

# Kompressionsverfahren für dreidimensionale Feldlinien

Bachelorthesis von  
SCHWAMMBERGER, Jonas

Betreuer:  
Csillaghý, André  
Felix, Simon

FHNW  
Hochschule für Technik  
Studiengang Informatik

Windisch, 8. Februar 2015

## Abstract

Ziel dieser Arbeit ist es eine verlustbehaftete Kompression für Feldlinien zu entwickeln, welche die Übertragung und das Caching im Arbeitsspeicher ermöglicht. In dieser Arbeit wurden drei Verfahren entwickelt: eine Kompression mit Adaptivem Subsampling, eine Kompression mit einer Diskreten Kosinus Transformation und eine Kompression mit Prädiktoren. Die höchste Kompressionsrate wurde mit der Diskreten Kosinus Transformation erreicht, während die Kompression mit Adaptiven Subsamplings minimale Kompressionsartefakte aufweist. Die Kompression mit Prädiktoren ist ein Kompromiss zwischen Kompressionsrate und Artefakte.

Die Übertragung und Zwischenspeicherung von Feldlinien ist mit den entwickelten Verfahren möglich. Der Limitierende Faktor für die Kompression ist die Artefaktbildung: Ringing oder Ringing-Ähnliche Artefakte sind in Visualisierungen der Feldlinien störend. Wenn die Visualisierung ein Heranzoomen erlaubt, sind auch schwach ausgeprägte Artefakte zu erkennen. Eine höhere Kompressionsrate ist mit Verfahren zu erreichen, welche kaum oder keine Ringing Artefakte produzieren wie Wavelet Transformation, Curve Fitting oder Compressive Sensing.

## Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>1</b>
<b>2 State of the Art</b>	<b>3</b>
2.1 JPEG/JFIF Bildkompression . . . . .	3
2.2 Point Cloud Kompression . . . . .	4
2.3 Curve Fitting . . . . .	4
2.4 Compressive Sensing . . . . .	5
2.5 Entropie Kodierung . . . . .	5
<b>3 Umgesetzte Kompressionsverfahren</b>	<b>6</b>
3.1 Ist-Kompression . . . . .	6
3.2 Kompressionsverfahren: Adaptives Subsampling . . . . .	6
3.2.1 Adaptives Subsampling . . . . .	7
3.2.2 Entropie Kodierung mittels RAR . . . . .	7
3.3 Kompressionsverfahren: Diskrete Kosinus Transformation . . . . .	8
3.3.1 Subsampling . . . . .	8
3.3.2 Ableitung . . . . .	8
3.3.3 Diskrete Kosinus Transformation . . . . .	9
3.3.4 Quantisierung . . . . .	9
3.3.5 Entropie Kodierung . . . . .	9
3.4 Kompressionsverfahren: Prädiktive Kodierung . . . . .	11
3.4.1 Angle Subsampling und Quantisierung . . . . .	11
3.4.2 Rekursiver Linearer Kodierung . . . . .	11
3.4.3 Quantisierung . . . . .	13
3.4.4 Entropie Kodierung . . . . .	13
<b>4 Qualitätsmessung der dekomprimierten Daten</b>	<b>14</b>
4.1 Auswahl und Erhebung der Testdaten . . . . .	14
4.2 Berechnung der Standardabweichung . . . . .	14
4.2.1 Berechnung der Standardabweichung . . . . .	15
4.3 Berechnung der angepassten PSNR-HVS-M . . . . .	15
4.3.1 Contrast Masking . . . . .	16
4.3.2 Umsetzung und Anpassung der PSNR-HVS-M für diese Arbeit . . . . .	17
<b>5 Implementation der Dekompression</b>	<b>18</b>
5.1 Software Architektur . . . . .	18
5.1.1 Vorladen und Caching von Komprimierten und Dekomprimierten Feldlinien . . . . .	18
5.1.2 Asynchrone Aufrufe mittels Executor Services . . . . .	20
<b>6 Variantenstudie</b>	<b>21</b>
6.1 Kompressionsverfahren: Adaptives Subsampling . . . . .	21
6.2 Kompressionsverfahren: Diskrete Kosinus Transformation . . . . .	23
6.2.1 Variante: DCT . . . . .	23
6.2.2 Variante: Ableitung+DCT . . . . .	24
6.2.3 Variante: PCA+Ableitung+DCT . . . . .	25
6.2.4 Variante: Ableitung+DCT+Byte Kodierung . . . . .	26
6.2.5 Variante: Randbehandlung+DCT+Byte Kodierung . . . . .	26
6.2.6 Ringing Artefakte . . . . .	26
6.2.7 Behandlung der Ringing Artefakte . . . . .	28
6.2.8 Abschliessende Variante . . . . .	30

6.3 Kompressionsverfahren: Prädiktive Kodierung . . . . .	32
6.3.1 Variante: einfaches Subsampling . . . . .	32
6.3.2 Variante: Adaptives Subsampling . . . . .	33
6.3.3 Rekursive Lineare Kodierung . . . . .	33
6.3.4 Rekursive Lineare Kodierung mit angepasster Quantisierung . . . . .	36
<b>7 Fazit</b>	<b>38</b>
<b>8 Anhang</b>	<b>44</b>
8.1 Installationsanleitung . . . . .	44
8.2 File Format and Decompression . . . . .	44
8.2.1 Encodings . . . . .	44
8.2.2 Transformation to cartesian Coordinates . . . . .	46
8.2.3 Decode Prediction . . . . .	46
8.3 Performance Tests . . . . .	49
<b>9 Ehrlichkeitserklärung</b>	<b>50</b>

## 1 Einleitung

Moderne Simulationen sind in der Lage grosse Mengen an Daten zu produzieren. Die rohe Datenmenge ist oft zu gross um sie zu archivieren oder in vernünftiger Zeit über eine Internetverbindung zu übertragen. In wissenschaftlichen Simulationen werden natürliche Phänomene wie Schwingungen, Flugbahnen, Kraftfelder etc. als Menge von Kurven im dreidimensionalen Raum abgebildet. Die Kurven sind dargestellt als Folge von Punkten. Ziel dieser Arbeit ist es eine verlustbehaftete Kompression für die Punktfolgen zu entwickeln, welche es ermöglicht die Daten über eine Internetverbindung zu übertragen und lokal zwischenspeichern.

Im Rahmen dieses Projekts werden Daten von Magnetfeldlinien der Sonne komprimiert, welche über eine Internetverbindung zum JHeliovewer übertragen werden. Der JHeliovewer ist eine Applikation zur Visualisierung von Satellitenmessdaten und Simulationen der Sonne. Die Applikation wird von der ESA und der FHNW entwickelt. Die Abbildung 1 zeigt eine Visualisierung des JHeliovewers, welche eine Aufnahme der Sonnenoberfläche mit den simulierten Feldlinien darstellt.

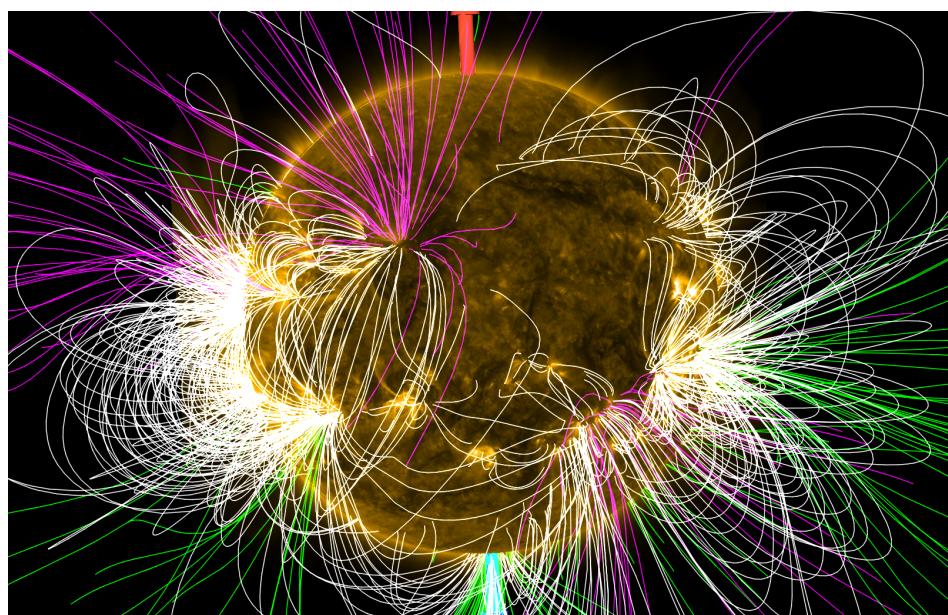


Abbildung 1: Visualisierung der Feldlinien im JHeliovewer

Auf der Visualisierung sind Feldlinien in drei unterschiedlichen Farben zu erkennen, welche drei unterschiedliche Typen darstellen: Linien, die auf der Sonne starten und wieder auf der Sonne landen, auf der Sonne starten und ins Weltall führen oder vom Weltall auf der Sonne landen. Die weißen Feldlinien repräsentieren "Sonne zu Sonne", die Grünen "Sonne ins Weltall" und die Violetten "Weltall zur Sonne".

Der JHeliovewer ist in der Lage eine Abfolge von Mess- und Simulationsdaten als Animation zu visualisieren. Die Daten der Animation werden entweder mit einem Streaming kontinuierlich heruntergeladen oder zuvor im Arbeitsspeicher abgelegt. Im Extremfall werden 1000 komprimierte Simulationen im Arbeitsspeicher zusammen mit anderen Mess- und Simulationsdaten abgelegt. Im Vorfeld wurde bereits eine verlustbehaftete Kompression entwickelt, welche die Datenmenge auf durchschnittlich 1 Megabyte pro Simulation reduziert. Das Caching von 1000 Feldlinien-Simulationen benötigt im Ist-Zustand durchschnittlich ein Gigabyte an Arbeitsspeicher. Das Ziel der Arbeit ist eine verlustbehaftete Kompression zu entwickeln, welche für das Caching der Simulationen um den Faktor 8 bis 10 weniger Arbeitsspeicher benötigt.

Zusätzlich soll erforscht werden unter welchen Bedingungen und Kompromisse das Streaming von Simulationen möglich ist. Für die Animation der Feldlinien werden im Allgemeinen 1 bis maximal 10 Simulationen in der Sekunde benötigt. Ein Fernziel des JHeliovewers ist die kontinuierliche Animation der Feldlinien. Im Ist-Zustand ist jeder Wechsel der Simulation markant. Eine Möglichkeit das Ziel zu erreichen ist die Kadenz

zu erhöhen und 10-30 Simulationen in der Sekunde zu animieren. In Zukunft besteht die Möglichkeit, dass eine höhere Kompressionsrate benötigt wird.

Im Laufe des Projekts wurden drei Kompressionen entwickelt. Die Kompressionsraten sind der Tabelle 1 ersichtlich. Die Visualisierung der komprimierten Feldlinien kann in den Abbildungen 2 verglichen werden. In dieser Arbeit wurde darauf geachtet, dass die Artefakte sich minimal ausprägen und sich die Feldlinien kaum von den Originaldaten unterscheiden.

Lösungsansatz	Kompressionsrate
Ist-Zustand	1
Adaptives Subsampling	11.6
Diskrete Kosinus Transformation	14.1
Prädiktive Kodierung	13.6

Tabelle 1: Kompressionsraten der Lösungsansätze.

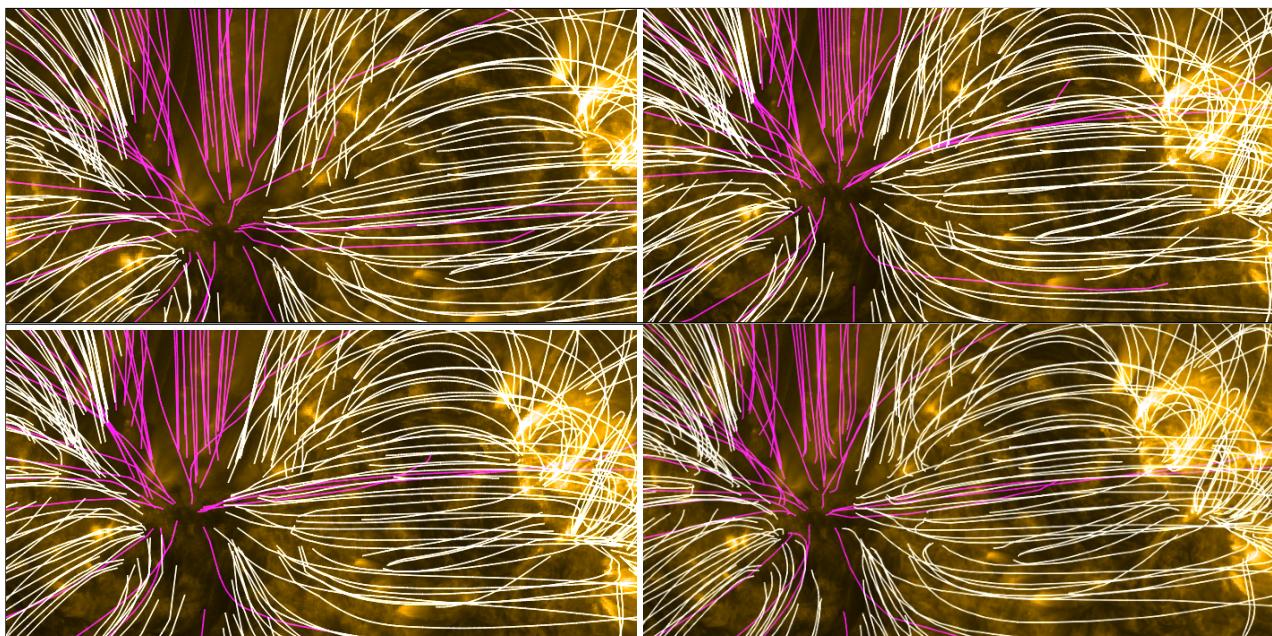


Abbildung 2: Dekomprimierte Feldliniendaten. Oben links ist das Original, rechts die Feldlinien des Adaptiven Subsamplings. Unten sind die Feldlinien der Kompressionen mit der Diskrete Kosinus Transformation und der Prädiktiven Kodierung zu sehen.

## 2 State of the Art

Die Teilschritte einer verlustbehaftete Kompressionen können in drei Verarbeitungsarten eingeteilt werden: Transformationen, Quantisierungen und Entropie Kodierungen. Die Abbildung 3 zeigt eine vereinfachte Abfolge. Die Inputdaten werden durch ein oder mehrere Verfahren transformiert. Die Transformationen haben das Ziel die Daten aufzubereiten, sodass die folgende Quantisierungen unwichtige Informationen löschen können. Im letzten Verarbeitungsschritt werden reduzierte Information Entropie Kodiert. Die Entropie Kodierung versucht die Information mit einem Minimum an Daten abzubilden



Abbildung 3: Vereinfachter Ablauf einer verlustbehafteten Kompression

### 2.1 JPEG/JFIF Bildkompression

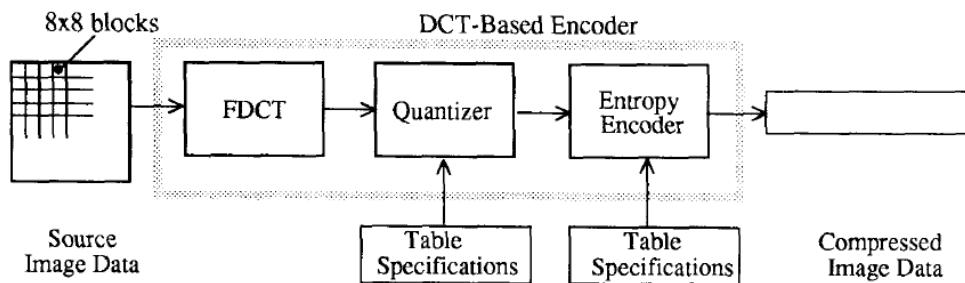


Abbildung 4: Aufbau der JPEG Kompression [1]

Der JPEG/JFIF Standard[1] ist eines der meist verwendeten Bildkompressionsalgorithmen für natürliche Bilder. Das Diagramm der Abbildung 4 zeigt den Aufbau der Kompressionspipeline. JPEG/JFIF unterteilt das Eingabebild in  $8 \times 8$  Blöcke und führt eine Diskrete Kosinus Transformation (DCT) durch. Der Bildblock ist als Folge von Kosinus Funktionen dargestellt.

Die Quantisierung versucht Frequenzen, welche das menschliche Auge schlecht erkennen kann, zu quantisieren und mit weniger Präzision darzustellen. Wenn die Quantisierung gut gewählt wurde, kann das menschliche Auge das dekomprimierte Bild nicht vom Original unterscheiden. JPEG/JFIF bietet vorgefertigte Quantisierungstabellen an. Der Benutzer kann auf das Eingabebild spezialisierte Tabellen anwenden. Wie die Quantisierungstabelle optimal gewählt wird, ist ein aktives Forschungsfeld [2] [3] und kann sich von Anwendungsfall zu Anwendungsfall unterscheiden.

Nach der Quantisierung werden die quantisierten Blöcke im Zick-Zack-Muster [1] angeordnet. So soll die Entropie Kodierung ähnliche Muster finden und eine höhere Kompressionsrate erreichen. Als Entropie Kodierung wird eine Kombination aus Run-Length [4] und einer Huffman-Kodierung [5] verwendet.

Wenn für die Kompression von wissenschaftlichen Daten eine Diskrete Kosinus Transformation eingesetzt wird, kann ein ähnlicher Aufbau verwendet werden wie der JPEG/JFIF Standard. Für eine optimale Kompression wird die Umsetzung der einzelnen Schritte vom Standard abweichen.

## 2.2 Point Cloud Kompression



Abbildung 5: Aufbau einer Octree basierten Point Cloud Kompression.

3d Laser Sampling Geräte produzieren grosse Mengen an dreidimensionalen Punkten von alltäglichen Objekten. Die Kompression von solchen Punktwolken ist ein aktives Forschungsfeld. Eine vorgeschlagene Kompression von Schnabel und Klein [6] verwendet Octrees [7]. Das Diagramm der Abbildung 5 verdeutlicht den Ablauf.

Die dreidimensionalen Punkte werden in einem Octree mit einer begrenzten Anzahl an Levels abgelegt. Im Quantisierungsschritt werden die Punkte durch die Zellenmittelpunkte des Octrees ersetzt. Die Anzahl an Levels ist gleichzusetzen mit der Genauigkeit in Bits welche für jede Koordinatenachse zur Verfügung stehen. Wenn die Levels auf 8 begrenzt sind, steht für jede Achse 8 Bit Genauigkeit zur Verfügung.

In der Entropie Kodierung wird der Octree in Breadth-First Ordnung binär abgebildet (0 für leere Knoten, 1 für gefüllte Knoten). Jeder Level im Octree repräsentiert eine Approximation der Punktwolke. Durch die Breadth-First Ordnung werden die ungenauerer Approximationen zuerst abgelegt. Diese Eigenschaft wird mit einer Prädiktiven Kodierung ausgenutzt: Aus den vorhergehenden Levels wird die Punktverteilung des nächsten Level vorhergesagt. Es wird nur Abweichung der Vorhersage gespeichert. Kann der Prädiktor eine zuverlässige Vorhersage treffen, wird eine hohe Kompressionsrate erreicht.

Um die Point Cloud Kompression auf die Feldlinien anzuwenden muss die Information gespeichert werden, welcher Punkt zu welcher Linie gehört. Schnabel und Klein stellen einen angepassten Algorithmus vor, welcher eine Punktwolke mit Farbinformationen komprimiert. Die Farbinformation kann verwendet werden um einen Punkt einer Feldlinie zuzuordnen. Weiter muss die Prädiktive Kodierung angepasst werden: Schnabel und Klein nehmen an, dass die Punktwolke eine Oberfläche darstellen. Die Feldlinien bilden im Allgemeinen keine Oberfläche und benötigen deshalb eine andere Prädiktive Kodierung.

## 2.3 Curve Fitting

Curve Fitting ist ein Prozess, welcher ein Signal durch eine oder mehrere Funktionen abbildet. Es ist zwischen einem exakten Curve Fitting und einer Approximation zu unterscheiden. Die exakte Repräsentation wird für die Signalinterpolation und eine Approximation in der Rauschunterdrückung verwendet. Eine Datenkompression ist mit Curve Fitting möglich, wenn die Parameter der approximierenden Funktionen weniger Speicherplatz benötigen, als das Signal.

Unser et al [8] zeigt einen Algorithmus auf, welcher ein diskretes Signal als Folge von B-Splines darstellt. Die Anwendungsmöglichkeiten des Verfahrens sind nicht auf Interpolation und Rauschunterdrückung beschränkt: Unser et al. stellt im Ansatz eine Bildkompression vor mittels B-Splines [9]. Der Vorteil des Verfahrens ist, dass sich die Artefakte der Kompression als Rauschunterdrückung und Schärfung des Originalbildes ausdrücken.

Eine Datenkompression mit Curve Fitting ist im Vergleich mit der Kosinus Transformation wenig erforscht: Es sind keine Kompressionsstandards bekannt, welche ein Curve Fitting für die Datenkompression verwenden. Ein Kompressionsverfahren zu entwickeln zieht zusätzlichen Aufwand mit sich als Verfahren, welche in der Datenkompression etabliert sind.

## 2.4 Compressive Sensing

Compressive Sensing ist ein Verfahren, welches ein Signal mit einer minimalen Anzahl an vordefinierten Funktionen darstellt (sparse representation). Die Funktionen sind durch keine Rahmenbedingungen begrenzt, jede allgemeine Funktion kann im Compressive Sensing verwendet werden.

Das Überführen in die sparse representation ist ein  $np-hard$  Problem [10]. Es existieren Heuristiken wie Orthogonal Matching Pursuit[11], welches Approximation der sparse representation mit linearer Komplexität erstellt. Die Approximation ist für praktische Anwendungsfälle wie eine Datenkompression mittels Compressive Sensing ausreichend genau.

Mit einem allgemeinen Dictionary von vordefinierten Funktionen ist jedes Signal rekonstruierbar. Die Stärke des Compressive Sensing Ansatzes ist, dass das Dictionary auf die Signale optimiert werden kann. Der K-SVD [12] Algorithmus erlernt mittels Unsupervised Learning ein Dictionary, welches auf die zu kodierenden Signale zugeschnitten ist.

Compressive Sensing hat das Potential eine hohe Kompressionsrate zu erreichen. Die Feldlinien sind oft ähnlich und unterscheiden sich hauptsächlich in Rotation, Verschiebung und Skalierung. Mit einem auf Feldlinien angepasstes Dictionary kann eine Feldlinie durch wenige Funktionen approximiert werden.

## 2.5 Entropie Kodierung

Die Entropie Kodierung findet in allen Bereichen der Informatik Anwendungen. Ziel der Kodierung ist die gleiche Information mit einem Minimum an Daten darzustellen. Verlustbehaftete Kompressionsverfahren reduzieren die Information und verwenden im letzten Schritt eine Entropie Kodierung um die Datenmenge zu reduzieren.

Es ist zwischen spezialisierten und allgemeinen Entropie Kodierer zu unterscheiden. Unter den spezialisierten Verfahren ist Beispielsweise die Arbeit von Ratanaworabhan et al. [13] einzuordnen, welche in der Lage ist Floating Point Daten performant zu kodieren und dekodieren.

Unter den allgemeinen Verfahren sind Archivierer wie GZIP [14], 7-ZIP [15] und RAR [16] angesiedelt. Archivierungsverfahren finden in allen Bereichen Anwendung und dienen in der Entwicklung von spezialisierten Kodierer als Messbasis. Im Vorfeld wurde die Kompressionsraten von GZIP, 7-ZIP und RAR von Feldliniendaten verglichen, wobei RAR die höchsten Raten erreichen konnte.

Die Verfahren der Archivierer bestehen aus Kombinationen von Entropie Kodierungen. Zum Beispiel verwendet GZIP den DEFLATE Algorithmus [17], welcher aus einer Kombination von LZ77 [18] und Huffman [5] Kodierung besteht. Rar hingegen besitzt unterschiedliche Kombinationen und wählt das Verfahren aus, welches die Inputdaten optimal kodieren kann. Dadurch kann RAR für unterschiedlichen Anwendungsfällen eine hohe Kompressionsrate erreichen. Die RAR Kompression ist urheberrechtlich geschützt und die verwendeten Algorithmen sind nicht bekannt.

### 3 Umgesetzte Kompressionsverfahren

In diesem Abschnitt werden die einzelnen Verfahren vorgestellt. Im ersten Abschnitt 3.1 wird die Ist-Kompression analysiert. In den nächsten Abschnitten 3.2 bis 3.4 werden die Lösungsansätze und ihre Teilschritte vorgestellt.

#### 3.1 Ist-Kompression



Abbildung 6: Aufbau der Ist-Kompression.

Das Ziel der Ist-Kompression ist es, die Datenmenge mit einfachen Mitteln zu reduzieren. Der Aufbau ist im Diagramm der Abbildung 6 dargestellt. Die Ist-Kompression führt zuerst ein Subsampling durch. Drei Viertel der Daten werden in diesem Schritt verworfen. Im Quantisierungsschritt werden die Daten von 32 Bit Floating Point auf 16-Bit Integer diskretisiert. In der Entropie Kodierung werden die Daten geordnet wie in Tabelle 2 dargestellt. Als erstes werden die Längen aller Feldlinien abgelegt. Danach folgt der X, Y und der Z Kanal

Anzahl Punkte der Feldlinien	X Kanal aller Punkte	Y Kanal aller Punkte	Z Kanal aller Punkte
------------------------------	----------------------	----------------------	----------------------

Tabelle 2: Anordnung der Simulationsdaten der Ist-Kompression

aller Punkte der Feldlinien. Diese Anordnung verbessert die Kompressionsrate der Entropie Kodierung. Je näher ähnliche Muster beieinander liegen, desto besser können sie komprimiert werden. Für die eigentliche Entropie-Kodierung wird GZIP verwendet.

Die Punktmenge ist für Low-End Grafikkarten zu gross. Um die Punktmenge für die Visualisierung zu verkleinern, führt der JHeliovewer ein weiteres Subsampling durch, welches im Abschnitt 3.2.1 beschrieben ist.

#### 3.2 Kompressionsverfahren: Adaptives Subsampling



Abbildung 7: Aufbau des Kompressionsverfahrens: Adaptives Subsampling.

Diesem Verfahren verwendet einen ähnlichen Aufbau wie die Ist-Kompression. Der Unterschied ist, dass das Subsampling Verfahren gewählt wurde, welches der JHeliovewer auf dem Client durchführt. Die Abbildung 7 zeigt den neuen Ablauf.

Es ist möglich dass die Visualisierung zu einem späteren Zeitpunkt mehr Punkte darstellen soll. In diesem Fall müssten entweder mehr Punkte übertragen werden, was eine schlechtere Kompressionsrate zur Folge hat, oder der JHeliovewer muss eine Interpolation durchführen.

### 3.2.1 Adaptives Subsampling

Konzeptionell approximiert das Adaptive Subsampling eine Punktfolge durch eine Folge von Strecken. Je stärker die Krümmung der Kurve, desto mehr Strecken wird für die Approximation benötigt.

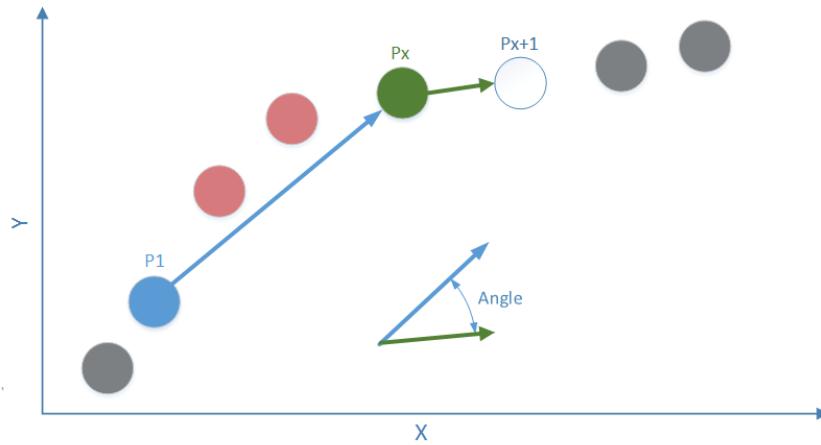


Abbildung 8: Darstellung des Adaptiven Subsapmlings. Rot sind die Punkte, welche geprüft und gelöscht wurden. Grün ist der Punkt dargestellt, welcher geprüft wird.

Das Diagramm der Abbildung 8 stellt das Subsampling im zweidimensionalen Raum dar. Das Adaptive Subsampling wählt nun Punkte  $P$  aus der Feldlinie aus, welche Start- und Endpunkte der Strecken darstellen.  $P_1$  wurde bereits ausgewählt. Es wird nun ein Punkt  $P_x$  gesucht, der als Endpunkt einer Strecke von  $P_1$  zu  $P_x$  die Daten approximiert. Dazu wird der Winkel der Strecke  $P_1$  zu  $P_x$  mit der Strecke  $P_x$  zu  $P_{x+1}$  verglichen. Wenn der Winkel kleiner ist, als ein Winkel  $\alpha$ , wird der nächste Punkt  $P_{x+1}$  überprüft. Wenn der Winkel grösser ist, wird  $P_x$  ausgewählt. Danach wird der wird eine nächste Strecke startend von  $P_x$  gesucht.

### 3.2.2 Entropie Kodierung mittels RAR

Die Anordnung der Daten wurde aus der Ist-Kompression übernommen, jedoch wird RAR anstatt GZIP verwendet. GZIP konnte bei den Ist-Komprimierten Daten eine Kompressionsrate von 1.2 erreichen, während RAR bei selben Daten eine Rate von 3.7 erreicht.

### 3.3 Kompressionsverfahren: Diskrete Kosinus Transformation

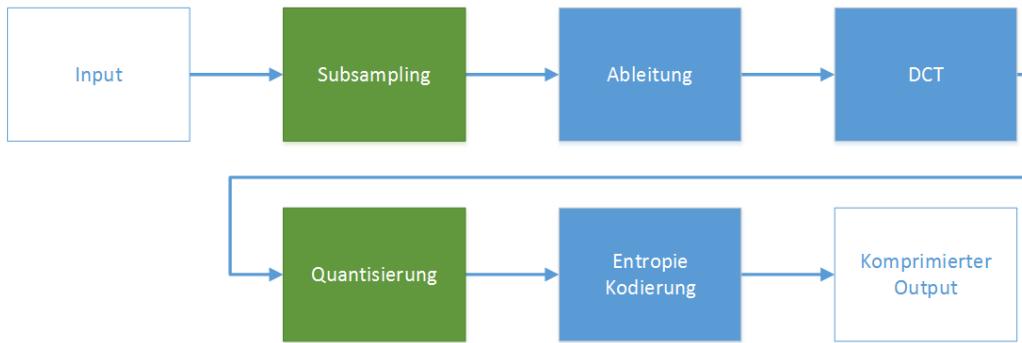


Abbildung 9: Aufbau der Kompression Adaptives Subsampling.

Die Kompression dieses Verfahrens ist Dargestellt im Diagramm der Abbildung 9. Konzeptionell ähnelt der Algorithmus der JPEG/JFIF Kompression (dargestellt in der Abbildung 4). Im Vergleich zum JPEG/JFIF Standard ist der grösste Unterschied, dass das Eingangssignal vor der Kosinus Transformation abgeleitet wird.

Die Feldlinien ähneln oft harmonischen Halbwellen, welche sich durch wenige Kosinusfunktionen approximieren lassen. Um eine optimale Kompression mit diesem Verfahren zu erreichen, müssen Ringing Artefakte [19] behandelt werden. Sie äussern sich als Oszillationen im dekomprimierten Signal, was das menschliche Auge als störend empfindet. Beispiele für die Ringing Artefakten von Feldlinien sind im Abschnitt 6.2.6 zu finden. Es existieren Ansätze die Ringing Artefakte mit Post-Processing Schritten zu dämpfen: Die simpelste Variante ist, das dekomprimierte Signal zu glätten. Die Glättung kann das Signal verfälschen und ist deshalb nicht die optimale Lösung. In der Bild- und Audioverarbeitung wird nach Post-Processing Filter geforscht, welche die Ringing Artefakte in der Dekompression vermindern [20] [21]. Ein angepasstes Postprocessing wurde im Rahmen dieser Arbeit nicht implementiert.

#### 3.3.1 Subsampling

Das Subsampling wurde aus der Ist-Kompression 3.1 übernommen. Die Daten werden um den Faktor vier verkleinert und dient, die DCT zu beschleunigen. Eine naive Implementation der DCT weist eine Komplexität von  $O(n^2)$  auf. Es wird eine Implementation des FDCT Algorithmus verwendet, welche eine tiefere Komplexität von  $O(n \log(n))$  besitzt. Der Rechenaufwand ist bei diesem Verfahren trotzdem höher als bei den Verfahren des Adaptiven Subsamplings und der Prädiktiven Kodierung.

Falls das Verfahren weiter beschleunigt werden soll, können die Linien in Blöcke unterteilt werden und die DCT pro Block ausführen. Dadurch wird die Komplexität auf  $O(n)$  gesenkt. Jedoch ist es wahrscheinlich, dass durch die Unterteilung die Kompressionsrate leidet. Die Approximation mehrere Blöcke des Eingabesignals benötigt schlussendlich mehr Kosinus Funktionen, als die Approximation des gesamten Signals.

#### 3.3.2 Ableitung

Das Eingabesignal wird abgeleitet und alle folgenden Transformationen werden auf den Steigungen des Signals ausgeführt. Damit die Transformation umkehrbar ist, muss der Startwert des Signals zusätzlich abgespeichert werden.

Die Artefakte, welche das abgeleitete dekomprimierte Signal beinhaltet, sind bei der Feldliniensimulation weniger störend für das menschliche Auge. Die Feldlinie bleibt tendenziell glatt und Artefakte äussern sich meist in veränderten Amplituden, welche erst erkennbar sind, wenn die Originalfeldlinie zum Vergleich bereit steht.

Diese Eigenschaft wirkt sich ebenfalls auf die Ringing Artefakte aus. Die Ableitung hat einen dämpfenden Effekt auf die Ringing Artefakte.

### 3.3.3 Diskrete Kosinus Transformation

Die Diskrete Kosinus Transformation stellt eine endliche Menge von  $N$  Datenpunkten als  $N$  Kosinus Funktionen dar. Die Werte der DCT-Koeffizienten stellen dar, wie hoch der Anteil einer bestimmten Frequenz im Originalsignal ist. Im optimalen Fall kann ein Signal durch niederfrequente Funktionen approximiert werden. Die hochfrequenten Anteile stellen Detailinformationen dar, welche weniger relevant sind für die Rekonstruktion des Signals als die niederfrequenten Anteile.

Es existieren verschiedene Möglichkeiten eine DCT umzusetzen. Hier wurde sich am JPEG/JFIF Standard orientiert, welche die DCT-II (3.1) als Forwärts und die DCT-III (3.2) als Rückwärtstransformation verwendet [1].

$$X_k = \sum_{n=0}^{N-1} x_n * \cos\left[\frac{\pi}{N}k(n + \frac{1}{2})\right] \quad k = 0, 1, \dots, N - 1 \quad (3.1)$$

$$x_n = \frac{1}{2}X_0 + \sum_{k=1}^{N-1} X_k * \cos\left[\frac{\pi}{N}k(n + \frac{1}{2})\right] \quad n = 0, 1, \dots, N - 1 \quad (3.2)$$

$N$  bezeichnet die Länge des Eingabesignals,  $x_n$  bezeichnet einen Wert im diskreten Signal und  $X_k$  ist der Anteil der Frequenz  $k$ . Ein Eingabesignal der Länge  $N$  resultiert in  $N$  Kosinus-Funktionen.

Die DCT transformiert ein periodisches, unendliches Signal. Um ein endliches Signal zu transformieren, wird das Inputsignal konzeptionell wiederholt. Die gewählten Verfahren wiederholen das Signal jeweils in umgekehrter Reihenfolge. Die Wiederholung beeinflusst die Transformation nicht, wenn das Signal an den Rändern abflacht. Falls das Signal nicht abflacht, können nach der Quantisierung an den Rändern markante Artefakte auftreten. Ein Beispiel für solche Artefakte ist im Diagramm der Abbildung 20 im Abschnitt 6.2.1 zu sehen.

### 3.3.4 Quantisierung

In der Visualisierung werden die Feldlinien in drei Typen unterschieden: "Sonne zu Sonne", "Sonne ins Weltall" und "Weltall zur Sonne". Die "Sonne zu Sonne" Feldlinien ähneln harmonischen Halbwellen und brauchen wenigerere Kosinus Funktionen für eine Approximation als die anderen Typen von Feldlinien. Für die "Sonne zu Sonne" Feldlinien werden maximal 35 DCT Koeffizienten abgelegt, wobei die letzten zehn Koeffizienten kaum noch Einfluss besitzen. Sie werden mit einem hohen Faktor quantisiert, sodass sie im Normalfall ebenfalls Null sind.

Die anderen Feldlinien benötigen mehr Koeffizienten für die Dämpfung der Ringing Artefakte. Bei ihnen liegt das Maximum bei 50 Koeffizienten.

### 3.3.5 Entropie Kodierung

Um die Entropie Kodierung zu verbessern wurden zwei Byte-Kodierungen hinzugefügt. Die Längenkodierung reduziert Null-Einträge der Kanäle und die adaptive Genauigkeits-Kodierung reduziert die benötigten Bytes pro Wert eines Kanals.

#### Längenkodierung

Die Quantisierung erlaubt eine maximale Anzahl an Koeffizienten. Ab dem 50 Koeffizienten, sind die Werte Null. Die Längenkodierung schneidet den Block von Null-Koeffizienten ab und fügt die Länge des Nicht-Null Blockes hinzu. Alle Längen und alle Blöcke werden zusammen abgespeichert, die Tabelle 3 verdeutlicht das

Konzept.  $n_i$  ist die Länge des Nicht-Null-Blocks der Feldlinie  $i$  und  $x_{i,j}$  ist der  $j$  DCT-Koeffizient der Feldlinie  $i$ .

X Kanäle							
Block Feldlinie 0					...		
$n_0$	$x_{0,0}$	$x_{0,1}$	$\dots$	$x_{0,n-1}$	$n_1$	$x_{1,0}$	$\dots$

Tabelle 3: Beispiel eines abgespeicherten Kanals mit der Längenkodierung.

Um einen Kanal zu Dekodieren, muss die Anzahl an DCT-Koeffizienten  $n_i$  gelesen werden und danach Nullen anfügen, bis die ursprüngliche Länge des Kanals erreicht wurde.

Die Längenkodierung ist effizient, wenn die hochfrequenten DCT-Koeffizienten einen grossen, zusammenhängenden Null-Block bilden. Im schlechtesten Fall wäre der letzte DCT-Koeffizient nicht Null. In diesem Fall würde die Längenkodierung nichts abschneiden.

### Adaptive Genauigkeits-Kodierung

Im Allgemeinen reichen sieben Bit Genauigkeit um die quantisierten DCT Koeffizienten darzustellen. Mit der adaptiven Genauigkeits-Kodierung wird ein Minimum an Bytes pro Koeffizient abgespeichert. Die Tabelle 4 zeigt die Byte-Kodierung. Das MSB wird als "Continue Flag" verwendet. Wenn es gesetzt ist, gehört das folgende Byte ebenfalls zur Zahl. Die Kodierung führt zu einer Datenreduktion, wenn die Werte im Allgemeinen mit einem Byte dargestellt werden können.

Byte							
Continue Flag	X	X	X		X	X	X

Tabelle 4: Aufteilung eines Bytes der adaptiven Genauigkeitskodierung. X sind Nutz-Bits.

### 3.4 Kompressionsverfahren: Prädiktive Kodierung

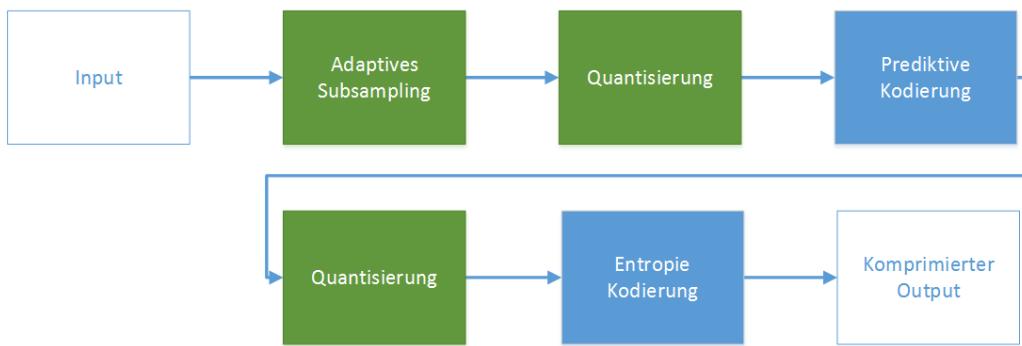


Abbildung 10: Aufbau der Kompressionsverfahren Prädiktive Kodierung.

Die Prädiktive Kodierung ist ein Verfahren, welches in der verlustfreien Kompressionen Anwendung findet. Mittels eines Prädiktors und den bereits kodierten Daten wird eine Vorhersage erstellt für die folgenden Daten erstellt. Die Prädiktive Kodierung speichert ausschliesslich den Vorhersagefehler.

Der Aufbau dieser Kompression ist im Diagramm der Abbildung 10 dargestellt. Im ersten Schritt wird ein Subsampling und eine Quantisierung der Daten durchgeführt. Darauf folgend werden die quantisierten Daten mit einer Prädiktiven Kodierung versehen. Der zweite Quantisierungsschritt löscht unwichtige Vorhersagefehler.

Das Adaptive Subsampling reduziert die Anzahl Punkte und verändert die Eigenschaften der Kanäle der einzelnen Feldlinien. Stetige Steigungen werden zerstört, ein Beispiel ist im Diagramm der Abbildung 36 im Abschnitt 6.3.2. Einfache Prädiktoren wie zum Beispiel ein linearer Prädiktor kann auf solche Daten keine zuverlässige Vorhersage treffen und verbessert die Kompressionsrate nicht. Es wurde deshalb der Rekursiver Linearer Prädiktor entwickelt, welche auf der Idee der Point Cloud Kompression aus Abschnitt 2.2 basiert. Die Daten werden in unterschiedlich genauen Approximationen dargestellt. Der Prädiktor benutzt die ungenaue Approximation um eine Vorhersage für die nächst genauere Approximation zu erstellen.

#### 3.4.1 Angle Subsampling und Quantisierung

Das Angle-Subsampling der Kompression aus Abschnitt 3.2 wurde mit angepassten Parametern übernommen. In dieser Kompression werden 50% mehr Daten übertragen. Die Quantisierung überführt die Punkte in ein diskretes, sphärisches Koordinatensystem. In diesem Koordinatensystem können die Koordinatenachse mit 16 Bit Genauigkeit abgebildet werden, ohne hohe Abweichungen vom Originalpunkt in Kauf zu nehmen.

#### 3.4.2 Rekursiver Linearer Kodierung

Der Algorithmus der Rekursiven Linearen Kodierung versucht aus einer ungenauen Approximation die nächst genauere vorherzusagen. Im Diagramm der Abbildung 11 ist der Kodierungsschritt des Algorithmus dargestellt. Es wird angenommen, dass der mittlere Wert des Kanals auf der Strecke zwischen Start- und Endwert liegt. Der Algorithmus wird rekursiv für die zwei Teilstücke des Kanals wiederholt, bis alle Werte zwischen Start- und Endwert des Kanals kodiert wurden. Start und Endwert des Kanals werden unkodiert abgespeichert. Die Kodierung der nächsten Rekursionsstufe ist im Diagramm der Abbildung 12 dargestellt.

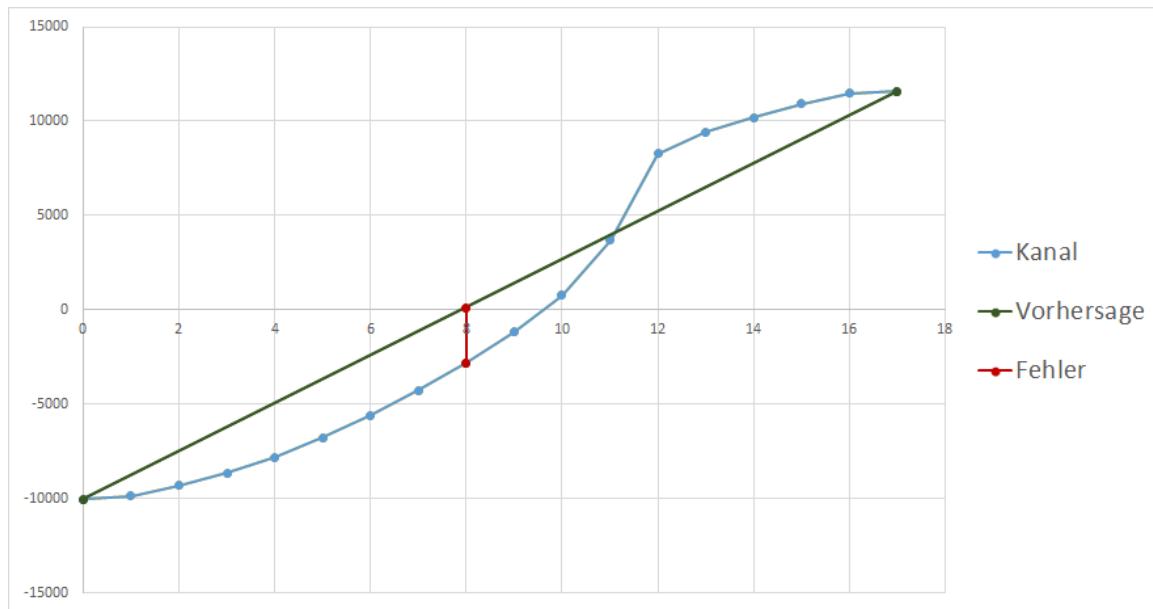


Abbildung 11: Erster Schritt der Rekursiver Linearer Kodierung. Zu sehen sind das zu kodierende Signal, die Vorhersage und den Fehler der Vorhersage.

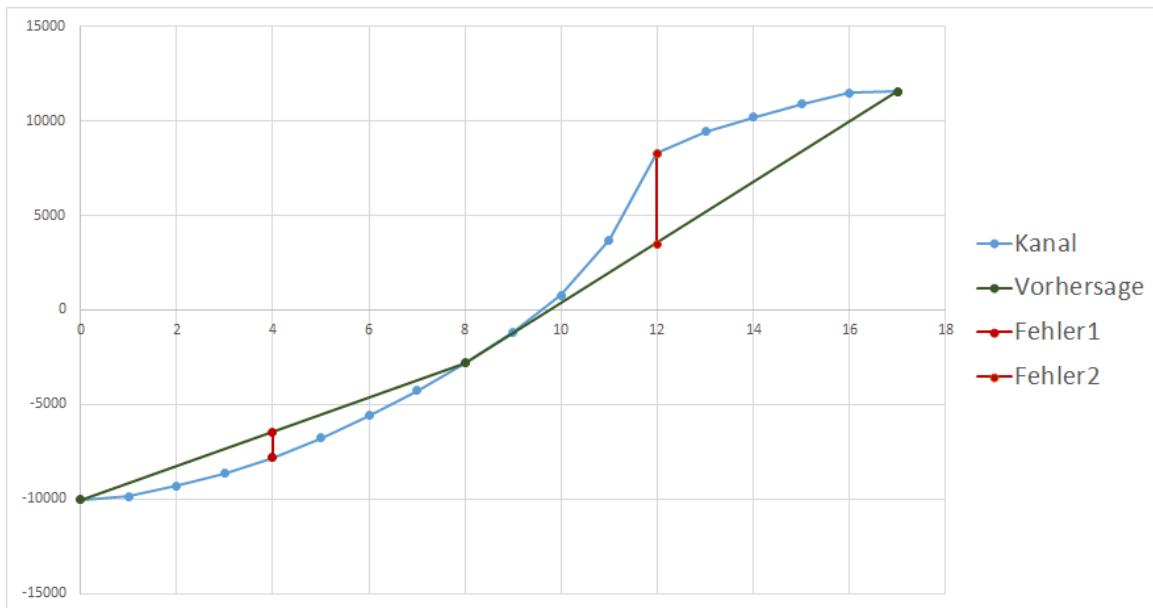


Abbildung 12: Zweiter Schritt der Rekursiver Linearer Kodierung. Die Vorhersage beinhaltet nun zwei Strecken.

### 3.4.3 Quantisierung

Die Vorhersagefehler der Rekursiven Linearen Kodierung werden im Allgemeinen von Rekursionsstufe zu Rekursionsstufe kleiner. In der letzten Rekursion werden Detailinformationen kodiert, welche für meist nicht relevant sind. Die Vorhersagefehler der ersten Rekursionsstufen werden mit höherer Genauigkeit abgespeichert. Dadurch kann eine hohe Kompressionsrate erreicht werden zu begrenzten Artefakten.

### 3.4.4 Entropie Kodierung

Die Vorhersagefehler werden in Breadth-First Ordnung abgespeichert. Der Fehler der Rekursionsstufe 0 wird zuerst abgelegt, gefolgt von den Fehlern der Stufe 1 etc. Die Tabelle 5 zeigt den Aufbau. Die Vorhersagefehler der ersten Levels sind grösser, als die der Letzten. Diese Ordnung hilft ähnliche Strukturen beieinander zu halten.

Vorhersagefehler							
Stufe 0	Stufe 1		Stufe 2			...	
$Fehler_0$	$Fehler_1$	$Fehler_2$	$Fehler_3$	$Fehler_4$	$Fehler_5$	$Fehler_6$	...

Tabelle 5: Breadth First Ordnung der Vorhersagefehler.

Die Vorhersagefehler werden mit der Adaptiven Genauigkeitskodierung aus Abschnitt 3.3.5 kodiert. Nach der Quantisierung reichen im Allgemeinen sieben Bit Genauigkeit aus um einen Fehler abzulegen. Die Längenkodierung wurde nicht verwendet.

## 4 Qualitätsmessung der dekomprimierten Daten

Bei verlustbehafteten Kompressionen muss die Qualität der dekomprimierten Daten sichergestellt werden. Im Verlauf der Arbeit wurden zwei Metriken verwendet: Die Standardabweichung und eine angepasste PSNR-HVS-M. Die Standardabweichung wird unabhängig von der Abtastrate berechnet und kann Subsampling Artefakte entdecken. Wie die Standardabweichung berechnet wird, ist im Abschnitt 4.2 beschrieben.

Die PSNR-HVS-M wird für die Aufdeckung von Ringing Artefakten [19] berechnet. Ein Beispiel für Ringing Artefakte ist in der Abbildung 26 im Abschnitt 6.2.6 zu sehen. Die PSNR-HVS-M stammt aus der Bildverarbeitung. Das Ziel des Fehlermasses ist es, eine hohe Korrelation zwischen der Metrik und dem menschlichen Augenmass zu erreichen. Wie die PSNR-HVS-M Metrik angepasst und umgesetzt wurde, ist im Abschnitt 4.3 beschrieben.

Für die Messungen wurden spezielle Aufnahmen der Feldlinien gewählt. Wie die Aufnahmen ausgewählt wurden, ist im Abschnitt 4.1 beschrieben.

### 4.1 Auswahl und Erhebung der Testdaten

Die Testdaten sollen zu einem alle Randfälle abdecken, als auch durchschnittliche Fälle enthalten. Aus diesem Grund wurden insgesamt zehn Datensätze ausgewählt: Vier Datensätze mit hoher Sonnenaktivität, zwei mit wenig und vier zufällig. Für die vier Datensätzen mit hoher Aktivität wurde in den Jahren 2014 und 2013 nach den grössten Solaren Flares gesucht. Für die Datensätze mit wenig Aktivität wurde nach Zeiträumen mit möglichst kleinen Solar Flares gesucht. Die Feldlinien werden alle sechs Stunden berechnet und Solar Flares sind spontane Ereignisse. Für die Solar Flares wurde deshalb beachtet, dass die Datensätze vor dem Ereignis verwendet wurden. Grosse Solar Flares entladen das Magnetfeld der Sonne. Vor dem Ereignis ist das Magnetfeld komplex.

Wie im Abschnitt 3.1 beschrieben, wird bereits eine einfache verlustbehaftete Kompression Simulation. Für die Testdaten wurde diese entfernt, was die rohe Datenmenge entsprechend anwachsen liess auf durchschnittlich 10 Megabyte pro Simulation.

### 4.2 Berechnung der Standardabweichung

Die dekomprimierte Linie ähnelt dem Original, wenn die Abweichungen konstant bleiben. Für diesen Fall wurde die Standardabweichung ausgewählt. Die originale und dekomprimierte Feldlinie können unterschiedliche Abtastraten aufweisen. Die Standardabweichung muss deshalb unabhängig von der Abtastrate berechnet werden.

Um die Abweichung zu berechnen wird konzeptionell zwischen die dekomprimierten Punkte eine Linie gezogen und den Abstand zwischen dieser Linie und den Originalpunkte berechnet. Der Vorgang ist dargestellt im Diagramm der Abbildung 13. Für jeden Punkt  $P'_i$  aus den dekomprimierten Punkten  $D$ , nehme  $P'_i, P'_{i-1}, P'_{i+1}$  und  $P'_{i+2}$ . Ziehe drei Strecken, von  $P'_{i-1}P'_i$ ,  $P'_iP'_{i+1}$  und  $P'_{i+1}P'_{i+2}$ . Suche von  $P'_i$  den Originalpunkt  $P_i$  aus allen Originalpunkten  $O$ , berechne den minimalen Abstand zu den drei Strecken. Führe das für alle folgenden Originalpunkte durch, bis  $P_{i+1}$  erreicht wurde.

Die Abstandsberechnung von Strecke  $s$  zu einem Punkt  $P$  erfolgt in zwei Schritten: Zuerst wird mit der Formel (4.1) überprüft, ob eine Senkrechte durch  $P$  auf der Strecke  $s$  zu liegen kommt. Falls das der Fall ist, wird der Abstand von  $P$  zu  $s$  berechnet (4.2). Falls nicht, wird die kürzeste Distanz der Eckpunkte der Strecke zu  $P$  berechnet.

$$t = \frac{\vec{AB} * \vec{AP}}{|\vec{AB}|^2} \quad 0 \leq t \leq 1 \quad (4.1)$$

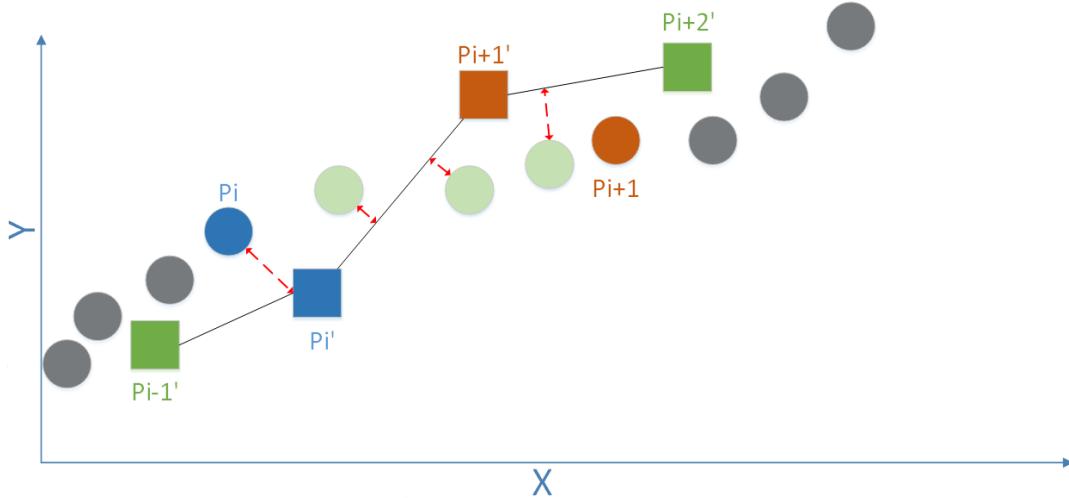


Abbildung 13: Darstellung der Fehlerberechnung. Die Punkte sind die Originaldaten, die Quadrate sind die Punkte nach der Kompression.

$$distance = \frac{|\vec{BA} \times \vec{BP}|}{|\vec{BP}|} \quad (4.2)$$

$A$  und  $B$  sind die Eckpunkte der Strecke. Falls  $0 \leq t \leq 1$ , existiert eine Senkrechte durch  $P$  mit Fußpunkt auf der Strecke  $s$ . Die Distanz von  $P$  zu  $s$  wird mit der Formel (4.2) berechnet.

Wenn das nicht möglich ist, wird der kürzere Distanz von  $P$  zu einem der Eckpunkte genommen.

#### 4.2.1 Berechnung der Standardabweichung

$$\sigma(X) = \sqrt{variance(X)} \quad (4.3)$$

$$variance(X) = \sum_{i=0}^{N-1} (x_i - E(x_i))^2$$

Die Standardabweichung  $\sigma$  einer Beobachtungsreihe  $X$  ( $x_1, x_2, x_3, \dots, x_n - 1$ ) ergibt sich aus der Wurzel der Varianz von  $X$ . Die Varianz von  $X$  kann errechnet werden, wenn man den Distanz jeder Beobachtung  $x_i$  mit dem Erwartungswert  $E(x_i)$  berechnet und quadriert. Die Beobachtung ist im diesen Fall ein Punkt der dekomprimierten Linie, während der Erwartungswert der Originalpunkt ist. Die Distanz wird mit dem besprochenen Verfahren 4.2 berechnet. Die Summe der quadratischen Abstände ergibt die Varianz. Die Varianz wird über alle Testdaten berechnet, somit erhält man für einen Test einen Wert für die Standardabweichung.

#### 4.3 Berechnung der angepassten PSNR-HVS-M

Die Peak-Signal-Noise-Ratio (PSNR) Metrik ist ein weitverbreitetes Fehlermass in der Bildverarbeitung. Es wird für die Messung des Fehlers zwischen dekomprimierten Bild und dem Original eingesetzt.

$$PSNR = 20 * log_{10}(MAX_I) - 10 * log_{10}(MSE) \quad (4.4)$$

$$MSE = \frac{1}{n} \sum_{i=0}^{N-1} [E(i) - D(i)]^2$$

Die PSNR (4.4) besteht aus dem maximal möglichen Wert  $MAX_I$  und dem "Mean Squared Error" ( $MSE$ ). Wobei  $E()$  die Originaldaten und  $D()$  die verzerrten Daten darstellen. Das Problem der PSNR ist, dass sie nicht mit der menschlichen Wahrnehmung übereinstimmt. Ponomarenko et al. [22] haben eine modifizierte PSNR entwickelt; die PSNR Human Visual System Masking (HVS-M). In ihren Messungen erreichten sie eine hohe Korrelation zwischen menschlicher Wahrnehmung von verzerrten Bildern und dem neuen Fehlermass.

Der Unterschied zwischen der PSNR und der PSNR-HVS-M liegt in der Berechnung des durchschnittlichen quadratischen Fehlers. Das Diagramm der Abbildung 14 zeigt den Ablauf der neuen Berechnung.

PSNR-HVS-M berechnet die Differenz zwischen dem Originalbild  $E$  und dem verrauschten Bild  $D$  und führt

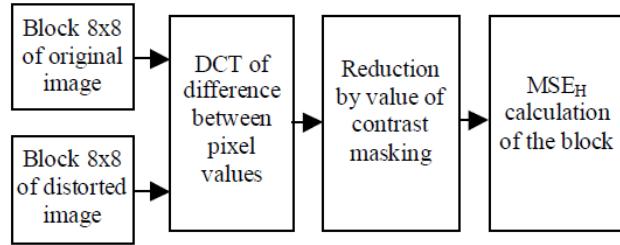


Abbildung 14: Flussdiagramm der PSNR-HVS-M Berechnung [22].

die Daten mittels einer DCT in den Frequenzraum. Der nächste Schritt Contrast Maskin reduziert die Differenz, wenn das menschliche Auge den Frequenzunterschied nicht erkennen kann. Aus den  $MSE_H$  Werten wird mit der Formel (4.4) die PSNR-HVS-M berechnet.

#### 4.3.1 Contrast Masking

Um das Contrast Masking zu berechnen, führt Ponomarenko et al. die gewichtete Energie der DCT Koeffizienten  $E_w$  (4.5) und den masking effect  $E_m$  (4.6) ein:

$$E_w(X) = \sum_{i=0}^7 \sum_{j=0}^7 [X_{ij}]^2 C_{ij} \quad (4.5)$$

$$E_m(X) = \frac{1}{f_m} E_w(X) \quad (4.6)$$

Wobei  $X$  die Kosinus-Koeffizienten eines  $8 * 8$  Bildblocks sind und  $C$  die Gewichtungen der Frequenzen. Der Normalisierungsfaktor  $f_m$  wurde experimentell ermittelt und auf 16 festgelegt. Ponomarenko et al. argumentiert, dass der Unterschied zwischen einem Block  $X_e$  und einem verrauschten Block  $X_d$  unsichtbar sind, wenn die Ungleichung (4.7) erfüllt ist.

$$E_w(X_e - X_d) < \max[E_m(X_e), E_m(X_d)] \quad (4.7)$$

Die Eigenschaft der Ungleichung (4.7) fließt mit der Formel (4.8) in die Distanzberechnung ein.

$$X_{\Delta ij} = \begin{cases} X_{eij} - X_{dij}, & i = 0, j = 0 \\ 0, & |X_{eij} - X_{dij}| \leq E_{norm} / C_{ij} \\ X_{eij} - X_{dij} - E_{norm} / C_{ij}, & X_{eij} - X_{dij} > E_{norm} / C_{ij} \\ X_{eij} - X_{dij} + E_{norm} / C_{ij}, & \text{otherwise} \end{cases} \quad (4.8)$$

Wobei  $E_{norm} = \sqrt{\max[E_m(X_e), E_m(X_d)]/64}$ .  $X_{eij}$  ist der DCT Koeffizient des Originalblockes und  $X_{dij}$  der Koeffizient des verrauschten Blockes. Dabei ist es irrelevant ob die DCT Koeffizienten subtrahiert oder

wie in der Abbildung 14 angedeutet, zuerst die Differenz der Pixelwerte berechnet und diese DC transformiert werden.

Wie gut die PSNR-HVS-M Metrik mit dem menschlichen Auge übereinstimmt, hängt vom Normalisierungsfaktor  $f_m$  und von der Wahl der Gewichtungen  $C$  ab. Ponomarenko et al. verwendeten die normalisierten ( $\frac{10}{x}$ ) und quadrierten Werte der JPEG/JFIF Quantisierungsmatrix [1]. Es ist zu beachten, dass der DC-Koeffizient [23] nicht im Contrast Masking berücksichtigt wird, für den Wert wird die normale PSNR berechnet. Der DC-Koeffizient stellt die durchschnittliche Helligkeit in einem Block dar. Das menschliche Auge kann auch kleine Unterschiede in dieser Frequenz erkennen.

#### 4.3.2 Umsetzung und Anpassung der PSNR-HVS-M für diese Arbeit

Der grösste Unterschied zur Arbeit von Ponomarenko et. al. ist der Einbezug des DC-Koeffizienten ins Contrast-Masking. Der DC-Koeffizient repräsentiert bei den Feldlinien eine Verschiebung. Das menschliche Auge kann leichte Verschiebungen der Feldlinien kaum unterscheiden. Des Weiteren musste für die Feldlinie eine eigene Quantisierungsmatrix gefunden werden, aus denen die Gewichtungen berechnet werden. Es wurde eine DCT-Kompression entwickelt, welche keine sichtbaren Artefakte aufweist. Die Quantisierungsmatrix wurde übernommen und auf dieselbe Weise normalisiert. Für die PSNR wird der maximale Wert  $MAX_I$  benötigt. Hier wurde der maximal mögliche Wert der PFSS Simulation verwendet, den vierfachen Sonnenradius.

Der letzte Wert, den es zu setzen gibt, ist der Normalisierungsfaktor  $f_m$ . Ponomarenko et. al. schrieb keine zusätzliche Information oder Begründung zu diesem Wert. Dieser Wert stellt ein, wie stark das Contrast-Masking einfließen soll. Je höher der Wert ist, desto ähnlicher ist die PSNR-HVS-M Metrik der normalen PSNR.

Die Tabelle 6 erforscht den Einfluss des  $f_m$  Faktors. Dazu wurde die PSNR-HVS-M zu drei dekomprimierten Simulationen berechnet, welche eine unterschiedliche Qualität aufweisen.

Qualität	$f_m$ 8	$f_m$ 16	$f_m$ 32
Keine sichtbare Artefakte	96.8 dB	<b>95.6</b> dB	94.5 dB
Kaum sichtbare Artefakte	95.8 dB	<b>94.4</b> dB	93.3 dB
Sichtbare Artefakte	90.0 dB	<b>88.3</b> dB	87,0 dB

Tabelle 6: Einfluss des  $f_m$  Faktors auf die PSNR-HVS-M.

Für diesen Anwendungsfall scheint der Faktor  $f_m$  sich stabil zu verhalten: der Faktor kann verdoppelt oder halbiert werden und die resultierenden Distanzen bleiben in derselben Größenordnung. Es wurde der Standardwert von 16 übernommen. Eine Anpassung von  $f_m$  scheint nicht die Metrik massgebend zu verändern und hat vermutlich weniger Einfluss als die Gewichtung der DCT Koeffizienten.

Die PSNR-HVS-M ist nicht unabhängig von der Abtastrate. Für diese Arbeit werden die Originaldaten auf dieselbe Punktmenge reduziert. Wenn pro Feldlinie genau ein Punkt exakt abgespeichert wird, würde die PSNR-HVS-M trotzdem eine hohe Ähnlichkeit zwischen Original und dekomprimierten Feldlinien ergeben. Dies ist Vertretbar, da die Standardabweichung diesen Fall abdeckt und eine hohe Distanz ausgeben wird. Die PSNR-HVS-M soll hauptsächlich Artefakte aufdecken, welche in der Standardabweichung nicht ins Gewicht fallen wie die Ringing Artefakte aus Abschnitt 6.2.6.

## 5 Implementation der Dekompression

Der JHeliovewer lädt eine Folge von komprimierten Simulationen über eine Internetverbindung und muss diese vor der Visualisierung dekomprimieren. Das Herunterladen und die Dekomprimierung sind zeitaufwändige Operationen und führen zu Wartezeiten beim Benutzer. Um die Wartezeit zu verkürzen werden im Ist-Zustand die Simulationen im Voraus asynchron heruntergeladen. Somit sind die Daten bereits im Arbeitsspeicher, bevor die Daten dekomprimiert und visualisiert werden. Die Dekompression wird direkt vor der Visualisierung durchgeführt. Bei einer Animation der Feldliniendaten führt das zu einer bemerkbaren Verzögerung bei jedem Wechsel. Um die Qualität der Animation zu verbessern und die Wartezeit weiter zu verkürzen wurden folgende Massnahmen umgesetzt:

1. Asynchrone Dekompression.
2. Vorladen der Dekomprimierten Feldlinien.
3. Caching von Komprimierten und Dekomprimierten Feldlinien.

### 5.1 Software Architektur



Abbildung 15: Zustandsdiagramm der Feldliniendaten

Die Daten der Feldlinien durchlaufen im JHeliovewer vier Zustände, welche durch vier Klassen abgebildet wurden. Die Klassen sowie die Zustandswechsel sind im Diagramm der Abbildung 15 dargestellt. Die Klasse FileDescriptor repräsentiert eine Simulation von Feldlinien auf dem Server. In diesem Zustand sind die Daten bereit für das Herunterladen. Die folgende Klasse PfssData symbolisiert Feldlinien, welche in den lokalen Arbeitsspeicher geladen wurden. In diesem Zustand sind die Daten komprimiert und nicht bereit für eine Visualisierung. Die Klasse PfssDekompressor ist ein Zwischenzustand und stellt den Wechsel von komprimierten zu unkomprimierten Daten dar. Da der Zustandswechsel aufwändig ist, wird es durch eine eigene Klasse abgebildet. Die letzte Klasse PfssFrame repräsentiert die dekomprimierten Feldlinien. In diesem Zustand sind die Daten bereit für die Darstellung. Die Darstellung wird ebenfalls von der PfssFrame Klasse übernommen.

#### 5.1.1 Vorladen und Caching von Komprimierten und Dekomprimierten Feldlinien

Um die Animation der Feldlinien möglichst Unterbrechungsfrei zu gestalten, werden die komprimierten und dekomprimierten Feldliniendaten vorgeladen und zwischenspeichern. Mit dem Vorladen wird erreicht, dass der Wechsel von der Visualisierung einer Simulation zur Nächsten möglichst ohne Unterbrechung durchgeführt werden kann. Das Caching hilft, wenn der Benutzer einen gespeicherten Zeitpunkt der Animation nochmals darstellen möchte. Die Implementation ist im Diagramm der Abbildung 16 dargestellt.

Die Klasse FrameManager ist zuständig für das Vorladen der dekomprimierten Daten, der PFSSFrame Objekte. Das Caching wird in der FrameCache Klasse umgesetzt. Die Klasse DataCache ist für das Vorladen und Caching der komprimierten Daten zuständig. Die FileDescriptor Objekte repräsentieren eine Feldliniensimulation auf dem Server und der FileDescriptorManager ist zuständig für das Auffinden der Simulationen. Die Klasse FrameManager repräsentiert die Facade der Vorladens- und Caching-Implementation. Sie abstrahiert das Zusammenspiel der verschiedenen Caches und der verschiedenen Zustände der Feldlinien.

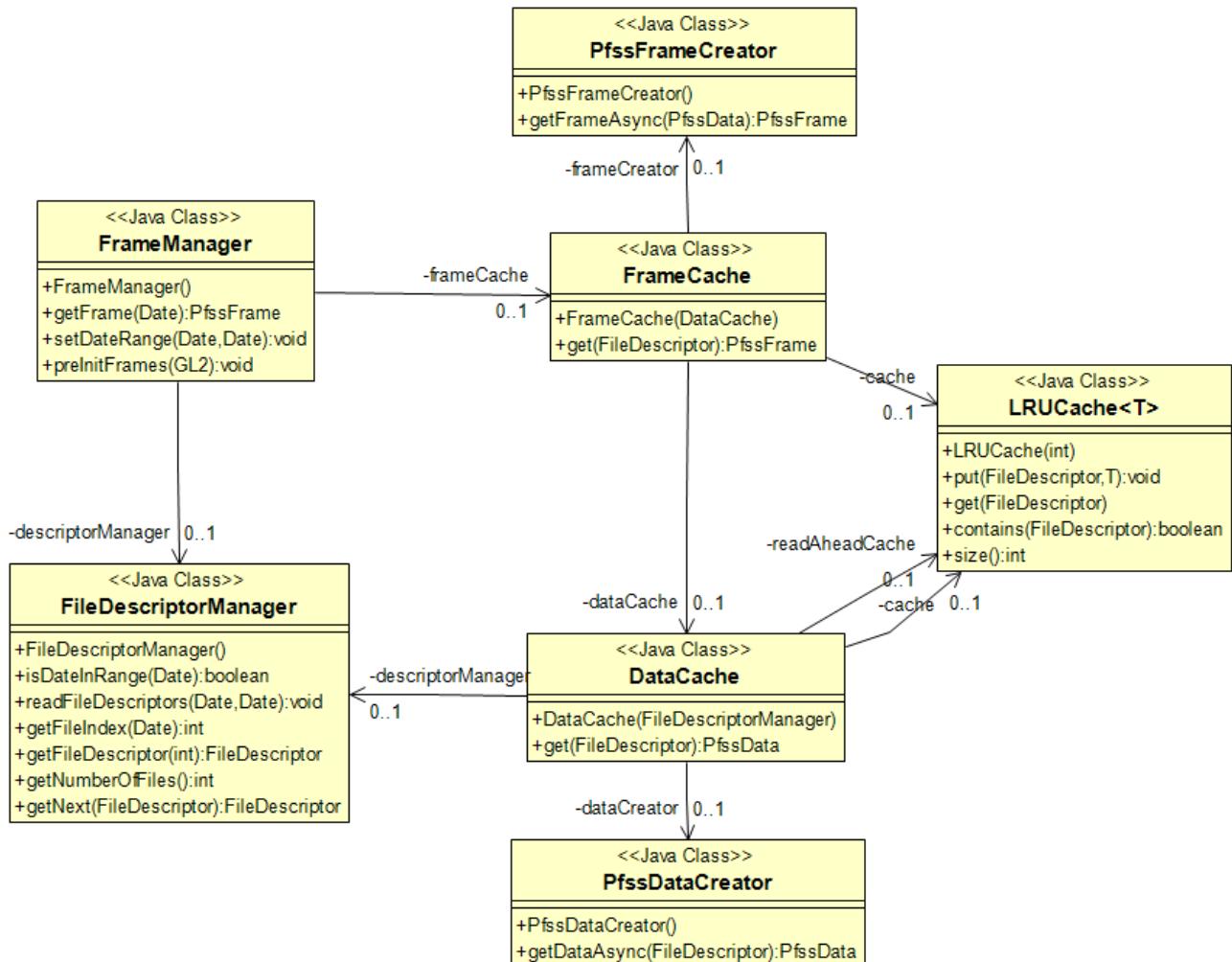


Abbildung 16: Diagramm der Implementation vom Vorladen und Caching

Die Vorladen- und Caching-Implementation der PfssFrame Objekte wurde in zwei Klassen aufgeteilt: Die PfssFrame Objekte müssen vor der Visualisierung Ressourcen der Grafikkarte allozieren. Nach der Visualisierung, müssen die Ressourcen des Objekts freigegeben werden. Der FrameManager ist zuständig die Allozierung anzustossen und die Freigabe sicherzustellen. Im FrameCache sind die Objekte, welche keine Ressourcen alloziert haben und sicher vom Garbage Collector gelöscht werden können.

Die PfssData Objekte besitzen keine zusätzlichen Ressourcen und können vom Garbage Collector verwaltet werden. Das Vorladen kann mit einer weiteren Instanz des LRU-Caches umgesetzt werden und wurde direkt in der DataCache Klasse implementiert.

In dieser Arbeit wurde ein Least-Recently-Used (LRU) Cache Algorithmus verwendet. Der LRU-Cache löscht das am längsten nicht verwendete Objekt, wenn der Cache gefüllt ist. Da der JHeliovewer im allgemeinen Fall sequenziell Objekte verlangt, kann der LRU Cache mit einer First-in-First-Out Queue implementiert werden. Das Objekt, welches am längsten nicht verwendet wurde, ist das Letzte in der Queue. Ein LRU-Cache funktioniert in diesem Anwendungsfäll optimal, wenn die Anzahl Objekte grösser ist, als der Cache. In einem Spezialfall ist der LRU-Algorithmus nicht optimal: Wenn der JHeliovewer zur letzten Simulation der Feldlinien angekommen ist, wird an wrap-around durchgeführt und wieder die erste Simulation verlangt. Wenn der Cache  $n - 1$  von  $n$  Simulation abspeichern kann, löscht der LRU-Algorithmus immer die Simulation, welches als übernächstes abgefragt wird. Das führt zu gleich viele Cache-Misses, als wenn der Cache wesentlich kleiner wäre.

### 5.1.2 Asynchrone Aufrufe mittels Executor Services

Im Diagramm der Abbildung 16 zu sehen ist, wird das Erstellen von PFSSData und PFSSFrame Objekten jeweils von zwei Klassen übernommen werden, den Creators. Sie sind zuständig für das asynchrone Herunterladen und Dekomprimieren der Feldliniendaten. Die asynchrone Ausführung ist mit dem Java Executor Service umgesetzt. Der Executor Service verwaltet und begrenzt die Anzahl an Threads welche die Aufrufe bearbeiten, sodass auch bei hoher Auslastung ein möglichst hoher Durchsatz erreicht wird. Im Ist-Zustand wurden alle asynchronen Aufrufe jeweils in einem eigenen Thread ausgeführt. Bei hoher Auslastung steigt der Verwaltungsaufwand der Threads und bremst das System.

Für die asynchrone Ausführung müssen die Klassen aus Abbildung 15 Threadsafe implementiert sein. Die FileDescriptor Klasse kann als Immutable Threadsafe umgesetzt werden. Für die Klassen PfssData und PfssFrame wird der Ablauf komplexer: Ein PfssData Objekt existiert, sobald das Herunterladen der Daten asynchron gestartet wurde. Die Daten können bei der Verwendung des Objekts komplett, teilweise oder gar nicht heruntergeladen sein. Dasselbe Problem gilt für die PfssFrame Objekte. Hier wird eine Wait-Signal-Logik umgesetzt. Sobald Ressourcen verlangt werden, wie die komprimierten Feldliniendaten des PfssData Objektes, wird ein non-busy waiting durchgeführt. Der aufrufende Thread wird schlafen gelegt, bis der Worker Thread die Daten heruntergeladen hat oder die Dekompression abgeschlossen ist.

## 6 Variantenstudie

In diesem Abschnitt werden die Ergebnisse der Variantenstudie zu den Kompressionsverfahren vorgestellt. Für die Verfahren wurden unterschiedliche Transformationen, Quantisierungen und Kodierungen getestet, welche die Kompressionsrate erhöhen. Im ersten Abschnitt 6.1 wird das Verfahren Adaptiven Subsamplings besprochen. In den folgenden Abschnitten 6.2 und 6.3 folgen die Resultate der Ansätze DCT und Prädiktive Kodierung.

### 6.1 Kompressionsverfahren: Adaptives Subsampling

Im Ist-Zustand führt der JHeliovewriter nach der Dekompression ein Adaptives Subsampling durch. Dieses Verfahren führt das Adaptive Subsampling vor der Datenübertragung durch und Kodiert die Daten mit Rar anstatt mit Gzip. Eine genauere Beschreibung der Kompression ist im Abschnitt 3.2 zu finden.

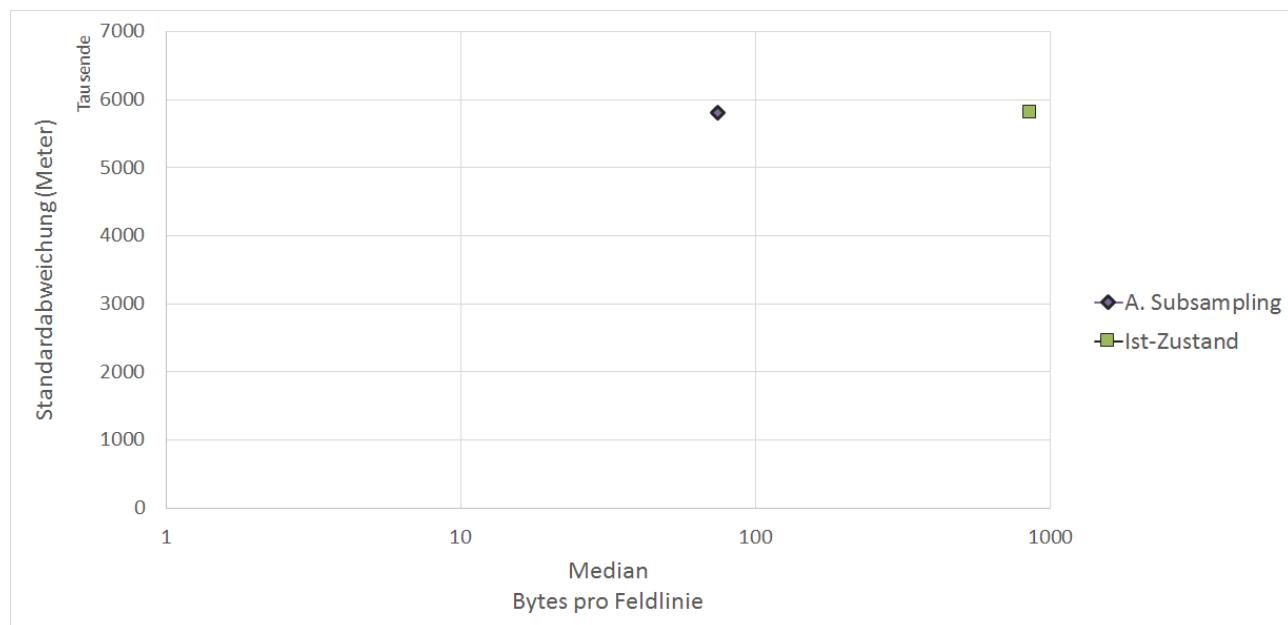


Abbildung 17: Vergleich des Lösungsansatzes: Adaptives Subsampling zur Ist-Kompression.

Wie im Diagramm der Abbildung 17 erkennbar ist, braucht dieser Lösungsansatz deutlich weniger Speicher als die Ist-Kompression zur selben Genauigkeit. Der Lösungsansatz des Adaptiven Subsamplings erreicht eine Kompressionsrate von 11.6 gegenüber dem Ist-Zustand, indem nur Daten übertragen werden, welche in der Visualisierung darstellen werden.

Das Diagramm Abbildung 18 zeigt die Artefakte der Kompression. Sie sind identisch mit den Artefakten der Visualisierung im Ist-Zustand. Die Artefakte sind in der JHeliovewriter Visualisierung erst bei höheren Zoom-Stufen erkennbar.

Der JHeliovewriter muss in der Lage sein 1000 komprimierte Simulationen im Arbeitsspeicher abzulegen. Mit dieser Kompression werden durchschnittlich 85 Megabyte an Arbeitsspeicher benötigt. Wenn von einer 10 Megabit Internetverbindung ausgegangen wird, werden für das Herunterladen von 1000 Simulationen 70 Sekunden benötigt. Der Ist Zustand benötigt bei denselben Bedingungen 790 Sekunden (13 Minuten) für die Übertragung. Mit dieser Kompression können etwa 14 komprimierte Simulationen pro Sekunde übertragen werden.

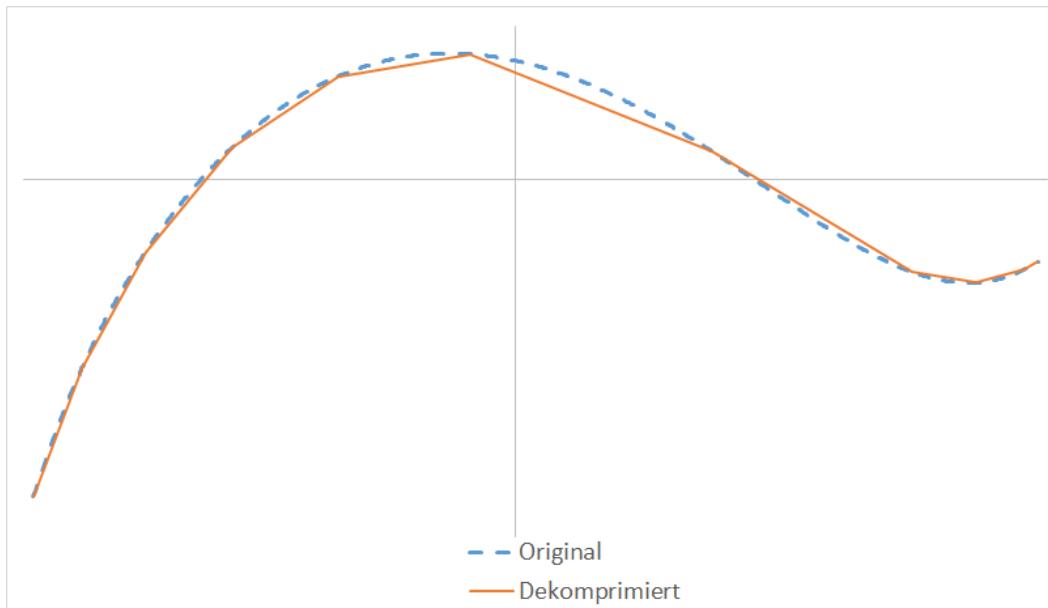


Abbildung 18: Artefakte des Verfahrens Adaptives Subsampling.

Ein Vorteil dieses Verfahrens ist die Laufzeit Dekompression: Da keine rechenaufwändige Transformationen verwendet werden 19 Millisekunden für eine Dekompression (Siehe Abschnitt 8.3) benötigt. Mit einem Thread ist die Testmaschine in der Lage 50 Simulationen pro Sekunde zu dekomprimieren.

In Zukunft ist es möglich, dass im JHeliovewer deutlich mehr Punkte dargestellt werden sollen. In diesem Fall müssen entweder mehr Daten übertragen werden oder der JHeliovewer mit einer Interpolation erweitert werden.

## 6.2 Kompressionsverfahren: Diskrete Kosinus Transformation

In diesem Abschnitt wird das Kompressionsverfahren mittels Diskreter Kosinus Transformation behandelt. Es wurden verschiedene Transformationen getestet, welche die Approximation mittels Kosinus Funktionen verbessern. Die Ableitung der Feldlinien dämpft die Kompressionsartefakte und ist massgebend für die Kompressionsrate verantwortlich.

Um die Feldlinien optimal mit der DCT zu approximieren, müssen Ringing Artefakte, behandelt werden. Das Auftreten der Artefakte wird im Abschnitt 6.2.6 und die Behandlung im Abschnitt 6.2.7 besprochen.

### 6.2.1 Variante: DCT

Diese Variante verwendet die Diskrete Kosinus Transformation. Es wird erforscht welche Kompressionsrate ohne zusätzliche Transformationen möglich ist.

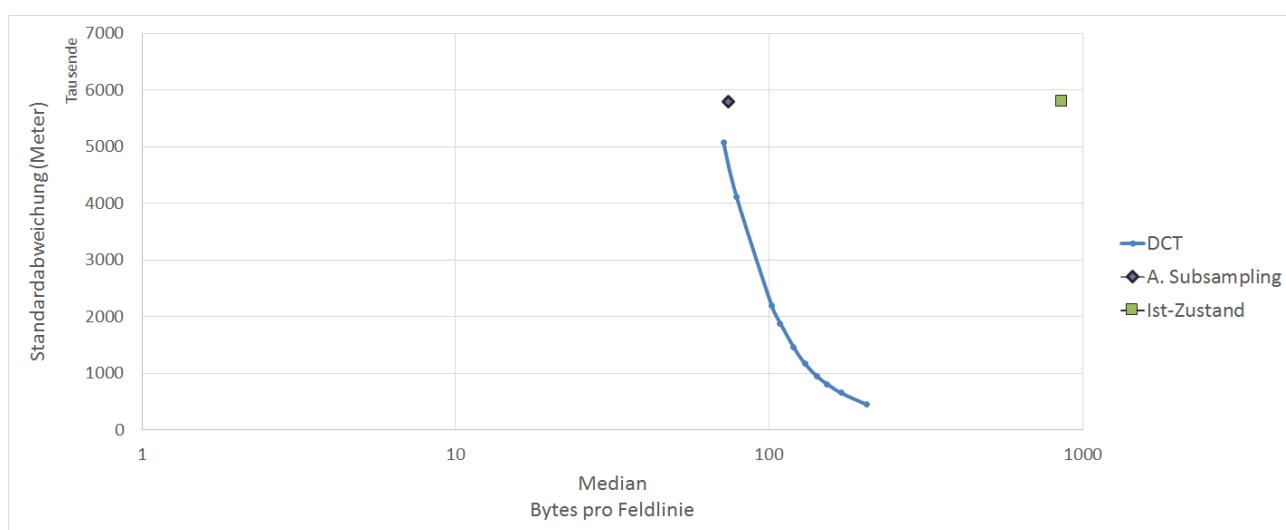


Abbildung 19: Vergleich der DCT Kompression mit dem Verfahren des Adaptiven Subsamplings

Die Abbildung 19 zeigt den Vergleich der DCT Kompression mit dem Verfahren des Adaptiven Subsamplings (siehe 6.1). Die Standardabweichung steigt unerwartet schnell an. Der Grund kann im Diagramm der Abbildung 20 entnommen werden.

Der Start- und Endpunkt der Feldlinie kann nicht dargestellt werden. Dies ist ein typisches Problem der DCT: Ein Inputsignal wird in der DCT konzeptionell wiederholt. Bei der implementierten Kosinus Transformation (siehe Abschnitt 3.3.3) wird das Signal jeweils in umgekehrter Reihenfolge wiederholt. Bei der Beispieldlinie aus Abbildung 20 führt die Wiederholung zu einer Diskontinuität im Signal. Die Diskontinuität wird in der DCT mit hochfrequenten Anteilen abgebildet, welche von der Quantisierung gelöscht werden. Das Ergebnis ist eine Verschiebung an den Stellen, wo die Diskontinuität auftritt. In diesem Fall ist es jeweils am Anfang und am Ende der Feldlinie.

Das Problem kann entweder durch eine andere Darstellung der Feldlinie oder durch zusätzliche Punkte am Anfang und am Ende der Linie gelöst werden. Die Variante, welche die Feldlinie mit Punkten erweitert, wird im Abschnitt 6.2.5 behandelt. Eine andere Darstellung der Feldlinie wird in den folgenden Abschnitten behandelt.

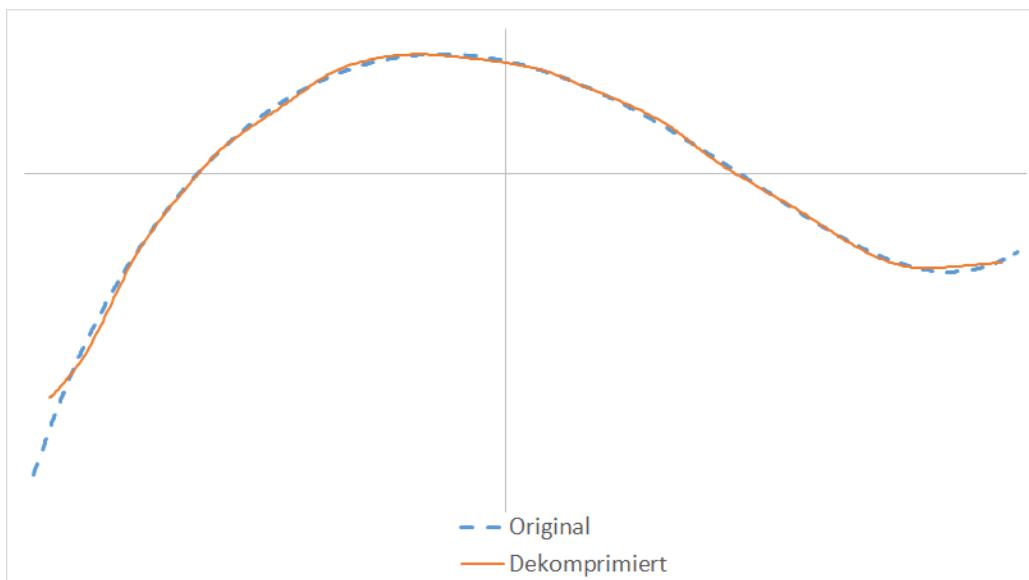


Abbildung 20: Artefakte der DCT Dekompression anhand Beispieldaten

### 6.2.2 Variante: Ableitung+DCT

Vor der Diskreten Kosinus Transformation werden die Feldlinien abgeleitet. Durch diese Darstellung werden die Artefakte aus der Abbildung 20 behoben.

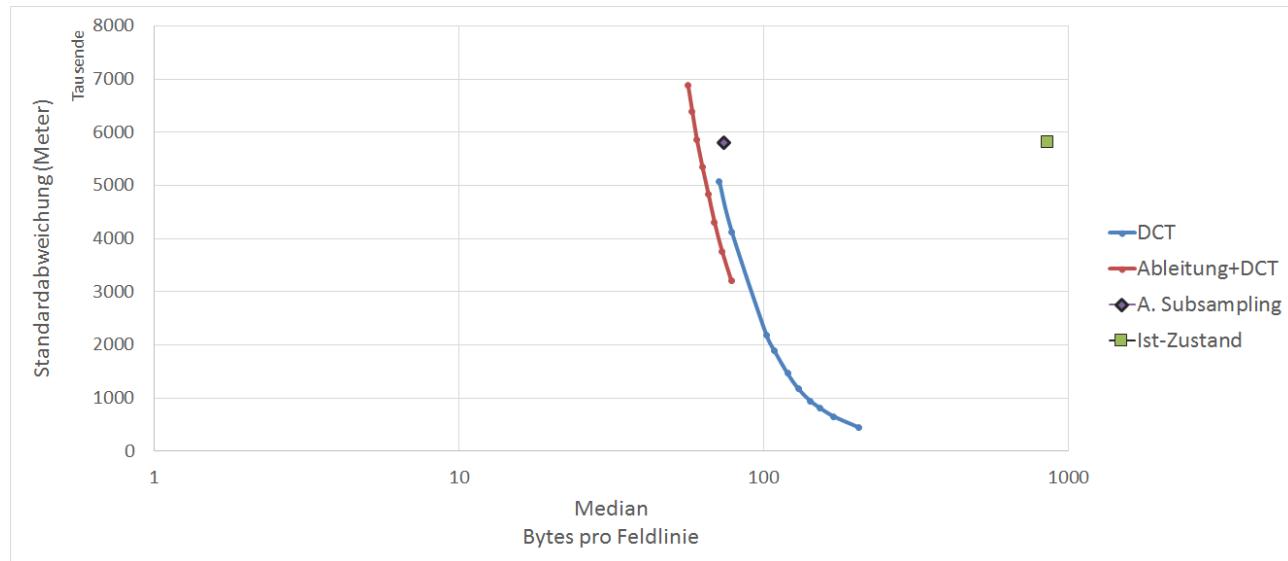


Abbildung 21: Vergleich der DCT Kompression der Ableitung mit der DCT Kompression

Das Diagramm der Abbildung 21 zeigt, dass die abgeleiteten Feldlinien besser approximiert werden können als die vorhergehende Variante. Bei einer vergleichbaren Genauigkeit wie der Ist-Zustand erreicht diese Variante eine Kompressionsrate von 14.3. Eine Darstellung der Artefakte ist im Diagramm der Abbildung 22 zu finden. Die Artefakte der Kompression äussern sich als Dämpfungen. Die Artefakte sind jedoch für das menschliche Auge nicht sichtbar. Die Dämpfung ist erst zu erkennen, wenn das Original zur Verfügung steht.

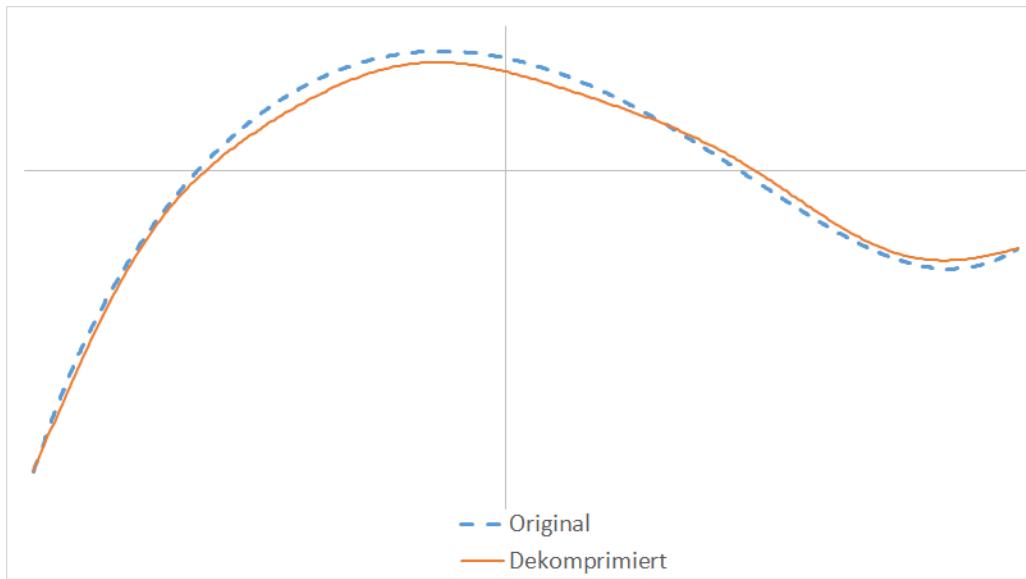


Abbildung 22: Artefakte der DCT Kompression der Ableitung

### 6.2.3 Variante: PCA+Ableitung+DCT

Die Feldlinien liegen im Allgemeinen in einer Ebene im dreidimensionalen Raum. Durch eine Principal Component Analysis (PCA)[24] können die Feldlinien in ein lokales Koordinatensystem transformiert werden, in welchem ein Kanal den meisten Fällen nicht verwendet wird.

Für die Rücktransformation ins Sonnen-Koordinatensystem werden pro Feldlinie zusätzlich sech Parameter für die neuen Koordinatenachsen und drei Parameter für die Verschiebung abgespeichert.

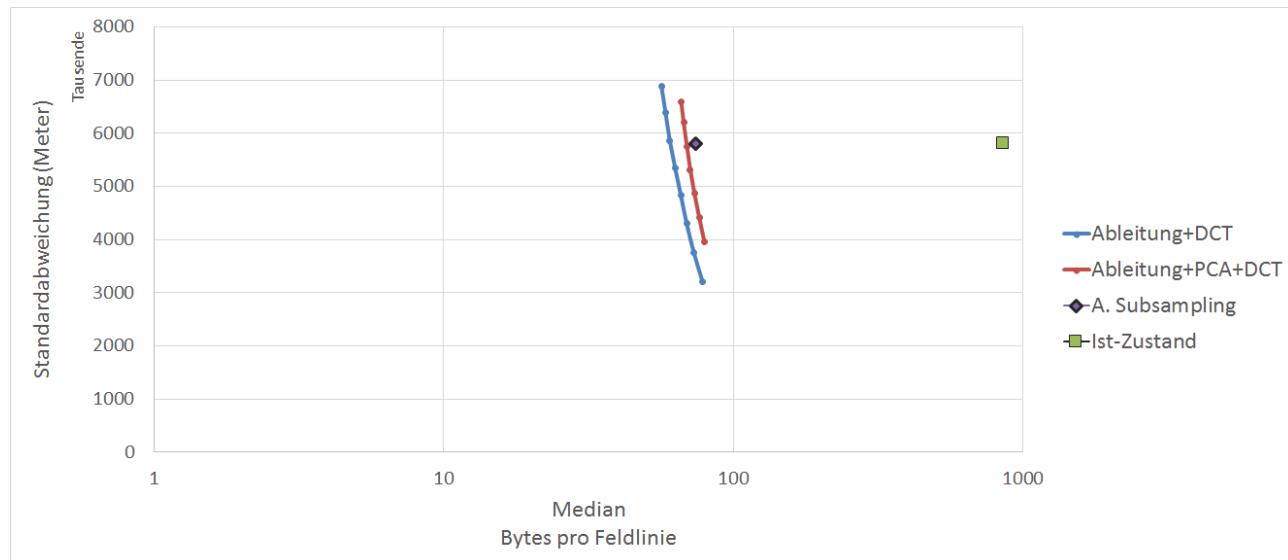


Abbildung 23: Vergleich der PCA DCT Kompression der Ableitung mit der DCT Kompression der Ableitung

Im Diagramm der Abbildung 23 sind die Resultate der Messung dargestellt. Die PCA konnte keine Verbesserung erbringen. Die zusätzlichen Parameter verbrauchen mehr Speicherplatz als durch die PCA gewonnen werden kann.

### 6.2.4 Variante: Ableitung+DCT+Byte Kodierung

Die quantisierten Koeffizienten können im Allgemeinen mit 8 Bit Genauigkeit dargestellt werden. Nur wenige Ausnahmen benötigen die 16 Bit Genauigkeit, mit der sie abgespeichert werden. Der Grossteil der DCT-Koeffizienten wird ebenfalls auf 0 Quantisiert. Mit einer Byte-Kodierung werden diese Eigenschaften ausgenutzt. Die Byte Kodierung ist im Abschnitt 3.3.5 beschrieben.

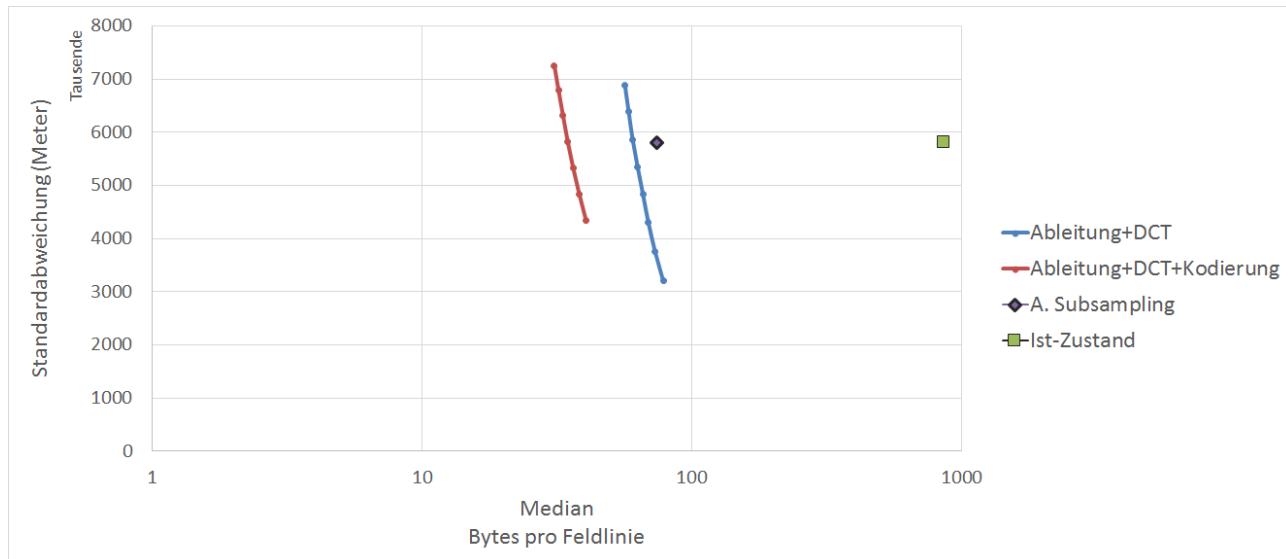


Abbildung 24: Vergleich der Kompression mit und ohne Byte-Kodierung

Das Diagramm der Abbildung 24 zeigt den Einfluss der Byte-Kodierung auf die Kompressionsrate. Es bewirkt eine deutliche Verbesserung gegenüber der vorhergehenden Variante, ohne die Qualität der Kompression negativ zu beeinflussen. Bei einer vergleichbaren Genauigkeit wie die Ist-Kompression weist diese Variante eine Kompressionsrate von 24.5 auf.

### 6.2.5 Variante: Randbehandlung+DCT+Byte Kodierung

Bei dieser Variante wird mit zusätzlichen Punkten am Anfang und am Ende der Feldlinie die Artefakte aus Abschnitt 6.2.1 behoben. Die Zusätzlichen Punkte lassen die Feldlinie abflachen. Es wird erforscht, ob diese Darstellung der Feldlinien eine bessere Kompressionsrate zur ähnlicher Abweichung erlaubt.

Das Diagramm der Abbildung 25 zeigt den Vergleich der Variante mit Randbehandlung und der vorgehenden Variante, welche die Feldlinie ableitet. Die zusätzlichen Punkte erlauben eine höhere Kompressionsrate zu einer ähnlichen Abweichung.

Diese Variante führt Artefakte ein, welche in der Standardabweichung nicht ins Gewicht fallen. Im folgenden Abschnitt 6.2.6 werden diese Artefakte besprochen.

### 6.2.6 Ringing Artefakte

Obwohl die Variante aus Abschnitt 6.2.5 eine vergleichbare Standardabweichung aufweist, wie die Ist-Kompression, sind auf der JHelioviewer Visualisierung deutliche Artefakte zu sehen. Die Abbildung 26 vergleicht die originalen mit dekomprimierten Feldlinien. Die Artefakte äußern sich als Oszillationen in den dekomprimierten Feldlinien.

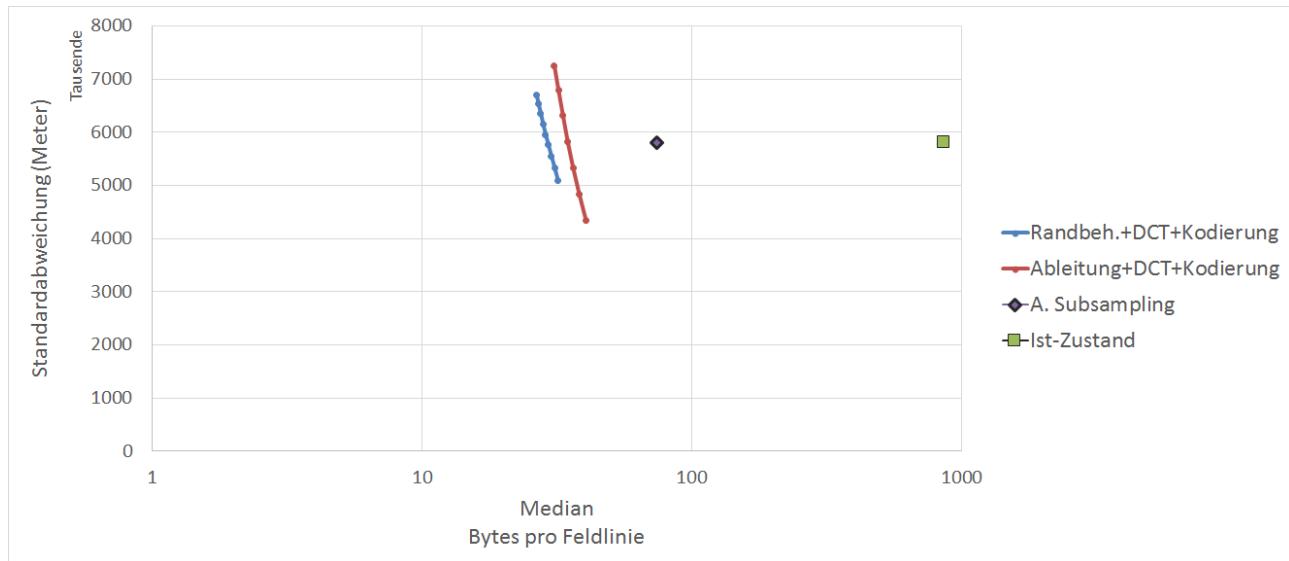


Abbildung 25: Vergleich des Einflusses der Randbehandlung

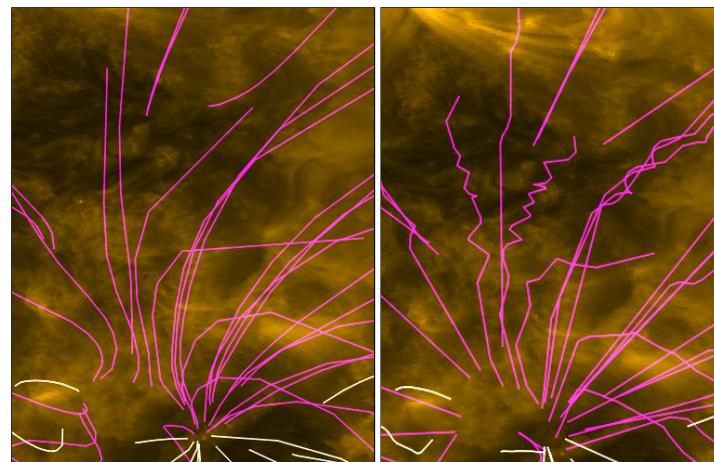


Abbildung 26: Artefakte der Kompression. Links sind die originalen Feldlinien, rechts die Dekomprimierten.

In diesem Fall scheint die Standardabweichung als Fehlermass zu versagen: Da die Oszillationen nahe an der originalen Feldlinie liegen, bleiben die Abweichung klein. Jedoch sind die Artefakte für das menschliche Auge inakzeptabel.

Interessant ist, dass die abgeleiteten Feldlinien aus Abschnitt 6.2.4 ähnliche Artefakte aufweisen, jedoch sind sie weniger ausgeprägt. Die Ableitung scheint die Artefakte zu dämpfen. Die Abbildung 27 zeigt die Artefakte der abgeleiteten Feldlinien.

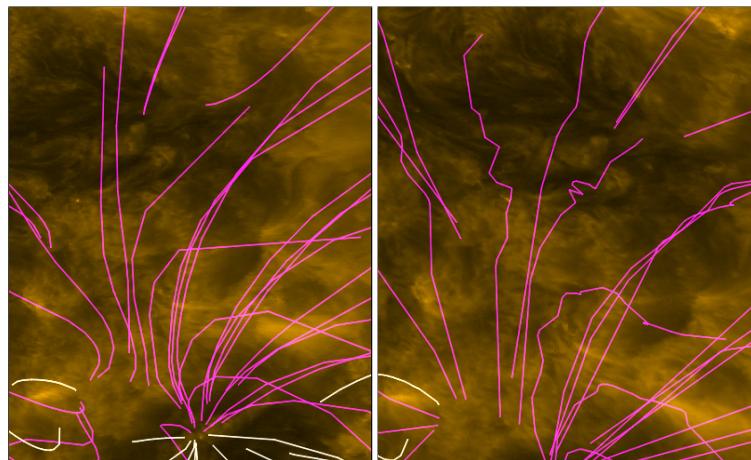


Abbildung 27: Artefakte der Kompression, links sind die originalen Feldlinien, rechts die Dekomprimierten der Variante 6.2.4.

Die oszillierenden Artefakte sind typisch für eine Datenkompression mit einer Diskreten Kosinus Transformation und sind als Ringing Artefakte [19] bekannt. Sie treten ebenfalls bei JPEG/JFIF oder MP3 Kompressionen auf: Abrupte Steigungen im Inputsignal werden in der DCT durch hochfrequente Anteile dargestellt. Durch die Quantisierung der hochfrequenten Anteile werden oszillierende Artefakte eingefügt.

Die Feldlinien, welche am stärksten von den Artefakten betroffen sind, sind die "Weltall zur Sonne" oder "Sonne ins Weltall" Feldlinien. Sie verhalten sich nicht wie harmonische Halbwellen sondern steigen oft monoton, mit teils abrupten Richtungswechseln in der Nähe der Sonnenoberfläche. Die Abbildung 28 zeigt ein Beispiel solcher Feldlinien. Die abrupten Wechsel führen zu abrupten Steigungen in den einzelnen Kanälen.

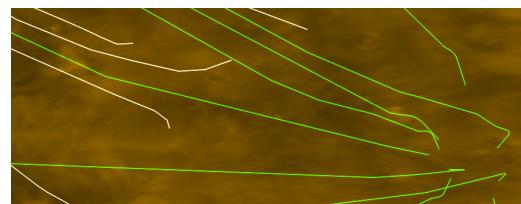


Abbildung 28: Abrupte Steigungen bei Feldlinien, welche von der Sonne ins Weltall führen.

### 6.2.7 Behandlung der Ringing Artefakte

Um eine optimale Kompression der Feldlinien zu erreichen, müssen die Ringing Artefakte behandelt werden. Im Abschnitt 6.2.6 wurde erwähnt, dass nicht alle Varianten gleich ausgeprägte Artefakte verursachen. Es wurde ebenfalls besprochen, dass die Feldlinien, welche von der Sonnenoberfläche zu Oberfläche führen, weniger ausgeprägte Artefakte mit sich bringen. In diesem Abschnitt wird erforscht, welche Variante die "Sonne zu Sonne" und welche die "Weltall zur Sonne" oder "Sonne ins Weltall" Feldlinien optimal approximieren kann, ohne zusätzliche Ringing Artefakte hinzuzufügen. Für die Messung der Artefakte wird die PSNR-HVS-M Metrik aus Abschnitt 4.3 verwendet.

Für den Test werden insgesamt vier Varianten verglichen. Die Variante der abgeleiteten Feldlinien aus Abschnitt 6.2.4 und die Variante der Randbehandlung aus Abschnitt 6.2.5. Die Varianten werden jeweils mit und ohne einer PCA gemessen. Die PCA wird hier nochmals überprüft, da sie die Ringing Artefakte auf einen Kanal beschränken kann.

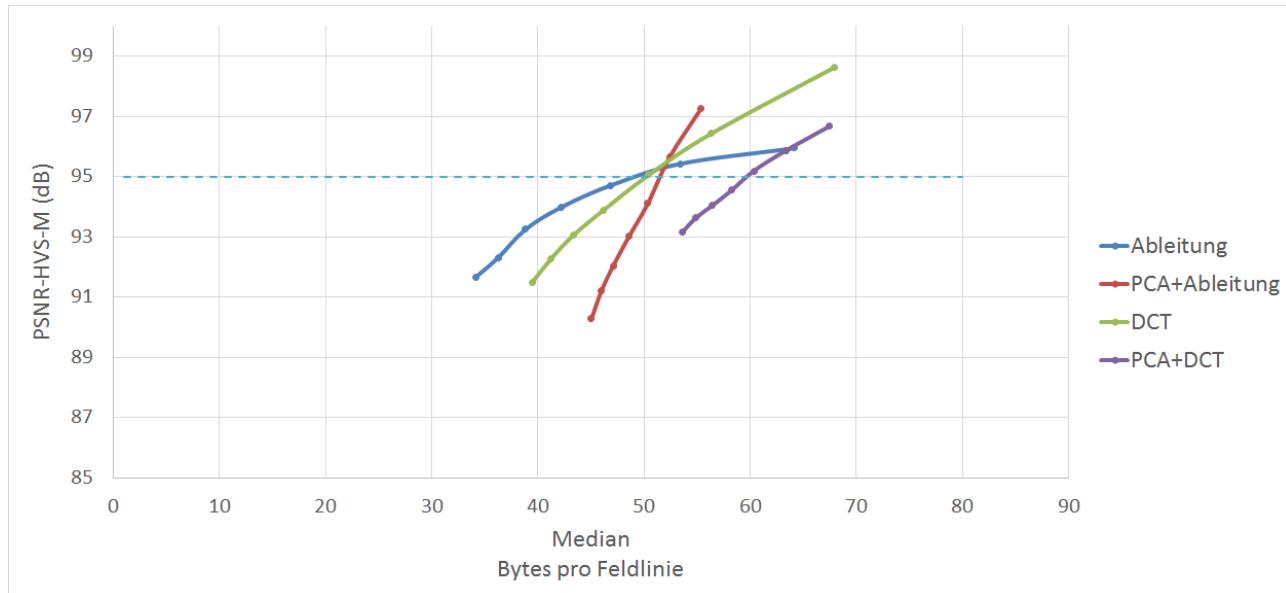


Abbildung 29: Approximation der Feldlinien "Sonnenoberfläche zu Sonnenoberfläche". Je höher die PSNR-HVS-M, desto besser ist die Approximation.

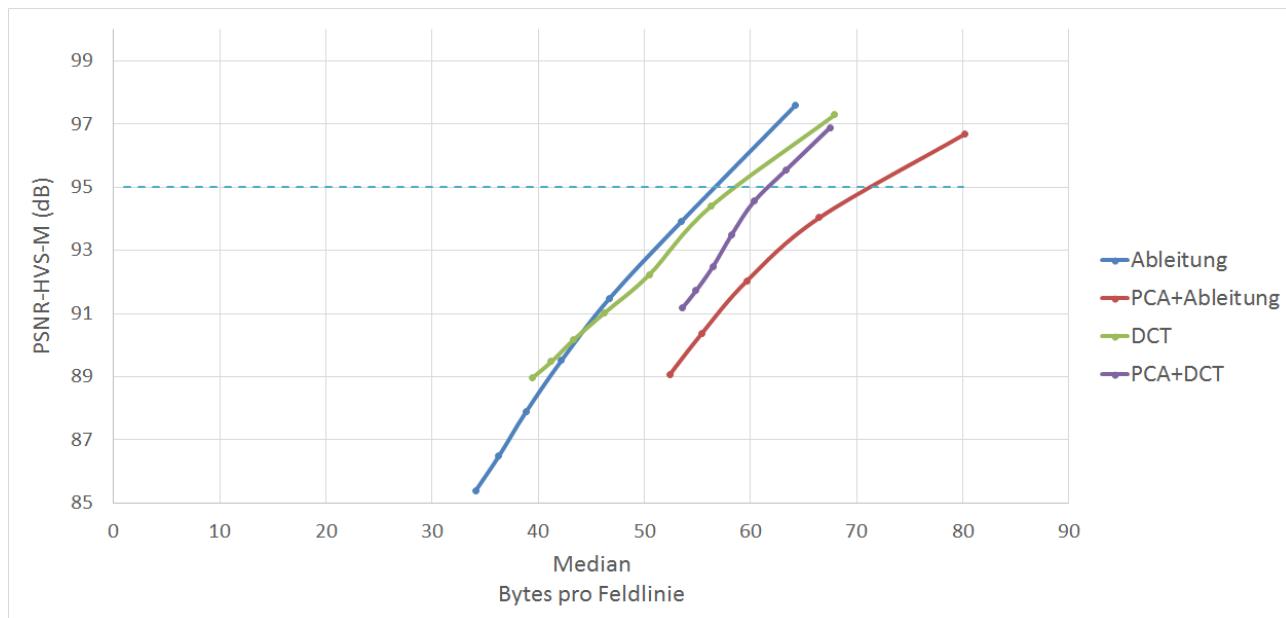


Abbildung 30: Approximation der der Feldlinien "Sonnenoberfläche ins Weltall" oder "Weltall zur Sonnenoberfläche". Je höher die PSNR-HVS-M, desto besser ist die Approximation.

Das Diagramm der Abbildung 29 zeigt die Resultate der unterschiedlichen Varianten bei den Feldlinien "Sonnenoberfläche zur Sonnenoberfläche". Bei einer PSNR-HVS-M von über 95 dB sind die Artefakte genügend schwach ausgeprägt, sodass das menschliche Auge sie nicht mehr erkennen kann. Interessant ist, dass drei Varianten ähnlich viel Speicherplatz benötigen, für eine artefaktfreie Approximation. Bei stärkerer Quantisierung treten bei den abgeleiteten Feldlinien deutlich weniger Artefakte auf. Dies deckt sich mit der Beobachtung

aus dem Abschnitt 6.2.6.

Ebenfalls interessant ist die Variante der PCA Transformation zusammen mit der Ableitung: Diese benötigt für eine beinahe artefaktfreie Approximation am wenigsten Speicherplatz, führt aber bei stärkerer Quantisierung viele Artefakte ein.

Das Diagramm der Abbildung 30 zeigt, wie gut die Varianten die Feldlinien "Sonne ins Weltall" und "Weltall zur Sonne" approximieren können. Bei diesen Typen von Feldlinien dämpft die Ableitung ebenfalls die Artefakte. Jedoch weniger deutlich als bei den "Sonne zur Sonne" Feldlinien. Die PCA konnte auch bei diesen Typen von Feldlinien keinen messbaren Vorteil erbringen. Die zusätzlichen Parameter der PCA verbrauchen mehr Speicherplatz als durch die Transformation gewonnen werden.

Die DCT Variante aus Abschnitt 6.2.5 fällt ab einer PSNR-HVS-M von 90dB weniger schnell ab als die abgeleiteten Feldlinien. Bei diesem PSNR-HVS-M Wert sind aber die Artefakte bereits zu deutlich und nicht mehr akzeptabel. Aufgrund dieser Resultate wurde die Variante der abgeleiteten Feldlinien von Abschnitt 6.2.4 ausgewählt.

### 6.2.8 Abschliessende Variante

Für den abschliessenden Tests wurde die Variante der abgeleiteten Feldlinien aus Abschnitt 6.2.4 ausgewählt. Durch die Ableitung können die Ringing Artefakte gedämpft werden. Die Quantisierung wurde angepasst, sodass die Feldlinien vom Typ "Sonne zu Sonne" stärker quantisiert werden, als die anderen Feldlinien. Durch diese Massnahme wird eine hohe Kompressionsrate erreicht ohne starke Ringing Artefakte mit sich zu ziehen.

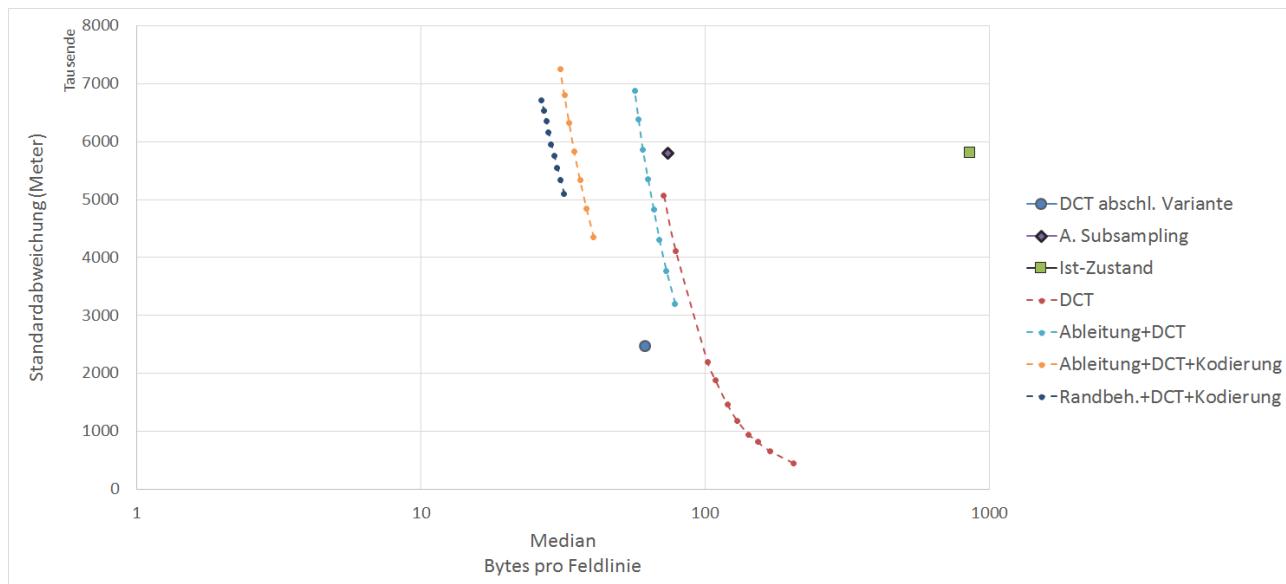


Abbildung 31: Standardabweichung der abschliessenden DCT Variante.

Das Diagramm der Abbildung 31 zeigt die Standardabweichung der abschliessenden DCT Variante. Sie erreicht eine höhere Kompression als das Verfahren des Adaptiven Subsamplings und kann die Feldlinie zu einer tieferen Standardabweichung approximieren. Es wurde eine durchschnittliche Kompressionsrate 14.1 erreicht, obwohl eine höhere Anzahl an Punkten übertragen wird. Die Problematik dieses Verfahrens liegt darin, dass die DCT Kompression Ringing Artefakte hinzufügt.

Durch die unterschiedliche Quantisierungen konnten die Ringing Artefakte in Grenzen gehalten werden und erreichen einen PSNR-HVS-M Wert von 94.0. Die "Sonne zu Sonne" Feldlinien können deutlich besser approximiert werden als die anderen Feldlinien. Würde die Simulation nur aus diesen Feldlinien bestehen, könnte eine Kompressionsrate von 15 – 18 erreicht werden zu einem ähnlichen PSNR-HVS-M Wert. Die Feldlinien

"Sonne ins Weltall und "Weltall zur Sonne" sind schwieriger mit einer DCT artefaktfrei zu approximieren. Durch das Zoom Feature vom JHeliovewer ist der Benutzer in der Lage die Ringing Artefakte zu finden, sobald sie existieren. Das Linke Bild der Abbildung 32 zeigt die Artefakte der Kompression. Die Artefakte sind aus der Simulation, welche die markantesten Ringing Artefakte aufweist.

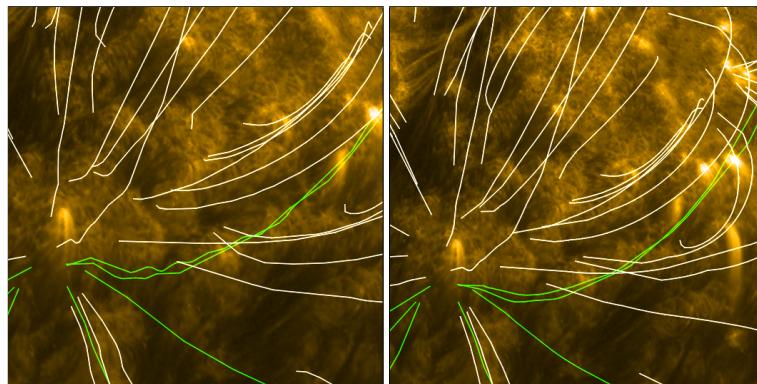


Abbildung 32: Die am stärksten ausgeprägten Artefakte der abschliessenden Variante. Links ohne Glättung, rechts mit Glättung

Durch eine Kurvenglättung kann der JHeliovewer die Artefakte verschleieren. Die komprimierten Daten enthalten mehr Punkte, als die JHeliovewer Visualisierung darstellen kann. Die zusätzlichen Punkte werden zur Glättung der Feldlinie verwendet. Den Effekt der Glättung ist in der Abbildung 32 abgebildet.

Im Abschnitt 3.3 wurde erwähnt, dass Reduktion von Ringing Artefakte ein aktives Forschungsfeld der Bildverarbeitung ist. Es existieren Post-Processing Filter, welche Ringing Artefakte von dekomprimierten Bildern vermindern. Eine Möglichkeit die Kompression zu verbessern ist es, einen Post-Processing Filter für wissenschaftliche Daten zu entwickeln. Eine weitere Möglichkeit ist die Diskrete Kosinus Transformation durch eine Wavelet Transformation zu ersetzen. Diese ist weniger anfällig auf Ringing Artefakte und hat das Potential eine ähnlich gute Kompression zu erreichen.

Beim Caching von 1000 Simulationen benötigt dieser Ansatz 70 Megabyte Arbeitsspeicher, etwa 15 Megabyte weniger als das Verfahren des Adaptiven Subsamplings. Wenn von derselben 10 Megabit Internenetverbindung ausgegangen wird, werden 56 Sekunden benötigt um 1000 Simulationen herunterzuladen. Pro Sekunde werden 17 anstatt 14 Simulationen übertragen. Die Simulationen können mit dieser Kompression zur gegebenen Internetverbindung on-the-fly heruntergeladen werden.

Die höhere Kompressionsrate führt zu einer höheren Komplexität der Dekompression. Die Laufzeit der Dekompression hängt im Wesentlichen von der Implementation der inversen DCT ab. Eine naive Implementation braucht für eine Dekompression etwa drei Sekunden (siehe Abschnitt 8.3. Die grösste Zeit wird in der Berechnung der Kosinus-Werte verbraucht. Bei der DCT Implementation in dieser Arbeit werden die Kosinus-Werte über SoftReferences gecached. Der Cache passt sich somit an den Arbeitsspeicher an. Wie schnell die Dekompression ist, hängt im Wesentlichen vom verfügbaren Arbeitsspeicher ab. Im besten Fall ergibt das eine Laufzeit von 65 Millisekunden und im schlechtesten Fall 350. Mit einer Java Implementation der Fast-Cosine-Transform wird eine Laufzeit von 100 bis 135 Millisekunden erreicht. Im besten Fall ist die Implementation für diese Arbeit schneller, da es die Eigenschaft der Quantisierung ausnutzen kann: Es werden pro Feldlinie maximal 50 Koeffizienten übertragen. Die Komplexität kann dadurch auf  $O(n * m)$  beschränkt werden, wobei  $m$  die Anzahl der Koeffizienten ist. Die FDCT weist eine Komplexität von  $O(n \log(n))$  auf. Die Implementation verliert Laufzeit gegenüber der FDCT, wenn die Kosinus-Koeffizienten neu berechnet werden. Im Durchschnittsfall ist die FDCT Implementation schneller, was einen Durchsatz von 8 Simulationen pro Sekunde pro Thread führt.

## 6.3 Kompressionsverfahren: Prädiktive Kodierung

In diesem Abschnitt wird das Kompressionsverfahren der Prädiktiven Kodierung besprochen. Es wurde der Einfluss von verschiedenen Prädiktoren getestet. Die Schwierigkeit dieses Verfahrens liegt in der Quantisierung des Vorhersagefehlers. Die Quantisierung der Fehler von einfacher Prädiktoren führt zu hohen Abweichungen in der Dekompression. Der Rekursive Lineare Prädiktor unter Abschnitt 6.3.3 löst das Problem.

### 6.3.1 Variante: einfaches Subsampling

Für die erste Variante wird das Subsampling des DCT-Kompressionsverfahren verwendet. Es reduziert die Punktmenge auf die Anzahl des Ist-Zustands. Die Feldlinien werden mit einer PCA in ein lokales Koordinatensystem transformiert. Im lokalen System können die Koordinaten mit 16 Bit Genauigkeit dargestellt werden. 16 Bit im Koordinatensystem der Sonne reichen nicht aus und führen zu einer grösseren Abweichung als der Ist-Zustand.

In dieser Variante wird der Einfluss von vier Prädiktoren getestet ( $x$  sind die Bekannten Punkte und  $y$  die Vorhersage):

- Konstanter Prädiktor: Nimmt an, dass der nächste Wert im Kanal gleich dem letzten Wert ist ( $x = y$ ).
- Linearer Prädiktor: Nimmt an, dass die Steigung die Steigung zum nächsten Wert konstant bleibt ( $x_1 + (-x_2 + x_1) = y$ ).
- Linearer Prediktor mit Moving Average: Nimmt die durchschnittliche Steigung der letzten Werte.
- Adaptiver Linearer Prädiktor mit Moving Average: Berücksichtigt den Fehler der letzten Vorhersage.

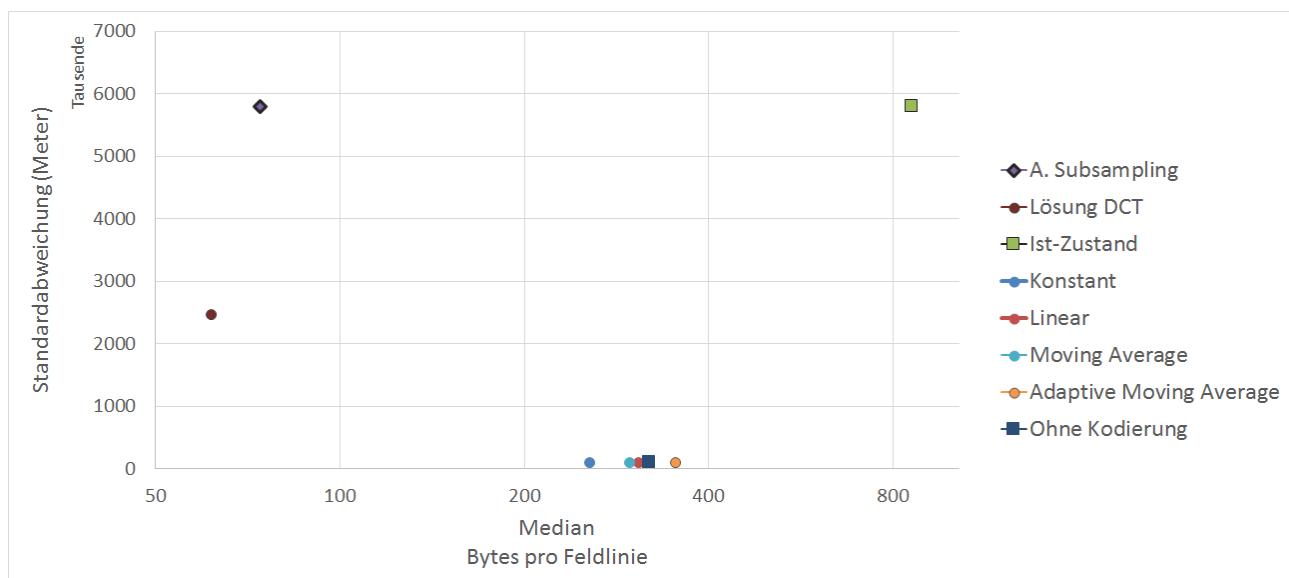


Abbildung 33: Kompressionsraten der vier Prediktoren im Vergleich zum Ist-Zustand.

Im Diagramm der Abbildung 33 sind die Kompressionsraten der jeweiligen Prädiktoren dargestellt. Ein Diagramm mit der PSNR-HVS-M wurde nicht erstellt. Sie ist für alle Prädiktoren gleich und liegt bei 140.7 dB. Unerwartet ist, dass der Konstante Prädiktor mit 255 Bytes pro Feldlinie die beste Kompression erreichte, obwohl die Daten nicht zuverlässig vorhersagen kann. Im Vergleich mit dem Moving Average Prädiktor sind die Fehler der Vorhersagen bis zu 5 Mal grösser, verbrauchen aber 40 Bytes weniger um eine Feldlinie abzuspeichern. Eine Erklärung ist, dass die RAR Kodierung sich wiederholende Muster findet.

Eine mögliche Optimierung ist die Adaptive Byte Kodierung der DCT-Variante, beschrieben im Abschnitt 3.3.5. Das Diagramm der Abbildung 34 zeigt die Resultate mit der Byte Kodierung. Der Konstante Prädiktor verbraucht mit der Adaptiven Kodierung mehr Speicherplatz. Die Kompressionsrate der anderen Prädiktoren wird durch die Adaptive Kodierung deutlich verbessert. Der Lineare Prädiktor erreicht mit 214 Bytes pro Feldlinie die beste Kompression. Es bestätigt die Vermutung, dass der Konstante Prädiktor keine zuverlässige Vorhersage erreichte und die Kompressionsrate auf die RAR Kodierung zurückzuführen ist. Der Lineare Prädiktor kann eine Feldlinie mit etwa 214 Bytes darstellen, was eine Kompressionsrate von 4 ergibt.

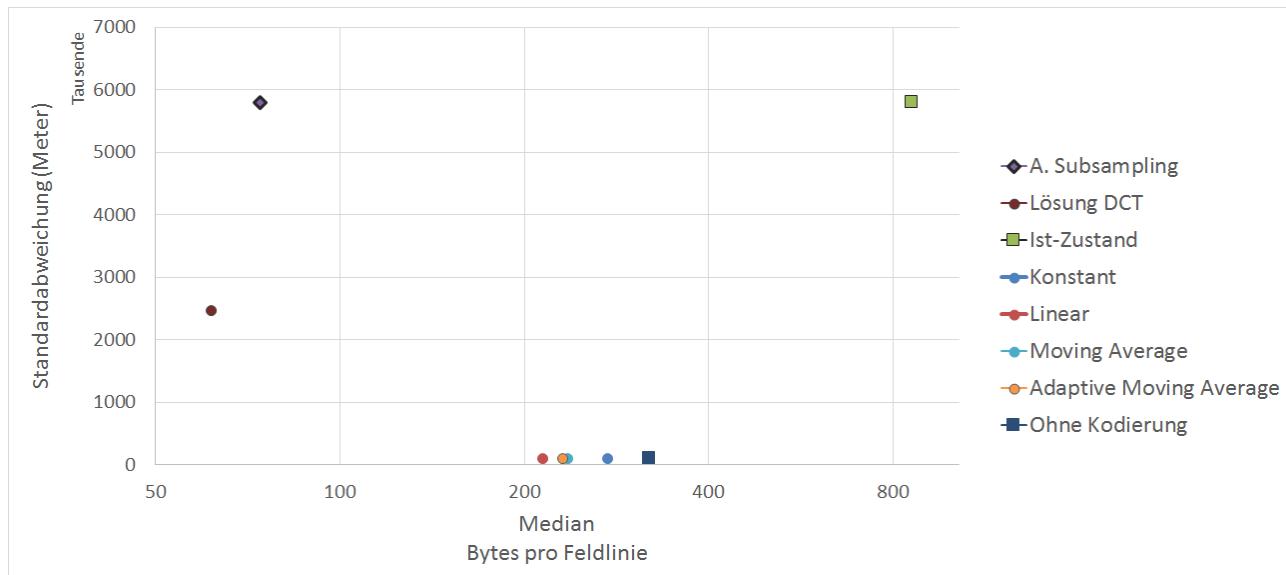


Abbildung 34: Kompressionsraten der Prediktoren mit Adaptiver Byte Kodierung.

### 6.3.2 Variante: Adaptives Subsampling

Um die Kompressionsrate der Prädiktiven Kodierung in dieselbe Größenordnung zu gelangen wie die Kompressionen des Adaptiven Subsampling oder der DCT müssen mehr Informationen gelöscht werden. Deshalb wird das Adaptive Subsampling eingesetzt, es werden andere Parameter verwendet und im Schnitt 50% mehr Daten übertragen als in der Kompression des Adaptiven Subsampling.

Das Diagramm der Abbildung 35 zeigt die Kompressionsraten. Die Resultate liegen dicht beieinander und der Abstand zwischen Kodierung und der Kompression, welche ohne Kodierung erreicht wird wurde drastisch vermindert. Die Kodierungen erreichen eine PSNR-HVS-M von 139,2. Das Angle Subsampling verändert die Eigenschaften der Daten. Die Diagramme der Abbildung 36 visualisieren die Veränderung. Monotone Steigungen sind nach dem Subsampling nicht mehr vorhanden. Die einfachen Prädiktoren können die Daten nicht zuverlässig vorhersagen.

### 6.3.3 Rekursive Lineare Kodierung

Die Rekursive Lineare Kodierung ist in der Lage nicht-stetigen Kanälen eine sinnvolle Vorhersage zu berechnen. Das Verfahren ist im Abschnitt 3.4 genauer erläutert. Die Punkte werden ins sphärische Koordinatensystem überführt. In diesem Koordinatensystem sind 16 Bit Genauigkeit pro Koordinatenachse ausreichend und die PCA muss nicht berechnet werden. Im Schnitt können dadurch 30 Bytes pro Feldlinie eingespart werden. Zusätzlich werden die Vorhersagefehler quantisiert.

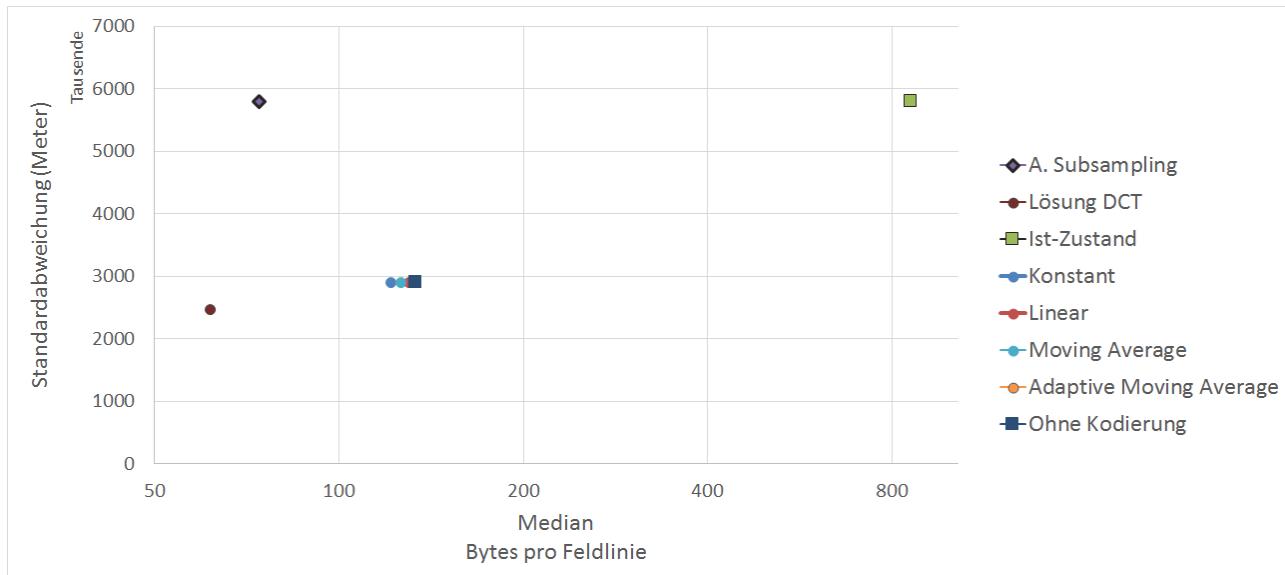


Abbildung 35: Kompressionsraten der Prediktiven Kodierungen mit dem adaptiven Subsampling.

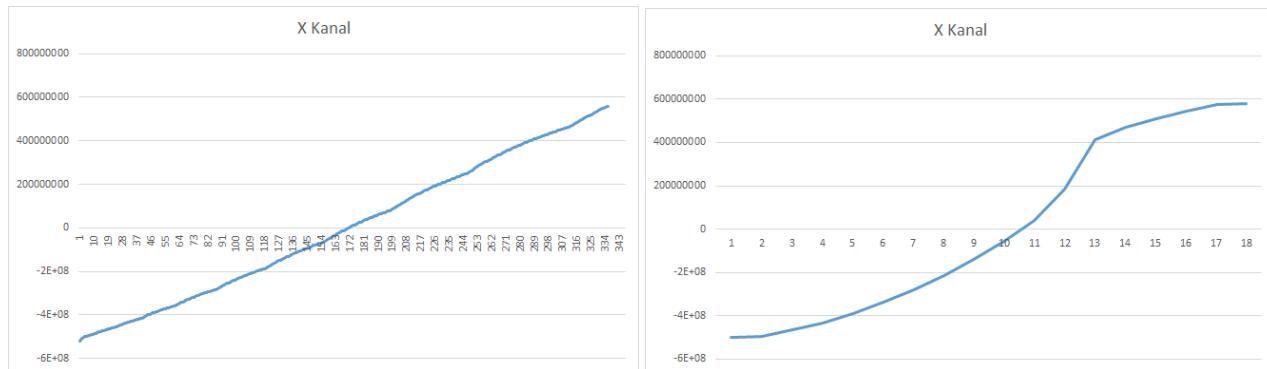


Abbildung 36: Änderung der Eigenschaften der Daten durch das Angle Subsampling. Links der Kanal einer Feldlinie vor, rechts der Kanal nach dem Angle Subsampling.

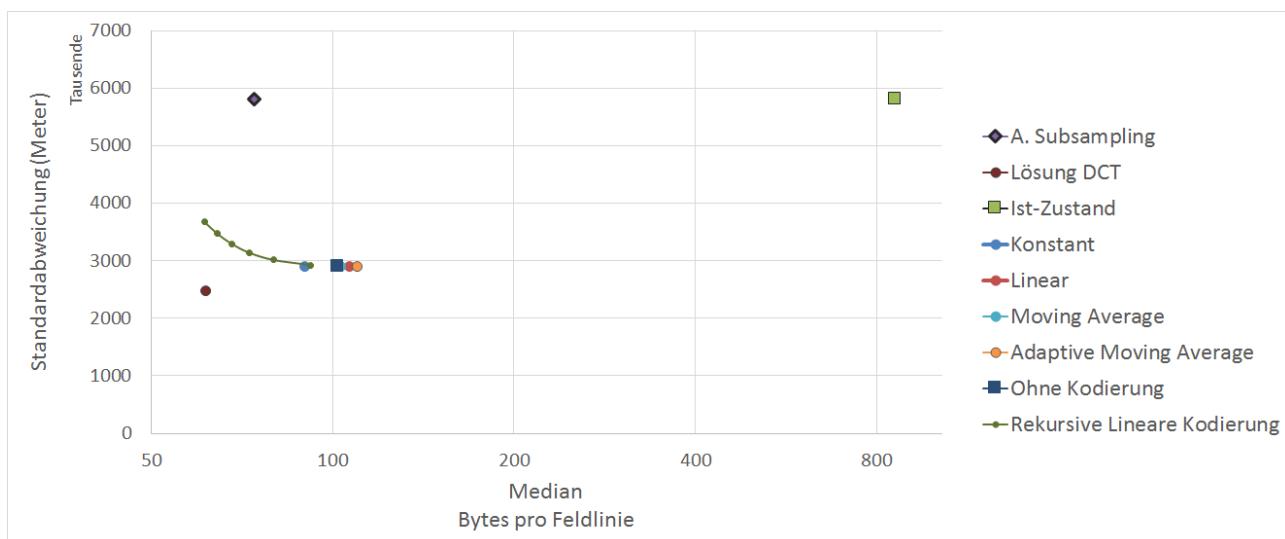


Abbildung 37: Kompressionsraten der Rekursiven Linearen Kodierung mit dem adaptiven Subsampling.

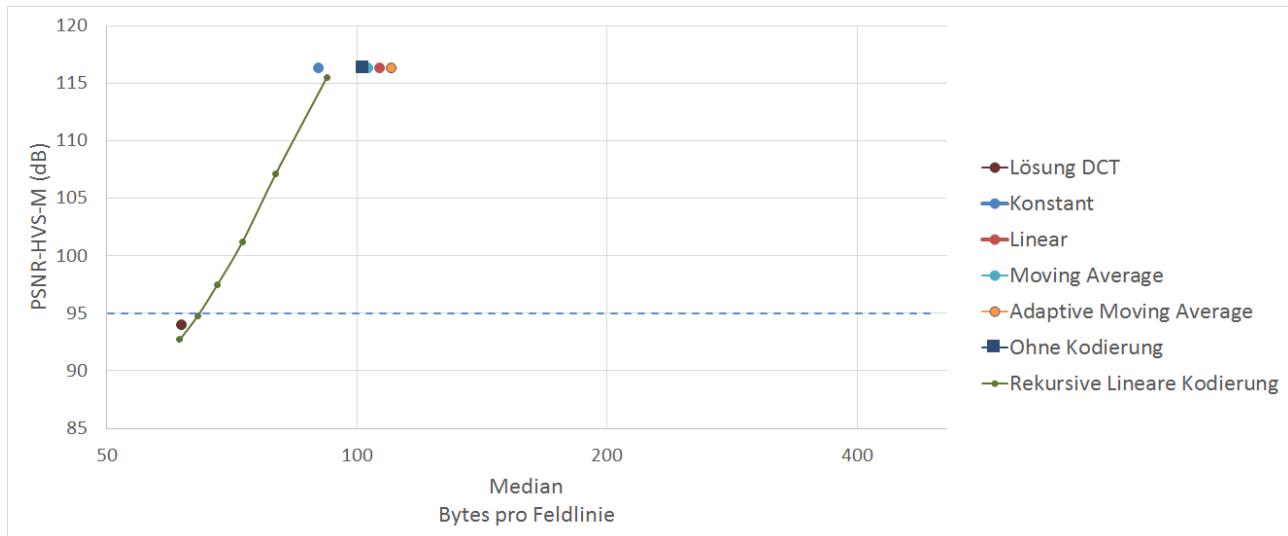


Abbildung 38: Kompressionsraten der Rekursive Lineare Kodierung mit dem adaptiven Subsampling.

Das Diagramm der Abbildung 37 zeigt die Standardabweichung der Rekursiven Linearen Kodierung zu unterschiedlichen Quantisierungen. Das Diagramm der Abbildung 38 zeigt die PSNR-HVS-M Werte der Variante. Die Resultate der einfachen Prädiktoren wurden ebenfalls ohne PCA im sphärischen Koordinatensystem gemessen. Die Rekursive Lineare Kodierung kann eine bessere Kompressionsrate erreichen als die Kompression mit Adaptiven Subsampling. Bei Datenmenge von 64 Bytes pro Feldlinie erreicht diese Variante eine tiefere Standardabweichung als das Adaptive Subsampling zu einem PSNR-HVS-M Wert von 94.7. Diese Variante besitzt weniger ausgeprägte Artefakte als das DCT Verfahren, weist aber eine höhere Standardabweichung auf. Die Variante erreicht eine Kompressionsrate von 13.4 und fällt somit zwischen den Kompressionen mittels DCT und Adaptives Subsampling.

Die Artefakte der Dekompression äussern sich meist als Verschiebungen einzelner Punkte. Im Extremfall können Ringing ähnlichen Artefakte entstehen, welche in der Abbildung 39 dargestellt sind. Der Grund für die Artefaktbildung liegt in der Rekursion: Wenn der Vorhersagefehler des ersten Wertes (siehe Diagramm der Abbildung 11) quantisiert wird, beeinflusst der Quantisierungsfehler den gesamten Kanal. Die Werte, welches als letztes Kodiert werden, erfahren die grösste Verschiebung bei der Dekompression. Diese Variante verwendet einen konstanten Faktor für die Quantisierung. Die Lösung ist eine angepasste Quantisierung, welche die ersten Vorhersagefehler der ersten Rekursionsstufe mit höherer Genauigkeit abspeichert.

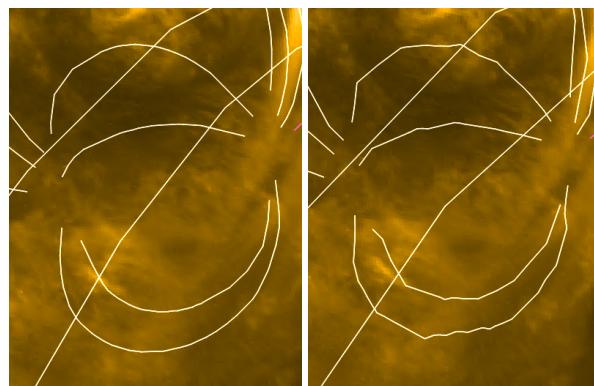


Abbildung 39: Artefakte der Rekursiven Linearen Kodierung. Links sind die originalen Feldlinien.

### 6.3.4 Rekursive Lineare Kodierung mit angepasster Quantisierung

Um die Artefakte der Dekompression zu dämpfen werden die Vorhersagefehler der Rekursionsstufe angepasst quantisiert. Die ersten Rekursionsstufen werden mit höherer Genauigkeit abgelegt.

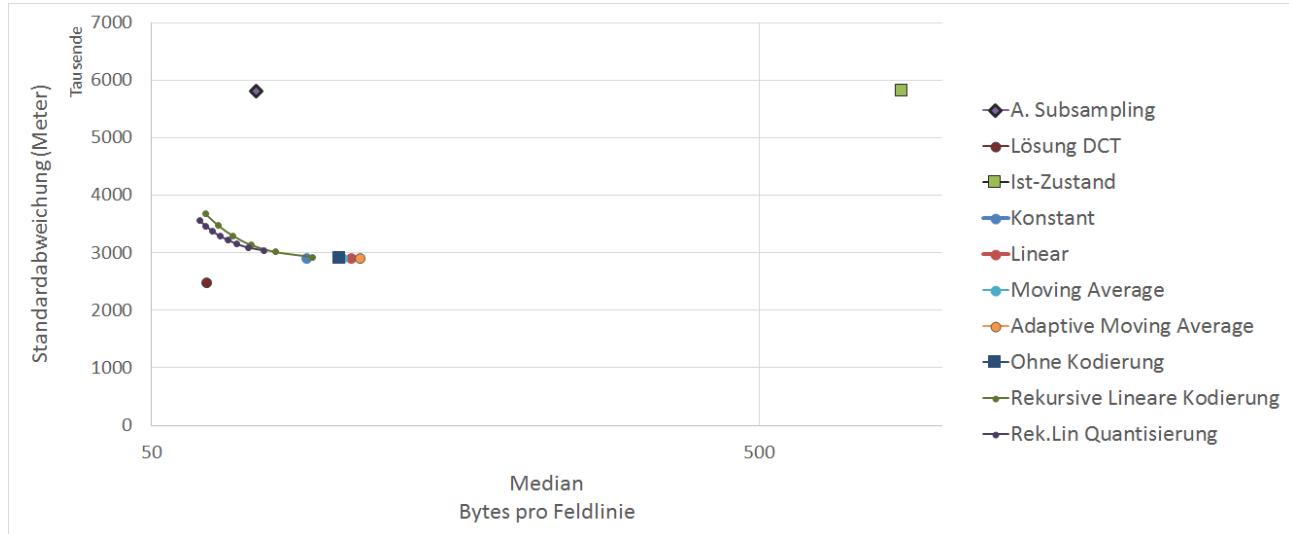


Abbildung 40: Standardabweichung der Rekursive Lineare Kodierung mit angepasster Quantisierung.

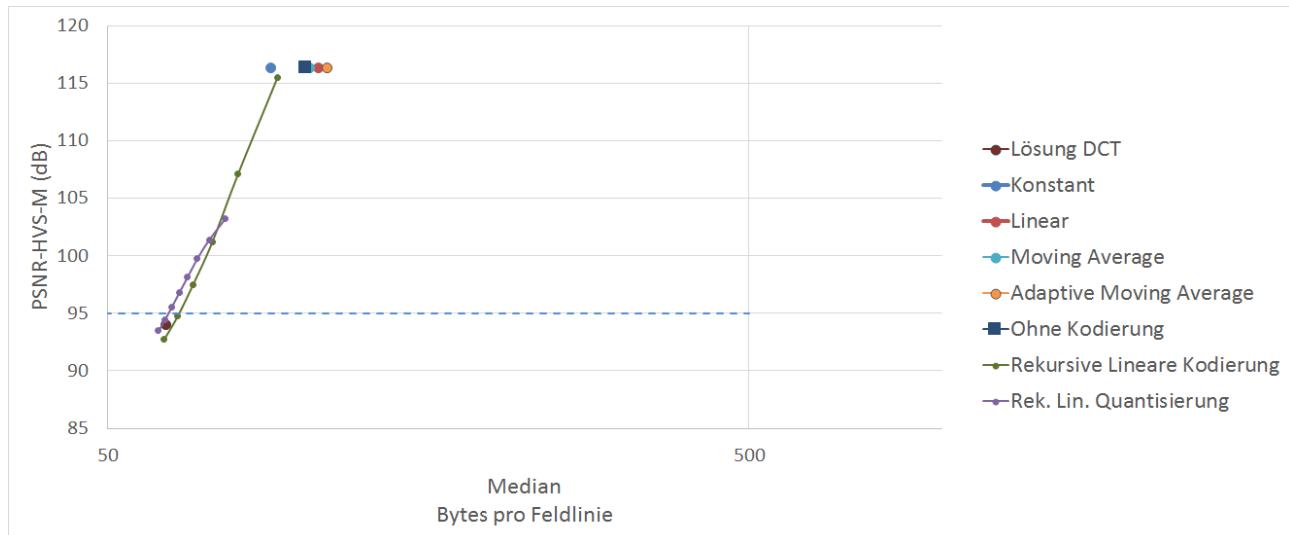


Abbildung 41: PSNR-HVS-M der Rekursiven Linearen Kodierung mit angepasster Quantisierung.

Die Angepasste Quantisierung bringt eine Verbesserung in der Standardabweichung mit sich. Die Diagramme der Abbildungen 40 und 41 visualisieren die Standardabweichung und PSNR-HVS-M zur jeweiligen Kompression. Mit der angepassten Quantisierung kann diese Variante eine ähnliche Kompression zu vergleichbaren PSNR-HVS-M erreichen, wie das DCT Kompressionsverfahren. Mit einer PSNR-HVS-M von 95.5 verbraucht diese Variante 63 Bytes pro Feldlinie, was zu einer Kompressionsrate von 13.6 führt. Der Vorteil dieser Variante gegenüber der DCT Kompression ist, dass wenige zusätzliche Bytes die Artefakte stark dämpfen können.

Die Artefakte haben sich im Vergleich zur vorhergehenden Variante verbessert: Die Abbildung 42 zeigt die Artefaktbildung dieser Variante. Die ringing-ähnlichen Artefakte sind verschwunden. In dieser Variante äußern sich Kompressionsartefakte als Verschiebung einzelner Punkte. Bei hoher Dichte der Feldlinien werden die

Verschiebungen sichtbar, da sie sich häufig kreuzen und eine unästhetische Visualisierung ergeben. Eine leichte Glättung der Feldlinien versteckt die Artefakte und gestaltet eine ästhetische Visualisierung.

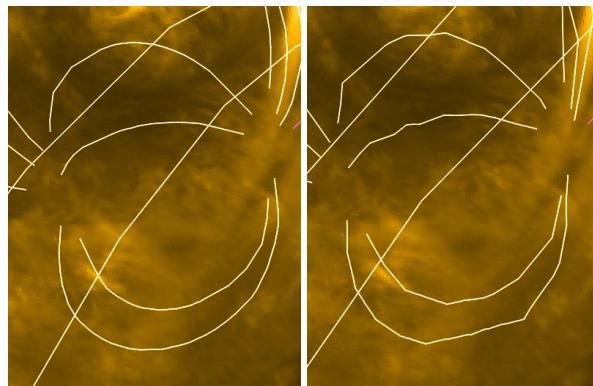


Abbildung 42: Artefakte der Rekursiven Linearen Kodierung mit angepasster Quantisierung. Links sind die originalen Feldlinien.

Das Caching von 1000 Simulationen und die Übertragungszeit fallen zwischen die anderen Kompressionsverfahren mit 72 Megabyte an Arbeitsspeicher beziehungsweise 58 Sekunden Übertragungszeit. Zum Vergleich: Die DCT Variante verbraucht unter denselben Bedingungen 70 und das Adaptive Subsampling 85 Megabyte an Arbeitsspeicher. Bei einer Internetverbindung von 10 Megabit ist bei diesem Verfahren mit einer Übertragungsrate von 16 Simulationen in der Sekunde zu rechnen. Die Laufzeit der Dekompression liegt mit 30.6 Millisekunden ebenfalls zwischen den Kompressionsverfahren Adaptives Subsampling und DCT (siehe Abschnitt 8.3). Mit dieser Laufzeit kann ein Thread durchschnittlich 32 Simulationen pro Sekunde dekomprimieren.

Dieses Verfahren ist ein Kompromiss zwischen Kompressionsrate und Qualität. Die Daten enthalten Kompressionsartefakte. Sie sind aber schwach ausgeprägt sodass auf eine Glättung gegebenenfalls verzichtet werden kann.

## 7 Fazit

Die Ist-Kompression benötigt durchschnittlich 1 Megabyte an Daten pro Simulation der Feldlinien. Das Caching von 1000 Simulationen verbraucht ein Gigabyte an Arbeitsspeicher. Da der Arbeitsspeicher ebenfalls für die Zwischenspeicherung weiterer Daten benötigt wird, ist diese Datenmenge nicht vertretbar. Um das Zwischenspeichern der Simulationen zu ermöglichen, ist eine Kompressionsrate von Faktor 8 – 10 notwendig. Die entwickelten Kompressionsverfahren Adaptives Subsampling, DCT Kompression und Prädiktive Kodierung erreichten eine Kompressionsrate von 11.6, 14.1 und 13.6. Das Zwischenspeichern von 1000 Simulationen benötigt zwischen 70 und 85 Megabyte Arbeitsspeicher. Somit ist das Caching mit allen entwickelten Kompressionsverfahren realisierbar.

Ein Forschungsziel ist, unter welchen Bedingungen ein Streaming der Simulationen möglich ist. Es wird angenommen, dass für die Feldlinien 5 Megabit Bandbreite zur Verfügung stehen. Mit dieser Bandbreite können im Ist-Zustand 0.6 Simulationen in der Sekunde heruntergeladen werden. Der JHeliovewers benötigt im Allgemeinen 1 bis maximal 10 Simulationen in der Sekunde für die Visualisierung. Mit derselben Bandbreite erreichen die entwickelten Kompressionen durchschnittlich 7, 9 und 8 Simulationen in der Sekunde. Der Maximalfall von 10 Simulationen in der Sekunde benötigt eine höhere Bandbreite oder eine höhere Kompression zu schlechterer Qualität. Für den allgemeinen Fall ist mit den entwickelten Kompressionen und einer modernen Internetverbindung das Streaming möglich.

Das Auftreten von Ringing oder Ringing ähnlichen Artefakten ist der limitierende Faktor für die entwickelten Kompression: Im JHeliovewer sind auch leicht ausgeprägte Ringing Artefakte störend, da sie durch das Zoom Feature entdeckt werden können. Wenn für das Fernziel, eine flüssige Animation der Feldlinien die Anzahl übertragenen Simulationen pro Sekunde erhöht wird, muss auf den artefaktfreien Zoom verzichtet werden. Das DCT Verfahren eignet sich für eine möglichst hohe Kompressionsrate: Bei steigender Kompressionsrate sinkt die Qualität der Daten langsamer als bei den anderen Kompressionsverfahren.

Das Verfahren des Adaptiven Subsamplings ist einfach umzusetzen und beinhaltet minimale Artefakte. Es werden ausschliesslich die Daten übertragen, welche der JHeliovewer für die Visualisierung benötigt. Die Datenmenge ist begrenzt durch die Leistung der Grafikkarte. Wenn in Zukunft die zu visualisierend Datenmenge erhöht wird, sinkt die Kompressionsrate des Verfahrens. Das Problem weisen die Kompressionsverfahren DCT und Prädiktive Kodierung nicht auf, da mehr Daten übertragen werden, als die Visualisierung benötigt.

Die Kompression mit der Diskreten Kosinus Transformation erreichte die höchste Kompressionsrate aller Verfahren. Die Kompression fügt Ringing Artefakte ein, welche in der Visualisierung stören. Die Artefakte können durch eine Glättung behoben werden. Die Umsetzung der Dekompression gestaltet sich Komplex: Eine naive Implementation der inversen Kosinus Transformation führt zu einer Laufzeit, die um Faktor 100 langsamer ist als eine naive Implementation der anderen Kompressionsverfahren. Forschungsinstitutionen wie Beispielsweise IRAP [25] sind an Feldliniensimulationen interessiert. Es ist von Vorteil wenn die Dekompression mit moderaten Programmierkenntnissen umgesetzt werden kann. Die Dekompression dieses Verfahrens beinhaltet die grösste Komplexität.

Die Prädiktive Kodierung erreichte eine vergleichbare Kompressionsrate wie die des DCT Verfahrens. Die Artefakte sind weniger ausgeprägt und erst bei hohen Zoomstufen erkennbar. Mit einer leichten Glättung sind die Artefakte in der Visualisierung nicht mehr zu erkennen. Das Verfahren verwendet keine komplexen Transformationen. Wenn Institutionen wie Beispielsweise IRAP [25] eine Dekompression entwickelt, sind für die Umsetzung der Kompression weniger Programmierkenntnisse notwendig als beim DCT-Verfahren. Das Verfahren der Prädiktiven Kodierung wurde als finale Lösung ausgewählt. Es ist ein Kompromiss zwischen Kompression und Artefaktbildung.

Das Kompressionsverfahren der Prädiktiven Kodierung kann für anderen Anwendungsfälle eingesetzt werden wie Kompression von Feldlinien von Spulen oder Flugbahnen der Teilchenphysik. Das entwickelte Dateiformat

ist auf kein Koordinatensystem oder maximale Genauigkeit begrenzt. Die Quantisierung muss auf den jeweiligen Anwendungsfall angepasst werden : Es ist ein Kompromiss zwischen Kompressionsrate und Artefakte, welche für den jeweiligen Fall entschieden werden muss. Eine ähnliche Kompressionsrate kann erwartet werden, wenn die Daten mit 16 Bit Genauigkeit abgespeichert werden und die Punktfolgen niederfrequenten Schwingungen beinhalten.

Für weitere Forschungen sind Kompressionsverfahren interessant, welche kaum oder keine Ringing Artefakte einführen wie die Wavelet Transformation, Compressive Sensing und Curve Fitting. Die Artefakte der Wavelet Transformation kann durch Auswahl des Wavelets beeinflusst werden. Compressive Sensing kann auf die zu komprimierenden Daten optimiert werden. Es kann die Eigenschaft genutzt werden, dass die Feldlinien sich hauptsächlich in Skalierung, Rotation und Verschiebung unterscheiden. Curve Fitting ist ebenfalls ein mögliches Verfahren. Die Artefakte einer Bildkompression mit Curve Fitting sind Rauschunterdrückung und allgemeine Schärfung des Bildes. Ähnliche Kompressionsartefakte können entstehen, wenn man es für die Feldliniendaten verwendet.

## Literatur

- [1] Gregory K Wallace. The jpeg still picture compression standard. *Consumer Electronics, IEEE Transactions on*, 38(1):xviii–xxxiv, 1992.
- [2] Siu-Wai Wu and Allen Gersho. Rate-constrained picture-adaptive quantization for jpeg baseline coders. In *Acoustics, Speech, and Signal Processing, 1993. ICASSP-93., 1993 IEEE International Conference on*, volume 5, pages 389–392. IEEE, 1993.
- [3] Ching-Yang Wang, Shiuh-Ming Lee, and Long-Wen Chang. Designing jpeg quantization tables based on human visual system. *Signal Processing: Image Communication*, 16(5):501–506, 2001.
- [4] Wikipedia. Run-length encoding — wikipedia, the free encyclopedia, 2014. [Online; accessed 28-January-2015].
- [5] David A Huffman et al. A method for the construction of minimum redundancy codes. *proc. IRE*, 40(9):1098–1101, 1952.
- [6] Ruwen Schnabel and Reinhard Klein. Octree-based point-cloud compression. In *SPBG*, pages 111–120, 2006.
- [7] Wikipedia. Octree — wikipedia, the free encyclopedia, 2014. [Online; accessed 4-January-2015].
- [8] Michael Unser, Akram Aldroubi, and Murray Eden. B-spline signal processing. i. theory. *Signal Processing, IEEE Transactions on*, 41(2):821–833, 1993.
- [9] Michael Unser, Akram Aldroubi, and Murray Eden. B-spline signal processing. ii. efficiency design and applications. *Signal Processing, IEEE Transactions on*, 41(2):834–848, 1993.
- [10] Wikipedia. Np-hard — wikipedia, the free encyclopedia, 2015. [Online; accessed 29-January-2015].
- [11] Joel A Tropp and Anna C Gilbert. Signal recovery from random measurements via orthogonal matching pursuit. *Information Theory, IEEE Transactions on*, 53(12):4655–4666, 2007.
- [12] Ori Bryt and Michael Elad. Compression of facial images using the k-svd algorithm. *Journal of Visual Communication and Image Representation*, 19(4):270–282, 2008.
- [13] Paruj Ratanaworabhan, Jian Ke, and Martin Burtscher. Fast lossless compression of scientific floating-point data. In *Data Compression Conference, 2006. DCC 2006. Proceedings*, pages 133–142. IEEE, 2006.
- [14] Jean-loup Gailly. The gzip home page, November 2014. [Online; accessed 18-November-2014].
- [15] Igor Pavlov. 7-zip, November 2014. [Online; accessed 18-November-2014].
- [16] Alexander Roshal. rarlab, November 2014. [Online; accessed 18-November-2014].
- [17] L Peter Deutsch. Deflate compressed data format specification version 1.3. 1996.
- [18] Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. *IEEE Transactions on information theory*, 23(3):337–343, 1977.
- [19] Wikipedia. Ringing artifacts — wikipedia, the free encyclopedia, 2014. [Online; accessed 23-December-2014].
- [20] André Kaup. Reduction of ringing noise in transform image coding using simple adaptive filter. *Electronics Letters*, 34(22):2110–2112, 1998.

- [21] HyunWook Park and Yung Lyul Lee. A postprocessing method for reducing quantization effects in low bit-rate moving picture coding. *Circuits and Systems for Video Technology, IEEE Transactions on*, 9(1):161–171, 1999.
- [22] Nikolay Ponomarenko, Flavia Silvestri, Karen Egiazarian, Marco Carli, Jaakko Astola, and Vladimir Lukin. On between-coefficient contrast masking of dct basis functions. In *Proceedings of the Third International Workshop on Video Processing and Quality Metrics*, volume 4, 2007.
- [23] Wikipedia. Dc bias — wikipedia, the free encyclopedia, 2014. [Online; accessed 4-February-2015].
- [24] Hervé Abdi and Lynne J Williams. Principal component analysis. *Wiley Interdisciplinary Reviews: Computational Statistics*, 2(4):433–459, 2010.
- [25] IRAP. Irap, November 2014. [Online; accessed 18-November-2014].
- [26] NASA. The fits support office, November 2014. [Online; accessed 18-November-2014].
- [27] Wikipedia. Endianness — wikipedia, the free encyclopedia, 2014. [Online; accessed 30-November-2014].

## Abbildungsverzeichnis

1	Visualisierung der Feldlinien im JHeliovewer . . . . .	1
2	Dekomprimierte Feldliniendaten. Oben links ist das Original, rechts die Feldlinien des Adaptiven Subsamplings. Unten sind die Feldlinien der Kompressionen mit der Diskrete Kosinus Transformation und der Prädiktiven Kodierung zu sehen. . . . .	2
3	Vereinfachter Ablauf einer verlustbehafteten Kompression . . . . .	3
4	Aufbau der JPEG Kompression [1] . . . . .	3
5	Aufbau einer Octree basierten Point Cloud Kompression. . . . .	4
6	Aufbau der Ist-Kompression. . . . .	6
7	Aufbau des Kompressionsverfahrens: Adaptives Subsampling. . . . .	6
8	Darstellung des Adaptiven Subsapmlings. Rot sind die Punkte, welche geprüft und gelöscht wurden. Grün ist der Punkt dargestellt, welcher geprüft wird. . . . .	7
9	Aufbau der Kompression Adaptives Subsampling. . . . .	8
10	Aufbau der Kompressionsverfahren Prädiktive Kodierung. . . . .	11
11	Erster Schritt der Rekursiver Linearer Kodierung. Zu sehen sind das zu kodierende Signal, die Vorhersage und den Fehler der Vorhersage. . . . .	12
12	Zweiter Schritt der Rekursiver Linearer Kodierung. Die Vorhersage beinhaltet nun zwei Strecken. . . . .	12
13	Darstellung der Fehlerberechnung. Die Punkte sind die Originaldaten, die Quadrate sind die Punkte nach der Kompression. . . . .	15
14	Flussdiagramm der PSNR-HVS-M Berechnung [22]. . . . .	16
15	Zustandsdiagramm der Feldliniendaten . . . . .	18
16	Diagramm der Implementation vom Vorladen und Caching . . . . .	19
17	Vergleich des Lösungsansatzes: Adaptives Subsampling zur Ist-Kompression. . . . .	21
18	Artefakte des Verfahrens Adaptives Subsampling. . . . .	22
19	Vergleich der DCT Kompression mit dem Verfahren des Adaptiven Subsamplings . . . . .	23
20	Artefakte der DCT Dekompression anhand Beispieldaten . . . . .	24
21	Vergleich der DCT Kompression der Ableitung mit der DCT Kompression . . . . .	24
22	Artefakte der DCT Kompression der Ableitung . . . . .	25
23	Vergleich der PCA DCT Kompression der Ableitung mit der DCT Kompression der Ableitung . . . . .	25
24	Vergleich der Kompression mit und ohne Byte-Kodierung . . . . .	26
25	Vergleich des Einflusses der Randbehandlung . . . . .	27
26	Artefakte der Kompression. Links sind die originalen Feldlinien, rechts die Dekomprimierten. . . . .	27
27	Artefakte der Kompression, links sind die originalen Feldlinien, rechts die Dekomprimierten der Variante 6.2.4. . . . .	28
28	Abrupte Steigungen bei Feldlinien, welche von der Sonne ins Weltall führen. . . . .	28
29	Approximation der Feldlinien "Sonnenoberfläche zu Sonnenoberfläche". Je höher die PSNR-HVS-M, desto besser ist die Approximation. . . . .	29
30	Approximation der der Feldlinien "Sonnenoberfläche ins Weltall" oder "Weltall zur Sonnenoberfläche". Je höher die PSNR-HVS-M, desto besser ist die Approximation. . . . .	29
31	Standardabweichung der abschliessenden DCT Variante. . . . .	30
32	Die am stärksten ausgeprägten Artefakte der abschliessenden Variante. Links ohne Glättung, rechts mit Glättung . . . . .	31
33	Kompressionsraten der vier Prediktoren im Vergleich zum Ist-Zustand. . . . .	32
34	Kompressionsraten der Prediktoren mit Adaptiver Byte Kodierung. . . . .	33
35	Kompressionsraten der Prediktiven Kodierungen mit dem adaptiven Subsampling. . . . .	34
36	Änderung der Eigenschaften der Daten durch das Angle Subsampling. Links der Kanal einer Feldlinie vor, rechts der Kanal nach dem Angle Subsampling. . . . .	34
37	Kompressionsraten der Rekursiven Linearen Kodierung mit dem adaptiven Subsampling. . . . .	34
38	Kompressionsraten der Rekursive Lineare Kodierung mit dem adaptiven Subsampling. . . . .	35

39	Artefakte der Rekursiven Linearen Kodierung. Links sind die originalen Feldlinien.	35
40	Standardabweichung der Rekursive Lineare Kodierung mit angepasster Quantisierung.	36
41	PSNR-HVS-M der Rekursiven Linearen Kodierung mit angepasster Quantisierung.	36
42	Artefakte der Rekursiven Linearen Kodierung mit angepasster Quantisierung. Links sind die originalen Feldlinien.	37
43	First recursive step.	47
44	Second recursive step.	47

## Tabellenverzeichnis

1	Kompressionsraten der Lösungsansätze.	2
2	Anordnung der Simulationsdaten der Ist-Kompression	6
3	Beispiel eines abgespeicherten Kanals mit der Längenkodierung.	10
4	Aufteilung eines Bytes der adaptiven Genauigkeitskodierung. X sind Nutz-Bits.	10
5	Breadth First Ordnung der Vorhersagefehler.	13
6	Einfluss des $f_m$ Faktors auf die PSNR-HVS-M.	17
7	Content of the FITS File	44
8	Unsigned Adaptive Byte Encoding. X are data bits.	45
9	Signed Adaptive Byte Encoding of the First Byte. X are data bits.	45
10	Breadth First ordering.	47

## 8 Anhang

### 8.1 Installationsanleitung

JHeliovewer ist zum Zeitpunkt der Arbeit noch in Entwicklung. Eine Version des JHeliovewers mit der umgesetzten Dekompression kann unter <http://github.com/Helldevastator/JHelioViewer> heruntergeladen werden. Alle verwendeten Programmblibliotheken und Abhängigkeiten sind im Projekt enthalten. Das Projekt kann in einer Java IDE wie Eclipse oder Netbeans importiert werden. Die Arbeit beschränkt sich auf das Paket `org.heliovewer.g13d.plugins.pfssplugin`. Im Paket `pfssplugin.data` ist die Dekompression und das Caching implementiert. Für die Ausführung soll die Klasse `PfssPluginLauncher` verwendet werden.

### 8.2 File Format and Decompression

This section describes the decoding and decompression algorithm used for compressing fieldline data. The data is written in the FITS Format [26] and compressed with RAR. The content of the FITS file is described in Table 7.

The following steps have to be taken to decompress the fieldline data:

Name	Datatype	Encoding	Content Description
B0	Double	None	Latitude to Earth
L0	Double	None	Longitude to Earth
LINE_LENGTH	Byte Array	Adaptive Precision U.	Length of each line
START_R	Byte Array	Adaptive Precision	Radius of start point for each line
END_R	Byte Array	Adaptive Precision	Radius of end point for each line
START_PHI	Byte Array	Adaptive Precision	Latitude of start point for each line
END_PHI	Byte Array	Adaptive Precision	Latitude of end point for each line
START_THETA	Byte Array	Adaptive Precision	Longitude of start point for each line
END_THETA	Byte Array	Adaptive Precision	Longitude of end point for each line
CHANNEL_R	Byte Array	Adaptive Precision	Radius Channel of prediction errors of each line
CHANNEL_PHI	Byte Array	Adaptive Precision	Latitude Channel of prediction errors of each line
CHANNEL_THETA	Byte Array	Adaptive Precision	Longitude Channel of prediction errors of each line

Tabelle 7: Content of the FITS File

1. Decode RAR
2. Split the fieldlines. Each line has 1 Length, 1 start point, 1 end point and the size of "Length" of prediction errors.
3. Do the Byte Decoding specified under section 8.2.1
4. reverse the predictive coding described under section 8.2.3
5. convert from spherical coordinate system to cartesian. Section 8.2.2 describes the process.

#### 8.2.1 Encodings

##### Unsigned Adaptive Precision Encoding

The Data is stored using the byte encoding in table 8. If the Continue Flag is set, then the next Byte also

Byte								
Continue Flag	X	X	X		X	X	X	X

Tabelle 8: Unsigned Adaptive Byte Encoding. X are data bits.

belongs to the value. The Bytes are Stored in Big-Endian[27] convention. The next Byte contains the seven lesser bits if the Continue Flag is set.

### Adaptive Precision Encoding

Same Principle as the Unsigned Adaptive Precision Encoding, but the First Byte of each value contains a

Byte								
Continue Flag	Sign Flag	X	X		X	X	X	X

Tabelle 9: Signed Adaptive Byte Encoding of the First Byte. X are data bits.

Sign Flag as demonstrated in table 9. Here is an example implementation:

```

1  public static final int continueFlag = 128;
2  public static final int signFlag = 64;
3  public static final int dataBitCount = 7;

4  public static int[] decodeAdaptive(byte[] data) {
5    int length = calcLength(data);
6    int[] output = new int[length];
7    int outIndex = 0;

8    //for each encoded byte
9    for (int i = 0; i < data.length; i++) {
10      byte current = data[i];
11      int value = (short) (current & (signFlag - 1));
12      int minus = -(current & signFlag);

13      //add encoded bytes as long as the continue flag is set.
14      boolean run = (current & continueFlag) != 0;
15      while (run) {
16        current = data[++i];
17        run = (current & continueFlag) != 0;
18        minus <= dataBitCount;
19        value <= dataBitCount;
20        value += current & (continueFlag - 1);
21      }
22      output[outIndex++] = (value + minus);
23    }

24    return output;
25  }

26  private static int calcLength(byte[] data) {
27    int out = 0;

28    for (int i = 0; i < data.length; i++) {
29      if ((data[i] & continueFlag) == 0)
30        out++;
31    }
32  }

```

```

        return out;
    }

```

---

### 8.2.2 Transformation to cartesian Coordinates

The spherical coordinates can be transformed into the sun's cartesian coordinate system with a few simple calculations. Step 1: Add the displacement of the Compression.

$$\begin{aligned}
 R_{raw} &= R_{raw} + 8192 \\
 Phi_{raw} &= Phi_{raw} + 16384 \\
 Theta_{raw} &= Theta_{raw} + 8192
 \end{aligned} \tag{8.1}$$

Step 2: Convert to Floating point numbers.

$$\begin{aligned}
 R &= R_{raw}/8192 * Sunradius(Meter) \\
 Phi &= Phi_{raw}/32768.0 * 2 * \pi \\
 Theta &= Theta_{raw}/32768.0 * 2 * \pi
 \end{aligned} \tag{8.2}$$

Step 3: Rotate the coordinates so they are centered around the viewpoint of Earth.

$$\begin{aligned}
 Phi &= Phi - l0/180 * \pi \\
 Theta &= Theta + b0/180 * \pi
 \end{aligned} \tag{8.3}$$

Step 4: Transform from spherical to cartesian.

$$\begin{aligned}
 X &= R * sinus(Theta) * sinus(Phi) \\
 Y &= R * cosinus(Theta) \\
 Z &= R * sinus(Theta) * cosinus(Phi)
 \end{aligned} \tag{8.4}$$

### 8.2.3 Decode Prediction

Step 1, multiply the prediction errors with:

$$\begin{aligned}
 &\text{factor 6 if index } < 5 \\
 &\text{factor 8 if } 5 \leq \text{index} < 16 \\
 &\text{factor 16 if } 5 \leq \text{index}
 \end{aligned} \tag{8.5}$$

Step 2, Decode Prediction.

It uses a recursive Algorithm which predicts the mid-value of the channel. The picture 43 shows the first recursive Step: The Prediction assumes that the mid-value of the channel is on the line between start- and end value. To decode a channel, calculate the prediction and subtract the prediction error from the value. Continue recursively as shown in picture 44. The prediction errors are sorted in breadth-first order. Table 10 shows the ordering.

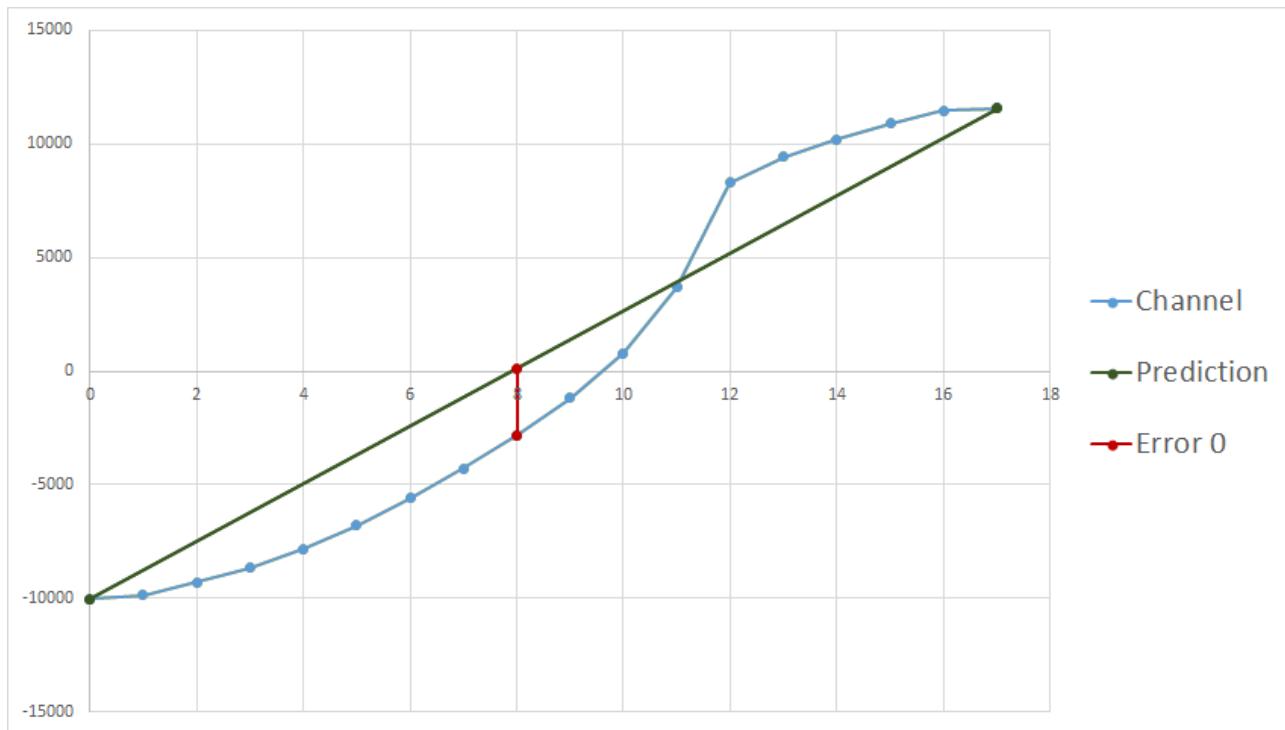


Abbildung 43: First recursive step.

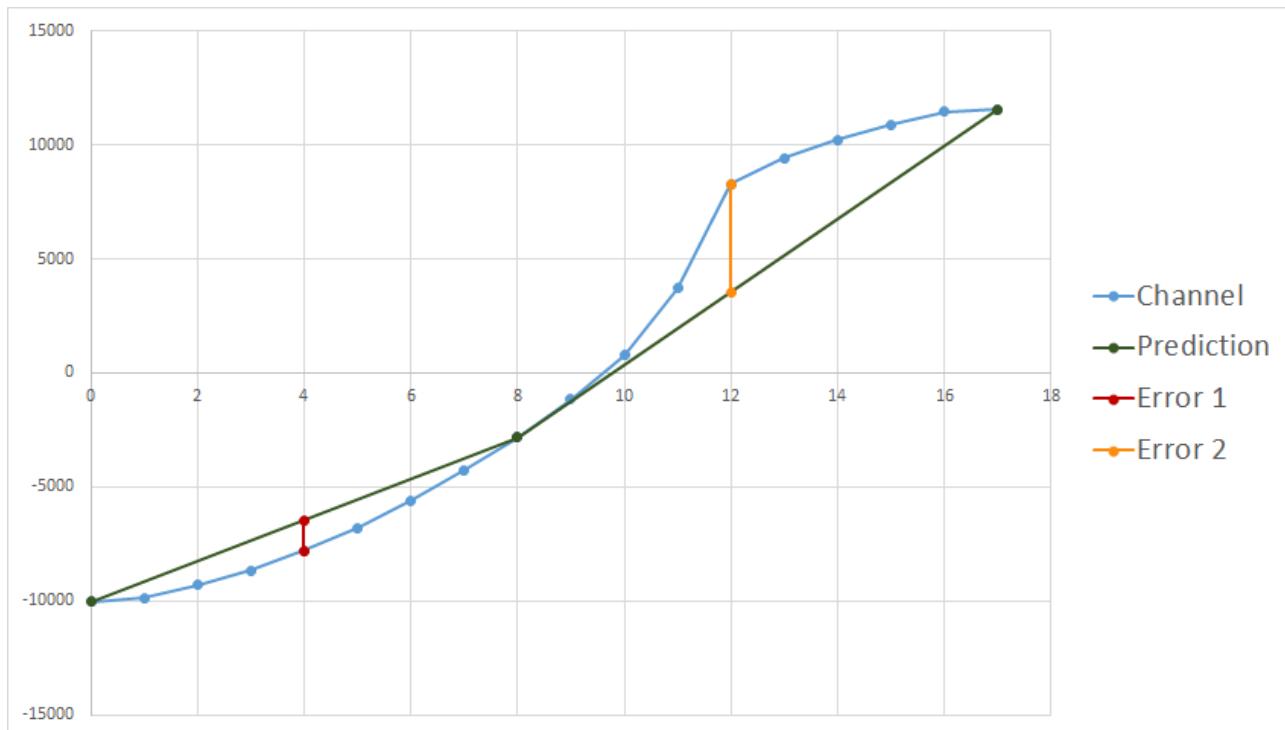


Abbildung 44: Second recursive step.

Prediction Errors of a Channel							
Step 0	Step 1		Step 2				...
Error <sub>0</sub>	Error <sub>1</sub>	Fehler <sub>2</sub>	Error <sub>3</sub>	Error <sub>4</sub>	Error <sub>5</sub>	Error <sub>6</sub>	...

Tabelle 10: Breadth First ordering.

Here is an example implementation in Java:

```
1 public static float[] decodeChannel(float[] encodedChannel, float startValue, float
   endValue) {
2     float[] decodedChannel = new float[encodedChannel.length+2];
3     decodedChannel[0] = startValue;
4     decodedChannel[decodedChannel.length-1] = endValue;
5     if(decodedChannel.length > 2) {
6         LinkedList<Indices> queue = new LinkedList<>();
7         queue.add(new Indices(0, decodedChannel.length-1));
8         while(!queue.isEmpty()) {
9             prediction(queue, decodedChannel, channels[i], channelIndex);
10            }
11        }
12
13        return decodedCannel;
14    }
15
16 private static void prediction(LinkedList<Indices> queue, float[] decodedChannel,
17   float[] encodedChanel, int nextIndex) {
18     Indices i = queue.pollFirst();
19     float start = decodedChannel[i.startIndex];
20     float end = decodedChannel[i.endIndex];
21
22     int toPredictIndex = (i.endIndex - i.startIndex) / 2 + i.startIndex;
23     float predictionError = encodedChanel[nextIndex];
24
25     //predict
26     float predictionFactor0 = (toPredictIndex-i.startIndex)/float(i.endIndex - i.
27       startIndex);
28     float predictionFactor1 = (i.endIndex-toPredictIndex)/float(i.endIndex - i.
29       startIndex);
30     float prediction = (int)(predictionFactor0* start + predictionFactor1*end);
31     decodedChannel[toPredictIndex] = prediction-predictionError;
32
33     //add next level of indices
34     if (i.startIndex + 1 != toPredictIndex){
35         Indices next = new Indices(i.startIndex,toPredictIndex);
36         queue.addLast(next);
37     }
38     if (i.endIndex - 1 != toPredictIndex) {
39         Indices next = new Indices(toPredictIndex,i.endIndex);
40         queue.addLast(next);
41     }
42
43 private static class Indices {
44     public int startIndex;
45     public int endIndex;
46
47     public Indices(int start, int end) {
48         this.startIndex = start;
49         this.endIndex = end;
50     }
51 }
```

### 8.3 Performance Tests

Lösungsansatz	Durchschnittliche Dekompressionszeit (ms)
Adaptives Subsampling	19 ms
DCT - Naive Inverse DCT	3100 ms
DCT - Mit Kosinus Caching	65 – 350 ms
DCT- Mit JTransform & Kosinus Caching	100 – 135 ms
Prediktive Kodierung	31 ms

#### Testmaschine

Prozessor	Intel i7-4600
Arbeitsspeicher	16 GB
OS	Windows 8.1 Enterprise 64bit

## 9 Ehrlichkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende schriftliche Arbeit selbstständig und nur unter Zuhilfenahme der in den Verzeichnissen oder in den Anmerkungen genannten Quellen angefertigt habe. Ich versichere zudem, diese Arbeit nicht bereits anderweitig als Leistungsnachweis verwendet zu haben. Eine Überprüfung der Arbeit auf Plagiate unter Einsatz entsprechender Software darf vorgenommen werden.

Windisch, 8. Februar 2015

Jonas Schwammburger