

Fortgeschrittenenpraktikum in der AG Numerik / Wavelet-Analysis

Philipps Universität Marburg

(Nichtlineare Diffusion in der Bildverarbeitung)

Ergebnisbericht von Thomas Wolfgang Geiger

28. Juli 2010

Inhaltsverzeichnis

1	Einleitung	2
1.1	Ausgangspunkt	2
1.2	Kurzwiederholung der Grundlagen nichtlinearer Diffusionsfilter und der gewählten Diskretisierung	2
1.2.1	Isotroper und anisotroper Filter - kontinuierliche Formulierung	2
1.2.2	Diskretisierung im isotropen Fall mit dem additiven Operator Splitting	5
2	Implementierung von Diffusionsfiltern	7
2.1	Isotroper Filter für n-dimensionale Bilder	7
2.1.1	Anordnung eines n-dimensionalen Bildes als Vektor	7
2.1.2	Die Gleichungssysteme für das AOS	10
2.1.3	Optimierte Randpixelerkennung	11
2.1.4	Beispiele	11
2.2	Filtern von Farbbildern	14
2.2.1	Vorstellung der verschiedenen Methoden	14
2.2.2	Gegenüberstellung der verschiedenen Methoden	16
2.3	Anisotroper Filter	17
2.3.1	Berechnung des Diffusionstensors	17
2.3.2	Diskretisierung des Divergenzoperators für das AOS	18
2.3.3	Beispiele	21
3	Bedienung des Filters mit der grafischen Oberfläche	25
3.1	Konfiguration der grafischen Oberfläche	25
3.2	Verwaltung geöffneter Bilder	27
3.3	Konfiguration eines Filters	30
3.4	Filtern von Bildern	32
3.5	Bearbeiten von Bildern	34
4	Details der Implementierung des Filters	37
5	Details der Implementierung der grafischen Oberfläche	38
5.1	Grobübersicht und Grundkonzepte	38
5.2	Verwaltung der geladenen Bilder	39
5.3	Verwaltung des Filtervorgang durch einen worker-thread	39
6	Zusammenfassung und offene Probleme	41
	Literatur	42

1 Einleitung

1.1 Ausgangspunkt

Dieses Fortgeschrittenenpraktikum basiert auf der Diplomarbeit *Nichtlineare Diffusion: Theoretische Analyse und Anwendungen bei Strömungssimulationen* von Thomas Künzel an der Philipps Universität Marburg ([1]) welche die Theorie eines isotropen nichtlinearen Diffusionsfilters beschreibt und der in diesem Rahmen angefertigten Implementierung in Form eines C++ Programms. Die Ziele waren zum einen die Erweiterung dieser Implementierung zu einem Filter, der auch mehrdimensionale Bilddaten, Farbbilder und anisotropes Filtern unterstützt und zum anderen die Bereitstellung einer einfach zu verwendenden grafischen Oberfläche. Da die ursprüngliche Implementierung einige Schwachpunkte in ihrer Strukturierung aufweist, wurde das Filterprogramm ebenfalls von einigen wenigen Ausnahmen abgesehen, von Grund auf neu geschrieben.

In diesem Ergebnisbericht werden im einleitenden Kapitel zunächst informell und nicht strikt die wesentlichen Grundlagen und Ideen von nichtlinearen Diffusionsfiltern und dem additiven Operator Splitting, welches als Diskretisierung verwendet wird, wiederholt, die unverzichtbar für alles weitere sind. Im zweiten Abschnitt werden speziell für die Erweiterung der Implementierung aus der Diplomarbeit wichtige Ergebnisse zusammengetragen. Im dritten Abschnitt wird die Bedienung des Filters mit der grafischen Oberfläche erläutert und die abschließenden beiden Abschnitte gehen auf die grobe Struktur der Implementierung ein.

1.2 Kurzwiederholung der Grundlagen nichtlinearer Diffusionsfilter und der gewählten Diskretisierung

1.2.1 Isotroper und anisotroper Filter - kontinuierliche Formulierung

Das Ziel eines nichtlinearen Diffusionsfilters ist die Reduktion unerwünschter Information in einem Bild durch einen Diffusionsprozess. Dabei hängt es von der Anwendung ab, was unerwünschte Information ist, der Filter kann beispielsweise zur Rauschreduktion, Bildwiederherstellung aber auch tatsächlich zum Herausarbeiten der wesentlichen Informationen in einem Bild verwendet werden.

Das Bild

$$f : \Omega \longrightarrow \mathbb{R} \quad \Omega = \prod_{i=1}^n [0, a_i] \subset \mathbb{R}^n$$

wird dazu in der kontinuierlichen Formulierung als Anfangswert einer Diffusionsgleichung, d.h. einer bestimmten partiellen Differentialgleichung, mit homogenen Neumann-Randbedingungen interpretiert:

$$\begin{aligned} \partial_t u &= \operatorname{div}(D \nabla u) && \text{auf } \Omega \times (0, \infty) \\ u(x, 0) &= f(x) && \text{auf } \Omega \\ \partial_{x_i} u &= 0 && \text{auf } \partial \Omega \times (0, \infty) \end{aligned}$$

Die Differentialgleichung $\partial_t u = \operatorname{div}(D \nabla u)$ entsteht dabei aus physikalischen Überlegungen: Wird $u(x, t)$ als Konzentration eines Mediums (z.B. ein Gas) an einer Stelle x im Raum zum Zeitpunkt t aufgefasst, so erzeugt ein Konzentrationsunterschied ∇u (stets nur in den Veränderlichen des Ortes zu lesen) einen Fluss j , der versucht diesen Unterschied auszugleichen gemäß der Gleichung (Ficksches Gesetz)

$$j = -D \nabla u$$

Der Diffusionstensor $D \in \mathbb{R}^{n \times n}$, eine positiv definite, symmetrische Matrix, beschreibt dabei den Zusammenhang zwischen Konzentrationsunterschied und induziertem Fluss. Da ein Diffusionsprozess Masse nicht zerstört sondern nur umverteilt muss noch die Kontinuitätsgleichung

$$\partial_t u = -\operatorname{div} j$$

gelten, womit man insgesamt die angegebene Diffusionsgleichung erhält.

Der Diffusionstensor darf hierbei sowohl vom Ort als auch von der Zeit abhängen. Der Fall $\nabla u \parallel j$ wird als isotrope Diffusion bezeichnet, der allgemeine Fall als anisotrope Diffusion. Im Fall isotroper Diffusion reduziert sich der Diffusionstensor D auf eine skalarwertige Funktion g , abhängig von Ort und Zeit.

Für die Anwendung in der Bildverarbeitung interpretiert man die Lösung dieser Differentialgleichung zu einem Zeitpunkt t als gefilterte Variante des Bildes. Die gewünschte Verhaltensweise des so entstehenden Filters kann man steuern, indem man den Diffusionstensor abhängig von Eigenschaften des Bildes zur Zeit t am Ort x wählt.

Beispielsweise ist $|\nabla u|^2$ eine von (x, t) abhängige Größe, welche im entsprechenden Punkt die Größe des Kontrastanstiegs in Richtung des stärksten Anstieges im Bild beschreibt. Anstatt u verwendet man üblicherweise die Faltung

$$u_\sigma := K_\sigma * u \quad K_\sigma(x) = \frac{1}{(2\pi\sigma^2)^{\frac{n}{2}}} \exp\left(-\frac{|x|^2}{2\sigma^2}\right)$$

mit einer Gaußkurve. Es ist bekannt (vgl. [2]) dass diese Faltung ebenfalls durch eine Diffusionsgleichung mit $D \equiv 1$ berechnet werden kann, also einer Glättung des Bildes u ohne Rücksicht auf die spezielle Bildstruktur entspricht. Man kann sich also $|\nabla u_\sigma|^2$ als lokale Bildinformation vorstellen, welche die Information $|\nabla u|^2$ gemittelt über eine in der Größe durch σ steuerbaren Umgebung des Punktes (x, t) darstellt. In einem Bildpunkt liefert dies ein Maß für den Kontrastunterschied in Richtung des stärksten Kontrastes.

Für einen isotropen Filter, der ja stets einen Ausgleichsfluss entgegen der Richtung des stärksten Kontrastes erzeugt, genügt diese Information zum Aufstellen des Diffusionstensors g . Wählt man beispielsweise wie in [1] vorgeschlagen

$$g : \mathbb{R}_0^+ \longrightarrow \mathbb{R}, \quad g(s) = \begin{cases} 1 & s = 0 \\ 1 - \exp\left(\frac{-3.315}{(s^4/\lambda^8)}\right) & s > 0 \end{cases}$$

so erhält man folgenden Graphen für die Stärke des induzierten Gegenflusses abhängig von der Stärke des Kontrastanstiegs:

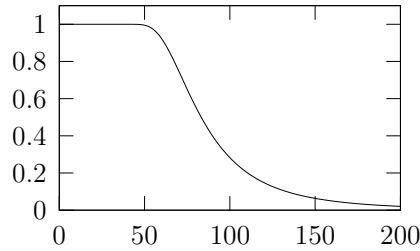


Abbildung 1: $g(s)$ für $\lambda = 7.5$

Für den Filter bedeutet dies, dass Kontrastunterschiede bis zu einer mit λ kontrollierbaren Stärke geglättet werden, wohingegen stärkere Kontraste erhalten bleiben, da kein Gegenfluss stattfindet. Es gibt viele Möglichkeiten wie man auf Grundlage der Information $|\nabla u_\sigma|^2$ im Fall isotroper Diffusion einen sinnvollen Diffusionstensor g aufstellen kann, vgl. [3, 4].

Es soll nun $n = 2$, d.h. ein zweidimensionales Bild angenommen werden. Bei einem anisotropen Filter soll der Diffusionsprozess, der durch den Diffusionstensor D beschrieben wird, ebenfalls von lokalen Bildinformationen abhängen. Hier kann jedoch auch die Richtung in die in einem Punkt ein Gegenfluss erzeugt wird (d.h. Glättung stattfindet) gesteuert werden, daher ist mehr lokale Bildinformation notwendig, als durch $|\nabla u_\sigma|^2$ gegeben. Zunächst soll folgendes Beispiel betrachtet werden:

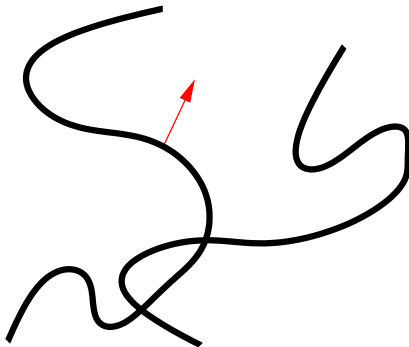


Abbildung 2: Kurven mit eingezeichneter Normale in einem Punkt

Es wurde die Richtung des stärksten Kontrastes in einem betrachteten Bildpunkt eingezeichnet. Wird ein isotroper Filter verwendet, so kann es sein, dass der Kontrast im Punkt so stark ist, dass keine Glättung mehr stattfindet, obwohl die Kurve selbst vielleicht verrauscht ist. Es wäre also in diesem Fall wünschenswert, eine Glättung in tangentialer Richtung an die Kurve zu erreichen. Für einen anisotropen Filter bietet es sich daher an, den Diffusionstensor D abhängig von der Richtung des stärksten Kontrastes ∇u_σ mit zugehöriger Stärke $|\nabla u_\sigma|^2$ und der dazu orthogonalen Richtung $(\nabla u_\sigma)^\perp$ aufzustellen. Zu diesem Zweck wird folgende Matrix betrachtet:

$$J_0(\nabla u_\sigma) := \nabla u_\sigma \otimes \nabla u_\sigma = \nabla u_\sigma (\nabla u_\sigma)^T$$

Man rechnet leicht nach, dass ∇u_σ ein Eigenvektor dieser Matrix zum Eigenwert $|\nabla u_\sigma|^2$ ist und $(\nabla u_\sigma)^\perp$ ein Eigenvektor zum Eigenwert 0 (wenn man bereit ist 0 vorübergehend als Eigenwert zu akzeptieren). Alle gewünschten (punktuellen) Informationen im Bildpunkt werden also durch die Eigenvektoren und Eigenwerte dieser Matrix wiedergegeben. Da man erneut an lokalen Informationen interessiert ist, mittelt man diese Informationen wieder durch komponentenweise Faltung mit einer Gaußkurve K_ρ und erhält:

$$J_\rho(\nabla u_\sigma) := K_\rho * J_0(\nabla u_\sigma)$$

Man rechnet leicht nach, dass diese Matrizen stets symmetrisch sind und bezeichnet

$$J_\rho = \begin{pmatrix} j_{11} & j_{12} \\ j_{12} & j_{22} \end{pmatrix}$$

als Strukturtensor des Bildes. Die Eigenvektoren und Eigenwerte stellen entsprechend Mittelungen der Kontrastinformationen aus J_0 über einer in ihrer Größe durch ρ steuerbaren Umgebung eines Bildpunktes dar. Man ordnet den Eigenvektor mit dem größeren Eigenwert entsprechend der Mittelung von ∇u_σ zu und nennt ihn Kontrastrichtung, da er die Richtungsinformationen der stärksten Kontraste mittelt, und den anderen Eigenvektor, den man Normalenrichtung nennt, analog $(\nabla u_\sigma)^\perp$. Eine einfache Rechnung ergibt für die Eigenvektoren v_1, v_2 von J_ρ :

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

$$v_2 \perp v_1$$

und für die zugehörigen Eigenwerte μ_i :

$$\mu_1 = j_{11} + j_{22} + \sqrt{(j_{11} - j_{22})^2 + 4j_{12}^2}$$

$$\mu_2 = j_{11} + j_{22} - \sqrt{(j_{11} - j_{22})^2 + 4j_{12}^2}$$

Zusammenfassend kann man sagen, dass die Wahl von σ beeinflusst, gegenüber welcher Umgebungsgröße der Strukturtensor invariant ist (irrelevante Informationen wie beispielsweise Rauschen) und die Wahl von ρ , wie groß die Umgebungen sind, mit denen lokale Orientierungsinformationen im Strukturtensor erfasst werden.

Der Diffusionstensor D wird nun abhängig vom Strukturtensor J_ρ als Träger lokaler Bildinformation beschrieben, ganz analog wie im isotropen Fall der skalare Diffusionstensor g abhängig von der lokalen Bildinformation $|\nabla u_\sigma|^2$ aufgestellt wurde. Da gewährleistet werden muss, dass D genau wie J_ρ die lokale Bildstruktur beschreibt, wird hier stets eine Matrix mit denselben Eigenvektoren v_1 und v_2 gewählt, die Eigenwerte werden jedoch angepasst um je nach lokaler Information den Filterprozess zu verstärken oder zu dämpfen. Von nun an sei stets vorausgesetzt, dass $\mu_1 \geq \mu_2$ gilt, d.h. dass v_1 die Kontrastrichtung darstellt.

Eine Möglichkeit, die neuen Eigenwerte λ_1, λ_2 von D zu wählen besteht darin, die bereits vom isotropen Fall bekannten Diffusionsfunktionen g zu benutzen, also

$$\lambda_1 := g_1(\mu_1)$$

$$\lambda_2 := g_2(\mu_2)$$

für zwei der bekannten Diffusionsfunktionen g zu setzen. In [2] wird vorgeschlagen

$$\lambda_1 := g(\mu_1)$$

$$\lambda_2 := 1$$

mit

$$g : \mathbb{R}_0^+ \longrightarrow \mathbb{R}, \quad g(s) = \begin{cases} 1 & s = 0 \\ 1 - \exp\left(\frac{-3.31488}{(s/\lambda)^4}\right) & s > 0 \end{cases}$$

zu verwenden. Dies entspricht in etwa der bereits oben angegebenen isotropen Filterfunktion in Kontrastrichtung (also dem bereits beschriebenen isotropen Filter) zusammen mit einer maximalen linearen Glättung in Normalenrichtung. Damit wäre beispielsweise im Eingangsbeispiel gewährleistet, dass der Filter Rauschen auf starken Kurven glättet, die Kurvenstruktur selbst jedoch nicht zerstört.

1.2.2 Diskretisierung im isotropen Fall mit dem additiven Operator Splitting

Die analytische Lösung u der Diffusionsgleichung besitzt viele für die Bildverarbeitung nützliche Eigenschaften, die zusammenfassend als Skalenraum bezeichnet werden. Diese Eigenschaften lassen sich analog auf die diskrete Formulierung übertragen und die Konvergenz von Diskretisierungen wird garantiert, indem nachgewiesen wird, dass die entsprechende Diskretisierung diese Skalenraumeigenschaften besitzt.

Für die Diskretisierung wird zunächst der Definitionsbereich Ω diskretisiert. Da Bilder in der Praxis als Rastergrafiken mit einem festen Gitter gegeben werden, wird auch hier der Diskretisierung ein festes, äquidistantes Gitter zugrunde gelegt. Man erhält also eine endliche Menge $\Omega' \subset \Omega$. Die Elemente von Ω' werden nun zunächst auf irgendeine Weise nummeriert ($\Omega' = \{x_1, \dots, x_N\}$) wodurch sich das gegebene Bild als Vektor

$$f \in \mathbb{R}^N$$

interpretieren lässt. Ein diskreter Filter berechnet nun stets im wesentlichen eine Folge $(u^{(k)})_{k \in \mathbb{N}}$ gemäß

$$\begin{aligned} u^{(0)} &= f \\ u^{(k+1)} &= Q(u^{(k)}) u^{(k)} \end{aligned}$$

wobei $Q(u^{(k)}) \in \mathbb{R}^{N \times N}$ ist. Das Ziel ist, $u_i^{(k)} \approx u(x_i, t_k)$ mit $t_k = k\tau$ zu erreichen, d.h. die Diskretisierung muss die Skalenraumeigenschaften besitzen. Man kann zeigen, dass dies stets der Fall ist, wenn die Matrizen Q eine bestimmte Menge an natürlichen Eigenschaften erfüllen (vgl. [1]) womit dann ein effektives Kriterium zur Verfügung steht mit dem die Konvergenz einer bestimmten Diskretisierung beurteilt werden kann.

Es bezeichne von nun an $\mathcal{N}_l(i)$ die Menge der Indizes der beiden Nachbarpixel von x_i entlang der l -ten Koordinatenachse. Als erste Idee für eine Diskretisierung erhält man durch diskretisieren auf der k -ten Zeitschicht ein explizites Verfahren:

$$\frac{u_i^{(k+1)} - u_i^{(k)}}{\tau} = \sum_{l=1}^n \sum_{j \in \mathcal{N}_l(i)} \frac{g_j^{(k)} - g_i^{(k)}}{2h_l^2} (u_j^{(k)} - u_i^{(k)})$$

Dabei approximieren die Terme $g_i^{(k)}$ den Diffusionstensor $g(|\nabla u_\sigma|^2)$ an der Stelle (x_i, t_k) . Dieses explizite Verfahren lässt sich sofort in die Form

$$\begin{aligned} u^{(0)} &= f \\ u^{(k+1)} &= Q(u^{(k)}) u^{(k)} \end{aligned}$$

umschreiben mit

$$\begin{aligned} Q(u^{(k)}) &= I + \tau A(u^{(k)}) \\ A(u^{(k)}) &= \sum_{l=1}^n A_l(u^{(k)}) \end{aligned}$$

wobei

$$a_{ij}^l = \begin{cases} \frac{g_j^{(k)} + g_i^{(k)}}{2h_l^2} & j \in \mathcal{N}_l(i) \\ - \sum_{j \in \mathcal{N}_l(i)} \frac{g_j^{(k)} + g_i^{(k)}}{2h_l^2} & j = i \\ 0 & \text{sonst} \end{cases}$$

Das explizite Verfahren konvergiert jedoch nur für kleine Zeitschrittweiten, daher geht man durch diskretisieren auf der $k+1$ ten Zeitschicht zu einem semi-impliziten Verfahren

$$\begin{aligned} u^{(0)} &= f \\ (I - \tau A(u^{(k)})) u^{(k+1)} &= u^{(k)} \end{aligned}$$

über, welches stets die Skalenraumeigenschaften garantiert. Das Problem hierbei ist jedoch die Lösung der mitunter recht großen Gleichungssysteme. Eine genauere Betrachtung der Matrix Q ergibt jedoch, dass sich diese aus n dünn besetzten Matrizen zusammensetzt. Die Idee besteht nun darin, dieses Gleichungssystem additiv in n einfachere Gleichungssysteme aufzuspalten und die Lösungen zu mitteln. Dieses als AOS (additive operator splitting) bezeichnete Verfahren wird gegeben durch

$$\begin{aligned} u^{(0)} &= f \\ u^{(k+1)} &= \frac{1}{n} \sum_{l=1}^n (I - n\tau A_l(u^{(k)}))^{-1} u^{(k)} \end{aligned}$$

Man kann sich überlegen, dass auch dieses Verfahren stets einen Skalenraum erzeugt und die gleiche Konvergenzgeschwindigkeit aufweist. Der Vorteil liegt nun darin, dass in den einzelnen Gleichungssystemen

$$v_l^{(k+1)} = \left(I - n\tau A_l(u^{(k)}) \right)^{-1} u^{(k)} \quad l = 1, \dots, n$$

in jeder Zeile der Matrix nur 3 Einträge besetzt sind. Durch eine geschickte Wahl der Nummerierung $\Omega' = \{x_1, \dots, x_N\}$ kann man erreichen dass diese Matrizen stets tridiagonal verschoben sind, so dass die Lösung in linearer Zeit mit dem Thomas-Algorithmus erfolgen kann.

Für $n = 2$ wurde diese Indizierung von Thomas Künzel in [1] beschrieben. Um den isotropen Filter im Rahmen dieses Fortgeschrittenenpraktikums auf n -dimensionale Bilder auszudehnen, war es notwendig, allgemeine Formeln für diese Indizierung zu finden. Darauf wird in Abschnitt 2.1 eingegangen werden.

Wie das AOS auf den Fall eines anisotropen Filters ausgedehnt werden kann, wird im Abschnitt 2.3 beschrieben.

2 Implementierung von Diffusionsfiltern

2.1 Isotroper Filter für n-dimensionale Bilder

Wird in n Dimensionen gefiltert, so sind in jedem Iterationsschritt für das AOS n Gleichungssysteme zu lösen. Es muss genau darauf geachtet werden, auf welche Weise ein n -dimensionales Bild in einem Vektor angeordnet wird, damit alle diese Gleichungssysteme Tridiagonalgestalt aufweisen (um sie effizient mit dem Thomas-Algorithmus lösen zu können). Dazu wird nun die Situation aus der Diplomarbeit ([1]) in direkter Weise verallgemeinert.

2.1.1 Anordnung eines n-dimensionalen Bildes als Vektor

Ein n -dimensionales Bild wird gegeben durch eine Abbildung

$$f : \prod_{k=1}^n I_k \longrightarrow [0, 1]$$

wobei $I_k = \{0, \dots, n_k\} \subset \mathbb{N} \ \forall 1 \leq k \leq n$ d.h.

$$\prod_{k=1}^n I_k \subset \mathbb{N}^n$$

Zur Abkürzung sei von nun an $N_k := n_k + 1$ ($= |I_k|$) $\forall 1 \leq k \leq n$ gesetzt. Als Anordnung eines solchen Bildes als Vektor liegt es nahe folgende Abbildung zu verwenden, welche die Dimensionen sukzessive verschachtelt, von der höchsten ausgehend:

$$\begin{aligned} \Phi : \prod_{k=1}^n I_k &\longrightarrow \left\{ 0, \dots, \left(\prod_{k=1}^n N_k \right) - 1 \right\} \\ (x_1, \dots, x_n) &\longmapsto \sum_{j=1}^n \left(x_j \prod_{k=1}^{j-1} N_k \right) \end{aligned}$$

Proposition:

Φ ist wohldefiniert und bijektiv.

Beweis:

Da $\forall 1 \leq j \leq n : 0 \leq x_j \leq n_j = N_j - 1$ folgt:

$$\begin{aligned} \sum_{j=1}^n \left(x_j \prod_{k=1}^{j-1} N_k \right) &\geq \sum_{j=1}^n \left(0 \prod_{k=1}^{j-1} N_k \right) = 0 \\ \sum_{j=1}^n \left(x_j \prod_{k=1}^{j-1} N_k \right) &\leq \sum_{j=1}^n \left((N_j - 1) \prod_{k=1}^{j-1} N_k \right) = \sum_{j=1}^n \left(\prod_{k=1}^j N_k - \prod_{k=1}^{j-1} N_k \right) = \prod_{k=1}^n N_k - \prod_{k=1}^{n-1} N_k = \prod_{k=1}^n N_k - 1 \end{aligned}$$

Damit ist die Wohldefiniertheit gezeigt. Da

$$\left| \prod_{k=1}^n I_k \right| = \left| \left\{ 0, \dots, \prod_{k=1}^n N_k - 1 \right\} \right| = \prod_{k=1}^n N_k$$

genügt es für die Bijektivität die Injektivität zu zeigen. Dazu nehmen wir an, dass für $(x_1, \dots, x_n), (y_1, \dots, y_n) \in \prod_{k=1}^n I_k$ gelte: $\Phi((x_1, \dots, x_n)) = \Phi((y_1, \dots, y_n))$ Es ist also:

$$\begin{aligned} (*) \quad \sum_{j=1}^n \left(x_j \prod_{k=1}^{j-1} N_k \right) &= \sum_{j=1}^n \left(y_j \prod_{k=1}^{j-1} N_k \right) \\ \iff \sum_{j=1}^{n-1} \left(x_j \prod_{k=1}^{j-1} N_k \right) + x_n \prod_{k=1}^{n-1} N_k &= \sum_{j=1}^{n-1} \left(y_j \prod_{k=1}^{j-1} N_k \right) + y_n \prod_{k=1}^{n-1} N_k \\ \iff \sum_{j=1}^{n-1} \left(x_j \prod_{k=1}^{j-1} N_k \right) &= \sum_{j=1}^{n-1} \left(y_j \prod_{k=1}^{j-1} N_k \right) + (y_n - x_n) \prod_{k=1}^{n-1} N_k \end{aligned}$$

Völlig analog zu den Abschätzungen bei der Wohldefiniertheit erhalten wir:

$$0 \leq \sum_{j=1}^{n-1} \left(x_j \prod_{k=1}^{j-1} N_k \right) \leq \prod_{k=1}^{n-1} N_k - 1$$

$$0 \leq \sum_{j=1}^{n-1} \left(y_j \prod_{k=1}^{j-1} N_k \right) \leq \prod_{k=1}^{n-1} N_k - 1$$

Angenommen $y_n \neq x_n$. Fall 1: $y_n - x_n < 0$ Dann erhalten wir:

$$\begin{aligned} 0 &\leq \sum_{j=1}^{n-1} \left(x_j \prod_{k=1}^{j-1} N_k \right) = \sum_{j=1}^{n-1} \left(y_j \prod_{k=1}^{j-1} N_k \right) + (y_n - x_n) \prod_{k=1}^{n-1} N_k \leq \prod_{k=1}^{n-1} N_k - 1 + (-1) \prod_{k=1}^{n-1} N_k \\ &= -1 \quad \# \end{aligned}$$

Fall 2: $y_n - x_n > 0$. Dann folgt aber:

$$\begin{aligned} \prod_{k=1}^{n-1} N_k - 1 &\geq \sum_{j=1}^{n-1} \left(x_j \prod_{k=1}^{j-1} N_k \right) = \sum_{j=1}^{n-1} \left(y_j \prod_{k=1}^{j-1} N_k \right) + (y_n - x_n) \prod_{k=1}^{n-1} N_k \\ &\geq 0 + \prod_{k=1}^{n-1} N_k \\ &= \prod_{k=1}^{n-1} N_k \quad \# \end{aligned}$$

Also muss in jedem Fall $y_n = x_n$ gelten. Damit formt sich (*) jedoch um zu

$$\sum_{j=1}^{n-1} \left(x_j \prod_{k=1}^{j-1} N_k \right) = \sum_{j=1}^{n-1} \left(y_j \prod_{k=1}^{j-1} N_k \right)$$

und durch weitere Anwendung des Arguments von oben erhalten wir schließlich $(x_1, \dots, x_n) = (y_1, \dots, y_n)$. \square

Es wird nun eine Eigenschaft dieser Abbildung Φ bewiesen welche später garantieren wird, dass die Gleichungssysteme für das AOS in allen Dimensionen verschobene Tridiagonalmatrizen sind.

Proposition:

Sei $1 \leq \ell \leq n$ und $(x_1, \dots, x_n) \in \prod_{k=1}^n I_k$. Dann gilt

$$\Phi((x_1, \dots, x_\ell + 1, \dots, x_n)) = \Phi((x_1, \dots, x_\ell, \dots, x_n)) + \prod_{k=1}^{\ell-1} N_k \quad \text{für } 0 \leq x_\ell < n_\ell$$

$$\Phi((x_1, \dots, x_\ell - 1, \dots, x_n)) = \Phi((x_1, \dots, x_\ell, \dots, x_n)) - \prod_{k=1}^{\ell-1} N_k \quad \text{für } 0 < x_\ell \leq n_\ell$$

Beweis:

Direktes Aufschreiben liefert:

$$\begin{aligned} \Phi((x_1, \dots, x_\ell + 1, \dots, x_n)) &= \sum_{\substack{j=1 \\ j \neq \ell}}^n \left(x_j \prod_{k=1}^{j-1} N_k \right) + (x_\ell + 1) \prod_{k=1}^{\ell-1} N_k = \sum_{j=1}^n \left(x_j \prod_{k=1}^{j-1} N_k \right) + \prod_{k=1}^{\ell-1} N_k \\ &= \Phi((x_1, \dots, x_\ell, \dots, x_n)) + \prod_{k=1}^{\ell-1} N_k \end{aligned}$$

Die zweite Gleichung folgt völlig analog. \square

Für die Implementierung ist es wichtig, anhand eines Index des Bildvektors ermitteln zu können, ob der zugehörige Pixel ein Randpixel bezüglich einer Dimension ℓ ist oder ob nicht. Hierfür gibt die folgende Proposition ein handliches Kriterium:

Proposition:

Sei $1 \leq \ell \leq n$. Für $c = \Phi((x_1, \dots, x_n))$ gilt:

$$x_\ell = 0 \iff \left(c \bmod \prod_{k=1}^{\ell} N_k \right) < \prod_{k=1}^{\ell-1} N_k$$

$$x_\ell = n_\ell \iff \left(c \bmod \prod_{k=1}^{\ell} N_k \right) \geq \left(\prod_{k=1}^{\ell} N_k \right) - \left(\prod_{k=1}^{\ell-1} N_k \right)$$

Beweis:

Nach Voraussetzung gilt

$$c = \sum_{j=1}^n \left(x_j \prod_{k=1}^{j-1} N_k \right)$$

Wegen

$$\frac{\sum_{j=1}^{\ell} \left(x_j \prod_{k=1}^{j-1} N_k \right)}{\prod_{k=1}^{\ell} N_k} \leq \frac{\sum_{j=1}^{\ell} \left((N_j - 1) \prod_{k=1}^{j-1} N_k \right)}{\prod_{k=1}^{\ell} N_k} = \frac{\sum_{j=1}^{\ell} \left(\prod_{k=1}^j N_k - \prod_{k=1}^{j-1} N_k \right)}{\prod_{k=1}^{\ell} N_k} = \frac{\left(\prod_{k=1}^{\ell} N_k - 1 \right)}{\prod_{k=1}^{\ell} N_k}$$

und

$$\frac{\sum_{j=\ell+1}^n \left(x_j \prod_{k=1}^{j-1} N_k \right)}{\prod_{k=1}^{\ell} N_k} = \sum_{j=\ell+1}^n \left(x_j \prod_{k=\ell+1}^{j-1} N_k \right)$$

erhalten wir:

$$\left(c \bmod \prod_{k=1}^{\ell} N_k \right) = \sum_{j=1}^{\ell} \left(x_j \prod_{k=1}^{j-1} N_k \right)$$

Ist also $x_\ell = 0$ so folgt mit der gewohnten Abschätzung:

$$\left(c \bmod \prod_{k=1}^{\ell} N_k \right) = \sum_{j=1}^{\ell} \left(x_j \prod_{k=1}^{j-1} N_k \right) = \sum_{j=1}^{\ell-1} \left(x_j \prod_{k=1}^{j-1} N_k \right) \leq \prod_{k=1}^{\ell-1} N_k - 1 < \prod_{k=1}^{\ell-1} N_k$$

Ist umgekehrt $\left(c \bmod \prod_{k=1}^{\ell} N_k \right) < \prod_{k=1}^{\ell-1} N_k$ vorausgesetzt, so erhalten wir analog:

$$\begin{aligned} \prod_{k=1}^{\ell-1} N_k &> \left(c \bmod \prod_{k=1}^{\ell} N_k \right) = \sum_{j=1}^{\ell} \left(x_j \prod_{k=1}^{j-1} N_k \right) = \sum_{j=1}^{\ell-1} \left(x_j \prod_{k=1}^{j-1} N_k \right) + x_\ell \prod_{k=1}^{\ell-1} N_k \\ &\geq 0 + x_\ell \prod_{k=1}^{\ell-1} N_k \end{aligned}$$

woraus $x_\ell = 0$ folgt. Damit ist die erste Äquivalenz gezeigt.

Die zweite kann völlig analog nachgerechnet werden. □

Es sollen nun alle Ergebnisse über Φ noch einmal übersichtlich zusammengefasst werden, wobei folgende Notation aus der Arbeit von Thomas Künzel aufgegriffen wird: Für $c \in \left\{ 0, \dots, \prod_{k=1}^n N_k - 1 \right\}$ bezeichne

$$\mathcal{N}_\ell(c)$$

die Menge der Indizes der Nachbarpixel von c bezüglich der Dimension ℓ .

Satz:

Es sei $k(\ell) := \prod_{k=1}^{\ell} N_k$. Für $c \in \left\{0, \dots, \prod_{k=1}^n N_k - 1\right\}$ gilt dann

$$\mathcal{N}_{\ell}(c) = \begin{cases} \{c + k(\ell - 1)\} & \text{falls } (c \bmod k(\ell)) < k(\ell - 1) \\ \{c - k(\ell - 1)\} & \text{falls } (c \bmod k(\ell)) \geq k(\ell) - k(\ell - 1) \\ \{c + k(\ell - 1), c - k(\ell - 1)\} & \text{sonst} \end{cases}$$

2.1.2 Die Gleichungssysteme für das AOS

Als Diskretisierung wurde in der Arbeit von Thomas Künzel das AOS-Verfahren (additive operator splitting) verwendet. Dies wird gegeben durch:

$$u^{(0)} := f$$

$$u^{(k+1)} = \frac{1}{n} \sum_{\ell=1}^n \left(I - n\tau A_{\ell}(u^{(k)}) \right)^{-1} u^{(k)}$$

Es sind in n Dimensionen also n Gleichungssysteme

$$\left(I - n\tau A_{\ell}(u^{(k)}) \right) v_{\ell}^{(k+1)} = u^{(k)}$$

zu lösen. Wie bereits erwähnt wurde die Indizierung Φ der Pixel des n -dimensionalen Bildes f derart gewählt, dass mit den $A_{\ell}(u^{(k)})$ und damit auch mit den zu lösenden Gleichungssystemen verschobene Tridiagonalmatrizen entstehen. Es wird nun kurz beschrieben, wie diese Matrizen zu besetzen sind. Nach der Arbeit von Thomas Künzel sind die Matrizen gegeben durch:

$$A_{\ell}(u^{(k)}) = (a_{ij}^{\ell}) \quad a_{ij}^{\ell} = \begin{cases} \frac{g_i^{(k)} + g_j^{(k)}}{2h_{\ell}^2} & \text{falls } j \in \mathcal{N}_{\ell}(i) \\ - \sum_{m \in \mathcal{N}_{\ell}(i)} \frac{g_i^{(k)} + g_m^{(k)}}{2h_{\ell}^2} & \text{falls } j = i \\ 0 & \text{sonst} \end{cases}$$

für einen gewissen Vektor $g^{(k)} \in \mathbb{R}^{\left(\prod_{k=1}^n N_k\right)}$ der die Diffusivitätsfunktion g in allen Pixelpositionen aus dem Bild approximiert, d.h.

$$g(x_1, \dots, x_n, \tau k) \approx g_{\Phi((x_1, \dots, x_n))}^{(k)}$$

Unter Verwendung des Satzes kann damit die Besetzung explizit angegeben werden, wobei nur die von 0 verschiedenen Einträge notiert werden.

Falls $(i \bmod k(\ell)) < k(\ell - 1)$ gilt für die Zeile i der Matrix $A_{\ell}(u^{(k)})$:

$$a_{i,i}^{\ell} = -\frac{g_i^{(k)} + g_{(i+k(\ell-1))}^{(k)}}{2h_{\ell}^2} \quad a_{i,(i+k(\ell-1))}^{\ell} = \frac{g_i^{(k)} + g_{(i+k(\ell-1))}^{(k)}}{2h_{\ell}^2} \quad (= -a_{i,i}^{\ell})$$

Falls $(i \bmod k(\ell)) \geq k(\ell) - k(\ell - 1)$ folgt für die Zeile i :

$$a_{i,i}^{\ell} = -\frac{g_i^{(k)} + g_{(i-k(\ell-1))}^{(k)}}{2h_{\ell}^2} \quad a_{i,(i-k(\ell-1))}^{\ell} = \frac{g_i^{(k)} + g_{(i-k(\ell-1))}^{(k)}}{2h_{\ell}^2} \quad (= -a_{i,i}^{\ell})$$

Ansonsten gilt für die Zeile i :

$$a_{i,i}^{\ell} = -\frac{g_{(i-k(\ell-1))}^{(k)} + 2g_i^{(k)} + g_{(i+k(\ell-1))}^{(k)}}{2h_{\ell}^2} \quad a_{i,(i-k(\ell-1))}^{\ell} = \frac{g_i^{(k)} + g_{(i-k(\ell-1))}^{(k)}}{2h_{\ell}^2} \quad a_{i,(i+k(\ell-1))}^{\ell} = \frac{g_i^{(k)} + g_{(i+k(\ell-1))}^{(k)}}{2h_{\ell}^2}$$

Man sieht nun direkt: $A_{\ell}(u^{(k)})$ ist mit der Indizierung Φ der Pixel eine Tridiagonalmatrix in der die beiden Nebendiagonalen um $k(\ell - 1)$ verschoben sind. Damit lassen sich die Gleichungssysteme im allgemeinen Fall ebenfalls mit dem effizienten Thomas-Algorithmus lösen.

Man beachte, dass im Fall $n = 2$ die Formeln für die beiden Matrizen $A_1(u^{(k)})$ und $A_2(u^{(k)})$ von oben gerade in die Formeln aus der Arbeit von Thomas Künzel übergehen.

2.1.3 Optimierte Randpixelerkennung

Im Fall $\ell = 1$ und $\ell = n$ kann aus den allgemeinen Formeln eine einfachere Randpixelerkennung abgeleitet werden, die zu optimierten Codes führt. Zunächst soll der Fall $\ell = 1$ behandelt werden.

Proposition:

Sei $c \in \left\{0, \dots, \prod_{k=1}^n N_k - 1\right\}$ gegeben, so dass auch $c + 1 \in \left\{0, \dots, \prod_{k=1}^n N_k - 1\right\}$. Dann gilt:

$$c = \Phi((n_1, x_2, \dots, x_n)) \iff c + 1 = \Phi((0, y_2, \dots, y_n))$$

d.h. auf einen “großen” Randpixel bezüglich Dimension $\ell = 1$ folgt stets ein “kleiner” Randpixel dieser Dimension.

Beweis:

Es gilt:

$$\begin{aligned} c = \Phi((n_1, x_2, \dots, x_n)) &\iff \left(c \bmod \prod_{k=1}^1 N_k\right) \geq \left(\prod_{k=1}^1 N_k\right) - \left(\prod_{k=1}^{1-1} N_k\right) \iff (c \bmod N_1) \geq N_1 - 1 \\ &\iff (c \bmod N_1) = N_1 - 1 \\ &\iff (c + 1 \bmod N_1) = 0 \\ c + 1 = \Phi((0, y_2, \dots, y_n)) &\iff \left(c \bmod \prod_{k=1}^1 N_k\right) < \prod_{k=1}^{1-1} N_k \iff (c \bmod N_1) < 1 \\ &\iff (c \bmod N_1) = 0 \end{aligned}$$

Also folgt die Behauptung. □

In Dimension 1 kann man die Randpixelerkennung also so gestalten, dass man die Indizes $c = 0$ (“kleiner” Randpixel) und $c = \prod_{k=1}^n N_k - 1$ (“großer” Randpixel) getrennt behandelt und für die restlichen nur noch auf große Randpixel prüft, denn nach der Proposition folgen die kleinen Randpixel gerade auf die großen. In Dimension n ist die Situation sogar noch einfacher:

Proposition:

Sei $c \in \left\{0, \dots, \prod_{k=1}^n N_k - 1\right\}$ gegeben. Dann gilt:

$$\mathcal{N}_n(c) = \begin{cases} \{c + k(n-1)\} & \text{falls } c < k(n-1) \\ \{c - k(n-1)\} & \text{falls } c \geq k(n) - k(n-1) \\ \{c + k(n-1), c - k(n-1)\} & \text{sonst} \end{cases}$$

Beweis:

Da $c < \prod_{k=1}^n N_k$ nach Voraussetzung erhalten wir

$$(c \bmod k(n)) = \left(c \bmod \prod_{k=1}^n N_k\right) = c$$

und damit folgt die Behauptung direkt aus dem Satz. □

In dieser Dimension lassen sich Randpixel also ohne weitere Rechnung direkt am Index c ablesen.

2.1.4 Beispiele

Um interessantere Beispiele bringen zu können, wird in diesem Abschnitt das Filtern von RGB-Bildern vorweggenommen. Für genauere Details hierzu, vgl. den nächsten Abschnitt.

Als ein erstes Beispiel soll die Rekonstruktion verlorener Information betrachtet werden. Folgendes ist das Originalbild, wobei man sich die Bilder von links nach rechts entlang der z -Achse angeordnet vorstellen muss:



Abbildung 3: Originalbild, es werden die Einzelbilder entlang der z -Achse gezeigt

Für Anwendungen ist dies sicherlich ein unrealistisches Extrembeispiel. Dennoch liefert der 3D-Filter ein recht gutes Resultat, welches aufzeigt, dass für praxisnahe Beispiele eine Rekonstruktion verlorener Information durchaus möglich ist:

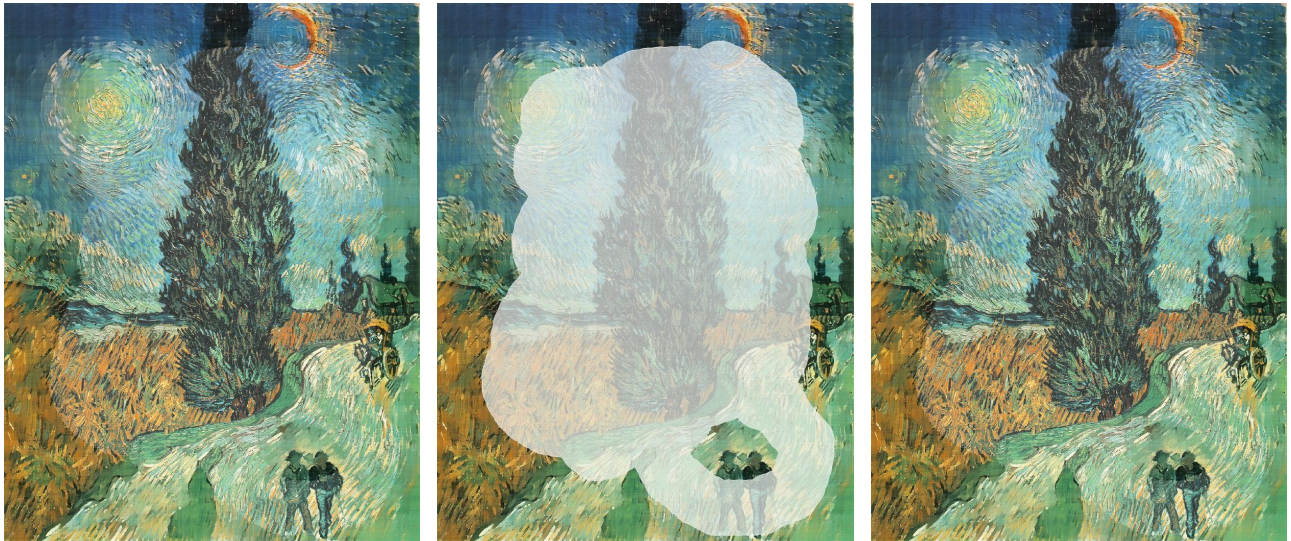


Abbildung 4: Gefiltertes Bild $\tau = 1 \cdot 10^{-1}$, $\sigma = 1 \cdot 10^{-4}$, $k = 2$, $\lambda = 35$ (Künzel)
(RGB-Filter: Normierte Gradienten)

Als zweites Beispiel soll demonstriert werden, wie mithilfe des 3D-Filters eine bessere Rauschreduktion möglich ist, wenn verschieden verrauschte Versionen desselben Bildes vorhanden sind. Zunächst demonstriert die folgende Grafik das Resultat eines 2D-Filters (Hinweis: Das Rauschen ist ggf. erst nach Vergrößerung des Dokumentes sichtbar):

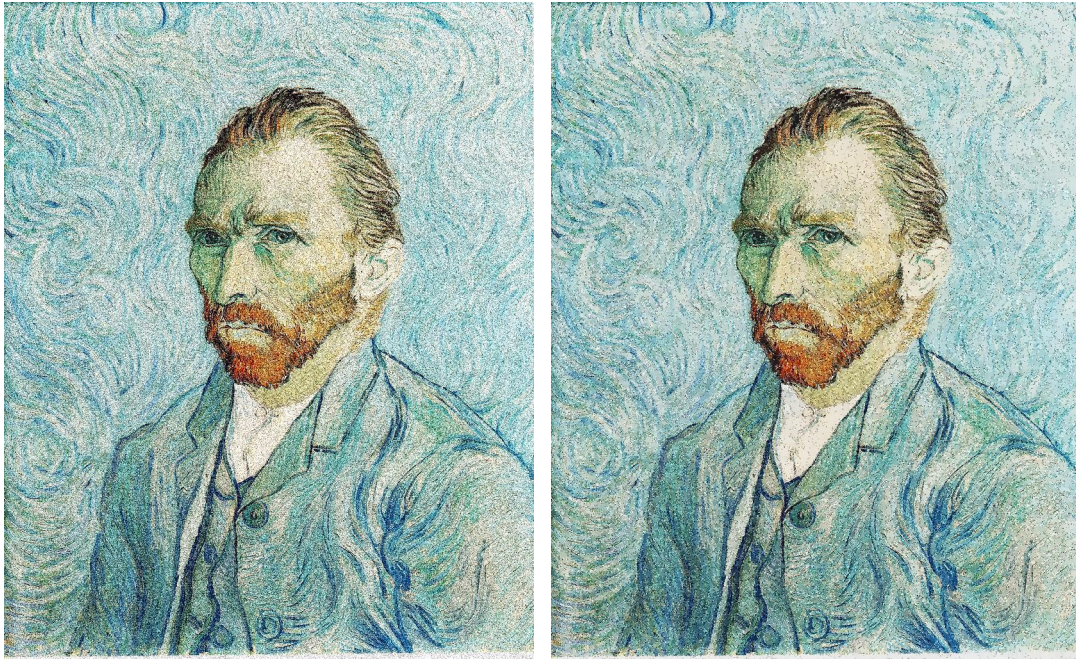


Abbildung 5: Linkes Bild: Verrauschtes Original, Rechtes Bild: 2D-Filter
 $\tau = 5 \cdot 10^{-1}$, $\sigma = 1 \cdot 10^{-4}$, $k = 10$, $\lambda = 12$ (Künzel) (RGB – Filter : Normierte Gradienten)

Für eine bessere Rauschreduktion durch den 3D-Filter sei angenommen, dass drei unterschiedlich verrauschte Versionen des Originalbildes verfügbar sind:

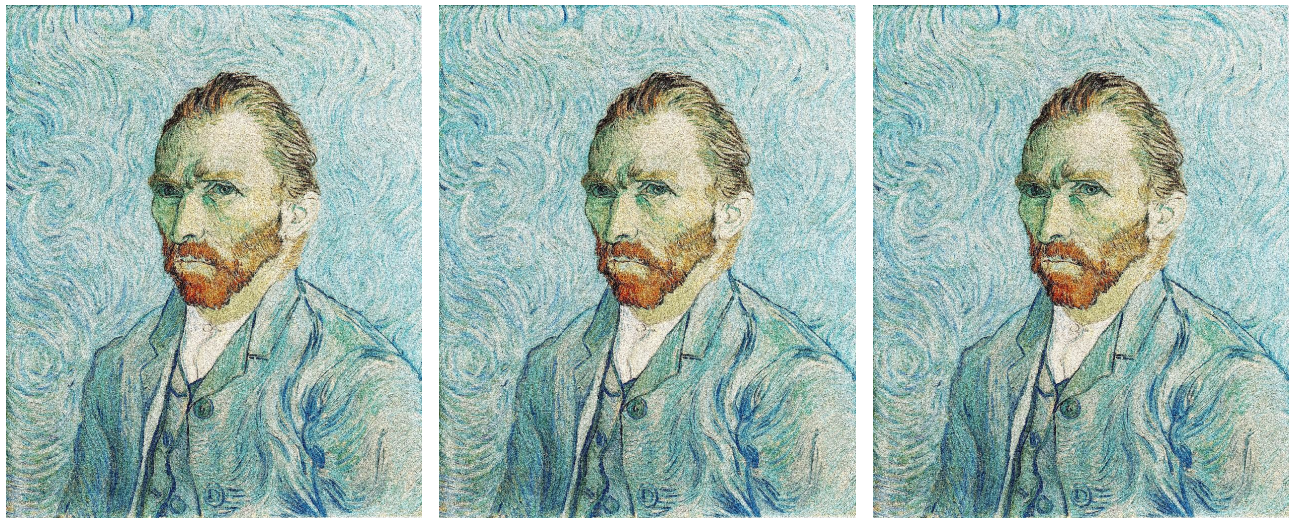


Abbildung 6: Originales 3D-Bild, es werden die Einzelbilder entlang der z -Achse gezeigt

Der 3D-Filter liefert dann folgendes Ergebnis:

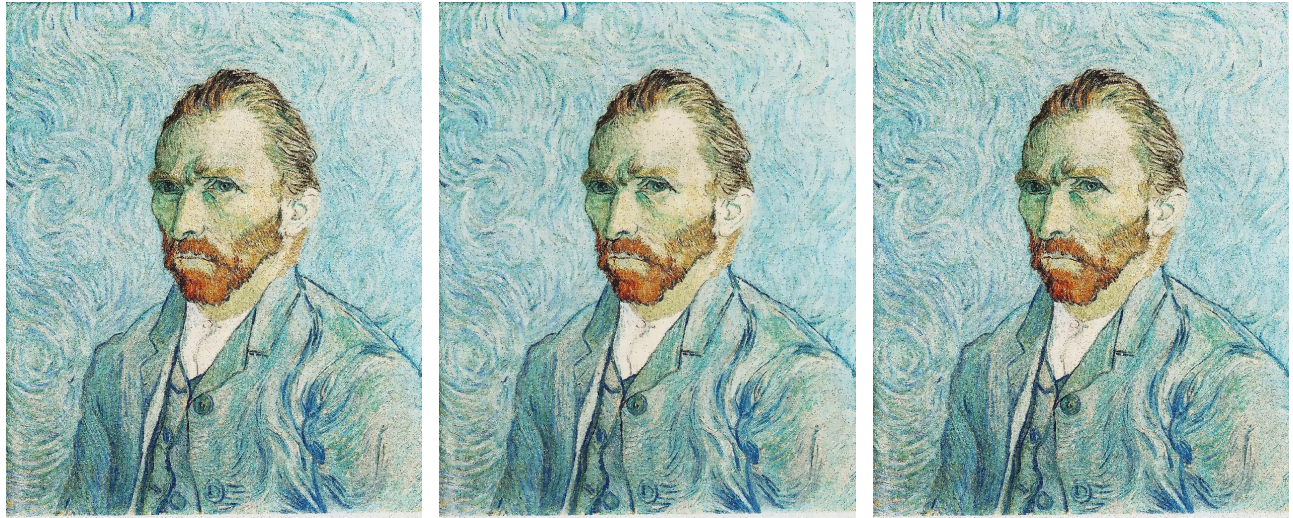


Abbildung 7: Gefiltert, $\tau = 5 \cdot 10^{-1}$, $\sigma = 1 \cdot 10^{-4}$, $k = 5$, $\lambda = 35$ (Künzel)(RGB-Filter: Normierte Gradienten)

Man kann deutlich erkennen, dass das Bild welches auf der z -Achse in der Mitte liegt, d.h. zwei Nachbarn und damit am meisten Information beim Filtern besitzt, das beste Filterresultat liefert. Stellt man dieses nun dem 2D-Filter gegenüber kann man leicht erkennen, dass der 3D-Filter ein besseres Resultat liefert:

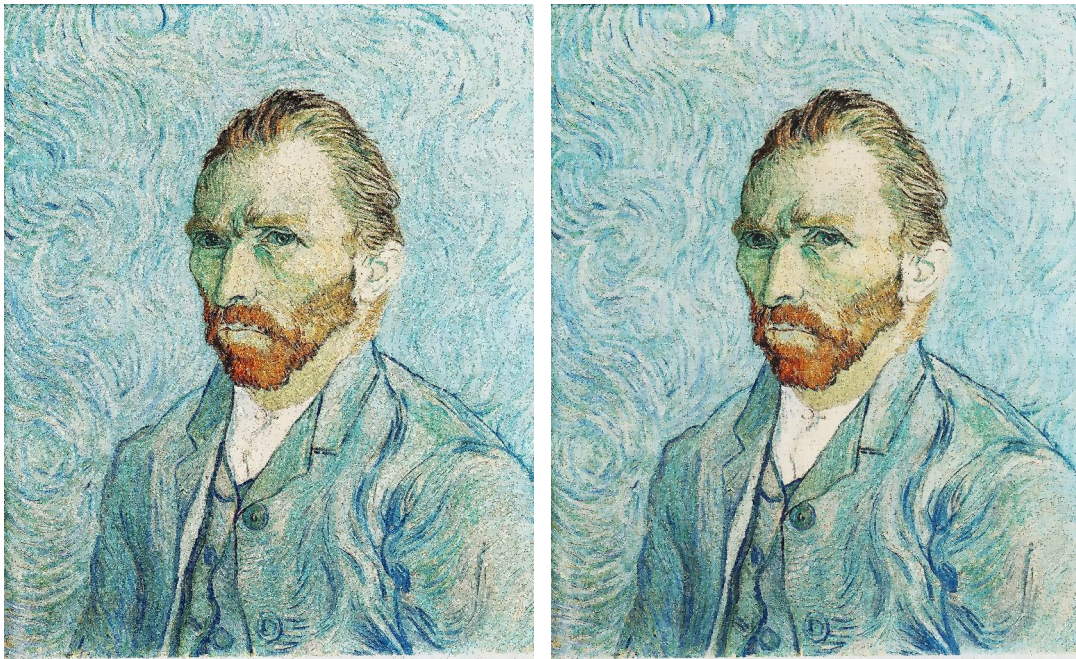


Abbildung 8: Linkes Bild: 2D-Filter, Rechtes Bild: 3D-Filter ($z = 1$)

2.2 Filtern von Farbbildern

2.2.1 Vorstellung der verschiedenen Methoden

Farbbilder bestehen nicht mehr nur aus einer Bildebene interpretiert als Grauwerte sondern aus drei Bildebenen jeweils interpretiert als Rot-, Grün- und Blauwerte. Wie man derartige Farbbilder filtert hängt stark davon ab, welche Interpretation diesen drei Farbebenen in der Anwendung zukommt d.h. wie sie interagieren sollen. In diesem Bericht kann stets nur ein ästhetischer Standpunkt vertreten werden. Als eine erste Idee kann man eine unabhängige Filterung vorsehen, in der die drei Farbebenen keinen Einfluss aufeinander nehmen, d.h. jede Farbebene wird wie

ein Graubild behandelt. Dem Filtervorgang liegen dann die Differentialgleichungen

$$\begin{aligned}\partial_t u^R &= \operatorname{div} \left(g \left(|\nabla u_\sigma^R|^2 \right) \nabla u^R \right) \\ \partial_t u^G &= \operatorname{div} \left(g \left(|\nabla u_\sigma^G|^2 \right) \nabla u^G \right) \\ \partial_t u^B &= \operatorname{div} \left(g \left(|\nabla u_\sigma^B|^2 \right) \nabla u^B \right)\end{aligned}$$

zugrunde. Diese Form des Filterns eignet sich, wenn die drei Farbebenen beispielsweise Teilchen modellieren, die nicht miteinander wechselwirken sollen. Von einem ästhetischen Standpunkt betrachtet ist diese Methode nicht optimal, da Kanten in den einzelnen Farbebenen unterschiedlich stark erhalten bleiben können was zu falschen Farben führt.

Um diese Probleme zu vermeiden müssen die Farbebenen während des Filtervorgangs Einfluss aufeinander nehmen können. Eine Möglichkeit hierfür bestand darin, den Diffusionstensor g in jeder Iteration aus einer Mittelung der drei Farbebenen zu berechnen und mit diesem Diffusionstensor dann die nächste Iteration der drei Farbebenen zu bestimmen. Der Filtervorgang basiert dann auf den Differentialgleichungen

$$\begin{aligned}\partial_t u^R &= \operatorname{div} \left(g \left(\left| \nabla \left(\frac{u_\sigma^R + u_\sigma^G + u_\sigma^B}{3} \right) \right|^2 \right) \nabla u^R \right) \\ \partial_t u^G &= \operatorname{div} \left(g \left(\left| \nabla \left(\frac{u_\sigma^R + u_\sigma^G + u_\sigma^B}{3} \right) \right|^2 \right) \nabla u^G \right) \\ \partial_t u^B &= \operatorname{div} \left(g \left(\left| \nabla \left(\frac{u_\sigma^R + u_\sigma^G + u_\sigma^B}{3} \right) \right|^2 \right) \nabla u^B \right)\end{aligned}$$

Hier werden die Gradienten für den Diffusionstensor also aus dem arithmetischen Mittel der drei Farbebenen in der jeweiligen Zeitebene berechnet. Optisch führt dies zu erheblich besseren Resultaten.

Es wurde schließlich noch eine dritte Möglichkeit implementiert, welche in [5] vorgeschlagen wird. Hierbei ersetzt die Summe der normierten Gradienten der einzelnen Farbebenen die lokale Bildinformation $|\nabla u_\sigma|^2$, d.h. dem Filtervorgang werden die Differentialgleichungen

$$\begin{aligned}\partial_t u^R &= \operatorname{div} \left(g \left(|\nabla u_\sigma^R|^2 + |\nabla u_\sigma^G|^2 + |\nabla u_\sigma^B|^2 \right) \nabla u^R \right) \\ \partial_t u^G &= \operatorname{div} \left(g \left(|\nabla u_\sigma^R|^2 + |\nabla u_\sigma^G|^2 + |\nabla u_\sigma^B|^2 \right) \nabla u^G \right) \\ \partial_t u^B &= \operatorname{div} \left(g \left(|\nabla u_\sigma^R|^2 + |\nabla u_\sigma^G|^2 + |\nabla u_\sigma^B|^2 \right) \nabla u^B \right)\end{aligned}$$

zugrunde gelegt. Da bei allen drei Methoden unterschiedlich große Gradienten entstehen, ist es schwer Ergebnisse vergleichbar gegenüberzustellen (da die Parameter nicht exakt passend gewählt werden können). Es sollte von Fall zu Fall durch Tests oder Überlegungen entschieden werden, welche Filtervariante vorzuziehen ist. Aus ästhetischer Sicht zu bevorzugen ist die Variante mit normierten Gradienten.

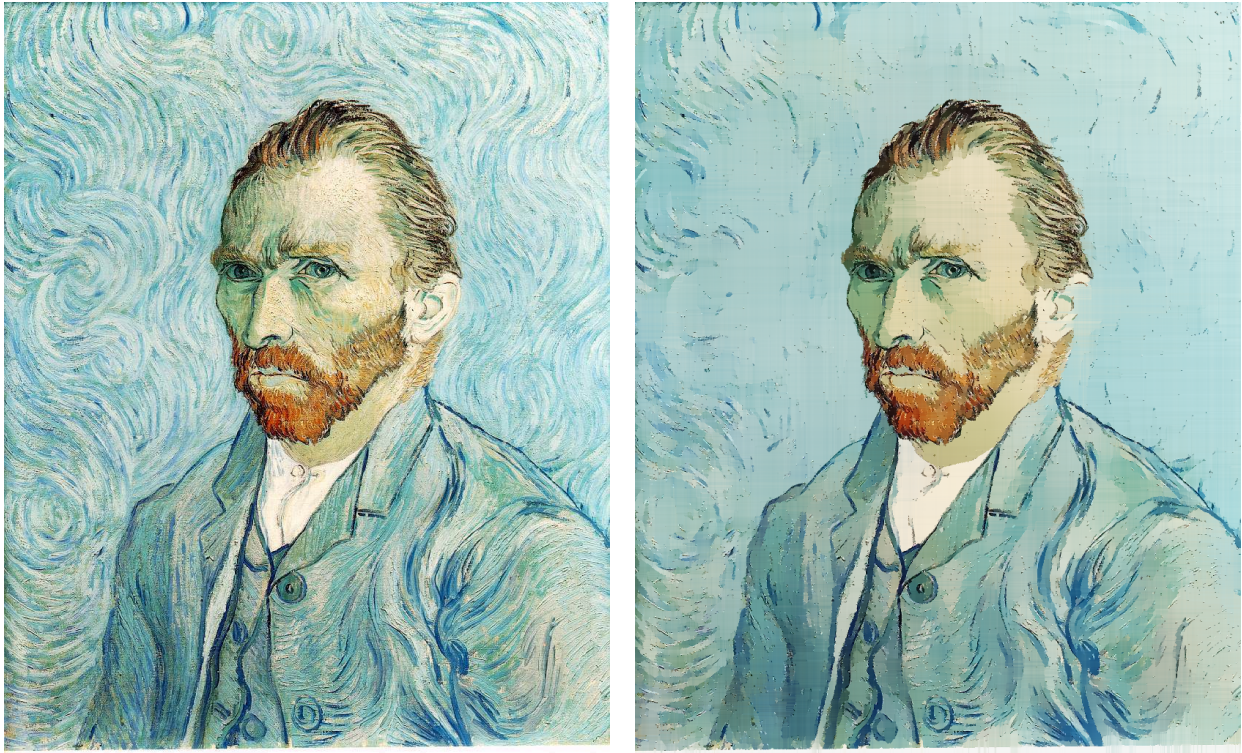


Abbildung 9: Beispiel für ein RGB-Filtervorgang: Als Methode wurden die normierten Gradienten benutzt ($\tau = 5 \cdot 10^{-2}$, $\sigma = 1 \cdot 10^{-4}$, 6 Iteration mit Künzel-Diffusion, $\lambda = 17$)

2.2.2 Gegenüberstellung der verschiedenen Methoden

Das Originalbild wurde mit einem normalverteilten Rauschen der Varianz 50 in allen 3 Farbebenen getrennt verrauscht. Unabhängiges Filtern der drei Farbebenen führt zu folgendem Resultat:

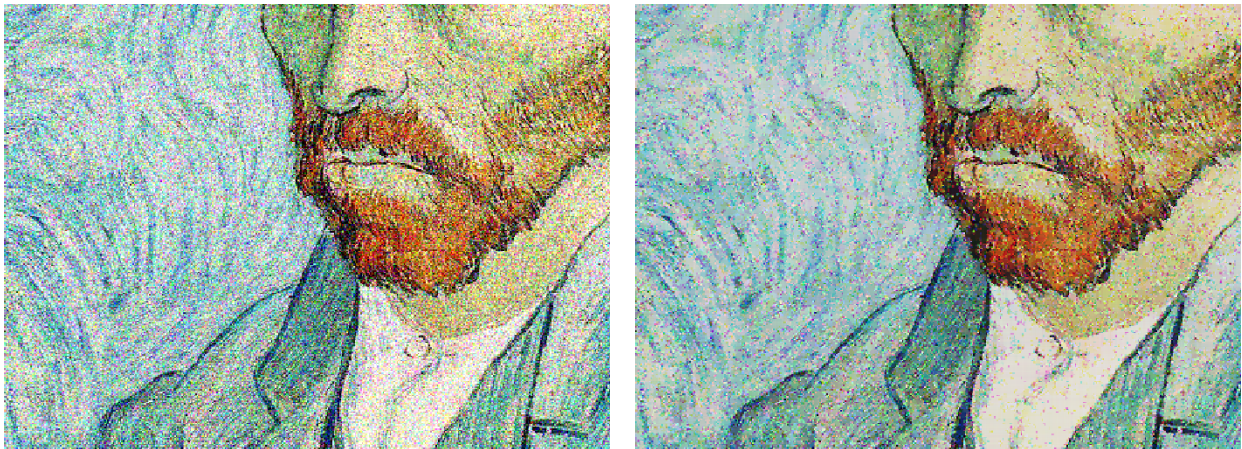


Abbildung 10: Links: Original mit Rauschen, Rechts: Gefiltert: $\tau = 5 \cdot 10^{-1}$, $\sigma = 1 \cdot 10^{-4}$, $k = 9$, $\lambda = 8$ (Künzel)

Man kann deutlich erkennen, dass noch einige starke verrauschte Stellen vorhanden sind und die Farben wirken insgesamt unecht, beispielsweise das starke Türkis im Hintergrund, das an manchen Wirbeln erkennbar ist. Filtert man hingegen mit den mittleren Gradienten, so erhält man folgendes, wesentlich bessere Resultat:

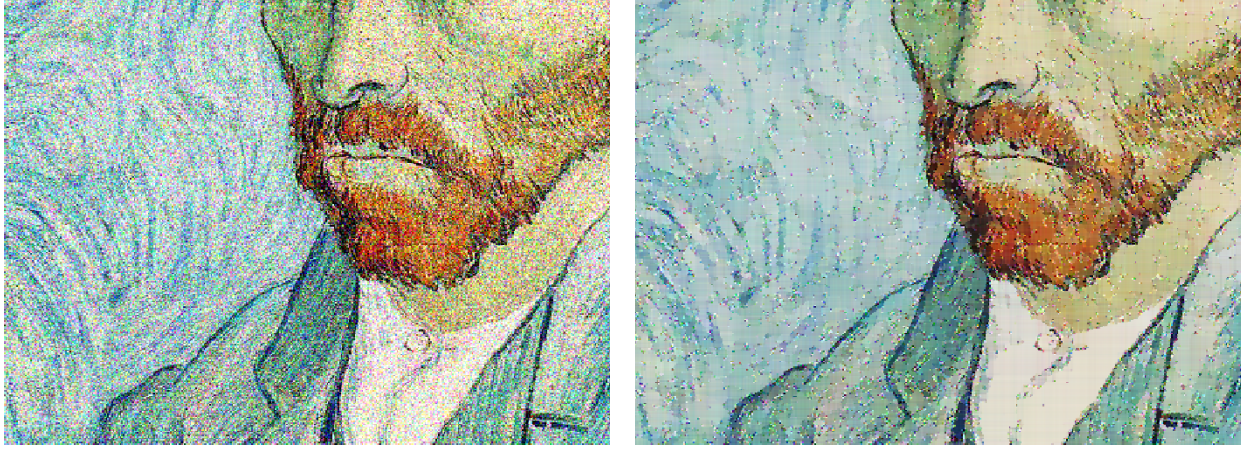


Abbildung 11: Links: Original mit Rauschen, Rechts: Gefiltert: $\tau = 5 \cdot 10^{-1}$, $\sigma = 1 \cdot 10^{-4}$, $k = 4$, $\lambda = 8$ (Künzel)

Man beachte, dass hier nur die Hälfte an Iterationen notwendig war. Das Filtern mit normierten Gradienten schließlich zeigt ein ähnlich gutes Ergebnis, jedoch war wieder die doppelte Iterationszahl erforderlich:



Abbildung 12: Links: Original mit Rauschen, Rechts: Gefiltert: $\tau = 5 \cdot 10^{-1}$, $\sigma = 1 \cdot 10^{-4}$, $k = 10$, $\lambda = 20$ (Künzel)

2.3 Anisotroper Filter

2.3.1 Berechnung des Diffusionstensors

Die Implementierung des anisotropen Filters muss zunächst in jedem Iterationsschritt an allen Bildpositionen den Diffusionstensor D aufstellen. Dazu wird wie in der Einleitung beschrieben zunächst der Strukturtensor $J_\rho(\nabla u_\sigma)$ aufgestellt und dann dessen Eigenvektoren $v_1 = \begin{pmatrix} v_1^1 \\ v_1^2 \end{pmatrix}$, $v_2 = \begin{pmatrix} v_2^1 \\ v_2^2 \end{pmatrix}$ und Eigenwerte μ_1, μ_2 mit den Formeln aus der Einleitung berechnet. Anschließend werden je nach Konfiguration der Diffusionsfunktionen in Kontrast- und Normalenrichtung (g_K, g_N) hieraus die gewünschten Eigenwerte λ_1, λ_2 des Diffusionstensors gemäß

$$\lambda_1 := g_K(\mu_1)$$

$$\lambda_2 := g_N(\mu_2)$$

berechnet. Damit kann dann der Diffusionstensor durch die Formel

$$\begin{aligned} D &:= \frac{1}{v_1^1 v_2^2 - v_2^1 v_1^2} \begin{pmatrix} v_1^1 & v_2^1 \\ v_1^2 & v_2^2 \end{pmatrix} \begin{pmatrix} \lambda_1 & 0 \\ 0 & \lambda_2 \end{pmatrix} \begin{pmatrix} v_2^2 & -v_2^1 \\ -v_1^2 & v_1^1 \end{pmatrix} \\ &= \frac{1}{v_1^1 v_2^2 - v_2^1 v_1^2} \begin{pmatrix} \lambda_1 v_1^1 v_2^2 - \lambda_2 v_1^2 v_2^1 & \lambda_2 v_1^1 v_2^1 - \lambda_1 v_1^2 v_2^2 \\ \lambda_1 v_1^2 v_2^2 - \lambda_2 v_1^1 v_2^1 & \lambda_2 v_1^2 v_2^2 - \lambda_1 v_1^1 v_2^1 \end{pmatrix} \end{aligned}$$

gewonnen werden. Man beachte, dass wegen $v_1 \perp v_2$ d.h.

$$\left\langle \begin{pmatrix} v_1^1 \\ v_1^2 \end{pmatrix}, \begin{pmatrix} v_2^1 \\ v_2^2 \end{pmatrix} \right\rangle = 0$$

sofort die Gleichung

$$\begin{aligned} (\lambda_2 v_1^1 v_2^1 - \lambda_1 v_1^1 v_2^1) - (\lambda_1 v_1^2 v_2^2 - \lambda_2 v_1^2 v_2^2) &= \lambda_2 (v_1^1 v_2^1 + v_1^2 v_2^2) - \lambda_1 (v_1^1 v_2^1 + v_1^2 v_2^2) \\ &= (\lambda_2 - \lambda_1) (v_1^1 v_2^1 + v_1^2 v_2^2) \\ &= (\lambda_2 - \lambda_1) \left\langle \begin{pmatrix} v_1^1 \\ v_1^2 \end{pmatrix}, \begin{pmatrix} v_2^1 \\ v_2^2 \end{pmatrix} \right\rangle \\ &= 0 \end{aligned}$$

folgt, d.h. der so aufgestellte Diffusionstensor D ist tatsächlich stets symmetrisch und es braucht insbesondere nur ein Eintrag der nicht auf der Diagonalen liegt ausgerechnet werden.

2.3.2 Diskretisierung des Divergenzoperators für das AOS

Für das AOS ist eine Diskretisierung des Divergenzoperators $\text{div}(D\nabla u)$ erforderlich, welche derart additiv aufgesplittet werden muss, dass die einzelnen linearen Teile durch verschobene Tridiagonalmatrizen darstellbar sind. Hierbei taucht das Problem auf, dass wegen der Verwendung einer Matrix als Diffusionstensor nun auch gemischte Ableitungen zu berücksichtigen sind. Dadurch können nicht mehr unmittelbar alle Eigenschaften für die entstehenden Matrizen Q garantiert werden, die notwendig wären um einen Skalenraum zu erzeugen, d.h. die Konvergenz steht zunächst in Frage.

Durch Untersuchung dieses Problems wird in [2] bewiesen, dass man diese Eigenschaft stets erreichen kann, indem zur Approximation des Divergenzoperators ein eventuell sehr großer Punktesternoperator benutzt wird. Da bei einem solchen dann jedoch wieder unklar ist, wie er für das AOS passend aufzuspalten ist wurde in diesem Fortgeschrittenenpraktikum fest ein 9-Punkte-Sternoperator implementiert, der direkt aus [2] abgeleitet wurde. Dieser wurde in 4 Komponenten additiv zerlegt, die in der folgenden Grafik skizziert werden.

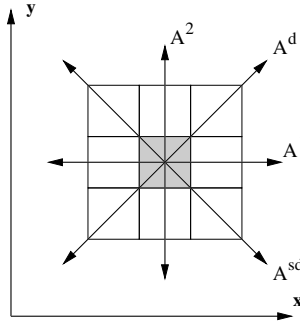


Abbildung 13: Richtungen, in die der 9-Punkte-Sternoperator aufgespalten wird

Im Folgenden wird nun die Besetzung der Matrizen für diese 4 Komponenten angegeben, wie sie auch implementiert wurde. Dabei muss beachtet werden, dass am Rand, wo nicht alle Nachbarn zur Verfügung stehen, homogene Neumann-Bedingungen zu setzen sind. Es sei nun stets

$$D = \begin{pmatrix} a & b \\ b & c \end{pmatrix}$$

der Diffusionstensor an allen Bildpunkten. Die Einträge a, b, c werden unten indiziert mit der Nummerierung aus der Kodierung eines zweidimensionalen Bildes als Vektor. Für diese Nummerierung sei noch folgende Notation vereinbart: Ist ein Index i gegeben, so werden die Indizes der Nachbapixel bzgl. einer Dimension ℓ mit $i_{\ell+}$ bzw. $i_{\ell-}$ bezeichnet. Analog sind Symbole i_{1+2-} zu lesen. Es werden stets nur die von 0 verschiedenen Einträge angegeben. Für die Fallunterscheidung der Diagonaloperatoren wird folgende Grafik in Kürze nützlich sein:

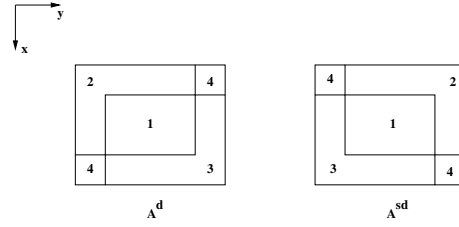


Abbildung 14: Fälle, die bei der Besetzung der Matrizen der Diagonaloperatoren zu unterscheiden sind

- A^1 : Sei i der Index eines Bildpunktes. Dann wird $A^1 = A^1_{i,j}$ zeilenweise gegeben durch:
Falls i kein Randpixel bezüglich Dimension 1 ist:

$$\begin{aligned} A^1_{i,i_{1-}} &= \frac{a_{i_{1-}} + a_i}{2h_1^2} - \frac{|b_{i_{1-}}| + |b_i|}{2h_1 h_2} \\ A^1_{i,i_{1+}} &= \frac{a_{i_{1+}} + a_i}{2h_1^2} - \frac{|b_{i_{1+}}| + |b_i|}{2h_1 h_2} \\ A^1_{i,i} &= -\frac{a_{i_{1-}} + 2a_i + a_{i_{1+}}}{2h_1^2} + \frac{|b_{i_{1-}}| + 2|b_i| + |b_{i_{1+}}|}{2h_1 h_2} \end{aligned}$$

Falls i ein kleiner Randpixel bezüglich Dimension 1 ist:

$$\begin{aligned} A^1_{i,i_{1+}} &= \frac{a_{i_{1+}} + a_i}{2h_1^2} - \frac{|b_{i_{1+}}| + |b_i|}{2h_1 h_2} \\ A^1_{i,i} &= -\frac{a_{i_{1+}} + a_i}{2h_1^2} + \frac{|b_{i_{1+}}| + |b_i|}{2h_1 h_2} \end{aligned}$$

Falls i ein großer Randpixel bezüglich Dimension 1 ist:

$$\begin{aligned} A^1_{i,i_{1-}} &= \frac{a_{i_{1-}} + a_i}{2h_1^2} - \frac{|b_{i_{1-}}| + |b_i|}{2h_1 h_2} \\ A^1_{i,i} &= -\frac{a_{i_{1-}} + a_i}{2h_1^2} + \frac{|b_{i_{1-}}| + |b_i|}{2h_1 h_2} \end{aligned}$$

Der Operator D^1 für das AOS wird dann gegeben durch

$$D^1 = (4E - 4^2 \tau A^1)$$

- A^2 : Sei i der Index eines Bildpunktes. Dann wird $A^2 = A^2_{i,j}$ zeilenweise gegeben durch:
Falls i kein Randpixel bezüglich Dimension 2 ist:

$$\begin{aligned} A^2_{i,i_{2-}} &= \frac{c_{i_{2-}} + c_i}{2h_2^2} - \frac{|b_{i_{2-}}| + |b_i|}{2h_1 h_2} \\ A^2_{i,i_{2+}} &= \frac{c_{i_{2+}} + c_i}{2h_2^2} - \frac{|b_{i_{2+}}| + |b_i|}{2h_1 h_2} \\ A^2_{i,i} &= -\frac{c_{i_{2-}} + 2c_i + c_{i_{2+}}}{2h_2^2} + \frac{|b_{i_{2-}}| + 2|b_i| + |b_{i_{2+}}|}{2h_1 h_2} \end{aligned}$$

Falls i ein kleiner Randpixel bezüglich Dimension 2 ist:

$$\begin{aligned} A^2_{i,i_{2+}} &= \frac{c_{i_{2+}} + c_i}{2h_2^2} - \frac{|b_{i_{2+}}| + |b_i|}{2h_1 h_2} \\ A^2_{i,i} &= -\frac{c_{i_{2+}} + c_i}{2h_2^2} + \frac{|b_{i_{2+}}| + |b_i|}{2h_1 h_2} \end{aligned}$$

Falls i ein großer Randpixel bezüglich Dimension 2 ist:

$$\begin{aligned} A_{i,i_2-}^2 &= \frac{c_{i_2-} + c_i}{2h_2^2} - \frac{|b_{i_2-}| + |b_i|}{2h_1h_2} \\ A_{i,i}^2 &= -\frac{c_{i_2-} + c_i}{2h_2^2} + \frac{|b_{i_2-}| + |b_i|}{2h_1h_2} \end{aligned}$$

Der Operator D^2 für das AOS wird dann gegeben durch

$$D^2 = (4E - 4^2\tau A^2)$$

- A^d : Sei i der Index eines Bildpunktes. Dann wird $A^d = A_{i,j}^d$ zweilenweise gegeben durch:
Falls i kein Randpixel bezüglich beider Dimensionen ist (Fall 1):

$$\begin{aligned} A_{i,i_1-2-}^d &= \frac{|b_{i_1-2-}| + |b_{i_1-2-}|}{4h_1h_2} + \frac{|b_i| + |b_i|}{4h_1h_2} \\ A_{i,i_1+2+}^d &= \frac{|b_{i_1+2+}| + |b_{i_1+2+}|}{4h_1h_2} + \frac{|b_i| + |b_i|}{4h_1h_2} \\ A_{i,i}^d &= -\left(\frac{|b_i| + |b_i|}{2h_1h_2} + \frac{|b_{i_1-2-}| + |b_{i_1-2-}| + |b_{i_1+2+}| + |b_{i_1+2+}|}{4h_1h_2} \right) \end{aligned}$$

Falls i ein kleiner Randpixel bezüglich einer der Dimensionen ist, aber bezüglich der anderen kein großer Randpixel (Fall 2):

$$\begin{aligned} A_{i,i_1+2+}^d &= \frac{|b_{i_1+2+}| + |b_{i_1+2+}|}{4h_1h_2} + \frac{|b_i| + |b_i|}{4h_1h_2} \\ A_{i,i}^d &= -\left(\frac{|b_{i_1+2+}| + |b_{i_1+2+}|}{4h_1h_2} + \frac{|b_i| + |b_i|}{4h_1h_2} \right) \end{aligned}$$

Falls i ein großer Randpixel bezüglich einer der Dimensionen ist, aber bezüglich der anderen kein kleiner Randpixel (Fall 3):

$$\begin{aligned} A_{i,i_1-2-}^d &= \frac{|b_{i_1-2-}| + |b_{i_1-2-}|}{4h_1h_2} + \frac{|b_i| + |b_i|}{4h_1h_2} \\ A_{i,i}^d &= -\left(\frac{|b_{i_1-2-}| + |b_{i_1-2-}|}{4h_1h_2} + \frac{|b_i| + |b_i|}{4h_1h_2} \right) \end{aligned}$$

Ansonsten ist i ein kleiner Randpixel bezüglich einer der Dimensionen und ein großer bezüglich der anderen (Fall 4):

$$A_{i,j}^d = 0$$

Der Operator D^d für das AOS wird dann gegeben durch

$$D^d = (4E - 4^2\tau A^d)$$

- A^{sd} : Sei i der Index eines Bildpunktes. Dann wird $A^{sd} = A_{i,j}^{sd}$ zweilenweise gegeben durch:
Falls i kein Randpixel bezüglich beider Dimensionen ist (Fall 1):

$$\begin{aligned} A_{i,i_1-2+}^d &= \frac{|b_{i_1-2+}| - |b_{i_1-2+}|}{4h_1h_2} + \frac{|b_i| - |b_i|}{4h_1h_2} \\ A_{i,i_1+2-}^d &= \frac{|b_{i_1+2-}| - |b_{i_1+2-}|}{4h_1h_2} + \frac{|b_i| - |b_i|}{4h_1h_2} \\ A_{i,i}^d &= -\left(\frac{|b_i| - |b_i|}{2h_1h_2} + \frac{|b_{i_1-2+}| - |b_{i_1-2+}| + |b_{i_1+2-}| - |b_{i_1+2-}|}{4h_1h_2} \right) \end{aligned}$$

Falls i wie in Fall 2 aus der Grafik (das Ausformulieren dieser Situation ist zu kompliziert und missverständlich):

$$\begin{aligned} A_{i,i_1+2-}^d &= \frac{|b_{i_1+2-}|-b_{i_1+2-}}{4h_1h_2} + \frac{|b_i|-b_i}{4h_1h_2} \\ A_{i,i}^d &= - \left(\frac{|b_{i_1+2-}|-b_{i_1+2-}}{4h_1h_2} + \frac{|b_i|-b_i}{4h_1h_2} \right) \end{aligned}$$

Falls i wie in Fall 3 aus der Grafik:

$$\begin{aligned} A_{i,i_1-2+}^d &= \frac{|b_{i_1-2+}|-b_{i_1-2+}}{4h_1h_2} + \frac{|b_i|-b_i}{4h_1h_2} \\ A_{i,i}^d &= - \left(\frac{|b_{i_1-2+}|-b_{i_1-2+}}{4h_1h_2} + \frac{|b_i|-b_i}{4h_1h_2} \right) \end{aligned}$$

Ansonsten (Fall 4):

$$A_{i,j}^{sd} = 0$$

Der Operator D^{sd} für das AOS wird dann gegeben durch

$$D^{sd} = (4E - 4^2\tau A^{sd})$$

Da die Mittelung mit $\frac{1}{4}$ oben bereits in die Matrizen der Operatoren eingebaut wurde ergibt sich damit als Diskretisierung der Differentialgleichung eines anisotropen Filters mit dem AOS Verfahren:

$$\begin{aligned} u^{(0)} &= f \\ u^{(k+1)} &= \left((D^1)^{-1} + (D^2)^{-1} + (D^d)^{-1} + (D^{sd})^{-1} \right) u^{(k)} \end{aligned}$$

Es sind also in jedem Iterationsschritt die Gleichungssysteme

$$\begin{aligned} D^1 v_1^{(k)} &= u^{(k)} \\ D^2 v_2^{(k)} &= u^{(k)} \\ D^d v_d^{(k)} &= u^{(k)} \\ D^{sd} v_{sd}^{(k)} &= u^{(k)} \end{aligned}$$

zu lösen und anschließend

$$u^{(k+1)} := v_1^{(k)} + v_2^{(k)} + v_d^{(k)} + v_{sd}^{(k)}$$

zu setzen.

2.3.3 Beispiele

Als Beispiel wird folgendes Originalbild betrachtet:

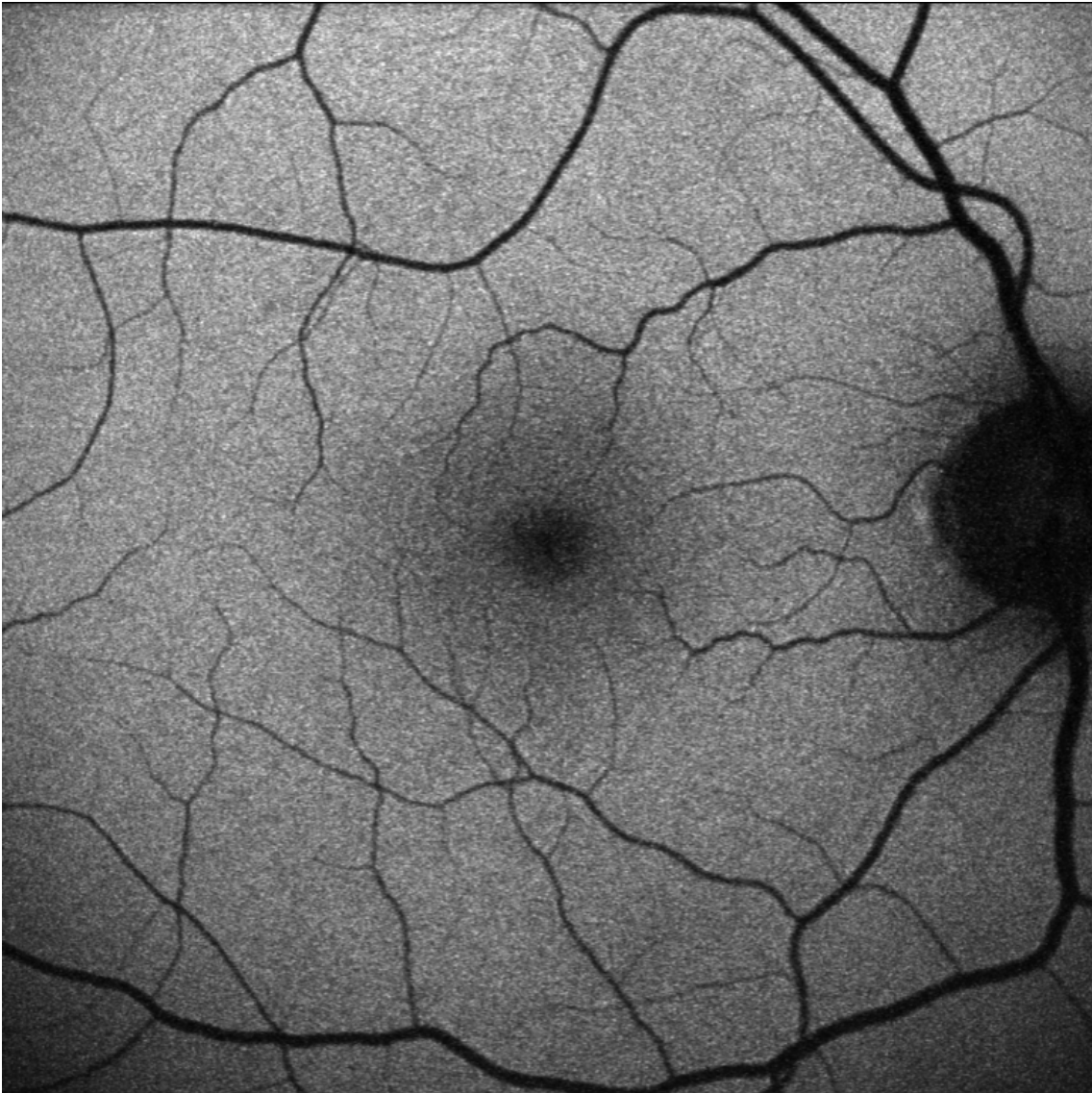


Abbildung 15: Ausgangsbild für die Beispiele

Wie im letzten Abschnitt bereits erwähnt, ist die Fixierung auf einen 9-Punkte-Sternoperator ungünstig, da nicht garantiert werden kann, dass die dadurch gegebene Diskretisierung die Skalenraumeigenschaften erfüllt, d.h. gegen die analytische Lösung konvergiert. In der Praxis taucht schnell das in der folgenden Abbildung dargestellte Problem auf:

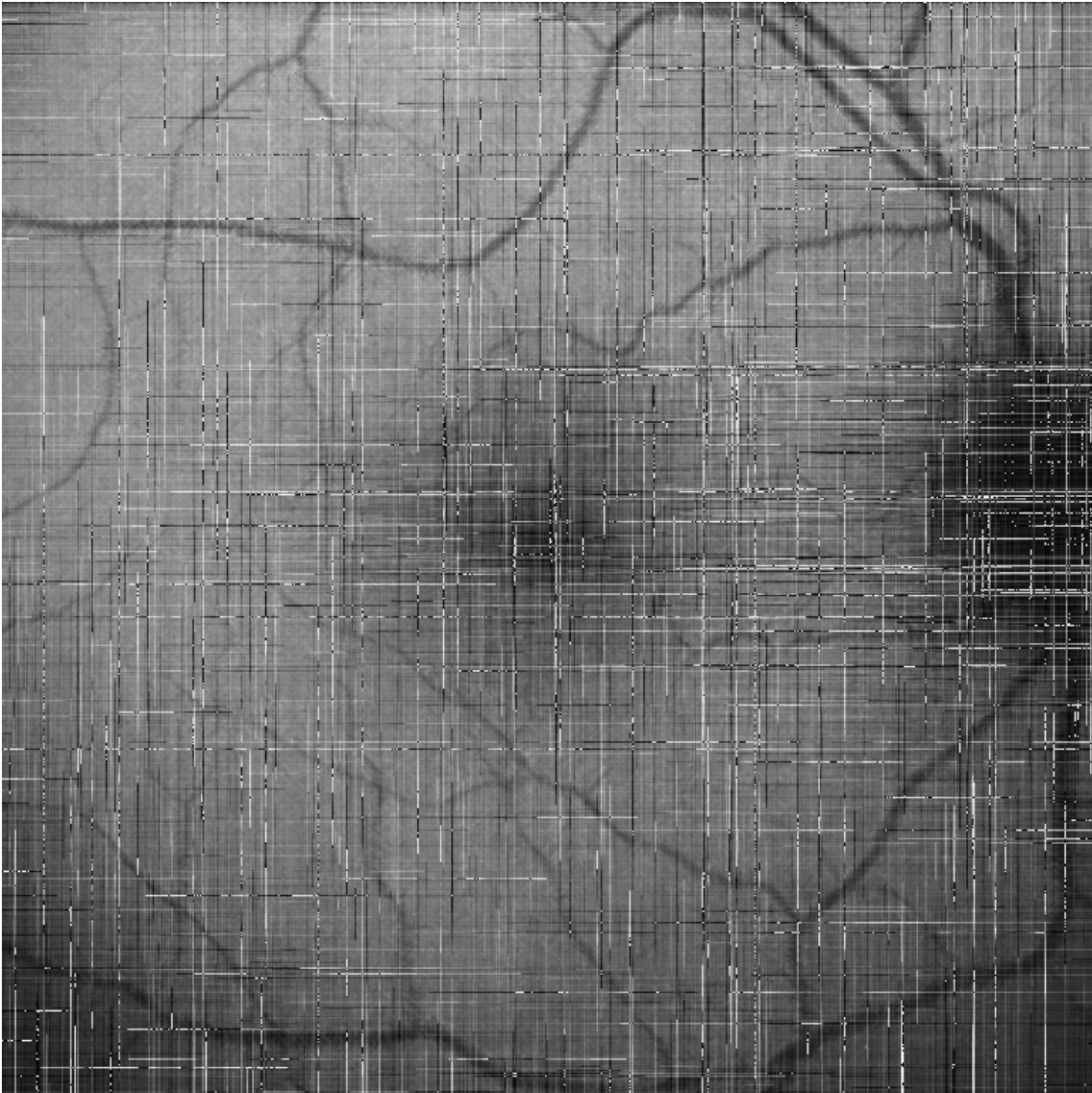


Abbildung 16: Artefakte, die bei einem anisotropen Filtervorgang entstanden sind
 $(\tau = 5 \cdot 10^{-3}, \sigma = 1 \cdot 10^{-4}, \rho = 5 \cdot 10^{-4}, 1 \text{ Iteration mit Künzel-Diffusion (Kontrastrichtung), } \lambda = 7.5)$

Diese Artefakte rühren wahrscheinlich von dem Problem des 9-Punkte-Sternoperators her. Wenn dagegen mit sehr kleinen Zeitschrittweiten gerechnet wird, so lassen sich diese Artefakte vermeiden und der Filter liefert brauchbare Ergebnisse:

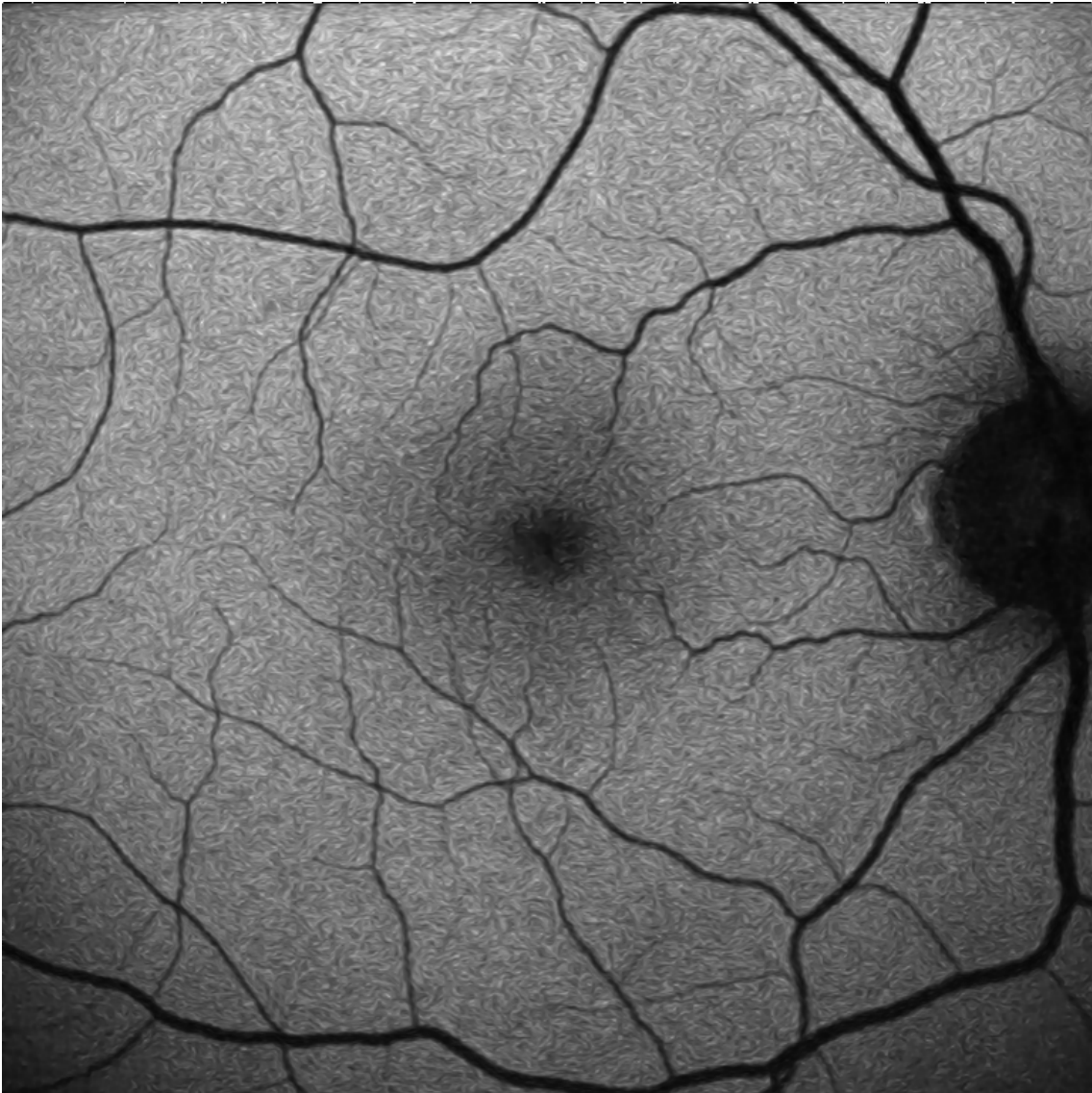


Abbildung 17: Anisotroper Filter mit kleiner Zeitschrittweite
($\tau = 5 \cdot 10^{-8}$, $\sigma = 1 \cdot 10^{-3}$, $\rho = 5 \cdot 10^{-3}$, 40 Iteration mit Künzel-Diffusion (Kontrastrichtung), $\lambda = 9$)

Hier ist deutlich zu erkennen, dass das Rauschen auf den Verästelungen fast vollständig verschwunden ist, auch an den kontrastreichen Rändern, wie man es von einem anisotropen Filter erwartet.

3 Bedienung des Filters mit der grafischen Oberfläche

Die gesamte Implementierung liegt in Form eines Quellcodearchives vor. Die grafische Oberfläche wurde in Java unter Verwendung von Swing implementiert und erfordert Java ab Version 1.6. Der Filter selbst ist in C++ geschrieben und kann beispielsweise mit der GNU Compiler Collection (GCC) kompiliert werden. Zu diesem Zweck werden entsprechende Makefiles bereitgestellt. Nach dem Kompilieren entsteht im Archiv ein Ordner bin, welcher das ausführbare Filterprogramm - ein Konsolenprogramm - und die grafische Oberfläche in Form eines JAR-Archivs enthält. Unter vielen Betriebssystemen lässt sich diese Oberfläche durch einen Doppelklick auf die JAR-Datei ausführen, ansonsten muss auf einer Konsole der Befehl

```
java -jar FilterControl.jar
```

benutzt werden. Die Verwendung der Kommandozeile bietet den Vorteil, dass der maximal von der Java Virtual Machine (JVM) verwendete Speicher festgelegt werden kann. Standardmäßig setzt die JVM eine Obergrenze bei 128 MB was in der Oberfläche schnell zu Fehlverhalten führen kann. Um die Obergrenze an verfügbarem Speicher festzulegen kann ein Befehl wie

```
java -Xmx512m -jar FilterControl.jar
```

benutzt werden, in diesem Fall würden der JVM 512 MB Speicher maximal verfügbar. Nach dem Start präsentiert die Oberfläche ihr Hauptfenster, was in etwa wie folgt aussieht:

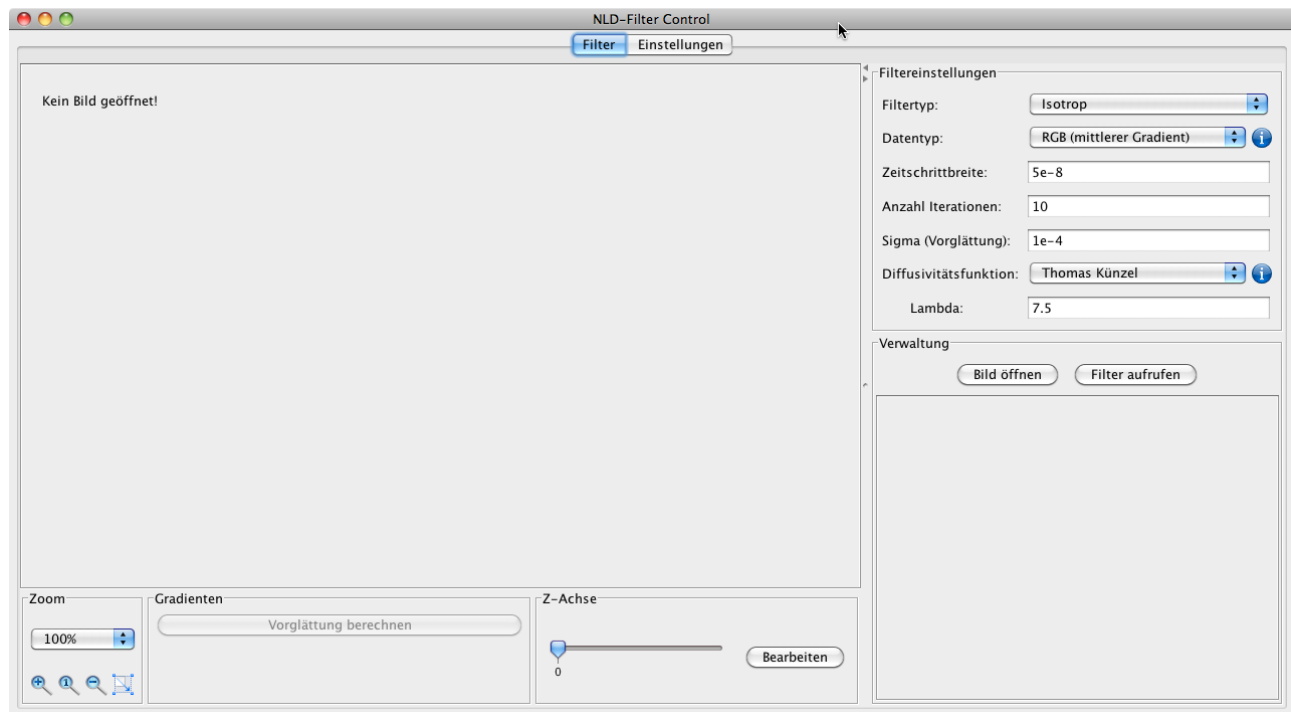


Abbildung 18: Hauptfenster der grafischen Oberfläche

Im rechten unteren Bereich findet sich die Verwaltung aller im Programm geladenen Bilder. Die Oberfläche kann mehrere Bilder gleichzeitig geöffnet haben, so dass beispielsweise die Ergebnisse von Filterläufen mit unterschiedlichen Parametern leicht verglichen werden können. Im rechten oberen Bereich befindet sich die Konfiguration der Filtereinstellungen. Man kann hier leicht einen gewünschten Filter und seine Eigenschaften konfigurieren um mit diesem dann Bilder zu filtern. Der linke Bereich beinhaltet die Anzeige des aktuell zum Bearbeiten geöffneten Bildes: Zu jedem Zeitpunkt kann nur eines der geladenen Bilder zum Bearbeiten geöffnet sein.

3.1 Konfiguration der grafischen Oberfläche

Am oberen Rand des Hauptfensters befindet sich eine Leiste zum Umschalten zwischen der Filter-Ansicht, die zum Arbeiten notwendig ist und der Einstellungen-Ansicht, welche die grafische Oberfläche konfiguriert:

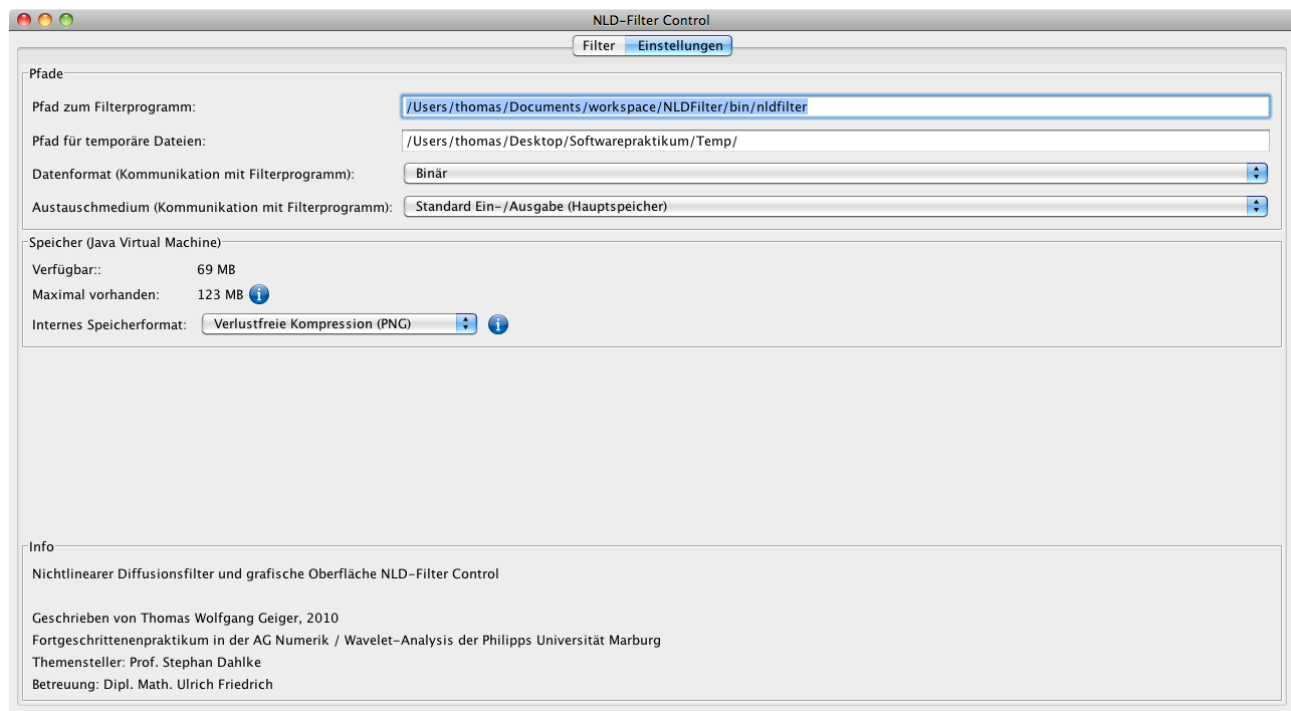


Abbildung 19: Die Einstellungen-Ansicht

Die Oberfläche versucht während des Startvorgangs anhand einiger Daten über das System, die sie ermittelt, die Einstellungen möglichst sinnvoll vorzubelegen. Je nach Situation kann es jedoch erwünscht oder erforderlich sein, diese Einstellungen zu korrigieren, daher wird hier kurz auf die Möglichkeiten eingegangen.

Da die grafische Oberfläche und das Filterprogramm zwei getrennte Programme sind, muss der grafische Oberfläche bekannt sein, wo sie den Filter findet. Außerdem benötigt die grafische Oberfläche, je nachdem wie sie mit dem Filterprogramm kommunizieren soll, einen Ordner, in dem sie temporäre Dateien ablegen kann. Daher müssen im ersten Abschnitt der Einstellungen korrekte Pfade zum Filterprogramm und zu einem Ordner für temporäre Dateien angegeben werden. Wenn die Oberfläche startet, setzt sie die Pfade auf die Situation passend, wenn sich das Filterprogramm und der Ordner für temporäre Dateien im selben Verzeichnis wie das JAR-Archiv befinden, d.h. auf exakt die Situation, die nach dem Kompilieren der Software vorliegt.

Darunter kann festgelegt werden, auf welche Weise die Kommunikation zwischen Oberfläche und Filter ablaufen soll. Als Kommunikationsmedien stehen temporäre Dateien und alternativ die Verwendung der Standard Ein-/Ausgabe zur Verfügung. Da bei letzterem die Kommunikation vollständig über den Hauptspeicher erfolgen kann ist diese Option im allgemeinen effizienter. Weiterhin kann noch das Format für die Kommunikation festgelegt werden: Das ASCII Format ist das von der Implementierung von Thomas Künzel einzig verwendete Format. Es bietet wenn temporäre Dateien benutzt werden den Vorteil, dass solche Dateien auch in ein Matlab-Programm eingelesen werden können. Da in diesem Format jedoch alle Pixelwerte in menschenlesbarer Form kommuniziert werden, ist die zu kommunizierende Datenmenge sehr groß. Außerdem kommt es stets zu Rundungen wenn eine Zahl in dieses Format konvertiert wird. Wenn keine besonderen Anforderungen es verlangen sollte dieses Format daher nicht verwendet werden. Vorzuziehen ist das Binärformat als zweites angebotenes Kommunikationsformat. Hier werden die Pixelwerte (8 Byte Fließkommazahlen) direkt durch Übertragen der 8 Byte die diese Zahl definieren kommuniziert. Dies reduziert den Speicherbedarf und die Pixelwerte können verlustfrei ausgetauscht werden.

Unter Windows-Betriebssystemen ist es unklar wie man binäre Daten über die Standard Ein-/Ausgabe kommunizieren kann, daher funktioniert diese Option nur unter Unix-Systemen. Unter Windows muss entsprechend auf die binäre Kommunikation mit temporären Dateien ausgewichen werden, was mehr Zeit erfordert. Daher ist es empfehlenswert, das Programm stets unter Unix-Systemen zu verwenden.

Der zweite Abschnitt in der Einstellungen-Ansicht bezieht sich auf die Speicherverwaltung. Wie bereits erwähnt setzt die JVM meist eine Maximalgrenze für den verfügbaren Speicher. Die Oberfläche zeigt diese Grenze und eine Abschätzung des noch verfügbaren Speichers in diesem Abschnitt an. Da Javas Garbage-Collection unvorhersehbar ausgeführt wird ist der Wert des verfügbaren Speichers nicht sonderlich verlässlich.

Da in der Oberfläche mehrere Bilder gleichzeitig geladen sein können und jedes einzelne leicht einen großen Speicher-

bedarf hat (wenn beispielsweise ein großes 3D-Bild geladen wird) ist es notwendig eine geeignete interne Kodierung der geladenen Bilder im Hauptspeicher zu finden. Es werden drei Möglichkeiten angeboten, aus denen man je nach Situation die passende wählen sollte:

- Unkomprimiert: Alle geladenen Bilder werden unkomprimiert im Hauptspeicher gehalten. Diese Kodierung ist selbstverständlich am schnellsten, da alle Bilder unmittelbar wenn sie benötigt werden auch ohne weiteren Dekodieraufwand verfügbar sind. Dennoch kann sie nur in gewissem Umfang sinnvoll genutzt werden, da sonst sehr schnell eine große Menge Hauptspeicher benötigt wird: Bei einem 3D-RGB-Bild mit der Auflösung $500 \times 500 \times 500$ fallen unkomprimiert beispielsweise knapp 360 MB Daten an.
- Verlustfreie Komprimierung mit PNG: Die geladenen Bilder werden intern im Hauptspeicher im PNG Bildformat kodiert abgelegt. Das PNG-Format komprimiert verlustfrei, d.h. es gehen durch die Komprimierung keinerlei Informationen im Bild verloren und es führt dennoch zu relativ kleinen benötigten Speichermengen. Der Preis der hierfür bezahlt werden muss liegt in der Geschwindigkeit: Der Algorithmus zum Kodieren und Dekodieren des PNG Formats benötigt einige Rechenzeit (auf einem modernen MacBook Pro mit Intel Core 2 Duo 2.26 GHz werden zur Kodierung eines RGB-Farbbildes, 827×1026 etwa 680 ms benötigt, zur Dekodierung etwa 240 ms). Da im Programm häufig Bilder kodiert und besonders häufig dekodiert werden müssen kann dies zu spürbaren Verzögerungen in der Reaktionszeit der Oberfläche führen.
- Verlustbehaftete Komprimierung mit JPEG: Die geladenen Bilder werden im Hauptspeicher im JPEG Bildformat kodiert. JPEG ist ein verlustbehaftetes Bildformat, d.h. bei jedem Kodieren geht effektiv Information im Bild verloren. Der Vorteil gegenüber PNG liegt darin, dass erheblich weniger Speicherplatz im Hauptspeicher pro Bild benötigt wird und dass Kodieren und Dekodieren schneller ausgeführt werden (erneut auf dem MacBook Pro wurde dasselbe Bild in 250 ms kodiert und in 55 ms dekodiert, der Speicherbedarf war um einen Faktor von 10 kleiner). Diese Kodierung kann also insbesondere bei Tests oder während der Vorbereitung einer umfangreichen Arbeit mit dem Programm benutzt werden, wenn beispielsweise passende Einstellungen für eine große Datenmengen zu finden sind, das Programm aber dennoch ohne Verzögerung arbeiten soll. Zum Erzeugen der effektiven Resultate ist es dagegen weniger geeignet.

Als Voreinstellung wählt das Programm stets die verlustfreie Komprimierung mit dem PNG-Format aus. Man sollte vorsichtig sein, wenn diese Einstellung geändert wird, während bereits viele Bilder geladen wurden. Steht dann nicht genügend Hauptspeicher zur Verfügung versucht das Programm zur alten Einstellung zurückzukehren, jedoch kann evtl. auch hierfür nicht ausreichend Speicher verfügbar sein. Es kann dann zu Datenverlusten bzw. im schlimmsten Fall dazu kommen, dass das Programm sich beenden muss.

3.2 Verwaltung geöffneter Bilder

Die folgende Abbildung zeigt die Verwaltung mit mehreren geladenen Bildern, eines aus einem Filtervorgang entstanden:



Abbildung 20: Verwaltung geladener Bilder

Zunächst soll die Schaltfläche mit dem Titel "Bild öffnen" oberhalb der Liste betrachtet werden. Mit dieser Schaltfläche wird ein Dialogfenster zum Öffnen von Dateien angezeigt, die anschließend in das Programm geladen werden

und in der Liste auftauchen. In obigem Bild wurde beispielsweise der untere Eintrag direkt aus einer Datei von der Festplatte geladen, der obere hingegen ist das Ergebnis eines Filterlaufs wie man an der Anzeige der Parameter, die zum Filtern benutzt wurden erkennen kann.

Das Programm behandelt 2D und 3D Bilder prinzipiell gleich: Ein 2D-Bild ist der Spezialfall eines 3D-Bildes bei dem nur an der Position $z = 0$ ein einziges Bild geladen wurde. Wird im angezeigten Dialogfenster zum Öffnen von Dateien nur eine Datei markiert und geöffnet, so wird ein 2D-Bild geladen, werden hingegen mehrere Dateien markiert und geöffnet, so werden diese (in lexikalischer Ordnung) entlang der z -Achse als 3D-Bild geöffnet (und nicht etwa alle markierten Dateien separat als 2D-Bilder).

Das Programm kann des weiteren mit zwei verschiedenen Farbraumtypen umgehen: RGB-Farbräume und Graustufen-Räume. Bei einem echten Graustufenbild ist in der Bilddatei nur eine Farbebene mit Werten gespeichert, die als Grauwerte interpretiert werden. Alle übrigen Farbräume (inklusive dem Fall, dass tatsächlich ein RGB-Bild vorliegt, also drei Farbebenen in der Bilddatei) werden vom Programm beim Laden automatisch in einen RGB-Farbraum konvertiert, d.h. insbesondere werden Farbräume die mit einer indizierten Farbtabelle arbeiten stets in einen vollen RGB-Farbraum konvertiert.

In der Verwaltung wird stets angezeigt, welchen Typ von Farbraum ein Bild besitzt. Des weiteren besitzt jeder Eintrag ein Palettensymbol mit welchem der Farbraum konvertiert werden kann. Wird ein RGB-Farbraum in einen Graustufenraum konvertiert, muss zwangsläufig angegeben werden, auf welche Weise dies geschehen soll (da in einem Graustufenraum weniger Information gespeichert werden kann).

Wie bereits erwähnt kann zu jedem Zeitpunkt nur eines der geladenen Bilder zum Bearbeiten geöffnet sein. Jeder Eintrag in der Verwaltung besitzt drei Schaltflächen von denen eine den Titel "Öffnen" trägt. Mit dieser Schaltfläche wird der jeweilige Eintrag zum Bearbeiten geöffnet, d.h. man kann das Bild in gewissem Maße bearbeiten. Außerdem wendet die Schaltfläche "Filter aufrufen" stets den konfigurierten Filter auf das gerade bearbeitete Bild an.

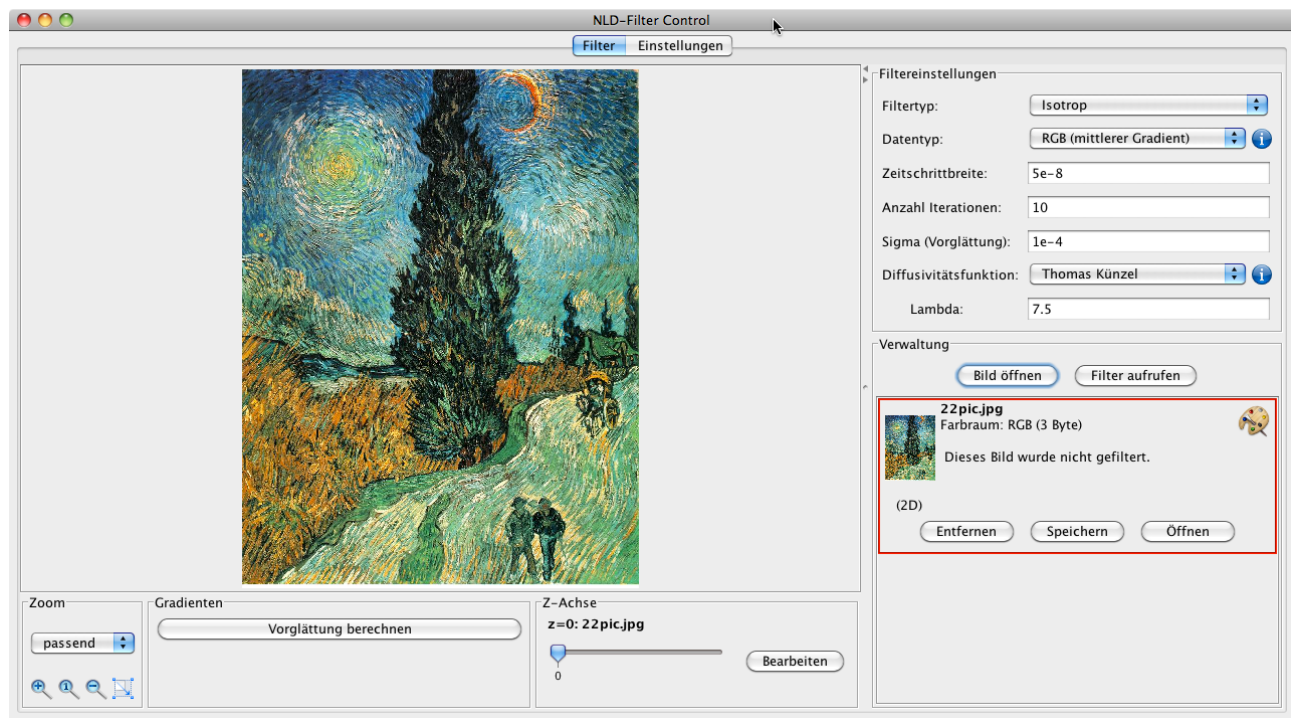


Abbildung 21: Grafische Oberfläche mit zum Bearbeiten geöffnetem Bild

Welches der geladenen Bilder gerade zum Bearbeiten geöffnet ist, wird in der Verwaltung durch einen roten Rahmen kenntlich gemacht.

Die Verwaltung speichert zusätzlich Informationen über die Verwandtschaft von einzelnen geladenen Bildern. Geht aus einem Bild durch Filtern ein neues hervor so sind beide verwandt: Das alte Bild ist das Elternbild des gefilterten. Ein Eintrag in der Verwaltung, zu dem Informationen über ein Elternbild vorhanden sind, erhält auf der rechten Seite ein Symbol mit einem grünen Pfeil.

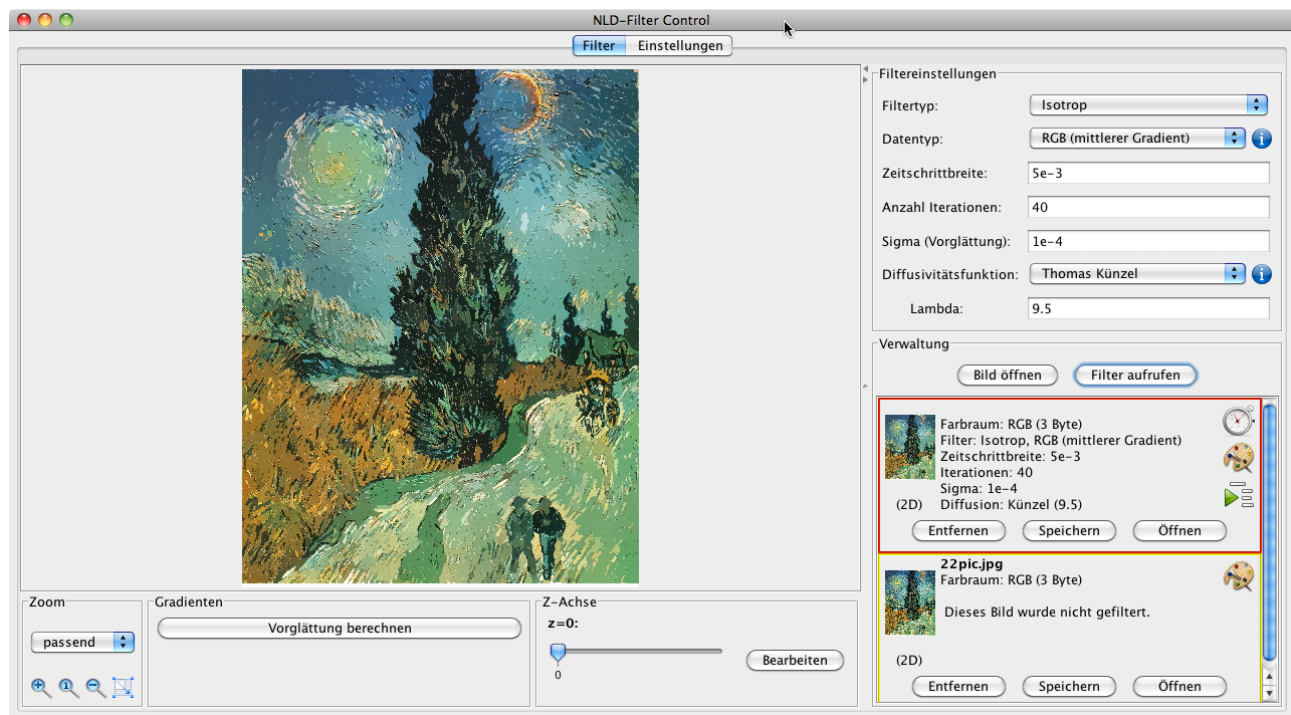


Abbildung 22: Veranschaulichung der Verwandtschaftsstruktur in der Verwaltung

Durch Klicken auf dieses Pfeilsymbol springt die Verwaltung zum Elternbild und öffnet dieses zum Bearbeiten. Ist hingegen ein Bild mit Informationen über ein Elternbild zum Bearbeiten geöffnet wie in der Abbildung oben, so wird zudem das Elternbild durch einen gelben Rahmen markiert. Man kann so leicht den Überblick darüber behalten, bzw. zurückverfolgen wie ein gefiltertes Bild entstanden ist.

Jeder Eintrag in der Verwaltung besitzt zusätzlich eine Schaltfläche “Entfernen”. Durch Auswählen dieser Schaltfläche wird der entsprechende Eintrag direkt ohne Rückfrage aus der Verwaltung entfernt und kann nicht wiederhergestellt werden (es sei denn das Bild ist auf einem Datenträger verfügbar).

Außerdem besitzt jeder Eintrag die Schaltfläche “Speichern”, die das Ablegen von Bildern auf Datenträgern ermöglicht. Als Dateiformate zum Speichern von Bilddaten werden stets png, jpeg und bmp unterstützt (wobei png das Format der Wahl sein sollte, da es kleine Dateigrößen bei verlustfreier Kompression liefert). Ob weitere Formate unterstützt werden hängt von der auf dem konkreten Rechner vorliegenden Java Installation ab. Daher gibt es keine Restriktionen bei der Wahl von Dateinamenerweiterungen. Welches Format dann zum Speichern benutzt wird entscheidet das Programm anhand der Dateinamenerweiterung wobei eine Fehlermeldung angezeigt wird, wenn das Format nicht bekannt ist.

Das Verhalten der Speichermöglichkeit ist unterschiedlich, je nachdem, ob ein 2D oder ein 3D-Bild gespeichert werden soll. Bei einem 2D-Bild genügt die Angabe des Dateinamens, bei einem 3D-Bild fragt das Programm ebenfalls zunächst nach einem Ort und Namen für eine Datei. Dieser wird jedoch nicht direkt verwendet, es muss ja nicht nur ein Bild, sondern für jeden z -Wert eines gespeichert werden. Zu diesem Zweck wird der eingegebene Dateiname (nicht jedoch der Ort) vom Programm automatisch durch einen Index ergänzt. Der Index wird entweder am Ende des Dateinamens vor der Dateinamenerweiterung eingefügt oder aber anstelle des letzten Unterstrichs im eingegebenen Dateinamen, falls ein solcher vorhanden ist. Nach der Wahl eines Dateinamens fordert folgendes Fenster dazu auf, die Art der Indizierung zu wählen:

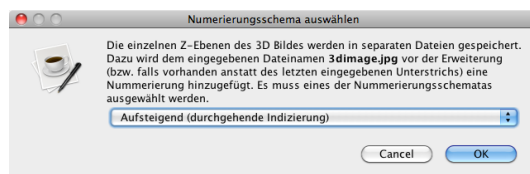


Abbildung 23: Wahl der Indizierung beim Speichern eines 3D-Bildes in der Verwaltung

Die Indizierung kann entweder aufsteigend in positiver Richtung entlang der z -Achse erfolgen, beginnend mit dem Index 0 oder es kann der z -Wert der Bilder direkt verwendet werden. Die folgende Abbildung zeigt das Ergebnis der aufsteigenden Indizierung nach dem Speichern eines 3D-Bildes mit 6 z -Ebenen wobei als Dateiname ursprünglich “3dimage.jpg” eingegeben wurde:

```
pcsd512:save3d thomas$ ls -lh
total 1816
-rw-r--r--  1 thomas  staff   174K Jul 20 12:30 3dimage0.jpg
-rw-r--r--  1 thomas  staff   106K Jul 20 12:30 3dimage1.jpg
-rw-r--r--  1 thomas  staff   154K Jul 20 12:30 3dimage2.jpg
-rw-r--r--  1 thomas  staff   134K Jul 20 12:30 3dimage3.jpg
-rw-r--r--  1 thomas  staff   105K Jul 20 12:30 3dimage4.jpg
-rw-r--r--  1 thomas  staff   221K Jul 20 12:30 3dimage5.jpg
pcsd512:save3d thomas$
```

Abbildung 24: Ergebnis des Speicherns eines 3D-Bildes mit aufsteigender Indizierung

3.3 Konfiguration eines Filters

Die Konfiguration eines Filters mit dem dann geladene Bilder gefiltert werden können erfolgt mit dem Abschnitt “Filtereinstellungen” rechts oben im Hauptfenster.

Abbildung 25: Filterkonfiguration

Die Anzeige baut sich dynamisch auf, je nach dem welchen Farbraumtyp das gerade zum Bearbeiten geöffnete Bild besitzt und welche Einstellungen bereits vorgenommen wurden. Die Anzeige unterscheidet sich beispielsweise stark je nach dem ob ein isotroper oder ein anisotroper Filter gewählt wurde. Außerdem benötigen manche Diffusionsfunktionen keine Parameter und andere mehrere. Grundsätzlich erscheint zunächst die Auswahl ob ein isotroper oder ein anisotroper Filter verwendet werden soll und der Datentyp. Daran schließen sich stets die allgemeinen Eigenschaften des Filters an, d.h. Iterationszahl, Schrittbreite, etc. Zum Schluss erscheint dann die Auswahl und Konfiguration der Diffusionsfunktionen. Parameter von Diffusionsfunktionen werden stets unterhalb der Auswahl der Diffusionsfunktion eingeblendet mit dem Parameternamen leicht eingerückt.

Das Auswahlfeld “Datentyp” vereinigt zwei notwendige Informationen: Zum einen ob ein Bild (wenn möglich) als RGB-Bild oder als Graustufenbild gefiltert werden soll und zum anderen bei RGB-Bildern noch den gewünschten Typ, wie in diesem Bericht im Abschnitt über das Filtern von Farbbildern beschrieben wurde. Ist ein echtes Graustufenbild zum Bearbeiten geöffnet, so kann dieses natürlich auch nur als Graustufenbild gefiltert werden, das Auswahlfeld Datentyp verschwindet zu Gunsten einer statischen Anzeige. Bei RGB-Bildern hingegen stehen im isotropen Fall (der implementierte anisotrope Filter kann nur Graubilder filtern) die verschiedenen Möglichkeiten in einem solchen Bild lokale Information für die Diffusionsfunktion bereitzustellen zur Verfügung und zudem die Möglichkeit das Bild vor dem Filtern in ein Graustufenbild zu konvertieren. Diese Konvertierung ist identisch damit, den Farbraum des RGB-Bildes vor dem Filtern in einen Graustufenraum zu konvertieren (mit dem Palettensymbol), der Unterschied besteht darin, dass der Farbraum des zu filternden Bildes erhalten bleibt: Die Konvertierung erfolgt temporär für den entsprechenden Filterlauf.

Die Konfiguration blendet an manchen Stellen Informationssymbole ein. Klickt man ein solches an erscheint ein kleines Fenster mit genaueren Informationen zur aktuellen Auswahl an der entsprechenden Stelle. So kann man beispielsweise die Funktionsdefinitionen der Diffusionsfunktionen einsehen.

Um die Wahl der Parameter die sich auf die Diffusionsfunktionen beziehen zu erleichtern wurde eine Möglichkeit implementiert, die Größe der Gradienten in einem Bild oder genauer den Wert der Variablen s wie er in die Diffusionsfunktion eingesetzt wird, zu berechnen. Diese Werte hängen vom Konfigurierten Filter ab, da zu ihrer Bestimmung eine vorgeglättete Version des Bildes benötigt wird. Ist ein Bild zum Bearbeiten geöffnet, so wird unterhalb der Hauptanzeige die Schaltfläche “Vorglättung berechnen” aktiviert.

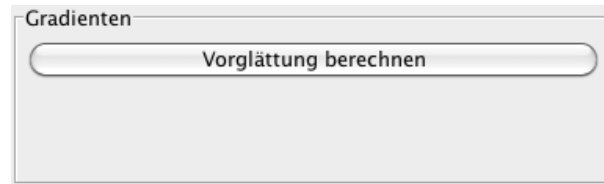


Abbildung 26: Vorglättung für ein Bild berechnen, um Informationen über Gradienten zu erhalten

Wenn man diese Schaltfläche anklickt, ruft die grafische Oberfläche das Filterprogramm auf, und lässt von ihm die Vorglättung des aktuell zum Bearbeiten geöffneten Bildes berechnen. Da dieser Vorgang einige Zeit benötigen kann, muss man dies explizit fordern.

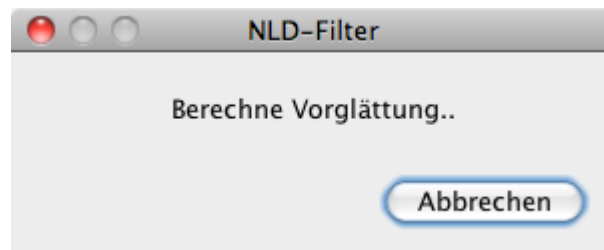


Abbildung 27: Berechnen der Vorglättung

Sobald die Vorglättung berechnet ist, stehen Informationen über die Gradienten zur Verfügung. Bewegt man die Maus über die Hauptanzeige, so werden Gradienteninformationen jeweils zur aktuellen Position des Mauszeigers unterhalb der nun deaktivierten Schaltfläche “Vorglättung berechnen” angezeigt.

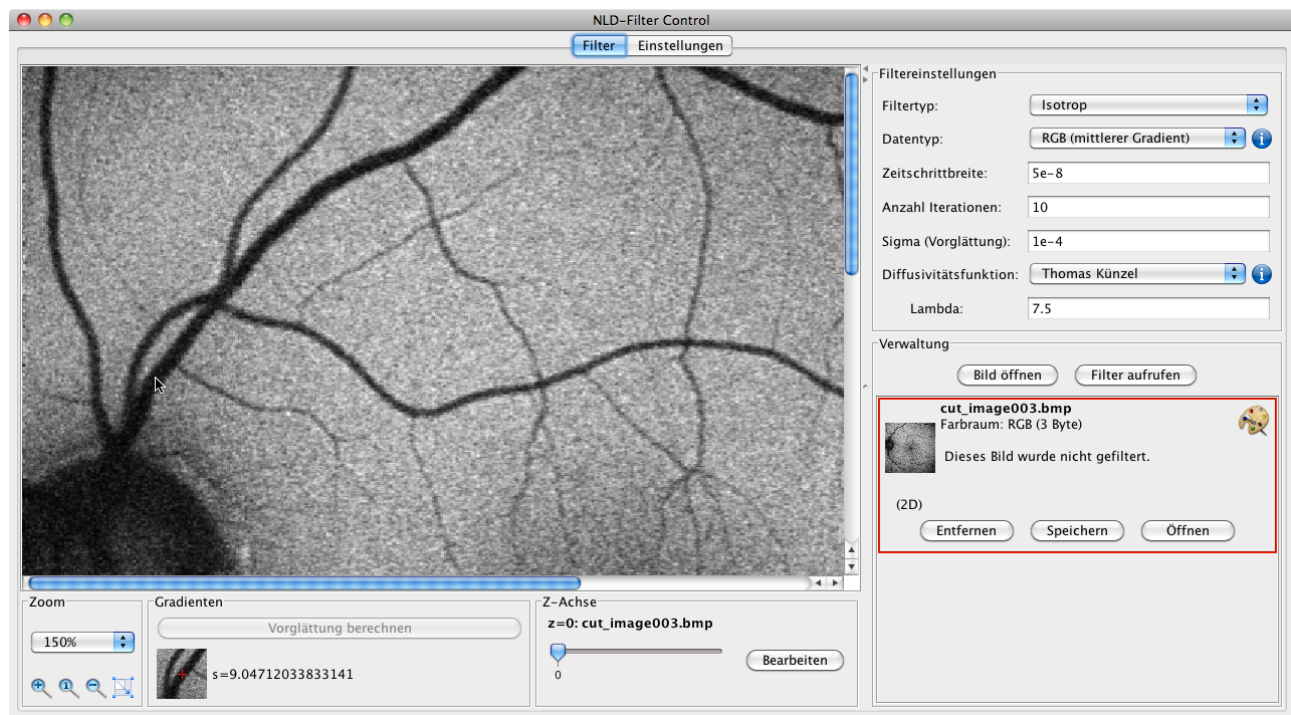


Abbildung 28: Anzeige von Gradienteninformationen

Da sich die Gradienten schnell ändern kann es für die genauere Untersuchung nützlich sein, das Bild maximal zu zoomen. Öffnet man ein neues Bild zum Bearbeiten oder ändert man eine Einstellung der Filterkonfiguration, welche die Vorglättung oder die Berechnung von s beeinflusst, so werden die gespeicherten Vorglättungsinformationen gelöscht und die Schaltfläche "Vorglättung berechnen" wieder aktiviert. Man muss die Vorglättung dann neu berechnen, bevor erneut Gradienteninformationen verfügbar sind.

Aus Effizienzgründen müssen die vorgeglätteten Bildinformationen unkomprimiert im Hauptspeicher abgelegt werden. Das in der Konfiguration der grafischen Oberfläche ausgewählte interne Kodierungsformat wird an dieser Stelle ignoriert. Es muss daher genügend Hauptspeicher verfügbar sein, wenn diese Funktionalität genutzt werden soll.

3.4 Filtern von Bildern

Um ein bestimmtes Bild zu filtern muss zunächst der gewünschte Filter konfiguriert und das entsprechende Bild geladen und zum Bearbeiten geöffnet werden. Anschließend wird der Filter aufgerufen, indem in der Verwaltung die Schaltfläche "Filter aufrufen" angeklickt wird. Das Programm prüft dann ob der konfigurierte Filter auf das Bild angewendet werden kann (beispielsweise kann ein 3D-Bild nicht mit einem anisotropen Filter behandelt werden) und ob die Programmeinstellungen korrekt sind und gibt eventuell eine Fehlermeldung aus.

Ist ein Filter mit mehr als einer Iteration konfiguriert erscheint folgendes Dialogfenster:

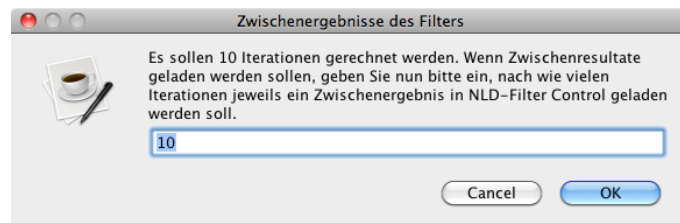


Abbildung 29: Zwischenresultate berechnen

Das Programm kann hier angewiesen werden, in bestimmten Abständen (gemessen in Iterationen) ein Zwischenresultat zu speichern. Die Vorgabe ist die angegebene Iterationszahl in einem Filterlauf zu berechnen, d.h. es wird in der Verwaltung nur exakt ein Bild nach der angegebenen Zahl der Iterationen eingefügt. Gibt man eine kleinere

Zahl ein, so wird nach jeweils dieser Anzahl an Iterationen das gefilterte Bild zu diesem Zeitpunkt in die Oberfläche geladen. Dazu später mehr, zunächst soll davon ausgegangen werden, dass keine Änderungen in diesem Fenster vorgenommen werden.

Die Oberfläche ruft dann das Filterprogramm auf und zeigt während dieses läuft folgendes Fenster an:



Abbildung 30: Anzeige während eines Filtervorgangs

Aus technischen Gründen kann eine Fortschrittsanzeige nur jeweils zwischen den berechneten Zwischenresultaten produziert werden, in diesem Fall, in dem keine Zwischenresultate gefordert wurden, also überhaupt nicht. Der Filtervorgang kann mit der Schaltfläche "Abbrechen" vorzeitig beendet werden, man sollte aber beachten, dass der Filter abhängig von seiner Konfiguration und der Bildgröße sehr viel Zeit in Anspruch nehmen kann, auch deshalb weil eventuell nicht genügend viel Hauptspeicher für das Filterprogramm zur Verfügung steht, so dass Auslagerungsmechanismen vom Betriebssystem bemüht werden müssen. Während eines Filtervorgangs kann mit der Oberfläche nicht weiter gearbeitet werden.

Sobald der Filtervorgang abgeschlossen ist wird in der Verwaltung ein neuer Eintrag mit dem Resultat eingefügt und ebenfalls dort die verwendete Filterkonfiguration angezeigt. Das Filterresultat wird automatisch zum Bearbeiten geöffnet.

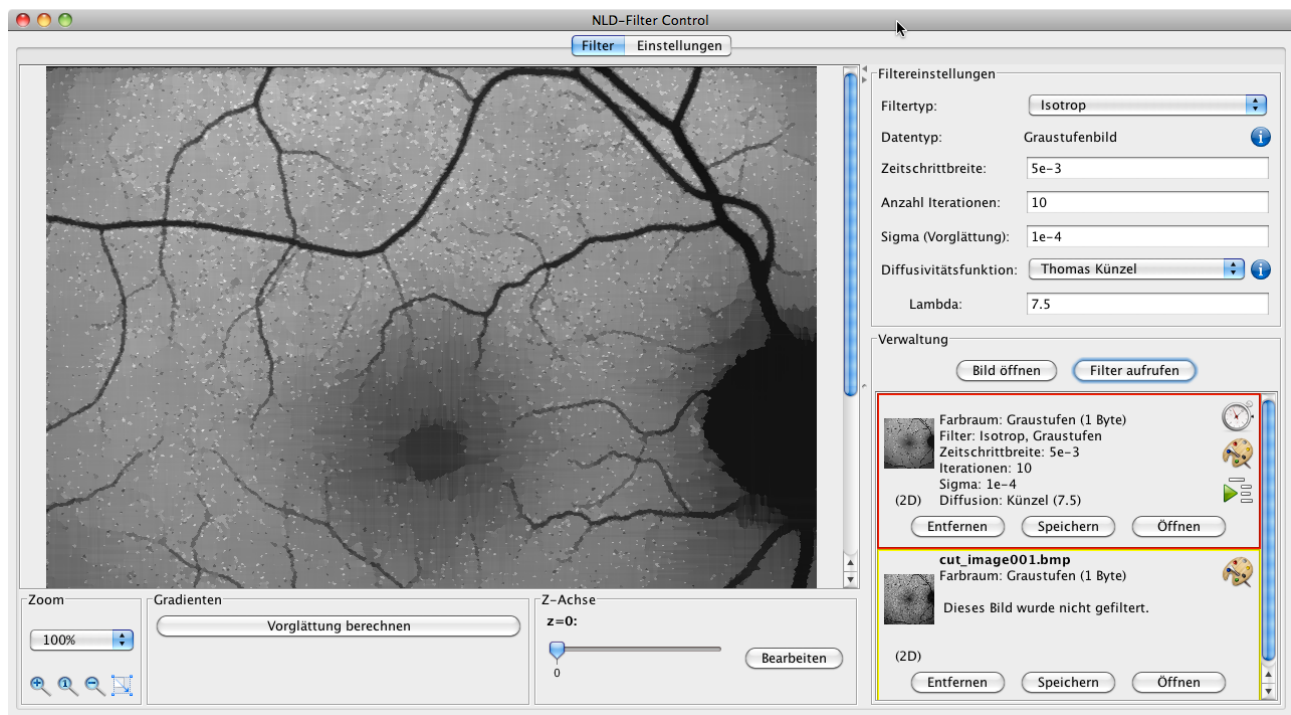


Abbildung 31: Hauptfenster nach einem erfolgreicher Filterlauf

Einträge in der Verwaltung, die durch einen Filterlauf erzeugt wurden, sind zusätzlich dadurch erkennbar, dass sie rechts ein weiteres Symbol einblenden ("Taschenuhr"). Klickt man auf dieses Symbol so werden einige technische Informationen über den Filterlauf der zu diesem Eintrag führte angezeigt, wie beispielsweise die benutzte Kommunikationsmethode, die kommunizierte Datenmenge aber auch interessante Informationen wie die für den Filterlauf benötigte Zeit und die verwendete Kommandozeile zum Aufruf des externen Filterprogramms.

Gibt man in dem weiter oben gezeigten Fenster zur Erzeugung von Zwischenresultaten an, dass solche erzeugt werden sollen, so ergeben sich einige Änderungen im Ablauf. Zunächst kann nun während des Filterlaufs in geringem Umfang eine Fortschrittsanzeige eingeblendet werden. Nach Abschluss des Filterlaufs besitzt der entsprechende Eintrag im Dateimanager oberhalb der Schaltflächen eine weitere Zeile mit Bedienelementen:



Abbildung 32: Verwaltung nach einem Filteraufruf mit Zwischenresultaten

Mit dem Slider kann man durch die verschiedenen Zwischenresultate navigieren, es ändert sich das kleine Vorschaubild und die im Eintrag angezeigten Filtereigenschaften, die zu diesem Resultat führten. Ist der entsprechende Eintrag zudem zum Bearbeiten geöffnet, so wird bei Bewegung des Sliders die Hauptanzeige des Bildes aktualisiert. Man beachte, dass es hierbei zu spürbaren Verzögerungen kommen kann, wenn in den Programmeinstellungen als interne Kodierung die verlustfreie Komprimierung im PNG-Format benutzt wird. Mit dem kleinen Symbol links neben dem Slider kann man direkt zu einem bestimmten Zwischenresultat springen. Ruft man einen Filter mit einem solchen Eintrag mit Zwischenresultaten auf, so wird das aktuell mit dem Slider markierte Zwischenresultat weitergefiltert. Auch die von der Verwaltung gespeicherten Verwandtschaftsinformationen dehnen sich auf diese Zwischenergebnis-Funktionalität in naheliegender Weise aus: Filtert man einen Eintrag mit Zwischenergebnissen weiter, so entsteht wie üblich ein neuer Eintrag. Klickt man in diesem dann das Symbol mit dem grünen Pfeil an, so springt die Verwaltung nicht nur zu dem Elternbild, sondern schiebt den Slider dort auf exakt das Zwischenresultat welches entsprechend weitergefiltert wurde.

3.5 Bearbeiten von Bildern

Im folgenden soll stets vorausgesetzt werden, dass ein Bild zum Bearbeiten geöffnet ist. Das entsprechende Bild wird dann in der großen Ansicht im linken Teil des Hauptfensters dargestellt. Die Oberfläche bietet am linken unteren Ende eine Zoom-Funktionalität an.

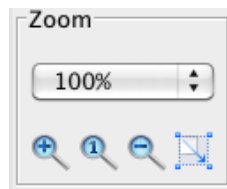


Abbildung 33: Zoomfunktionalität der Oberfläche

Es können Skalierungen im 10% Abstand in einem Bereich von 10% bis 200% gewählt werden. Man beachte, dass für eine große Skalierung ebenfalls eine große Menge an Hauptspeicher erforderlich ist. Außerdem bietet die Zoomfunktionalität einen Modus mit dem Namen "passend" an. In dieser Einstellung wird das Bild optimal skaliert, so dass es exakt in den verfügbaren Platz des Hauptfensters passt ohne Scrollfunktionalität zu benötigen. Die wesentliche Zoomfunktionalität wird auch durch vier Symbole unter der Zoomauswahl bereitgestellt, um eine schnellere Arbeit zu ermöglichen, außerdem ändern sich die Zoomeinstellungen nicht, wenn ein anderes Bild zum Bearbeiten geöffnet

wird und das Programm versucht stets auch die aktuelle Scrollposition beizubehalten, so dass gerade in gefilterten Bildern mit Zwischenresultaten leicht der Einfluss des Filters auf bestimmte Regionen eines Bildes über die Zeit hinweg untersucht werden kann.

Unterhalb der Hauptanzeige blendet die Oberfläche außerdem stets einen Abschnitt für die z -Achse ein. In dem entsprechenden Slider werden alle auf der z -Achse vorhandenen Positionen angezeigt, bei einem 2D-Bild ist dies stets nur $z = 0$. Mit diesem Slider kann man also einfach durch die z -Dimension eines Bildes navigieren.



Abbildung 34: Unterstützung für die z -Achse eines Bildes

Außerdem erscheint in diesem Abschnitt eine Schaltfläche mit dem Titel “Bearbeiten”. Alle Funktionalität zum Bearbeiten eines (3D aber auch 2D) Bildes findet sich in dem Fenster, welches daraufhin geöffnet wird. Man beachte, dass alle Änderungen, die in diesem Fenster vorgenommen werden erst in dem Moment tatsächlich ins Bild übernommen werden, wenn das Fenster mit der Schaltfläche “Ok” geschlossen wird. Hat man also während des Bearbeitens einen Fehler begangen kann man stets auf “Abbrechen” klicken und alle Änderungen gehen verloren.

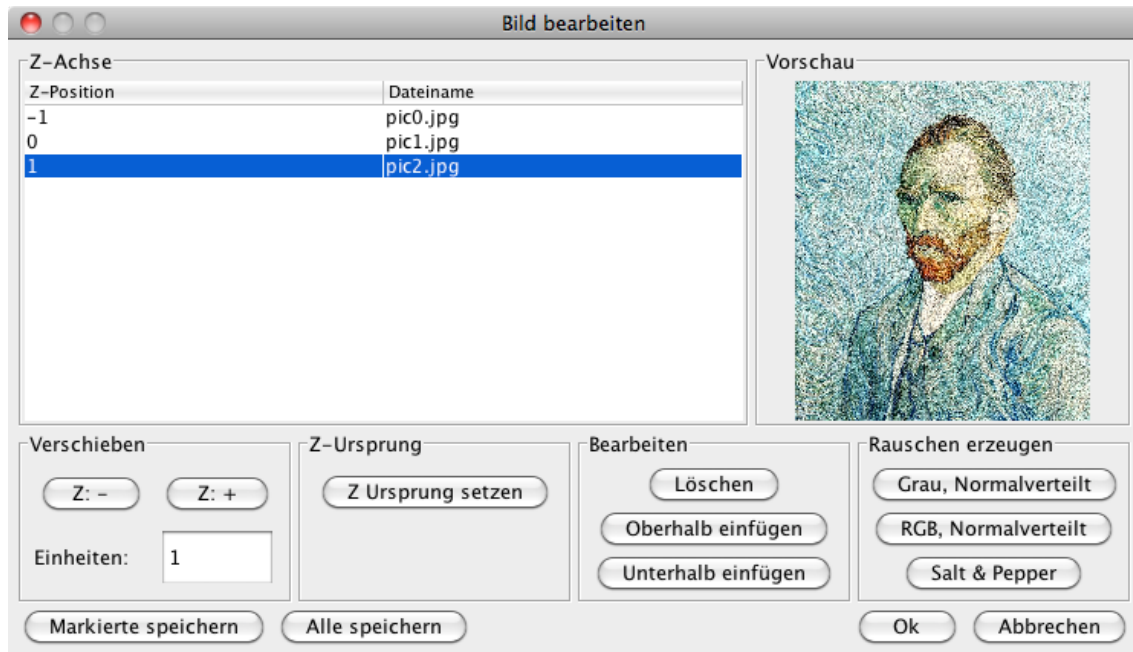


Abbildung 35: Bearbeiten eines Bildes

Im obersten Fensterabschnitt befindet sich die Auswahl von 2D-Teilbildern auf der z -Achse. Wird nur eines ausgewählt, so kann dort eine Vorschau angezeigt werden, werden mehrere ausgewählt entsprechend nicht. Die Funktionalität, die im Bereich darunter bereitgestellt wird bezieht sich stets auf die markierten Teilbilder. Zunächst lassen sich diese um eine bestimmte Einheit auf der z -Achse verschieben. Man beachte, dass alle Funktionen stets versuchen, das Bild an der Position $z = 0$ fixiert zu halten. Dem liegt die Überlegung zugrunde, dass man sich auch beim Filtern eines 3D-Bildes eventuell nur für ein ganz bestimmtes 2D-Bild interessiert. Legt man dieses dann an die Position $z = 0$, so kann man es stets wiederfinden. Möchte man ein anderes Bild in diesen Ursprung legen, so ist dies explizit mit der Schaltfläche “Z Ursprung setzen” anzukündigen - die z -Achse wird daraufhin entsprechend so umskaliert, dass das gewünschte Bild in $z = 0$ liegt. Der nächste Abschnitt der angebotenen Funktionen ermöglicht es alle ausgewählten Bilder zu löschen (geschieht ohne Rückfrage!) bzw. unterhalb des letzten markierten Bildes oder oberhalb des ersten markierten Bildes neue Bilder (von Dateien auf Datenträgern) einzufügen. Auch hier können im entsprechenden Dialogfenster stets mehrere Dateien ausgewählt werden, die dann lexigrafisch an der entsprechenden

Position eingefügt werden. Hierbei wird auch entsprechend die z -Achse so umskaliert, dass das Bild, das vorher an der Position $z = 0$ lag, auch nach dem Einfügen noch dort liegt.

Im letzten Teil werden schließlich Funktionen zum künstlichen Erzeugen von Rauschen bereitgestellt. Stets verfügbar ist “Salt & Pepper”, welches einen gewissen Prozentsatz der Bildpixel willkürlich mit schwarzen und weißen Pixeln überschreibt und das graue, normalverteilte Rauschen, bei welchem man die Varianz des Rauschens vorgeben kann. Es wird dann mit der entsprechenden Verteilung additiv ein Rauschen erzeugt, bei RGB-Bildern in allen Farbebenen dasselbe. Bei RGB-Bildern steht weiterhin ein RGB normalverteiltes Rauschen zur Verfügung, welches analog zu dem grauen ist, jedoch alle drei Farbebenen getrennt verrauscht. Auch die Rauscherzeugung wird stets in allen markierten Bildern durchgeführt.

Am unteren Rand findet sich noch eine Möglichkeit die markierten bzw. alle Bilder zu speichern. Das Speichern aller Bilder entspricht der Schaltfläche “Speichern” in der Verwaltung, bezieht jedoch alle Änderungen mit ein, die in diesem Fenster vorgenommen wurden, auch wenn diese noch nicht übernommen wurden. “Markierte Speichern” funktioniert analog, bietet jedoch als weitere Indizierungsmöglichkeit die aufsteigende Indizierung relativ zu der Markierung an, d.h. man erhält eine kontinuierliche Indizierung anstatt einer Indizierung mit Lücken die sich aus den Lücken zwischen den markierten Bildern ergeben.

4 Details der Implementierung des Filters

Die Filtercodes sind vollständig in der Sprache C++ formuliert und lassen sich mit der GNU Compiler Collection (GCC) kompilieren, beispielsweise auch unter Windows mit dem vom MinGW bereitgestellten GCC. Da sie aber lediglich die C++-Standardbibliotheken benutzen, sollte es prinzipiell möglich sein, einen beliebigen C++ Compiler zu verwenden. Die Codes zerfallen dabei in zwei Teile: Zum einen Codes für ein Konsolen-Rahmenprogramm, welches gerade darauf ausgerichtet ist, von der grafischen Oberfläche (ein Java Programm) aufgerufen zu werden und zum anderen die Codes im Unterordner `src/filter` welche die eigentliche Implementierung der nichtlinearen Diffusionsfilter bereitstellen und von den Codes des Rahmenprogramms verwendet werden. Diese Implementierung der Diffusionsfilter ist jedoch nicht an das vorliegende Rahmenprogramm und die grafische Oberfläche gebunden. Sie lassen sich ohne weiteres auch direkt in andere Software integrieren. Im Folgenden wird in groben Zügen die Struktur dieser Implementierung der Filter vorgestellt. Als Referenz dienen die kommentierten Quellcodes. Folgendes Diagramm verdeutlicht die Vererbungsstruktur zwischen den verschiedenen Filterklassen:

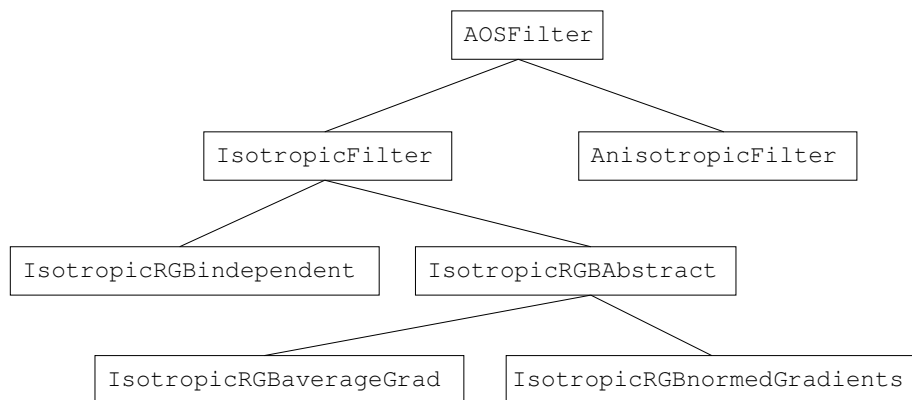


Abbildung 36: Vererbungsstruktur der Filterklassen

Neben diesen Klassen werden in einem Unterordner `src/filter/diffusionfunctions` die verschiedenen Diffusionsfunktionen bereitgestellt, welche alle die abstrakte Klasse `DiffusionFunction` implementieren. Die oben aufgeführten Klassen benutzen dabei ebenfalls diese abstrakte Klasse, um auf die Diffusionsfunktionen zuzugreifen.

`AOSFilter` auf oberster Ebene stellt eine Implementierung des AOS-Verfahrens bereit und benutzt als GleichungssystemlÖser den Thomas-Algorithmus. Ein konkreter Filter, der zum Lösen der Differentialgleichung eine Diskretisierung mit dem AOS-Verfahren verwendet (was alle implementierten Filter tun) muss von dieser Klasse erben. Die Klasse `AnisotropicFilter` implementiert auf dieser Grundlage einen anisotropen Graufilter für 2D-Daten. Sie ist damit vom gesamten Rest vollkommen unabhängig. Auf ihr aufbauend könnte man beispielsweise einen anisotropen Farbfiler implementieren.

Als Grundlage für alle isotropen Filter dient die Klasse `IsotropicFilter`, welche selbst einen isotropen Filter für n-dimensionale Graudaten implementiert. Hierauf aufbauend werden die verschiedenen Typen von Farbfilttern bereitgestellt, welche größere Teil der Codes von `IsotropicFilter` weiterverwenden können. Da zwischen den Farbfilttern `IsotropicRGBAverageGrad` und `IsotropicRGBNormedGradients` weiterhin viele Gemeinsamkeiten bestehen werden diese gemeinsam verwendbaren Codes in der Zwischenklasse `IsotropicRGBAbstract` bereitgestellt.

Um einen Filter zu verwenden, muss eine Instanz der entsprechenden Klasse erzeugt werden. Der Konstruktor erwartet dabei die Filterparameter als Argumente. Als Ergebnis erhält man dann einen konkreten Filter, der auf Bilddaten, die in Form eines Vektors kodiert sind, angewendet werden kann. Zu diesem Zweck stellen alle Filterklassen eine Methode `filter` bereit, deren Parameter jedoch je nach Filtertyp unterschiedlich sind.

In den gesamten Codes wird die Bibliothek `MathTL` der AG Numerik / Wavelet-Analyse verwendet, um Vektoren zu definieren. Beim Kompilieren der Codes muss entsprechend diese Bibliothek verfügbar sein.

5 Details der Implementierung der grafischen Oberfläche

5.1 Grobübersicht und Grundkonzepte

Dieser Abschnitt soll als Einführung in die umfangreichen Quellcodes der grafischen Oberfläche dienen. Er kann nicht den Anspruch der Vollständigkeit erheben, lediglich einige Grundkonzepte sollen verdeutlicht werden. Als universelle Referenz dienen die ausführlich kommentierten Quellcodes.

Die Oberfläche ist in der Sprache Java formuliert und greift zur Erzeugung der grafischen Oberfläche auf die Swing-Bibliothek zurück, die in jeder Java Installation vorhanden ist. Daher ist das konkrete Aussehen der Oberfläche vom Betriebssystem und der installierten Java Version abhängig. Die Swing-Bibliothek zwingt den Programmierer in nahezu jeder Hinsicht objektorientiert zu arbeiten, was aber letztlich zu einem schönen Programmdesign führt, sogar im Fall dieser Software, in der nicht zu Beginn ein festes Konzept vorlag, sondern die im Laufe der Zeit schrittweise erweitert wurde. Entsprechend bestehen die Quellcodes aus einer Sammlung von Klassen die instanziiert werden, ein Teil dieser Instanzen bietet eine grafische Schnittstelle zum Benutzer hin an.

Der Programmeinsprungspunkt findet sich in `NLDCControl/FilterControl.java`. Hier werden Systemüberprüfungen durchgeführt und anschließend das Hauptfenster, welches in `NLDCControl/windows/WindowMain.java` implementiert wird, angezeigt, indem eine Instanz dieser Klasse erzeugt wird. Dabei ist zu beachten, dass die Codes in `FilterControl.java` nicht direkt das Hauptfenster erzeugen. Um dies zu verstehen muss man sich in das Thread-Konzept von Swing einarbeiten, was dringend anempfohlen wird - an dieser Stelle kann nur eine kleine Einführung gegeben werden. Swing ist grundsätzlich nicht thread-safe, es dürfen also nicht verschiedene Threads im Programm Methoden von Swing-Klassen aufrufen. Swing zieht daraus die Konsequenz vom Programmierer ein striktes Thread Konzept zu erzwingen. Wenn Java in das Programm eintritt erzeugt es zu diesem Zweck einen einzelnen Thread, der in Swing-Terminologie als initialer Thread bezeichnet wird. Methoden von Swing-Klassen dürfen allerdings nur von einem einzigen, von Swing erzeugten Thread aufgerufen werden, welcher event-dispatching-thread heißt. Dieser Thread verteilt, wie der Name bereits andeutet, Ereignisse wie beispielsweise Interaktionen des Benutzers mit der Oberfläche, auf die entsprechenden Ereignissbehandlungen in den Instanzen von Klassen, welche die Oberfläche verfügbar machen. Da unter Umständen sehr viele solcher Ereignisse erzeugt werden rät Swing dazu, dass die Behandlungsmethoden keine zeitaufwändigen Arbeiten verrichten sollten. Ansonsten müsste der event-dispatching-thread zunächst diese Aufgaben abarbeiten bevor er eine andere Aufgabe verteilen kann. Da auch Forderungen des Betriebssystems, beispielsweise solche die verlangen, dass sich ein Teil der Oberfläche neu zeichnet, durch diesen Thread behandelt werden, würde dies bedeuten, dass alle diese Aufgaben nicht mehr wahrgenommen werden können, aus Sicht des Benutzers würde die Oberfläche nicht mehr reagieren. Daher führt Swing - unterstützt durch spezielle Klassen - das Konzept der sogenannten worker-threads ein: Immer wenn an einer Stelle im event-dispatching-thread eine größere Arbeit zu verrichten ist, sollte zu diesem Zweck ein eigener worker-thread angelegt werden, damit die Oberfläche stets interaktiv bleibt. Die einzige Stelle, an der die Codes der Oberfläche von diesem Konzept gebrauch machen ist der Filtervorgang.

Für dieses Programm bedeutet dass zunächst, um zu `FilterControl.java` zurückzukehren, dass die Codes dort nicht direkt das Hauptfenster erzeugen dürfen, da hierfür Swing-Methodenaufrufe benötigt werden, die Codes in `FilterControl.java` jedoch im initialen Thread laufen. Daher registrieren die Codes dort am Ende lediglich eine bestimmte (ebenfalls in dieser Klasse implementierte) Methode für die Ausführung durch den event-dispatching-thread. Diese Registrierung sorgt dann dafür, dass Swing diesen neuen Thread erzeugt und als erste Aufgabe das Hauptfenster erzeugen lässt. Der initiale Java-Thread wird kurz darauf beendet, da er von diesem Moment an keine Aufgaben mehr zu erledigen hat - das Programm befindet sich dann rein im objekt- und ereignisorientierten Swing-Setting.

Wie bereits erwähnt ist das Programmdesign stark objektorientiert gehalten. Beispielsweise ist die Verwaltung in der Oberfläche (der Abschnitt rechts unten) komplett in der Klasse `NLDCControl/controls/CtlFileManager.java` implementiert. Instanzen dieser Klasse (tatsächlich wird nur eine einzige erzeugt) stellen die grafische Oberfläche der Liste in der Verwaltung bereit (im wesentliche die Scrollfunktionalität). Dazu wird ganz wesentlich Gebrauch von der privaten Unterklasse `CtlElement` gemacht. Diese Unterklasse stellt als grafische Oberfläche die Listeneinträge zur Verfügung, d.h. das Programm erzeugt für jedes geladene Bild eine Instanz dieser Klasse. `CtlElement` stellt jedoch nicht nur die grafische Oberfläche eines Listeneintrags bereit. Tatsächlich werden von diesen Instanzen auch die Bilddaten im Hauptspeicher verwaltet, d.h. die grafischen Einträge in der Liste der Verwaltung sind tatsächliche die Objekte, welche auch die zugehörigen Daten zu dem Eintrag speichern und verwalten.

Als weiteres Beispiel ist der komplette Abschnitt zur Konfiguration eines Filters in der Klasse `NLDCControl/controls/CtlFilterProperties.java` implementiert, die sowohl die grafische Schnittstelle hierfür bereitstellt als auch den größten Teil der Kommandozeile für den Filteraufruf darauf aufbauend erzeugt. Alle diese einzelnen Komponenten, die Teilfunktionalität implementieren werden dann schließlich vom Hauptfenster zusammengefügt. Dieses sorgt

dafür, dass Ereignisse von Einzelkomponenten - beispielsweise löst das Klicken auf “Öffnen” in einem Verwaltungseintrag ein Ereignis aus, welches das Hauptfenster behandelt - passend verteilt werden und stellt damit sicher, dass diese Einzelkomponenten sinnvoll zusammenarbeiten.

Grob sind die Quellcodes anhand ihrer Ordnerhierarchie strukturiert:

- **NLDCControls:** Enthält Klassen die ganz allgemein von vielen weiteren Klassen benötigt werden und den Programmeneinsprungspunkt
- **NLDCControls/windows:** Enthält Klassen, welche Fenster der Oberfläche implementieren und solche mit denen das Resultat solcher Fenster kommuniziert werden kann
- **NLDCControls/controls:** Enthält komplexere Komponenten (“Steuerelemente”), die immer wieder benötigt werden oder Teilfunktionalität in sich abgeschlossen bereitstellen
- **NLDCControls/images:** Hier werden die Grafiken bereitgestellt, welche die Anwendung benötigt und eine Klasse, damit der Rest der Codes leicht auf diese Ressourcen zugreifen kann, ohne konkret wissen zu müssen, wie sie gespeichert werden
- **NLDCControls/extern:** Hier liegt eine Klasse von Sun und eine vom W3C (beide unter freien, auch kommerziell verwendbaren Lizenzen), welche benötigt werden, aber nicht durch eigene Codes ersetzt wurden.

5.2 Verwaltung der geladenen Bilder

Als Typ für ein (gegenenfalls 3D) Bild wird im gesamten Programm der Typ `LinkedList<Picture>` verwendet, wobei `LinkedList` eine im JDK vorliegende Implementierung einer verketteten Liste darstellt und die Typklasse unter `NLDCControl/Picture.java` zu finden ist. `Picture.java` kümmert sich dabei um die interne Kodierung des Bildes im Hauptspeicher, d.h. wenn gewünscht um die Komprimierung und Dekomprimierung von Bilddaten und die Einträge der `LinkedList` entsprechen den verschiedenen *z*-Positionen des vorliegenden Bildes. Als Austauschformat für dekodierte Bilder wird `BufferedImage` verwendet, eine von AWT im JDK bereitgestellte Klasse, welche nahtlos mit Swing-Methoden zusammenarbeitet und die Bilddaten unkomprimiert beinhaltet. Entsprechend werden von `Picture` Methoden bereitgestellt, mit welchen man das verwaltete Bild in Form eines `BufferedImage` auslesen kann und Methoden, welche das verwaltete Bild durch ein in Form eines `BufferedImage` gegebenes Bildes ersetzen. Durch diese Struktur können alle Details über die Repräsentation von Bildern im Hauptspeicher in der Klasse `Picture` abstrahiert werden, ein weiteres Beispiel für das grundsätzlich objektorientierte Design der gesamten Quellcodes. Um Zwischenresultate von Filtervorgängen (im Code als “Bilderserien” bezeichnet) zu verwalten wird als Format (beispielsweise in `CtlFileManager`) der Typ `LinkedList<IntermediatePicture>` verwendet, wobei die Typklasse unter `NLDCControl/IntermediatePicture.java` zu finden ist. Letztlich ist dieser Typ eine Verkettung von `LinkedList<Picture>` Bildern zusammen mit Informationen über die Skalierung der *z*-Achse, welche von `LinkedList<Picture>` nicht bereitgestellt wurde. Dieser Typ taucht jedoch nur an den Stellen auf, an denen Zwischenresultate erzeugt oder verwaltet werden, d.h. in den Codes, die den Filtervorgang steuern (siehe den nächsten Abschnitt) und in `CtlFileManager`.

5.3 Verwaltung des Filtervorgangs durch einen worker-thread

Die komplette Verwaltung des Filtervorgangs wird durch zwei Klassen bereitgestellt:

`NLDCControl/windows/WindowFilter.java` und `NLDCControl/NLDFilter.java` Die erste dieser beiden Klassen implementiert dabei das Wartefenster, welches während des Filtervorgangs angezeigt wird. Tatsächlich wird der Prozess des Filterprogramms komplett in diesem Fenster gesteuert. Die zweite Klasse dient der Kommunikation mit dem und der Erzeugung des Filterprozesses selbst. Da der Filterprozess unter Umständen viel Zeit benötigt und die Codes der Oberfläche auf das Terminieren dieses Prozesses warten müssen bzw. darauf, dass er alle Resultate kommuniziert hat, steht es außer Frage die Verwaltungscodes für den Filterprozess im event-dispatching-thread ablaufen zu lassen. `WindowFilter` erzeugt zu diesem Zweck einen separaten worker-thread, welcher in der privaten Unterklasse `NldFilterThread` von `WindowFilter` implementiert ist. In diesen Codes muss man stets im Hinterkopf behalten, welche Methode in welchem Thread ausgeführt wird, da der worker-thread keine Swing-Methoden aufrufen darf. Die Codes in `NldFilterThread` kümmern sich daher in erster Linie um die Kommunikation zwischen event-dispatching-thread und worker-thread aber auch um die Behandlung des Filterns mit Zwischenresultaten und um einen Teil der Übersetzung zwischen der Oberflächenwelt und der Filterprogramm-Welt. Der wesentliche Teil des worker-threads jedoch ist in der Klasse `NLDFilter.java` implementiert - die Codes in dieser Klasse müssen sich der Tatsache bewusst sein, dass sie nie mit dem event-dispatching-thread ausgeführt werden, sondern stets mit

einem worker-thread, sie dürfen also nicht auf Swing-Methoden zurückgreifen. Entsprechend muss dieser Teil der Übersetzung zwischen beiden Welten bereits von `WindowFilter` erledigt werden.

Die Codes in `NLDFilter.java` hingegen sind für die Erzeugung des Filterprozesses zuständig und für die korrekte Kommunikation der zu filternden Bilddaten bzw. der Resultatdaten des Filterlaufs. Hierzu gehört auch das Kodieren der Bilddaten als Spaltenvektor, wie es das Filterprogramm erwartet. Außerdem muss man in diesen Codes sehr vorsichtig sein, da hier große Datenmengen transportiert und verrechnet werden. Aktuell bedeutet dies, dass die Codes so optimiert sind, dass beim Zugriff auf die gegebenen Bilddaten möglichst selten die Einzelbilder des zu filternden Bildes dekodiert werden müssen und möglichst jedes z -Bild nur einmal (höchstens aber drei mal) kodiert wird. Ansonsten finden sich hier auch die Codes, die auf das Terminieren des Filterprozesses warten.

Auf Umwegen werden über eine Instanz der Klasse `NLDControl/FilterProcess.java` alle Informationen über worker-thread und Filterprozess zurück bis zu `WindowFilter` kommuniziert, damit dieses eine Funktionalität zum Abbrechen des Filtervorgangs bereitstellen kann. Man beachte, dass diese Funktionalität aktuell den worker-thread mit der von Java bereitgestellten Methode `stop()` mit sofortiger Wirkung beendet. Diese Methode ist eigentlich als deprecated gekennzeichnet, sollte also nicht verwendet werden, da sie zu Synchronisationsproblemen (im Extremfall sogar Deadlocks) mit den übrigen Threads des Programmprozesses führen kann. Beim erweitern der Quellcodes sollte man dies stets im Hinterkopf behalten. Aktuell entstehen keine Probleme, da es im Grunde während der worker-thread läuft überhaupt keinen relevanten Datenaustausch zwischen event-dispatching-thread und worker-thread gibt und daher auch keine Synchronisierungsmechanismen benutzt werden. Daher ist es in den momentan vorliegenden Codes legitim die `stop()` Methode zu benutzen, ansonsten gibt es keine saubere Möglichkeit den worker-thread anzuhalten, gerade auch weil er an einer bestimmten Stelle auf den Filterprozess warten muss.

6 Zusammenfassung und offene Probleme

Alle eingangs gestellten Ziele wurden erreicht. Die mit [1] bereitgestellten Codes wurden gründlich überarbeitet und damit eine praktische Verwendung auch außerhalb des Settings der grafischen Oberfläche und des Rahmenprogramms ermöglicht. Mit der grafischen Oberfläche steht nun weiter eine komfortable Möglichkeit bereit, um in Anwendungen mit dem Filter zu arbeiten und zu experimentieren.

Gleichzeitig zeigt aber die Implementierung des anisotropen Filters die Grenzen des Ansatzes auf: Es müsste gemäß [2] ein größerer Differenzenstern zur Approximation des Divergenzoperators implementiert werden. Alternativ könnte man in Erwägung ziehen das AOS als Löser für die Differentialgleichungen zugunsten eines ausgefeilteren Verfahrens aufzugeben. Als Möglichkeiten würde sich eventuell ein Ansatz mit Finite-Elemente-Methoden anbieten, beispielsweise unter Verwendung des Deal II Projekts (<http://www.dealii.org/>) der Numerical Methods Group der Universität Heidelberg oder ein Wavelet-basierter Ansatz. Da solche Ansätze scheinbar in der Bildverarbeitung grundsätzlich noch nicht weit verbreitet sind müssten hierfür aber zunächst noch Untersuchungen über die Eignung angestellt werden.

Des Weiteren ist die Wahl der für ein Bild und ein gewünschtes Verhalten des Filters passenden Filterparameter relativ schwierig. Hier sollten durch Tests und mathematische Untersuchungen zumindest Richtlinien erarbeitet werden, die es erleichtern die Parameter zu finden und richtig zu verstehen.

Eine Möglichkeit wäre auch die gezielte histogrammbasierte Manipulation bestimmter Klassen von Häufigkeiten im Bild. Dazu könnte in der grafischen Oberfläche ein Statistik-Modul bereitgestellt werden, welches auf einfache Weise Histogramme erstellt, um darauf aufbauend günstige Parameter zu finden.

Literatur

- [1] Thomas Künzel *Nichtlineare Diffusion: Theoretische Analyse und Anwendungen bei Strömungssimulationen* Diplomarbeit an der Philipps Universität Marburg, 2007
- [2] Joachim Weickert *Anisotropic Diffusion in Image Processing* B. G. Teubner Stuttgart Verlag, 1998
- [3] Pavel Mrázek, Joachim Weickert *Diffusion-Inspired Shrinkage Functions and Stability Results for Wavelet Denoising* International Journal of Computer Vision 64(2/3), 171–186, 2005
- [4] Frank Eckhardt, Fabian Feise *Fortgeschrittenenpraktikum* Ergebnisbericht an der Philipps Universität Marburg, 2010
- [5] Thomas Brox *Von Pixeln zu Regionen: Partielle Differentialgleichungen in der Bildanalyse* Dissertation, Fakultät für Mathematik und Informatik, Universität des Saarlandes, 2005