der Schichtindex bei 0.

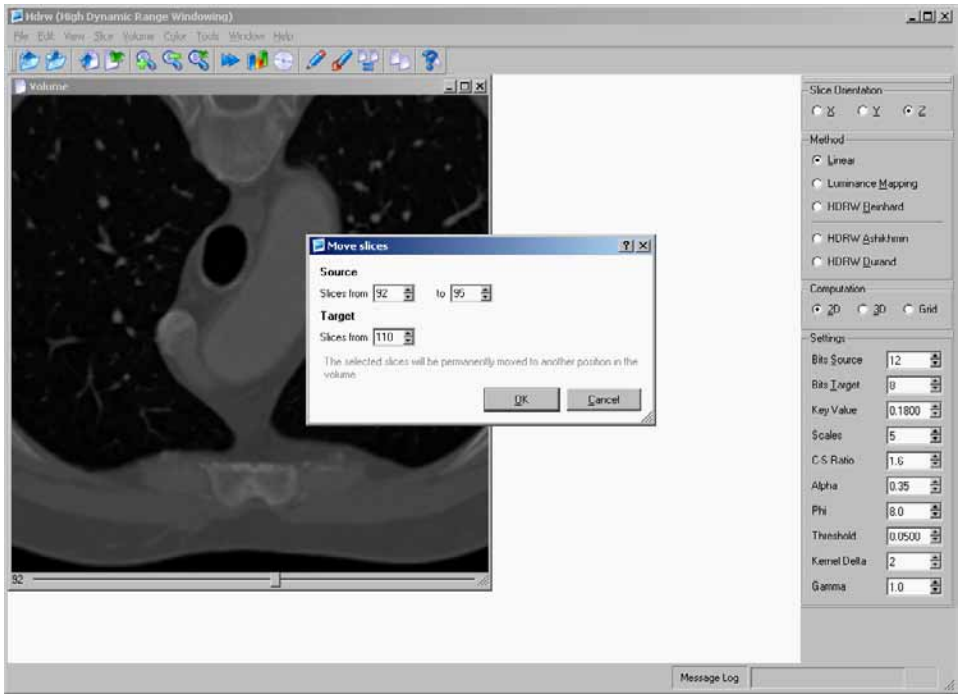

Abbildung 30: Slice > Move Slices (Images\Schnaidt\SliceMoveSlices.png)

Die Funktionsweise der Verschiebung lässt sich aber an einem einfacheren Beispiel besser erklären: Nehmen wir an, Sie haben ein Volumen mit 5 Schichten und verschieben Schicht 1 an Schichtposition 3, dann verändert sich das Volumen folgendermaßen:

- Vor der Verschiebung : 0-1-2-3-4 (Schicht 1 auf Schichtposition 1)
- Nach der Verschiebung : 0-2-3-1-4 (Schicht 1 auf Schichtposition 3)

Wichtige Anmerkungen:

- **Ändert das Volumen:** Diese Funktion ändert das Volumen des aktiven Fensters. Wenn Sie diese Funktion rückgängig machen wollen, speichern Sie das Volumen zuerst.

2.3.4.4 Slice > Delete Slices

Das Windowing des gesamten Volumens kann bei größeren Volumen oder auf langsameren Rechnern einige Zeit in Anspruch nehmen. In einigen Fällen wäre aber eigentlich kein Windowing des kompletten Volumens nötig. Dann können Sie vor dem Windowing nicht benötigte Schichten des Volumens einfach löschen. Öffnen Sie dazu den Dialog und geben Sie die zu löschenden Schichten an. In **Abbildung 31** werden beispielsweise die Schichten 92 bis 95 gelöscht. Wie auch sonst im Programm beginnt dabei der Schichtindex bei 0.

Abbildung 31: Slice > Delete Slices (Images\Schnaidt\SliceDeleteSlices.png)

Wichtige Anmerkungen:

- **Ändert das Volumen:** Diese Funktion ändert das Volumen des aktiven Fensters. Wenn Sie diese Funktion rückgängig machen wollen, speichern Sie das Volumen zuerst.

2.3.5 Menü Volume

2.3.5.1 Volume > Start Windowing

Dies ist die zentrale Funktion von *Hdrw* zum Windowing des gesamten Volumens, wie sie bereits in der Einführung erwähnt wurde.

Im Fenster sehen Sie immer die Vorschau des Windowing für die aktuelle Schicht des Volumens. Wenn Sie beispielsweise rechts in den Optionen **Hdrw Reinhard** einstellen, wird dies für die aktuelle Schicht direkt angezeigt. Für ein ganzes Volumen kann das Windowing aber mehr Zeit beanspruchen. Diese Funktion startet das Windowing für den vollen Datensatz.

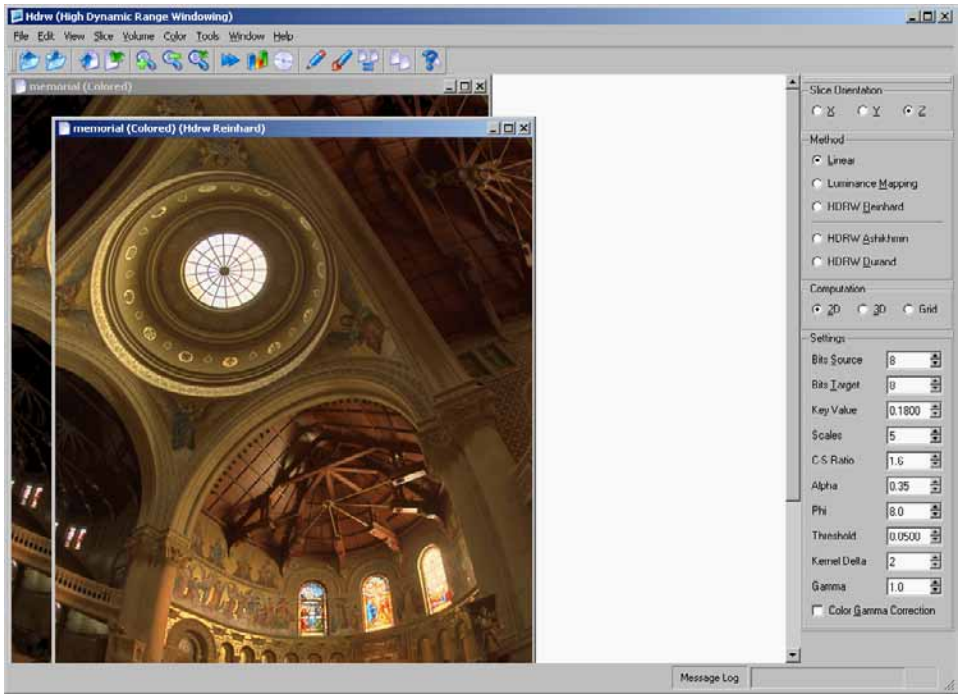

Abbildung 32: Volume > Start Windowing (Images\Schnaidt\VolumeStartWindowing.png)

Abbildung 32 zeigt hierfür ein Beispiel. Nach dem Windowing öffnet sich ein neues Fenster mit dem Resultat. Das neue Volumen hat nun lediglich noch 256 verschiedene Helligkeitswerte (wie sie an **Bits Source 8** erkennen können). Es lässt sich nun auf jedem normalen Monitor darstellen und kann so auch wieder als Volumendatei oder in Bilddateien gespeichert werden.

Im Titel des Fensters wird zum Dateiname noch der Name der Methode hinzugefügt, mit der das Windowing durchgeführt wurde (In **Abbildung 32** war dies **Hdrw Reinhard**).

Wichtige Anmerkungen:

- **Öffnet ein neues Fenster:** Das Ergebnis dieser Funktion wird in ein neues Fenster eingefügt und ändert nicht das aktuelle Fenster (bzw. Volumen).

Weiterführende Informationen:

- **File > Save Volume As** (2.3.1.2, Seite 17)
- **File > Save Volume As Images** (2.3.1.4, Seite 19)

2.3.5.2 Volume > Histogram

Das Histogramm ist ein typisches Mittel zur *Bildanalyse*. Mit ihm lassen sich beispielsweise wichtige Eigenschaften des Datensatzes wie Materialgrenzen analysieren (Grenzen zwischen zwei Objekten oder in medizinischen Datensätzen zwischen Organen).

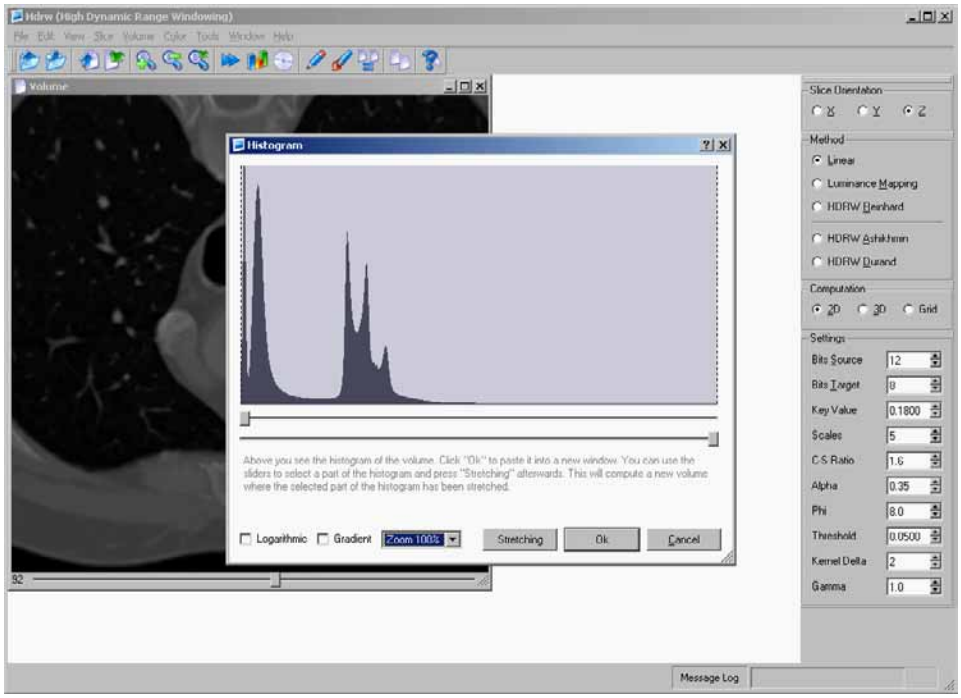

Abbildung 33: Volume > Histogram (Images\Schnaidt\VolumeHistogram.png)

Diese Funktion berechnet zuerst das Histogram und zeigt es dann wie in **Abbildung 33** an. Das Histogram ist im Prinzip ein mathematisches Schaubild, bei dem nach rechts der Grauwert und nach oben die Häufigkeit abgetragen werden. In diesem Fall hat das CT-Volumen Grauwerte von 0 bis 4095 und man sieht vereinzelte *Peaks* mit besonders häufigen Grauwerten.

In einigen Fällen sind Details im Histogramm besser erkennbar, wenn man sich dieses logarithmisch abtragen lässt (**Abbildung 34** links). Dies bedeutet, dass die Häufigkeit nach oben nicht mehr linear, sondern logarithmisch ansteigt. Dadurch werden gleichzeitig Schwankungen bei großen und bei sehr kleinen Häufigkeiten (wie rechts im Histogramm) sichtbar.

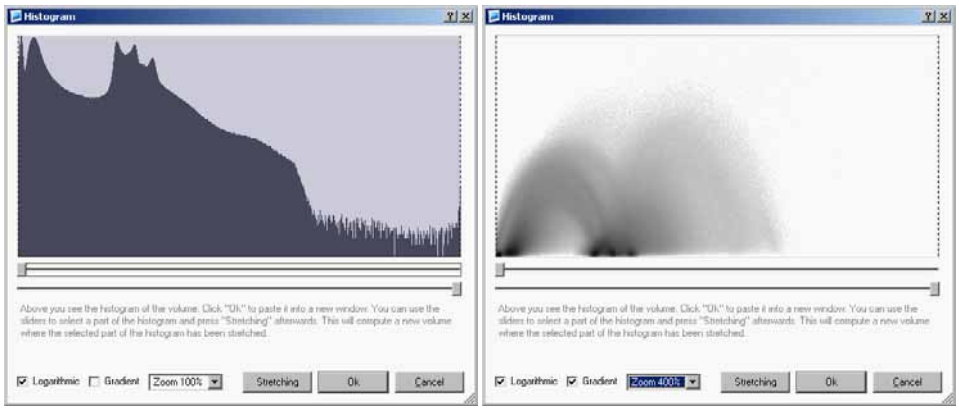

Abbildung 34: Volume > Histogram (Images\Schnaidt\VolumeHistogram1, 2.png)

Im normalen Histogramm deuten ansteigende und abfallende Flanken auf Materialgrenzen hin. Diese werden noch deutlicher im Gradientenhistogramm (**Abbildung 34** rechts). Nach rechts wird weiterhin der Grauwert abgetragen, nach oben aber nun der Gradient des Grauwertes. Die Häufigkeit eines Paares (Grauwert, Gradient) ist in der Helligkeit kodiert (Besonders dunkle Bereiche haben eine sehr hohe Häufigkeit).

Der Gradient entspricht der 3-dimensionalen Ableitung. Er beschreibt den Anstieg oder Abstieg des Grauwertes von einem Voxel im Volumen zu seinen Nachbarvoxeln. Große Helligkeitssprünge von einem Voxel zum nächsten Voxel liefern einen großen Gradienten. Dagegen haben eher gleichmäßige, homogene Bereiche einen geringen Gradienten. Die Maxima der Bögen (in **Abbildung 34** rechts) deuten daher auf große Helligkeitsunterschiede und auf Materialgrenzen hin. In diesem Fall sind dies die Trennlinien zwischen den verschiedenen Teilen der Lunge und dem schwarzen Hintergrund.

Da das Volumen nicht als mathematische Funktion vorliegt, muss die Ableitung bzw. der Gradient approximiert werden. Dies geschieht typischerweise über die zentralen Differenzen:

$$\begin{pmatrix} V_{x+1,y,z} - V_{x-1,y,z} \\ V_{x,y+1,z} - V_{x,y-1,z} \\ V_{x,y,z+1} - V_{x,y,z-1} \end{pmatrix} \quad (1)$$

Wobei $V_{x,y,z}$ in Formel (1) den Grauwert des Voxels an Raumposition (x,y,z) angibt.

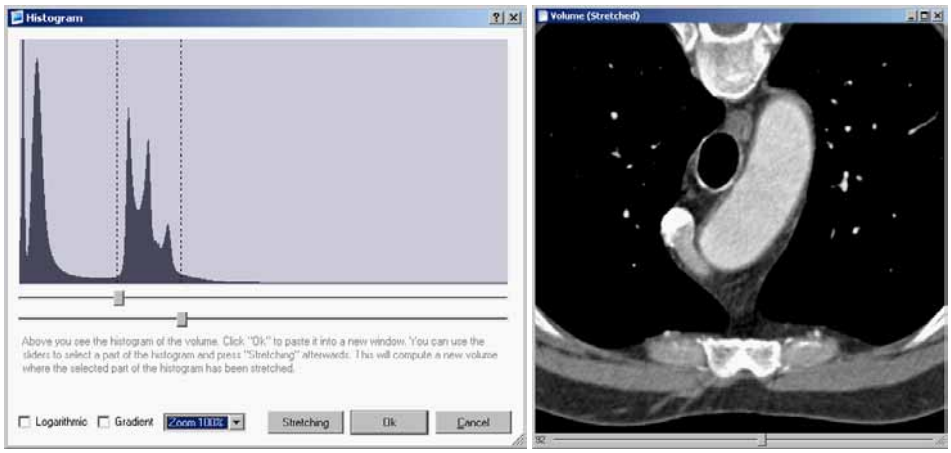

Abbildung 35: Volume > Histogram (Images\Schnaidt\VolumeHistogram3, 4.png)

In einigen Fällen ist es sinnvoll das Histogramm zu strecken, wie dies auch bereits in Kapitel 1.1, Seite 5 beschrieben wurde. Mit den beiden Schiebereglern in **Abbildung 35** können Sie einen minimalen und maximalen Grauwert angeben. Wenn Sie nun auf **Stretching** klicken, wird das Histogramm so gestreckt, dass der Bereich zwischen den beiden vertikalen Markierungen im Histogramm nun das gesamte Histogramm füllt (Ergebnis in **Abbildung 35** rechts).

Dies ist besonders dann nützlich, wenn der Histogrammbereich vom Datensatz nicht komplett ausgenutzt wird. Denn wenn Sie **Abbildung 33** und **Abbildung 35** vergleichen, fällt Ihnen sicherlich schnell auf, dass in einigen Bereichen nun mehr Details erkennbar sind (Das die Lunge umgebende Gewebe erscheint detailreicher). Allerdings werden alle Grauwerte unterhalb des angegebenen Minimalwerts auf Schwarz abgebildet und alle Grauwerte oberhalb des Maximalwerts auf

Weiß. Dadurch gehen in anderen Bereichen wieder viele Details verloren, gerade in der Lunge selbst.

Nutzt der Datensatz aber erst gar nicht den kompletten Bereich aus, kann oberhalb und unterhalb nichts verloren gehen, wie das Beispiel in **Abbildung 36** zeigt. Der eigentliche Maximalwert im Volumen nutzt nicht den kompletten Histogrammbereich aus. Wird das Volumen gestreckt, erscheint es dadurch heller und deutlicher (Ergebnis in **Abbildung 36** rechts).

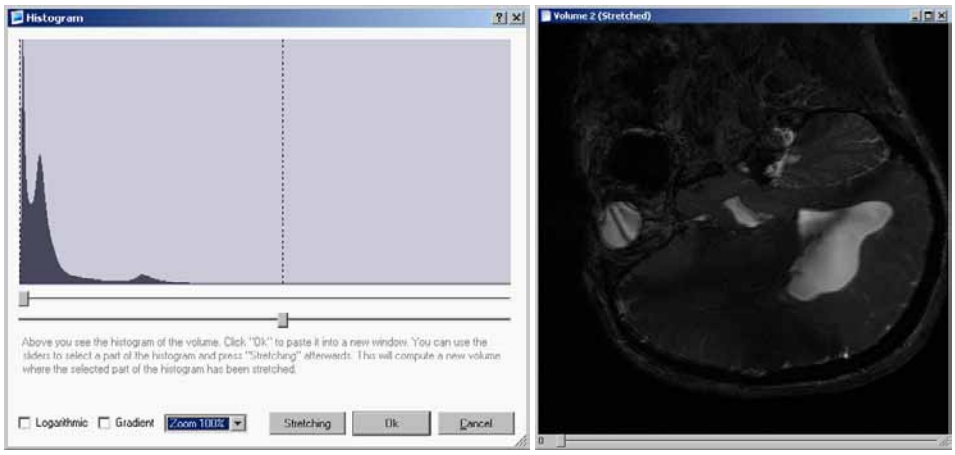

Abbildung 36: Volume > Histogram (Images\Schnaidt\VolumeHistogram5, 6.png)

Da das **Einstellen** des richtigen Minimal- und Maximalwertes schwierig sein kann, übernimmt dies *Hdrw*. Direkt nach dem Öffnen des Histogramms werden die Schieberegler korrekt eingestellt. Falls diese jeweils ganz links und ganz rechts sind, wie in **Abbildung 33**, dann benötigt das Volumen den kompletten Bereich.

Die Streckung auf die volle Breite des Histogramms geschieht dabei *linear* und ist im Normalfall nur dann sinnvoll bzw. nötig, wenn Sie als Windowing Methode **Linear** gewählt haben. Die anderen Windowing Methoden gleichen dies bereits selbstständig aus.

Die Breite des Histogramms, d. h. die Anzahl der angezeigten Grauwerte, lässt sich mit der Option **Bits Source** festlegen. Im obigen Beispiel 12 Bits, was 4096 Grauwerten entspricht. Im Normalfall ist dieser Wert aber passend eingestellt und braucht nicht verändert werden.

Wichtige Anmerkungen:

- **Arbeitet auf dem Volumen:** Diese Funktion arbeitet auf dem Volumen und nicht auf der Vorschau für die aktuelle Schicht, die Sie im aktiven Fenster sehen. Wenn Sie dies nach dem Windowing anwenden wollen, wählen Sie zuerst **Volume > Start Windowing** aus, um das Windowing für das ganze Volumen durchzuführen.

2.3.5.3 Volume > Resize

Falls Sie das Volumen permanent vergrößern oder verkleinern möchten, können Sie diese Funktion benutzen. Im Dialog (**Abbildung 37**) lässt sich die neue Größe angeben.

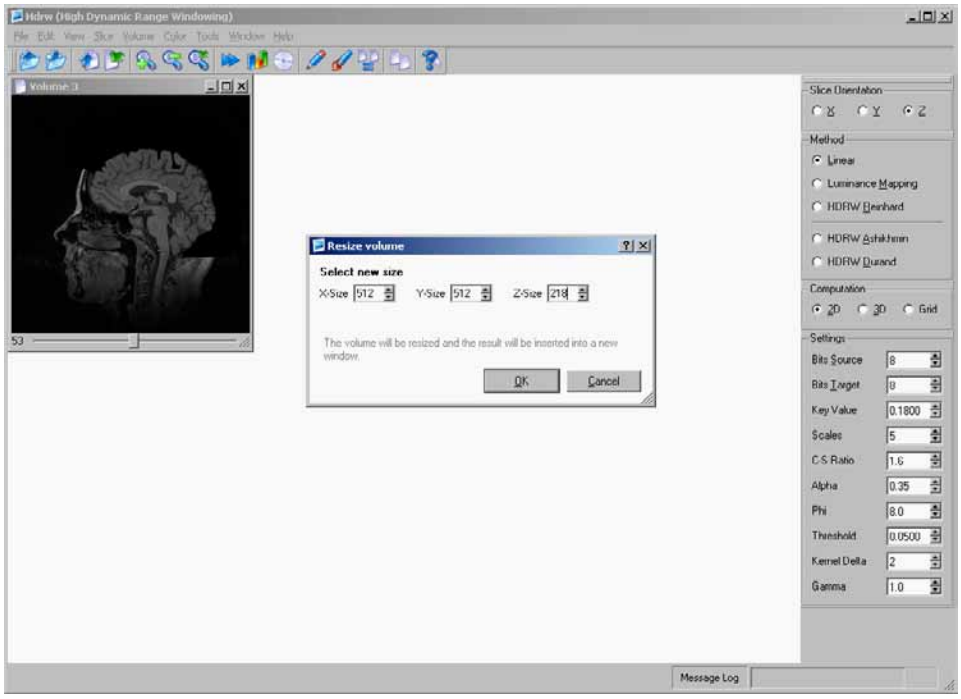

Abbildung 37: Volume > Resize (Images\Schnaidt\VolumeResize.png)

In diesem Fall wurde die Größe des Volumens verdoppelt (**Abbildung 38**).

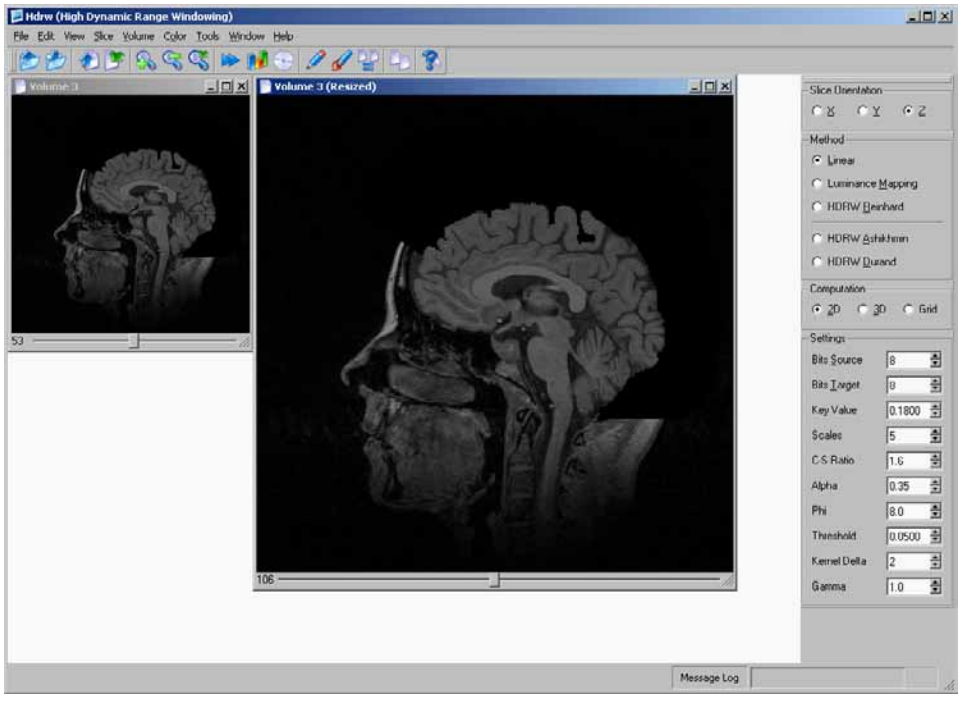

Abbildung 38: Volume > Resize (Images\Schnaidt\VolumeResize2.png)

Die Methode führt allerdings aus Geschwindigkeitsgründen kein Anti-Aliasing durch (weder Interpolation noch Subsampling), wodurch das vergrößerte Volumen blockig erscheinen kann.

Wichtige Anmerkungen:

- **Öffnet ein neues Fenster:** Das Ergebnis dieser Funktion wird in ein neues Fenster eingefügt und ändert nicht das aktuelle Fenster (bzw. Volumen).

2.3.5.4 Volume > Rotate Z-Axis

In einigen Fällen kann es nützlich sein, das Volumen zu drehen. Beispielsweise Patientendaten im *SLC* Format haben zum Teil eine bestimmte Orientierung, die aber in der *SLC* Datei nicht mit gespeichert wird (**Abbildung 39**).

Abbildung 39: Volume > Rotate Z-Axis (Images\Schnaidt\VolumeRotateZ-Axis.png)

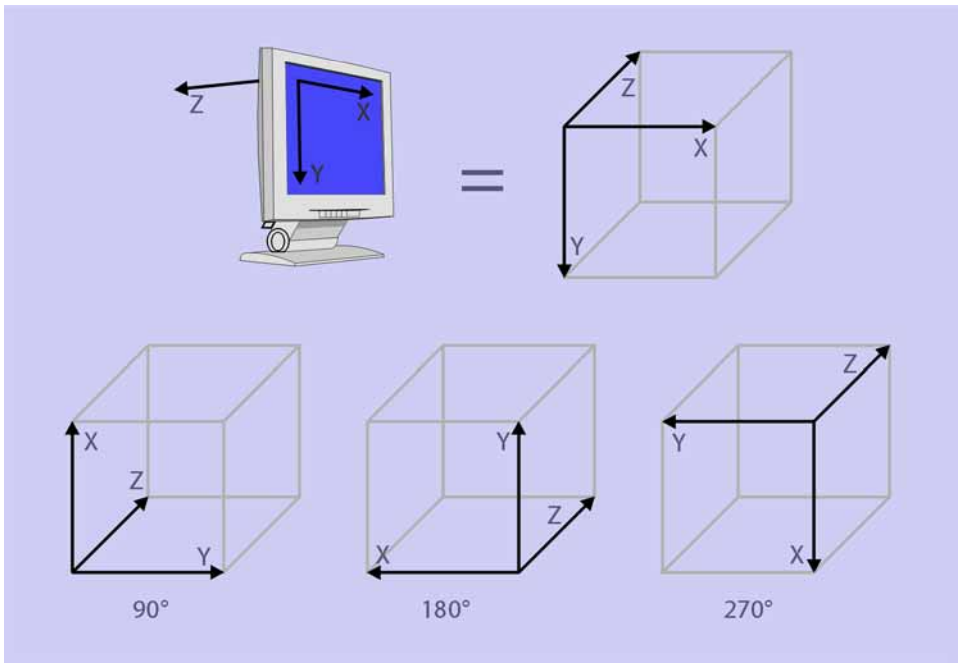

Abbildung 40: Volume > Rotate Z-Axis (Images\Schnaidt\VolumeRotateZ-Axis2.png)

Dabei stehen folgende Rotationen um die Z-Achse zur Auswahl ...

- **90° CCW:** Im Gegenuhrzeigersinn 90° um die Z-Achse drehen

- **180° CCW:** Im Gegenuhrzeigersinn 180° um die Z-Achse drehen
- **270° CCW:** Im Gegenuhrzeigersinn 270° um die Z-Achse drehen

Die Bedeutung wird in **Abbildung 40** dargestellt. Dabei wird das rechtshändige Koordinatensystem verwendet, das Ihr Monitor aufspannt (X-Achse nach rechts, Y-Achse nach unten, Z-Achse nach hinten).

Wichtige Anmerkungen:

- **Öffnet ein neues Fenster:** Das Ergebnis dieser Funktion wird in ein neues Fenster eingefügt und ändert nicht das aktuelle Fenster (bzw. Volumen).

Weiterführende Informationen:

- **File > Open Volume** (2.3.1.1, Seite 15)

2.3.5.5 Volume > Rotate Y-Axis

Mit dieser Funktion können Sie das Volumen um die Y-Achse drehen (Siehe auch 2.3.5.4, Seite 46). Dabei stehen folgende Rotationen zur Auswahl ...

- **90° CCW:** Im Gegenuhrzeigersinn 90° um die Y-Achse drehen
- **180° CCW:** Im Gegenuhrzeigersinn 180° um die Y-Achse drehen
- **270° CCW:** Im Gegenuhrzeigersinn 270° um die Y-Achse drehen

Die Bedeutung wird in **Abbildung 41** dargestellt.

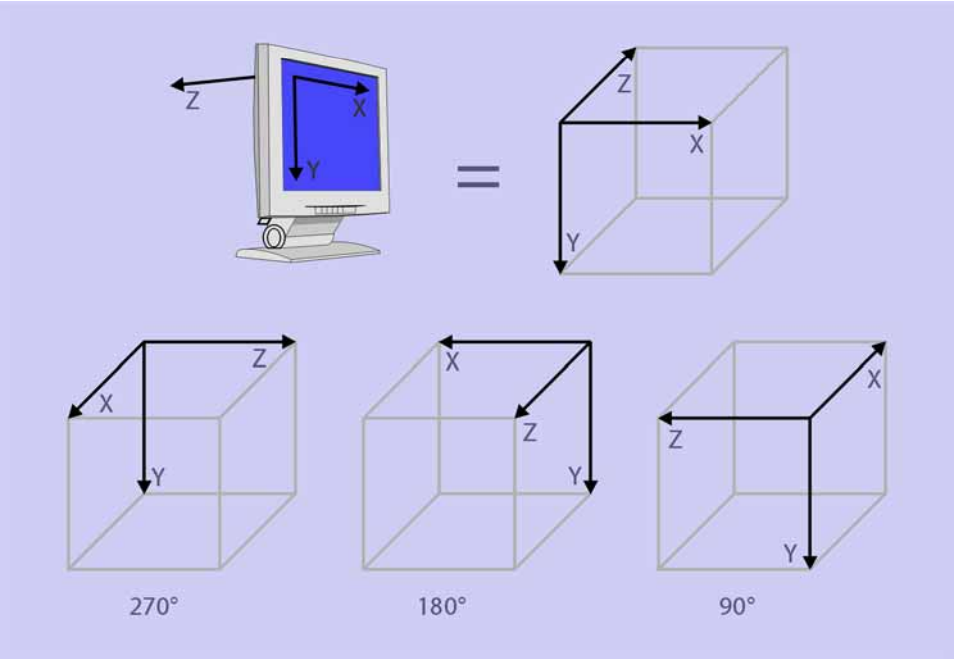

Abbildung 41: Volume > Rotate Y-Axis (Images\Schnaidt\VolumeRotateY-Axis.png)

Wichtige Anmerkungen:

- **Öffnet ein neues Fenster:** Das Ergebnis dieser Funktion wird in ein neues Fenster eingefügt und ändert nicht das aktuelle Fenster (bzw. Volumen).

Weiterführende Informationen:

- **Volume > Rotate Z-Axis** (2.3.5.4, Seite 46)

2.3.5.6 Volume > Rotate X-Axis

Mit dieser Funktion können Sie das Volumen um die X-Achse drehen (Siehe auch 2.3.5.4, Seite 46). Dabei stehen folgende Rotationen zur Auswahl ...

- **90° CCW:** Im Gegenuhrzeigersinn 90° um die X-Achse drehen
- **180° CCW:** Im Gegenuhrzeigersinn 180° um die X-Achse drehen
- **270° CCW:** Im Gegenuhrzeigersinn 270° um die X-Achse drehen

Die Bedeutung wird in **Abbildung 42** dargestellt.

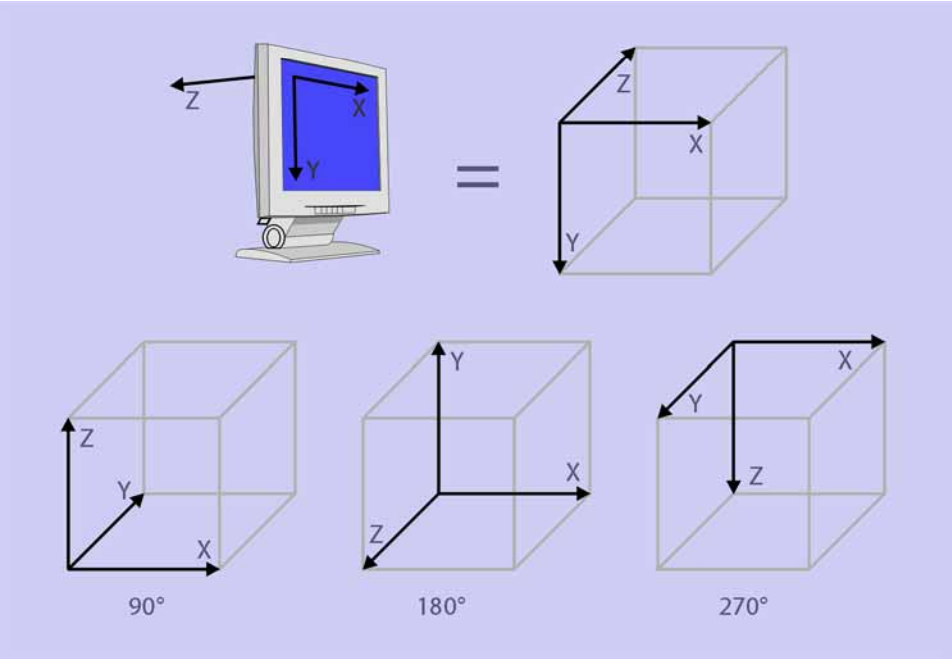

Abbildung 42: Volume > Rotate X-Axis (Images\Schnaidt\VolumeRotateX-Axis.png)

Wichtige Anmerkungen:

- **Öffnet ein neues Fenster:** Das Ergebnis dieser Funktion wird in ein neues Fenster eingefügt und ändert nicht das aktuelle Fenster (bzw. Volumen).

Weiterführende Informationen:

- **Volume > Rotate Z-Axis** (2.3.5.4, Seite 46)

2.3.5.7 Volume > Flip

Hiermit können Sie das Volumen entlang der Z-Achse, der Y-Achse oder der X-Achse wenden. Beispielsweise können Sie oben und unten vertauschen, falls das Bild auf dem Kopf steht (**Flip Y-Axis**).

Wie **Abbildung 43** zeigt, wird hierfür das rechtshändige Koordinatensystem benutzt, das Ihr Monitor aufspannt (X-Achse nach rechts, Y-Achse nach unten, Z-Achse nach hinten).

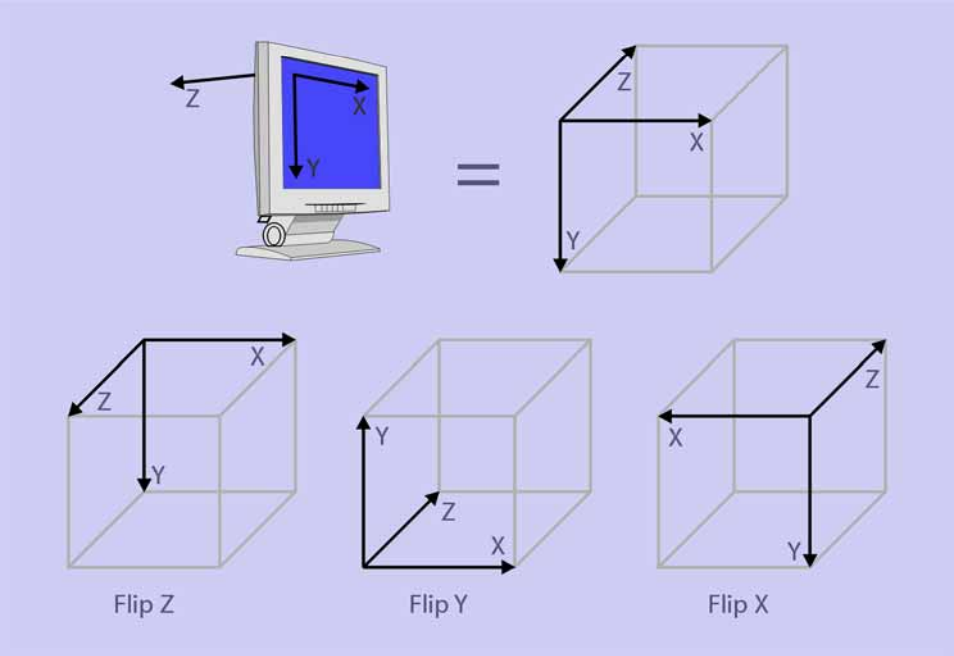

Abbildung 43: Volume > Flip (Images\Schnaidt\VolumeFlip.png)

Wichtige Anmerkungen:

- **Öffnet ein neues Fenster:** Das Ergebnis dieser Funktion wird in ein neues Fenster eingefügt und ändert nicht das aktuelle Fenster (bzw. Volumen).

Weiterführende Informationen:

- **Volume > Rotate Z-Axis** (2.3.5.4, Seite 46)

2.3.5.8 Volume > Volume Information

Hdrw kann viele verschiedene Arten von Volumen laden und verwalten. Um mehr Informationen über das Volumen des aktiven Fensters zu erhalten, steht diese Funktion zur Verfügung. **Abbildung 44** enthält hierfür ein Beispiel.

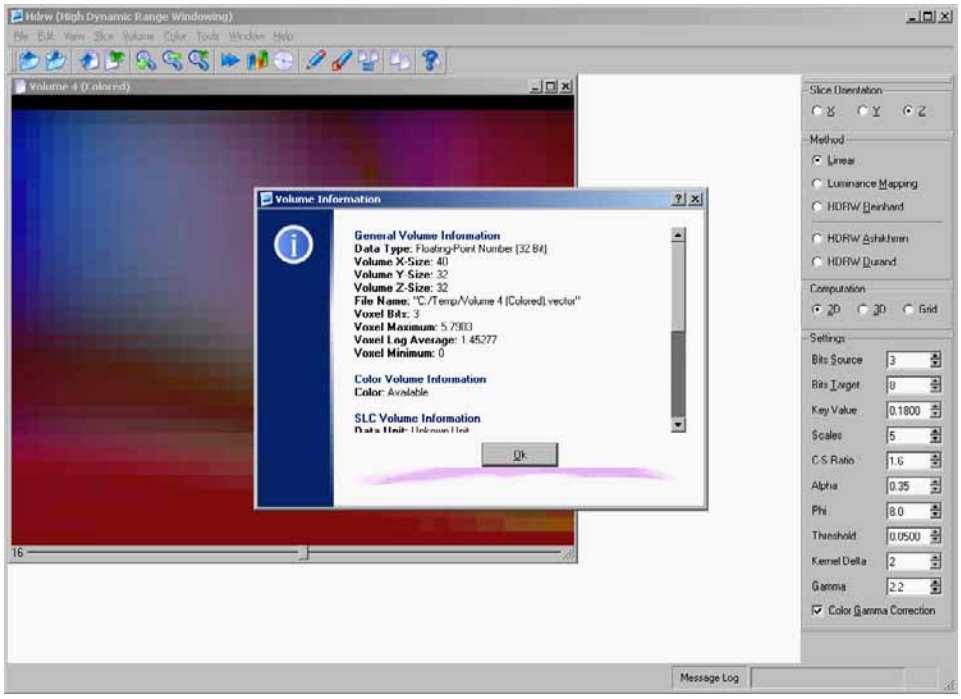

Abbildung 44: Volume > Volume Information (Images\Schnaidt\VolumeVolumeInformation.png)

Es werden folgende Informationen angezeigt ...

- **General Volume Information:** Allgemeine Informationen
  - **Data Type:** Gibt den Datentyp des Volumens an, d. h. **Integer** (dt. Ganzzahl) oder **Floating-Point Number** (dt. Gleitkommazahl) und die Anzahl der Bits (8, 16, 32, 64 Bit).
  - **Volume X-Size:** Ausdehnung des Volumens in X-Richtung.
  - **Volume Y-Size:** Ausdehnung des Volumens in Y-Richtung.
  - **Volume Z-Size:** Ausdehnung des Volumens in Z-Richtung.
  - **File Name:** Datei, aus der das Volumen geladen wurde.
  - **Voxel Bits:** Anzahl der benutzten Bits pro Voxel. Beispielsweise bei einem 8 Bit Integer Volumen sind eventuell nur 7 Bits benutzt. Floating Point Volumen werden hierzu als Integer Volumen betrachtet, d. h. der Grauwert 255.234 wird zu 255 gerundet und benötigt 8 Bits.
  - **Voxel Maximum:** Dies ist der maximale Grau- bzw. Helligkeitswert, der im Volumen vorkommt.
  - **Voxel Log Average:** Dies ist der durchschnittliche Grau- bzw. Helligkeitswert, der im Volumen vorkommt (Genauer handelt es sich hierbei um das geometrische Mittel, das in Kapitel 3, Seite 99 definiert wird).

    - **Voxel Minimum:** Dies ist der minimale Grau- bzw. Helligkeitswert, der im Volumen vorkommt.

- **Color Volume Information:** Informationen bezüglich Farbe
    - **Color:** Farbinformation ist entweder **Available** (dt. verfügbar) oder **Not Available** (dt. nicht verfügbar). Das Bild wird automatisch in Farbe angezeigt, falls möglich.
- **SLC Volume Information:** Information für das *SLC* Format
    - **Data Unit:** Die Maßeinheit des Volumens (Zum Beispiel **Meter**).
    - **Data Source:** Der Ursprung der Volumens (Zum Beispiel **CT Data**).
    - **Data Transformation:** Die Transformation, die auf das Volumen angewendet wurde (Zum Beispiel **Resampled Data**).
    - **Data Compression:** Die verwendete Kompressionsart (Dieser Wert ist grundsätzlich **No Compression**).
    - **Volume X-Spacing:** Der Abstand zwischen zwei benachbarten Voxeln in X-Richtung.
    - **Volume Y-Spacing:** Der Abstand zwischen zwei benachbarten Voxeln in Y-Richtung.
    - **Volume Z-Spacing:** Der Abstand zwischen zwei benachbarten Voxeln in Z-Richtung.
    - **Bits Per Voxel:** Die Anzahl der Bits pro Voxel. Dies ist ein Wert speziell für das *SLC* Format. Die wirklich verwendeten Bits pro Voxel finden Sie weiter oben unter **Voxel Bits**.
- **SCALAR Volume Information:** Informationen für das *SCALAR* Format
    - **Grid:** Gibt an, ob das Gitter **Available** oder **Not Available** ist. Die Voxel des Volumens sind auf diesem Gitter angeordnet. Falls kein Gitter verfügbar ist, wird ein normales kartesisches Gitter verwendet (Wobei **Volume X-Spacing**, **Y-Spacing**, **Z-Spacing** das kartesische Gitter mit würfelförmigen Zellen zu einem uniformen Gitter mit quaderförmigen Zellen verzerren können).
    - **Average Grid Unit:** Die Koordinaten der Gitterpunkte sind in wirklichen, physikalischen Koordinaten gegeben. Diese können sich in der Größe stark vom Koordinatensystem unterscheiden, das Sie am Bildschirm sehen. Zusätzlich können die physikalischen Koordinaten auch ein gekrümmtes (curvilineares) Gitter ergeben. Dieser Wert gibt Ihnen sozusagen die durchschnittliche Längeneinheit von einem Voxel zum nächsten im physikalischen Koordinatensystem an. Ein in Wirklichkeit sehr kleines Volumen hat hier auch einen sehr kleinen Wert.
- **VECTOR Volume Information:** Information für das *VECTOR* Format
    - **Vector:** Gibt an, ob pro Gitterpunkt ein ganzer Vektor **Available** oder **Not Available** ist. Ein Vektor besteht aus 3 einzelnen Werten. Im aktiven Fenster sehen Sie in diesem Fall den Betrag den Vektors.

Weiterführende Informationen:

- **File > Open Volume** (2.3.1.1, Seite 15)

2.3.5.9 Volume > Convert Data Type

Jedes Volumen liegt im Speicher in einem bestimmten Datentyp vor. Diesen können Sie bereits beim Laden des Volumens angeben. Normalerweise wird aber der Datentyp benutzt, in dem die Datei vorlag. Mit dieser Funktion können Sie den Datentyp auch noch im Nachhinein konvertieren.

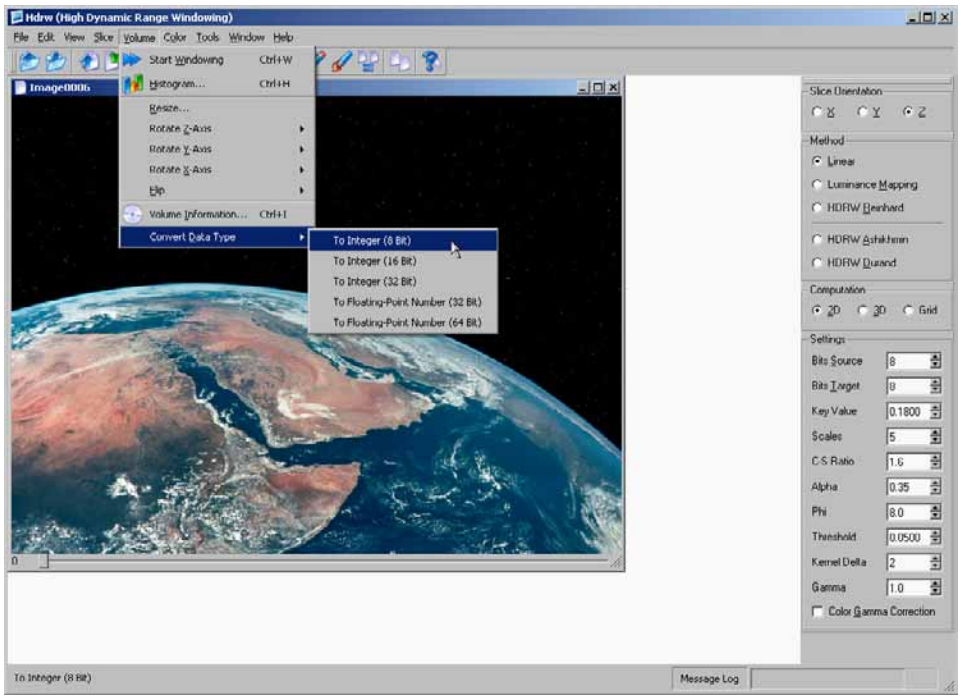

Abbildung 45: Volume > Convert Data Type (Images\Schnaidt\VolumeConvertDataType.png)

Abbildung 45 zeigt die verschiedenen Datentypen, die dabei verfügbar sind. Nochmals genauer:

- **Integer (8 Bit):** 8 Bit Ganzzahl, d.h. maximal 256 verschiedene Grauwerte
- **Integer (16 Bit):** 16 Bit Ganzzahl, d.h. maximal 2^{16} verschiedene Grauwerte
- **Integer (32 Bit):** 32 Bit Ganzzahl, d.h. maximal 2^{32} verschiedene Grauwerte
- **Floating-Point Number (32 Bit):** 32 Bit Gleitkommazahl
- **Floating-Point Number (64 Bit):** 64 Bit Gleitkommazahl

Von oben nach unten benötigen die Datentypen mehr Speicher, liefern aber auch eine bessere Genauigkeit. Im Normalfall müssen Sie den Datentyp des Volumens nicht konvertieren. Nur falls Artefakte auftauchen sollten, lassen sich diese eventuell mit **Floating-Point Number (64 Bit)** beseitigen (Dies ist der Datentyp mit der höchsten Genauigkeit).

Wichtige Anmerkungen:

- **Öffnet ein neues Fenster:** Das Ergebnis dieser Funktion wird in ein neues Fenster eingefügt und ändert nicht das aktuelle Fenster (bzw. Volumen).

Weiterführende Informationen:

- **File > Open Volume** (2.3.1.1, Seite 15)

2.3.6 Menü Color

2.3.6.1 Color > Convert Vector To Color

Volumen im *VECTOR* Format enthalten pro Gitterpunkt einen Vektor aus 3 Werten. In einigen Fällen macht es Sinn, diesen Vektor als *RGB* Wert zu interpretieren (3 Werte für Rot, Grün und Blau). Diese Funktion wandelt die Vektoren in Farben um.

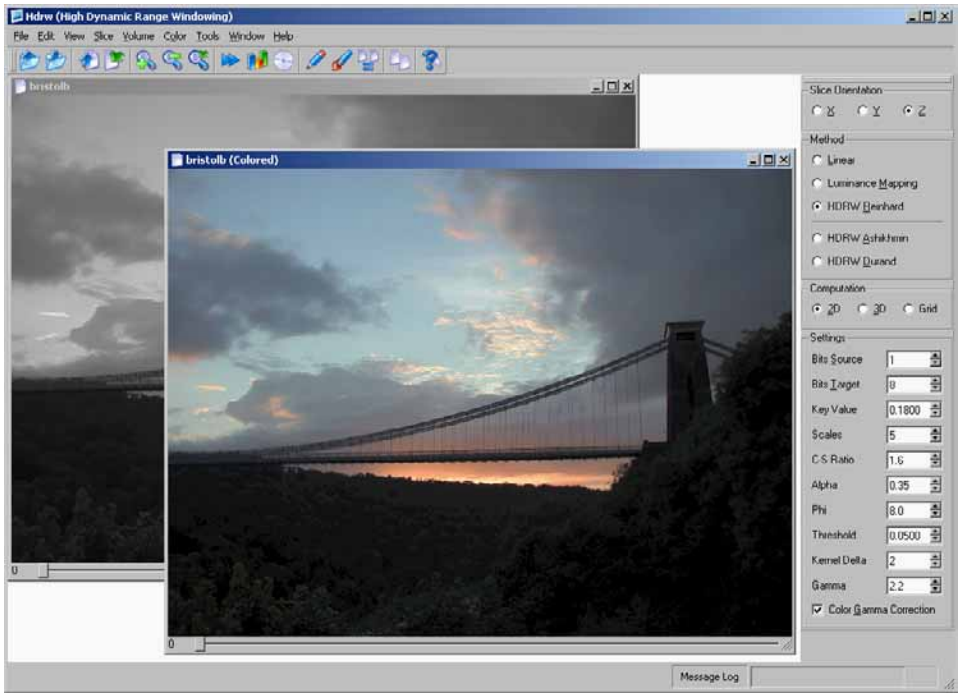

Abbildung 46: Color > Convert Vector To Color (Images\Schnaidt\ColorConvertVectorToColor.png)

Im Beispiel in **Abbildung 46** sehen Sie im Hintergrund das Originalvolumen. Von den Vektoren wird hier einfach der Betrag angezeigt, was in diesem Fall wie ein Graustufenbild aussieht. Nachdem die Funktion das Volumen konvertiert hat, wurden alle Vektoren als *RGB* Werte interpretiert und in Farbinformation umgewandelt, wie man im Vordergrund sehen kann.

Wichtige Anmerkungen:

- **Arbeitet auf dem Volumen:** Diese Funktion arbeitet auf dem Volumen und nicht auf der Vorschau für die aktuelle Schicht, die Sie im aktiven Fenster sehen. Wenn Sie dies nach dem Windowing anwenden wollen, wählen Sie zuerst **Volume > Start Windowing** aus, um das Windowing für das ganze Volumen durchzuführen.

- **Öffnet ein neues Fenster:** Das Ergebnis dieser Funktion wird in ein neues Fenster eingefügt und ändert nicht das aktuelle Fenster (bzw. Volumen).

Weiterführende Informationen:

- **File > Open Volume** (2.3.1.1, Seite 15)

- **Volume > Volume Information** (2.3.5.8, Seite 51)

2.3.6.2 Color > Color Space

Daten aus Bilddateien liegen meist im *RGB* Format vor. Das bedeutet, pro Voxel ist ein Rot, Grün und Blau Wert definiert, der die Farbe des Voxels angibt. Die in *Hdrw* implementierten Windowing Methoden arbeiten aber auf Grau- bzw. Helligkeitswerten. Daher werden Farbbilder von *Hdrw* in einen anderen Farbraum umgerechnet, in dem dies möglich ist. Es stehen der *HSV* und der *Yxy* Farbraum zur Verfügung.

In **Abbildung 47** sehen Sie links ein Bild mit dem *HSV* und rechts mit dem *Yxy* Farbraum. In einigen Fällen wirken die Bilder im *HSV* Farbraum etwas blasser. Da der *Yxy* Farbraum aber aufwendiger zu berechnen ist, benutzt *Hdrw* standardmäßig *HSV*.

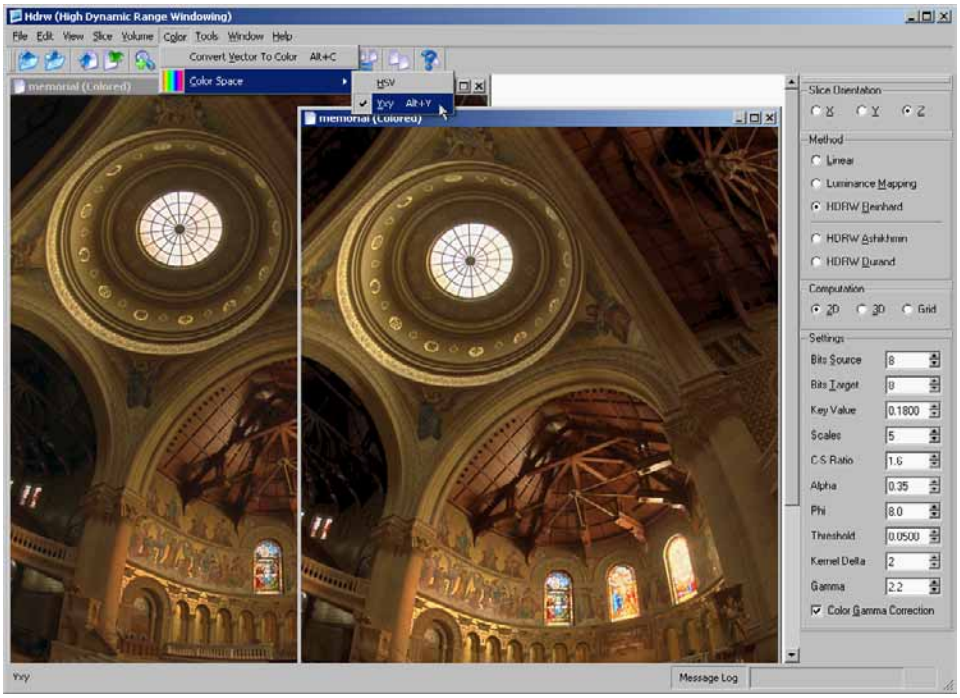

Abbildung 47: Color > Color Space (Images\Schnaidt\ColorColorSpace.png)

Farbvolumen lassen sich direkt laden (**File > Open Images As Volume**) oder aus einem Volumen im *VECTOR* Format erzeugen (**Color > Convert Vector To Color**). Die Umrechnung vom *RGB* Farbraum findet beim Laden bzw. Erzeugen des Farbvolumens statt.
Wird das Volumen bereits im *HSV* Farbraum angezeigt und wechseln Sie danach in den *Yxy* Farbraum, werden die Farben falsch angezeigt, bis Sie wieder zum *HSV* Farbraum zurückwechseln.

HSV Farbraum:

Im *HSV* Farbraum werden Farben durch *Hue* (Farbton), *Saturation* (Farbsättigung) und *Value* (Helligkeit) beschrieben. Das eigentliche Windowing arbeitet auf den Helligkeitswerten (V). Farbton und -sättigung bleiben unverändert (HS).

Die Umrechnung vom *RGB* Farbraum in den *HSV* Farbraum und zurück geschieht mit einem speziellen Algorithmus. Dieser ist für verschiedene Datentypen auf maximale Geschwindigkeit optimiert (Details zu diesem Algorithmus finden Sie im Source Code in *HdrwTools.cpp*).

Yxy Farbraum:

Von der CIE (Commission Internationale d'Éclairage) wurde 1932 ein spezielles Farbsystem entworfen, das aufwendig auf Basis von Versuchen standardisiert wurde und dabei Schwächen anderer Farbsysteme ausgleicht. Dieses Farbsystem wird mit *XYZ* bezeichnet, wobei Y für die Helligkeitswert steht (genauer *photopische spektrale Empfindlichkeit*), X etwa für den rot-grün Unterschied und Z für den blau-gelb Unterschied.

Alternativ zum *XYZ* Farbsystem kann man das *Yxy* Farbsystem verwenden, in dem x und y normiert werden ...

$$x = \frac{X}{X+Y+Z}, \quad y = \frac{Y}{X+Y+Z} \tag{2}$$

Y beschreibt weiterhin den Helligkeitswert und wird, wie beim *HSV* Farbraum, für das Windowing verwendet (x und y bleiben unverändert).

Die Umrechnung vom *RGB* Farbraum in den *XYZ* Farbraum und zurück geschieht mit Hilfe zweier Transformationsmatrizen, die von Reinhard *et al.* (2002, p.267-276) übernommen wurden.

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 0.5141364 & 0.32387860 & 0.16036376 \\ 0.2650680 & 0.67023428 & 0.06409157 \\ 0.0241188 & 0.12281780 & 0.84442666 \end{pmatrix} \cdot \begin{pmatrix} R \\ G \\ B \end{pmatrix} \tag{3}$$

$$\begin{pmatrix} R \\ G \\ B \end{pmatrix} = \begin{pmatrix} 2.5651 & -1.1665 & -0.3986 \\ -1.0217 & 1.9777 & 0.0439 \\ 0.0753 & -0.2543 & 1.1892 \end{pmatrix} \cdot \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \tag{4}$$

Weiterführende Informationen:

- **File > Open Images As Volume** (2.3.1.3, Seite 18)
- **Volume > Volume Information** (2.3.5.8, Seite 51)
- **Color > Convert Vector To Color** (2.3.6.1, Seite 55)

2.3.7 Menü Tools

2.3.7.1 Tools > Region Growing Segmentation

Segmentierung wird besonders auf medizinischen Volumen oft angewendet. Ein typisches Beispiel wäre die Segmentierung eines Organs im menschlichen Körper. Allgemein ist das Ziel der Segmentierung, bestimmte Strukturen bzw. Objekte in einem Datensatz zu identifizieren, d.h. vom Rest des Datensatzes zu unterscheiden. Oft spricht man hier auch davon, dass dem Datensatz eine Semantik zugewiesen wird, da nach der Segmentierung bestimmte Bereiche im Datensatz beispielsweise als Herz, als Lunge oder als Knochen gekennzeichnet sind.

Es gibt viele verschiedene Methoden für die Segmentierung, da die Segmentierung nicht nur ein typisches, sondern auch ein algorithmisch kompliziertes Problem ist. Kaum eine Methode wird auf allen Datensätzen zum Ziel führen, daher werden typischerweise verschiedene Methoden ausprobiert und oft auch kombiniert.

Dennoch gibt es eine Methode, die fast immer einsetzbar ist und, in Kombination mit anderen Methoden, auch eingesetzt wird: Das *Region Growing* (dt. Regionenwachstum).

Die Idee besteht darin, von einem *Seed Point* (dt. Saatpunkt) aus zu starten, der durch einen Mausklick vom Benutzer angegeben wird. Er ist der Beginn der Segmentierung. Dann werden die Voxel untersucht, die direkt zu diesem Saatpunkt benachbart sind. Sind sie "ähnlich" zum Saatpunkt, gehören diese Voxel auch zur Segmentierung. Im nächsten Schritt werden die gerade neu hinzugekommenen Voxel genommen und deren Nachbarn untersucht. Dies wird solange fortgesetzt, bis keine neuen Voxel mehr hinzukommen und (mit etwas Glück) das gewünschte Objekt segmentiert ist.

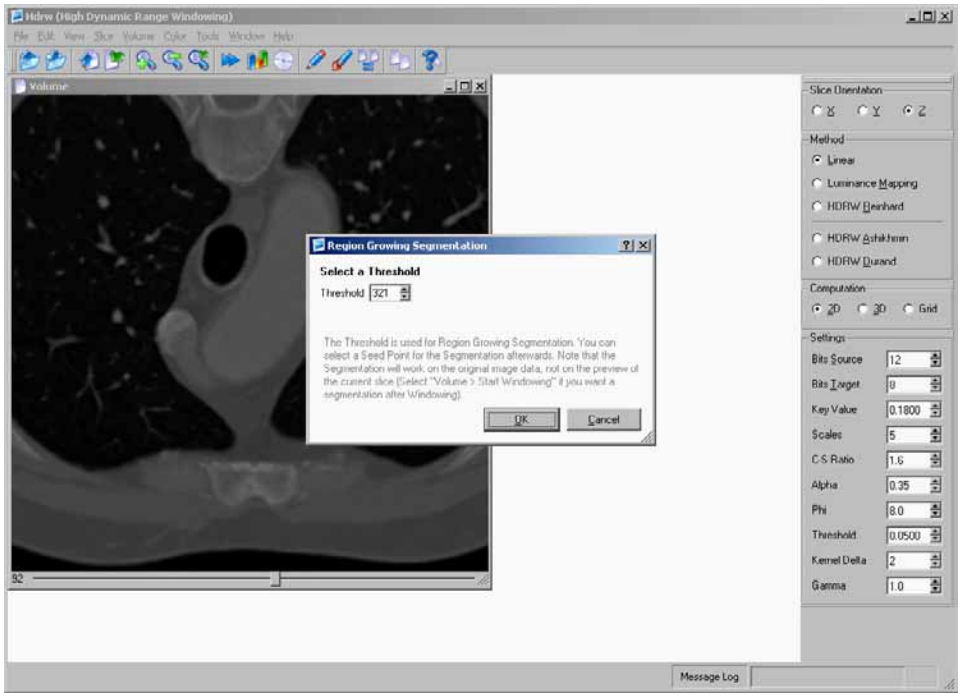

Abbildung 48: Tools > Region Growing (Images\Schnaidt\ToolsRegionGrowingSegmentation.png)

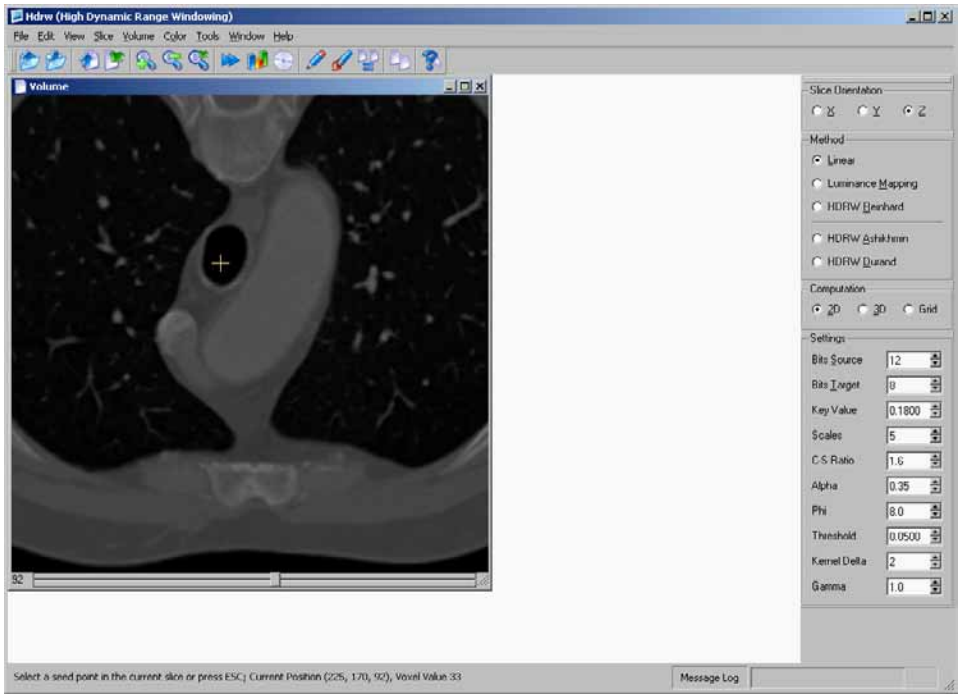

Abbildung 49: Tools > Region Growing (Images\Schnaidt\ToolsRegionGrowingSegmentation2.png)

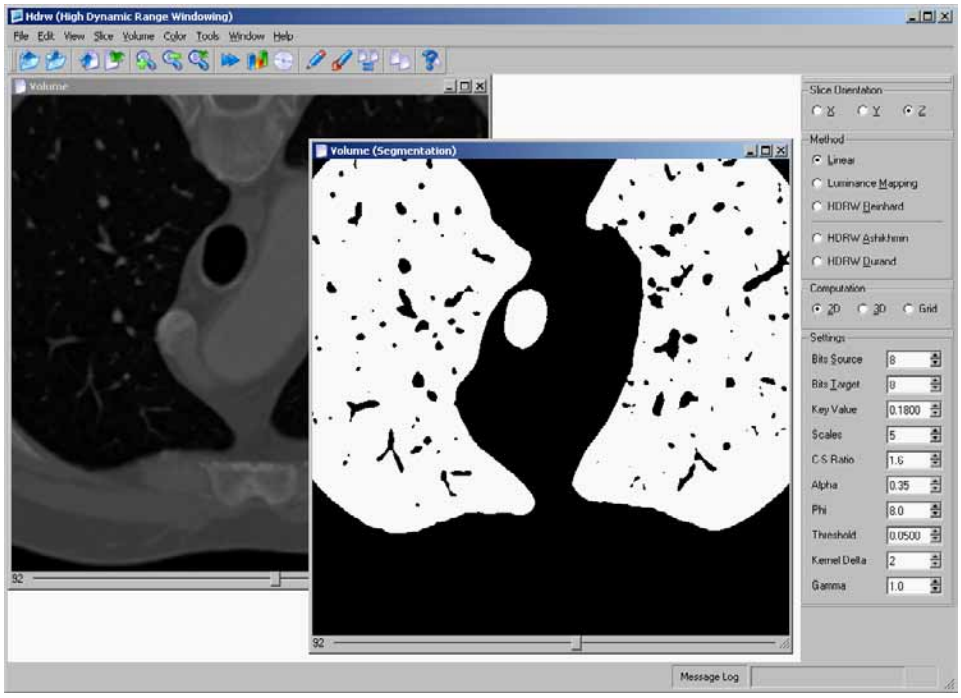

Abbildung 50: Tools > Region Growing (Images\Schnaidt\ToolsRegionGrowingSegmentation3.png)

Die Segmentierung läuft in 3 Schritten ab:

- Öffnen Sie zuerst den Dialog und geben Sie den *Threshold* (dt. Schwellwert) an (**Abbildung 48**). Je höher der Schwellwert ist, desto mehr Voxel werden als "ähnlich" betrachtet.
- Wählen Sie nun den Seed Point (**Abbildung 49**).
- Nun wird die Segmentierung berechnet und danach angezeigt. Alle Voxel mit Grauwert 255 (Weiß in **Abbildung 50**) gehören zur Segmentierung.

Im Beispiel wurden die luftgefüllten (schwarzen) Teile der Lunge segmentiert.

Der Algorithmus:

Im 3-dimensionalen hat ein Voxel 6 Nachbarn entlang der Achsen, wie **Abbildung 51** zeigt. Zusätzlich könnte man an dieser Stelle noch Diagonalen mit berücksichtigen und würde dann von einer 26-Nachbarschaft statt einer 6-Nachbarschaft sprechen. Allerdings sind die Nachbarn entlang der Diagonalen weiter entfernt als die Nachbarn entlang der Achsen (Genauso wie bei einem Würfel die Diagonale länger ist als eine Seite). Dadurch könnte es zum Auslaufen der Segmentierung kommen, das heißt, dass zuviel segmentiert wird. *Hdrw* arbeitet daher mit der 6-Nachbarschaft.

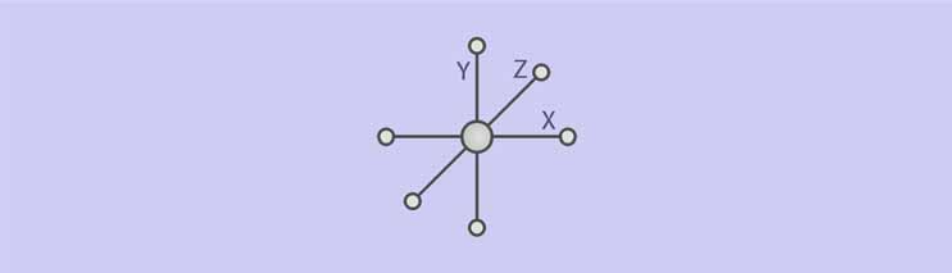

Abbildung 51: Tools > Region Growing (Images\Schnaidt\ToolsRegionGrowingSegmentation4.png)

Die Ähnlichkeit zweier Voxel wird durch ein so genanntes *Zusammenhangskriterium* bestimmt. Im Falle von *Hdrw* wird dies durch ein Grauwertintervall festgelegt. Mathematisch sieht dies folgendermaßen aus:

$$|V_{x,y,z}-V_S|<\varepsilon \tag{5}$$

In Formel (5) ist V_S der Grauwert des Seed Points und $V_{x,y,z}$ der Grauwert des betrachteten Voxels, der eventuell zur Segmentierung hinzugefügt werden soll. Liegt der Betrag der Differenz unterhalb von ε, dann gehört der Voxel zur Segmentierung. Im Programm entspricht ε dem *Threshold* (dt. Schwellwert).

In Pseudo Code sieht der Algorithmus folgendermaßen aus:

```
Wähle eine Threshold (Durch den Benutzer);
ε = Threshold;
Wähle einen Seed Point (Durch den Benutzer);
V_S = Helligkeitswert des Seed Point;
Lege die Koordinaten des Seed Points auf den VoxelStack;
while (VoxelStack nicht leer)
{
    V_x,y,z = Oberster Voxel des VoxelStacks;
    Markiere V_x,y,z als "bereits besucht" (Helligkeitswert 1);
    if (|V_x,y,z-V_S| < ε)
    {
        Markiere V_x,y,z als "segmentiert" (Helligkeitswert 255);
        if (V_x+1,y,z nicht "bereits besucht" && im Volumen)
            Lege die Koordinaten von V_x+1,y,z auf den VoxelStack;
        if (V_x-1,y,z ...) ...
        if (V_x,y+1,z ...) ...
        if (V_x,y-1,z ...) ...
        if (V_x,y,z+1 ...) ...
        if (V_x,y,z-1 ...) ...
    }
}
```

Wichtige Anmerkungen:

- **Arbeitet auf dem Volumen:** Diese Funktion arbeitet auf dem Volumen und nicht auf der Vorschau für die aktuelle Schicht, die Sie im aktiven Fenster sehen. Wenn Sie dies nach dem Windowing anwenden wollen, wählen Sie zuerst **Volume > Start Windowing** aus, um das Windowing für das ganze Volumen durchzuführen.
- **Öffnet ein neues Fenster:** Das Ergebnis dieser Funktion wird in ein neues Fenster eingefügt und ändert nicht das aktuelle Fenster (bzw. Volumen).

Weiterführende Informationen:

- **File > Save Volume As** (2.3.1.2, Seite 17)
- **Tools > Threshold Segmentation** (2.3.7.2, Seite 62)

2.3.7.2 Tools > Threshold Segmentation

Eine Einführung zur Segmentierung finden Sie in 2.3.7.1, Seite 58.

Die Threshold Segmentierung (dt. Schwellwertsegmentierung) ist sehr einfach, denn Sie wählt nur die Voxel in einem bestimmten Grauwertintervall aus. Im Dialog können Sie das Intervall angeben (**Abbildung 52**).

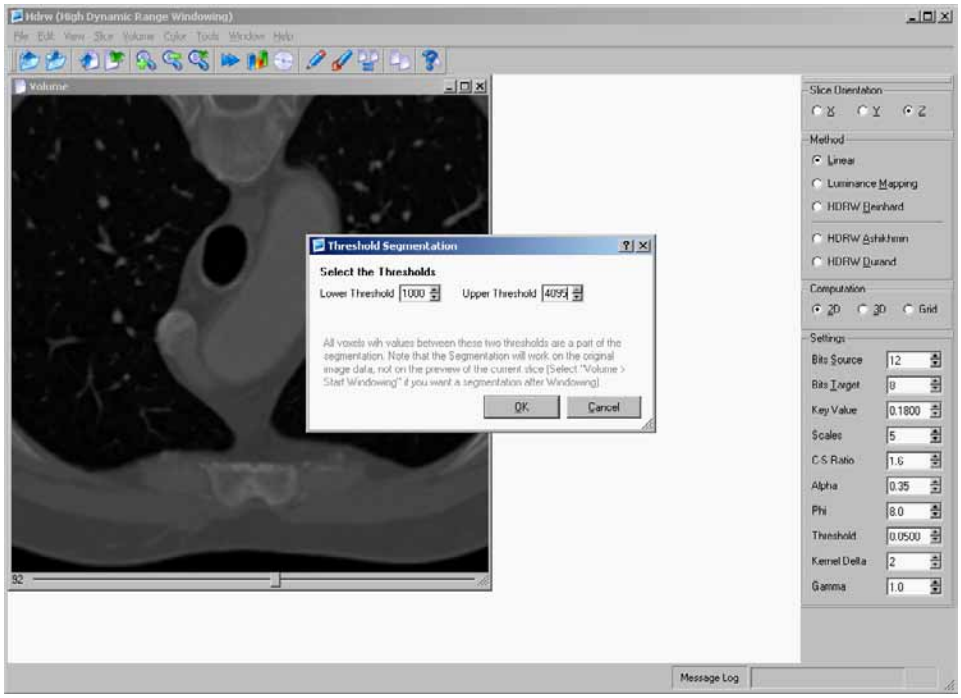

Abbildung 52: Tools > Threshold Segm. (Images\Schnaidt\ToolsThresholdSegmentation.png)

Alle Voxel in diesem Intervall werden segmentiert (**Abbildung 53**).

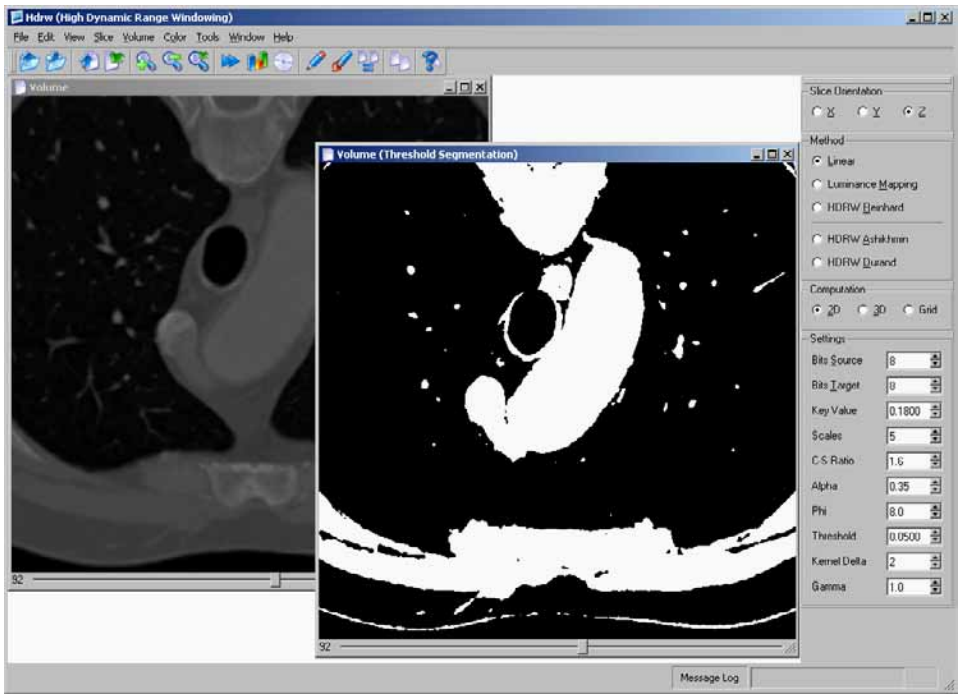

Abbildung 53: Tools > Threshold Segm. (Images\Schnaidt\ToolsThresholdSegmentation2.png)

Die segmentierten Voxel haben den Grauwert 255 (Weiß im Bild).

Wichtige Anmerkungen:

- **Arbeitet auf dem Volumen:** Diese Funktion arbeitet auf dem Volumen und nicht auf der Vorschau für die aktuelle Schicht, die Sie im aktiven Fenster sehen. Wenn Sie dies nach dem Windowing anwenden wollen, wählen Sie zuerst **Volume > Start Windowing** aus, um das Windowing für das ganze Volumen durchzuführen.
- **Öffnet ein neues Fenster:** Das Ergebnis dieser Funktion wird in ein neues Fenster eingefügt und ändert nicht das aktuelle Fenster (bzw. Volumen).

Weiterführende Informationen:

- **File > Save Volume As** (2.3.1.2, Seite 17)
- **Tools > Region Growing Segmentation** (2.3.7.1, Seite 58)

2.3.7.3 Tools > Compare Slices

Um die Unterschiede zwischen verschiedenen Windowing Methoden zu verdeutlichen, eignet sich gut ein Vergleich zweier Schichten. Mit dieser Funktion lassen sich die beiden aktuellen Schichten von zwei Fenstern vergleichen. In **Abbildung 54** sehen Sie hierfür ein Beispiel.

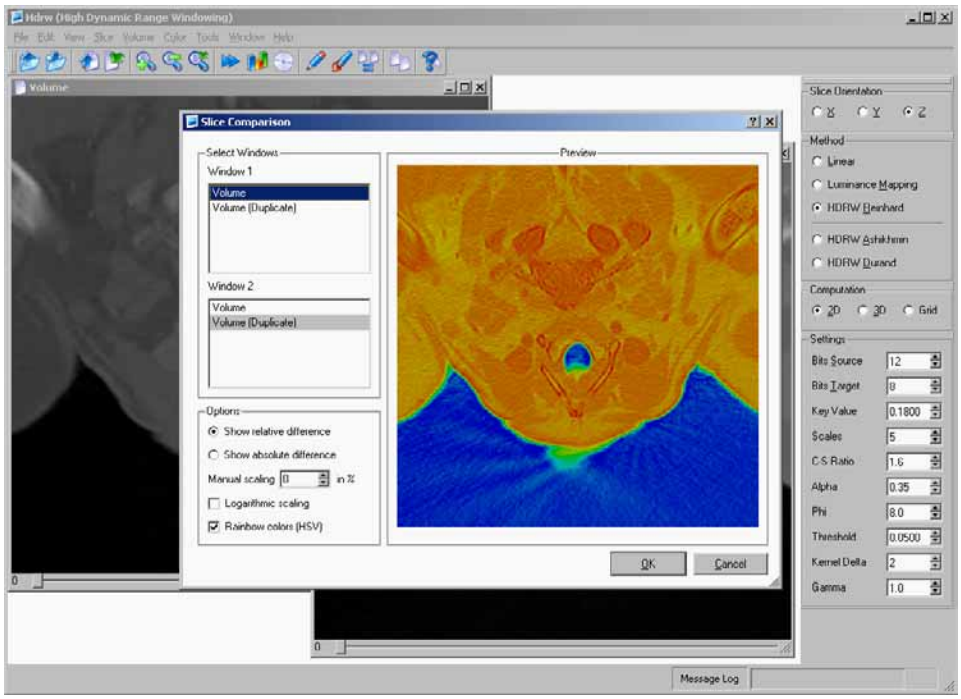

Abbildung 54: Tools > Compare Slices (Images\Schnaidt\ToolsCompareSlices.png)

Dabei gibt es verschiedene Einstellungen:

- **Rainbow colors (HSV):** Die Unterschiede zwischen den beiden Bildern werden normalerweise als Grauwerte dargestellt. Schwarz ist eine geringe Differenz, Weiß eine sehr große Differenz. Wenn sie dieses Kontrollkästchen aktivieren, wird stattdessen der Farbton variiert. Blau ist nun eine geringe Differenz und Rot eine sehr große Differenz (**Abbildung 54**).

- **Show relative difference:** Dies zeigt den Unterschied relativ an. Dies bedeutet, dass die maximale Differenz der beiden Bilder als Weiß bzw. Rot dargestellt wird.

- **Show absolute difference:** Dies zeigt den Unterschied absolut an. D.h. eine Differenz von beispielsweise 10 Grauwerten, wird als Grauwert 10 angezeigt.

- **Manual scaling:** Hiermit können Sie die angezeigte Differenz manuell vergrößern oder verkleinern (letzteres mit negativen Prozentwerten).

- **Logarithmic scaling:** In der logarithmischen Darstellung können Sie sowohl kleine als auch große Grauwertdifferenzen gleichzeitig sichtbar machen.

Wichtige Anmerkungen:

- **Arbeitet auf der Vorschau:** Diese Funktion arbeitet auf der Vorschau für die aktuelle Schicht, die Sie im aktiven Fenster sehen.

Weiterführende Informationen:

- **Color > Color Space** (2.3.6.2, Seite 56)

2.3.7.4 Tools > Transfer Function

Beim medizinischen Volumen und dem Windowing spielen oft so genannte Transferfunktionen eine wichtige Rolle. Im Prinzip ist eine Transferfunktion eine ganz normale Funktion, die allen Grauwerten jeweils einen anderen Grauwert zuordnet.

Beim Windowing wird den alten Grauwerten ebenfalls jeweils ein neuer Grauwert zugeordnet, was sich genau in solch einer Transferfunktion ausdrücken lässt. In **Abbildung 55** sehen Sie hierfür ein Beispiel. Dem Grauwert 0 wird 0 zugeordnet, dem Grauwert 1 wird 205 zugeordnet, etc. . Die Auflistung können Sie mit **Save** auch in einer Datei speichern.

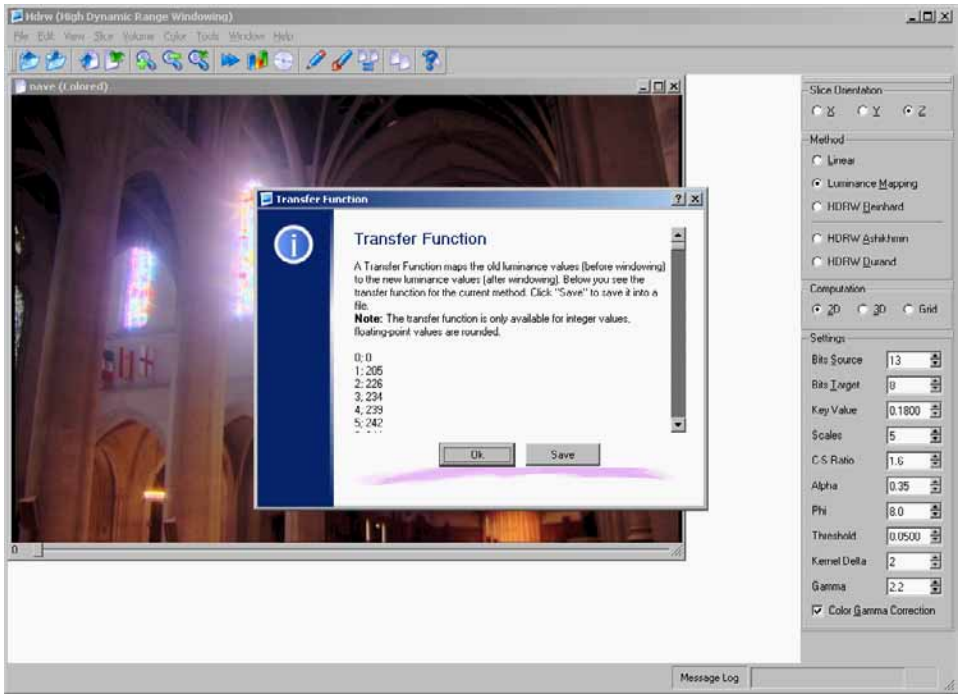

Abbildung 55: Tools > Transfer Function (Images\Schnaidt\ToolsTransferFunction.png)

Leider ist dies nicht für alle Windowing Methoden möglich, sondern nur für **Linear** und **Luminance Mapping**. Diese Methoden bezeichnet man auch als *global*. Die anderen Methoden dagegen sind *lokal*. Das bedeutet, dass einem Voxel mit Grauwert 127 nicht nur ein neuer Grauwert zugewiesen werden kann, sondern mehrere verschiedene. Welcher davon ausgewählt wird, hängt von den umgebenden Voxeln dieses Voxels ab.

Außerdem arbeitet die Auflistung nur für Ganzzahlen, da für Gleitkommazahlen viel zu viele Werte aufgelistet werden müssten.

Trotz dieser beiden Einschränkungen ist diese Funktion im medizinischen Bereich dennoch nützlich. Zum einen arbeitet man hier meist mit Ganzzahlen, beispielsweise CT Daten mit 4096 verschiedenen Grauwerten. Zum anderen liefert die **Luminance Mapping** Methode eine gute Abschätzung der Transferfunktion für **Hdrw Reinhard**. Auch wenn **Hdrw Reinhard** Details des Bildes lokal verändert, stellt das **Luminance Mapping** bereits die Grundhelligkeit des Bildes richtig ein. In der Medizin wird solch eine Transferfunktion für die *Segmentierung* und die *Klassifikation* benutzt. Da die verwendeten Verfahren dort selbst oft Schätzwerte verwenden, die durch Austesten ermittelt wurden, eignet sich hier durchaus die Transferfunktion des **Luminance Mapping** als gute Näherung für **Hdrw Reinhard**.

Mehr zur *Segmentierung* finden Sie in **Tools > Region Growing Segmentation**. Hier benötigt man die Transferfunktion, um die Schwellwerte richtig übertragen zu können.

Im Gegensatz zur Segmentierung bestimmt die *Klassifikation*, wie etwas im Datensatz dargestellt wird und nicht was dargestellt. Beispielsweise können Sie mit der Segmentierung die Lunge in einem Patientendatensatz auswählen. Die Klassifikation bestimmt danach die Darstellung der Lunge (Farbe, Transparenz). Sie arbeitet dafür ausschließlich mit Transferfunktionen, die nach dem gleichen Prinzip wie die oben beschriebene Transferfunktion arbeiten. Eine Transferfunktion für die Klassifikation hat oft die Form:
Grauwert → (Rotwert, Grünwert, Blauwert, Transparenzwert)
Typischerweise wird dabei zur Festlegung der Transferfunktion das Histogramm benutzt, eventuell zusätzlich das Gradientenhistogramm.
Die Transferfunktion von *Hdrw* kann nun dazu benutzt werden, um die Transferfunktion der Klassifikation nach dem Windowing anzupassen.

Wichtige Anmerkungen:

- **Arbeitet auf der Vorschau:** Diese Funktion arbeitet auf der Vorschau für die aktuelle Schicht, die Sie im aktiven Fenster sehen.

Weiterführende Informationen:

- **Volume > Histogram** (2.3.5.2, Seite 41)
- **Tools > Region Growing Segmentation** (2.3.7.1, Seite 58)

2.3.8 Menü Window

2.3.8.1 Window > Cascade

Diese Funktion ordnet alle Fenster überlappend in Originalgröße an (**Abbildung 56**).

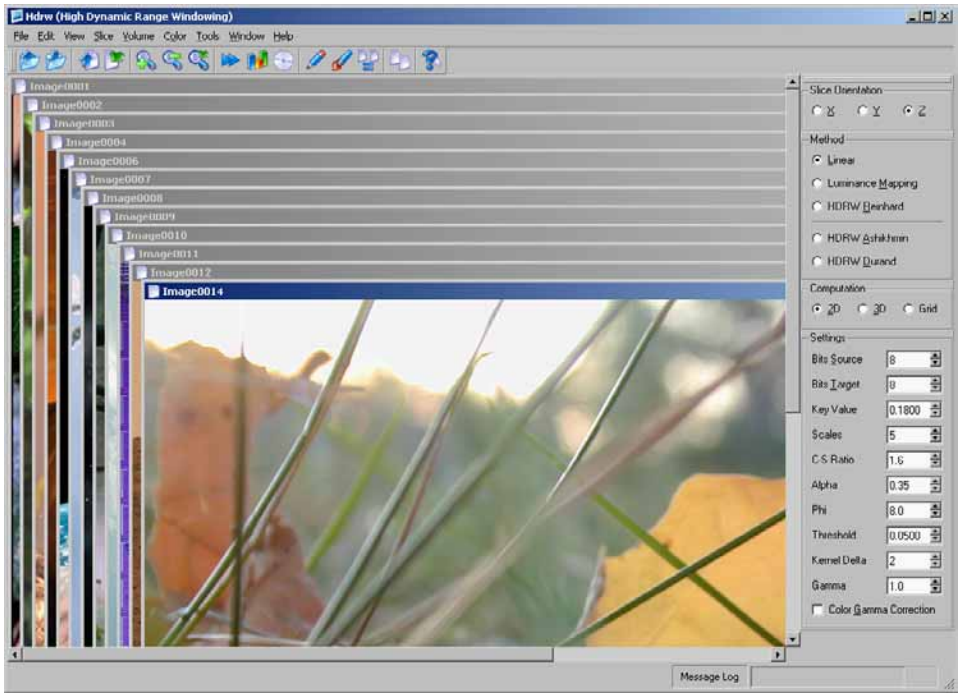

Abbildung 56: Window > Cascade (Images\Schnaidt\WindowCascade.png)

2.3.8.2 Window > Tile

Hiermit können Sie alle Fenster mosaikförmig anordnen (**Abbildung 57**).

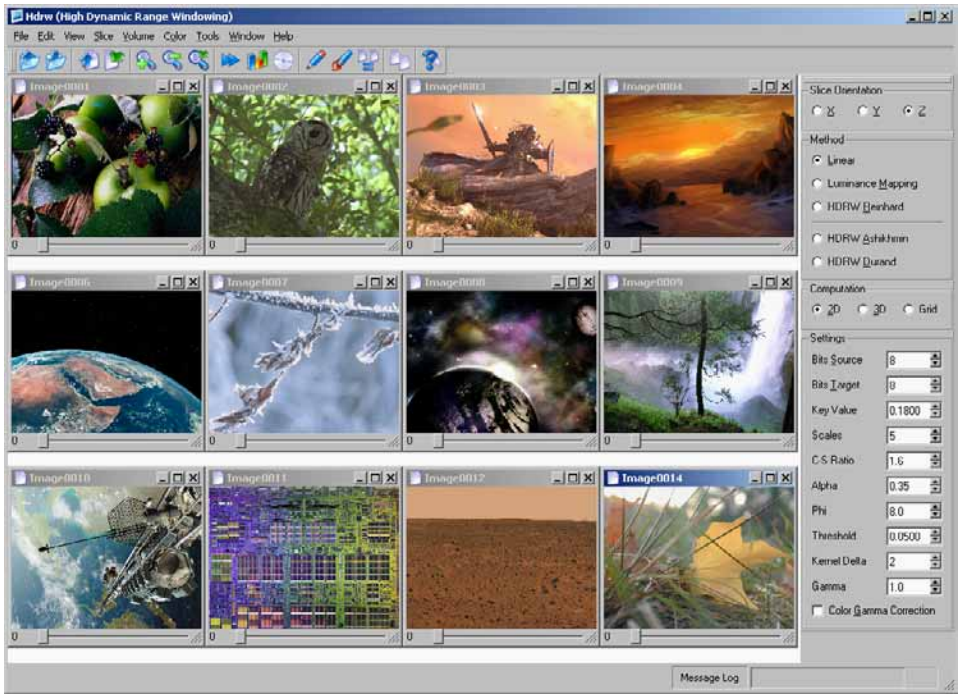

Abbildung 57: Window > Tile (Images\Schnaidt\WindowTile.png)

2.3.8.3 Window > Duplicate Window

Dies dupliziert das aktuelle Fenster und eignet sich besonders gut zum Testen verschiedener Einstellungen oder Windowing Methoden (**Abbildung 58**).

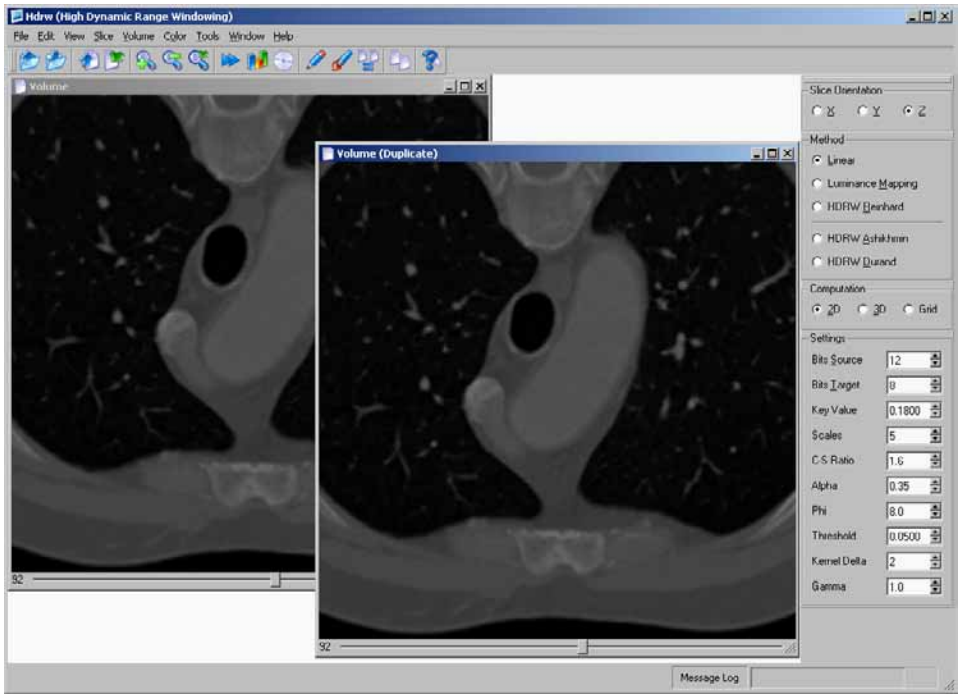

Abbildung 58: Window > Duplicate Window (Images\Schnaidt\WindowDuplicateWindow.png)

Gerade bei größeren Volumen hat dies außerdem den Vorteil, dass dadurch kein zusätzlicher Speicher benötigt wird.

2.3.8.4 Window > Close Window

Schließt das aktive Fenster.

2.3.8.5 Window > Close All Windows

Schließt alle geöffneten Fenster (**Abbildung 59**).

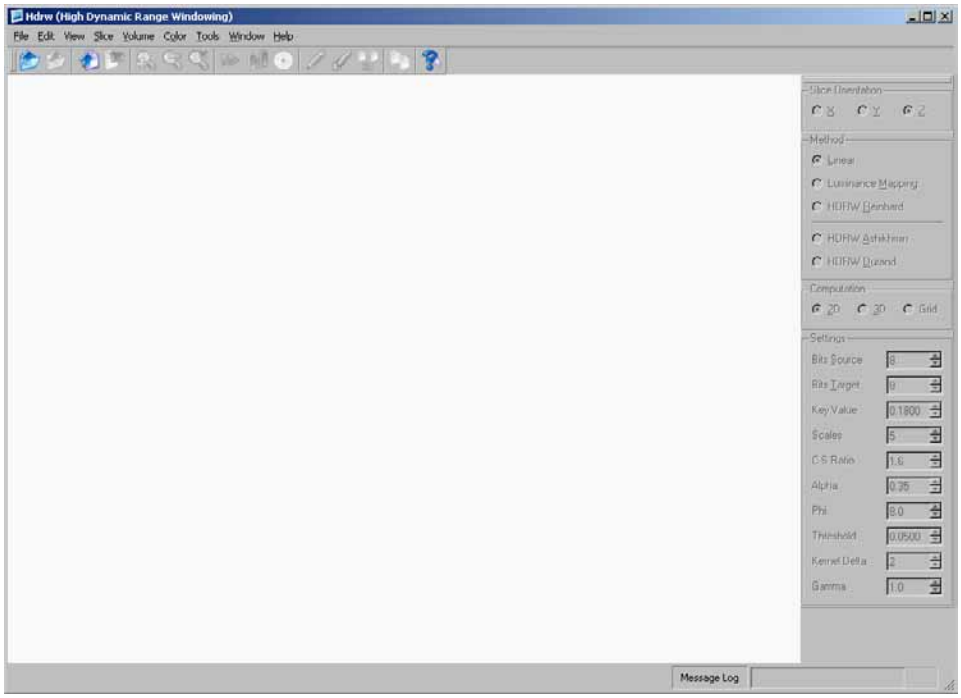

Abbildung 59: Window > Close All Windows (Images\Schnaidt\WindowCloseAllWindows.png)

2.3.8.6 Window > Clear Recent Files

Im Menü **File** werden die Volumen angezeigt, die Sie als letztes geöffnet haben (**Abbildung 60**). Hiermit können Sie diese Liste löschen.

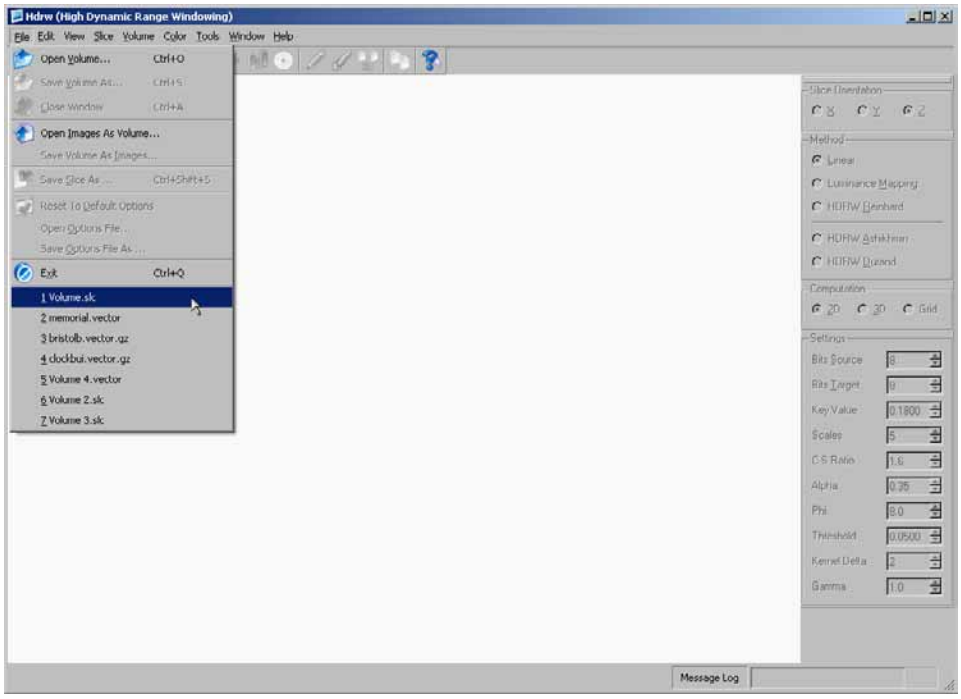

Abbildung 60: Window > Clear Recent Files (Images\Schnaidt\WindowClearRecentFiles.png)

2.3.9 Menü Help

2.3.9.1 Help > First Steps

Dieser Dialog kann Ihnen als Hilfe dienen, falls Sie *Hdrw* zum ersten Mal benutzen (**Abbildung 61**). Er erklärt die wichtigsten Funktionen und kann zum Öffnen des Beispielvolumens dienen (**Sample Volume**) und zum Öffnen von normalen Volumen (**Open Volume**). Mehr dazu finden Sie in der Einführung in 2.2.1, Seite 11.

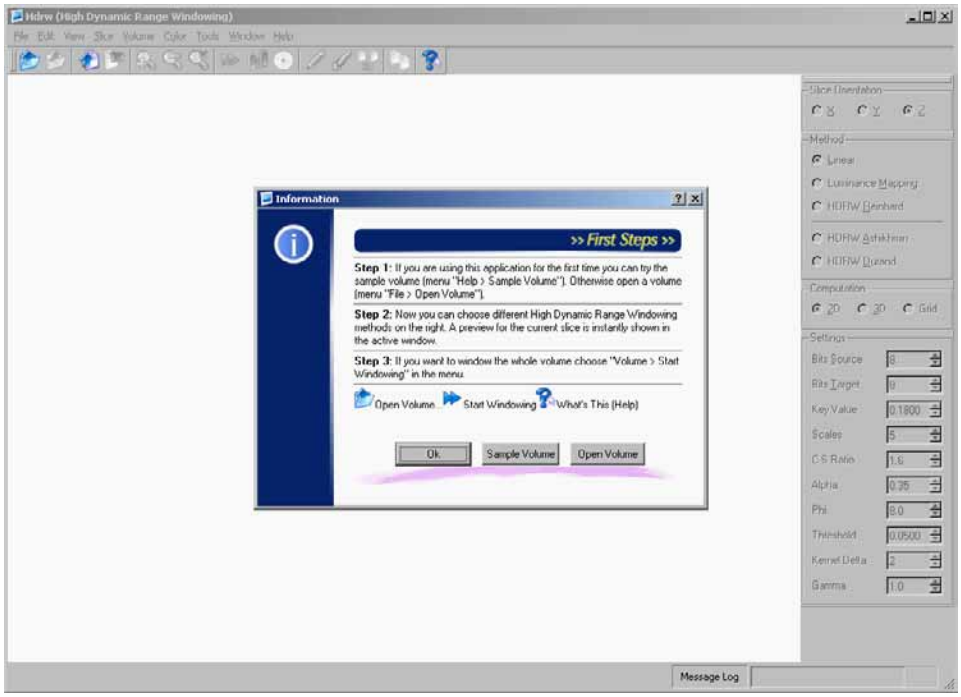

Abbildung 61: Help > First Steps (Images\Schnaidt\HelpFirstSteps.png)

Weiterführende Informationen:

- **File > Open Volume** (2.3.1.1, Seite 15)
- **Help > Sample Volume** (2.3.9.2, Seite 75)

2.3.9.2 Help > Sample Volume

Das Beispielvolumen (**Abbildung 62**) kann zum Testen der verschiedenen Methoden benutzt werden. Mehr dazu finden Sie in der Einführung in 2.2.1, Seite 11.

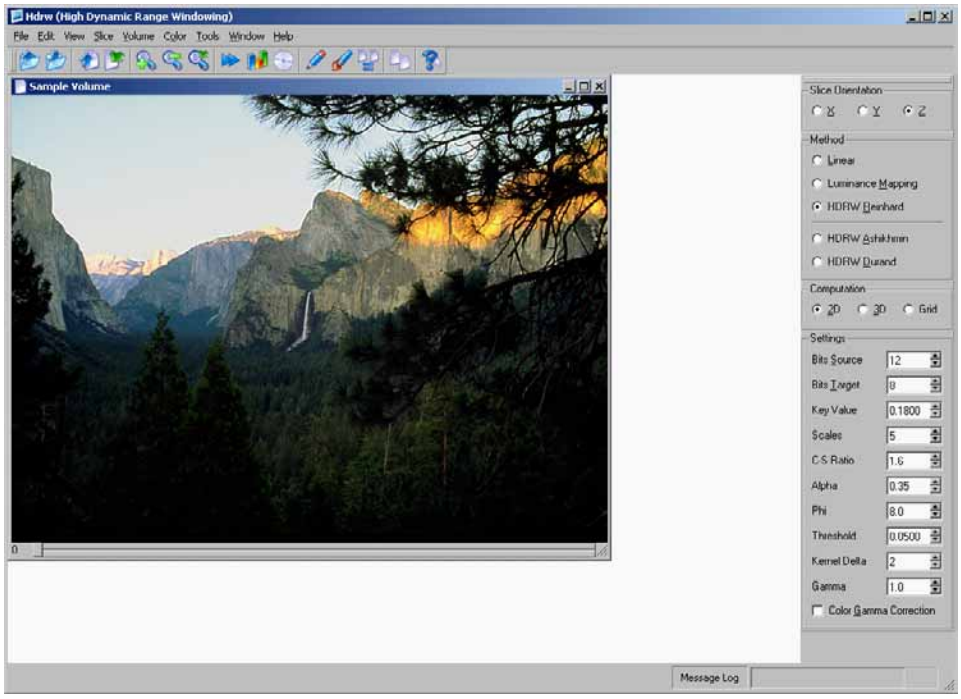

Abbildung 62: Help > Sample Volume (Images\Schnaidt\HelpSampleVolume.png)

2.3.9.3 Help > What's This

Wenn Sie zuerst auf und dann auf ein beliebiges anderes Icon oder Interface Element klicken, erhalten Sie direkt den zugehörigen Hilfetext. Ein Beispiel hierfür sehen Sie in **Abbildung 63**. Dieser Hilfetext beschreibt kurz die Funktion und wichtige Eigenschaften. Eine ausführliche Erklärung aller Funktionen finden Sie in diesem Kapitel.

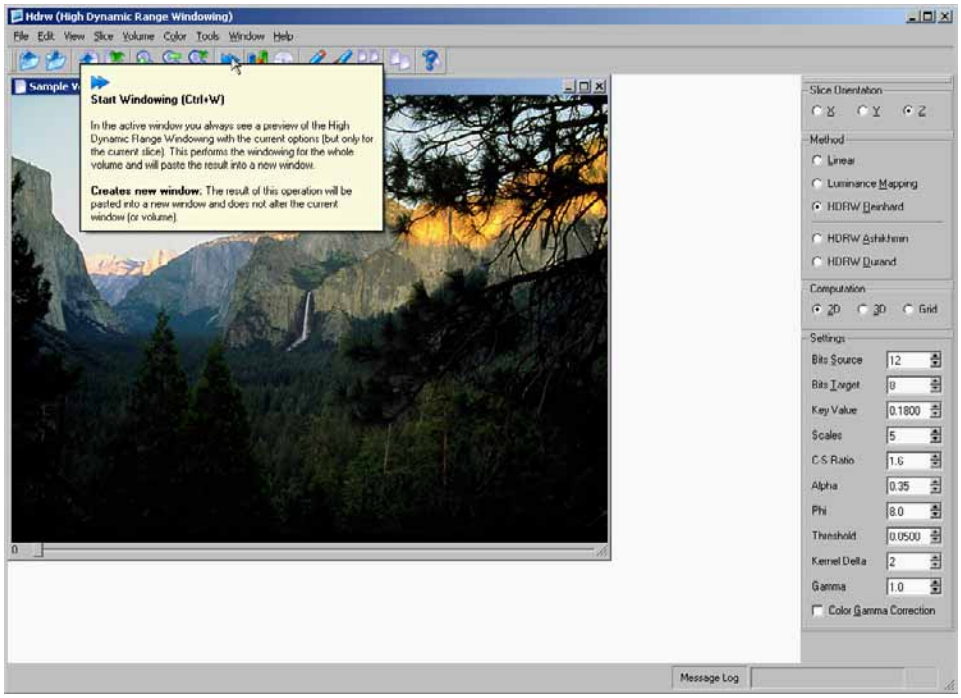

Abbildung 63: Help > What's This (Images\Schnaidt\HelpWhatsThis.png)

2.3.9.4 Help > About

Der **About** (dt. Info) Dialog von *Hdrw* (**Abbildung 64**).

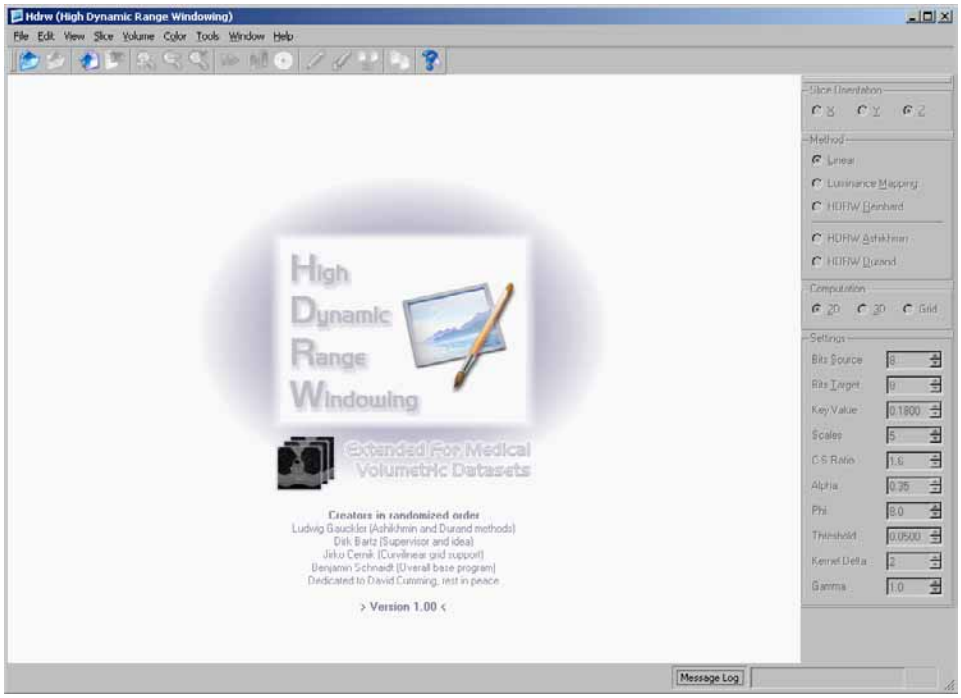

Abbildung 64: Help > About (Images\Schnaidt\HelpAbout.png)

2.4 Die Optionen

Hdrw bietet eine ganze Reihe an Optionen bzw. Einstellungen, mit denen Sie das Windowing anpassen können. In **Abbildung 65** rechts sehen Sie die Standardeinstellungen.

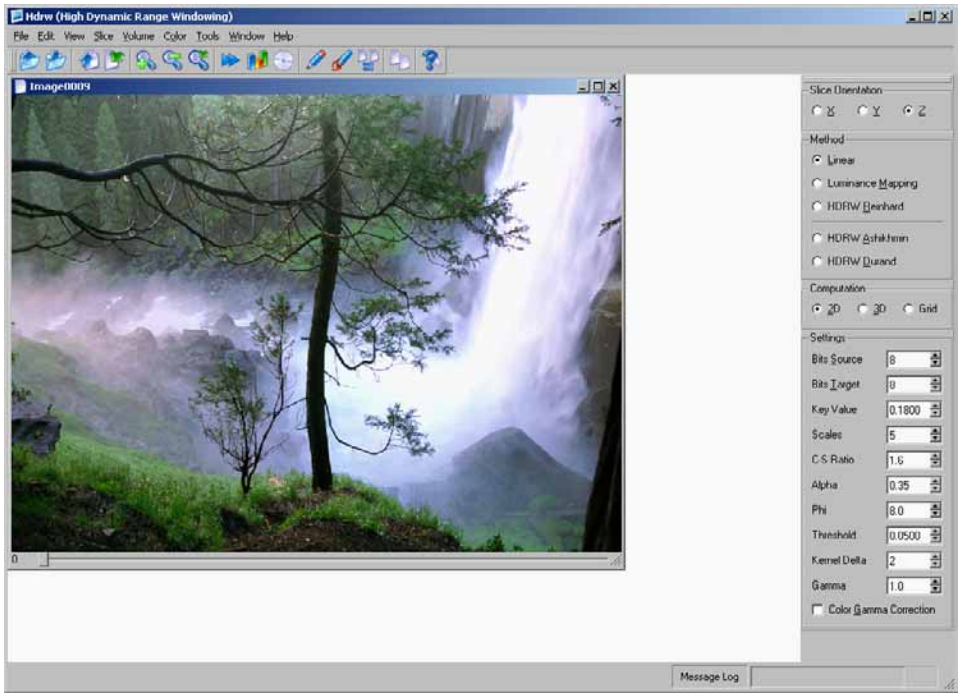

Abbildung 65: Standardeinstellungen (Images\Schnaidt\DefaultOptions.png)

Falls Sie sich noch nicht näher mit Windowing Algorithmen beschäftigt haben, werden Ihnen wahrscheinlich einige Bezeichnung unter **Settings** nichts sagen, was auch völlig normal ist. Sie brauchen sich über diese Optionen meist auch keine Gedanken machen, da die Standardeinstellungen für eine Vielzahl von Bildern gute Ergebnisse liefern.

Nur eine Gamma-Korrektur ist bei Farbbildern oft nötig. Um das Bild aufzuhellen, geben Sie einfach einen größeren **Gamma** Wert an, beispielsweise 1,6 oder 2,2. Bei hohen **Gamma** Werten und Farbbildern empfehlen wir außerdem **Color Gamma Correction** zu aktivieren (Dies benötigt etwas mehr Zeit, liefert aber meist bessere Resultate).

Das folgende Kapitel liefert einen groben Überblick über alle Einstellungen. Es wird dabei bewusst auf Details der Algorithmen verzichtet, die ab Kapitel 3, Seite 99 folgen. Wenn Sie sich nur grob für die Einstellungen interessieren, können Sie außerdem den größten Abschnitt **Settings** (2.4.4, Seite 83) komplett überspringen.

2.4.1 Slice Orientation

Ein Volumen ist im Grunde nichts anderes als ein Quader. Im aktiven Fenster sehen Sie immer nur eine Schicht des Quaders. Da der Quader verschiedene Seiten hat, kann man in 3 unterschiedlichen Orientierungen Schichten durch diesen Quader legen. Normalerweise betrachten man das Volumen entlang der Z-Achse. Dies ist in **Abbildung 66** dargestellt.

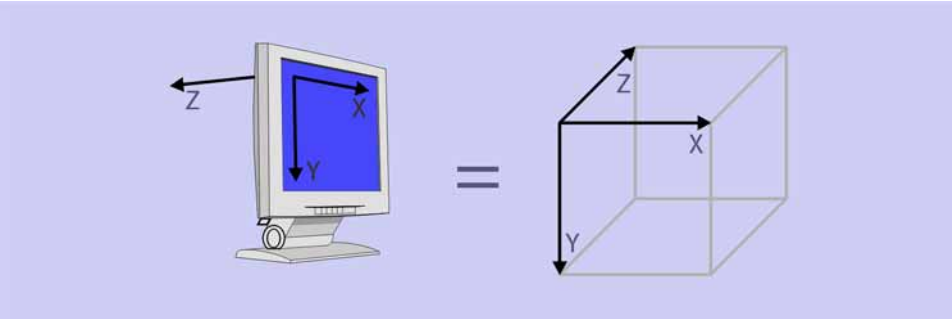

Abbildung 66: Slice Orientation (Images\Schnaidt\SliceOrientation.png)

Auf dem Monitor sehen Sie die XY-Ebene der aktuellen Schicht und mit dem Schieberegler unterhalb des Volumens können Sie die aktuelle Schicht verändern.

Mit **Slice Orientation** (Rechts oben in **Abbildung 67**) lässt sich die Blickrichtung wechseln. Im Bild sehen Sie links das Volumen entlang der Z-Achse (die Standardeinstellung), unten entlang der Y-Achse und rechts entlang der X-Achse.

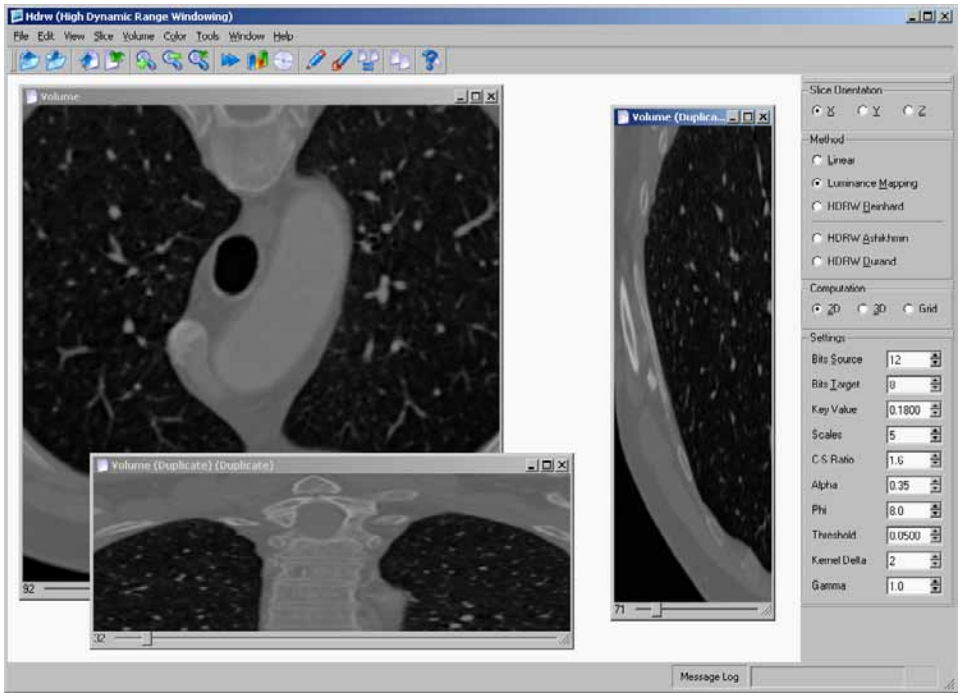

Abbildung 67: Slice Orientation (Images\Schnaidt\SliceOrientation2.png)

2.4.2 Method

Mit **Method** in **Abbildung 68** rechts oben können Sie zwischen den verschiedenen Windowing Methoden wechseln.

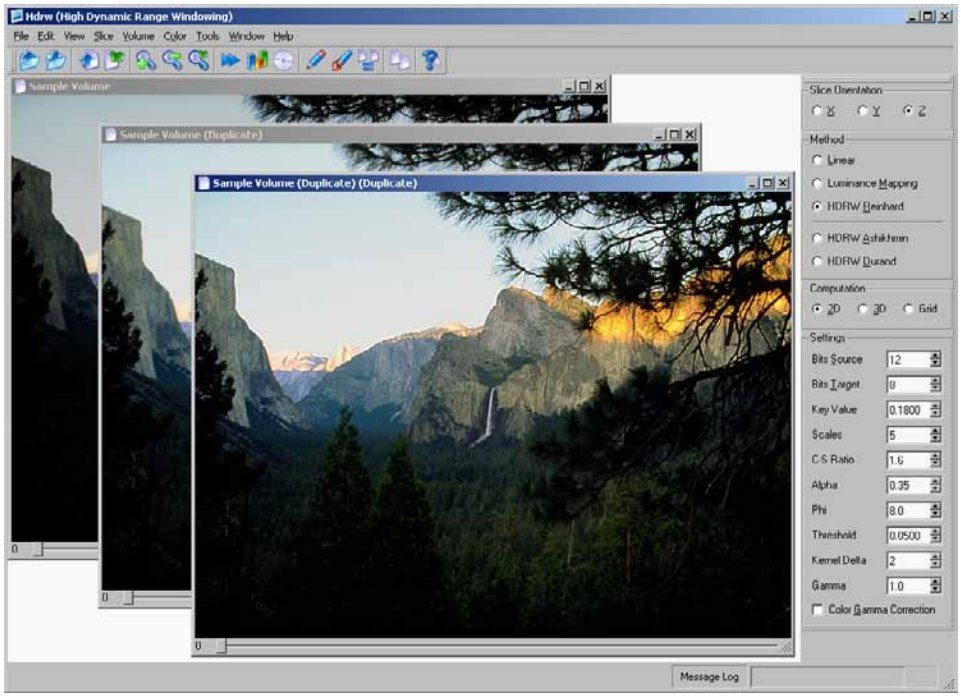

Abbildung 68: Method (Images\Schnaidt\Method.png)

In Kapitel 1, Seite 5 finden Sie eine allgemeine Einführung zum Windowing und zur Methode **Linear**.

Zusammengefasst haben die verschiedenen Methoden grob folgende Bedeutung:

- **Linear:** Das lineare Windowing ist das einfachste und schnellste Verfahren, es liefert aber normalerweise auch die schlechtesten Resultate.
- **Luminance Mapping:** Das Luminance Mapping ist im Prinzip die Vorstufe zu Hdrw Reinhard. Es stellt die Grundhelligkeit des Bildes ein und ist auch ein schnelles Verfahren.
- **Hdrw Reinhard:** Dies ist die zentrale Windowing Methode von *Hdrw*, die auf den meisten Bildern die besten Resultate erzeugt. Es wird die Grundhelligkeit des Bildes eingestellt (wie beim Luminance Mapping) und zusätzlich wird versucht möglichst viel von der Genauigkeit des Originalvolumens zu erhalten.
- **Hdrw Ashikhmin:** Dies ist eine alternative Windowing Methode von Ashikhmin. Sie dient nur zum Vergleich und ist nicht auf Geschwindigkeit optimiert.
- **Hdrw Durand:** Auch Durand dient wie Ashikhmin nur zum Vergleich und ist (in unserer Implementierung) das zeitaufwendigste Verfahren.

Ab Kapitel 3, Seite 99 folgen die Details der Methoden.

2.4.3 Computation

Nicht alle, aber einige der Windowing Methoden können ihre Berechnungen auf unterschiedliche Weise durchführen. In *Hdrw* wird dies unter dem Begriff **Computation** zusammengefasst (**Abbildung 69** rechts).

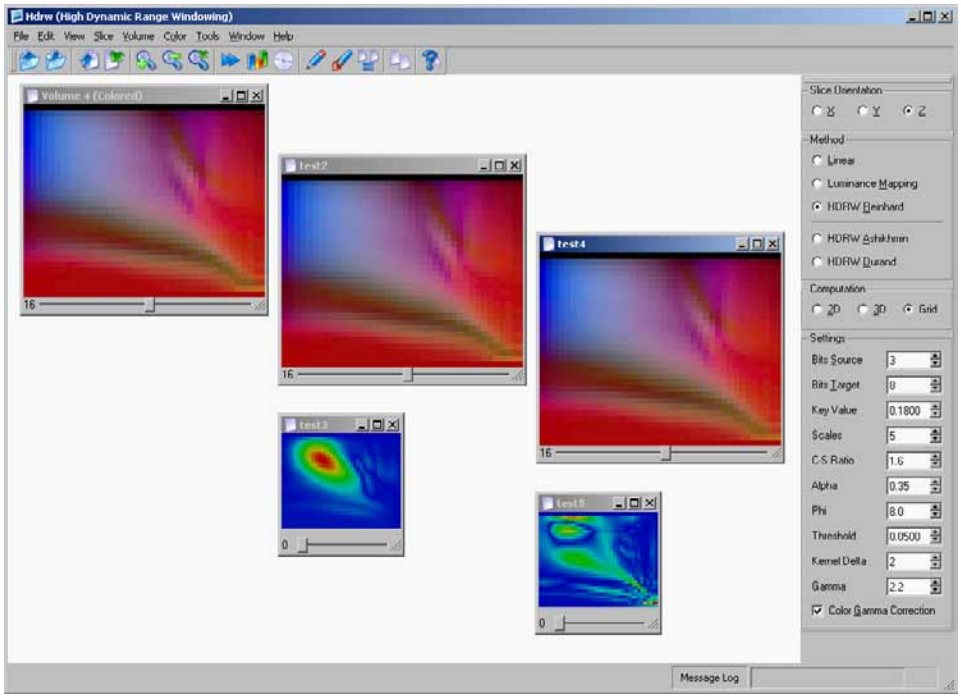

Abbildung 69: Computation (Images\Schnaidt\Computation.png)

Die verschiedenen Berechnungsarten haben dabei folgende Bedeutung:

- **2D:** Ein 3-dimensionales Volumen besteht aus einzelnen 2-dimensionalen Schichten. Im **2D** Modus werden diese Schichten getrennt betrachtet. Das heißt der Algorithmus arbeitet sich Schicht für Schicht vor und betrachtet dabei nur die aktuelle Schicht.

- **3D:** Im **3D** Modus wird stattdessen das 3-dimensionale Volumen genutzt. Zwar wird normalerweise nicht das komplette Volumen auf einmal betrachtet, aber statt nur einer Schicht arbeitet die Methode hier schichtübergreifend. Beispielsweise werden einige zusätzliche Schichten um die aktuelle Schicht herum in Betracht gezogen.
  Der **3D** Modus hat den Vorteil, dass das Ergebnis des Windowing auch dann korrekt ist, wenn Sie das Volumen zum Beispiel von der Seite betrachten (mit einer anderen **Slice Orientation**). Gerade bei medizinischen Volumen ist dies wichtig. Der 3D Modus benötigt dafür etwas mehr Zeit, allerdings nicht viel.

- **Grid:** Das Volumen, wie es in *Hdrw* dargestellt wird, ist immer ein Quader. In Wirklichkeit kann aber ein gebogenes Gitter den Daten zu Grunde liegen, welches in *Hdrw* nicht mit angezeigt wird. Das heißt statt einem Quader kann das Volumen im Prinzip eine beliebige Form haben (Siehe auch 2.3.5.8, Seite 51). Der **Grid** Modus berücksichtigt beim Windowing das wirkliche Aussehen des Volumens. Dies ist damit natürlich auch der aufwendigste Modus.

Verwendet das Volumen kein spezielles Gitter und ist ein ganz normaler Quader, wird stattdessen automatisch der **3D** Modus verwendet.

In **Abbildung 69** sehen Sie ein Beispiel eines Volumens mit einem curvilinearen Gitter: Links im **2D** Modus, in der Mitte im **3D** Modus und rechts im **Grid** Modus. Da die Detailunterschiede auf den ersten Blick nur schwer zu erkennen sind, finden Sie unter den Bildern die Differenzbilder. Diese geben die Differenz zwischen dem aktuellen Modus und dem vorherigen Modus an (Blau ist eine geringe Differenz, Rot eine hohe Differenz).

In einigen Fällen sind die Unterschiede aber auch wesentlicher deutlicher, wie man in **Abbildung 70** sehen kann. Das Ventrikelsystem in der Mitte des Gehirns ist im **2D** Modus links deutlicher heller als im **3D** Modus rechts. Die schichtübergreifende **3D** Berechnung führt hier zu einem anderem Ergebnis, das auch dann noch korrekt ist, wenn man das Volumen von der Seite oder von Oben betrachtet.

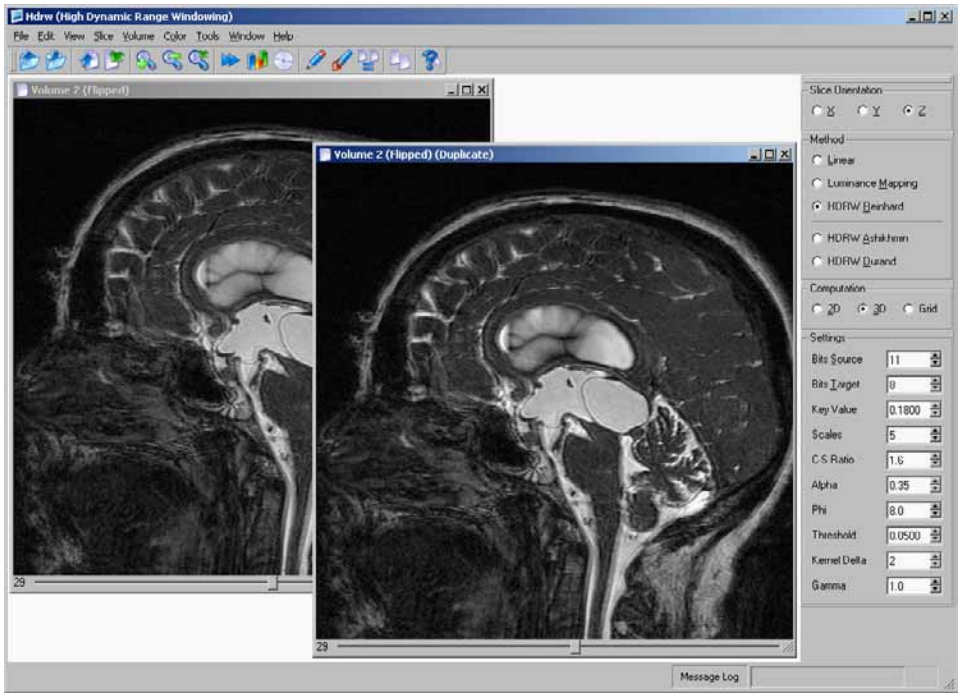

Abbildung 70: Computation (Images\Schnaidt\ComputationMedical.png)

Betreffende Methoden:

- **2D:** Alle Methoden
- **3D:** Hdrw Reinhard, Hdrw Ashikhmin, Hdrw Durand
- **Grid:** Hdrw Reinhard

2.4.4 Settings

Nicht alle **Settings** haben auf allen Windowing Methoden exakt dieselbe Bedeutung. Da aber **Hdrw Reinhard** die zentrale Methoden von *Hdrw* sind, beziehen sich die Beispiele normalerweise darauf.

Unter der jeweiligen Option finden Sie die Methoden aufgelistet, auf die diese Einstellung einen Einfluss hat. Ansonsten benötigt die Methode die Option nicht.

2.4.4.1 Bits Source

Diese Option gibt an, wie viel Bits im Volumen benutzt werden, das Sie geladen haben. Der Wert wird von *Hdrw* automatisch beim Laden des Bildes gesetzt und braucht normalerweise nicht verändert zu werden.

Beispielsweise ein Volumen aus einem CT Scan mit 4096 Grauwerten würde 12 Bit benutzen. Ein normales Bild aus einer PNG Datei mit 256 Grauwerten dagegen nur 8 Bit.

Ist der Wert zu hoch oder zu niedrig, wird das Volumen nicht richtig dargestellt. **Abbildung 71** zeigt hierfür ein Beispiel.

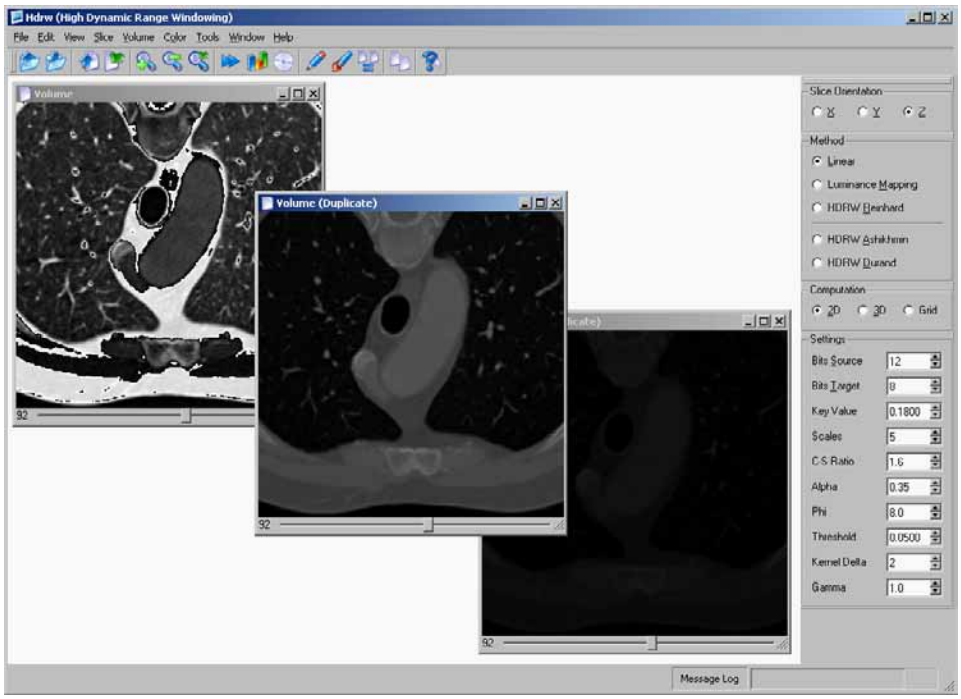

Abbildung 71: Bits Source (Images\Schnaidt\BitsSource.png)

Werden nur 10 Bit verwendet (Links), kommt es zu Artefakten. Werden 14 Bits verwendet (Rechts), erscheint das Bild zu dunkel. Nur bei 12 Bits (Mitte) wird das Volumen korrekt angezeigt.

Betreffende Methoden:

- **Linear**

2.4.4.2 Bits Target

Mit dieser Option können Sie einstellen, wie viel Bits das Volumen nach dem Windowing haben soll. Fast alle Monitore und Drucker arbeiten mit dem 24 Bit *RGB* Format, das 256 verschiedene Helligkeits- bzw. Grauwerte zulässt. Dies entspricht 8 Bit. Daher müssen Sie diesen Wert im Normalfall nicht anpassen.

Abbildung 72 zeigt das Ergebnis, wenn dieser Wert zu klein eingestellt wird (6 Bit links) oder zu groß (10 Bit rechts). Entweder ist das Bild zu dunkel oder es kommt zu Artefakten. Ein optimale Ergebnis ergibt sich bei 8 Bits (Mitte).

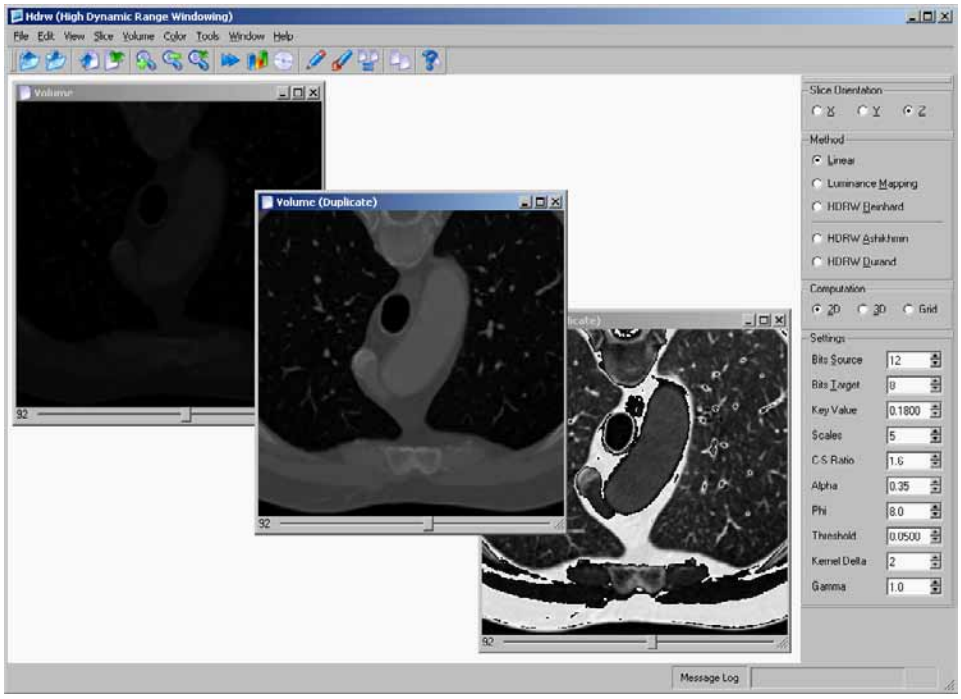

Abbildung 72: Bits Target (Images\Schnaidt\BitsTarget.png)

Betreffende Methoden:

- **Linear**
- **Luminance Mapping**
- **Hdrw Reinhard**
- **Hdrw Ashikhmin**
- **Hdrw Durand**

2.4.4.3 Key Value

Der **Key Value** ist ein wichtiger Wert für das **Luminance Mapping** und **Hdrw Reinhard**. Grundsätzlich gibt er einen Helligkeitswert auf einer Skala zwischen 0 und 1 an. Das "Mittelgrau" das Volumens hat nach dem Windowing diesen Helligkeitswert.
Normalerweise möchte man das Mittelgrau das Volumens auf Mittelgrau des Bildschirms abbilden. Daher wird die Standardeinstellung auf 0.18 gesetzt (Dies entspricht auf dem Bildschirm einem mittleren Grauton).

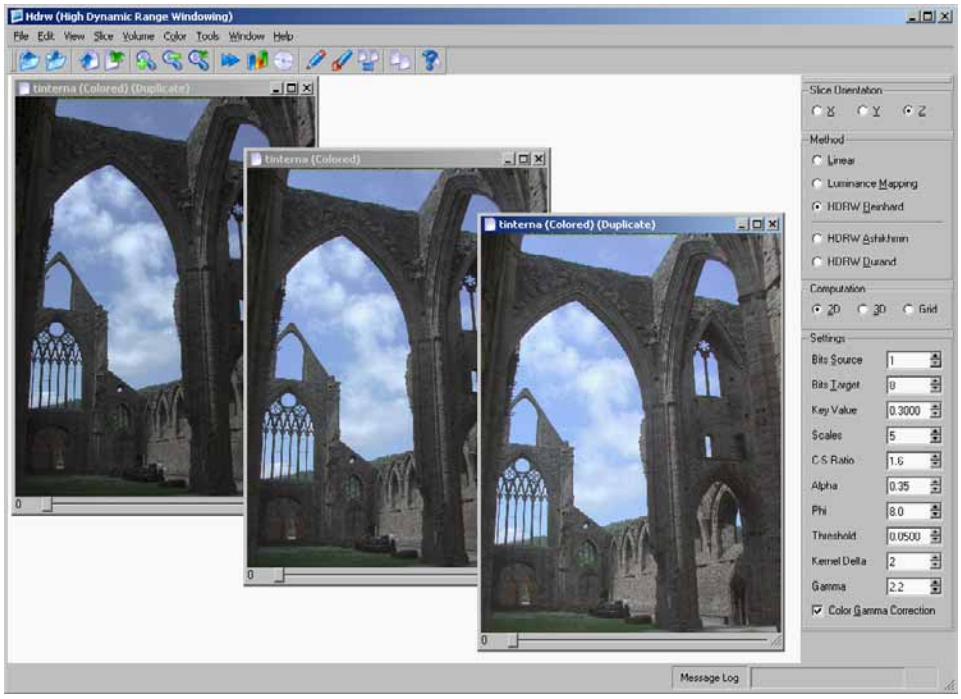

Abbildung 73: Key Value (Images\Schnaidt\KeyValue.png)

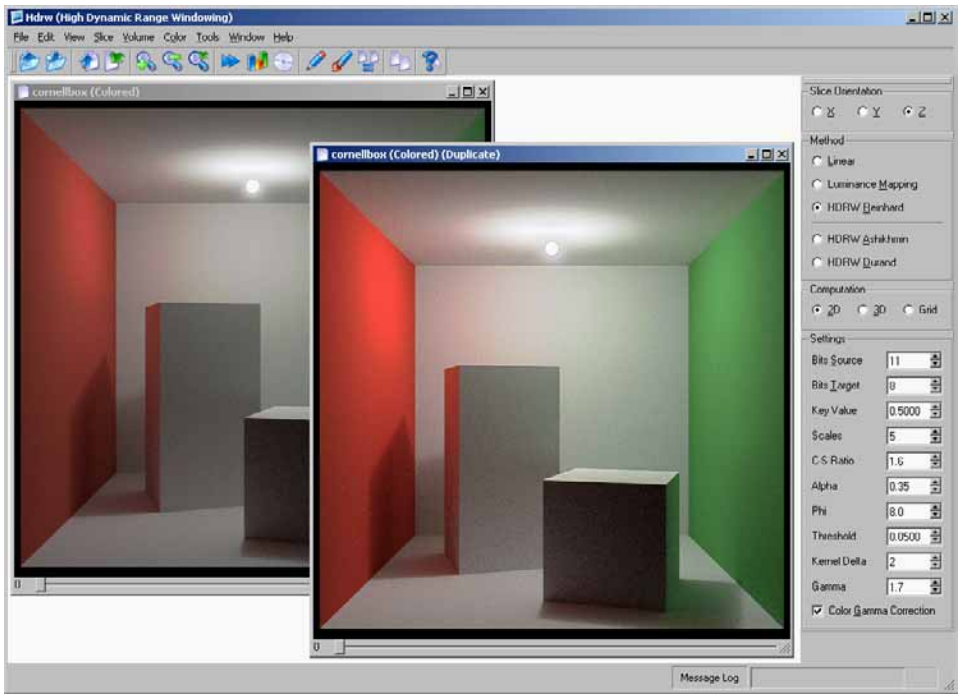

Abbildung 74: Key Value (Images\Schnaidt\KeyValueSharpening.png)

Ein Beispiel zeigt **Abbildung 73**. Von links nach rechts wurden die Key Values 0,10, 0,18 und 0,30 verwendet. Das mittlere Bild hat in etwa die richtige Helligkeit.

Erscheint das Bild zu dunkel, ist aber normalerweise ein Erhöhen des **Gamma** Wertes einfacher, da **Gamma** intuitivere Ergebnisse liefert. Denn ein höherer **Key Value** verändert nicht nur die Helligkeit. In **Abbildung 74** sehen sie links ein Bild mit normalem **Key Value** (0,18) und rechts mit erhöhtem **Key Value** (0,5). Beide wurde über die Gamma-Korrektur auf etwa dieselbe Helligkeit gebracht. Das rechte Bild erscheint dadurch deutlich schärfer, in diesem Fall beinahe zu scharf (Man hat den Eindruck von leichten Artefakten im Bild). In manchen Fällen kann die zusätzliche Schärfe aber erwünscht sein, dann macht eine Erhöhung des Key Value Sinn.

Mehr über den Key Value erfahren Sie in 3.4.1.2, Seite 104.

Betreffende Methoden:

- **Luminance Mapping**
- **Hdrw Reinhard**

2.4.4.4 Scales

Die Optionen **Scales**, **CS-Ratio**, **Alpha** und **Kernel Delta** können Sie im Normalfall auf den Standardeinstellungen belassen.

Die Vergrößerung von **Kernel Delta** und **Scales** gemeinsam kann zu etwas besseren Resultaten bei **Hdrw Reinhard** führen:

- **Kernel Delta** 2, **Scales** 5 (Standardeinstellung)
- **Kernel Delta** 5, **Scales** 7
- **Kernel Delta** 10, **Scales** 8

Jede Einstellung braucht dabei etwa doppelt soviel Zeit wie die vorherige (im **2D** oder **3D** Modus).

Wenn Sie die Optionen genauer anpassen möchten, ergeben sich die besten Resultate, wenn Sie alle vier aufeinander abstimmen. Mehr dazu finden Sie in 3.5.3, Seite 125.

Betreffende Methoden:

- **Hdrw Reinhard**
- **Hdrw Ashikhmin**

2.4.4.5 CS-Ratio

Die Optionen **Scales**, **CS-Ratio**, **Alpha** und **Kernel Delta** können Sie im Normalfall auf den Standardeinstellungen belassen.

Wenn Sie die Optionen doch anpassen möchten, ergeben sich die besten Resultate, wenn Sie alle vier aufeinander abstimmen. Mehr dazu finden Sie in 3.5.3, Seite 125.

Betreffende Methoden:

- **Hdrw Reinhard**
- **Hdrw Ashikhmin**

2.4.4.6 Alpha

Die Optionen **Scales**, **CS-Ratio**, **Alpha** und **Kernel Delta** können Sie im Normalfall auf den Standardeinstellungen belassen.

Wenn Sie die Optionen doch anpassen möchten, ergeben sich die besten Resultate, wenn Sie alle vier aufeinander abstimmen. Mehr dazu finden Sie in 3.5.3, Seite 125.

Betreffende Methoden:

- **Hdrw Reinhard**
- **Hdrw Ashikhmin**

2.4.4.7 Phi

Mit **Phi** können Sie die Schärfe des Bildes anpassen. Ein kleiner Wert sorgt für ein weicheres Bild, während ein größerer Wert ein schärferes Bild ergibt. In **Abbildung 75** wird **Phi** von 4 auf 8 und weiter auf 16 erhöht. Besonders die Glaskuppel im oberen Teil des Bildes wird dadurch deutlich schärfer.

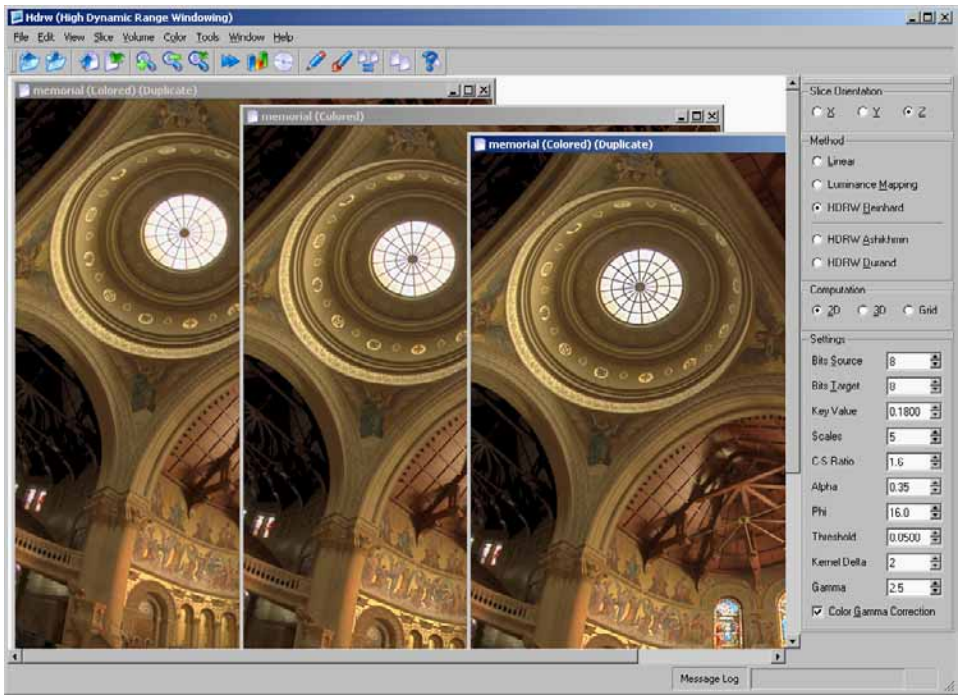

Abbildung 75: Phi (Images\Schnaidt\Phi.png)

Allerdings hat **Phi** nicht auf allen Volumen solch einen Effekt. Oft führt eine Änderung zu keinen sichtbaren Unterschieden.

Phi geht im Algorithmus exponentiell ein. Dadurch ist der Unterschied zwischen großen Werte von **Phi** meist nur noch unwesentlich, da große Werte zu einer Art "Sättigung" der Reinhard Methode führen. Die genaue Bedeutung von **Phi** finden Sie in 3.4.2.2, Seite 110.

Betreffende Methoden:

- **Hdrw Reinhard**

2.4.4.8 Threshold

Der Threshold ist ein wichtiger Wert der Methode **Hdrw Reinhard**. Die Standardeinstellung $0{,}05$ eignet sich hier aber für praktisch alle Bilder.

Eine Änderung dieses Werts hat meist schwer vorhersehbare Auswirkungen auf das Ergebnis und kann dazu führen, dass das Verfahren nicht den gewünschten Effekt hat oder gar Artefakte produziert. Dieses Problem wird in 3.4.2.3, Seite 111 genauer besprochen und **Abbildung 88** zeigt dort auch ein Beispiel. Grundsätzlich kann eine Erhöhung zu mehr Schärfe führen, erhöht aber auch die Wahrscheinlichkeit von Artefakten.

Auch Reinhard selbst empfiehlt diesen Wert unverändert zu lassen.

Betreffende Methoden:

- **Hdrw Reinhard**

2.4.4.9 Kernel Delta

Die Optionen **Scales**, **CS-Ratio**, **Alpha** und **Kernel Delta** können Sie im Normalfall auf den Standardeinstellungen belassen.

Die Vergrößerung von **Kernel Delta** und **Scales** gemeinsam kann zu etwas besseren Resultaten bei **Hdrw Reinhard** führen:

- **Kernel Delta** 2, **Scales** 5 (Standardeinstellung)
- **Kernel Delta** 5, **Scales** 7
- **Kernel Delta** 10, **Scales** 8

Jede Einstellung braucht dabei etwa doppelt soviel Zeit wie die vorherige (im **2D** oder **3D** Modus).

Wenn Sie die Optionen genauer anpassen möchten, ergeben sich die besten Resultate, wenn Sie alle vier aufeinander abstimmen. Mehr dazu finden Sie in 3.5.3, Seite 125.

Betreffende Methoden:

- **Hdrw Reinhard**
- **Hdrw Ashikhmin**
- **Hdrw Durand**

2.4.4.10 Gamma

Die Gamma-Korrektur ist der einfachste Weg die Helligkeit des Bildes anzupassen. Besonders Farbbilder benötigen oft einen **Gamma** Wert von beispielsweise 1,6 oder 2,2. Dies hellt die Bilder auf. **Gamma** Werte kleiner 1,0 machen das Bild dagegen dunkler.

Die Standardeinstellung für den **Gamma** Wert ist 1,0. Damit hat die Gamma-Korrektur keinen Effekt und ist praktisch ausgeschaltet. Gerade bei medizinischen Volumen ist die Gamma-Korrektur oft nicht nötig, braucht aber sehr viel Zeit, daher haben wir diese Standardeinstellung gewählt. Bei Werten $\neq 1{,}0$ wird die Korrektur wirklich berechnet und kann die Zeit beim Windowing durchaus deutlich beeinflussen.

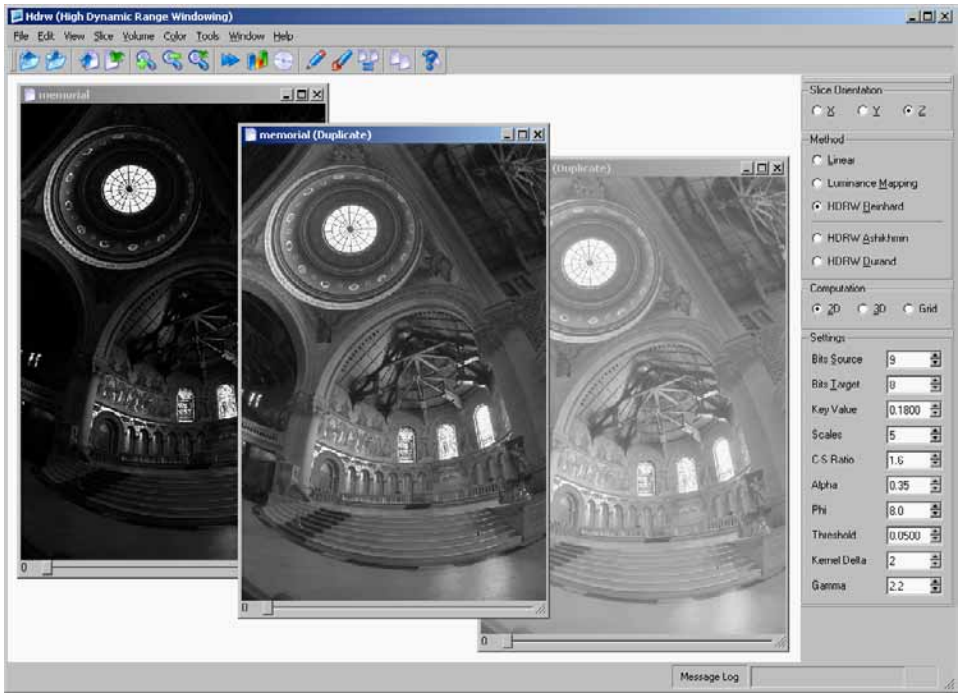

Abbildung 76: Gamma (Images\Schnaidt\Gamma.png)

Abbildung 76 ist ein Beispiel für die Gamma-Korrektur. Das Bild links hat den **Gamma** Wert 1,0, das Bild in der Mitte 2,2, das Bild rechts 5,0. Das mittlere Bild wirkt dabei am natürlichsten.

Betreffende Methoden:

- **Linear**
- **Luminance Mapping**
- **Hdrw Reinhard**
- **Hdrw Ashikhmin**
- **Hdrw Durand**

2.4.4.11 Color Gamma Correction

Speziell auf Farbbildern ist die **Color Gamma Correction** verfügbar. Sie kann mit dem Kontrollkästchen rechts unten in **Abbildung 77** eingeschaltet werden.

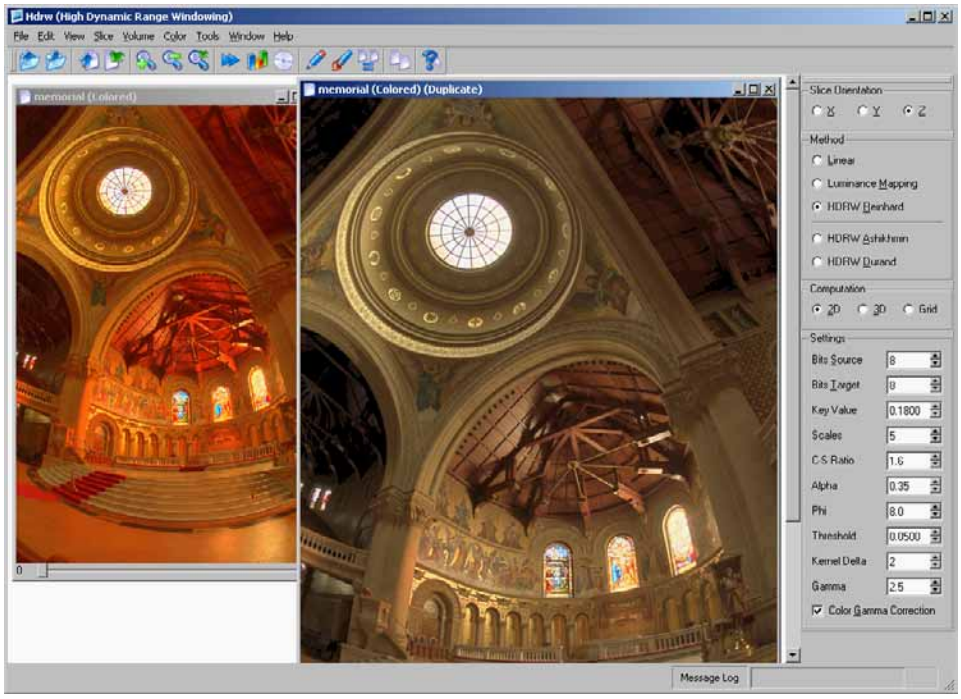

Abbildung 77: Color Gamma Correction (Images\Schnaidt\ColorGammaCorrection.png)

Auf dem linken Bild wurde eine normale Gamma-Korrektur durchgeführt. Dies bedeutet, dass die Gamma-Korrektur direkt auf den Helligkeitswerten arbeitet. Dies hat zur Folge, dass die Farben bei hohen **Gamma** Werten zu stark sättigen, anstatt wirklich heller zu werden. Im Bild rechts wurde die Gamma-Korrektur auf den Farben durchgeführt, wodurch sich ein besseres Ergebnis ergibt.

Um den Grund dafür zu erklären, wird das *RGB*, das *HSV* und das *Yxy* Farbsystem benötigt. Diese werden in 2.3.6.2, Seite 56 eingeführt.
Das Windowing von *Hdrw* arbeitet auf dem Helligkeitswert des *HSV* oder des *Yxy* Farbsystems. Um das Bild auf dem Monitor anzuzeigen, muss es zurück ins *RGB* Farbsystem konvertiert werden. Führt eine hohe Gamma-Korrektur zu einem hohen Helligkeitswert, so zeigen beide Farbsysteme bei der Konvertierung zu *RGB* eine Sättigung der Farben. Die **Color Gamma Correction** führt daher zuerst die Konvertierung nach *RGB* durch und erst danach die Gamma-Korrektur auf den einzelnen Rot, Grün und Blau Werten. Auf diese Weise hat die Gamma-Korrektur wieder den gewünschten Effekt der Aufhellung bei etwa unveränderter Sättigung der Farben.

Falls Sie mit komplett schwarzen Flächen im Ergebnis Probleme haben, die sich nicht gleichmäßig in den Rest des Bildes einfügen, gehen Sie folgendermaßen vor: Konvertieren Sie den Datentyp des Volumens in 64 Bit Float (**Volume > Convert Data Type > To Floating Point Number (64 Bit)**). Stellen Sie dann die Windowing Optionen ein und führen Sie anschließend ein Windowing des gesamten Volumens durch (**Volume > Start Windowing**). Danach sollten die Probleme beseitigt sein. Der Grund dafür ist fehlende Genauigkeit bei der Berechnung der Gamma-Korrektur auf den *RGB* Werten, was mit obiger Vorgehensweise vermieden werden kann.

Betreffende Methoden:

- **Linear**
- **Luminance Mapping**
- **Hdrw Reinhard**
- **Hdrw Ashikhmin**
- **Hdrw Durand**

2.5 Die Styles

Wie auch die meisten anderen Programme, passt sich *Hdrw* automatisch an den aktuellen *Style* bzw. das *Theme* (dt. Design) Ihres Desktops an. **Abbildung 78**, **Abbildung 79** und **Abbildung 80** zeigen hierfür einige Beispiele.

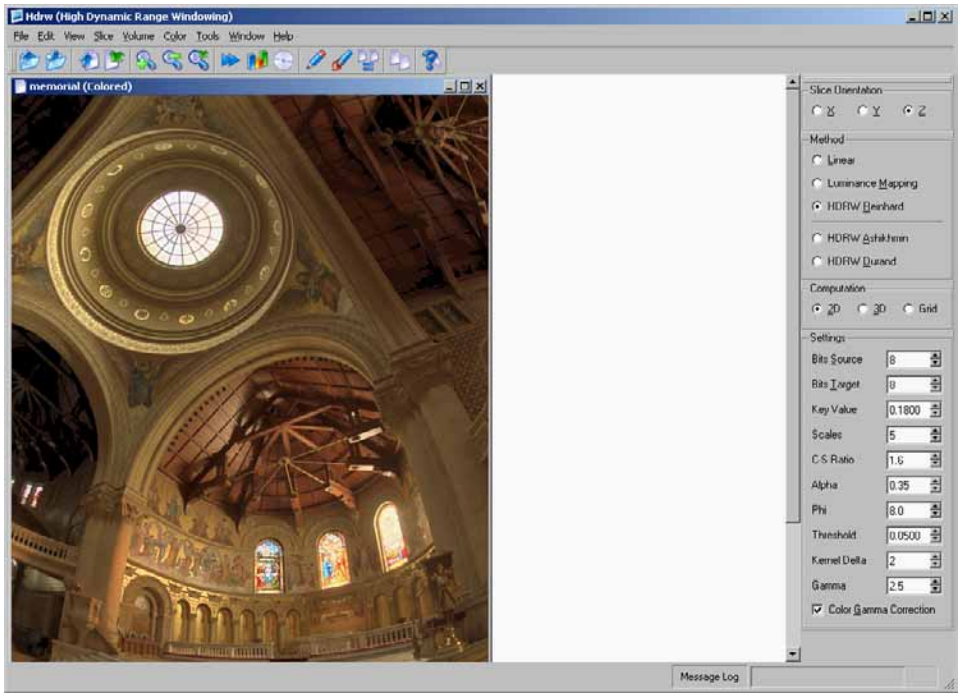

Abbildung 78: Windows XP Normal (Images\Schnaidt\Style1.png)

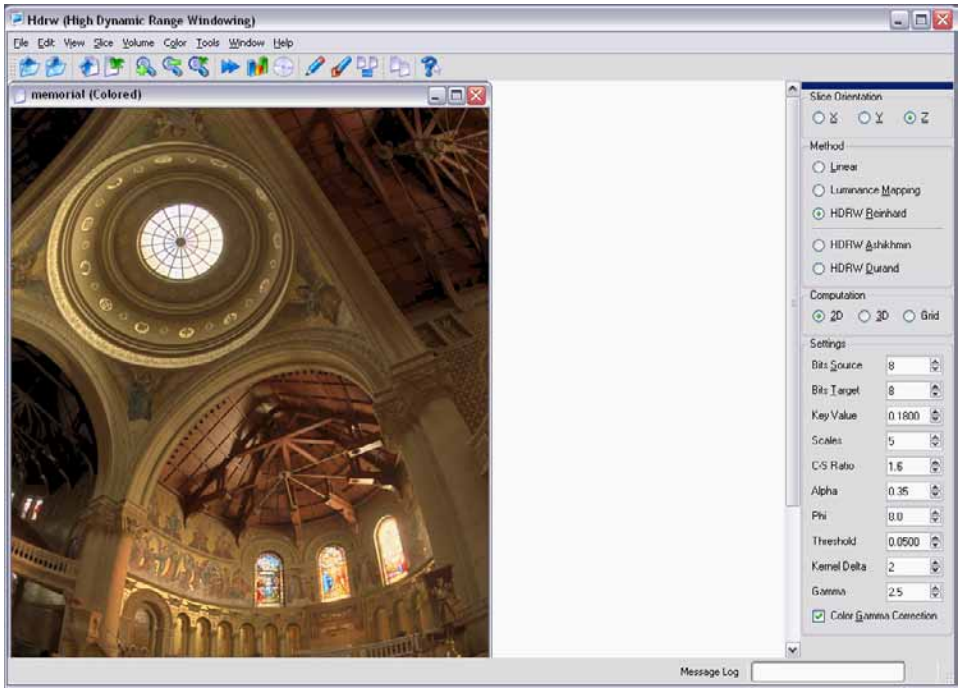

Abbildung 79: Windows XP Silver (Images\Schnaidt\Style2.png)

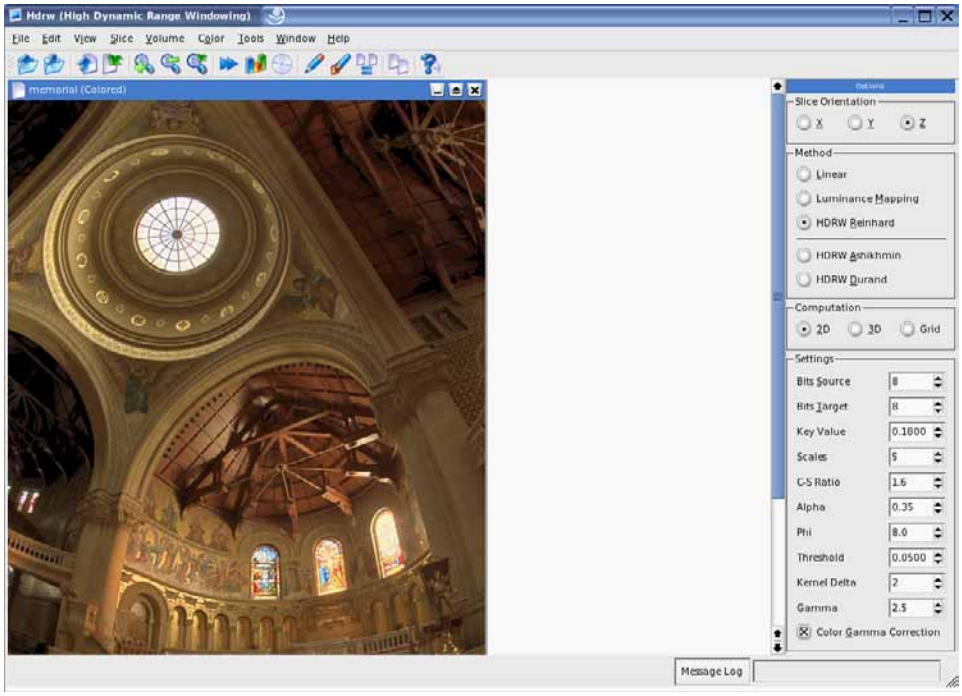

Abbildung 80: SUSE Linux 9.1 (Images\Schnaidt\Style3.png)

Benjamin Schnaidt

3 Die Reinhard Methode

Die folgenden Kapitel betrachten die Details der Windowing Methoden, welche in *Hdrw* eingesetzt werden. Sie erhalten einen kompletten Überblick, wie die Methoden mathematisch aufgebaut sind und algorithmisch implementiert wurden.

Die Kapitel richten sich an mathematisch und technisch interessierte Leser und setzen dabei einige grundlegende Kenntnisse im Bereich der Informatik voraus.

3.1 Einführung

Reinhard *et al.* (2002, p.267-276) definiert in seinem Paper zwei unterschiedliche Methoden bzw. Operatoren (Im Programm **Luminance Mapping** und **Hdrw Reinhard** genannt). In dieser Arbeit erweitern wir diese Operatoren auf 3-dimensionale Volumendaten und passen den Algorithmus auch sonst für medizinische Volumendaten an. In *Hdrw* ist der angepasste Algorithmus sowohl in einer **2D** Version (die Schicht für Schicht arbeitet) als auch in einer **3D** Version implementiert. Da beide vom Prinzip her gleich funktionieren, beziehen sich die folgenden Erklärungen direkt auf die **3D** Version.

Da aber bisher das lineare Windowing nicht mathematisch eingeführt wurde und praktisch alle Windowing Methoden darauf in gewisser Weise aufbauen, beginnen wir zuerst hiermit.

3.2 Lineares Windowing

Die Idee des linearen Windowing wurde bereits in Kapitel 1.1, Seite 5 am Beispiel von CT Daten eingeführt. CT Daten arbeiten üblicherweise mit 12 Bit, dies entspricht 4096 verschiedenen Grauwerten. Der Grund dafür liegt darin, dass CT Daten meist in *Hounsfield Units* gemessen werden. Diese sind von -1024 bis +3071 definiert, wobei -1024 für Luft und 0 für Wasser steht (Rowlett 2000).

Handelsübliche Monitor und Grafikkarten benutzen dagegen das 24 Bit *RGB* Format, welches 8 Bit für Rot, Grün und Blau bereitstellt (und evtl. weitere 8 Bit für die Transparenz). Für Grau- bzw. Helligkeitswerte stehen damit nur 8 Bit zur Verfügung. Farbdrucker können dementsprechend natürlich auch 24 Bit *RGB* Bilder drucken, arbeiten dafür aber intern mit CMY oder CMYK (Zyan, Magenta, Gelb, Schwarz).

Auch wenn demnach zumindest 8 Bit für Grauwerte zur Verfügung stehen, liegen die Werte in der Praxis meist noch niedriger. Typische Geräte wie LCD Monitore, Tinten- und Laserdrucker verarbeiten zwar das 24 Bit *RGB* Format, stellen aber bei weitem nicht alle 16,8 Millionen verschiedenen Farben mit exakter Helligkeit, Farbton und Sättigung dar. Gerade bei Druckern werden vom Hersteller selbst dabei oft komplizierte Algorithmen im Druckertreiber mitgeliefert, die dieses Problem durch verschiedene Techniken lösen sollen. Ebenso werden Korrekturverfahren für Monitore angeboten. *Hdrw* kann als praktisch einsetzbares Programm dabei natürlich kaum auf spezielle Hardware Rücksicht nehmen. Dutzende verschiedene Hersteller haben tausende verschiedene Produkte auf dem Markt, die allesamt unterschiedliche Eigenschaften haben. Wir gehen daher davon aus, dass zumindest die oben genannten 8 Bit für Grauwerte zur Verfügung

stehen. Bei der eigentlichen Umsetzung des Bildes auf dem Bildschirm oder auf dem Drucker müssen wir auf den Treiber des Hersteller vertrauen. Bisher liefern hier herkömmliche Röhrenmonitore immer noch das beste Preis-Leistungs-Verhältnis.

Das lineare Windowing von 12 Bit auf 8 Bit ist sehr einfach und besteht im Grunde im Vernachlässigen der unteren 4 Bits mit der niedrigsten Wertigkeit. Wie bereits in früheren Kapiteln angesprochen wurde, geht dabei Genauigkeit verloren. Mathematisch ausgedrückt sieht dies folgendermaßen aus:

$$L_w(x,y,z)\cdot\frac{1}{(2^{b_S}-1)}\cdot(2^{b_T}-1) \tag{6}$$

Dabei bezeichnet L_w (World Luminance) die Helligkeit des jeweiligen Pixels (x,y) bzw. Voxels (x,y,z). b_S (**Bits Source**) ist die Anzahl der Bits der Daten, b_T (**Bits Target**) die Anzahl der Bits des Monitors.

Im Programm ist L_w ein Integer (8, 16, 32 Bit) oder ein Float (32, 64 Bit). Da die Berechnung selbst auf beliebigen Datentypen ablaufen können muss, arbeitet *Hdrw* hier mit 64 Bit Floats. Beim Übertragen des Ergebnisses von beispielsweise 64 Bit Float zurück auf 8 Bit Integer, wird aus Geschwindigkeitsgründen ein normaler Typecast verwendet, der den Wert grundsätzlich abrundet (Ähnlich zum vernachlässigen der unteren Bits von oben). Dies führt zu einem maximalen Fehler von einer Helligkeitsstufe, der für den Menschen im Normalfall nicht wahrnehmbar ist, selbst wenn der Monitor dies auflösen könnte. Denn nach dem CIE $L^*u^*v^*$ Farbsystem kann mehr Mensch ungefähr 150 Helligkeitsstufen L^* wahrnehmen, im Vergleich zu den 256 Stufen bei 8 Bit (Schumann, Müller, 2000, p.156).

3.2.1 Gamma-Korrektur

Beim linearen Windowing und auch bei den anderen Windowing Methoden kann eine Gamma-Korrektur hinzukommen. Diese arbeitet bei *Hdrw* auf Helligkeitswerten zwischen 0 und 1. Damit ergibt sich für das lineare Windowing Formel (7):

$$(L_w(x,y,z)\cdot\frac{1}{(2^{b_S}-1)})^{1/\gamma}\cdot(2^{b_T}-1) \tag{7}$$

Da die Gamma-Korrektur auf 64 Bit Daten aufwendig ist, wird Sie nur im Fall **Gamma** $\gamma \neq 1$ durchgeführt (Je nach Verfahren kann dies die Rechenzeit um 50 bis 100% erhöhen).

3.3 Der Hintergrund

Die Operatoren, die in Reinhard *et al.* (2002, p.267-276) definiert werden, haben Techniken aus der Photographie als Grundlage, welche schon seit über 150 Jahren eingesetzt und verfeinert werden.

Von Beginn an hatten die Photographen als Ziel möglichst realistische Bilder zu erzeugen, was aber durch die Einschränkungen von Dias und gedruckten Photos erschwert wurde. Wie auch ein Monitor können diese nicht die extremen Helligkeitsunterschiede widerspiegeln, die in der Natur oft auftauchen und für die unser Auge komplexe Adaptionsmechanismen entwickelt hat. Zum einen entstanden verschiedene photographische Techniken, die versuchten diese Probleme auszugleichen. Zum anderen wurde aber auch die Entwicklung der Photographien verbessert. Während der erste Ansatz eher künstlerische Aspekte berücksichtigt, basierte der zweite Ansatz auf technischen Aspekten, wie beispielsweise konkreten Messungen.

In den 40er Jahren des 20. Jahrhunderts versuchte Ansel Adams beide Ansätze zu verbinden. Er definierte dazu das so genannte *Zone System*. Um dieses System zu erklären, müssen wir an dieser Stelle zuerst einige wichtige Ausdrücke bzw. Schlüsselwörter einführen. Damit es im Vergleich zur Literatur zu keinen Missverständnissen kommt, benutzen wir hier die englischen Ausdrücke.

- **Zone:** Eine Zone deckt einen bestimmten Helligkeitsbereich in der Szene ab (*Scene Zone*). Auf einem gedruckten Photo entspricht dies einem Bereich eines bestimmten Reflektionsgrades (*Print Zone*). Jede Zone hat dabei die doppelte Helligkeit der vorangehenden Zone.
  Während es in der realen Welt sehr viele Scene Zones geben kann, so gibt es genau 11 Print Zones. Diese werden von 0 (Reines Schwarz) bis römisch X (Reines Weiß) durchnummeriert.

- **Middle-Grey (dt. Mittelgrau):** Dies beschreibt die Scene Zone, welche vom Photographen als Mittelgrau wahrgenommen wird. Dies ist damit natürlich ein subjektiver Eindruck. Diese Scene Zone wird typischerweise auf Print Zone V abgebildet.

- **Dynamic Range (dt. Dynamikbereich, Helligkeitsumfang):** In der Computergraphik ist die Dynamic Range eines Bildes üblicherweise das Verhältnis des größten Helligkeitswertes zum kleinsten Helligkeitswert. Im Zone System beziehen sich diese Werte genauer auf die Bereiche des Bildes, in denen Details erkennbar sind. Das heißt, dies ist auch wieder ein subjektiver Eindruck des Betrachters der Szene.
  Da sich die Helligkeit von einer zur nächsten Zone verdoppelt, ähnlich wie bei einer zusätzlichen Stelle einer Binärzahl, kann die Dynamic Range einer Szene auch als Differenz zwischen der höchsten und niedrigsten Scene Zone ausgedrückt werden. Ein einfaches Beispiel: Hat die höchste Scene Zone die Helligkeit $2^{12} = 4096$ und die niedrigste die Helligkeit $2^{7} = 128$, dann ist das Verhältnis $4096 / 128 = 32 = 2^{5}$ und die Differenz $12 - 7 = 5$. Wir verwenden im Folgenden die Differenz zur Angabe der Dynamic Range.

- **Key (dt. Schlüssel):** Einer der entscheidenden Werte im Zone System ist der *Key* der Szene. Er gibt an, ob die Szene subjektiv betrachtet eher hell, normal oder dunkel ist. Beispielsweise wäre eine Szene mit viel Schnee im Winter subjektiv hell und damit *high-key*.

- **Dodging-and-Burning (dt. Verdunkeln & Aufhellen):** Dies ist eine Technik, bei der während Entwicklung des gedruckten Bildes ein Teil des Bildes verdunkelt (Dodging) und ein Teil aufgehellt (Burning) wird. Anstatt also das Bild insgesamt gleich zu entwickeln, können so einzelne Bereiche in ihrer Helligkeit angepasst werden.

Das Zone System geht nun folgendermaßen vor: Zuerst wählt der Photograph eine Region oder Oberfläche im Bild, die er subjektiv für die Scene Zone mit *Middle-Grey* hält. Diese wird typischerweise auf die mittlere Print Zone V abgebildet, welche einem Reflektionsgrad des Drucks von 18% entspricht. Für ein helles Bild mit Schnee (*high-key*) wäre dies eher ein dunklerer Bereich des Bildes. Um die Auswahl zu erleichtern kann ein Papier mit solch einem 18% Grauton vor die Kamera gehalten werden

Danach wählt der Photograph Bereiche im Bild mit der höchsten und niedrigsten Scene Zone, um die *Dynamic Range* des Bildes bestimmen zu können, wie sie oben erklärt wurde.

Ist die Differenz zwischen der höchsten und niedrigsten Scene Zone höchstens 9, so kann mit der richtigen Wahl der *Middle-Grey* Zone garantiert werden, dass die komplette *Dynamic Range* der Szene auch auf dem Ausdruck sichtbar ist. Müssen mehr Scene Zones abgedeckt werden, so werden Zones mit hoher und niedriger Helligkeit auf die Print Zones 0 (Reines Schwarz) und X (Reines Weiß) abgebildet.

Abbildung 81: Abbildung von Scene auf Print Zones (Images\Schnaidt\SceneToPrintZones.png)

In **Abbildung 81** sehen Sie hierfür ein Beispiel. 15 Scene Zones werden auf die 11 verfügbaren Print Zones abgebildet. Es wird angenommen, dass die mittlere Scene Zone gerade die *Middle-Grey* Zone ist, was nicht immer der Fall sein muss.

Teilweise kann es erwünscht sein, dass die höchsten Scene Zones alle auf Weiß abgebildet werden, beispielsweise bei hellen Objekten wie der Sonne oder einer anderen Lichtquelle. Allerdings kann dies auch zum Verlust an Details führen, wenn dies nicht erwünscht ist. In diesem Fall setzt der Photograph die oben beschriebene *Dodging-and-Burning* Technik ein. Diese passt die Helligkeit bestimmter Bereiche lokal an und sorgt dafür, dass immer genug "Dynamik" im Bild vorhanden ist, um die Details wahrzunehmen.

Das Zone System stattet den Photographen also mit einer kleinen Anzahl von subjektiven Parametern aus, die er benutzen kann, um die Helligkeit des Ausdrucks an die wirkliche Helligkeit der Szene anzupassen: Die niedrigste, die mittlere und die höchste Scene Zone. Zusätzlich können in einem zweiten Schritt mit *Dodging-and-Burning* Bereiche des Bildes lokal angepasst werden.

Das Zone System hat sich in der Praxis bewährt und wird daher auch noch über 50 Jahre nach seiner Erfindung eingesetzt. Der folgende Algorithmus orientiert sich daher an diesem System und versucht die Ideen und Konzepte für das High Dynamic Range Windowing zu nutzen.

3.4 Der Algorithmus

Der Algorithmus teilt sich in zwei Stufen. Das *Luminance Mapping* entspricht der Anpassung der Grundhelligkeit aus dem Zone System des letzten Abschnitts. Darauf folgt ein *Dodging-and-Burning*.

Ziel ist dabei nicht eine 1:1 Umsetzung des Zone System, sondern die Umsetzung der grundlegenden Konzepte. Beispielsweise benötigt man eine Alternative zur Bestimmung der wichtigsten Scene Zones durch den Photographen, da dies bei digitalen Volumendaten oft nicht möglich ist (beispielsweise bei medizinischen CT Scans). Auf der anderen Seite kann ein Dodging-and-Burning auf digitalen Daten sehr viel genauer und automatisiert durchgeführt werden.

3.4.1 Luminance Mapping

3.4.1.1 Der Key des Volumens

Bereits im letzten Abschnitt wurde der *Key* einer Szene im Zone System definiert. Er gibt grundsätzlich an, ob eine Szene subjektiv betrachtet eher hell oder dunkel ist. Eine Szene im Winter mit Schnee hätte einen hohen *Key*, während eine Szene bei der Dämmerung oder in der Nacht einen niedrigen *Key* hätte. Als eine gute rechnerische Näherung für den *Key* wird der *Log-Average* der Helligkeitswerte eines Bildes betrachtet.

Der *Log-Average* ist im Deutschen besser bekannt unter dem Namen *geometrisches Mittel* und hat im Gegensatz zum *arithmetischen Mittel* (meist einfach nur Mittelwert genannt) den Vorteil, dass einzelne stark abweichende Werte weniger das Ergebnis beeinflussen. Der *Log-Average* ist mathematisch folgendermaßen definiert:

$$\bar{L}_w = \exp\left(\frac{1}{N} \cdot \sum_{x,y,z} \log(\delta + L_w(x,y,z))\right) = \left(\prod_{x,y,z} \delta + L_w(x,y,z)\right)^{1/N} \quad (8)$$

In Formel (8) bezeichnet L_w (World Luminance) die Helligkeit des Voxels (x,y,z), $\bar{L}_w$ den Log-Average der Helligkeit und N die Anzahl aller Voxel. Zur Berechnung wird üblicherweise die Summenform verwendet, während man in der Mathematik oft die Produktform als Definition findet.

Da $\log(0)$ nicht definiert ist, benötigt man den Wert δ, falls das Bild den Helligkeitswert 0 enthält, was meist der Fall ist. Noch deutlicher wird dies in der Produktform, da nur ein Helligkeitswert mit 0 bereits das komplette Produkt auf 0 setzen würde. Damit hätte man keinesfalls mehr den Mittelwert der Helligkeit. Reinhard schlägt an dieser Stelle daher $\delta = 0{,}00001$ vor, das heißt, zu jedem Helligkeitswert wird eine geringe Zahl addiert, die vernachlässigbar klein ist.

Während Reinhards Bilder nur eine geringe Anzahl von schwarzen Pixeln bzw. Voxeln enthalten, stellt sich diese Wahl aber als ungeeignet für medizinische Volumendaten heraus, bei denen meist der komplette Hintergrund schwarz ist. Auch hier macht die Produktform schnell das Problem deutlich: Während der Helligkeitswert $0{,}00001$ das komplette Produkt um 5 Größenordnungen verändert, so beeinflusst $1{,}00001$ das Produkt fast überhaupt nicht. Obwohl beide Werte für fast dieselbe Helligkeit stehen, wenn man ein medizinisches Volumen mit 4096 Helligkeitswerten zu Grunde legt, beeinflusst der erste Wert den Durchschnitt wesentlich stärker. Das Resultat wäre ein viel zu niedriger *Key*. Für unse

ren Algorithmus, der noch folgen wird, führt dies letztendlich zu einem zu hellen, überstrahlten Bild.

Um dieses Problem zu umgehen, wird Formel (8) folgendermaßen verbessert:

$$\bar{L}_w = \exp\left(\frac{1}{N} \cdot \sum_{x,y,z} \log(1+L_w(x,y,z))\right)-1 \;= \left(\prod_{x,y,z} 1+L_w(x,y,z)\right)^{1/N}-1 \qquad (9)$$

Anstatt δ sehr klein zu wählen, wird nun $\delta = 1$ gesetzt. Dadurch wird jeder Helligkeitswert im Bild um $+1$ angehoben. Dementsprechend wird am Ende vom berechneten *Log-Average* nun -1 abgezogen, um dies wieder auszugleichen. Enthält das Bild kaum schwarze Voxel, ergeben Formel (8) und (9) fast beinahe identische Werte. Erst bei vielen schwarzen Voxeln kommt es zu Abweichungen, die aber dann einen wesentlich besseren *Key* ergeben.

3.4.1.2 Der Key Value

Bei einer normal hellen Szene entspricht der *Log-Average* (als Mittelwert) gerade *Middle-Grey*. Wie im Zone System möchten wir nun *Middle-Grey* auf 18% Helligkeit abbilden. Auf einer Skala von 0 bis 1 entspricht dies $0{,}18$. Dazu berechnen wir Folgendes:

$$L(x,y,z) = \frac{a}{\bar{L}_w} \cdot L_w(x,y,z) \qquad (10)$$

Mit $L(x,y,z)$ wird die "skalierte Helligkeit" bezeichnet, sie liegt im Intervall zwischen 0 und 1 (Dies wird durch Formel (10) nicht garantiert, aber im Programm sichergestellt). Für eine normal helle Szene bildet $a = 0{,}18$ den *Log-Average* genau auf $0{,}18$ ab, wie wir dies beabsichtigt haben. In fast allen Fällen liefert dies gute Resultate.

Für besonders helle (*high-key*) oder dunkle (*low-key*) Szenen kann der Benutzer aber a variieren. Da dies ein subjektiver Eindruck ist und, wie erwähnt, keine Informationen vom Photographen zur Verfügung stehen, wie er die Szene in Wirklichkeit mit seinen Augen wahrgenommen hat, wäre dieser Wert nur schwer zuverlässig automatisch einstellbar. Die eigentliche Abschätzung der Helligkeit geschieht daher durch den *Key* bzw. *Log-Average* $\bar{L}_w$, wodurch man a meist bei $0{,}18$ belassen kann.

In *Hdrw* wird a mit *Key Value* bezeichnet und diese Bezeichnung wird auch im Rest dieser Ausarbeitung verwendet. Dieser Begriff wird benutzt, da der *Key* $\bar{L}_w$ nach der Skalierung der Helligkeit dem *Key Value* a entspricht. Um aber Verwechselungen zu vermeiden, werden wir nun nur noch den Begriff *Log-Average* für $\bar{L}_w$ verwenden.

3.4.1.3 Hohe Helligkeitswerte

Die meisten realen Photographien haben eine durchschnittliche *Dynamic Range*, aber einige sehr hohe Helligkeitswerte bei *Highlights* (dt. Glanzpunkten) oder Lichtquellen. Selbst für medizinische Aufnahmen ist dies nicht ungewöhnlich, wobei hier solche Highlights durch Störungen bei der Aufnahme entstehen können. Die Röntgentechnik eines CT-Scanners wird beispielsweise durch Metallplomben in den Zähnen beeinflusst, wie **Abbildung 82** zeigt.

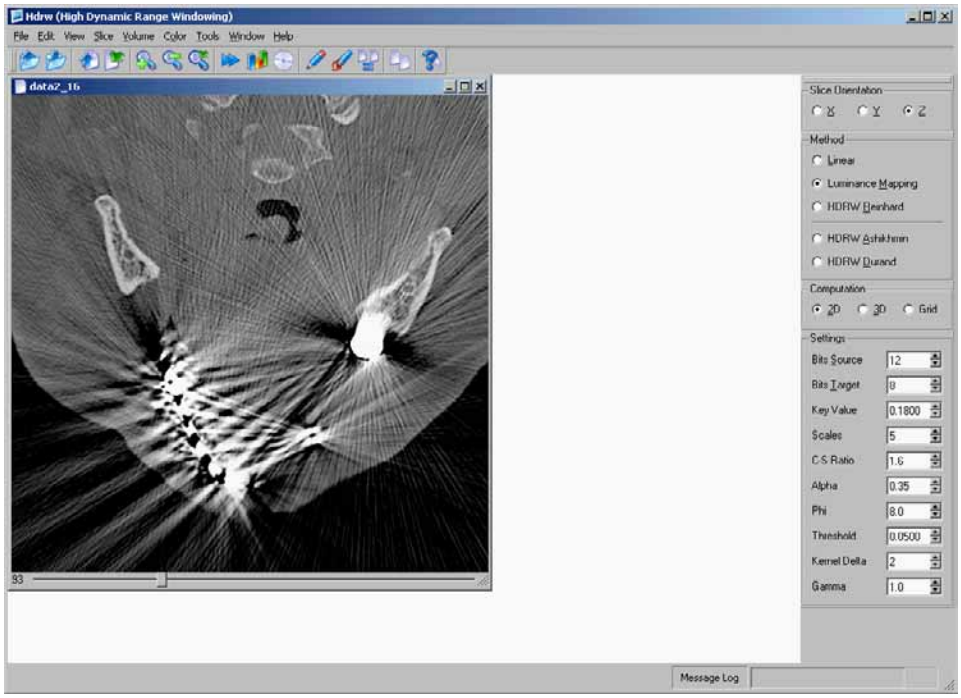

Abbildung 82: Metall-Artefakte (Images\Schnaidt\MetallArtefacts.png)

In der modernen Photographie werden daher meist Transferfunktionen verwendet, die hohe Helligkeitswerte komprimieren, wodurch mehr Platz und eine höhere Genauigkeit für die übrigen Helligkeitswerte bleibt.

$$L_d(x,y,z) = \frac{L(x,y,z)}{1+L(x,y,z)} \tag{11}$$

In Formel (11) werden hohe Helligkeitswerte nahe 1 etwa mit $1/L(x,y,z)$ skaliert, während niedrige Helligkeitswerte nahe 0 etwa mit 1 skaliert werden. Gleichzeitig gibt es einen kontinuierlichen Übergang zwischen beiden Extremen.

In *Hdrw* wird eine etwas erweiterte Formel benutzt:

$$L_d(x,y,z) = \frac{L(x,y,z)\cdot\left(1+\frac{L(x,y,z)}{L_{max}^2}\right)}{1+L(x,y,z)} \tag{12}$$

Reinhard definiert Formel (12) mit L_{white} statt L_{max} noch allgemeiner. Die Bedeutung hierbei ist, dass zusätzlich zu Formel (11) nun ein Weißpunkt L_{white} angegeben werden kann, über dem alle Helligkeitswerte auf reines Weiß abgebildet werden. Das heißt anstatt nur die hohen Helligkeitswerte zu komprimieren, werden Sie oberhalb ganz auf Weiß gesetzt (auch als *Burn-Out* bezeichnet). Solch einen Wert automatisch festzulegen ist aber riskant, da dadurch Details komplett verloren gehen können. Auf der anderen Seite fällt auch dem Benutzer eine Festlegung bei medizinischen Volumen oder gar künstlichen Datensätzen, wie sie in Kapitel 4 eingeführt werden, schwer. Daher übernehmen wir an dieser Stelle Reinhards Standardeinstellung, die L_{white} auf L_{max} (maximaler Wert aller L im Volumen) setzt. In diesem Fall geschieht kein *Burn-Out*, allerdings hat Formel (12) gegenüber Formel (11) dennoch einige andere Vorteile:

- Die Helligkeit wird oft leicht erhöht, was sich bei medizinischen Volumen positiv auswirkt.

- Nach Reinhard eine leichte Verbesserung des Kontrasts bei Bildern mit einer geringen *Dynamic Range*, genauer mit $L_{max}<1$.
- Rechnerisch ist L_d nun garantiert im Intervall zwischen 0 und 1, was eine If-Abfrage erspart (Moderne Prozessoren verarbeiten arithmetische Befehle fast immer schneller als If-Abfragen, die den Kontrollfluss des Programms beeinflussen).

3.4.1.4 Zusammenfassung

Damit werden beim **Luminance Mapping** folgende Formeln auf die Helligkeitswerte $L_w(x,y,z)$ des Volumens angewendet:

$$\bar{L}_w = \exp\left(\frac{1}{N}\cdot \sum_{x,y,z} \log(1+L_w(x,y,z))\right)-1 \tag{13}$$

$$L(x,y,z) = \frac{a}{\bar{L}_w}\cdot L_w(x,y,z) \tag{14}$$

$$L_d(x,y,z) = \frac{L(x,y,z)\cdot\left(1+\frac{L(x,y,z)}{L_{max}^2}\right)}{1+L(x,y,z)} \tag{15}$$

Da L_d im Intervall zwischen 0 und 1 liegt, ist nun wie beim linearen Windowing noch eine Skalierung mit b_T (Bits Target) nötig:

$$L_d(x,y,z)\cdot(2^{b_T}-1) \tag{16}$$

Ebenfalls wie beim linearen Windowing kommt ggf. noch eine Gamma-Korrektur hinzu (3.2.1, Seite 100).

3.4.2 Dodging-and-Burning

Das Luminance Mapping aus dem letzten Abschnitt stellt bei den meisten Datensätzen erfolgreich die Grundhelligkeit ein und komprimiert hohe Helligkeitswerte zu Gunsten der übrigen Helligkeitswerte. Aber wie beim *Zone System* reicht dies eventuell nicht aus, um in allen wichtigen Regionen des Bildes genügend Details zu erhalten.

Wie beschrieben, benutzt das *Zone System* an dieser Stelle ein Dodging-and-Burning. Das heißt, der Entwickler des Photographie passt manuell die Helligkeit in bestimmten Regionen des Bildes an. Solch eine manuelle Anpassung durch einen Menschen kann zwar zu einem maßgeschneiderten Bild führen, erfordert aber auch gleichzeitig viel Erfahrung und wäre bei medizinischen Volumen mit vielen Einzelbildern auch zeitlich viel zu aufwendig. Daher beschreiben wir im Folgenden einen Algorithmus, der das Dodging-and-Burning automatisch durchführt. Dies hat außerdem den entscheidenden Vorteil, dass jeder Voxeln lokal betrachtet und in seiner Helligkeit angepasst werden kann. Dadurch wird das Dodging-and-Burning auf digitalen Daten zu einem sehr mächtigen Werkzeug.

3.4.2.1 Die Faltung

Da der folgende Algorithmus eine Faltung benötigt, möchten wir an dieser Stelle kurz die wichtigsten Grundlagen hierzu einführen. Die Faltung von Bild- oder Volumendaten ist ein sehr umfangreiches Thema und kann hier leider nur kurz dargestellt werden. Falls Sie sich näher darüber informieren wollen, finden Sie in den meisten Computergraphik Büchern hierzu ein entsprechendes Kapitel. Um die Erklärung einfach zu halten, beschränken wir uns zunächst auf 1-dimensionale Bilder.

Stellen Sie sich ein 1-dimensionales Bild mit verschiedenen Grauwerten vor, die von links nach rechts abwechselnd heller und dunkler werden. Würde man den Grauwert wie in einem mathematischen Schaubild nach oben abtragen, so ergäbe sich eine Funktion. Grundsätzlich kann man daher jedes Bild und jedes Volumen als eine diskretisierte Form einer Funktion betrachten.

Die *Faltung* zweier Funktionen ist definiert als:

$$(f*g)(x) = \int_{-\infty}^{\infty} f(u)\cdot g(x-u)du \tag{17}$$

Normalerweise ist dabei $f(\cdot)$ die Funktion des Bildes bzw. Volumens und $g(\cdot)$ die Funktion des Filters. Allgemein verändert der Filter $g(\cdot)$ die Funktion $f(\cdot)$ in irgendeiner Weise, möglichst natürlich zu einem besseren Ergebnis hin.

Was bisher sehr abstrakt aussieht, ist an einem Beispiel wesentlich einfacher zu erklären: Eine typische Anwendung von Filtern ist das Glätten von Bildern. Der einfachste Filter hierzu ist der *Boxfilter* bzw. Mittelwertfilter. Wie sein Name schon vermuten lässt, bildet er Mittelwerte, wodurch das Bild glatter bzw. verwaschener erscheint.

Da man das Bild bzw. Volumen nicht als Funktion vorliegen hat, sondern mit diskreten Pixeln bzw. Voxeln, gibt man auch den Filter $g(\cdot)$ diskret an. Dies geschieht mit Hilfe eines so genannten *Filterkernels* (Filtermaske), in **Abbildung 86** sehen Sie hierfür ein Beispiel.

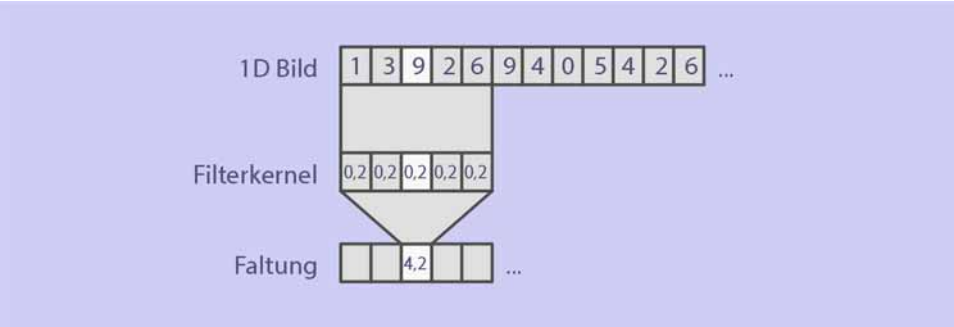

Abbildung 83: Die Faltung (Images\Schnaidt\Convolution.png)

Dieser Filterkernel hat eine bestimmte Ausdehnung (im Beispiel fünf Pixel) und läuft dabei von links nach rechts über das Bild. Im Beispiel ist der weiße Pixel mit der Nummer 9 der aktuelle Pixel im Bild. Die Faltung für ihn sieht folgendermaßen aus:

$$1 \cdot 0{,}2 + 3 \cdot 0{,}2 + 9 \cdot 0{,}2 + 2 \cdot 0{,}2 + 6 \cdot 0{,}2 = \frac{1+3+9+2+6}{5} = 4{,}2 \tag{18}$$

Der aktuelle Pixel 9 im Bild entspricht dabei der Mitte des Filterkernels, die direkt darunter liegt. Links und rechts von der Mitte hat der Filterkernel einige zusätzliche Pixel. Diese werden ebenfalls zusammen mit den darüber liegenden Pixel im Bild betrachtet. Zuerst wird jeder Pixel des Filterkernels mit dem darüber liegenden Pixel im Bild multipliziert. Anschließend wird dieses Produkt für alle Pixel des Filterkernels aufsummiert. In diesem Fall ist der Filterkernel gerade so gewählt, dass sich der Mittelwert der fünf Pixel des Bildes ergibt, wie Formel (18) zeigt.

Genauso fährt man für die anderen Pixel des Bildes fort und schiebt dabei den Filterkernel nach rechts weiter. Insgesamt ergibt sich ein geglättetes Bild.

Die Elemente des Filterkernels summieren sich dabei zu 1 auf: $5 \cdot 0{,}2 = 1$. Dies ist kein Zufall, denn wäre dies nicht gewährleistet, würde das Bild nach der Faltung heller oder dunkler erscheinen. Daher muss ein Filterkernel grundsätzlich *normiert* werden (Dies bedeutet gerade, dass sich die Elemente zu 1 aufsummieren).

Ein Boxfilter für 2D Bilder oder 3D Volumen sieht im Grunde genauso aus wie der 1D Filterkernel von oben. Im 2D Fall ist der Filterkernel ein Quadrat, beispielsweise mit 3×3 Einträgen und $1/(3 \cdot 3) = 1/9$ in jedem Feld. Im 3D Fall ergibt sich ein Würfel, beispielsweise mit 3×3×3 Einträgen und $1/(3 \cdot 3 \cdot 3) = 1/27$ in den Feldern. Immer wird der Mittelwert der Voxel gebildet, die durch den Filterkernel in den Daten abgedeckt werden.

Neben dem Boxfilter gibt es auch noch andere Glättungsfilter. Ein typischer Filter hierbei ist der *Gaußfilter*. Dieser bildet im Grunde auch eine Art Mittelwert. Er hat aber die besondere Eigenschaft, dass der aktuelle Pixel in der Mitte des Kernels am stärksten berücksichtigt wird und die Pixel zur Seite hin immer schwächer, je weiter sie von der Mitte entfernt sind. Allgemein ist ein Gaußfilter im n-dimensionalen Raum folgendermaßen definiert (Annemiek *et al.*, 2004, p.5):

$$g(\mathbf{v}) = \frac{1}{(\sqrt{2\pi} \cdot \sigma)^n} \cdot e^{-\frac{|\mathbf{v}|^2}{2\sigma^2}} \tag{19}$$

Dabei ist $\mathbf{v}$ ein n-dimensionaler Vektor, der vom Mittelpunkt des Kernels aus angegeben wird, und σ die Standardabweichung, mit der man die Stärke des Abfalls zu den Seiten festlegen kann. Ein Beispiel für einen 2D Gaußkernel, der als

Funktion in Mathematica aufgetragen wurde, sehen Sie in **Abbildung 84**. Er hat die Form einer Glocke und wird daher auch als Gaußglocke bezeichnet.

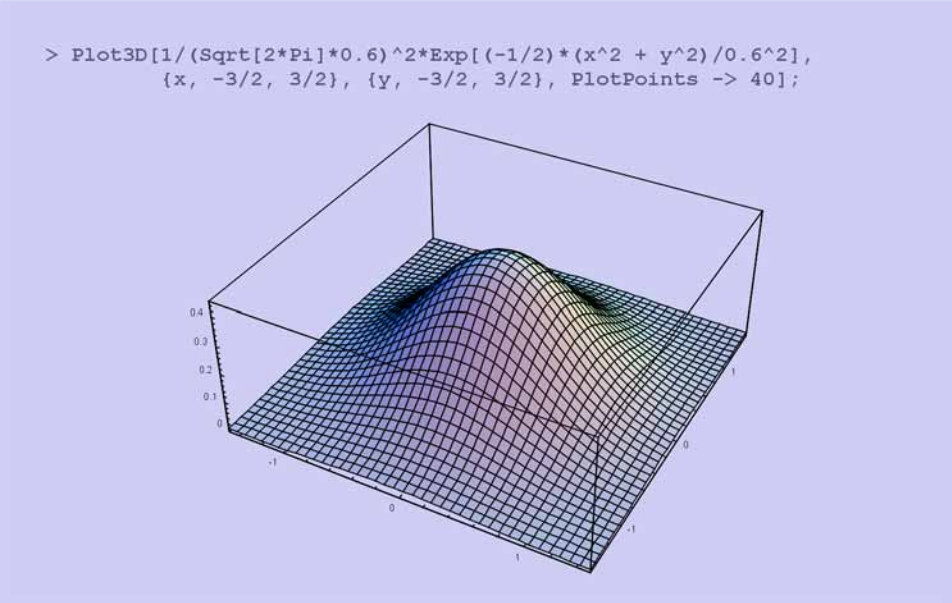

Abbildung 84: Gaußglocke (Images\Schnaidt\Gaussian.png)

Im 3-Dimensionalen kann man sich den Gaußkernel in etwa wie eine Kugel vorstellen. Innerhalb der Kugel liegt der stark berücksichtigte Bereich, alles außerhalb der Kugel wird nur noch schwach berücksichtigt.

Obwohl der Gaußkernel im kontinuierlichen Raum in etwa eine Kugel ist, muss auch er in einen diskreten 3×3×3 oder 5×5×5 würfelförmigen Kernel umgewandelt werden, damit eine Faltung wie im obigen Beispiel möglich ist. Ein typisches Beispiel hierzu zeigt **Abbildung 85**. Der diskrete Filterkernel ist dabei am besten gerade so groß, dass er die wichtigsten Bereiche des Gaußkernel abdeckt.

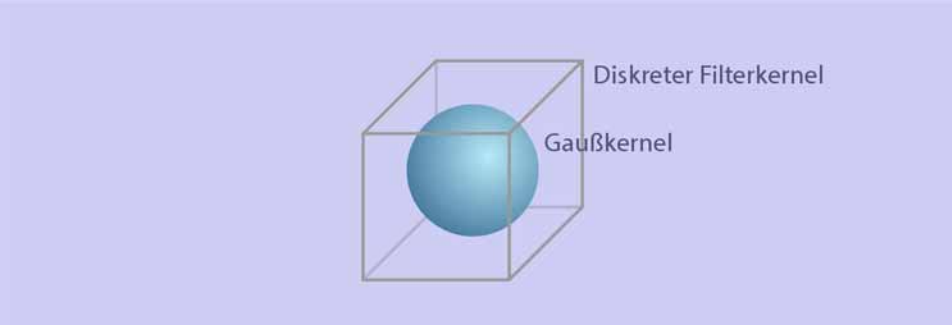

Abbildung 85: Gaußkernel als diskreter Filterkernel (Images\Schnaidt\DiscreteFilterKernel.png)

Der Vorfaktor vor der Eulerfunktion in Formel (19) dient zur Normierung. Allerdings kann dieser im Programm nicht zur Normierung benutzt werden, da er einen Filterkernel normiert, der kontinuierlich als Funktion gegeben ist und sich beliebig weit zu den Seiten ausdehnen kann. Hat man einen diskreten 3×3×3 oder 5×5×5 Filterkernel, wie er oben beschrieben wurde, muss man bei der Normierung anders vorgehen: Zuerst wird der Filterkernel ohne Vorfaktor berechnet. Dann werden alle Felder des Kernels aufaddiert und anschließend alle Felder durch diese Summe geteilt. Dann ist sicher gestellt, dass sich die Felder zu 1 aufaddieren.

3.4.2.2 Die Center-Surround Funktion

Dodging-and-Burning wird nach Adams Ansel auf eine ganze Region eines Photos angewendet. Die Größe und Form der Region wird dabei durch den Kontrast bestimmt. Innerhalb der Region soll der Kontrast möglichst ähnlich sein und am Rand der Region kommt es dagegen zu einem größeren Kontrastsprung. Ein Beispiel hierfür wäre ein dunkler Baum, der sich von einem hellen Hintergrund abtrennt. In diesem Fall würde also der Baum als einheitliche Region mit ähnlichem Kontrast betrachtet.

Algorithmisch wird dieser Ansatz so umgesetzt, dass man Regionen verschiedener Größe auf ein Kontrastmaß testet. Die verschiedenen Größen werden von Reinhard hierbei als *Scales* (dt. Ausmaße) bezeichnet. Die Idee ist beginnend von der kleinsten *Scale* jede *Scale* zu testen und diejenige zu wählen, die gerade noch eine Bildregion mit ähnlichem Kontrast abdeckt. Damit erhalten wir schließlich die Region, die Adams Ansel beschreibt.

Als erstes benötigen wir daher ein Kontrastmaß, mit dem wir die richtige *Scale* identifizieren können. Nach Reinhard werden hierfür oft so genannte *Center-Surround* Funktionen benutzt. Dabei wird der Datensatz mit zwei verschieden großen Gaußkernels gefaltet. Solch ein Gaußkernel kann man sich, wie im letzten Abschnitt erwähnt, im 3-Dimensionalen in etwa wie eine Kugel vorstellen. Die kleinere Gaußkugel entspricht dem *Center* (dt. Zentrum), die größere dem *Surround* (dt. Umgebung), siehe **Abbildung 86.** Dann werden die beiden gefalteten Datensätze voneinander abgezogen und damit die eigentliche *Center-Surround* Funktion berechnet.

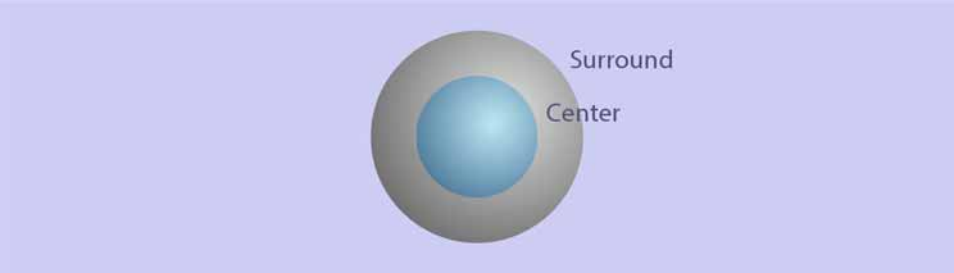

Abbildung 86: Center-Surround Funktion (Images\Schnaidt\CenterSurroundFunction.png)

An dieser Stelle gibt es verschiedene Ansätze in der Literatur. Reinhard wählte dabei die *Center-Surround* Funktion von Blommaert, die sich in seinen Tests als am besten geeignet herausstellte. Auch sie baut auf dem obigem Schema. Der kugelförmige Gaußkernel wird dabei folgendermaßen berechnet:

$$R_i(x,y,z) = \exp\left(-\frac{x^2+y^2+z^2}{(\alpha s^i)^2}\right) \tag{20}$$

i gibt die Nummer der getesteten Scale an und läuft von 0 bis **Scales**-1. s ist die so genannte **Center-Surround Ratio**. Sie gibt sozusagen das Größenverhältnis von zwei aufeinander folgenden Scales an. Zusätzlich kann man die Größe aller Scales noch mit α (**Alpha**) skalieren. **Scales**, **Center-Surround Ratio** und **Alpha** sind Optionen in *Hdrw*, die Sie einstellen können (Im Normalfall genügen aber die Standardeinstellungen).

Hat man mit obiger Funktion den Filterkernel berechnet und normiert, folgt die eigentliche Faltung:

$$V_i(x,y,z) = (L * R_i)(x,y,z) \tag{21}$$

Dabei ist $L(x,y,z)$ die skalierte Helligkeit, die in 3.4.1.2, Seite 104 definiert wurde. Insgesamt hat man nun für jeden Pixel und für jede Scale (um diesen Pixel) einen Wert $V_i(x,y,z)$.

Schließlich wird die eigentliche *Center-Surround* Funktion berechnet:

$$act_i(x,y,z) = \frac{V_{i-1}(x,y,z) - V_i(x,y,z)}{2^{\phi}a/(s^{i-1})^2 + V_{i-1}(x,y,z)} \tag{22}$$

Wobei hier i von 1 bis **Scales**-1 läuft. ϕ (**Phi**) ist ein zusätzlicher Wert, mit dem sich unter Umständen die Schärfe des Bildes anpassen lässt. a ist der bereits bekannte **Key Value** (Siehe 3.4.1.2, Seite 104). Auch hier sind **Scales**, **Phi** und **Key Value** wieder einstellbare Optionen in *Hdrw*.

Insgesamt wurden also für Schritt i zwei Faltungen mit zwei Gaußkernels (*Center* und *Surround*) berechnet, davon die Differenz gebildet und diese Differenz mit dem Nenner aus Formel (22) normalisiert. Damit erhalten wir die *Center-Surround* Funktion $act_i(x,y,z)$.

Wahrscheinlich haben Sie an dieser Stelle bereits bemerkt, dass der äußere Gaußkernel von Schritt i zum inneren Gaußkernel von Schritt $i+1$ wird. In obigen Begriffen wird der *Surround* von Schritt i zum *Center* von Schritt $i+1$. Daher wird a, welches das Größenverhältnis zwischen zwei aufeinander folgenden Scales angibt, oben als **Center-Surround Ratio** bezeichnet. Rechnerisch hat dies natürlich den Vorteil, dass man das Ergebnis der Faltung $V_i(x,y,z)$ wieder verwenden kann und damit Rechenzeit spart.

3.4.2.3 Die Auswahl der passenden Scale

Zurück zum Zone System: Unser Ziel war es, Regionen im Bild zu finden, die einen ähnlichen Kontrast haben und beim Dodging-and-Burning daher auch ähnlich behandelt werden sollen. Algorithmisch ist hierbei die Idee, dass man um den aktuellen Voxel im Zentrum Kugeln mit verschiedenen Radien legt. Ziel ist es dabei ausgehend von der kleinsten Kugel die Kugel zu finden, die gerade noch eine Region mit ähnlichem Kontrast abdeckt (Die nächst größere Kugel dagegen würde zuviel abdecken und bereits größere Änderungen des Kontrasts mit einschließen).

Die *Center-Surround* Funktion aus dem letzten Abschnitt kann genau diesen Zweck erfüllen. Ein Gaußkernel ist, wie erwähnt, im Prinzip einer Kugel nicht unähnlich. Das Ergebnis der Faltung $V_i(x,y,z)$ entspricht etwa dem Mittelwert der Voxel innerhalb der Gaußkugel mit Radius αs^i um den aktuellen Voxel (x,y,z). Die *Center-Surround* Funktion zieht nun zwei solche Kugeln unterschiedlicher Größe voneinander ab. Decken sowohl die kleinere als auch die größere Kugel noch gemeinsam eine Region mit ähnlichem Kontrast ab, so sind trotz der unterschiedlichen Größen die beiden Mittelwerte $V_i(x,y,z)$ und $V_{i-1}(x,y,z)$ noch sehr ähnlich. Die Differenz $|act_i(x,y,z)|$ ist dementsprechend klein.

Beginnend bei $i = 1$ wird nun $act_i(x,y,z)$ berechnet. Während man i erhöht, verlässt man irgendwann die Region mit ähnlichem Kontrast und der Mittelwert $V_i(x,y,z)$ wird sich plötzlich ändern. Dies führt dazu, dass $|act_i(x,y,z)|$ größer wird. Im Programm wird dies über einen **Threshold** (dt. Schwellwert) ε getestet. Ausgehend von $i = 1$ bis **Scales**-1 wird das kleinste i gesucht, welches die folgende Bedingung erfüllt:

$$|act_i(x,y,z)| > \varepsilon \tag{23}$$

Falls keine der getesteten Scales dies erfüllt, wird $i = $ **Scales** verwendet. Der Nenner von Formel (22) hat dabei folgende Funktion: $V_{i-1}(x,y,z)$ macht die *Center-Surround* Funktion unabhängig von absoluten Helligkeitswert, das heißt $|act_i(x,y,z)|$ ist ein relativer Wert. $2^{\Phi}a/(s^{i-1})^2+V_{i-1}(x,y,z)$ insgesamt verhindert Probleme bei sehr kleinen Werten $V_{i-1}(x,y,z)$, denn die kleinen Werte im Nenner würden zu einem fälschlicherweise großen $|act_i(x,y,z)|$ führen.

Wurde die passende Scale i gewählt, änderte sich also von $V_{i-1}(x,y,z)$ zu $V_i(x,y,z)$ plötzlich der Mittelwert. Damit ist $V_{i-1}(x,y,z)$ der Mittelwert über der Region mit ähnlichem Kontrast, die wir gesucht haben. Diesen Mittelwert können wir nun benutzen, um aus dem globalen Operator des **Luminance Mapping** den lokalen Operator der **Hdrw Reinhard** Methode zu machen:

$$L_d(x,y,z) = \frac{L(x,y,z)}{1+V_{i-1}(x,y,z)} \tag{24}$$

Genauso lässt sich auch diese Formel mit L_{max} erweitern. Dies ist die endgültige Fassung, wie sie in *Hdrw* implementiert ist.

$$L_d(x,y,z) = \frac{L(x,y,z)\cdot(1+\frac{L(x,y,z)}{L_{max}^2})}{1+V_{i-1}(x,y,z)} \tag{25}$$

Den Unterschied zwischen dem reinen **Luminance Mapping** und **Hdrw Reinhard** (mit Dodging-and-Burning) können Sie in **Abbildung 87** sehen.

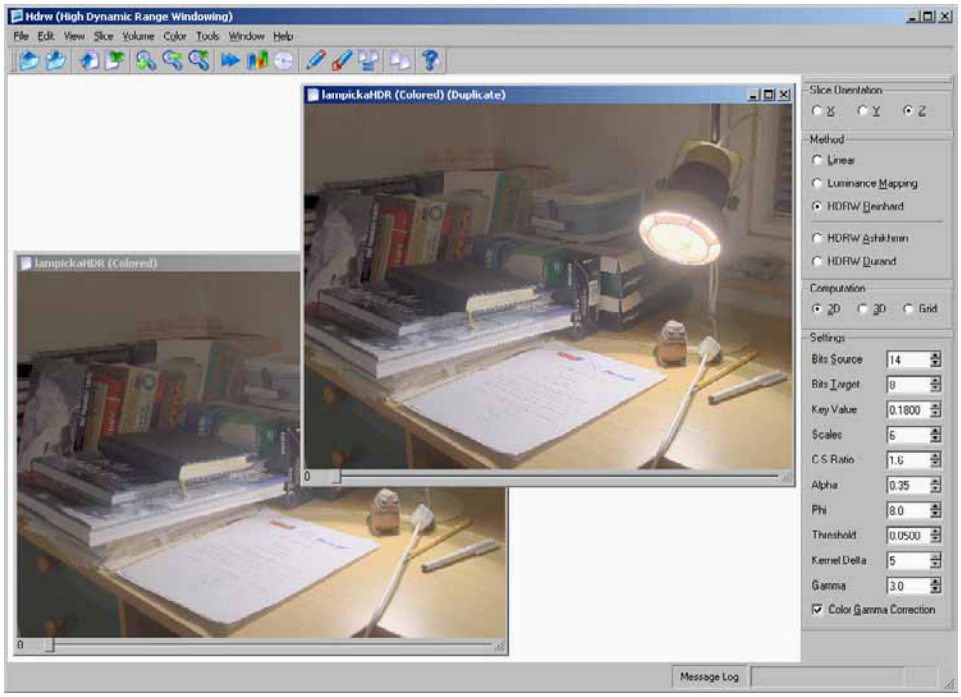

Abbildung 87: Hdrw Reinhard (Images\Schnaidt\HdrwReinhard.png)

Unten links die ursprüngliche Version mit **Luminance Mapping**. Im grellen Licht der Tischlampe wird der Text auf dem Papier mit aufgehellt und hebt sich nur noch geringfügig vom weißen Papier ab. Rechts oben mit Dodging-and-Burning, hier bleiben die Buchstaben deutlich besser erhalten[2]. Hier können lokal Details

[2] Da das Bild verkleinert wurde, bleibt der Text an sich natürlich immer noch unleserlich. Im Bilderarchiv *ReportImages.zip* finden Sie das Bild in Originalgröße.

des Bildes angepasst werden, so dass ein Teil der *Dynamic Range* bzw. Genauigkeit des Originalbildes erhalten bleibt.

In Formel (25) drückt sich dies so aus: Die schwarzen Buchstaben in einer hellen Umgebung sorgen dafür, dass der durchschnittliche Grauwert $V_{i-1}(x,y,z)$ wesentlich größer ist als Grauwert $L(x,y,z)$ des Buchstabens selbst. Da durch $V_{i-1}(x,y,z)$ geteilt wird, senkt sich die Helligkeit des Buchstabens ab. Das heißt dunkle Objekte in einer hellen Umgebung werden noch dunkler. Dies entspricht dem *Dodging*. Genauso werden helle Objekte in einer dunklen Umgebung noch heller (*Burning*).

Insgesamt wird also der Kontrast des Bildes erhöht. Dies unterscheidet sich aber wesentlich von einer herkömmlichen Kontrasterhöhung, die Sie aus Bildverarbeitungsprogrammen wie Adobe Photoshop kennen. Denn während die herkömmliche Kontrasterhöhung nur vorhandene Kontraste erhöht, werden hier die Informationen des Originalbildes mit seiner hohen *Dynamic Range* verarbeitet. In Verbindung mit dem Luminance Mapping führt dies dazu, dass weniger dieser *Dynamic Range* bzw. Genauigkeit des Originalbildes verloren geht. Ein lineares Windowing mit anschließender reiner Kontrastverstärkung könnte dies nicht leisten, da hier bereits die Informationen nach dem Windowing verloren sind.

Die Auswahl der passenden Scale mit dem **Threshold** ε ist dabei ein wichtiger Punkt des Algorithmus. **Abbildung 88** zeigt den Effekt, wenn der **Threshold** zu niedrig (links) und zu hoch (rechts) eingestellt wird. Im linken Bild wurde $\varepsilon = 0{,}0001$ verwendet, was dazu führt, dass grundsätzlich eine sehr kleine Scale ausgewählt wird. Diese Scale entspricht grob dem aktuellen Voxel selbst, wodurch sich in etwa die Formel und der Effekt des **Luminance Mapping** ergeben. Im rechten Bild dagegen würde $\varepsilon = 0{,}5$ gesetzt, was zu Auswahl von zu großen Scales führt. Dann entstehen deutliche Artefakte um beispielsweise Lichtquellen herum. Die richtige Wahl $\varepsilon = 0{,}05$ in der Mitte sorgt für eine Erhöhung des Kontrasts, ohne Artefakte ins Bild einzubringen.

Abbildung 88: Verschiedene Thresholds (Images\Schnaidt\Threshold1.png)

3.4.2.4 Zusammenfassung

Hier noch einmal die Formeln von **Hdrw Reinhard** auf einen Blick:

$$\bar{L}_w = \exp\left(\frac{1}{N}\cdot\sum_{x,y,z}\log(1+L_w(x,y,z))\right)-1 \tag{26}$$

$$L(x,y,z) = \frac{a}{\bar{L}_w}\cdot L_w(x,y,z) \tag{27}$$

$$R_i(x,y,z) = \exp(-\frac{x^2+y^2+z^2}{(\alpha s^i)^2}) \tag{28}$$

$$V_i(x,y,z) = (L * R_i)(x,y,z) \tag{29}$$

$$act_i(x,y,z) = \frac{V_{i-1}(x,y,z) - V_i(x,y,z)}{2^{\phi} a/(s^{i-1})^2 + V_{i-1}(x,y,z)} \tag{30}$$

$$|act_i(x,y,z)| > \varepsilon \tag{31}$$

$$L_d(x,y,z) = \frac{L(x,y,z) \cdot (1+\frac{L(x,y,z)}{L_{max}^2})}{1+V_{i-1}(x,y,z)} \tag{32}$$

$L_d(x,y,z)$ ist im Intervall zwischen 0 und 1 definiert. Da aber Formel (32) dies nicht garantiert, müssen Werte oberhalb von 1 abgefangen werden.

Anschließend ist wieder eine Skalierung mit b_T (Bits Target) nötig und ggf. noch eine Gamma-Korrektur (3.2.1, Seite 100):

$$L_d(x,y,z) \cdot (2^{b_T}-1) \tag{33}$$

3.5 Die Implementierung

Die Implementierung der Methoden **Luminance Mapping** und **Hdrw Reinhard** richtet sich weitestgehend nach der mathematischen und algorithmischen Beschreibung, auf die in 3.4 näher eingegangen wurde. Einige Berechnungen, insbesondere die Faltungen, sind aber rechnerisch aufwendig. Damit diese nur soviel Rechenzeit verbrauchen wie nötig, haben wir alle besonders rechenintensiven Teile des Algorithmus optimiert. In diesen Unterkapitel finden Sie eine nähere Beschreibung dieser Optimierungen und auch einen groben Überblick über die Implementierung.

Konkrete Details der Implementierung finden Sie direkt im Source Code in *HdrwMethodReinhard.cpp* kommentiert. In 1300 Zeilen Code (ungefähr 19 Seiten) werden sämtliche Formeln, Optimierungen und Details ausführlich erläutert (Sowohl in Kommentarform als auch in verständlichem C++ Code). Weitere Dateien mit nützlichen Informationen hierzu sind:

- *HdrwMethodReinhard.h*: Header Datei zu *HdrwMethodReinhard.cpp*
- *HdrwSliceTemplate.h*: Enthält das **Luminance Mapping**
- *HdrwMethodReinhardFFT.cpp*: **Hdrw Reinhard** im Frequenzraum
- *HdrwMethodReinhardGrid.cpp*: **Hdrw Reinhard** auf beliebigen Gittern

3.5.1 Luminance Mapping

Die Formeln des Luminance Mapping können direkt in der Form, die in 3.4.1.4, Seite 106 beschrieben wird, implementiert werden. Hierbei gibt es aber dennoch einige Besonderheiten, die auch auf die Methode **Hdrw Reinhard** zutreffen.

3.5.1.1 Berechnungen auf 64 Bit Floats

Hdrw arbeitet auf Volumen mit verschiedenen Datentypen (Siehe 2.3.5.9, Seite 54). Die Datentypen umfassen Integer (8, 16, 32 Bit) und Float (32, 64 Bit) Typen. In einer älteren Version des Programms arbeiteten wir grundsätzlich mit 64 Bit Float Volumen. Während dies grundsätzlich immer möglich ist, da dieser Datentyp die höchste Genauigkeit verwendet, verbraucht es sehr viel Speicher. Beispielsweise ein 16 Bit Volumen mit 80 MB verbraucht dann 320 MB. Dies kann die Verarbeitung verlangsamen oder gar zum Auslagern von Speicher in eine Datei auf der Festplatte führen (Wodurch sich die Speicherzugriffszeiten etwa vertausendfachen vom ns Bereich in den ms Bereich).

Hdrw legt daher die Daten automatisch im besten passenden Format im Speicher ab (D. h. im Format, das auch in der Volumendatei verwendet wird, aus der das Volumen geladen wurde).

Dennoch werden die eigentlichen algorithmischen Berechnungen weiterhin auf 64 Bit Floats durchgeführt. Der Grund dafür ist, dass unsere Algorithmen mit allen verschiedenen Datentypen und auch verschiedensten Volumendaten zurecht kommen müssen. Hat man aber konkrete Daten gegeben, die gewissen Anforderungen genügen, ließe sich dies für eine weitere Optimierung nutzen. Dabei wären folgende Szenarien denkbar:

- 32 Bit Float Werte sparen im Vergleich zu 64 Bit Float Werten zwar Speicher aber keine Rechenzeit auf aktuellen Prozessoren. Allerdings ließen

sich Befehlssatzerweiterungen (wie beispielsweise MMX, SSE, SSE2) dazu nutzen, um auf zwei 32 Bit Float Werten gleichzeitig zu rechnen.

- 16 Bit Integer Werte mit einer beschränkten Größe, wie beispielsweise 4096 Grauwerten bei CT Daten, ließen sich bei den globalen Methoden **Linear** und **Luminance Mapping** für alle 4096 Grauwerte vorberechnen und in einer Tabelle speichern. Dadurch würden beide Verfahren sehr schnell, da beim eigentlichen Windowing nichts mehr berechnet werden muss (Ein Nachschlagen in der Tabelle genügt).

- 16 Bit Integer Werte mit einer beschränkten Größe wären auch bei **Hdrw Reinhard** nutzbar. Die Werte könnten hoch skaliert werden, so dass die Faltung statt auf Floats zwischen 0 und 1 dann auf skalierten Integern arbeiten würde.

Da *Hdrw* aber keinerlei Einschränkungen an die Eingangsdaten stellt, sind diese Optimierungen nicht möglich und die Berechnungen werden mit 64 Bit Floats bei höchster Genauigkeit durchgeführt.

3.5.1.2 Divisionen auf 64 Bit Floats

Zu den teuersten arithmetischen Operationen gehören normalerweise Divisionen auf 32 oder 64 Bit Floats. Daher werden diese im Programm grundsätzlich vermieden, wenn dies möglich ist. Beispielsweise gibt die beiden folgenden Ansätze:

- Gleichungen können umgestellt werden, so dass sich nur noch Multiplikationen statt Divisionen ergeben. Dies ist beispielsweise bei Formel (30) der Fall.

- Werte werden oft in äußeren Schleifen mit $x' = 1/x$ vorberechnet, wodurch in der inneren Schleife nur ein $\cdot x'$ statt ein $/x$ benötigt wird.

3.5.1.3 Der Log-Average

Wie bereits erwähnt, ist das **Luminance Mapping** (unter Beachtung der letzten Abschnitte) fast genauso implementiert, wie dies in den mathematischen Formeln angegeben ist. Allerdings ist die Berechnung des *Log-Average* leider problematisch.

$$\bar{L}_w = \exp\left(\frac{1}{N} \cdot \sum_{x,y,z} \log(1+L_w(x,y,z))\right)-1 \tag{34}$$

In der in Formel (34) angegeben Form würde Sie mehr Rechenzeit benötigen als das eigentliche Windowing selbst. Der Grund dafür liegt bei der Logarithmus Berechnung $\log(\cdot)$, welche für jeden Voxel einmal durchgeführt wird und dann viel Zeit in Anspruch nimmt.

$$\bar{L}_w = \exp\left(\frac{1}{N} \cdot \sum_{x,y,z} \log(1+L_w(x,y,z))\right)-1 \;= \left(\prod_{x,y,z} 1+L_w(x,y,z)\right)^{1/N}-1 \tag{35}$$

Wie in Formel (35) gezeigt und in 3.4.1.1, Seite 103 bereits erwähnt wurde, lässt sich die *Log-Average* in eine Produktform umwandeln. Diese ist aber leider überhaupt nicht für den praktischen Einsatz geeignet, da das Produkt über alle Voxel des Volumens einen riesigen Wert ergeben würde, der auch die 64 Bit Float Genauigkeit bei Weitem überschreitet.

Obwohl beide Formeln einzeln ungeeignet sind, lassen sie sich erfolgreich kombinieren. Dazu wird die einfache Rechenregel $\log(a)+\log(b) = \log(a \cdot b)$ eingesetzt. Grundsätzlich beginnt *Hdrw* mit der Produktform. Es werden zuerst einige Voxel multipliziert. Erst danach wird auf das Produkt der Logarithmus angewendet. Nach der Rechenregel ist dies aber identisch mit der Summe über die einzelnen Logarithmen:

$$\log(a \cdot b \cdot c \cdot d \cdot e \cdot ...) = \log(a)+\log(b)+\log(c)+\log(d)+\log(e)+... \quad (36)$$

Dies lässt sich damit in die Summenform aus Formel (35) einfügen.

Entscheidend ist hier die Frage, wie viele Voxel multipliziert werden sollen. Bei beliebigen Eingangsdaten können bereits wenige Voxel einen hohen Wert liefert. Sehr kleine Werte sind zumindest durch das $1+$ in Formel (35) ausgeschlossen. Daher wird die Multiplikation solange durchgeführt, bis eine bestimmte Konstante (ein sehr großer Wert) erreicht worden ist, erst dann folgt die Logarithmus Berechnung.

64 Bit Floats haben können Werte bis $1{,}7 \cdot 10^{308}$ mit mindestens 15 Stellen Genauigkeit speichern. Unsere Konstante liegt bei $1 \cdot 10^{100}$. Was sich zunächst wie ein sehr großer Wert anhört und gravierende Abweichungen im Endergebnis befürchten lässt, stellte sich bei unseren Tests sogar als sehr sicher heraus.

	Zeit (s)	Konstante	*Log-Average*
Volume 1	11,166	0	345,203
Volume 1	1,893	$1 \cdot 10^{100}$	345,203
Volume 1	1,712	$1 \cdot 10^{300}$	345,203
Volume 2	21,561	0	349,068
Volume 2	3,605	$1 \cdot 10^{100}$	349,068
Volume 2	3,275	$1 \cdot 10^{300}$	349,068
Volume 3	3,665	0	64,9478
Volume 3	0,600	$1 \cdot 10^{100}$	64,9478
Volume 3	0,541	$1 \cdot 10^{300}$	64,9478
Volume 4	0,641	0	53,2591
Volume 4	0,111	$1 \cdot 10^{100}$	53,2591
Volume 4	0,110	$1 \cdot 10^{300}$	53,2591

Tabelle 1: Log-Average Berechnung mit verschiedenen Konstanten

Tabelle 1 zeigt die Ergebnisse unserer Tests. Wir verwenden als Konstanten 0 (Dies führt die Logarithmus Berechnung für jeden Voxel einzeln durch), $1 \cdot 10^{100}$ und $1 \cdot 10^{300}$. Selbst $1 \cdot 10^{300}$, welches nahe an der Genauigkeitsgrenze für 64 Bit Floats ist, liefert für die angegebene Genauigkeit von 6 Stellen keine Abweichung beim *Log-Average*. Da der *Log-Average* im Algorithmus als Schätzwert für die mittlere Helligkeit verwendet wird, würde bereits eine Genauigkeit von 3 Stellen ausreichen.

$1 \cdot 10^{100}$ liefert eine deutliche Beschleunigung von etwa 500%, wie die Tabelle zeigt. Da die Beschleunigung bei $1 \cdot 10^{300}$ nur noch unwesentlich ist und $1 \cdot 10^{300}$ in machen Spezialfällen eventuell für Probleme sorgen könnte, haben wir uns in *Hdrw* für $1 \cdot 10^{100}$ entschieden.

3.5.2 Hdrw Reinhard

3.5.2.1 Der Grundalgorithmus

Auch die **Hdrw Reinhard** Methode wird prinzipiell genauso umgesetzt, wie der Algorithmus in Unterkapitel 3.4 beschrieben wurde.

Zuerst verdeutlichen wir noch einmal den Algorithmus an Hand von Pseudo Code. Die Formeln dazu finden Sie in der Zusammenfassung 3.4.2.4, Seite 113. Um den Pseudo Code verständlicher zu halten, verwendet er als Beispiel einen 5×5×5 Filterkernel.

```
Berechne den Log-Average des Volumens;
for (i von 0 bis YScales-1)
    Berechne den Filterkernel R_i (mit Normierung);
Lade 5 Schichten des Volumens (Für den 5×5×5 Kernel);
for (Alle Schichten z des Volumens)
{
    for (Alle Zeilen y des Volumens)
    {
        Lade den ersten 5×5×5 Kernel der Zeile;
        for (Alle Spalten x des Volumens)
        {
            Faltung V_0(x,y,z);
            for (i von 1 bis YScales-1)
            {
                Faltung V_i(x,y,z);
                if (|act_i(x,y,z)| > ε)
                    break;
            }
            Setzte einen Pixel in der Ausgabeschicht mit Hilfe
              von V_i-1(x,y,z);
            Update den 5×5×5 Kernel der Zeile inkrementell;
        }
    }
    Schreibe die Ausgabeschicht in das Ausgabevolumen;
    Update die 5 Schichten des Volumens inkrementell;
}
```

Grob ausgedrückt wird für jeden Voxel (x,y,z) die passende Scale i gesucht und mit Hilfe vom Ergebnis der Faltung $V_{i-1}(x,y,z)$ die Helligkeit des neuen Voxels an Position (x,y,z) bestimmt.

Wer die Formeln in 3.4.2.4, Seite 113 genau betrachtet, der wird sich wahrscheinlich fragen, an welcher Stelle die Umwandlung der Helligkeitswerte des Volumens $L_w(x,y,z)$ in die skalierten Helligkeiten $L(x,y,z)$ geschieht:

$$L(x,y,z) = \frac{a}{\bar{L}_w} \cdot L_w(x,y,z) \tag{37}$$

Da jeder Voxel bei der Faltung mit einem Element des Filterkernels multipliziert wird, ist es am effizientesten, den Faktor $a/\bar{L}_w$ zum Filterkernel zu multiplizieren (nach der Normierung). Außerdem benötigt man $L(x,y,z)$ noch beim Setzen der Helligkeit des neuen Voxels, d. h., hier ist genauso eine Multiplikation nötig.

3.5.2.2 Die Schichten

Im Pseudo Code wird außerdem das Laden der 5 Volumenschichten erwähnt, die für einen 5×5×5 Filterkernel benötigt werden. Wie in 3.5.1.1, Seite 115 bereits beschrieben, verwendet *Hdrw* grundsätzlich 64 Bit Floats für seine Berechnungen. Liegt das Volumen aber beispielsweise im 16 Bit Integer Datentyp vor, dann muss hierzu eine Konversion stattfinden. Diese Konversion kostet zwar nicht viel Zeit, aber dennoch macht ein mehrmaliges Durchführen keinen Sinn. Daher werden die aktuell benötigten 5 Schichten in separate Arrays geladen, die bereits im 64 Bit Float Datentyp gespeichert sind. Alle folgenden Zugriffe können dadurch ohne Konversion geschehen.

Genauso geschieht dies für die Ausgabeschicht, die erst als 64 Bit Float Array gespeichert und dann als Ganzes in das endgültige Volumen übertragen wird, welches im obigen Beispiel auch wieder den 16 Bit Integer Datentyp benutzen würde.

Geht man eine Schicht weiter, so werden 4 der 5 Volumenschichten des letzten Durchgangs erneut benötigt. Damit diese nicht mehrfach geladen werden oder umkopiert werden müssen, benutzt *Hdrw* eine Art Ringpuffer. Sind beispielsweise die Schichten 0-1-2-3-4 im Speicher und soll die nächste Schicht geladen werden, wird der Ringpuffer um eins nach links verschoben zu 1-2-3-4-0 und dann die neue Schicht geladen zu 1-2-3-4-5. Dies geschieht lediglich mit *Pointern* (dt. Zeigern), die eigentlichen Daten werden nicht verschoben.

3.5.2.3 Die Faltung

Die Faltung ist der kritischste Punkt im Algorithmus und zwar aus zwei Gründen:

- Die Faltung ist die innerste Schleife und verbraucht daher am meisten Zeit.
- Außerdem hängt von ihrer Genauigkeit das Ergebnis direkt ab.

Daher haben wir speziell bei der Faltung viele Tests und Überlegungen durchgeführt, bis der endgültige Algorithmus fest stand.

Zuerst zur Optimierung der Rechenzeit: Bei modernen Prozessoren wird die Geschwindigkeit einer Operation ganz entscheidend davon beeinflusst, ob die benötigten Werte im Cache des Prozessors oder im Speicher vorliegen. Ein typisches Verhältnis wäre eine Speicherzugriffszeit von 100 ns und eine Cachezugriffszeit von 1 ns (Hennessey, Patterson, 2003, p.390). Beide Werte sind in aktuellen Prozessoren niedriger, aber das Verhältnis bleibt meist ähnlich. Zusätzlich haben Prozessoren meist nicht nur einen sondern mehrere Caches (L1 Cache, L2 Cache, teilweise sogar L3 Cache).

5 Volumenschichten, wie Sie im Beispiel geladen werden, brauchen im 64 Bit Float Datentyp und einer typischen Größe von 512×512 Pixeln pro Schicht insgesamt bereits 10 MB Speicher. Dies ist weit mehr als übliche Prozessoren in ihren Caches zur Verfügung haben (Einige wenige Prozessoren haben einen so großen L2 oder L3 Cache, der dann aber normalerweise aus normalen Speicherbausteinen besteht und langsamer ist). Selbst eine einzelne Schicht mit 2 MB wird kaum im Cache wieder gefunden, wenn sie einmal komplett abgearbeitet wurde.

Auch wenn sich dies ungünstig anhört, ist dies für die Performance des Algorithmus weit weniger entscheidend als die Faltung. Da sie die innerste Schleife des Algorithmus ist und sogar mehrere Male pro Voxel mit denselben Volumendaten durchgeführt wird, muss hier auf jeden Fall dafür gesorgt werden, dass sie im Cache durchgeführt - möglichst im L1 Cache.

Daher kopieren wir den aktuell benötigten 5×5×5 Bereich des Volumens, der für die Faltung benötigt wird, in einen separaten Array. Zusammen mit den typischerweise 5 verschiedenen Filterkernels R_i werden hierfür $5 \cdot 5 \cdot 5 \cdot (5+1) \cdot 8 \approx 6$ KB benötigt. Typische aktuelle Prozessoren wie der Intel Pentium III, der Intel Pentium IV und der AMD Athlon arbeiten mit 8 bis 16 KB L1 Cache (Hennessey, Patterson, 2003, p.505). Auch wenn damit ein *Cache Hit* (dt. Cache Treffer) im L1 Cache nicht zwangsweise garantiert ist, stehen die Chancen dafür recht gut.

Ist der 5×5×5 Array mit den Volumendaten für die Faltung geladen, muss die Faltung selbst möglichst effizient durchgeführt werden. Daher ist im Source Code die 5×5×5 Faltung komplett ausgeschrieben, was dem Compiler maximalen Spielraum für Optimierungen lässt. Die übrigen Faltungen werden mit Schleifen abgedeckt und sind dadurch leicht langsamer.

Ähnlich wie bei den Schichten im letzten Abschnitt, muss der 5×5×5 Array mit den Volumendaten für die Faltung beim nächsten Voxel nicht komplett neu geladen werden. Auch hier aktualisieren wir den 5×5×5 Array inkrementell mit 5×5 neuen Voxeln. **Abbildung 89** zeigt die Aktualisierung des Arrays mit einer neuen Schicht, während der Kernel auf der Zeile nach rechts wandert.

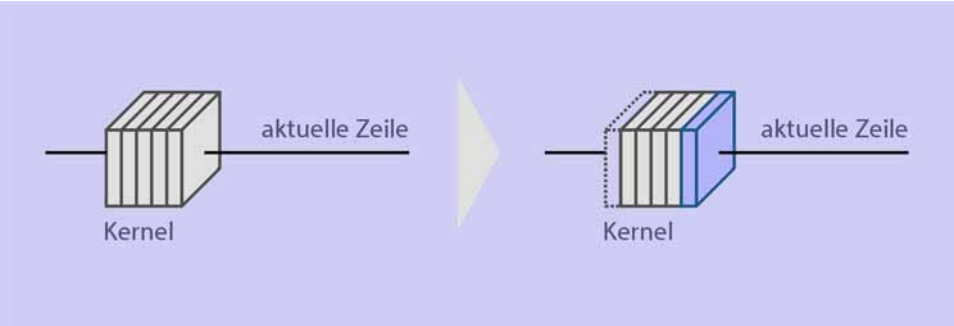

Abbildung 89: Update des 5×5×5 Array (Images\Schnaidt\ConvolutionArray.png)

Damit hier keine Daten kopiert werden müssen, benutzen wir wieder einen Ringpuffer. Hätte der Kernel in **Abbildung 89** links die Schichten 0-1-2-3-4, so würden die Schichten um ein links verschoben zu 1-2-3-4-0 und die neue Schicht rechts geladen (1-2-3-4-5).

Während das Schema relativ einfach ist, musste bei der Umsetzung im Source Code leicht anders vorgegangen werden. Stellen Sie sich vor, Sie möchten zu einem Volumen eine Schicht hinzufügen. Da das Volumen ein Quader ist, haben Sie dabei die Wahl zwischen drei verschiedenen Orientierungen der Schicht, d. h. eine weitere Schicht entlang der Z-Achse, der Y-Achse oder der X-Achse. Wer bereits mit Volumendaten gearbeitet hat, kommt an dieser Stelle schnell zum Schluss, dass das Hinzufügen entlang der Z-Achse am einfachsten ist. Dies liegt daran, dass Volumendaten im Prinzip als einzelne Bilder bzw. Blöcke im Speicher abgelegt sind, die nacheinander an der Z-Achse aufgereiht werden. Für eine neue Schicht in Richtung der Z-Achse benötigt man also nur einen neuen solchen Block. Nun wandert der Kernel in **Abbildung 89** aber entlang der Zeilen des Volumens, also der X-Achse, was ungünstig ist. Um dieses Problem zu lösen, benutzt *Hdrw* einen Trick: Für den 5×5×5 Array mit den Volumendaten und auch für die 5×5×5 Filterkernel werden die X- und die Z-Achse vertauscht. Damit entspricht eine neue Schicht auch wirklich einem neuen Block im Speicher.

Nun zur Genauigkeit der Faltung: Die Faltung selbst wird auf 64 Bit Float Daten ausgeführt und ist damit sicherlich genau genug. Allerdings betrachten wir an dieser Stelle die Genauigkeit bei der Berechnung der Filterkernel. Noch einmal zur Wiederholung die Formel hierfür:

$$R_i(x,y,z) = \exp(-\frac{x^2+y^2+z^2}{(\alpha s^i)^2}) \tag{38}$$

Auch diese Formel lässt sich problemlos genau genug mit 64 Bit Float Daten auswerten. Aber dabei wird nicht berücksichtigt, dass sie streng genommen nur für exakt einen Punkt des Raums gilt. Da wir aber mit diskreten Daten arbeiten, wie einem 5×5×5 Filterkernel, deckt jeder Voxel nicht genau einen Punkt ab, sondern einen kleinen würfelförmigen Raum um den Voxel. In **Abbildung 90** sehen Sie hierfür ein Beispiel.

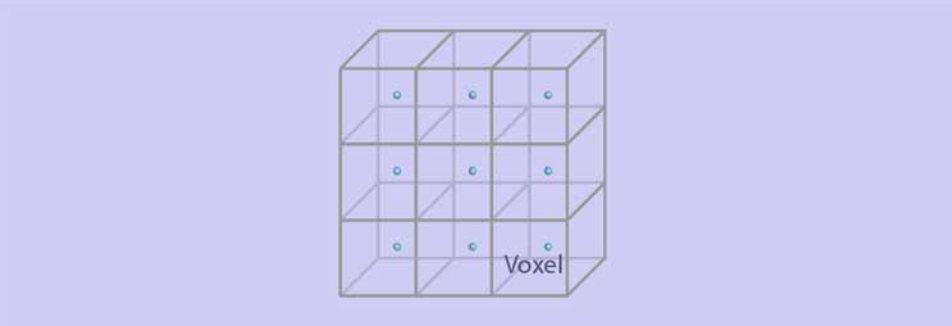

Abbildung 90: Raum um einen Voxel (Images\Schnaidt\Voxel.png)

Die Grundidee ist nun den Gaußkernel über diesen Raum aufzuintegrieren und dann durch sein Volumen zu teilen. Damit gilt $R_i(x,y,z)$ nicht mehr für exakt einen Punkt, sondern ist der Mittelwert für den Raum, den der Voxel abdeckt. Das Volumen des Raums ist einfach 1, da wir den Raum für 1 Voxel abdecken. Die Integration kann mit Hilfe der *Error Function* $\mathrm{erf}(\cdot)$ berechnet werden. Sie ist folgendermaßen definiert:

$$\mathrm{erf}(x) = \frac{2}{\sqrt{\pi}} \cdot \int_0^x e^{-t^2} dt \tag{39}$$

Für Formel (38) heißt dies nun:

$$R_i(x,y,z) = (\mathrm{erf}(\frac{x-0,5}{\alpha s^i}) - \mathrm{erf}(\frac{x+0,5}{\alpha s^i})) \cdot (\mathrm{erf}(\frac{y-0,5}{\alpha s^i}) - \mathrm{erf}(\frac{y+0,5}{\alpha s^i})) \cdot (\mathrm{erf}(\frac{z-0,5}{\alpha s^i}) - \mathrm{erf}(\frac{z+0,5}{\alpha s^i})) \tag{40}$$

Wie in 3.4.2.1, Seite 107 beschrieben, wird dabei der Filterkernel danach noch normiert, daher fällt der Faktor $2/\sqrt{\pi}$ von selbst weg. Zusätzlich wird in *Hdrw* noch das Spacing des Volumen mit (x,y,z) verrechnet (Siehe 2.3.5.8, Seite 51).

3.5.2.4 Update: Die neue Faltung

Sozusagen in letzter Minute wurde von uns der zeitkritischste Teil des Algorithmus, die Faltung, angepasst. Alle bisherigen Formeln bleiben unverändert und auch der größte Teil der Implementierung, lediglich die Faltung selbst wurde angepasst. Aus Zeitgründen haben wir dies nicht in die bisherigen Kapitel eingebunden und gehen daher in diesem Abschnitt getrennt vom Rest darauf ein.

Bei **Hdrw Reinhard** im **3D** Modus liefert die neue Faltung auf einem Testvolumen einen beträchtlichen Speedup von über 300%. Noch entscheidender ist aber, dass sowohl im **2D** als auch im **3D** Modus die nötige Rechenzeit nun linear mit der Größe des Filterkernels wächst (anstatt quadratisch im **2D** bzw. kubisch im **3D** Modus). Während ein 11×11×11 Filterkernel im Vergleich zu einem 5×5×5 Fil

terkernel zuvor ungefähr die 10 fache Zeit benötigte, ist das Verhältnis nun etwa Faktor 2.

Der Grund dafür ist, dass Faltungen mit einem Gaußkernel *separable* (dt. trennbar) sind. Dies bedeutet, eine 3-dimensionale Faltung mit einem würfelförmigen 5×5×5 Kernel kann in drei 1-dimensionale Faltungen aufgespalten werden. Zuerst wird das Volumen mit einem 1×1×5 Kernel gefaltet, dann das Ergebnis mit einem 1×5×1 Kernel und dieses Ergebnis wiederum mit einem 5×1×1 Kernel.

Abgesehen von eventuellen Rundungsfehlern ist dabei das Ergebnis der Faltung völlig identisch. Auch in unseren praktischen Tests ergaben sich keinerlei Unterschiede bei den Bildern.

Wir möchten an dieser Stelle keinen kompletten mathematischen Beweis dafür liefern, da das Verfahren ausführlich getestet ist und auch in renommierten Implementierungen benutzt wird - beispielsweise vom bekannten Hersteller von Grafikkarten ATI (Mitchell, 2003, p.7). Man kann die Idee aber leicht an der Formel für den Filterkernel erkennen:

$$R_i(x,y,z) = \exp\left(-\frac{x^2+y^2+z^2}{(\alpha s^i)^2}\right) = \exp\left(-\frac{x^2}{(\alpha s^i)^2}\right)\cdot\exp\left(-\frac{y^2}{(\alpha s^i)^2}\right)\cdot\exp\left(-\frac{z^2}{(\alpha s^i)^2}\right) \qquad (41)$$

Die Eulerfunktion lässt sich in drei Teile aufspalten, so dass sie für jede der drei Achsenrichtungen separat betrachtet werden kann. Dies trifft auch auf die Abwandlung mit der *Error Function* erf(·) zu (Siehe Formel (40)).

Der Pseudo-Code für den Algorithmus wird dabei folgendermaßen abgeändert (Wie bereits erwähnt, bleibt der eigentliche Kernalgorithmus bestehen und es ändern sich auch keinerlei Formeln):

```
Berechne den Log-Average des Volumens;
for (i von 0 bis YScales-1)
{
    Berechne den 1×1×5 Filterkernel R_i (mit Normierung);
    Berechne den 1×5×1 Filterkernel R_i (mit Normierung);
    Berechne den 5×1×1 Filterkernel R_i (mit Normierung);
}
Lade 5 Schichten des Volumens (Für den 1×1×5 Kernel);
for (Alle Schichten z des Volumens)
{
    for (i von 0 bis YScales-1)
    {
        Faltung der Schicht mit dem 1×1×5 Filterkernel;
        Erneute Faltung mit dem 1×5×1 Filterkernel;
        Erneute Faltung mit dem 5×1×1 Filterkernel;
                                                          }
    for (Alle Zeilen y des Volumens)
        for (Alle Spalten x des Volumens)
        {
            Lade V_0(x,y,z) aus dem Ergebnis der Faltung oben;
            for (i von 1 bis YScales-1)
            {
                Lade V_i(x,y,z) aus dem Ergebnis der Faltung oben;
                if (|act_i(x,y,z)| > ε)
                    break;
            }
```

```
                Setzte einen Pixel in der Ausgabeschicht mit Hilfe
                  von V_{i-1}(x,y,z);
            }
        Schreibe die Ausgabeschicht in das Ausgabevolumen;
        Update die 5 Schichten des Volumens inkrementell;
    }
```

Der wesentliche Unterschied besteht darin, dass die Faltung nicht mehr für jeden Voxel einzeln durchgeführt wird, sondern alle Faltungen für die aktuellen Schicht zuvor abgearbeitet werden. Dies geschieht dabei für alle Scales auf einmal (Im alten Algorithmus mussten nicht unbedingt alle Scales für jeden Voxel gefaltet werden, die Zeitersparnis daraus ist aber geringer als die des neuen Algorithmus).

Weiterhin geschieht die Umwandlung von normalen Helligkeitswerten in skalierte Helligkeitswerte über die Filterkernel (Siehe 3.5.2.1, Seite 118), genauer den 5×1×1 Filterkernel.

Im obigen Pseudo-Code wurde die Berechnungen der Faltung leicht vereinfacht dargestellt. Die eigentliche Faltung besteht aus mehreren verschachtelten Schleifen, die (1) über alle Scales, (2) über alle Pixel der Schicht und (3) über alle Pixel des Filterkernels hinweggehen. Die Reihenfolge der Schleifen (1), (2) und (3) kann dabei entscheidend für die Geschwindigkeit sein, da davon abhängt, ob Werte im Cache wieder gefunden werden. Daher wurde dies in unserem Algorithmus soweit optimiert wie möglich.

Noch einmal wird erwähnt, dass alle Windowing Resultate nicht von dieser Änderung betroffen sind. Ausschließlich die Rechenzeit wurde stark reduziert.

3.5.2.5 Standardeinstellungen

Die Standardeinstellungen wurden von uns von Reinhard übernommen. Nach Reinhard liefern Sie gute Ergebnisse auf einer Vielzahl von Bildern, was auch von uns bestätigt werden konnte, daher ließen wir die Werte weitestgehend unverändert.

- **Bits Source** : b_S ist vom Volumen abhängig
- **Bits Target** : $b_T = 8$
- **Key Value** : $a = 0{,}18$
- **Scales** : 5
- **C-S Ratio** : $s = 1{,}6$
- **Alpha** : $\alpha = 0{,}35$
- **Phi** : $\varphi = 8.0$
- **Threshold** : $\varepsilon = 0{,}05$
- **Kernel Delta** : 2
- **Gamma** : $\gamma = 1{,}0$

Sie liefern nicht zwangsweise auf allen Bildern optimale Resultate, aber dienen gleichzeitig auch als gute Anfangskonfiguration, wenn man die Einstellungen manuell nachregeln möchte.

Im Normalfall genügt eine Anpassung der beiden folgenden Werte:

- **Gamma:** Dies ist der einfachste Weg, um die Helligkeit des Bildes nachzuregulieren. Besonders bei Farbbildern macht meist ein höherer **Gamma** Wert wie beispielsweise $1{,}6$, $2{,}2$ oder $2{,}5$ Sinn. Wir belassen die Standardeinstellung bei $1{,}0$, da dieser Wert für medizinische Volume am besten geeignet ist. Gleichzeitig benötigt die Berechnung der Gamma-Korrektur gerade auf solch großen Volumendaten einen deutlich größeren Zeitaufwand. Da $1{,}0$ die Gamma-Korrektur praktisch "ausschaltet", wird Sie am besten erst dann eingeschaltet ($\neq 1{,}0$), wenn Sie wirklich benötigt wird.

- **Key Value:** Der Wert von *Middle-Grey* im Volumen (vor dem Windowing) wird nach dem Windowing auf den **Key Value** gesetzt. Der **Key Value** ist dabei ein Helligkeitswert zwischen 0 und 1. Die Standardeinstellung $0{,}18$ ist passend für die meisten Bilder. Bei besonders hellen Bildern (wie einer Schneelandschaft) oder besonders dunklen Bildern (wie einer Nachtaufnahme) können Sie den Wert anpassen. Ein höherer **Key Value** führt aber nicht zu einer reinen Helligkeitserhöhung, dies wird genauer in 2.4.4.3, Seite 85 erläutert.

Abweichend von Reinhards Konfiguration wurde **Scales** von uns auf 5 eingestellt. Eine genauere Erklärung hierzu finden Sie 3.5.3, Seite 125 (Dieser Wert wird zusammen mit **C-S Ratio**, **Alpha** und **Kernel Delta** passend eingestellt).

3.5.3 Die Größe des Filterkernels

Die Größe der Werte **Scales**, **C-S Ratio** s, **Alpha** α und **Kernel Delta** sind mit entscheidend für unseren Algorithmus. In vorangehenden Kapiteln wurde eher davon abgeraten, diese Werte manuell anzupassen. Dies ist zwar eigentlich nicht schwierig, aber da alle vier Werte voneinander abhängen, werden sie am besten gemeinsam angepasst.

Um die Hintergründe voll zu verstehen, wäre es am besten, wenn Sie den Abschnitt zur Faltung (3.4.2.1, Seite 107) selbst und auch die darauf folgenden Abschnitte zum Reinhard Algorithmus bereits gelesen haben. Ansonsten können Sie aber auch nur die hier gegebene Formel benutzen.

Die Faltungen in **Hdrw Reinhard** im **2D** und **3D** Modus arbeiten mit verschieden großen Gaußkernels. Solche Gaußkernels kann man sich grob wie Kugeln vorstellen. Dies bedeutet grundsätzlich, dass alle Voxel innerhalb der Kugel am meisten zählen und die umliegenden Voxel nur noch relativ wenig. Die eigentliche Faltung geschieht aber nicht mit einem kugelförmigen Gaußkernel, sondern einem würfelförmigen, diskreten Filterkernel - wie dies in der Computergraphik üblich ist. Dieser hat in den Standardeinstellungen die Größe 5×5×5, d. h., er deckt insgesamt 125 Voxel ab.

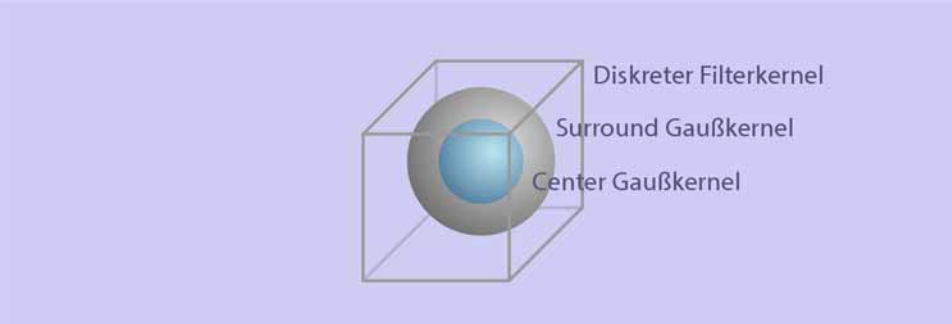

Abbildung 91: Gaußkernel und Filterkernel (Images\Schnaidt\CenterSurroundGaussians.png)

In **Abbildung 91** sehen Sie den würfelförmigen diskreten Filterkernel und zwei Gaußkernel. Die Reinhard Methode arbeitet dabei immer mit zwei Gaußkernels verschiedener Größe. Einem kleinen *Center* Gaußkernel und einem größeren *Surround* Gaußkernel. Die beiden Faltungen mit den beiden Kernels werden nacheinander durchgeführt und sozusagen die Differenz gebildet.

Nun arbeitet die Reinhard Methode aber zusätzlich noch mit verschiedenen Scales, bei denen die Gaußkernel noch größer werden. Dabei wird der *Surround* Gaußkernel der vorherigen Scale zum *Center* Gaußkernel der nächst größeren Scale (und der neue *Surround* Gaußkernel ist noch größer).

Da die Faltungen aber immer mit einem 5×5×5 würfelförmigen Filterkernel arbeiten, sollte die größte dieser "Gaußkugeln" in diesen Würfel passen.

Kernel Delta gibt dabei die Größe des würfelförmigen Filterkernels an, allerdings nur von der Mitte zur Seite (Ähnlich wie der Radius im Vergleich zum Durchmesser). Die Standardeinstellung ist 2, was einem Filterkernel entspricht, der **Kernel Delta**$\cdot 2+1 = 5$ Pixel breit ist. Im **2D** Modus hat man damit einen 5×5 Block, um **3D** Modus einen 5×5×5 Block.

Die größte Gaußkugel hat etwa den Radius von:

$$\alpha s^{j-1} \tag{42}$$

Mit **Alpha** α, **C-S Ratio** s und **Scales** j.

Insgesamt werden die Werte am besten so gewählt, dass die folgende Gleichung erfüllt ist:

$$\alpha s^{j-1} = \textbf{Alpha} \cdot (\textbf{C-S Ratio})^{\textbf{Scales}-1} \approx \textbf{Kernel Delta} \tag{43}$$

Für die Standardeinstellungen bedeutet dies:

$$0{,}35 \cdot 1{,}6^4 \approx 2{,}3 \approx 2 \tag{44}$$

Wie Sie sehen, ist die größte Gaußkugel etwas größer, was sich in der Praxis als günstiger Wert erwiesen hat.

Grundsätzlich kann man die Qualität des Algorithmus erhöhen, wenn man noch größere Gaußkugeln zulässt (**Scales** erhöht) und dafür auch den würfelförmigen Filterkernel vergrößert (**Kernel Delta** erhöht). Beispielsweise sind folgende Einstellungen nützlich:

- **Kernel Delta** 2 (5×5×5), **Scales** 5 (Standardeinstellung)
- **Kernel Delta** 5 (11×11×11), **Scales** 7
- **Kernel Delta** 10 (21×21×21), **Scales** 8

Die Rechenzeit wächst beim Erhöhen von **Kernel Delta** linear und ist damit in den Beispielen oben jeweils doppelt so groß wie in der vorherigen Einstellung (Sowohl im **2D** als auch im **3D** Modus).

3.5.3.1 Frequenzraum

In 3.4.2.1, Seite 107 wurde kurz darauf eingegangen, dass man Bilder oder Volumen auch als Funktionen betrachten kann.

Hier noch einmal ein Beispiel: **Abbildung 92** zeigt ein 1-dimensionales Bild, in dem die Grauwerte zwischen hell und dunkel variiert werden. Den Grauwert kann man stattdessen auch in einem Schaubild nach oben abtragen. Dadurch ergibt sich die Funktion im Bild. Natürlich ist das 1-dimensionale Bild dabei in Wirklichkeit diskret, d.h. es besteht aus einzelnen Pixeln. Dadurch wäre die Funktion in Wirklichkeit auch keine glatte Sinuskurve, sondern würde aus vielen kleinen Stufen bestehen.

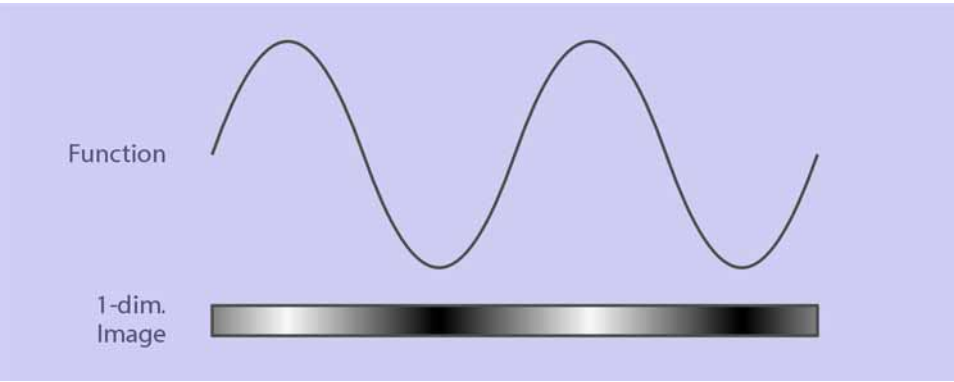

Abbildung 92: 1-dim. Bild als Funktion (Images\Schnaidt\Function.png)

Dieses Prinzip lässt sich auf 2-dimensionale Bilder und 3-dimensionale Volumen übertragen.

Die Funktion, die Sie in **Abbildung 92** sehen, kann auch wie ein elektrisches Signal oder ein Tonsignal betrachtet werden. Solchen Signalen wird eine Frequenz zugeordnet. Je höher die Frequenz ist, desto schneller schwingt das Signal hin und her. Im Bild entspricht dies einem schnellen Wechsel zwischen Schwarz und Weiß. Ein kontrastreiches Bild enthält viele solche schnellen Wechsel bzw. hohe Frequenzen.

Nun kann man ein ganzes Bild bzw. Volumen vom so genannten *Ortsraum* in den *Frequenzraum* übertragen. Der *Ortsraum* ist einfach der normale Raum, der auch bei der Anzeige auf dem Bildschirm verwendet wird. Im Grunde handelt es sich dabei um nichts anderes als den realen 3-dimensionalen Raum, in dem wir leben. Er wird üblicherweise mit den kartesischen Koordinaten (x,y,z) beschrieben. Wandelt man diesen Ortsraum komplett in einen Raum um, der nur noch Frequenzen anzeigt, gelangt man in den Frequenzraum. Hier werden statt (x,y,z) drei Frequenzen als Koordinaten verwendet.

Die Umwandlung geschieht dabei normalerweise mit der *Fourier Transformation*. Diese sieht für den 1-dimensionalen Fall folgendermaßen aus (Encarnação, Straßer, Klein, 1997, p.236):

$$\begin{aligned} F(\nu) &= \int_{-\infty}^{\infty} f(x) e^{-2\pi i \nu x} dx \\ f(x) &= \int_{-\infty}^{\infty} F(\nu) e^{2\pi i \nu x} d\nu \end{aligned} \tag{45}$$

Dabei ist $f(x)$ die Originalfunktion im Ortsraum und $F(\nu)$ die Fouriertransformation mit der Frequenz ν als 1-dimensionale Koordinate.

Wie bei der Faltung selbst kann man nicht diese kontinuierliche Formel für die diskrete Implementierung benutzen. Diese arbeitet daher mit der *Fast Fourier Transformation* (kurz *FFT*), die eine schnelle Umrechnung vom Ortsraum in den Frequenzraum bei diskreten Daten liefert.

Die effiziente Implementierung einer solchen *FFT* ist keineswegs trivial. In *Hdrw* wird daher die verbreite Bibliothek *fftw* (www.fftw.org) dafür verwendet.

Der Grund, weshalb solch eine Umrechnung nützlich ist, liegt in der Faltung. Sie ist die zentrale Komponente der Reinhard Methode und lässt sich im Frequenzraum wesentlich einfacher ausdrücken:

$$(f*g)(x) = \int_{-\infty}^{\infty} f(u)\cdot g(x-u)du \;\Leftrightarrow\; (f*g)(x) = F(\nu)\cdot G(\nu) \tag{46}$$

Die linke Seite von Formel (46) kennen Sie bereits aus 3.4.2.1, Seite 107. Im Frequenzraum ist das komplizierte Integral dagegen äquivalent zu einer einfachen Multiplikation.

Das Volumen f und der Filter g werden also zuerst mit der *FFT* in den Frequenzraum transformiert zu F und G. Dabei ergeben sich im Grunde auch wieder normale Volumendaten, die aber statt Grauwerten nun Frequenzen enthalten. Nun muss man nur noch jeden Voxel in F mit jedem Voxel in G einmal multiplizieren.

Auf den ersten Blick hört sich dies nach einer guten Methode an, den Algorithmus stark zu beschleunigen. In unserem Fall trifft dies aber leider nicht zu. Die Gründe dafür sind:

- Auch wenn die *fftw* Bibliothek eine sehr effiziente Implementierung ist, nimmt die Umrechnung eines ganzen Volumens in den Frequenzraum zuviel Zeit in Anspruch (im **3D** Modus). Außerdem müssen Filterkernel berechnet werden, die genauso groß wie das Volumen sind, was auch sehr zeitaufwendig sein kann. Genaueres hierzu folgt weiter unten.

- Noch ungünstiger ist aber, dass im **3D** Modus die Faltungen für alle Scales und dabei jedes Mal für das komplette Volumen vorberechnet werden müssen. Das heißt, von einem Volumen müssen viele Kopien angelegt werden und dies auch noch in hoher Genauigkeit (64 Bit Float). Bereits bei mittleren Volumen führt das schnell zu einer Überschreitung der 2 GB Speichergrenze. Dies macht den Einsatz auf 32 Bit Rechnern sogar unmöglich.

Wir haben das Verfahren dennoch implementiert, um einen Vergleich zu unserem Algorithmus ziehen zu können. Denn das Besondere an der Faltung im Frequenzraum ist, dass der Filterkernel keine beschränkte Größe wie 5×5×5 mehr hat, sondern das komplette Volumen abdeckt. Damit besteht das in 3.5.3, Seite 125 beschriebe Problem nicht mehr, dass eine Gaußkugel zu groß für den Filterkernel sein kann. Wir werden weiter unten aber zeigen, dass sich dadurch in der Praxis keine Vorteile ergeben.

Zuerst noch eine grobe Beschreibung unserer Implementierung mit *FFT*. Die **2D** und **3D** Implementierungen weichen dabei leicht in Details ab, arbeiten aber beide nach dem folgenden Schema:

- Zuerst werden die Filterkernels R_i für alle Scales berechnet. Die Filterkernel sind dabei im **2D** Modus so groß wie eine ganze Schicht und im **3D** Modus so groß wie das Volumen selbst. Die Berechnung geschieht dabei noch im Ortsraum und ist daher auch unverändert zu den bisher verwendeten Formeln (Siehe 3.5.2.3, Seite 119). Dann werden die Filterkernels normalisiert und in den Frequenzraum transformiert.

- Nun wird eine Schicht (**2D** Modus) oder das ganze Volumen (**3D** Modus) genommen und ebenfalls in den Frequenzraum transformiert.

- Für alle Scales werden nun das Volumen und der passende Filterkernel im Frequenzraum gefaltet. Dies geschieht durch einfache Multiplikation aller Voxel (Genau genommen handelt es sich um eine Multiplikation komplexer Zahlen mit Imaginär- und Realteil).

- Jetzt muss nur noch das Ergebnis der Faltung zurück in den Ortsraum transformiert werden - wieder für alle Scales. Danach arbeitet der Algorithmus völlig unverändert zur Implementierung ohne *FFT*.

Sie finden das Verfahren allerdings nicht in *Hdrw* direkt zur Auswahl. Die *FFT* Version kann durch das Flag **`HDRWREINHARDFFT=1`** in der Datei *Hdrw.pro* aktiviert werden. Nachdem *Hdrw* neu compiliert ist, können Sie die Methode **Hdrw Reinhard** im **2D** oder **3D** Modus wie gewöhnt benutzen und erhalten dabei die Ergebnisse der *FFT* Variante.

Wie bereits erwähnt, ist der praktische Nutzen der Implementierung eher gering. Wir wollen an dieser Stelle nur damit zeigen, dass ein Filterkernel in der Größe einer Schicht oder eines Volumens in der Praxis auch keine Verbesserung der

Bildqualität zur Folge hat. Wir testen dabei zwei Volumen sowohl für den **2D** als auch für den **3D** Modus und benutzen dabei maximale Genauigkeit (64 Bit Float Volumen).

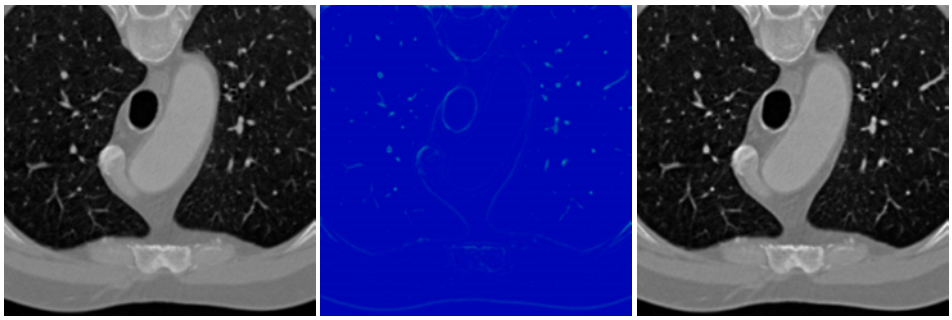

Abbildung 93: 2D, links ohne, rechts mit FFT, Mitte Differenz (Images\Schnaidt\FFT1-2D-1, 2, 3.png)

In **Abbildung 93** sehen Sie das erste Beispiel. Das Bild links verwendet die normale **Hdrw Reinhard** Methode im **2D** Modus, genauso das Bild **rechts** aber mit *FFT*, dazwischen die *absolute* Differenz beider Bilder. Das bedeutet beispielsweise bei einer Differenz von 10 Grauwerten zwischen beiden Bildern, dass diese Differenz auch als absoluter Wert "10" angezeigt wird - aber nicht als Grauwert, sondern zwischen blau-grün-rot moduliert. Der größte Teil des Bildes ist Blau, d.h. ohne Differenz. Einige wenige Teile sind leicht grünlich (Geringe Differenz). Dies liegt grundsätzlich daran, dass das linke Bild die Standardeinstellungen und damit einen 5×5×5 großen Filterkernel verwendet (**Kernel Delta** 2).

Diese Größe ist normalerweise ausreichend und die Unterschiede im letzten Beispiel sind auch gering. Mit einem größeren Filterkernel können aber auch diese Unterschiede beseitigt werden. **Abbildung 94** zeigt dasselbe Beispiel mit einem 11×11×11 Filterkernel (**Kernel Delta** 5) und **Abbildung 95** schließlich mit einem 21×21×21 Filterkernel (**Kernel Delta** 10). Bereits im ersten Fall sind die Unterschiede nur noch gering, im zweiten Fall fallen Sie ganz weg.

Abbildung 94: 2D, links ohne, rechts mit FFT, Mitte Differenz (Images\Schnaidt\FFT2-2D-1, 2, 3.png)

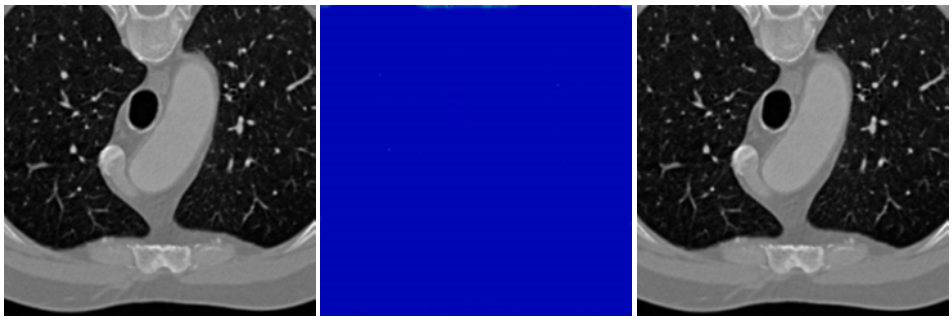

Abbildung 95: 2D, links ohne, rechts mit FFT, Mitte Differenz (Images\Schnaidt\FFT3-2D-1, 2, 3.png)

Um die Ergebnisse zu untermauern, haben wir einige weitere Tests durchgeführt. Die Abbildungen **Abbildung 96**, **Abbildung 97** und **Abbildung 98** benutzen alle die Standardeinstellungen, aber mit einem 21×21×21 Kernel (**Kernel Delta** 10). Dabei werden sowohl der **2D** als auch der **3D** Modus verglichen. Relativ zu den Standardeinstellungen benötigt der größere Kernel mehr Zeit, aber nicht sehr viel (Etwa die doppelte Zeit im **2D** Modus und die dreifache Zeit im **3D** Modus).

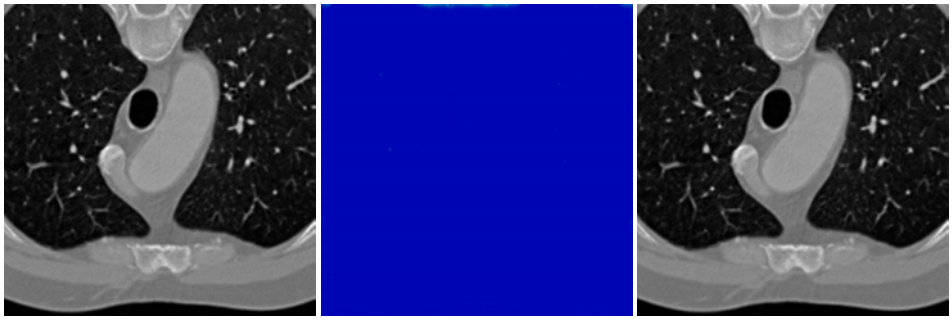

Abbildung 96: 3D, links ohne, rechts mit FFT, Mitte Differenz (Images\Schnaidt\FFT3-3D-1, 2, 3.png)

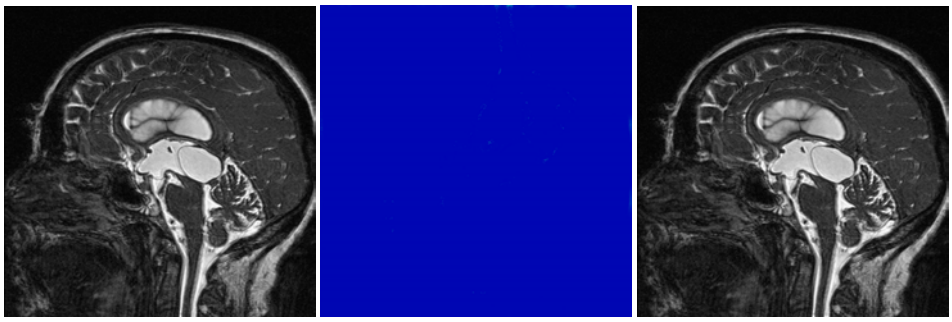

Abbildung 97: 2D, links ohne, rechts mit FFT, Mitte Differenz (Images\Schnaidt\FFT4-2D-1, 2, 3.png)

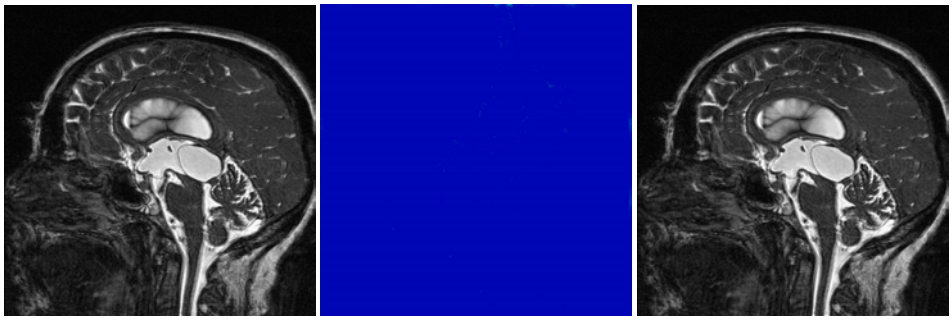

Abbildung 98: 3D, links ohne, rechts mit FFT, Mitte Differenz (Images\Schnaidt\FFT4-3D-1, 2, 3.png)

Es ergeben sich praktisch keine Unterschiede. Ganz wenige verstreute grünliche Pixel sind dabei wahrscheinlich auch eher auf Rundungsfehler bei der *FFT* Berechnung zurückzuführen, die nicht vollkommen verlustfrei arbeitet, wenn sie auf diskreten Daten durchgeführt wird.

Ganz an den Rändern sehen Sie einige größere Unterschiede. Diese sind aber normal, da beide Verfahren den Rand unterschiedlich behandeln. Ein 21×21×21 Filterkernel im Ortsraum benötigt eigentlich ein leicht größeres Bild, da sonst Teile des Filterkernels außerhalb des Bildes liegen, wenn er über den Rand geschoben wird. In *Hdrw* werden daher die Randpixel dupliziert, was normalerweise die beste Lösung hierfür ist.

3.5.4 Die Dateiformate

Hdrw unterstützt verschiedene Volumen- und Bildformate. Die Bildformate verwenden dabei bekannte Standards (*JPG* Format, *PNG* Format, ...). Die Volumenformate sind leider nicht soweit standardisiert, daher gehen wir in dieser Ausarbeitung genauer auf sie ein. Dies ist für Sie aber nur dann interessant, wenn Sie das Format in einem eigenen Programm auslesen möchten. Eine benutzerorientierte Einführung zu den Dateiformaten finden Sie stattdessen in 2.3.1.1, Seite 15 und 2.3.1.3, Seite 18.

In diesem Abschnitt wird das *SLC* Format beschrieben. Das *SCALAR* und *VECTOR* Format folgen in Kapitel 4, Seite 153, da diese für curvilineare Gitter geeignet sind.

3.5.4.1 *SLC* Format

Das *SLC* Format ist recht einfach aufgebaut und enthält grundsätzlich die Volumendaten in unkomprimierter Form. Bei Volumen mit 8 Bit pro Voxel kann *Hdrw* auch *RLE* komprimierte Volumen lesen, aber die Speicherung geschieht dabei wieder unkomprimiert (Den *Hdrw* unterstützt eine wesentlich bessere Kompression im *GZ* Format, die auf alle Dateitypen angewendet werden kann, siehe 2.3.1.1, Seite 15).

Zuerst beschreiben wir den Header einer *SLC* Datei. Er besteht grundsätzlich aus Zahlenwerten, die direkt im ASCII Format angegeben sind. Wir verwenden dabei **<Name,Datentyp>** , um ein einzelnes Feld mit solch einer Zahl zu beschreiben. Außerdem beschreibt **\n** einen Zeilenumbruch.

```
10101\n
<SizeX,int> <SizeY,int> <SizeZ,int> <BitsPerVoxel,int>\n
<SpacingX,double> <SpacingY,double> <SpacingZ,double>\n
<Unit,int> <Source,int> <Transformation,int> <Compression,int>\n
0 0 X
```

Wichtig ist dabei das Feld **BitsPerVoxel**. Es enthält die Anzahl von Bits, die pro Voxel verwendet werden. Es wurde von uns in *Hdrw* erweitert, um auch Volumen mit 32 und 64 Bit Float Genauigkeit speichern zu können. Es hat dabei folgende Bedeutung:

- 1..8 8 Bit Integer
- 9..16 16 Bit Integer
- 17..31 32 Bit Integer
- 32 32 Bit Float
- 64 64 Bit Float

Die Felder **Unit**, **Source**, **Transformation** und **Compression** sind Integer Werte, genauer vom **enum** Typ. Sie sind in *Hdrw* im Source Code folgendermaßen definiert:

```
enum EDataUnit { ZUnkownUnit = -1, ZMeter = 0, ZMillimeter = 1,
                 ZMicron = 2, ZFoot = 3, ZInch = 4 };
enum EDataSource { ZUnkownOrigin = -1, ZBioradConfocalData= 0,
                   ZMRData = 1, ZCTData = 2, ZSimulationData = 3 };
```

```
enum EDataTransformation { ZUnkownModif = -1, ZOriginalData = 0,
                           ZResampledData = 1 };
enum EDataCompression { ZUnkownComp = -1, ZNoComp = 0, ZRLE = 1 };
```

Direkt auf den eigentlichen Header folgen dann die unkomprimierten Daten mit der in **`BitsPerVoxel`** angegeben Größe pro Voxel (Das heißt nicht mehr im ASCII sondern im Binärformat). Dabei wird das *Big Endian* (*Most Significant Byte First*) Format verwendet.

3.5.5 Der Source Code

Wenn Sie sich für konkrete Details der Implementierung interessieren, können Sie direkt im Source Code von *Hdrw* nachschlagen. Alle Teile des Codes sind ausführlich dokumentiert und speziell auf die Algorithmen wurde besonders Wert gelegt. Insgesamt umfasst der Source Code etwa 24 000 Zeilen (Dies entspricht ca. 350 gedruckten DIN A4 Seiten).

In *HdrwDocumentation.h* finden Sie eine kurze Einführung, die Ihnen allgemeine Tipps zum Programm gibt (Beispielweise die Regeln für Variablen- und Funktionsnamen, die Vorgehensweise zum Erzeugen eines Builds unter verschiedenen Systemen, etc.).

Alle Klassen und Methoden sind in den Header Dateien speziell beschrieben. Diese eignen sich daher besonders gut zum Nachschlagen. Außerdem haben wir ein Tool benutzt, das diese Kommentare aus den Header Dateien extrahiert und in *HTML* Form zusammenfasst.

Die entsprechenden Verzeichnisse:

- **`/Source/Hdrw/`** (Kompletter Source Code)
- **`/Source/Code Comments 1.xx/index.html`** (Kommentare in *HTML*)

3.5.5.1 Kompatibilität

Beim Programmieren haben wir besonders auf die Kompatibilität von *Hdrw* Wert gelegt. Daher verwendeten wir den Framework *Trolltech Qt*, mit dem sich Programme gleichzeitig für verschiedene Systeme schreiben lassen.

Hdrw wurde unter folgenden Konfigurationen getestet ...

- Microsoft Windows XP (Intel IA-32 platform)
  - *Microsoft Visual Studio .Net 2003 Enterprise Architect*
  - *Qt 3.3.3 Enterprise Edition*
- SUSE Linux 9.1 (Intel IA-32 platform)
  - *gcc 3.3.3*
  - *Qt 3.3.3 Free Edition*

Zusätzlich haben wir auf die Kompatibilität zu *Big Endian* (*Most Significant Byte First*) und 64 Bit Maschinen geachtet. Da uns aus finanziellen Gründen dafür keine Test-Hardware zur Verfügung stand, können wir für Folgendes keine volle Garantie geben:

- *Hdrw.pro* ist sozusagen die Konfigurationsdatei für das Programm. Sie wird von Qt zum Erstellen der Builds verwendet. Hier können Sie das Flag **`MSB=1`** setzen, um das Programm auf *Big Endian* (*MSB*) umzustellen.
- Alle normalen Integer Werte sollten automatisch von 64 Bit statt 32 Bit Gebrauch machen, wenn diese zur Verfügung stehen. Dies schließt mehr adressierbaren Speicher und größere Dateien mit ein. Wir haben zum Test eine spezielle Einstellung des Microsoft Compilers verwendet, welche auch bei 32 Bit Builds Probleme mit der 64 Bit Kompatibilität testet.

3.5.5.2 Builds

Builds lassen sich am einfachsten in der Shell erstellen:

- Win32
  > qmake
  > nmake

- Unix
  > qmake
  > make

Wichtig ist allerdings, dass Sie zuvor einen Compiler für ihr System und *Trolltech Qt* (www.trolltech.com) installiert haben. Außerdem müssen die Umgebungsvariablen für *Qt* gesetzt sein - darauf werden Sie aber in der *Qt* Installation hingewiesen. Probleme bei älteren Compilern oder älteren Versionen von *Qt* können wir natürlich nicht ausschließen.

Außerdem können Sie auch das von uns geschriebene Tool *BuildMe* benutzen. *Binaries* dafür finden Sie direkt im Verzeichnis mit dem Source Code. Es kann allerdings sein, dass Sie *BuildMe* erst selbst compilieren müssen. Dies funktioniert auf genau dieselbe Weise wie oben angegeben. Der Source Code zu *BuildMe* findet sich unter `/Misc/BuildMe/` .

Wenn Sie beispielsweise *Microsoft Visual Studio .Net* oder den *KDeveloper* benutzen, können Sie dort natürlich auch entsprechende Projektdateien anlegen. Eine Anleitung hierzu finden Sie in *HdrwDocumentation.h* .

3.6 Die Resultate

Nachdem in den vorangegangen Unterkapiteln genau auf die Hintergründe der Reinhard Methode eingegangen wurde, wollen wir nun einige konkrete Beispiele betrachten und dabei die Unterschiede zwischen den verschiedenen Methoden hervorheben. Zuerst werden die Methoden **Linear** und **Luminance Mapping** verglichen, danach **Luminance Mapping** mit **Hdrw Reinhard**. Damit werden aber keineswegs alle Methoden von *Hdrw* abgedeckt. Die Vergleiche zu Volumen mit curvilinearen Gittern finden Sie in Kapitel 4, Seite 153. Ein Vergleich mit alternativen Windowing Methoden von *Ashikhmin* und *Durand* wird in Kapitel 5, Seite 187 aufgezeigt.

Die Methoden **Linear**, **Luminance Mapping** und **Hdrw Reinhard** liefern der Reihe nach eine bessere Qualität, aber brauchen dafür auch etwas mehr Zeit. Die **Tabelle 2** enthält einige Beispiele für den Zeitaufwand der Verfahren auf drei Volumen mit den Standardeinstellungen. Mit zunehmender Schichtanzahl werden die Volumen größer und damit steigt natürlich auch die Rechenzeit aller Verfahren. Die Messungen wurden dabei auf einem Intel Pentium IV 3,066 GHz (1 GB RAM) durchgeführt.

	Größe (MB)	Volumen	Methode	Zeit (s)
Data3	14,9	512×512×54	Linear	0,062 s
Data3	14,9	512×512×54	Lum. Mapping	0,094 s
Data3	14,9	512×512×54	Hdrw Reinhard 2D/3D	3,500 s / 7,094 s
Data1	51,3	512×512×168	Linear	0,171 s
Data1	51,3	512×512×168	Lum. Mapping	0,210 s
Data1	51,3	512×512×168	Hdrw Reinhard 2D/3D	14,125 s / 16,312 s
Data2	112	512×512×324	Linear	2,953 s
Data2	112	512×512×324	Lum. Mapping	4,266 s
Data2	112	512×512×324	Hdrw Reinhard 2D/3D	21,390 s / 30,485 s

Tabelle 2: Zeitaufwand bei drei verschiedenen Volumen

3.6.1 Linear vs. Luminance Mapping

In diesem Abschnitt werden die Methoden **Linear** (3.2, Seite 99) und **Luminance Mapping** (3.4.1, Seite 103) näher verglichen.

Während des lineare Verfahren die Helligkeitswerte von einem großen auf einen kleinen Bereich einfach linear abbildet, versucht das **Luminance Mapping** dabei zusätzlich die Grundhelligkeit des Volumens einzustellen. Außerdem werden hohe Helligkeitswerte komprimiert, wodurch mehr Raum für die übrigen Helligkeitswerte bleibt. Die Kompression hoher Helligkeitswerte ist sinnvoll, da sowohl reale Bilder als auch medizinische Volumendaten meist einige sehr helle *Highlights* enthalten, deren detaillierte Darstellung aber gar nicht nötig ist (Mehr dazu in 3.4.1.3, Seite 104).

Mit dem **Gamma** Wert lässt sich ebenfalls die Helligkeit des Bildes erhöhen. Daher werden Sie sich an dieser Stelle fragen, weshalb man die Grundhelligkeit nicht mit dem linearen Windowing und einer Gamma-Korrektur anpassen kann. **Abbildung 99** zeigt deutlich, dass dies nicht möglich ist. Links wurde das lineare

Windowing mit **Gamma** 10,0 eingesetzt, rechts das **Luminance Mapping** mit **Gamma** 2,5. Obwohl das linke Bild eigentlich einen noch höheren **Gamma** Wert benötigen wurde, kann man bereits deutlich sehen, dass das Bild ausbleicht. Generell liefern zu hohe **Gamma** Werte keine brauchbaren Resultaten mehr.

Abbildung 99: Linear (Gamma 10) vs. Lum. Mapping (Gamma 2,5) (Images\Schnaidt\LINvsLM-*-G*-1, 2.png)

Daher haben wir in diesem Abschnitt auf die Beispiele jeweils dieselbe **Gamma** Korrektur angewendet. Sie werden dabei aber sehen, dass die linearen Bilder in fast allen Fällen zu dunkel sind - je nach Datensatz variierend von viel zu dunkel bis etwas zu dunkel.

3.6.1.1 Die Vorgehensweise

Unsere Tests wurden so durchgeführt, dass sich für alle Verfahren die maximal mögliche Qualitätsstufe ergibt und damit keine Fehler durch beispielsweise Farbumrechnungen geschehen können. Dabei sind einige zusätzliche Arbeitsschritte nötig, die Sie aber beim normalen Windowing meist nicht brauchen:

- Zuerst werden die Bilder bzw. Volumen in den Datentyp 32 Bit Float umgewandelt (**Volume > Convert Data Type > To Floating-Point Number (64 Bit)**), dadurch geschehen sämtliche Berechnung in höchster Genauigkeit.

- Dann werden Farbbilder, die im *VECTOR* Format gespeichert wurden, in richtige Farbbilder umgewandelt (**Color > Convert Vector To Color**). Das bedeutet, dass die in der Datei gespeicherten Vektoren als *RGB* Farbwerte interpretiert werden.

- Dann lässt sich das entsprechende Verfahren einstellen. Farbbilder brauchen dabei meist einen erhöhten **Gamma** Wert und die **Color Gamma Correction** (Bei medizinischen Volumen dagegen eignet sich meist ein **Gamma** Wert 1,0, das heißt, die Gamma Korrektur ist ausgeschaltet).

- Schließlich wird das Windowing durchgeführt (**Volume > Start Windowing**).

Nur bei der Methode **Linear** ist noch ein zusätzlicher Schritt vor dem Einstellen der Optionen wichtig:

- In den Optionen von *Hdrw* können Sie den Bereich der Originaldaten mit **Bits Source** angegeben (Siehe 2.4.4.1, Seite 83). Beispielsweise CT Daten mit 4096 verschiedenen Grauwerten haben hier 12 Bits. *Hdrw* stellt diesen Wert dabei automatisch für Sie ein. Nun kann ein Volumen aber beispielsweise nur 3500 Grauwerte haben, was auch 12 Bits entspricht. Die übrigen Methoden werden hiervon nicht beeinflusst, aber **Linear** geht auch dann von 4096 Grauwerten aus, wodurch das Bild insgesamt zu dunkel erschei

nen kann. Mit einer Streckung des Histogramms (**Volume > Histogram > Stretching**) kann man dieses Problem aber lösen. Dabei werden die 3500 Grauwerte so skaliert, dass Sie den vollen Bereich von 4096 Grauwerten abdecken.

3.6.1.2 Farbbilder

Das erste Beispiel in **Abbildung 100** zeigt bereits sehr deutlich das Problem beim linearen Windowing (Links). Die Lampe selbst ist sehr hell und ihre Helligkeitswerte sind über einen weiten Bereich im Originalbild verteilt. Auf dem Bildschirm wird dieser Bereich auf nur 256 Werte verkleinert. Wie man sieht, bleibt dabei für den Rest des Bildes nicht mehr viel Genauigkeit übrig. Er scheint hier fast ganz Schwarz und mit einem höheren **Gamma** Wert in **Abbildung 99** nur sehr kontrastarm.

Abbildung 100: Linear vs. Lum. Mapping, Gamma 2,5 (Images\Schnaidt\Reinhard\Lampicka-*-K5-G25.png)

Mit dem **Luminance Mapping** (Rechts) wird das Bild insgesamt aufgehellt, wodurch man auch die Bücher im Hintergrund und außerdem die Schrift auf dem Papier unterhalb der Lampe erkennen kann. Die Lampe selbst bleibt zwar hell, aber weniger genau aufgelöst zu Gunsten des Rest des Bildes.

Abbildung 101: Linear vs. Lum. Mapping, Gamma 2,5 (Images\Schnaidt\Reinhard\Memorial-*-K5-G25.png)

Abbildung 101 zeigt prinzipiell dasselbe Verhalten. Nur wenige helle Bereiche bei den Fenstern werden beim linearen Windowing zum Problem, während das **Luminance Mapping** fast alle Bereiche des Bildes deutlich zeigt.

Abbildung 102: Linear vs. Lum. Mapping, Gamma 2,2 (Images\Schnaidt\Reinhard\Nave-*-K5-G22.png)

Genauso in **Abbildung 102**, wobei hier ein Problem des Luminance Mapping deutlich wird. Im linken Bild kann man erkennen, dass das Kirchenfenster viele verschiedenen Farben enthält. Im rechten Bild dagegen erscheint es fast Weiß wie ein normales Fenster. Auf dieses Problem gehen wir weiter unten noch einmal ein.

3.6.1.3 Medizinische Volumen

Medizinische Volumen haben normalerweise keine solch starken Highlights wie von Lichtquellen oder Fenstern, aber dennoch erscheint das rechte Bild in **Abbildung 103** heller und deutlicher.

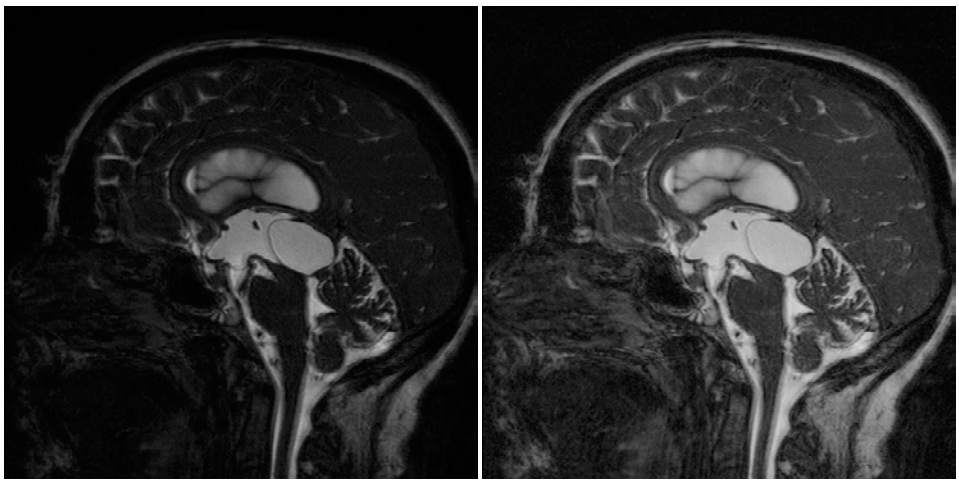

Abbildung 103: Linear vs. Lum. Mapping (Images\Schnaidt\Medical\Data3-29-*-K5.png)

Die meisten Bereiche mit Gehirn- und Weichgewebe sind im linken Bild bereits sehr dunkel und erscheinen relativ homogen. Mathematisch betrachtet bleibt für sie links nur einer kleinerer (dunkler) Grauwertbereich übrig, dadurch wird die Genauigkeit stärker reduziert und das Bild erscheint kontrastärmer.

Beim Volumen selbst handelt es sich um einen MRT Scan (Magnetresonanztomographie). Dabei wird mit einem starken Magnetfeld die Dichte von Wasserstoff im Körper gemessen. Besonders hell sind daher flüssigkeitsgefüllte Bereiche wie die Augen und das Ventrikelsystem (in der Mitte des Gehirns). Knochen dagegen erscheinen völlig schwarz.

Abbildung 104 zeigt einen CT Scan (Computertomographie). Dabei handelt es sich im Grunde um ein 3-dimensionales Röntgenverfahren. Speziell Knochen sind hier

hell, wie die Wirbelsäule ganz oben im Bild. Der Hauptteil des Bildes zeigt die Lunge, deren luftgefüllte Bereiche völlig Schwarz sind. Das meiste Weichgewebe erscheint in einem mittleren Grauton.

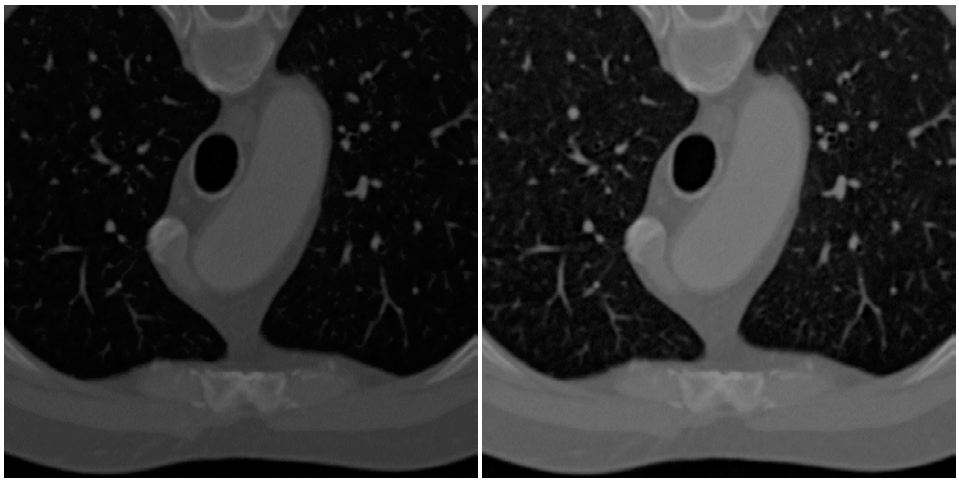

Abbildung 104: Linear vs. Lum. Mapping (Images\Schnaidt\Medical\Data1-92-*-K5.png)

In dieser Abbildung lässt sich der Unterschied zwischen **Linear** und **Luminance Mapping** noch besser erkennen. Die luftgefüllten Bereiche der Lunge enthalten feinste Blutgefäße und Bronchiolen (Luftröhrenäste). Während im linken Bild bereits große Teile fast Schwarz sind, bleiben rechts mehr Details erhalten.

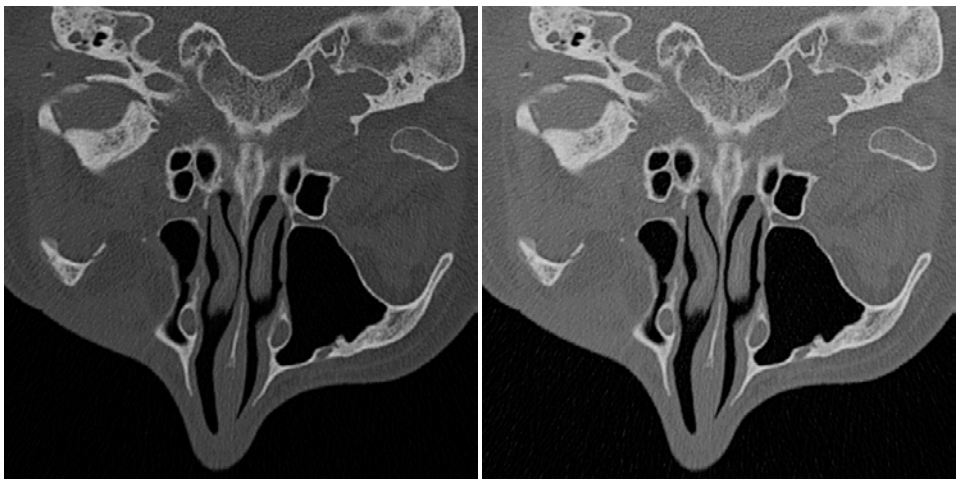

Abbildung 105: Linear vs. Lum. Mapping (Images\Schnaidt\Medical\Data2-192-*-K5.png)

Das letzte Beispiel in **Abbildung 105** ist ebenfalls ein CT Scan. Es zeigt einen axialen (horizontalen) Querschnitt eines Kopfes. Das rechte Bild erscheint durch seine angepasste Helligkeit natürlich, wobei Details in diesem Fall in beiden Bildern gut erkennbar sind. Details zu verstärken (genauer aus dem Originalvolumen zu erhalten) ist auch nicht die Hauptaufgabe des **Luminance Mapping**, denn hierfür ist das *Dodging-and-Burning* gedacht, das in **Hdrw Reinhard** zum Einsatz kommt.

3.6.2 Luminance Mapping vs. Hdrw Reinhard

Nachdem die Grundhelligkeit durch das **Luminance Mapping** korrigiert wurde, wird in **Hdrw Reinhard** jeder Voxel des Volumens separat betrachtet und in seiner Helligkeit angepasst. Die verwendete Technik hierzu nennt sich *Dodging-and-Burning* und wird genauer in 3.4.2, Seite 107 vorgestellt.

Es lassen sich verschiedene Größen des dabei verwendeten Filterkernels einstellen. Die Standardeinstellung hierfür ist **Kernel Delta** 2 und **Scales** 5. Im praktischen Einsatz ist der Unterschied im Normalfall unerheblich, aber da wir hier die Unterschiede genau herausarbeiten möchten, benutzen wir die etwas größere Einstellung **Kernel Delta** 5 und **Scales** 7, welche etwa die doppelte Zeit benötigt.

Zusätzlich lässt sich noch der **2D** oder **3D** Modus auswählen. Da der **2D** Modus im Vergleich zum **3D** Modus nur um etwa ein Drittel schneller ist, wenden wir hier den am besten passenden Modus für die Daten an. Sprich bei 2-dimensionalen Bildern den **2D** Modus, bei 3-dimensionalen Volumen den **3D** Modus. Oft kann auch der **2D** Modus als "Näherung" für 3-dimensionale Volumen benutzt werden, ohne dass sich das Endergebnis stark ändern würde, dabei kommt es aber zu einem prinzipiellen Fehler im Algorithmus: Um die neue Helligkeit des Voxels zu bestimmen, wird dessen Umgebung betrachtet. Der **2D** Modus benutzt hierfür in der aktuellen Schicht verschieden große 2-dimensionale Kreise um den Voxel herum, während der **3D** Modus wirklich 3-dimensionale Kugeln verwendet. Im **2D** Fall wird also nur ein Teil der notwendigen Information betrachtet.

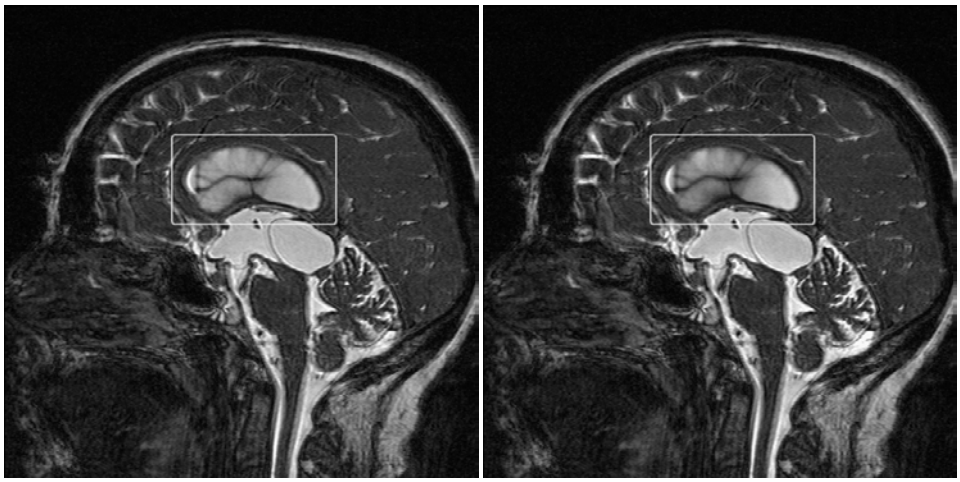

Abbildung 106: 2D Modus vs. 3D Modus (Images\Schnaidt\2Dvs3D-HR2D-K5-1, 2.png)

In **Abbildung 106** können Sie sehen, dass dies auch einen konkreten Unterschied bei den Resultaten bewirken kann. Rechts sehen Sie das korrekte Resultat im **3D** Modus und links die Näherung des **2D** Modus. Die dunklen Teile des markierten Ventrikelsystems in der Mitte des Gehirns erscheinen im **2D** Modus zu hell. Dies liegt daran, dass die Information über die umgebenden Schichten nicht miteinbezogen wird. Dort wird das Ventrikelsystem fortgeführt und ist dabei heller als in dieser Schicht, denn im Bild sehen Sie eigentlich einen Teil der dunkleren Trennwand zwischen dem linken und rechten Ventrikel (Das Ventrikelsystem besteht aus insgesamt vier Kammern, wovon zwei Kammern sich im markierten Bereich befinden und in eine linke und rechte Kammer aufgeteilt sind).

3.6.2.1 Farbbilder

Das Ziel des *Dodging-and-Burning* ist es, Teile der Dynamik bzw. Genauigkeit des Originalbildes zu erhalten. Dies wird dadurch erreicht, dass dunkle Voxel in einer hellen Umgebung noch weiter verdunkelt werden (*Dodging*). Helle Voxel in einer dunklen Umgebung dagegen werden weiter aufgehellt (*Burning*). Insgesamt bleiben so Helligkeitsunterschiede besser erhalten.

Abbildung 107: Lum. Mapp. vs. Reinhard 2D, Gamma 2,5 (Images\Schnaidt\Reinhard\Lampicka-*-K5-G25.png)

Der Text auf dem Blatt Papier unterhalb der Lampe in **Abbildung 107** ist hierfür ein Beispiel und wurde auch bereits in 3.4.2.3, Seite 111 angesprochen. Der schwarze Text auf dem hellen Papier (als Umgebung) erscheint rechts deutlicher abgesetzt. Links trennt er sich kaum mehr vom Untergrund.

Die genauen Unterschiede ergeben sich im Differenzbild (**Abbildung 108**).

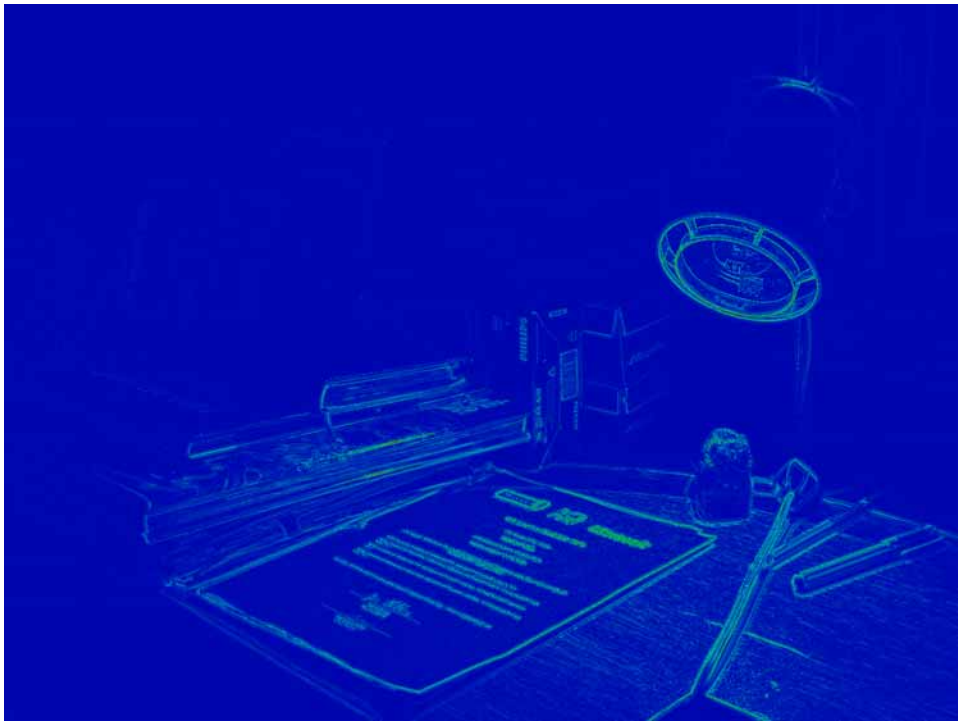

Abbildung 108: Differenz Lum. Mapp. & Reinhard 2D (Images\Schnaidt\Reinhard\Lampicka-HR2D-K5-G25-Diff.png)

Im Bild abgetragen ist die relative Differenz beider Bilder. Das heißt auf einer Skala zwischen blau-grün-rot wird *keine Differenz* als Blau und die *maximale Differenz* als Rot identifiziert.

Besonders die Buchstaben das Textes unterscheiden sich in beiden Bildern. Aber auch die Kanten des Papiers, die Kanten der Bücher und sogar der einzelnen Buchseiten heben sich ab. Außerdem kann man erkennen, dass das Innere der Lampe nicht komplett Weiß ist.

Diesen Teil betrachten wir in einer Vergrößerung noch einmal genauer (**Abbildung 109**). Die Innenseite der Lampe enthält einen Aufkleber mit der Wattzahl der Lampe. Zusätzlich ist auf dem oberen Teil der Glühbirne selbst ein halbtransparenter Text aufgebracht. Erhöht man die Option **Phi** von 8,0 (Links) auf 20,0 (Rechts), lassen sich die Kanten des Bildes noch weiter verschärfen. Dann werden diese Details trotz der starken Helligkeitsschwankungen sichtbar.

Abbildung 109: Phi 8 vs. 20 (Images\Schnaidt\Reinhard\LampickaCloseup-HR2D-K5-G25-Phi8, 20.png)

Generell kann man mit **Phi** die Kantenschärfe erhöhen, allerdings ist der Effekt nicht auf allen Bildern so drastisch wie im letzten Beispiel (Siehe 2.4.4.7, Seite 90).

Abbildung 110: Lum. Mapp. vs. Reinhard 2D, Gamma 2,5 (Images\Schnaidt\Reinhard\Memorial-*-K5-G25.png)

Abbildung 111: Lum. Mapp. vs. Reinhard 2D, Gamma 2,2 (Images\Schnaidt\Reinhard\Nave-*-K5-G22.png)

In den Beispielen in **Abbildung 110** und **Abbildung 111** wird auch mehr der Dynamik des Originalbildes erhalten. In der ersten Abbildung ist beispielsweise die Verzierung der Fensterkuppel links nur verschwommen erkennbar und rechts ausgeprägter. Im zweiten Bild weisen die Kirchenfenster rechts mehr Details auf, trotz der starken Überstrahlung im Vergleich zum Rest des Raumes.

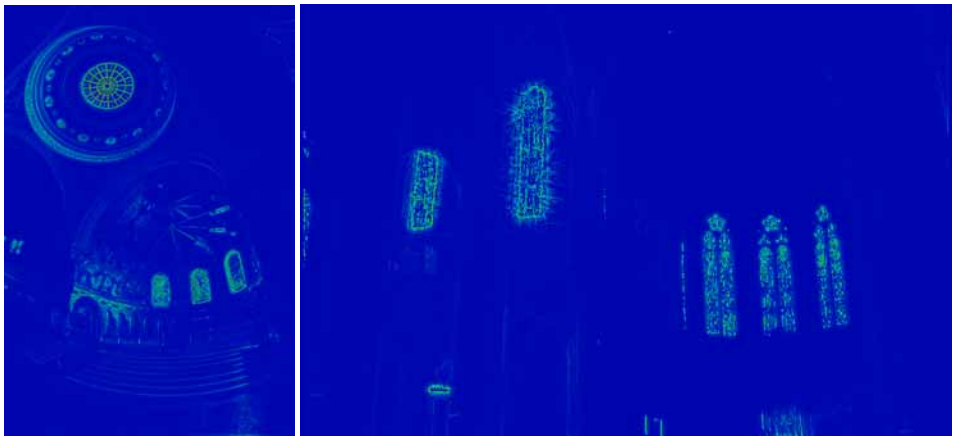

Abbildung 112: Differenz Lum. Mapp. & Reinhard 2D (Images\Schnaidt\Reinhard\Memorial, Nave-HR2D-K5-G*-Diff.png)

Die Differenzbilder in **Abbildung 112** demonstrieren die Unterschiede in den oben genannten Arealen noch einmal genauer.

Wichtig hierbei ist, dass trotz der Unterschiede der Bilder keine Artefakte oder unnatürlich wirkende Bereiche hinzugefügt werden, was beim Windowing keineswegs eine Selbstverständlichkeit ist. Ähnlich zum *Dodging-and-Burning* in der Photographie werden Teile des Bildes aufgehellt und abgedunkelt und diese Teile nahtlos in den Rest des Bildes eingefügt.

3.6.2.2 Medizinische Volumen

Auch bei medizinischen Volumen kann das *Dodging-and-Burning* erfolgreich eingesetzt werden. **Abbildung 113** zeigt hierfür unser erstes Beispiel.

Während die Grundhelligkeit beider Bilder benahe identisch ist, sieht man rechts einen deutlichen höheren Kontrast zwischen den einzelnen Organen und Gehirnstrukturen. Beispielsweise heben sich die Kammern des Ventrikelsystems in der Mitte des Gehirns vom umliegenden Gewebe ab.

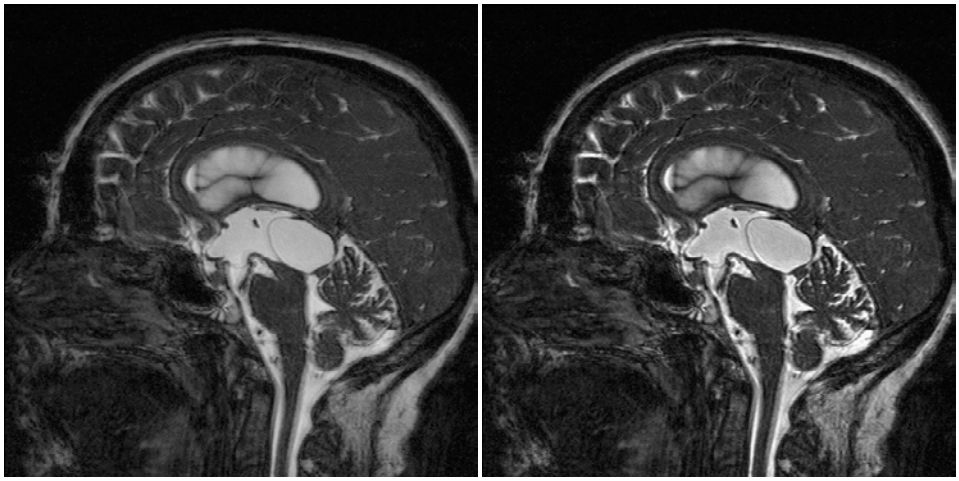

Abbildung 113: Lum. Mapping vs. Hdrw Reinhard 3D (Images\Schnaidt\Medical\Data3-29-*-K5.png)

Wie bereits in 3.4.2.3, Seite 111 angesprochen wurde, ist das aber keine herkömmliche Kontrasterhöhung, wie sie auch mit wesentlich einfacheren Methoden erreicht werden könnte (Beispielsweise gängige Bildverarbeitungsprogramme wie Adobe Photoshop bieten solche Funktionen an). Hier wird stattdessen die Dynamik bzw. Genauigkeit des Originalbildes mit in diesem Fall etwa 4096 verschiedenen Grauwerten als "Informationsquelle" benutzt. Diese Information fließt in das Ausgabebild mit nur 256 Grauwerten mit ein und verbessert es, so dass zumindest ein Teil der Dynamik erhalten bleibt. Konkret bedeutet dies, dass trotz von nur 256 Grauwerten gewisse Helligkeitssprünge zwischen verschiedenen Organen und Strukturen erhalten bleiben. Dadurch kann das menschliche Auge oder auch maschinelle Algorithmen diese Strukturen leichter trennen oder eventuell sogar überhaupt erst als Strukturen identifizieren.

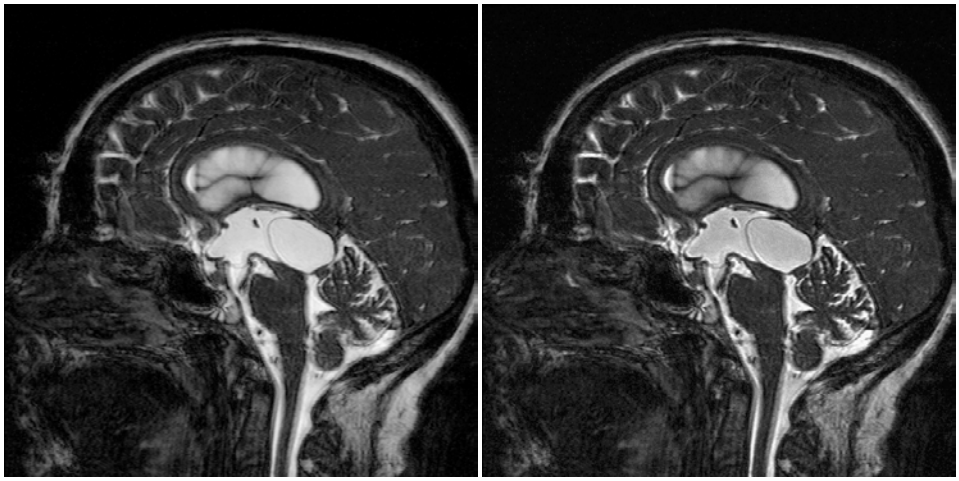

Abbildung 114: Links Photoshop, rechts Hdrw (Images\Schnaidt\Medical\Data3-29-LM-K5-Photoshop, HR3D-K5.png)

Nun wollen wir die Behauptung von oben bezüglich der herkömmlichen Kontrasterhöhung mit einem Beispiel untermauern. **Abbildung 114** links zeigt das **Luminance Mapping** Bild, wobei der Kontrast in Adobe Photoshop erhöht wurde - rechts weiterhin das Originalbild mit **Hdrw Reinhard**. Auf den ersten Blick haben die Bilder nun einen ähnlichen Kontrast. Betrachtet man die Bilder genauer, fallen aber schnell Unterschiede auf. Die Grenzflächen zwischen den meisten Organen und Strukturen sind im rechten Bild deutlich stärker ausgeprägt. Auch eine weitere Erhöhung des Kontrasts links hätte nicht zu diesem Resultat geführt, denn dies resultiert in Photoshop schnell in einem völligen Ausbleichen aller hellen Regionen.

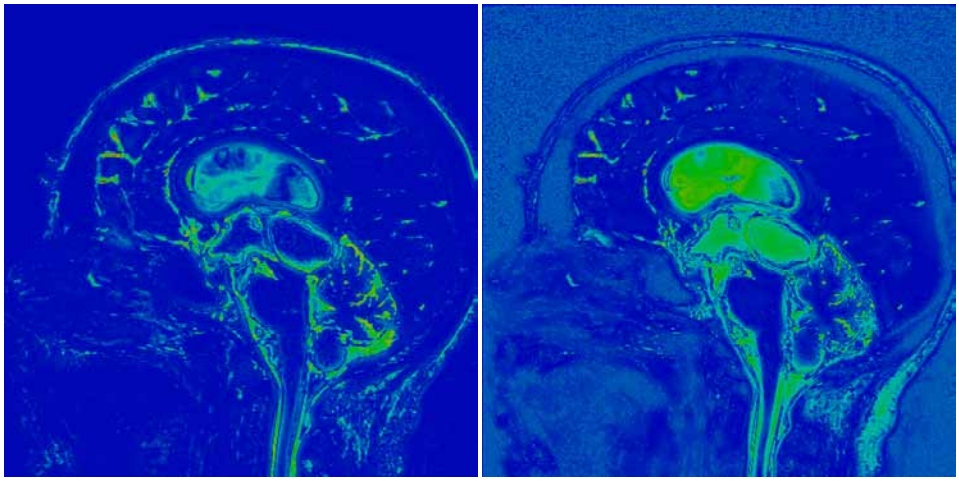

Abbildung 115: Differenz (Images\Schnaidt\Medical\Data3-29-HR3D-K5-Diff, LM-K5-Photoshop-Diff.png)

Die detaillierten Unterschiede zwischen **Luminance Mapping** und **Hdrw Reinhard** (**3D** Modus) sehen Sie noch einmal in **Abbildung 115** links. Die rechte Abbildung zeigt die Differenz zwischen **Luminance Mapping** mit Photoshop Kontrasterhöhung und **Hdrw Reinhard** (**3D** Modus).

Die Veränderungen sind auch im Histogramm sichtbar (2.3.5.2, Seite 41 enthält eine Einführung zu Histogrammen). **Abbildung 116**, **Abbildung 117** und **Abbildung 118** enthalten jeweils links das normale Histogramm und rechts das Gradientenhistogramm.

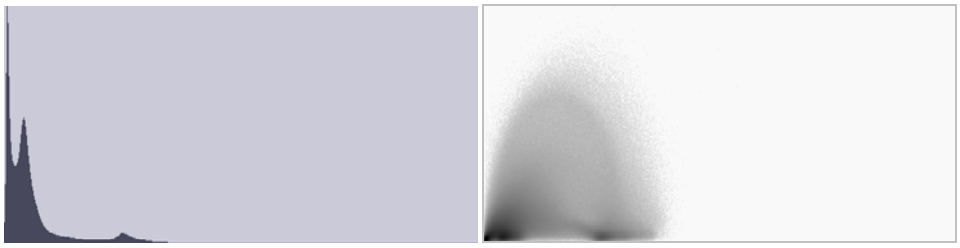

Abbildung 116: Histogramm Original (Images\Schnaidt\Medical\Data3-(Gradient)Histogram-Original-K5.png)

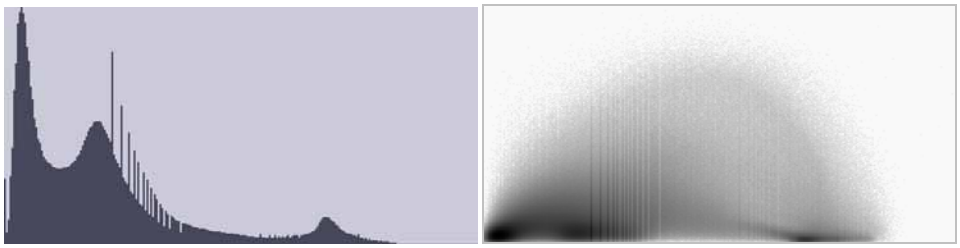

Abbildung 117: Histogramm Lum. Mapp. (Images\Schnaidt\Medical\Data3-(Gradient)Histogram-LM-K5.png)

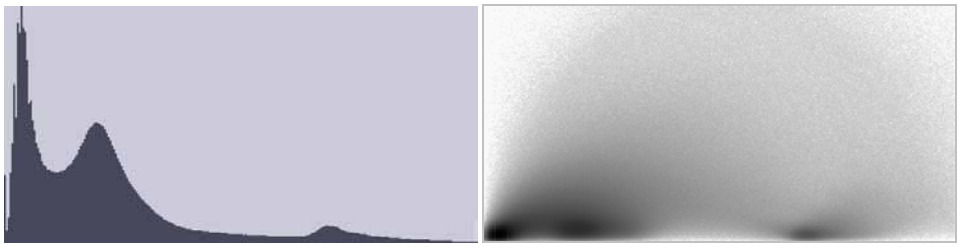

Abbildung 118: Histogr. Hdrw Reinhard 3D (Images\Schnaidt\Medical\Data3-(Gradient)Histogram-HR3D-K5.png)

Abbildung 116 bezieht sich auf die Originaldaten und insgesamt 4096 Grauwerte. Die meisten der hohen Grauwerte werden allerdings nicht genutzt. Nach dem

Luminance Mapping ergibt sich **Abbildung 117** bezogen auf 256 Grauwerte. Sowohl das normale als auch das Gradientenhistogramm wurden automatisch auf fast den kompletten Wertebereich verteilt. Dies entspricht der Anpassung der Grundhelligkeit.
Schließlich führt **Hdrw Reinhard** in **Abbildung 118** zu einer deutlichen Glättung des Histogramms. Das heißt während die Dynamik in den Originaldaten zunimmt, wird gleichzeitig das Histogramm gleichmäßiger. Dies erscheint zunächst ungewöhnlich, da Materialien im Histogram als Peaks wieder zu finden sind und Materialgrenzen als steigende und fallende Flanken. Bei einem erhöhten Kontrast würde man daher besonders viel Peaks und Flanken erwarten. **Hdrw Reinhard** hat aber gerade den umgekehrten Effekt: einzelne Peaks werden eliminiert, nur die vorhandenen Peaks bleiben erhalten. Genau dies garantiert die Vermeidung von Artefakten, da nur vorhandene Materialien und Materialübergange erhalten und verstärkt werden. Einzelne starke Abweichungen wie in **Abbildung 117** dagegen können zu Artefakten oder unnatürlichen Bereichen im Bild führen.

Im Gradientenhistogramm (**Abbildung 118** rechts) sind Materialübergänge als Maxima sichtbar. Auch hier zeigt ein Vergleich, dass **Hdrw Reinhard** zu einer Erhöhung dieser Maxima führt. Dies ist äquivalent zu einem größeren Gradienten und einem stärkeren Kontrast.

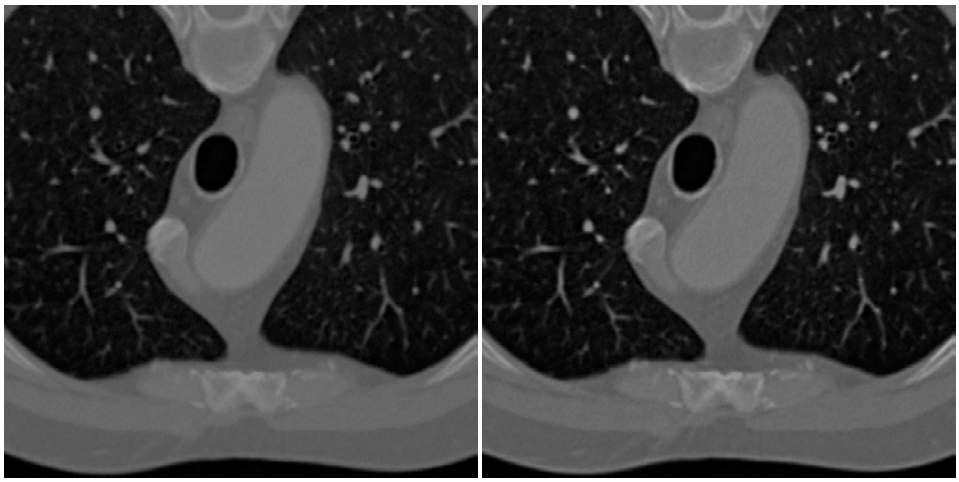

Abbildung 119: Lum. Mapping vs. Hdrw Reinhard 3D (Images\Schnaidt\Medical\Data1-92-*-K5.png)

Im nächsten Beispiel (**Abbildung 119**) werden ebenfalls vorhandene Strukturen und Übergänge verstärkt, der Effekt ist hier aber im Weichgewebe (in der Mitte und unten im Bild) nicht ganz so stark ausgeprägt. **Hdrw Reinhard** hat nicht auf allen Daten einen gleich starken Einfluss, was - wie sollte es anders sein - an den Daten selbst liegt. In diesem Fall sehen Sie den Grund im Histogramm (**Abbildung 120** links).

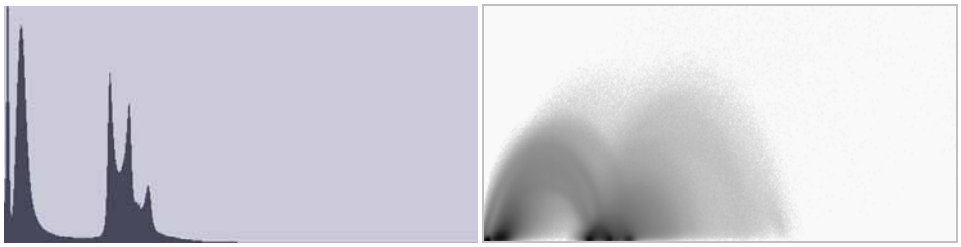

Abbildung 120: Histogramm Original (Images\Schnaidt\Medical\Data1-(Gradient)Histogram-Original-K5.png)

Insgesamt sieht man fünf Peaks, wovon die drei rechten Peaks dem Weichgewebe entsprechen. Diese drei Peaks sind bereits in den Originaldaten relativ nahe beieinander, wodurch *Hdrw* die Unterschiede auch weniger verstärkt. Letztend

lich soll der Algorithmus zu keinem Ergebnis führen, das die Abstände solcher Peaks unter allen Umständen maximiert und dabei aber das Originalbild komplett verändert. Ziel ist ein realistisches Endergebnis, was in der Medizin für Ärzte genauso entscheidend wie gut sichtbare Details ist, da sie sich auf ihre Erfahrungen mit anderen CT oder MRT Bildern verlassen können müssen.

Wieder deutlicher wird der Effekt von **Hdrw Reinhard** in **Abbildung 121**. Die weißen Knochenstrukturen werden aufgehellt, das Weichgewebe erscheint noch etwas kontrastreicher und generell werden alle Übergänge dazwischen verstärkt.

Die Asymmetrie der linken und rechten Gesichtshälfte im Bild entstand dabei durch einen Unfall, nach welchem die Augenhöhle des Patienten rekonstruiert werden musste.

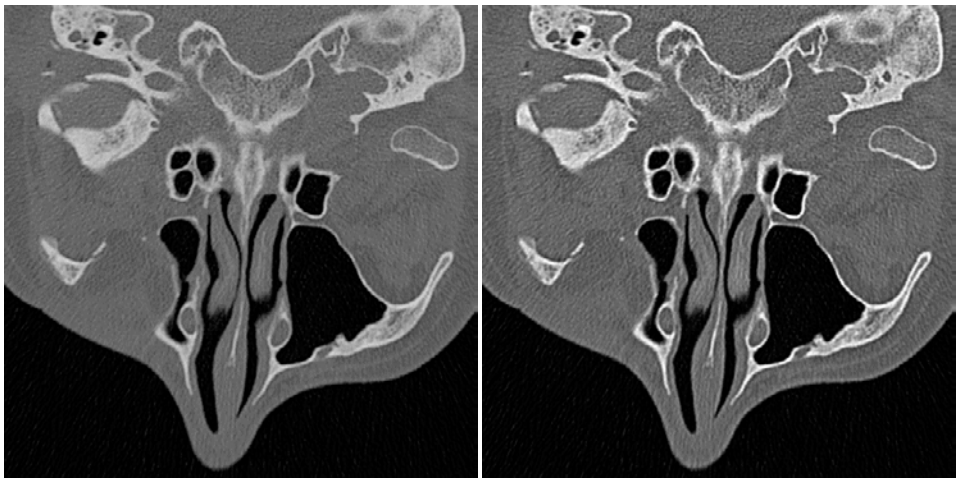

Abbildung 121: Lum. Mapping vs. Hdrw Reinhard 3D (Images\Schnaidt\Medical\Data2-192-*-K5.png)

Abschließend folgen noch die Differenzbilder zu den beiden obigen Beispielen in **Abbildung 122**.

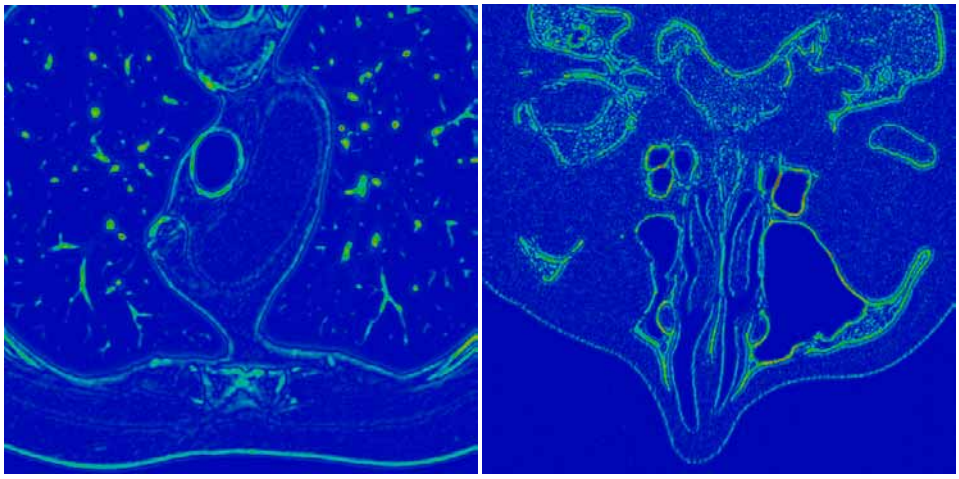

Abbildung 122: Differenz Lum. Mapp. & Reinhard 3D (Images\Schnaidt\Medical\Data1-92, 2-192-HR3D-K5-Diff.png)

3.6.3 Weitere Beispiele

Im letzten Abschnitt dieses Kapitels folgt eine Sammlung weiterer Beispielbilder, auf die alle drei Methoden **Linear**, **Luminance Mapping** und **Hdrw Reinhard** angewendet wurden. Die Bilder sind verkleinert, um Speicherplatz zu sparen. Im Archiv *ReportImages.zip* sind die Bilder in Originalgröße gespeichert.

3.6.3.1 Farbbilder

Abbildung 123: Linear & Lum. Mapp. & Reinhard 2D, Gamma 3,0 (Images\Schnaidt\Reinhard\BristolBridge-*-K5-G3.png)

Abbildung 124: Linear & Lum. Mapp. & Reinhard 2D, Gamma 3,0 (Images\Schnaidt\Reinhard\Clock-*-K5-G3.png)

Abbildung 125: Linear & Lum. Mapp. & Reinhard 2D, Gamma 2,2 (Images\Schnaidt\Reinhard\CornellBox-*-K5-G22.png)

Abbildung 126: Linear & Lum. Mapp. & Reinhard 2D, Gamma 2,2 (Images\Schnaidt\Reinhard\Crowfoot-*-K5-G22.png)

Abbildung 127: Linear & Lum. Mapp. & Reinhard 2D, Gamma 2,2 (Images\Schnaidt\Reinhard\GroveC-*-K5-G22.png)

Abbildung 128: Linear & Lum. Mapp. & Reinhard 2D, Gamma 2,2 (Images\Schnaidt\Reinhard\GroveD-*-K5-G22.png)

Abbildung 129: Linear & Lum. Mapp. & Reinhard 2D, Gamma 2,2 (Images\Schnaidt\Reinhard\Lamp-*-K5-G22.png)

Abbildung 130: Linear & Lum. Mapp. & Reinhard 2D, Gamma 3,0 (Images\Schnaidt\Reinhard\Oaks-*-K5-G3.png)

Abbildung 131: Linear & Lum. Mapp. & Reinhard 2D, Gamma 2,5 (Images\Schnaidt\Reinhard\Rosette-*-K5-G25.png)

Abbildung 132: Linear & Lum. Mapp. & Reinhard 2D, Gamma 3,0 (Images\Schnaidt\Reinhard\Sunrendering-*-K5-G3.png)

Abbildung 133: Linear & Lum. Mapp. & Reinhard 2D, Gamma 3,0 (Images\Schnaidt\Reinhard\Tahoe-*-K5-G3.png)

Abbildung 134: Linear & Lum. Mapp. & Reinhard 2D, Gamma 2,2 (Images\Schnaidt\Reinhard\Tinterna-*-K5-G22.png)

Abbildung 135: Linear & Lum. Mapp. & Reinhard 2D, Gamma 2,5 (Images\Schnaidt\Reinhard\VineSunset-*-K5-G25.png)

Abbildung 136: Linear & Lum. Mapp. & Reinhard 2D, Gamma 2,5 (Images\Schnaidt\Reinhard\Wreathbu-*-K5-G25.png)

3.6.3.2 Medizinische Volumen

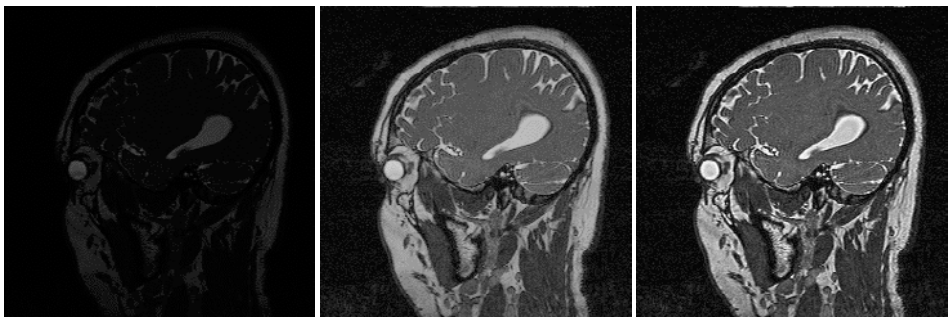

Abbildung 137: Linear & Lum. Mapp. & Reinhard 3D (Images\Schnaidt\Medical\Bciss-29-*-K5.png)

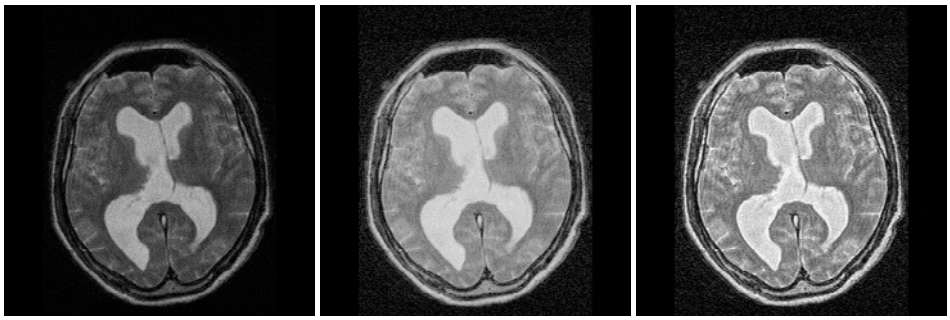

Abbildung 138: Linear & Lum. Mapp. & Reinhard 3D (Images\Schnaidt\Medical\Ktse-16-*-K5-KV010.png)

Jirko Cernik

4 Die Reinhard Methode auf Curvilinearen Gittern

4.1 Einleitung

Beim Windowing, oder auch Fensterung genannt, werden Bilddaten zum Beispiel von 12 Bit auf 8 Bit heruntergerechnet. Um dies zu ermöglichen, gibt es verschiedene Verfahren, das einfachste davon ist das *lineare Windowing*, wobei hier die Werte direkt linear umgerechnet werden, was zu einer äußerst schlechten Darstellung führt.

Bei medizinischen Volumendaten liegt dies meist daran, dass die 12 Bit nicht voll ausgeschöpft werden und auf diese Weise ein großer Zahlenbereich leer bleibt. Um dies zu kompensieren, gibt es das Modell des *Luminance Mapping*, das die Gesamthelligkeit des Volumens entsprechend anpasst.

Erweiterte Methoden, die auch unter dem Begriff *High Dynamic Range Windowing* oder *High Dynamic Range Imaging* zusammengefasst werden, bessern das Ergebnis des Luminance Mappings noch weiter auf. *High Dynamic Range Windowing* Methoden sorgen zudem für ein klareres und schärferes Bild. Die Bildverbesserung, wenn man Hdrw Methoden wählt, funktioniert auch dann, wenn keine direkte Fensterung stattfindet, also der Quelldatensatz z. B. aus 8 Bit Datenelementen bestand und auch der Ausgabedatensatz 8 Bit Datenelemente enthält. Das Ergebnis ist hierbei meist schärfer und besser kontrastiert, wenn die Bilddaten eine hohe Dynamik aufweisen.

In unserem Falle wurden die Hdrw Methoden direkt auf 3D Volumendaten angewendet, wobei speziell die Reinhard Methode, die Ashikhmin Methode und die Durand Methode verwendet wurden. Die einschränkende Eigenschaft dieser Methoden ist, dass sie nur auf uniformen Gittern anwendbar ist. Uniforme Gitter sind dabei Gitter, die eine regelmäßige Geometrie und Topologie aufweisen und lediglich einen anderen Pixelabstand als Schichtabstand haben.

Wendet man die Algorithmen direkt auf *Curvilinear Grids* können die Ergebnisse durch die Biegung verfälscht werden. Curvilineare Gitter besitzen ebenfalls eine regelmäßige Topologie jedoch sind in der Geometrie gebogen. Um dennoch *High Dynamic Range Windowing* auf curvilinearen Gittern betreiben zu können, muss (a) eine exakte Interpolation erfolgen, was - wie später gezeigt wird - selbst auf schnellen Rechnern extrem langsam wäre, oder (b) die Hdrw Methode muss sich direkt an die curvilinearen Daten anpassen.

Im Folgenden wird die Reinhard Methode so modifiziert, dass curvilineare Daten korrekt entsprechend ihrer Biegung berechnet werden können. Das Ergebnis wird dann direkt mit der traditionellen Reinhard-Methode verglichen. An den meisten Stellen fällt der Unterschied optisch kaum auf, jedoch merkt man im Differenzbild an Stellen großer Biegung eine deutliche Abweichung zur traditionellen Reinhard Methode.

Curvilineare Datensätze stellen oft nicht nur Skalardaten sondern auch Vektordaten bereit, so dass der Algorithmus auch in der Lage sein muss Vektordaten zu

verarbeiten. Die Skalardaten selbst liegen meistens in Fliesskomma-Arithmetik vor und weisen eine sehr hohe Dynamik an möglichen Werten auf.

Es wird hier auf die verschiedenen Datenformate der curvilinearen Gitter eingegangen und der Reinhard Algorithmus erklärt, um dann die Veränderungen des neuen Algorithmus auf curvilinearen Daten zu verdeutlichen. Anhand von Beispielen werden die Auswirkungen und die Unterschiede zum alten Reinhard-Algorithmus dargestellt.

Die Darstellung der curvilinearen Volumen erfolgt in einem speziell dafür angefertigten Programm. Da man bei Vektordaten 3 Komponenten pro Vektor hat und diese nicht direkt in den Hdrw Algorithmus einbeziehen kann, wird es auch möglich sein, die Volumen gemäß der einzelnen Komponenten jedes Vektors einzufärben.

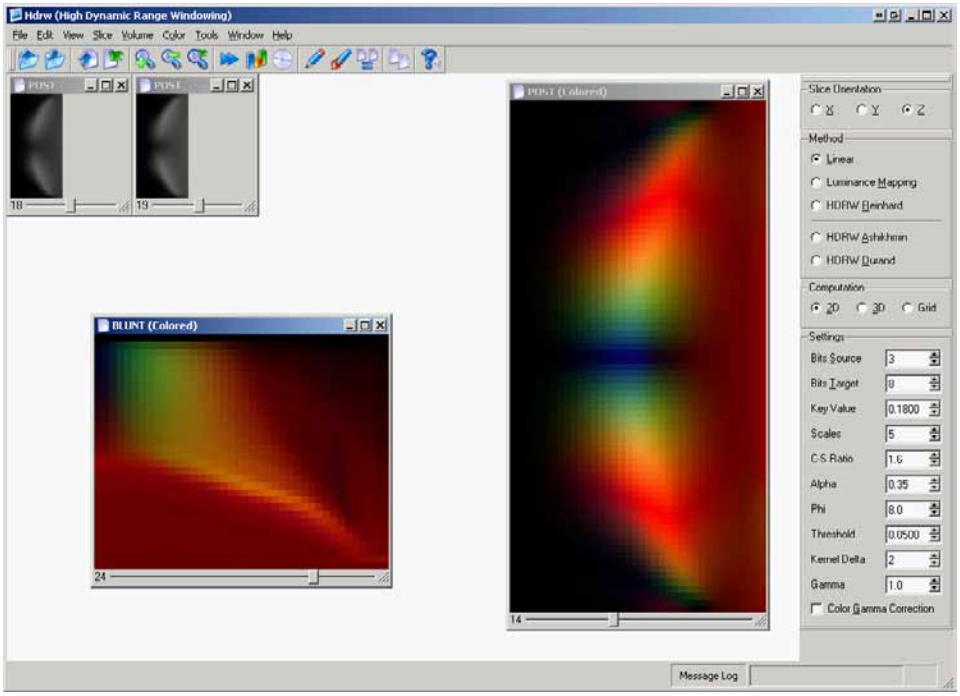

Abbildung 139: Überblick über das Programm (Images\Cernik\Overview.png)

4.1.1 Bedienung des Programms für curvilineare Volumen

Im Folgenden soll die Bedienung des Programms anhand eines konkreten Beispiels vorgestellt werden. Dabei wird ein curvilineares Vektor-Volumen geöffnet, eine Anwendung des modifizierten Reinhard-Algorithmus auf curvilinearen Daten durchgeführt und anschließend das Volumen wieder als Vektor-Volumen abgespeichert.

Da der Hdrw Algorithmus nur auf skalaren Werten operieren kann, betrachtet das Programm den Betrag des jeweiligen Vektors. Der Algorithmus arbeitet dann auf den Beträgen und skaliert die Vektoren beim Abspeichern mit den neu errechneten Beträgen, so dass das Ergebnis wieder ein curvilineares Vektor-Volumen ist.

Das Programm bietet die Möglichkeit curvilineare Volumen in verschiedenen Formaten zu öffnen. Unterstützt wird das *SCALAR* und *VECTOR* Format, wobei jeweils die Koordinaten für die Punkte aus einem *GRID* oder alternativ *MESH* File ausgelesen werden. Des Weiteren dürfen die Files auch *GZ* komprimiert vorliegen, um Platz zu sparen.

4.1.2 Das Blunt Fin Beispielvolumen

Im Folgenden betrachten wir als Beispielvolumen das Volumen *Blunt.vector*, welches eine Luftströmung über eine glatte Fläche simuliert, wobei auf der Fläche ein stumpfer Flügel angebracht worden ist. Der Luftstrom verläuft parallel zu der Fläche in X-Richtung. Da angenommen wird, dass der Luftstrom symmetrisch um das Zentrum des Flügels verläuft, ist nur der eine Teil des Luftstroms in der Geometrie repräsentiert.

Der interessante Teil des Volumens ist der so genannte *Lambda Shock*, der an der vorderen Kante des Flügels entsteht und die daraus resultierenden Verwirbelungen.

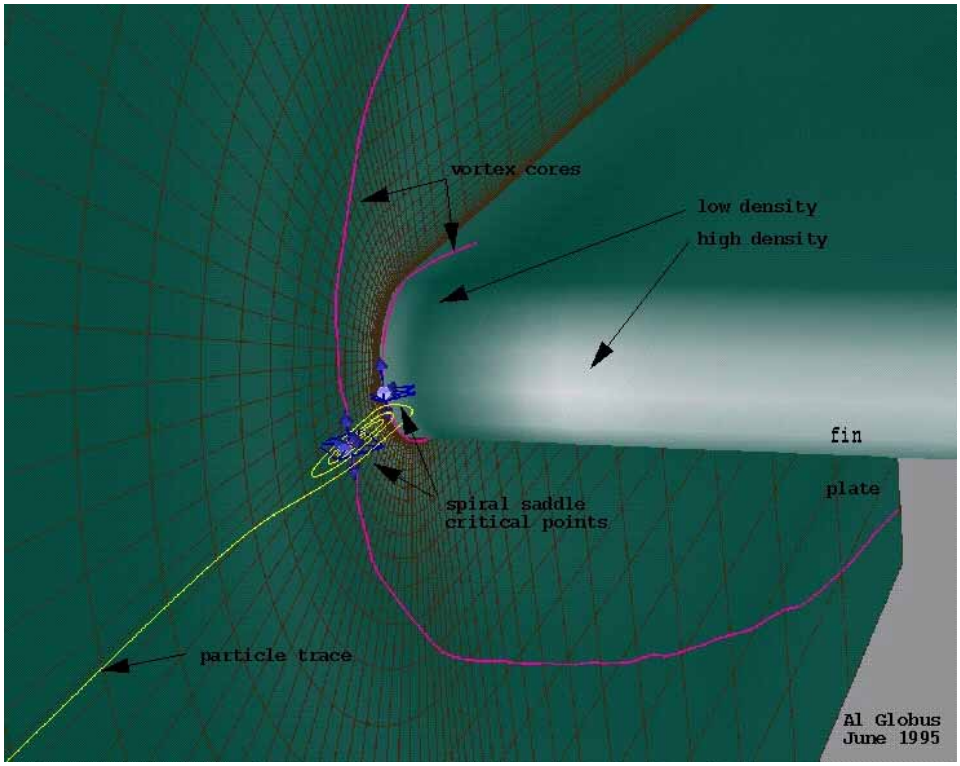

Abbildung 140: Blunt Fin Volumen in curvilinearer Darstellung (NASA, 2003)
(Images\Cernik\Blunt3DVis.png)

4.1.3 Curvilineare Volumen öffnen

Als erstes wird das Programm gestartet. Um nun ein Volumen zu öffnen, betätigt man entweder die Tastenkombination **Ctrl+O** oder geht direkt über das Menü **File > Open Volume**. Der "Datei öffnen" Dialog erscheint und als **File Type** sollte **All Volume Formats** ausgewählt sein, was die curvilinearen Dateitypen *.scalar* und *.vector* mit einschließt. Nun wählt man die gewünschte Datei - in unserem Fall öffnen wir das Volumen *blunt.vector*.

Wichtig: Im gleichen Verzeichnis eines curvilinearen Volumens muss eine *.grid* oder *.mesh* Datei mit demselben Namen liegen, welche die Koordinatenpunkte für das Volumen enthält. In unserem Falle heißt diese Datei *blunt.grid*. Auf die genauen Dateiformate wird später eingegangen.

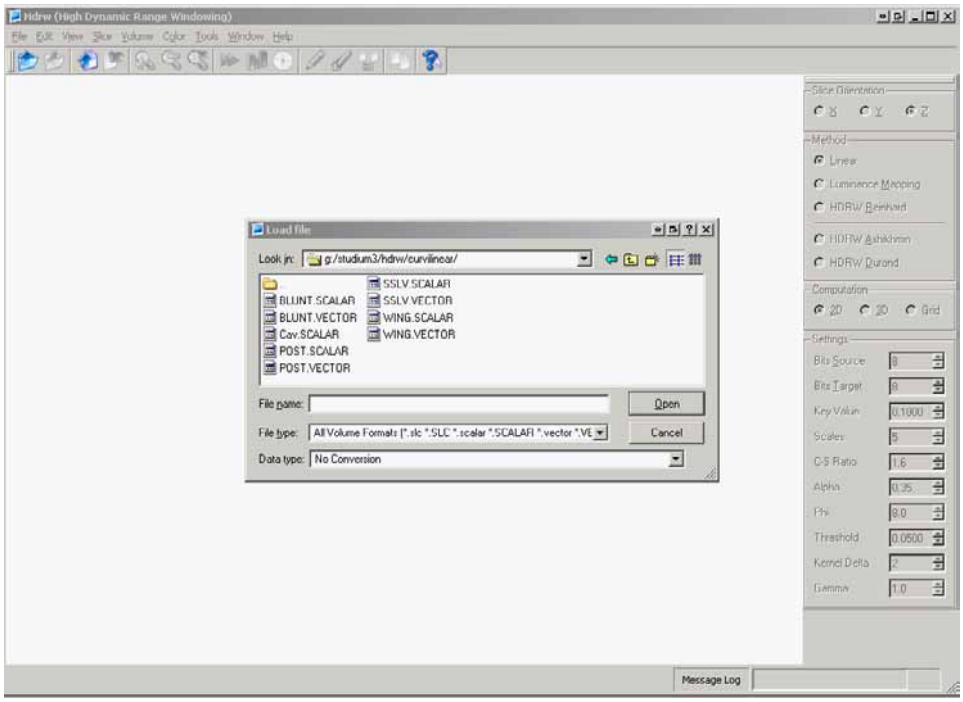

Abbildung 141: Der Dialog um Volumen zu öffnen (Images\Cernik\FileOpen.png)

Nach Öffnen des Volumens sieht man ein kleines Fenster mit dem Volumen. Über den Schieberegler unter dem Volumen kann das entsprechende Slice ausgewählt werden. Da die curvilinearen Volumen meist recht klein sind, empfiehlt es sich das Volumen zu vergrößern. Im ersten Slice unseres *Blunt* Volumens sieht man nichts, erst auf den weiteren Slices sind visuelle Daten zu erkennen. Beim Öffnen des Volumens werden die Intensitäten gemäß dem Betrag der Vektoren dargestellt, sofern es sich um ein Vektor-Volumen handelt.

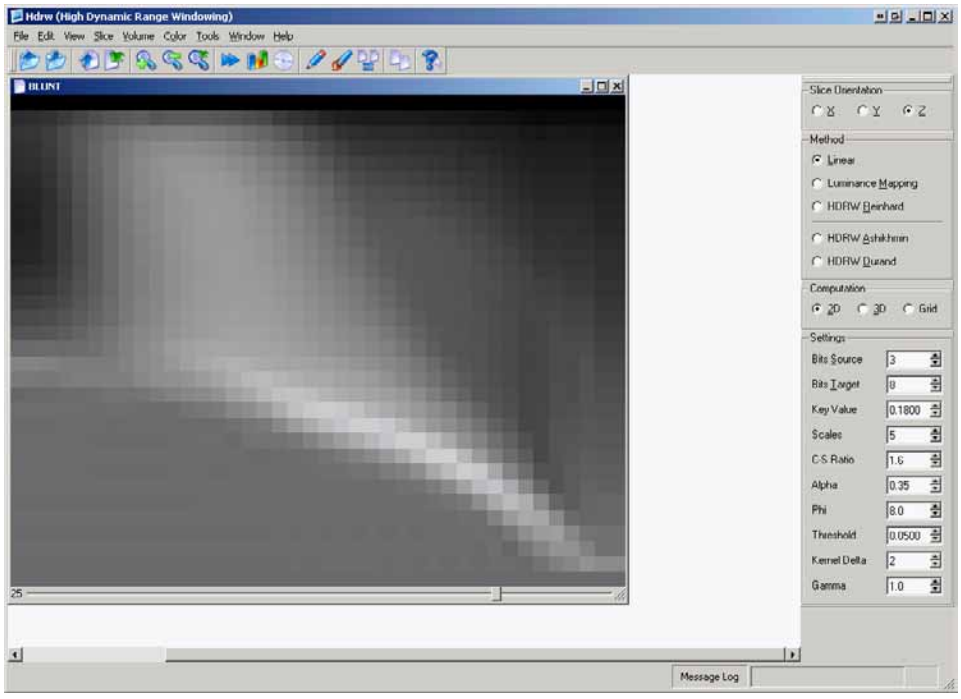

Abbildung 142: Das geöffnete Volumen, Darstellung von Slice 25 (Images\Cernik\BluntVolume.png)

4.1.4 Windowing curvilinearer Volumen

Die **Options** des Programms finden sich rechts. Auf die genauen Parameter wird im späteren Verlauf eingegangen. Für die Demonstration des Algorithmus am *Blunt* Volumen werden die Standardeinstellungen verwendet. In den Einstellungen unter **Method** wählt man nun die Methode **Hdrw Reinhard** aus.

Da es sich hier um ein curvilineares Volumen handelt, das auf einem eigenen *.grid* File (oder *.mesh* File) basiert, wird unter **Computation** die Option **Grid** ausgewählt.

Um nun das ganze Volumen mit dem Algorithmus zu bearbeiten, muss das Windowing gestartet werden, was über die Tastenkombination **Ctrl+W** oder direkt über den Menüpunkt **Volume > Start Windowing** geschieht.

Der Algorithmus benötigt einige Zeit und zeigt das Ergebnis dann in einem neuen Volumenfenster an.

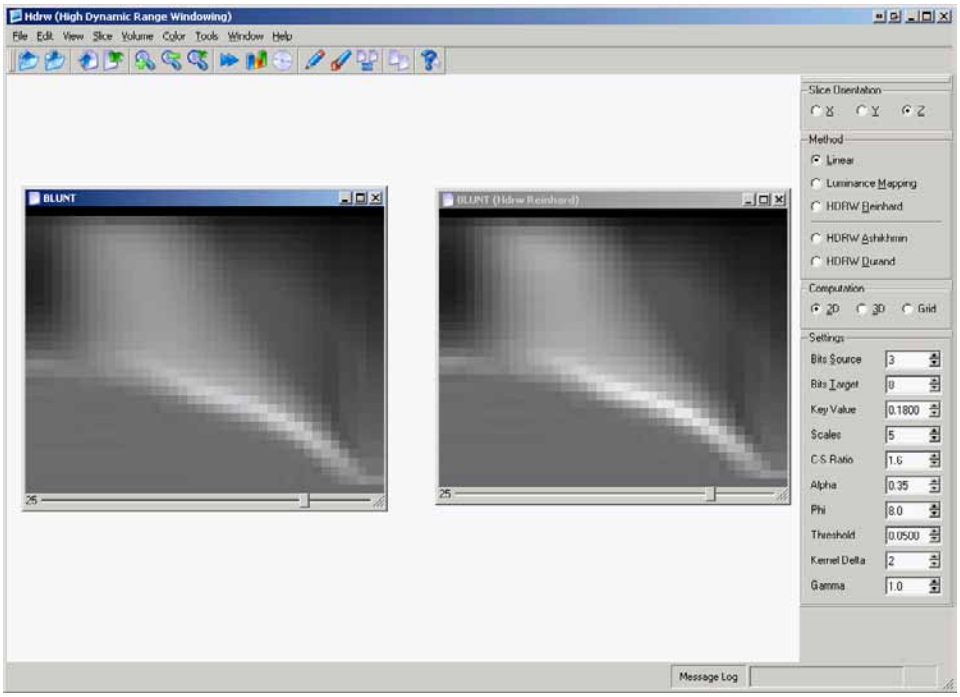

Abbildung 143: Links das alte Volumen, rechts das neu berechnete (Images\Cernik\BluntCompare.png)

4.1.5 Vergleich von Volumen

Der Unterschied zum Original-Volumen mag auf den ersten Blick gering erscheinen. Um diesen zu verdeutlichen, ist es möglich ein Differenzbild des aktuellen Slice anzufertigen. Dort ist der Unterschied dann sehr viel deutlicher zu sehen. Differenzbilder können per Tastenkombination **Ctrl+P** oder direkt über den Menüpunkt **Tools > Compare Slices** angefertigt werden. Als Beispiel haben wir Slice 25 mit dem Original vergleichen.

Als Einstellung wurde hierbei eine relative Differenz gewählt und zur besseren Visualisierung wurde das Differenzbild farbig mit Hilfe des *HSV* Modells dargestellt. Bereits im *Preview* Fenster sieht man direkt das Differenzbild und kann hier bereits deutlich die Unterschiede erkennen. Der Hauptunterschied verläuft entlang der roten Kante, welche im Bild als heller Strich zu sehen ist.

In den blauen Bereichen ist der Unterschied nur gering oder kaum zu sehen.

Wichtig für einen genauen Vergleich ist, dass man jeweils dasselbe Slice in beiden Volumen miteinander vergleicht, in unserem Falle wird Slice 25 verglichen.

Des Weiteren muss das erste Volumen, welches als Ausgangsvolumen diente, wieder auf die Methode **Linear** eingestellt werden, damit das Original Slice mit dem neuen Slice, das mit der **Hdrw Reinhard** Methode berechnet wurde, verglichen werden kann.

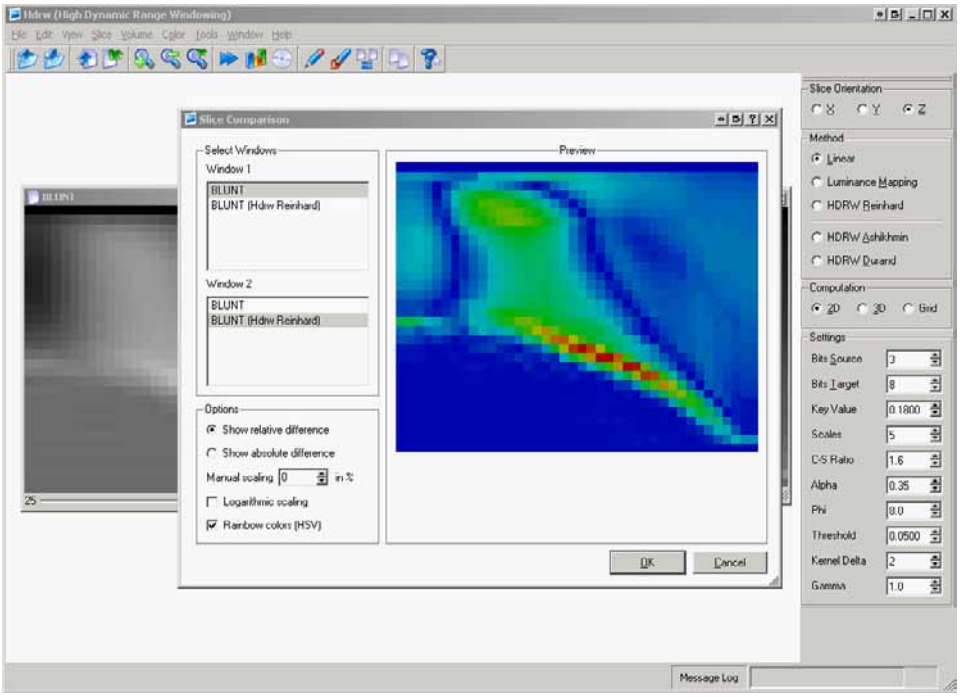

Abbildung 144: Das Differenzbild zwischen Original und HDRW (Images\Cernik\BluntCompare2.png)

Die Berechnung des Differenzbildes wurde jetzt wieder abgebrochen, da dies nur Demonstrationszwecken diente.

Interessant ist auch ein Vergleich der Histogramme der beiden Volumen wie in **Abbildung 145**. Das Histogramm zu einem Volumen kann man über die Tastenkombination **Ctrl+H** aufrufen. Alternativ kann man das Histogramm über das Menü **Volume > Histogramm** berechnen lassen. Das jeweilige Volumen muss dabei aktiv angewählt sein. In unserem Falle wählen wir eine logarithmische Skalierung, um Erhebungen im Histogramm besser zu erkennen.

Man kann deutlich erkennen, dass das Histogramm des Originalvolumens nicht den gesamten Wertebereich ausfüllt. Im rechten Bereich des Histogramms sind leere Stellen. Bei dem mit der **Hdrw Methode** gefilterten Volumen erstreckt sich das Histogramm über den gesamten Wertebereich. Auch kann man kleinere Unterschiede bei den einzelnen Peaks wahrnehmen.

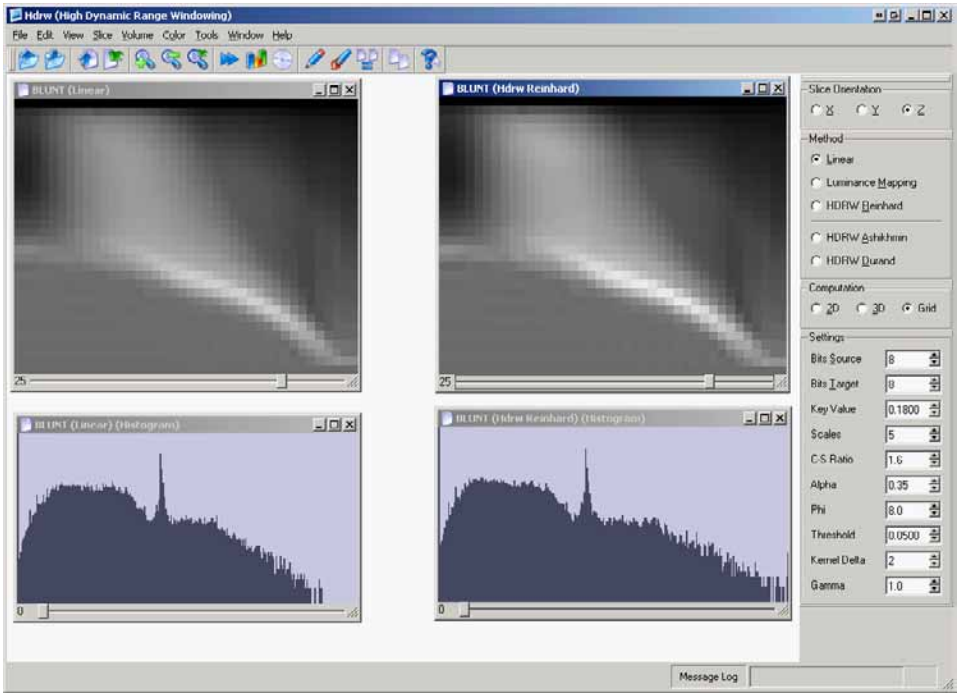

Abbildung 145: Die Histogramme der Volumen im Vergleich (Images\Cernik\BluntHistogramm.png)

4.1.6 Curvilineare Volumen speichern

Das neu berechnete Volumen kann nun abgespeichert werden, wahlweise nur als *SCALAR* File, wobei hier dann die Vektorinformation verloren geht, oder als *VECTOR* File. Die Vektoren werden dann mit dem neu berechneten Betrag entsprechend skaliert. Um das neue *Blunt* Volumen als *VECTOR* File zu speichern, muss dieses selektiert werden, danach ist das Speichern mittels der Tastenkombination **Ctrl+S** möglich oder direkt über das Menü **File > Save Volume As**.

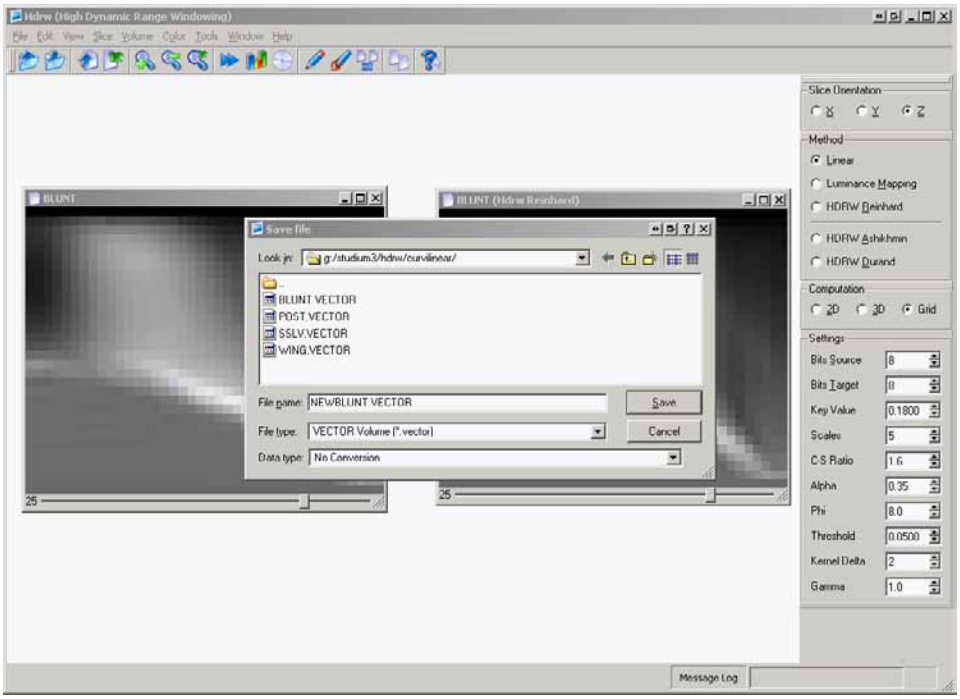

Abbildung 146: Curvilineare Volumen speichern (Images\Cernik\FileSave.png)

Der **File Type** ist bereits standardmäßig auf **VECTOR Volume** eingestellt, sofern ein *VECTOR* Volumen geladen wurde. Der File Type lässt sich bei Bedarf beliebig ändern, so kann man die Volumen auch im *SLC* Format speichern, jedoch muss bedacht werden, dass dann die *GRID* Information verloren geht und die Biegung des Volumens nicht mit abgespeichert wird.

Auch ist es möglich das Volumen direkt als *.vector.gz* oder *.scalar.gz* komprimiert im *GZ* Format abzulegen.

4.1.7 Darstellung von Vektorkomponenten als Farben

Da *VECTOR* Volumen mehr enthalten als nur den Betrag der Vektoren - nämlich die einzelnen Komponenten eines Vektors, welche jedoch in der normalen Darstellung nicht visualisiert werden - gibt es noch die Möglichkeit, das Volumen entsprechend der Vektor Komponenten einzufärben.

Als erstes wählen wir den Farbraum *Yxy* direkt über **Alt+Y** oder im Menü über **Color > Color Space > Yxy**. Es lässt sich auch alternativ das *HSV* Modell einsetzen, was im Falle des *Blunt* Volumens jedoch schlechtere Ergebnisse erzielt, dafür jedoch etwas schneller zu berechnen ist.

Die Konversion geschieht über die Tastenkombination **Alt+C** oder direkt im Menü über **Color > Convert Vector to Color**. Dies funktioniert nicht auf *Scalar* Volumen, da diese nicht die nötige Information mitbringen. Die 3 Komponenten jedes Vektors (x,y,z) werden dabei als *RGB* Komponenten interpretiert, wobei jeweils der Absolutbetrag jeder Komponente betrachtet wird.

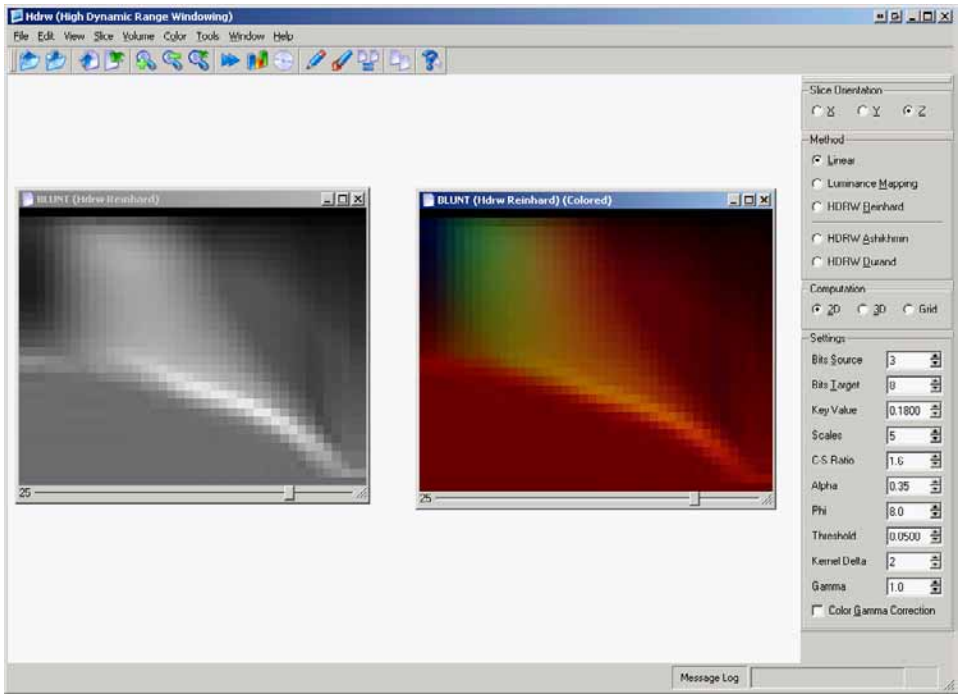

Abbildung 147: Rechts sieht man das eingefärbte Volumen (Images\Cernik\ColoredBlunt.png)

Deutlich kann man erkennen, dass das eingefärbte Volumen in Slice 25 links oben mehr Grün- und Blautöne enthält, der Bereich links unten jedoch eher aus Rottönen besteht. Dies liegt daran, dass sich der Betrag der einzelnen Komponenten der Vektoren über das Volumen hinweg verschieden darstellt.

4.2 Curvilineare Volumendaten

4.2.1 Beschreibung curvilinearer Volumendaten

Anschaulich betrachtet entstehen curvilineare Volumen, wenn man kartesische Gitter mit Hilfe einer analytischen Funktion "biegt".

Curvilineare Volumendaten sind Daten, in denen die Gitterpunkte analytisch gegeben sind. Es handelt sich deswegen um strukturierte Gitter. Die Topologie, also die Nachbarschaftsbeziehung der einzelnen Knoten, ist regelmäßig, in unserem Fall hat jeder Knotenpunkt 4 Nachbarn, sofern es kein Randknotenpunkt ist.
Der Zellentyp im Volumen ist gleichmäßig. Das Hdrw Programm kann nur mit Hexaedern als Zelltypen arbeiten. Lediglich die Geometrie der curvilinearen Volumen ist irregulär.

Die curvilinearen Volumendaten haben eine Ausdehnung in drei Dimensionen und können in zwei Grundkomponenten zerlegt werden: Die *Merkmalswerte* und die *Datenpunkte*.

Die Merkmalswerte geben hierbei die konkreten Vektoren oder die entsprechenden skalaren Werte an. Die Merkmalswerte selbst können in einem 3-dimensionalen Array regelmäßig abgelegt werden. Das Array hat dabei eine definierte Dimension in alle drei Richtungen. Die curvilinearen Volumen bilden also anschaulich gesprochen keine Pyramiden oder dergleichen, sondern sind immer gebogene Hexaeder.

Die Datenpunkte zeigen, an welchen Stellen sich die jeweiligen Merkmalswerte befinden. Die Gitterstruktur wird also über die Datenpunkte definiert.
Da Merkmalswerte und Datenpunkte als zwei verschiedene Komponenten eines curvilinearen Volumens betrachtet werden können, werden diese in den folgenden Formaten auch in getrennten Dateien abgespeichert.

Auf diese Weise können die Merkmalswerte unabhängig von den Datenpunkten ersetzt werden. Man kann dieselben Datenpunkte also z. B. einmal für Vektor Merkmalswerte und einmal für skalare Merkmalswerte verwenden.

Um einen Eindruck von curvilinearen Gittern zu vermitteln, hier ein Beispiel, welches den Unterschied zwischen einem kartesischen und einem curvilinearen Gitter im 2D Fall visualisiert:

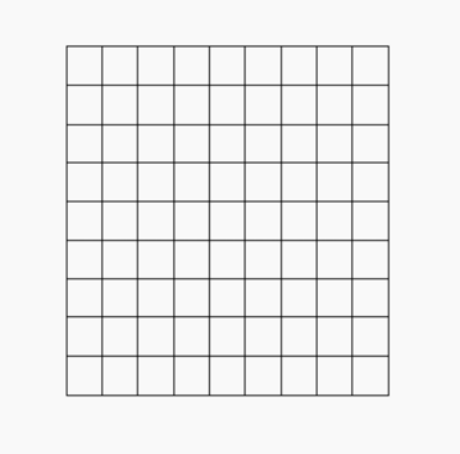

Abbildung 148: Kartesisches Gitter (Images\Cernik\VolGraph1.png)

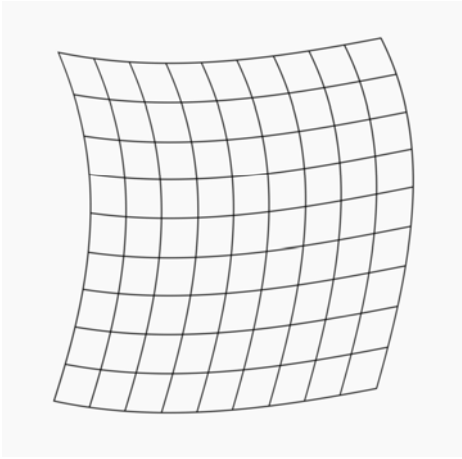

Abbildung 149: Curvilineares 2D-Gitter (Images\Cernik\VolGraph2.png)

Für die Datenwerte wird das *SCALAR* Format unterstützt, welches skalare Werte fassen kann und das *VECTOR* Format, welches vektorielle Werte an den Datenpunkten abspeichern kann. Die Information über den Aufbau des Gitters kommt aus den *.grid* Dateien oder den *.mesh* Dateien. Zu einem *.scalar* Format muss also mindestens ein *.grid* oder *.mesh* mit demselben Dateinamen vor der Endung existieren. Existiert eine *GRID* und eine *MESH* Datei mit demselben Namen, so öffnet das Programm die *GRID* Datei, da diese mehr Informationen tragen kann als das *MESH* Format.

Merkmalswerte	Datenpunkte
.SCALAR Dateien	.MESH Dateien
.VECTOR Dateien	.GRID Dateien

Abbildung 150: Dateiformate (Images\Cernik\merkmal.png)

Auf Seiten der Merkmalswerte haben wir, wie in **Abbildung 150** gezeigt, die Dateiformate *.scalar* und *.vector*. Die Datenpunkte für die Gitterstruktur werden dagegen in den Dateiformaten *.mesh* und *.grid* bereitgestellt.

4.2.2 Die Datenformate für Merkmalswerte

4.2.2.1 Das *SCALAR* Format

Das *SCALAR* Format dient dazu skalare Merkmalswerte curvilinearer Daten zu beschreiben. Das Format beschreibt keine Datenpunkte sondern nur die Datenwerte an den einzelnen Datenpunkten. Dateien im *SCALAR* Format tragen die Dateiendung *.scalar*.

Der Header des *SCALAR* Formates wird im ASCII Format abgelegt, der Rest sind Float-Werte, die direkt binär abgespeichert sind. Als erstes kommt in der *SCALAR* Datei das *Header Cookie*. Eine Kennzeichnung die deutlich macht, dass es sich hier um eine Datei im *SCALAR* Format handelt. Das *Header Cookie* lautet "110011" und wird durch das *Newline* Zeichen "\n" (oder Hexadezimal A0) abgeschlossen. Das *Header Cookie* ist dabei direkt als ASCII Text in der Datei zu finden.

Als Nächstes folgt direkt die Versionsnummer der *SCALAR* Datei. Im Hdrw Programm wird generell mit Version 1 gearbeitet, hier sollte also derzeit immer eine

ASCII 1 stehen. Nach der Versionsnummer folgt ein Leerzeichen (Hexadezimal 20), um die Versionsnummer abzutrennen.

Die *Number of Items* werden durch die nächste Zahl im Header repräsentiert, in unserem Fall sollte diese Zahl immer auf 3 sein, da wir 3-dimensionale Volumen betrachten. Abgetrennt wird dies durch ein weiteres Leerzeichen.

Jetzt folgen die konkreten Angaben zur Größe des Volumens. Dies sind drei Integer Werte im ASCII Format, welche ebenfalls durch Leerzeichen getrennt sind. Als erstes folgt die Größe des Volumens in X-Richtung, dann die Größe in Y-Richtung und schließlich die Größe des Volumens in Z-Richtung. Abgeschlossen werden die Größenangaben des Volumens durch eine *Newline* Zeichen "\n". Der Header ist damit auch beendet und es folgen direkt die Nutzdaten.

Die Nutzdaten des *SCALAR* Volumens sind Floats, die binär in der *MSB* Darstellung gespeichert werden. Die Daten erscheinen dann im normalen Verlauf des Arrays nacheinander, d. h. die Daten können direkt in ein Array übernommen werden, das die Größe des Volumens hat.

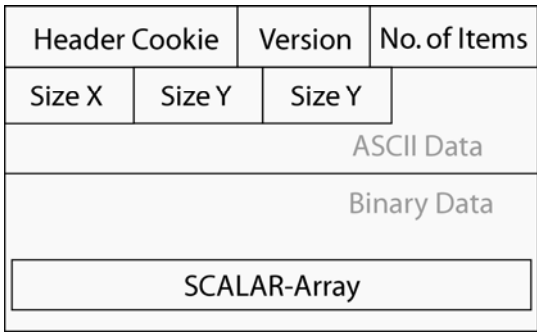

Abbildung 151: Schematischer Aufbau des *SCALAR* Formates (Images\Cernik\scalarformat.png)

4.2.2.2 Das *VECTOR* Format

Das *VECTOR* Format dient dazu vektorielle Merkmalswerte curvilinearer Daten zu beschreiben. Das Format beschreibt keine Datenpunkte sondern nur die Datenwerte an den einzelnen Datenpunkten. Dateien im *VECTOR* Format tragen die Dateiendung *.vector*. Da Vektoren aus 3 Komponenten bestehen, benötigt dieses Format (wenn man vom Header absieht) etwa 3 mal mehr Platz als das *SCALAR* Format.

Der Header des *VECTOR* Formates wird im ASCII Format abgelegt, der Rest sind Float-Werte, die direkt binär abgespeichert sind. Als erstes kommt in der *VECTOR* Datei das *Header Cookie*. Eine Kennzeichnung die deutlich macht, dass es sich hier um eine Datei im *VECTOR* Format handelt. Das *Header Cookie* lautet "330033" und wird durch das *Newline* Zeichen "\n" (oder Hexadezimal A0) abgeschlossen. Das *Header Cookie* ist dabei direkt als ASCII Text in der Datei zu finden.

Als nächstes folgt direkt die Versionsnummer der *VECTOR* Datei. Im Hdrw Programm wird generell mit Version 1 gearbeitet, hier sollte also derzeit immer eine ASCII 1 stehen. Nach der Versionsnummer folgt ein Leerzeichen (Hexadezimal 20), um die Versionsnummer abzutrennen.

Die *Number of Items* werden durch die nächste Zahl im Header repräsentiert, in unserem Fall sollte diese Zahl immer auf 3 sein, da wir 3-dimensionale Volumen betrachten. Abgetrennt wird dies durch ein weiteres Leerzeichen.

Jetzt folgen die konkreten Angaben zur Größe des Volumens. Dies sind drei Integer Werte im ASCII Format, die ebenfalls durch Leerzeichen getrennt sind. Als erstes folgt die Größe des Volumens in X-Richtung, dann die Größe in Y-Richtung und schließlich die Größe des Volumens in Z-Richtung. Abgeschlossen werden die Größenangaben des Volumens durch eine *Newline* Zeichen "\n". Der Header ist damit auch beendet und es folgen direkt die Nutzdaten.

Die Nutzdaten des *VECTOR* Volumens sind Floats, die binär in der *MSB* Darstellung gespeichert werden. Die Daten werden in 3 aufeinander folgenden Arrays abgelegt, wobei erst die X-Komponente aller Vektoren in einem Array erfolgt, dann die Y-Komponente und schließlich die Z-Komponente. Jedes einzelne Array ist dabei so organisiert wie bei den *SCALAR* Dateien.

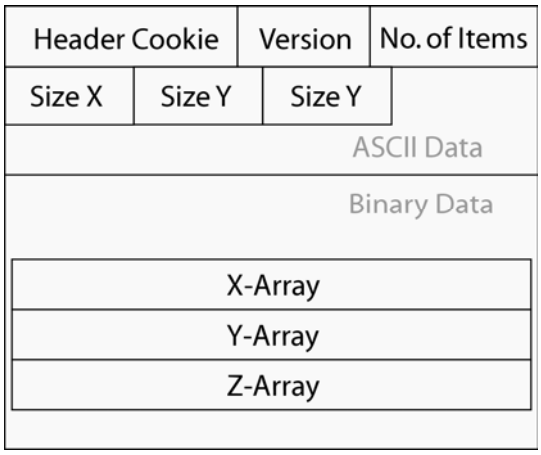

Abbildung 152: Schematischer Aufbau des *VECTOR* Formates (Images\Cernik\vectorformat.png)

4.2.3 Die Datenformate für die Datenpunkte

4.2.3.1 Das *MESH* Format

Das *MESH* Format ist ein sehr einfaches ASCII Text Format, das ein Grid beschreibt, welches nicht gebogen ist. Im Mesh Format selbst werden sozusagen nur die Randpunkte des Volumens definiert. Das Format selbst ist sehr simpel gehalten, man erkennt es an der Endung *.mesh*. Ein Header zur Identifikation existiert nicht.

Die Datei beginnt direkt mit den Angaben zur Größe des Volumens. 3 Integer Werte im ASCII Format, jeweils getrennt durch ein Leerzeichen, geben die Menge der Datenpunkte in X-, Y- und Z-Richtung an. Abgeschlossen wird die Dimensionierung des Volumens durch ein *Newline* "\n" Zeichen.

Nun folgen Zeilen, die jeweils durch ein Newline Zeichen beendet werden und Float Werte in ASCII Darstellung enthalten. Hat man als Beispiel eine Dimensionierung von 64 Datenpunkten in X-Richtung, 64 Datenpunkten in Y-Richtung und 64 Datenpunkten in Z-Richtung, so folgen nach der Größenangabe von "64 64 64" genau 192 Zeilen, wobei die ersten 64 Zeilen die X-Koordinaten entlang der X-Achse außen am Volumen angeben. Die nächsten 64 Zeilen geben die Y-Koordinaten entlang der Y-Achse außen am Volumen an und die restlichen 64 Zeilen geben letztendlich die Z-Koordinaten entlang der Z-Achse außen am Volumen an.

Es handelt sich bei den *.mesh* Files also eigentlich um keine curvilinearen Volumen, weswegen die *.grid* Files vom Programm bevorzugt werden, wenn ein *.grid* File und ein *.mesh* File mit gleichem Namen vorliegt.

4.2.3.2 Das *GRID* Format

Das *GRID* Format beschreibt Datenpunkte im 3-dimensionalen Raum für eine korrespondierende *SCALAR* oder *VECTOR* Datei. Dabei ist das *VECTOR* Format und das *GRID* Format prinzipiell komplett identisch, mit der Ausnahme dass hier eben die Gitterstruktur selbst definiert wird.

Der Header des *GRID* Formates wird im ASCII Format abgelegt, der Rest sind Float-Werte, die direkt binär abgespeichert sind. Als erstes kommt in der *GRID* Datei das *Header Cookie*. Eine Kennzeichnung die deutlich macht, dass es sich hier um eine Datei im *GRID* Format handelt. Das *Header Cookie* lautet "330033" und wird durch das *Newline* Zeichen "\n" (oder Hexadezimal A0) abgeschlossen. Das *Header Cookie* ist dabei direkt als ASCII Text in der Datei zu finden.

Als nächstes folgt direkt die Versionsnummer der *GRID* Datei. Im Hdrw Programm wird generell mit Version 1 gearbeitet, hier sollte also derzeit immer eine ASCII 1 stehen. Nach der Versionsnummer folgt ein Leerzeichen (Hexadezimal 20), um die Versionsnummer abzutrennen.

Die *Number of Items* werden durch die nächste Zahl im Header repräsentiert, in unserem Fall sollte diese Zahl immer auf 3 sein, da wir 3-dimensionale Volumen betrachten. Abgetrennt wird dies durch ein weiteres Leerzeichen.

Jetzt folgen die konkreten Angaben zur Größe des Volumens. Dies sind drei Integer Werte im ASCII Format, die ebenfalls durch Leerzeichen getrennt sind. Als erstes folgt die Größe des Volumens in X-Richtung, dann die Größe in Y-Richtung und schließlich die Größe des Volumens in Z-Richtung. Abgeschlossen werden

die Größenangaben des Volumens durch eine *Newline* Zeichen "\n". Der Header ist damit auch beendet und es folgen direkt die Nutzdaten.

Die Nutzdaten des *GRID* Volumens sind Floats, die binär in der *MSB* Darstellung gespeichert werden. Die Daten werden in 3 aufeinander folgenden Arrays abgelegt, wobei erst die X-Komponente aller Datenpunkte in einem Array erfolgt, dann die Y-Komponente und schließlich die Z-Komponente. Jedes einzelne Array ist dabei so organisiert wie bei den *SCALAR* Dateien.

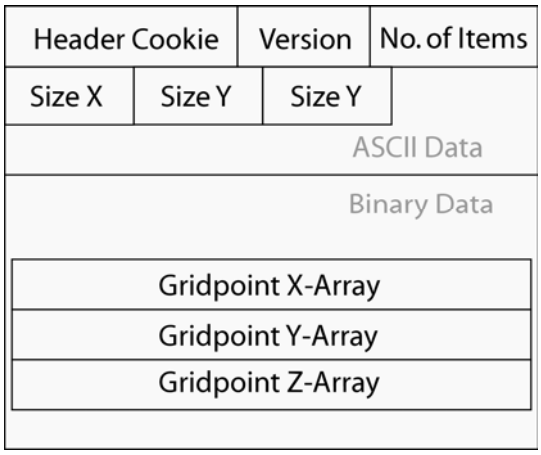

Abbildung 153: Schematischer Aufbau des *GRID* Formates (Images\Cernik\gridformat.png)

4.3 Die Reinhard Methode

4.3.1 Die konventionelle Reinhard Methode

Im Folgenden wird nun der Reinhard Algorithmus auf curvilinearen Daten besprochen, wenn die Krümmung des Gitters nicht berücksichtigt wird. Dies entspricht dem konventionellen Reinhard Algorithmus im 3D. Als Beispielvolumen wird das bereits vorgestellte *Blunt Fin* Volumen verwendet.

Um den Reinhard Algorithmus an die Krümmung des Volumens anzupassen, muss der konventionelle Algorithmus verändert werden. Dies wird im nächsten Kapitel besprochen.

4.3.1.1 Luminance Mapping

Eine globale Methode, die Helligkeit eines Volumens zu regulieren und anzupassen, stellt das Luminance Mapping dar. Dieses dient dazu, das Volumen auf eine gewisse durchschnittliche Helligkeit zu bringen. Es handelt sich hier um ein globales Illuminationsmodell, wobei die logarithmische Helligkeit des Volumens betrachtet und mit einem **Key Value** skaliert wird.

Der **Key Value** gibt die mittlere Intensität des Volumens an, umso höher der Wert ist, umso heller ist das gesamte Volumen im Mittel.
Um den **Key Value** zu berechnen, wird Formel (47) verwendet. Hierbei wird das logarithmische Mittel über das gesamte Volumen gebildet.

$$\bar{L}_w = \exp\left(\frac{1}{N} \cdot \sum_{x,y,z} \log(1+L_w(x,y,z))\right)-1 \tag{47}$$

Der so ermittelte **Key Value** $\bar{L}_w$ des Volumens ist jedoch oft nicht optimal, d. h. meist ist das Volumen zu dunkel oder zu hell. Dies tritt besonders oft beim Windowing auf, bei welchem ein Teil der möglichen Werte gar nicht verwendet wird und sich die Dynamik auf diese Weise nur in einem gewissen Intensitätsbereich abspielt. N bezeichnet die Anzahl aller Voxel. $L_w(x,y,z)$ gibt die Intensität eines Voxels an der Stelle (x,y,z) an.

Ein guter **Key Value** bei Bildern in der Photographie ist meist $0{,}18$, allerdings kann man dies so nicht ohne Weiteres auf curvilineare Volumen übertragen, da sich hier die Intensitäten meist über den Betrag von Vektoren ergeben und der optimale **Key Value** für ein optisch ansprechendes Ergebnis eventuell anders gewählt werden muss. Der **Key Value** als neuer Parameter, auf welchen man ein Volumen bringen kann, wird ab jetzt als a bezeichnet.

Um das gesamte Volumen auf einen definierten **Key Value** a zu bringen, bedienen wir uns Formel (48).

$$L(x,y,z) = \frac{a}{\bar{L}_w} \cdot L_w(x,y,z) \tag{48}$$

Die Vorgehensweise ist dabei recht einfach, es wird jeder Voxel mit dem Wert $a/\bar{L}_w$ skaliert. Da man durch $\bar{L}_w$ teilt, wird der **Key** der Szene erst auf 1 gebracht und danach mit a multipliziert, um genau den **Key Value** a für das gesamte Volumen zu erhalten.

Da bei der Umrechnung des **Key Values** unter Umständen sehr hohe Intensitäten entstehen können, kann man bei Bedarf die Intensitäten über die Formel (49) anpassen.

$$L_d(x,y,z) = \frac{L(x,y,z)\cdot(1+\frac{L(x,y,z)}{L_{max}^2})}{1+V_{i-1}(x,y,z)} \tag{49}$$

Die Maximalintensität sollte dabei für ein gutes Ergebnis vorher ermittelt werden, indem man über das gesamte Volumen die maximale Intensität sucht. Die Berechnung sorgt des Weiteren dafür, dass die Intensitäten kontrolliert gegen den Maximalwert gehen und kein hartes Abschneiden am Maximalwert nötig ist. $L_d(x,y,z)$ ist dann die resultierende Intensität nach dem Luminance Mapping.

Die gesamte Berechnung des Luminance Mapping betrachtet immer lokale Voxel und keine Nachbarschaftsbeziehungen oder Entfernungen zwischen den Voxeln. Deswegen ist dieser Vorverarbeitungsschritt unabhängig von der Art des Gitters und muss für curvilineare Volumen nicht geändert werden.

Die Wahl des **Key Values** a sollte vom Volumen abhängig gemacht werden, in Tests hat es sich jedoch als sinnvoll erwiesen mit einem Key Value von 0.18 zu beginnen und je nach Intensität des Volumens diesen entsprechend einer optimalen visuellen Ausgabe anzupassen.

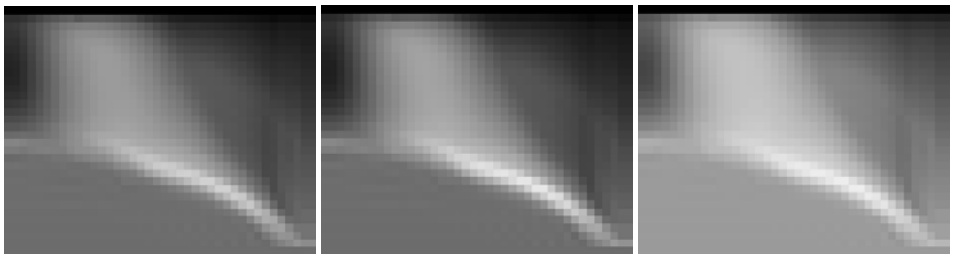

Abbildung 154: Links Original, Mitte a=0.18, rechts a=0.50 (Images\Cernik\BluntKey050.png)

In **Abbildung 154** sieht man nun die Schicht 25 (Slice 25) mit verschiedenen Einstellungen für den **Key Value**. Das Bild ganz rechts ist das Originalvolumen ohne **Luminance Mapping** und ohne weitere Intensitätsanpassungen. Optisch wirkt dies schon recht ansprechend und bedarf keiner großen Korrektur mehr.

Das Bild in der Mitte zeigt nun die gleiche Schicht mit einem **Key Value** von 0.18. Der Unterschied zum Original ist sehr gering, man erkennt jedoch, dass dieses Bild leicht heller ist. Ein **Key Value** von 0.18 bietet sich also auf den ersten Blick scheinbar für dieses Volumen an.

Das rechte Bild zeigt noch einmal die gleiche Schicht, aber diesmal mit einem **Key Value** von 0.50. Man sieht, dass es deutlich heller als die anderen beiden Bilder ist und in diesem Fall eher zu hell wirkt. Die optimale Einstellung scheint also für das *Blunt Fin* Volumen bei 0.18 zu liegen.

Da der **Key Value** jedoch über das ganze Volumen berechnet wird, gestaltet sich die Suche nach der optimalen **Key Value** nicht ganz so einfach, wie man eventuell auf den ersten Blick annehmen könnte. Macht man nun für die Schicht 17 das Gleiche wie für Schicht 25, erhält man das Resultat aus **Abbildung 155**.

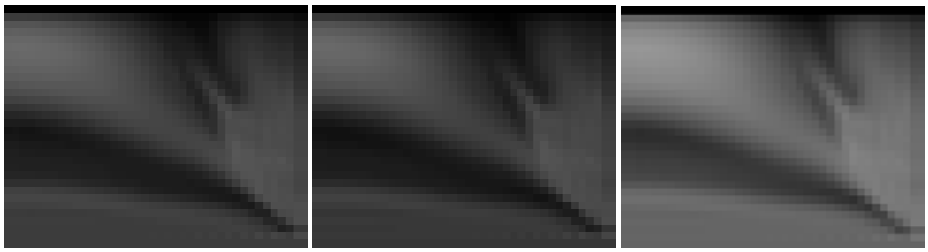

Abbildung 155: Links Original, Mitte a=0.18, rechts a=0.50 (Images\Cernik\BluntSlice17Key050.png)

Bei der Schicht 17 sieht man diesmal nur einen sehr geringen Unterschied zwischen dem Original und einem **Key Value** von 0.18, was daran liegt, dass der **Key Value** des Originals bereits sehr nah an 0.18 liegt. Interessant ist jedoch, dass ein **Key Value** von 0.50 diesmal deutlich ausgeglichener und besser scheint als in der Schicht 25. Dies liegt eben auch daran, dass der **Key Value** für alle Schichten gleichzeitig gilt und im 3-Dimensionalen berechnet wird.

Es kann also durchaus darauf ankommen, welchen Bereich des Volumens man genauer betrachten will und demzufolge sollte auch der **Key Value** gewählt werden. In den Außenbereichen des *Blunt* Volumens, wo eher geringe Intensitäten vorherrschen, kann man mit hohem **Key Value** mehr erkennen als in Innenbereichen. Dort sind die Intensitäten eher hoch.

Das **Luminance Mapping** lässt sich, wie gezeigt, im *Hdrw* Programm testen, hierfür muss man lediglich ein Volumen laden und die Methode auf **Luminance Mapping** einstellen. Der Key Value a lässt sich dann direkt über den Parameter **Key Value** bei den **Settings** eingeben.

4.3.1.2 Der Reinhard-Algorithmus auf Volumendaten

Der nun folgende Hdrw Algorithmus arbeitet nicht auf curvilinearen Volumen, das heißt, man kann ihn im Programm auf curvilineare Volumen anwenden, er beachtet jedoch nicht die Gitter Information und geht deswegen von kartesischen Gittern aus.

Anders als das Luminance Mapping wird beim Hdrw Algorithmus kein globaler Operator verwendet. Hier wird die Helligkeit adaptiv und lokal für jeden Pixel anhand bestimmter Kriterien ermittelt. In Volumendatensätzen mit sehr großer Dynamik kann durch Hdrw die optische Visualisierung deutlich verbessert werden.

Man wird auch sehen, dass bei Hdrw Methoden, anders als beim Luminance Mapping, die Gitter Information durchaus eine Rolle spielt. Wird diese im naiven Fall nicht beachtet, erhält man andere Ergebnisse, die sich umso mehr von den korrekten Ergebnissen unterscheiden, je stärker die "Biegung" des curvilinearen Volumens an der entsprechenden Stelle ist.

Um die Pixel lokal entsprechend zu gewichten, wird ein Filter benötigt, im 3D bietet sich dafür ein Gauß-Filter an, konkret wird auf eine leicht abgewandelte Form, das *Blommaert* Modell, zurückgegriffen.

$$R_i(x,y,z) = \exp\left(-\frac{x^2+y^2+z^2}{(\alpha s^i)^2}\right) \tag{50}$$

In Formel (50) ist die Berechnung des Filters angegeben. α ist hierbei in unseren Beispielen auf 0.35 gesetzt. Technisch müsste man für den korrekten Filter noch einen Vorfaktor berechnen, welcher jedoch wegfällt, da der Filter im Algorithmus selbst normiert wird, d. h. alle Filterelemente müssen aufsummiert 1 ergeben.

Der Parameter s gibt die **Center-Surround Ratio** des Filters an. Im Algorithmus selbst werden verschiedene Scales getestet, um den optimalen Filter zu berechnen. Die Filter können bei kartesischen Gittern für verschiedene Scales direkt vorberechnet werden. Auch für curvilineare Gitter könnte man diese Filter verwenden, allerdings benötigt man dann später eine Hexaeder Interpolation, welche sehr langsam ist und sich rechnerisch nicht lohnt.

Die Filterberechnung muss bei curvilinearen Gittern unter Umständen also bereits anders erfolgen als bei kartesischen Gittern. Das Problem, das sich mit dem Filter auf curvilinearen Gittern ergibt, sieht man konkret bei der Faltung.

$$V_i(x,y,z) = (L * R_i)(x,y,z) \tag{51}$$

Formel (51) beschreibt nun die konkrete Faltung für eine vorgegebene Scale. V_i ist der Ergebnisvoxel nach der Faltung. $L(x,y,z)$ entspricht dabei der skalierten Helligkeit, die weiter oben definiert wurde.

Hier ergibt sich nun auch das Problem mit curvilinearen Volumen. In kartesischen Gittern kann die Faltung so ablaufen, lässt man jedoch die Faltung im curvilinearen Gitter derartig berechnen und benutzt die lokalen Arraykoordinaten des Filterkernels, ist der Filterkernel in Weltkoordinaten ebenfalls gemäß der curvilinearen Biegung gebogen.

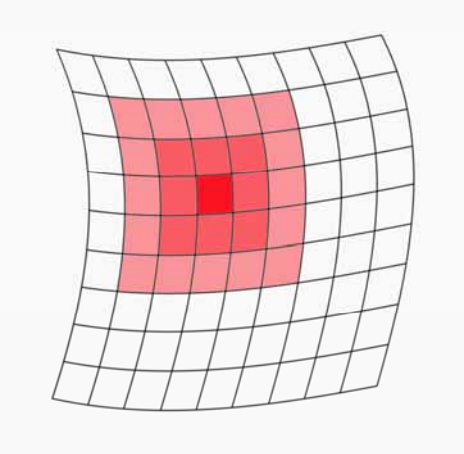

Abbildung 156: Filterkernel im curvilinearen Volumen (Images\Cernik\VolGraph3.png)

In **Abbildung 156** sieht man wie der Filterkernel ohne spezielle Anpassung in Arraykoordinaten in einem curvilinearen Volumen aussieht. Hierbei wird nur eine 2D-Schicht des Volumens betrachtet. Es handelt sich bei dem Filterkernel um einen 5×5 Kernel, der über das Volumen läuft. Korrekterweise müsste bei der Faltung entweder der Filterkernel in Weltkoordinaten berechnet werden oder aber das zugrunde liegende Volumen müsste in Weltkoordinaten ausgelesen werden. Beide Vorgehensweisen würden wieder zu einer korrekten Visualisierung im curvilinearen Volumen führen. Das Programm entscheidet sich für die Möglichkeit den Kernel in Weltkoordinaten zu überführen und "gerade" zu biegen. Dies wird in einem späteren Kapitel verdeutlicht.

Im Falle des regulären Reinhard Algorithmus gehen wir nun davon, dass es sich um ein kartesisches Volumen handelt und das *Grid* File nicht existiert. Auf die Unterschiede zur korrekten Berechnung wird später eingegangen.

Hat man die Faltungsfunktion, kann man die Aktivität $act_i(x,y,z)$ an jedem Voxel über Formel (52) berechnen.

$$act_i(x,y,z) = \frac{V_{i-1}(x,y,z) - V_i(x,y,z)}{2^{\phi}a/(s^{i-1})^2 + V_{i-1}(x,y,z)} \quad (52)$$

Die Aktivitätsfunktion wird auch als *Center-Surround* Funktion bezeichnet. Der Parameter φ wird dabei für die gewünschte Schärfe verwendet und der Parameter a gibt den gewünschten **Key Value** an. Die Funktion $act_i(x,y,z)$ gibt die Aktivität am Voxel (x,y,z) für eine bestimmte **Center Surround Ratio** s an. In Regionen großer Dynamik liefert die Aktivität einen großen Absolutbetrag, in Regionen kleiner Dynamik ist dieser Betrag eher gering.

Die konkrete Arbeitsweise des Algorithmus funktioniert nun so, dass bei der niedrigsten Scale begonnen wird und Formel (53) auswertet wird.

$$|act_i(x,y,z)| > \varepsilon \quad (53)$$

i läuft dabei von der niedrigsten Scale bis zur höchsten eingestellten Scale, welche im Programm über den Parameter **Scales** unter **Settings** eingestellt werden kann. Wird für eine Scale der **Threshold** ε überschritten, so bricht der Algorithmus ab und berechnet die neue Helligkeit des Voxels über Formel (54).

$$L_d(x,y,z) = \frac{L(x,y,z)}{1+V_{i-1}(x,y,z)} \quad (54)$$

Genau wie beim Luminance Mapping kann die Darstellung besser an den Maximalwert angepasst werden, wenn L_d nach Formel (55) berechnet wird.

$$L_d(x,y,z) = \frac{L(x,y,z)\cdot(1+\frac{L(x,y,z)}{L_{max}^2})}{1+V_{i-1}(x,y,z)} \quad (55)$$

Besonders in Bereichen von hoher Intensität passt sich mit dieser Formel der Wert von L_d besser an.

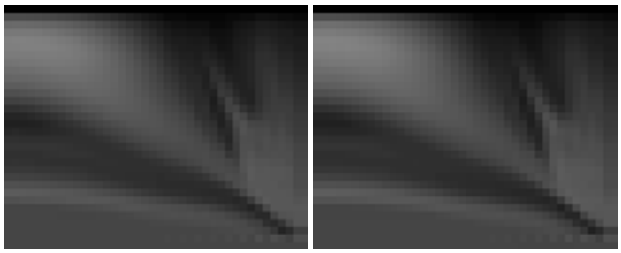

Abbildung 157: Links Scales auf 2, rechts Scales auf 10 (Images\Cernik\BluntScale10.png)

In **Abbildung 157** sieht man nun Slice 19 des *Blunt* Volumens in 2 Aufnahmen, das linke Bild wurde mit dem Parameter **Scales** 2, das rechte Bild mit 10 möglichen **Scales** erzeugt. Auf den ersten Blick sieht man kaum eine Veränderung, was auch daran liegt, dass die curvilinearen Volumen äußerst klein sind. Das rechte Bild bietet durch die höhere Zahl an Scales eine feinere Abstufung in der Dynamik. Da zwischen den Bildern jedoch ein echter Unterschied besteht, sieht man erst richtig, wenn man das Differenzbild betrachtet.

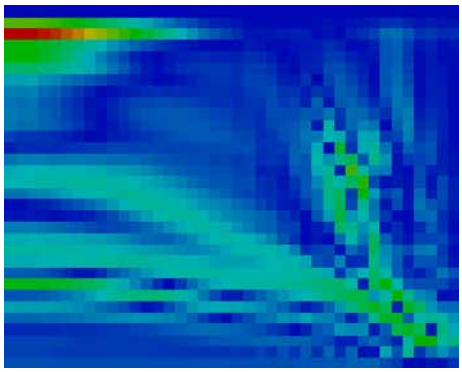

Abbildung 158: Differenzbild von verschiedenen Scales (Images\Cernik\BluntScaleCompare.png)

Umso roter desto größer die Änderung an dieser Stelle zwischen den beiden Bildern in **Abbildung 157** und umso blauer desto geringer die Änderung. Man sieht an dem Differenzbild in **Abbildung 158**, dass die beiden Bilder durchaus verschieden sind. Die größten Differenzen finden sich oft dort, wo sich die Helligkeit des Voxels am meisten ändert und an Kanten. Hier hat die größere Anzahl an Scales einen größeren Einfluss.

Man muss jedoch bedenken, dass die curvilinearen Volumen äußerst klein sind. So hat ein Slice im *Blunt* Volumen gerade einmal eine Abmessung von 40×32 Pixeln. Eine Scale von 10 ist bereits sehr groß, da der Kernel hier im Verhältnis bereits recht klein ist. Noch größere Scales bringen auf einem derartigen Kernel praktisch keinen Unterschied mehr.

4.3.2 Die Reinhard Methode für curvilineare Gitter

Die Reinhard Methode - wie in Kapitel 4.3.1, Seite 167 beschrieben - arbeitet auf kartesischen Gittern. Da es sich bei den Daten, die wir hier betrachten, um Volumendaten auf curvilinearen Gittern handelt, muss die Methode angepasst werden, um eine korrekte visuelle Präsentation zu ermöglichen.

Der entscheidende Teil, an welchem die Gitterinformation mit eingeht, ist die Faltung des Filterkernels mit dem entsprechenden Volumenteil. Es sind prinzipiell zwei mögliche Ansätze denkbar, um die Faltung korrekt für curvilineare Gitter auszuführen. Betrachtet man das curvilineare Gitter eingebettet in Weltkoordinaten und berechnet den Filterkernel in Weltkoordinaten, müsste bei der Faltung eine Interpolation in Hexaedern geschehen, um den korrekten Datenwert innerhalb des curvilinearen Volumens bei der Faltung zu erhalten. Die andere Möglichkeit, die auch im Programm verwendet wird und deutlich schneller zu berechnen ist, betrachtet die Datenwerte des Volumens in lokalen Arraykoordinaten, jedoch wird nicht der Filterkernel einmal global berechnet, sondern für jeden einzelnen Datenpunkt entsprechend "gebogen", damit der Filterkernel in Weltkoordinaten nicht verzerrt ist, sondern eine symmetrische Gaußkugel bildet.

Die eine Methode bedient sich also der Möglichkeit der Interpolation in Hexaedern, die andere berechnet den Filter individuell für jeden Voxel. Die Methode zur Interpolation wird im Folgenden kurz dargestellt, dann wird genauer auf die im Programm verwendete Methode der "Filterbiegung" eingegangen.

4.3.2.1 Interpolation in Hexaedern

Der Zelltyp in curvilinearen Gittern ist normalerweise ein Hexaeder, also ein Körper mit 8 Ecken die mit 12 Kanten verbunden sind. In Weltkoordinaten betrachtet kann das curvilineare Volumen beliebig gebogen sein und die Hexaeder können beliebig im Raum liegen.

Bei der Interpolation mit Hexaedern wird im Reinhard Algorithmus so vorgegangen, dass die Filterkernel für jede Scale einmal global berechnet werden. Die Daten in den Filterkernels entsprechen Weltkoordinaten. Die Formel (56) gibt noch einmal die Faltung an.

$$V_i(x,y,z) = (L * R_i)(x,y,z) \tag{56}$$

Problematisch ist hier $L(x,y,z)$, da L in Weltkoordinaten vorliegen muss. Um L korrekt zu bestimmen, muss also ermittelt werden, in welchem Hexaeder (x,y,z) liegt und dann muss eine Interpolation in diesem Hexaeder erfolgen, um den korrekten Datenwert an der Stelle (x,y,z) zu bestimmen. Die Interpolation lässt sich aus bilinearen Interpolationen zusammensetzen. Das Hauptproblem, bezüglich der Rechenzeit bei diesem Verfahren, ist jedoch nicht die Interpolation im Hexaeder, sondern das Finden des richtigen Hexaeders.

Um zu zeigen, welcher Aufwand betrieben werden muss, um den richtigen Hexaeder zu finden, wird die Vorgehensweise kurz grob skizziert (Peikert, 2004).

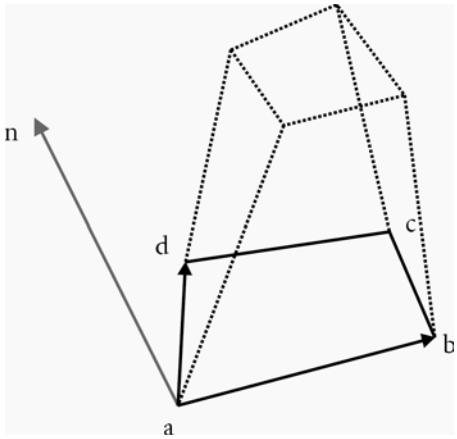

Abbildung 159: Hexaeder Interpolation (Images\Cernik\hexaeder1.png)

In **Abbildung 159** ist eine Hexaeder Zelle aus dem Volumen dargestellt. n gibt dabei die Normale an. Die Punkte a, b, c, d liegen nicht zwingend in einer Ebene. Es soll nun als erstes getestet werden, ob sich ein Punkt p im Hexaeder befindet oder nicht.

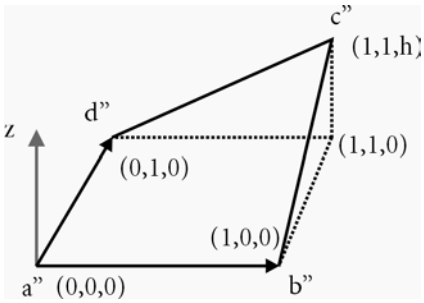

Abbildung 160: Hexaeder nach Transformation (Images\Cernik\hexaeder2.png)

Um dies festzustellen, wird der Punkt a auf den Ursprung $(0,0,0)$ transformiert. Auf die restlichen Punkte wird dieselbe Transformation angewendet. Wir nennen diese dann b', c', d' und p'. Als nächstes muss eine lineare Transformation gesucht werden, welche die Punkte b', c' und d' auf $(1,0,0)$, $(1,1,h)$ und $(0,1,0)$ abbildet. h wird dabei über $(b' \times d') \cdot c'$ berechnet.

Abbildung 160 visualisiert die Situation. Die neu transformierten Punkte werden als a'', b'', c'', d'' und p'' bezeichnet. Wichtig ist, dass die Punkte gegen den Uhrzeigersinn angeordnet sind. Wenn $h=0$ ist, kann man an der Stelle bereits abbrechen und den Inklusionstest über eine einfache If-Abfrage mit der entsprechenden Ebene lösen, welche planar ist. Die Transformation lässt sich berechnen über die folgende Matrix-Multiplikation:

$$\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & h \end{pmatrix} \cdot \begin{pmatrix} b'_x & d'_x & c'_x \\ b'_y & d'_y & c'_y \\ b'_z & d'_z & c'_z \end{pmatrix}^{-1} \tag{57}$$

Die Projektion unserer Fläche, die durch a'', b'', c'', d'' aufgespannt wird, auf die xy-Ebene ist ein Einheitsquadrat. Deswegen entsprechen die x und y Koordinaten von p'' bereits den Interpolationskoordinaten. Die Interpolation der z-Koordinate ergibt sich über $z=xyh$.

Um nun zu berechnen, ob der Punkt p'' oberhalb der Oberfläche liegt, genügt es zu testen, ob $p''_z > hp''_x p''_y$ gilt. Damit der Punkt auch wirklich im Hexaeder liegt, muss dies für alle weiteren Flächen auch gezeigt werden. Die Interpolationskoordinaten erhält man zwar gleich mit, dennoch sieht man schnell, dass der Aufwand für die Hexaederinterpolation enorm ist.

Geht man nun von 5 **Scales** aus und einem Filterkernel der Größe 5×5×5 hat, so umfasst bereits der Filterkernel $5 \cdot 5 \cdot 5 = 125$ Voxel. Müssen effektiv 5 Scales durchgetestet werden, bis man die richtige Scale gefunden hat, so wären dies im *Worst Case* 625 Hexaederinterpolationen pro Voxel. Es lassen sich durch Wiederverwendung und *Caching* zwar einige Interpolationen einsparen, aber man bleibt bei deutlich über 125 Hexaederinterpolationen pro Voxel. Dieser Aufwand ist so immens, dass er bereits bei den recht kleinen curvilinearen Volumen kaum vertretbar erscheint.

4.3.2.2 Adaptive Filterkernel Anpassung

Eine schnellere Methode als die Interpolation stellt die Möglichkeit dar, den Filterkernel direkt zu verbiegen. Hat man den Filterkernel in Arraykoordinaten gegeben, so ist dieser in Weltkoordinaten betrachtet verbogen.

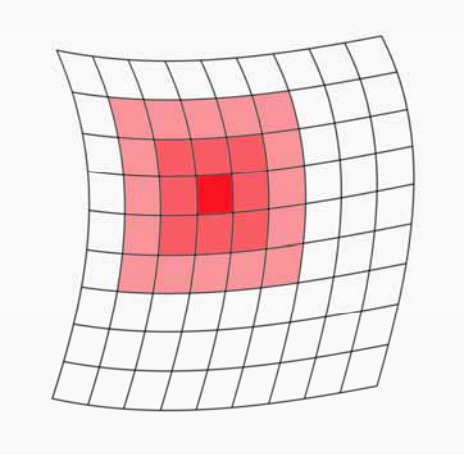

Abbildung 161: Filterkernel in Arraykoordinaten (Images\Cernik\VolGraph3.png)

Abbildung 161 zeigt diesen Aspekt noch einmal. Der rot markierte Filterkernel, hier in einem 2D curvilinearen Slice, passt sich an die Arraykoordinaten an und ist da

durch verbogen. Die Idee ist nun, den Filterkernel in Weltkoordinaten wieder zurecht "zu biegen", d. h. wieder eine symmetrische Gaußkugel zu erzeugen.

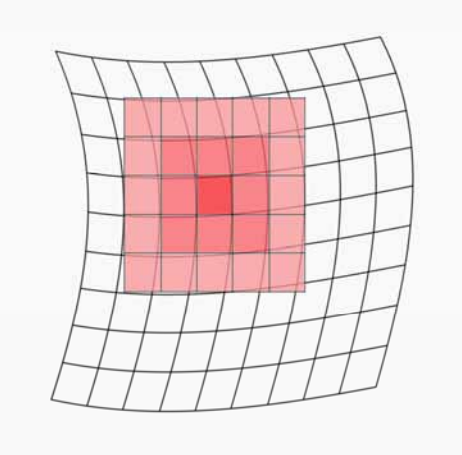

Abbildung 162: Angepasster Filterkernel (Images\Cernik\VolGraph4.png)

In **Abbildung 162** sieht man den korrekten Kernel wie er in Arraykoordinaten aussehen sollte. Man muss also diesen Kernel, der in Weltkoordinaten symmetrisch ist (in Arraykoordinaten muss dieser nicht symmetrisch sein), konstruieren. Man sieht hier, dass ein derartiger Kernel für jeden Voxel neu berechnet werden muss. Eine globale Berechnung des Filterkernels ist nicht mehr möglich.

Verglichen jedoch mit den theoretisch anfallenden Hexaeder Interpolationen der Methode in 4.3.2.1, Seite 172 ist die Filterkernel Neuberechnung deutlich schneller.

$$R_i(x,y,z) = \exp(-\frac{x^2+y^2+z^2}{(\alpha s^i)^2}) \tag{58}$$

Der Filterkernel wurde für kartesische Gitter über Formel (58) berechnet. Um den Kernel nun symmetrisch in Weltkoordinaten zu bekommen, benötigt man den Mittelpunkt des Filterkernels in Weltkoordinaten, den wir hier nun CenterPoint nennen. Die einzelnen Komponenten des Kernelmittelpunktes in Weltkoordinaten bezeichnen wir als x_{center}, y_{center}, z_{center}. Dieser Mittelpunkt lässt sich direkt aus der *GRID* Datei bestimmen, je nachdem um welchen Voxel sich der Kernel befindet.

Des Weiteren müssen für die Kernelparameter x, y, z entsprechende Werte in Weltkoordinaten ermittelt werden, die ab jetzt als x_{world}, y_{world}, z_{world} bezeichnet werden. Auch diese Werte ergeben sich direkt aus den *GRID* Dateien für curvilineare Volumen. Sollte sich ein entsprechender Wert nicht in den *GRID* Dateien vorfinden, weil er sich außerhalb des curvilinearen Volumens befindet, so kann man an dieser Stelle die entsprechenden Werte ignorieren und clippen. Sie fließen ohnehin nicht in die eigentliche Faltung mit ein.

$$R_i(x,y,z) = \exp(-\frac{(x_{world}-x_{center})^2+(y_{world}-y_{center})^2+(z_{world}-z_{center})^2}{(\alpha s^i)^2}) \tag{59}$$

Formel (59) gibt die neue, angepasste Berechnung des Filterkernels an. Diese Berechnung muss nun anders als bei der konventionellen Reinhard Methode für jeden Voxel ausgeführt werden. Der Filter kann also nach der Faltung mit einem Voxel verworfen werden, um für den nächsten Voxel jeweils neu berechnet zu werden. Der Kernel muss zusätzlich nach der Berechnung jeweils noch normiert werden, so dass alle Kernelelemente aufsummiert 1 ergeben.

Der restliche Teil des Algorithmus geschieht fast gleich wie beim konventionellen Reinhard Algorithmus. Jedoch sind hier noch kleinere Anpassungen vorzunehmen, um ein besseres Bild zu erhalten und die Parameter aus dem Reinhard Algorithmus verwenden zu können.

4.3.3 Erweiterte Anpassungen der Reinhard Methode

Da curvilineare Volumen nicht nur gebogen sind, sondern eventuell in Weltkoordinaten gemessen der Abstand zwischen zwei Gitterpunkten sehr viel kleiner sein kann als der Abstand der Gitterpunkte in kartesischen Gittern, kann dies dazu führen, dass der Parameter für die Größe des Filterkernels vielleicht zu klein ist. D. h., dadurch dass das Volumen sehr viel kleiner ist als ein entsprechendes nicht curvilineares Volumen, erhält man dieselbe Stärke des *Hdrw* Effektes erst, wenn die Größe des Filterkernels als Parameter etwas erhöht wird.

Da es für den Benutzer in der Regel nicht wünschenswert ist, die Parameter an curvilineare Volumen ganz anders als gewohnt anzupassen, kann diese Anpassung auch automatisch geschehen (und wird im Programm so durchgeführt).

Um eine dynamische Anpassung der Größe des Filterkernels an die aktuellen Verhältnisse zu ermöglichen, muss als erstes eine Standard Einheit für das curvilineare Volumen ermittelt werden, die wir hier *Average Unit* nennen. Die *Average Unit* gibt die durchschnittliche Distanz zwischen zwei Gitterpunkten im curvilinearen Volumen an.

$$Average\ Unit = \exp\left(\frac{LogSumDistances}{\#Gridpoints \cdot 3}\right) \tag{60}$$

Formel (60) beschreibt die Berechnung für die *Average Unit*. Es handelt sich hier um eine logarithmische Mittelung, die gleichmäßigere Werte produziert als das arithmetische Mittel. $\#Gridpoints \cdot 3$ gibt dabei die Anzahl der Gitterpunkte mal drei an, was der Anzahl aller Summierungen entspricht. Die Summierungen *LogSumDistances* ergeben sich, in dem für jeden Voxel die drei möglichen Distanzen zu seinen Nachbarvoxeln aufsummiert werden (es werden nur die Nachbarn betrachtet, die einen größeren Index im Array aufweisen, jeweils in x-, y- und z-Richtung). Es wird dabei jeweils nur der Logarithmus der Distanz aufsummiert. Die *Average Unit* im curvilinearen Volumen gibt sozusagen das Einheitsmaß an. Damit kann man diese Distanz mit dem Gitterabstand in einem entsprechenden kartesischen Gitter vergleichen.

Nach der Berechnung der *Average Unit* kann nun die korrekte Größe für den Kernel ausgewählt werden. Man kann hierfür z. B. 3 Filtergrößen vorgeben. Der Parameter für die Filtergröße nennt sich im Programm **Kernel Delta**. Die Länge des Kernels in eine Richtung ergibt sich jeweils als $\text{Kernel Delta} \cdot 2+1$. Gegeben ist die vom Benutzer angegebene Auswahl des **Kernel Delta**. Getestet wird nun, ob sich eventuell $\text{Kernel Delta} \cdot 2$ oder $\text{Kernel Delta} \cdot 3$ besser für die Darstellung eignen würde. Natürlich könnte man auch weitere Tests mit $\text{Kernel Delta} \cdot 4$ (etc.) ausführen, dies geht jedoch auf die Rechenzeit.

Der Test läuft nun so ab, dass mit dem kleinsten Kernel Delta begonnen wird und die Distanz vom $CenterPoint$ zu allen 8 Eckpunkten des Kernels in Weltkoordinaten berechnet und aufsummiert wird. Der $CenterPoint$ gibt dabei den Mittelpunkt des Filterkernels in Weltkoordinaten an. Dieses Ergebnis wird dann durch 8 geteilt, um die durchschnittliche Distanz vom $CenterPoint$ zu einem Eckpunkt zu bekommen. Um dies direkt mit der *Average Unit* vergleichen zu können, muss diese noch angepasst werden, um die normale Distanz vom Mittelpunkt zu einem Eckpunkt zu erhalten. Diese Normdistanz nennen wir *AverageNormalDistance*. Sie

entspricht dem Normradius für die aktuelle Scale, muss also für jede Scale neu berechnet werden.

$$AverageNormalDistance = \alpha \cdot s^{Scale} \cdot AverageUnit \tag{61}$$

Aus dem **Center-Surround Ratio** wird dabei direkt die Größe der normalen Scale berechnet, diese wird dann mit der *Average Unit* skaliert.

Um dann zu testen, ob das aktuelle Kernel Delta angemessen ist, wird verglichen, ob die berechnete mittlere Distanz vom CenterPoint zu den Eckpunkten größer ist als die *AverageNormalDistance*. In diesem Fall ist die gewählte Größe des Kernels groß genug, um ein optimales visuelles Ergebnis zu gewährleisten. Sollte die mittlere Distanz kleiner als die *AverageNormalDistance* sein, ist der Filterkernel zu klein gewählt, um ein ähnliches visuelles Ergebnis zu erzielen, wie die eingestellte Größe des Kernels bei nicht curvilinearen Gittern erzielen würde. In diesem Fall wird dann der nächst größere Kernel dem gleichen Test unterzogen. Stellen sich alle Filterkernel als zu klein heraus, wird der größte getestete Filterkernel verwendet.

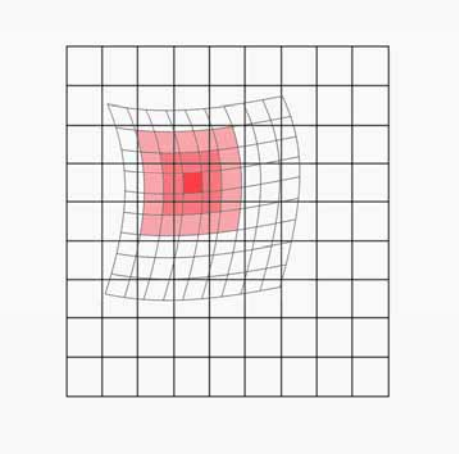

Abbildung 163: Curvilineares Gitter in Weltkoordinaten (Images\Cernik\VolGraph5.png)

Abbildung 163 zeigt die Situation noch einmal anschaulich. Das kartesische, äußere Gitter stellt die Weltkoordinaten dar. Eingebettet ist ein feineres, curvilineares Gitter. Man sieht, dass der Filterkernel, der im curvilinearen Gitter ein **Kernel Delta** von 2 und damit eine Länge von 5 Voxeln in Weltkoordinaten hat, nur etwa 3 Voxel Kantenlänge umfasst, also einem Kernel entspricht mit einem **Kernel Delta** von 1.

Bei dem Hdrw Algorithmus würde ein Benutzer also in diesem Falle als Einstellung im Programm ein **Kernel Delta** von 1 angeben, so würde das Programm durch die Länge ein optimales Kernel Delta von 2 ermitteln, was dem Ergebnis eines **Kernel Delta**s von 1 in Weltkoordinaten entspricht.

Wichtig ist, dass die Suche nach dem optimalen **Kernel Delta** für jeden einzelnen Voxel ausgeführt werden muss, da es durchaus vorkommen, dass das curvilineare Volumen an verschiedenen Stellen ganz andere Abstände zwischen den einzelnen Gitterpunkten hat. So kann ein Bereich des Volumens sehr weitläufig mit großen Abständen und ein anderer Bereich im selben Volumen sehr dicht gepackt mit Gitterpunkten sein.

4.3.4 Vergleich der Methoden

In diesem Kapitel sollen die visuellen Ergebnisse des konventionellen Reinhard Algorithmus direkt mit dem angepassten Algorithmus für curvilineare Volumen verglichen werden.

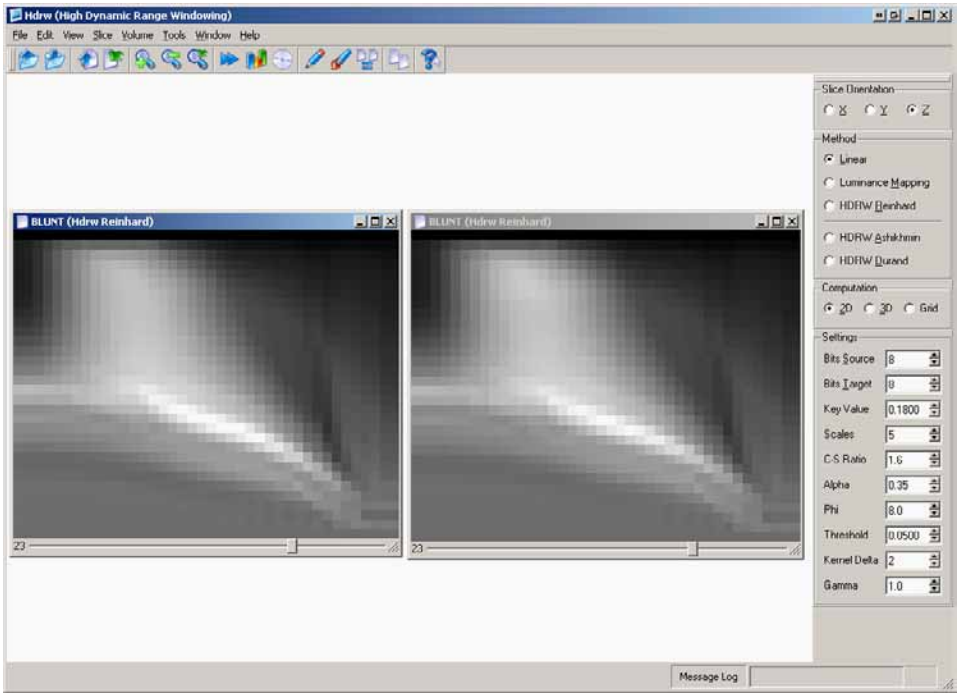

Abbildung 164: Vergleich regulär zu curvilinearem Algorithmus (Images\Cernik\BluntCompare3.png)

In **Abbildung 164** sieht man das curvilineare Volumen *Blunt*, das auch in den vorangegangenen Beispielen verwendet wurde. Als Settings wurden die Standardeinstellungen des Programms verwendet. Das linke Bild wurde durch den konventionellen Reinhard Hdrw Algorithmus erstellt. Das gebogene Gitter des curvilinearen Volumens wurde dafür nicht beachtet, sondern das Gitter wird als kartesisches Gitter betrachtet. Das rechte Bild wurde mit dem neuen Hdrw Algorithmus für curvilineare Volumen erstellt. Es wurde bei beiden Verfahren jeweils die Schicht 23 des Volumens herangezogen und verglichen.

Die Methode, um dies zu berechnen, ist dabei nicht die Hexaeder Interpolation, sondern die adaptive Filterkernel Anpassung. Auf den ersten Blick wird man kaum Unterschiede sehen, was auch nicht weiter verwunderlich ist, da die Unterschiede nur dort vorhanden sind, wo sich das Gitter des curvilinearen Volumens deutlich vom kartesischen Gitter unterscheidet.

Am Differenzbild in **Abbildung 165** sieht man deutlich, dass es durchaus Unterschiede zwischen den beiden Bildern gibt. Vor allem die Nahtstelle in der Mitte, in welcher das curvilineare Volumen die größte Biegung aufweist, zeigt auch den größten Unterschied zum konventionellen Hdrw Algorithmus.

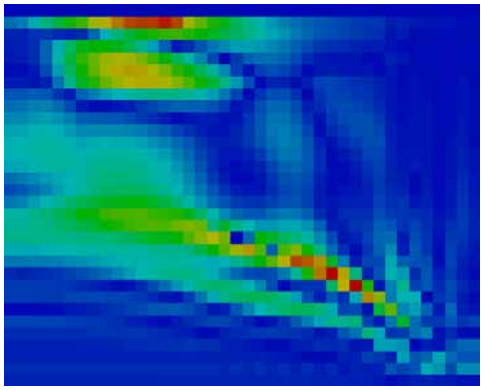

Abbildung 165: Differenzbild der Hdrw Methoden (Images\Cernik\BluntCompare4.png)

Als weiteres Beispiel wird ein Vergleich der beiden Verfahren beim *post.vector* Volumen gezeigt.

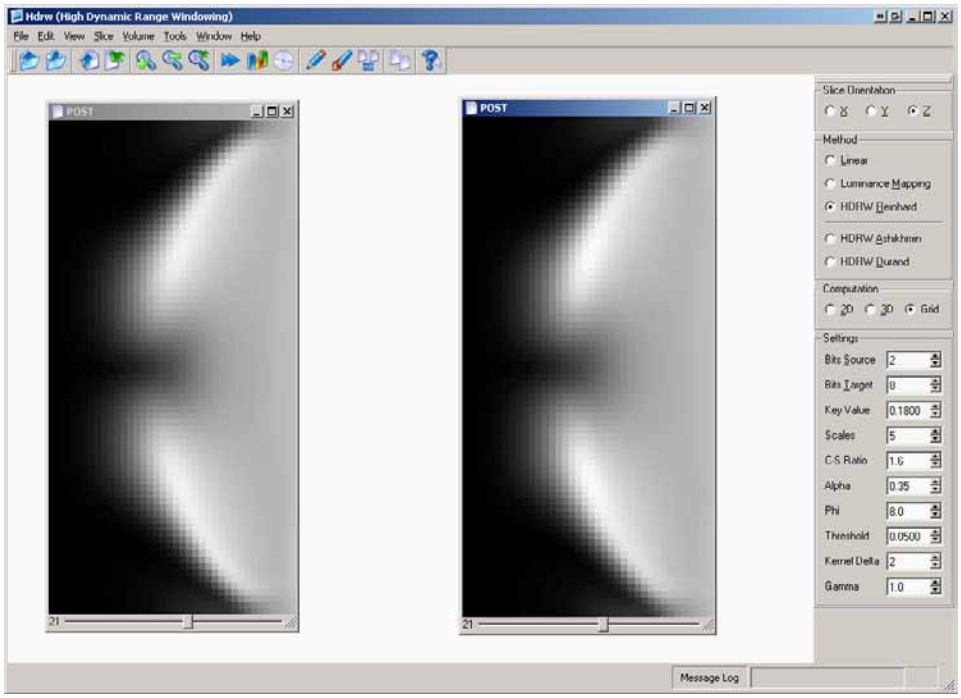

Abbildung 166: Vergleich regulär zu curvilinearem Algorithmus (Images\Cernik\PostCompare1.png)

In **Abbildung 166** sieht man das *Post* Volumen. Das linke Bild gibt dabei Schicht 21 wieder, nachdem der konventionelle Reinhard Hdrw Algorithmus das Volumen bearbeitet hat. Als Settings wurden die selben Einstellungen genommen wie beim obigen Beispiel. Dies entspricht den Standardeinstellungen. Das rechte Bild zeigt den Hdrw Algorithmus mit der adaptiven Filterkernel Anpassung. Auf den ersten Blick sind die Unterschiede auch hier eher schwer zu sehen.

Im Differenzbild des *Post* Volumens, das in **Abbildung 167** dargestellt ist, kann man nun deutlich die Unterschiede erkennen. In den roten Bereichen ist die Abweichung am größten. Auch hier findet man die größte Abweichung in Bereichen mit der höchsten Biegung. Dort unterscheidet sich der neue Hdrw Algorithmus für curvilineare Gitter am deutlichsten von dem konventionellen Hdrw Algorithmus auf kartesischen Gittern.

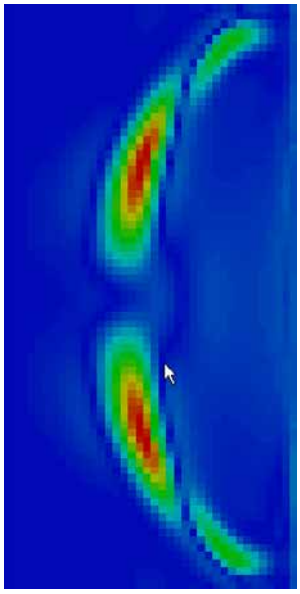

Abbildung 167: Differenzbild der Hdrw Methoden (Images\Cernik\PostCompare2.png)

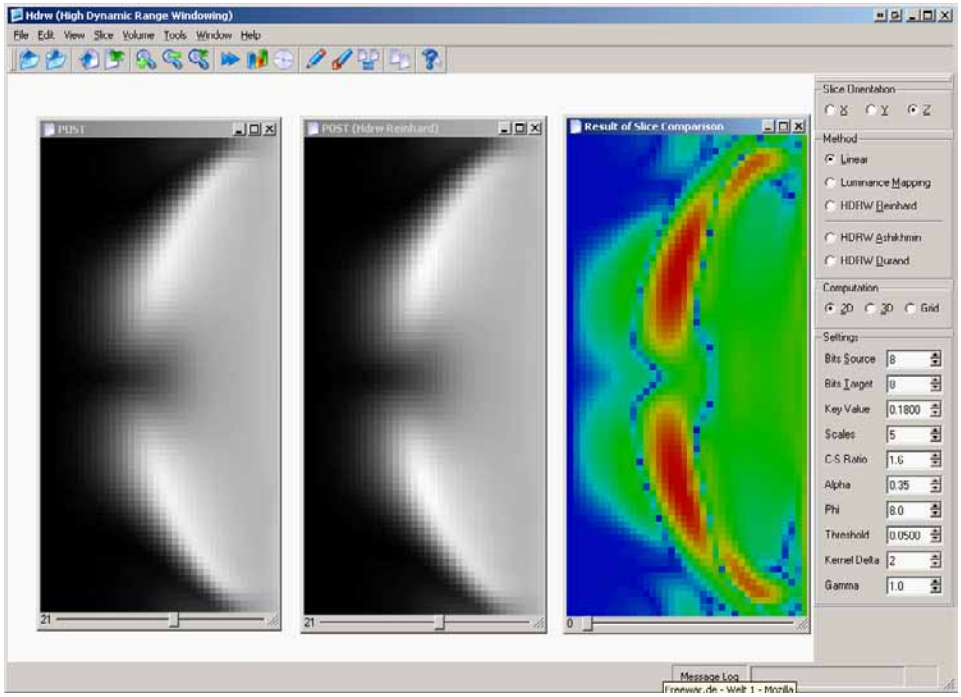

Abbildung 168: Originale und Differenzbild (Images\Cernik\PostCompare3.png)

In **Abbildung 168** sieht man noch einmal die beiden Volumen, einmal links mit dem konventionellen Hdrw Algorithmus berechnet und in der Mitte mit dem Algorithmus, der die Gitter Information mit berücksichtigt. Das Differenzbild rechts ist hier nun logarithmisch skaliert. Damit sieht man auch, dass in den Bereichen, in welchen im normalen relativen Differenzbild kein Unterschied zu sehen waren, auch noch Differenzen vorkommen. Nur an den wenigsten Stellen sind die Bilder wirklich identisch.

4.4 Implementierung

Im Folgenden wird auf die Implementierung des Algorithmus im Source Code eingegangen. Der Hauptteil des entsprechenden Code im Programm findet sich in den Dateien *HdrwVolumeSCALAR.cpp*, *HdrwVolumeVECTOR.cpp* und *HdrwMethodReinhardGrid.cpp*. Definitionen von Variablen und Volumenarrays finden sich in den entsprechenden Header Dateien.

Es wird hier nur Pseudo Code verwendet, um die Umsetzung grob zu skizzieren und klar zu machen; Genauere Implementierungsdetails finden sich direkt im Source Code. Optimierungen werden kurz erwähnt, um die Vorgehensweise zu veranschaulichen.

4.4.1 Laden der Dateien

Das Laden der curvilinearen Dateien im *SCALAR* Format findet in der Datei *HdrwVolumeSCALAR.cpp* statt. Das Laden von *VECTOR* Dateien dagegen geschieht in der Datei *HdrwVolumeVECTOR.cpp*. Die grobe Vorgehensweise beim Laden einer Datei wird anhand der Laderoutine für *SCALAR* Dateien erklärt. Es muss des Weiteren darauf geachtet werden, dass Binärdaten in der *MSB* Darstellung vorliegen. Dies wird im Source Code beachtet, hier im Pseudo Code jedoch nicht weiter veranschaulicht.

```
VolumeLoadScalar()
{
   OpenScalarFile();
   //Header des Scalar Files auslesen
   ReadHeaderCookie();
   ReadHeaderVersion();
   ReadHeaderNumberOfItems();
   ReadXDimension();  //Größe des Volumens einlesen
   ReadYDimension();
   ReadZDimension();

   AllocateMemoryforVolume(); //Speicherplatz reservieren
   CopyVolumeIntoMemory(); //Volume einlesen und in den Speicher
                           kopieren
   CloseScalarFile();
   LoadGridFile();
}
```

An obigem Pseudo Code kann man die prinzipielle Vorgehensweise für das Öffnen von *SCALAR* Dateien sehen. Wichtig ist jedoch nun noch die Funktion **`LoadGridFile()`** (Welche im Programm selbst den Namen **`YVolumeLoadGrid()`** trägt).

```
LoadGridFile()
{
   //Zuerst wird nach einem Grid file gesucht, wird dies nicht gefunden
   //wird ein entsprechendes Mesh File eingelesen

   If(GridFileExists)
      OpenGridFile();
   else
      OpenMeshFile();
```

```
   ReadHeaderOfFile();
   AllocateMemoryforGrid(); //Hier wird 3 mal soviel Speicher wie für
                            //das Skalar File benötigt, da jeder Grid
                            //Punkt aus 3 Komponenten (x,y,z) besteht
   CopyGridIntoMemory();
   CalculateAverageUnit();  //Berechne die mittlere Distanz im Volumen
                            //zwischen zwei Gitterpunkten
}
```

Wie man sieht, wird nach dem Einlesen des Gitters die bereits im letzten Kapitel besprochene **AverageUnit** berechnet. Diese gibt die mittlere logarithmische Distanz zwischen zwei Gitterpunkten an.

```
CalculateAverageUnit()
{
   double Distance=0;
   double LogSum=0; //Gibt die logarithmische Summe an
   for(z=0; z<(SizeZ-1); z++)
      for(y=0; y<(SizeY-1); y++)
         for(x=0; x<(SizeX-1) x++)
         {
            //Distanz zum nächsten Nachbarn in X Richtung
            Distance = GetDistanceToXNeighbor(x,y,z);
            LogSum += log(Distance);

            //Distanz zum nächsten Nachbarn in X Richtung
            Distance = GetDistanceToYNeighbor(x,y,z);
            LogSum += log(Distance);

            //Distanz zum nächsten Nachbarn in X Richtung
            Distance = GetDistanceToZNeighbor(x,y,z);
            LogSum += log(Distance);
         }
   AverageUnit = exp(LogSum/((SizeZ-1)*(SizeY-1)*(SizeX-1));
}
```

Nach Berechnung der **AverageUnit** ist die Vorberechnung fertig, die beim Laden durchgeführt wird und das Volumen kann visualisiert werden. Da sich bei dem Hdrw Algorithmus auf curvilinearen Gittern nur die Faltung anders darstellt, wird nur diese hier im Pseudo Code erklärt. Der Rest des Algorithmus findet sich bei der Beschreibung des konventionellen Reinhard Verfahren.

4.4.2 Die Faltung

In der Faltung wird nun die konkrete Faltung am Voxel (x,y,z) betrachtet, mit der Scale i. Der folgende Code ist nur Pseudo Code und gibt den groben schematischen Aufbau des eigentlichen Source Code wieder. Der hier angegebene Pseudo Code ist nicht compilierbar und es fehlen einige Sachen, die im richtigen Source Code vorhanden sind, wie Fehlerabfragen, Bereichsüberprüfungen und Anpassungen.
Im richtigen Source Code befindet sich der Hdrw Algorithmus für curvilineare Gitter in der Datei *HdrwMethodReinhardGrid.cpp*.

```
ConvolveGrid(int x, int y, int z, int i)
{
   double AverageNormalDistance = Alpha*pow(s, i)*AverageUnit;
   CenterPoint = GetCenterPointKoords(x,y,z);

   //Als nächstes werden die möglichen KernelDeltas berechnet
   //In unserem Fall wurden 3 Kernel zur Anpassung bereit gehalten
   int KernelDeltas[3];
   KernelDeltas[0] = KernelDelta;
   KernelDeltas[1] = KernelDelta*2;
   KernelDeltas[2] = KernelDelta*3;
   int KernelDeltaNumber = 2; //Ausgewählter Kernel, Standard ist der
                              //Kernel mit dem größten KernelDelta
   double distance;  //Gibt die mittlere Distanz Mittelpunkt zur Ecke
                     //des Kernels an

   //Nun wird berechnet welche der KernelDeltas angemessen wäre
   for(count=0; count<3; count++)
   {
      //Distanz vom CenterPoint zu allen 8 Ecken aufsummieren
      //Dabei in der Funktion CalcDistanceCorner evt. Clipping
      //an Arraygrenzen beachten
      distance = 0;
      distance += CalcDistanceToCorner1();
      distance += CalcDistanceToCorner2();
      distance += CalcDistanceToCorner3();
      distance += CalcDistanceToCorner4();
      distance += CalcDistanceToCorner5();
      distance += CalcDistanceToCorner6();
      distance += CalcDistanceToCorner7();
      distance += CalcDistanceToCorner8();

      double divisor = 1.0/8.0; //Aus Speedgründen Multiplikation
                                //statt Division wählen
      distance *= divisor; //Mittelwert bilden
      if(distance>AverageNormalDistance) //passenden Kernel gefunden
      {
         KernelDeltaNumber = count; //KernelDelta auswählen
         break;
      }
   }

   //Nachdem das passende KernelDelta ausgewählt wurde wird nun die
   //eigentliche Faltung durchgeführt.
   double ExpFactor = -1.0/pow(Alpha*pow(s, i)*AverageUnit, 2);
   double Sum = 0; //Wird später für die Normalisierung verwendet

   //Nun die Seitenlänge des Kernels anhand des KernelDelta berechnen
   int Kernelsize = KernelDeltas[KernelDeltaNumber]*2+1;
   double ConvolutionResult=0;

   //Um die Faltung deutlich zu beschleunigen wird der Bereich unter
   //dem Filterkernel direkt in lokale Caching-Arrays kopiert.
   //Hierbei muss entsprechendes Clipping berücksichtigt werden.
   int KernelstartX, KernelstartY; //X und Y wo der Kernel beginnt
   int ArrayOffsetX, ArrayOffset Y; //Offset wo Daten im Array beginnen
   int KernelWidth, KernelHeight; //Echte Länge des Kernels im Volumen,
                                  //in jeweils einer Z-Schicht. Hier
                                  //wird das entsprechende Clipping mit
                                  //berücksichtigt.

   KernelStartX = MAX((0-KernelDeltas[KernelDeltaNumber])+x, 0);
   KernelStartY = MAX((0-KernelDeltas[KernelDeltaNumber])+y, 0);
   ArrayOffsetX = -MIN((0-KernelDeltas[KernelDeltaNumber])+x, 0);
   ArrayOffsetY = -MIN((0-KernelDeltas[KernelDeltaNumber])+y, 0);
   KernelWidth = KernelSize-ArrayOffsetX-(MAX(0, (0-
      KernelDeltas[KernelDeltaNumber])+x+Kernelsize-SizeX-1;
   KernelHeight = KernelSize-ArrayOffsetY-(MAX(0, (0-
     KernelDeltas[KernelDeltaNumber])+y+Kernelsize-SizeY-1;

   //Die entsprechenden Arrays die reserviert werden müssen
   double KernelArea[SQUARE(Kernelsize)];
   Point KernelAreaPoint[SQUARE(Kernelsize)];
```

```
   for(FilterZ=0; FilterZ<Kernelsize; FilterZ++)
   {
      //Die aktuellen Griddaten der aktuellen Z-Schicht werden in das
      //Array KernelAreaPoint kopiert
      KernelAreaPoint = GetGridPointsArray(Kernelsize, KernelStartX,
         KernelStartY, ArrayOffsetX, ArrayOffsetY, KernelWidth,
         KernelHeight);

      //Die Scalardaten unter dem Filter kommen in das Array KernelArea
      KernelArea = GetScalarValuesArray(Kernelsize, KernelStartX,
         KernelStartY, ArrayOffsetX, ArrayOffsetY, KernelWidth,
         KernelHeight);

      for(FilterY=0; FilterY<Kernelsize; FilterY++)
      {
         for(FilterX=0; FilzerX<Kernelsize; FilterX++)
         {
            KernelPoint = GetPointKoords(x,y,z); //Weltkoordinaten für den
                                                 //Arraypunkt x,y,z

            //Berechne Filter an Stelle x,y,z, speichere Wert in Value
            Value = exp(ExpFactor*(SQUARE(KernelPoint.x-CenterPoint.x)+
                                   SQUARE(KernelPoint.y-CenterPoint.y)+
                                   SQUARE(KernelPoint.z-CenterPoint.z)));
            Sum += Value; //Alle Filterwerte aufsummieren für spätere
                          //Filter Normalisierung
            ConvolutionResult += KernelArea[FilterX][FilterY]*Value;
         }
      }
   }
   //Ergebnispunkt normalisieren und mit dem Faktor a/Lw skalieren
   ConvolutionResult = ConvolutionResult/Sum*aLw;
}
```

Im Source Code selbst wurde das Programm möglichst auf Geschwindigkeit optimiert. Die grundlegenden Ansätze dafür sieht man bereits im Pseudo Code von oben, wie z. B. das lokale Caching des Bereichs unterhalb des Filterkernel.

4.4.3 Speichern der Dateien

Der grobe Vorgang beim Speichern von curvilinearen Daten wird hier kurz anhand von Pseudo Code erklärt. Im Source Code des Programms findet man die Speicherroutine in *HdrwVolumeSCALAR.cpp* und *HdrwVolumeVECTOR.cpp*.

Im Folgenden wird die Speicherung von skalaren curvilinearen Daten betrachtet. Hierbei ist es irrelevant welche Art von Volumendatei geladen wurde. Liegt zum Beispiel keine *GRID* Datei vor, sondern wurde ein medizinisches Volumen im *SLC* Format geladen, kann jenes dennoch als *SCALAR* Datei gespeichert werden. Eine *GRID* Datei wird dann automatisch berechnet und auch abgespeichert, dabei wird die *GRID* Datei über das korrekte Spacing der Originaldatei berechnet.

Beim Speichern muss des Weiteren darauf geachtet werden, dass die binären Float Werte in der *MSB* Darstellung gespeichert werden. Dies geschieht im Source Code, wird im Pseudo Code jedoch nicht dargestellt.

```
VolumeSaveScalar()
{
   OpenScalarFileToSave();
   WriteHeaderCookie();
   WriteHeaderVersion();
   WriteNumberOfItems();
   WriteXDimension();
   WriteYDimension();
   WriteZDimension();
```

```
   WriteScalarDataFromArray(); //Hier werden die Scalardaten in die
                               //SCALAR Datei geschrieben.
   VolumeSaveGrid(); //Speicherung der GRID Datei
}
```

Nachdem die *SCALAR* Datei als solche gespeichert wurde, wird im nächsten Schritt die korrespondierende *GRID* Datei erzeugt, die denselben Namen hat, jedoch als Dateiendung *.grid* aufweist.

```
VolumeSaveGrid()
{
   GridFileName = SetExtensionToGrid();
   OpenGridFile(GridFileName);
   WriteHeaderCookie();
   WriteHeaderVersion();
   WriteNumberOfItems();
   WriteXDimension();
   WriteYDimension();
   WriteZDimension();

   if(GridInformationAvailable) //Falls Informationen zum Grid
                               //vorliegen
   {
      SaveXComponentArray(); //X-Komponente der Gitterpunkte speichern
      SaveXComponentArray(); //Y-Komponente der Gitterpunkte speichern
      SaveXComponentArray(); //Z-Komponente der Gitterpunkte speichern
   }
   else //Es liegen keine Gitterinformationen vor
   {
      //X-Komponente der Gitterpunkte berechnen und abspeichern
      for(z=0; z<YSizeZ; z++)
         for(y=0; y<YSizeY; y++)
            for(x=0; x<YSizeX; x++)
               WriteFloat(x*YSpacingX);

      //Y-Komponente der Gitterpunkte berechnen und abspeichern
      for(z=0; z<YSizeZ; z++)
         for(y=0; y<YSizeY; y++)
            for(x=0; x<YSizeX; x++)
               WriteFloat(y*YSpacingY);

      //Z-Komponente der Gitterpunkte berechnen und abspeichern
      for(z=0; z<YSizeZ; z++)
         for(y=0; y<YSizeY; y++)
            for(x=0; x<YSizeX; x++)
               WriteFloat(z*YSpacingZ);
   }
}
```

4.5 Fazit

Beim Windowing gibt es viele Möglichkeiten und Methoden dies effizient und visuell ansprechend zu verwirklichen. Immer interessanter ist es auch, den Bereich der Hdrw Methoden, die ursprünglich für die Fensterung entworfen wurden, für allgemeine Bild- und Volumenverbesserung einzusetzen.

Die Reinhard Methode hat sich dabei als besonders robust herausgestellt und kann auch auf 3D Daten schnell und problemlos angewandt werden. Um diese jedoch auch auf curvilinearen Daten korrekt durchzuführen, sind entsprechende Anpassungen erforderlich. Wir haben uns für die adaptive Filterkernel Berechnung entschieden, um den HDRW Algorithmus in möglichst kurzer Rechenzeit durchzuführen.

Besonders in Regionen, in denen sich das Gitter der curvilinearen Dateien deutlich von einem kartesischen Gitter unterscheidet, sind Unterschiede wahrnehmbar und der Algorithmus liefert eine bessere visuelle Darstellung. Die Ergebnisse sind dennoch sehr ähnlich zum konventionellen Reinhard Hdrw Algorithmus, was auch nicht weiter verwundert, da es sich lediglich um eine angepasste Version dieses Algorithmus handelt.

Beachten sollte man auch, dass im Hdrw Programm keine echte 3D-Darstellung der curvilinearen Volumen möglich ist und das korrekte Ergebnis sinnvoller Weise abgespeichert und in einem entsprechenden Viewer betrachtet werden sollte.

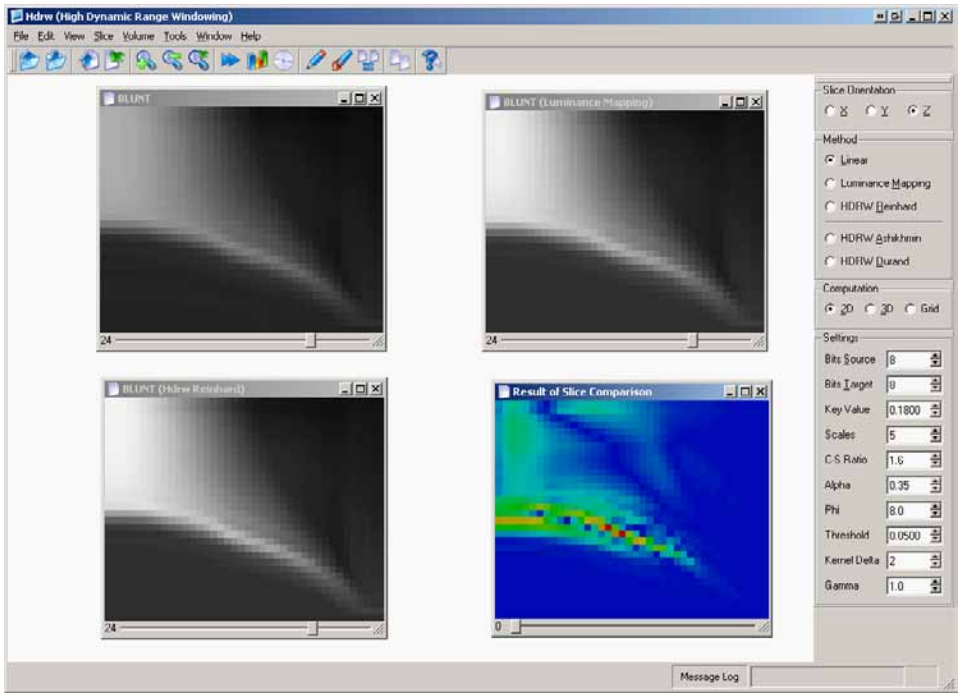

Abbildung 169: Verschiedene Methoden (Images\Cernik\BluntCompare5.png)

Abbildung 169 zeigt noch einmal die Methoden von oben links nach unten rechts: **Linear**es Mapping, **Luminance Mapping**, angepasstes **Hdrw Reinhard** Verfahren in 3D auf curvilinearen Gittern und rechts unten ein Differenzbild zwischen dem Luminance Mapping und dem Reinhard Verfahren.

Ludwig Gauckler

5 Die Ashikhmin und Durand Methoden

5.1 Unterkapitel

Text

6 Quellen

- ANNEMIEK, B., FAESEN, A., SANDERS, E., SCHOONENBERG, G., 2004. *Thickness mapping on surfaces* [online]. Available from:
  http://www.bmi2.bmt.tue.nl/image-analysis/people/hbouma/education/ogo/OGO02_thickness.pdf
  [Accessed 25 November 2004]

- ENCARNAÇÃO, J., STRAßER, W., KLEIN, R., 1997. *Graphische Datenverarbeitung 2*. München: Oldenburg.

- HENNESSEY, J., PATTERSON, D., 2003. *Computer Architecture*. San Francisco: Morgan Kaufmann.

- HUANG, R., 2003. *Performance Enhancement of Convolution* [online]. Available from:
  http://www.cs.ucdavis.edu/~bai/ECS231/finalhuang.pdf
  [Accessed 25 November 2004]

- MITCHELL, J., 2003. *Real-Time 3D Scene Post-Processing* [online]. Available from:
  http://www.ati.com/developer/gdc/GDC2003_ScenePostprocessing.pdf
  [Accessed 28 November 2004]

- NASA, 2003. Research & Technology, Sample Datasets [online]. Available from:
  http://marsoweb.arc.nasa.gov/Research/Datasets/Hung/index.shtml
  [Accessed 25 November 2004]

- PEIKERT, R., 2004. *When is a point contained in a hexahedral cell?* [online] . Available from:
  http://www.cg.inf.ethz.ch/~peikert/personal/HexCellTest/
  [Accessed 28 November 2004]

- REINHARD, E., MICHAEL, S., PETER, S., JAMES F., 2002. Photographic Tone Reproduction for Digital Images. *In: ACM Transactions on Graphics (TOG), Volume 21, Issue 3, Special issue: Proceedings of ACM SIGGRAPH 2002, Session: Images and video table of contents, July 2002 San Antonio, Texas, USA*. New York, NY, USA: ACM Press, 267-276.

- ROWLETT, R., 2000. *How Many? A Dictionary of Units of Measurement* [online]. Available from:
  http://www.unc.edu/~rowlett/units/dictH.html
  [Accessed 25 November 2004]

- SCHUMANN, H., MÜLLER, W., 2000. *Visualisierung, Grundlagen und allgemein Methoden*. Heidelberg: Springer.

7 Index

- Z -