Ambient Occlusion für Blutfluss repräsentierende Integrallinien

Hannes Seibt

Mat.-Nr: 209851

Hannes.Seibt@st.ovgu.de

Betreut durch:

Dr.-Ing. Benjamin Köhler,

Prof. Dr.-Ing Bernhard Preim

Otto-von-Guericke-Universität Magdeburg

Institut für Simulation und Graphik

28. Februar 2017

Abstract

Bei einem gesunden Menschen sind Hämodynamik und kardiale und zerebrale Morphologie so aufeinander abgestimmt, dass ein effizienter Bluttransport vom Herz in die Gefäße ermöglicht wird. Zerebrale Gefäßerkrankungen sowie kardiovaskuläre Störungen führen dazu, dass dieses Zusammenspiel gestört wird und sich charakteristische Blutflussmuster in den Gefäßen und im Herzen ändern. Häufig sind es helixförmige Flussmuster, welche die Scherkräfte an den Gefäßwänden erhöhen und zu einer Deformation des Gefäßes führen, so wie es beispielsweise bei Aneurysmen geschieht. Um eine frühzeitige Diagnose sowie Therapie-Entscheidungen zu ermöglichen, ist es wichtig, dass den behandelnden Ärzten aussagekräftige Flussanimationen zur Verfügung stehen. In dieser Arbeit wurde der für statische Diffusions-Tensor-Bildgebung entwickelte LineAO-Algorithmus von Eichelbaum et al. [1] auf gemessene, aortale 4D-Blutflussdaten angewandt und entsprechend der Anforderungen an eine Animation angepasst.

Inhaltsverzeichnis

I Einleitung und Grundlagen

Zu Beginn werden einige, diese Arbeit betreffende Grundlagen erläutert sowie für das Thema im Allgemeinen motiviert. Das heißt, es wird zunächst diskutiert, wo qualitative Blutflussanalyse eingesetzt wird und welche klinische Relevanz sie besitzt. Weiterhin werden alle wichtigen Schritte, welche zur Erstellung von Blutflussanimationen nötig sind, kurz vorgestellt. Hierzu zählen die Datenakquise und die Vorverarbeitung der Daten, welche jeweils von der Modalität der Bilder abhängt. Abschließend soll kurz besprochen werden, welche weiteren Beleuchtungs- und Animationstechniken es gibt, wieso sich in dieser Arbeit für LineAO entschieden wurde und welche Vor- und Nachteile dies mit sich bringt.

I.a Qualitative Blutflussanalyse

Am häufigsten tritt in der Aorta und Pulmonalarterie laminarer Blutfluss mit parabelförmigem Geschwindigkeitsprofil auf, was bedeutet, dass Flussgeschwindigkeiten im Zentrum des Gefäßes am höchsten sind. Ihnen gegenüber stehen spiralförmige Blutflussmuster, welche jedoch nicht nur krankheitsbedingt auftreten, sondern sogar teilweise als optimale Blutflussmechanismen anzusehen sind. In der aufsteigenden Aorta und im Aortenbogen konnte bei zehn gesunden Probanden von Kilner et al. [2] ein durch die bogenförmige Geometrie der Aorta hervorgerufener, spiralförmiger Blutfluss nachgewiesen werden. Es wird teilweise sogar angenommen, dass dieser spiralförmige Fluss in der Aorta Arteriosklerose vorbeugen kann [3] [4]. Weitere Strukturen, in denen spiralförmiger Blutfluss bei gesunden Menschen beobachtet werden konnte, sind die Pulmonalarterie sowie der linke und rechte Ventrikel, um nur einige exemplarisch aufzuführen.

Andererseits sind in einigen Fällen spiralförmige Blutflussmuster[5] ein Indikator für krankhafte Veränderungen. So können nach Verengungen, Dilatationen, Stenosen oder Aneurysmen in der Aorta strudelartiger Blutfluss auftreten. Neben der Aorta können auch weitere Strutkuren, wie die Pulmonalarterie oder der linke Ventrikel, betroffen sein, worauf hier jedoch nicht im Detail eingegangen werden soll.

Ein besonders hohes Risiko stellen Aneurysmen, speziell zerebrale Aneurysmen, dar, welche bei Ruptur eine Sterblichkeitsrate von über 52% aufweisen [6]. Basierend auf der patienten-spezifischen Aneurysma-Anatomie können Forscher mittels numerischer Strömungsmechanik Simulationen des Blutflusses erstellen oder mittels 4D PC-MRI den Fluss messen, um den Zusammenhang zwischen Blutflusscharakteristika und Aneurysma-Morphologie zu untersuchen.

Qualitative Analyse der Blutflussmuster ist in allen beschriebenen Fällen hilfreich, um die Diagnose und Therapieentscheidungen zu treffen. Ein unübliches Flussmuster zu analysieren und dadurch zu charakterisieren, kann bei der Bestimmung des Schweregrades und somit der Handlungs-Dringlichkeit nützlich sein. Dies gilt besonders für Aneurysmen. Ziel ist es, anhand einer qualitativen Blutflussanalyse abschätzen zu können, wie das Aneurysma wachsen wird, wie

hoch das Risiko einer Ruptur ist und welche Therapieentscheidung sich daraus ergibt. Hierfür sind aussagekräftige und vor allem anpassbare Blutflussvisualisierungen notwendig, um es zu ermöglichen, Blutflussdaten gezielt zu explorieren.

I.b Datenakquise

Da das Anwendungsbeispiel dieser Arbeit Blutflussdaten sind, soll im Folgenden kurz darauf eingegangen werden, wie diese erzeugt werden. Im gegebenen Fall handelt es sich um Blutflussdaten der aufsteigenden Aorta und des Aortenbogens, welche mittels 4D PC-MRI (four-dimensional phase-contrast magnetic resonance imaging) gemessen wurden. Eine andere Möglichkeit stellt Simulation mittels CFD (Computational Fluid Dynamics) dar. Beide Methoden bringen ihre Vor- und Nachteile mit. So sind gemessene Daten zwar realistischer und näher an der Physiognomie des Patienten, dafür sind sie jedoch sehr viel niedriger aufgelöst und sind häufig stark verrauscht. Simulierte Daten sind hingegen sehr hoch aufgelöst und rauschfrei, dafür ist das Ergebnis der Simulation von sehr vielen, zu Beginn der Simulation zu bestimmenden, Parametern abhängig.

Wie bereits erwähnt, handelt es sich beim genutzten Datensatz um einen gemessenen Blutflussdatensatz des linken Aortenbogens. Er wurde mittels des 4-stufigen Runge-Kutta-Verfahrens integriert. Um zu gewährleisten, dass ein dichter Datensatz entsteht, wurde durch eine Routine sichergestellt, dass jedes Voxel von mindestens einer Integrallinie durchkreuzt wird. Anschließend wurden unwesentliche Elemente, welche keine weitere Einsicht generieren, entfernt. Hierzu zählen zu kurze Integrallinien oder zu dicht beieinander liegende. Abschließend wurden alle Integrallinien mit einem 1-D Binomialfilter geglättet. Die Oberfläche der Aorta wurde mittels Marching Cubes extrahiert.

Der resultierende Datensatz besteht aus 3169 Linien, welche wiederum zusammen aus 644 104 Punkten bestehen. Die Animationszeit beträgt 18 s. Für jeden Punkt liegt ein Zeitstempel und die jeweilige Geschwindigkeit vor.

I.c Beleuchtungsrechnung und -perzeption

Im Laufe der ca. letzten 50 Jahre wurden verschiedene Shading-Verfahren entwickelt, welche meist zum Ziel haben, eine Szene möglichst realitätsgetreu zu rendern. Einige dieser Verfahren sind echtzeitfähig, andere wiederum liefern realistischere Ergebnisse, sind jedoch aufgrund aufwändigerer Berechnungen nicht mehr echtzeitfähig. Eine besondere Position kommt dem Rendern von Linien zu, da für eine Linie keine eindeutige Normale bestimmt ist, welche für die meisten Shading-Verfahren jedoch nötig ist. Hinzu kommt, dass im Vergleich zu natürlichen Szenen, sehr viele sich überschneidende und durchkreuzende Oberflächen existieren. Diesbezüglich sind besondere Methoden erforderlich um Echtzeitfähigkeit zu realisieren.

An dieser Stelle eine ausführliche Einführung in die Historie des Linienrenderns zu präsentieren, würde den Rahmen dieser Arbeit sprengen. Daher werden im Folgenden nur einige wenige Techniken exemplarisch vorgestellt. Eine der meistzitierten Veröffentlichungen zu diesem Thema waren *Illuminated Streamlines* [8]. Zöckler et al. stellten eine auf Linienrendering zugeschnittene Shader-Methode vor, welche Echtzeitfähigkeit erreichte und dabei die räumliche Wahrnehmung von Größenverhältnissen und räumlichen Relationen verbessern konnte. Mit fortschreitender Entwicklung wurde es möglich globale Beleuchtungseffekte zu realisieren, von welchen gezeigt wurde, dass sie für die Wahrnehmung der Position und räumlichen Relationen eines Objektes sehr wichtig sind [9][10]. Da globale Beleuchtung jedoch immer noch nicht echtzeitfähig realisierbar ist, wurden Techniken entwickelt, welche phänomenologisch globale Beleuchtungseffekte approximieren. Eine von diesen ist LineAO, um welche es im Folgenden noch häufig gehen wird.

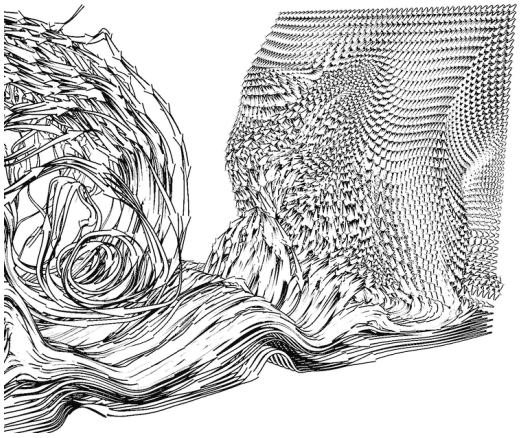


Abbildung 1: *Kombination von ikonographischen Visualisierungstechniken und Halo-basierten Techniken [7].*

Entgegen diesem zunehmenden Streben nach realistischer Beleuchtung stehen illustrative Techniken, welche darauf ausgelegt sind, besondere Features in den Daten hervorzuheben. Eine der meistgenutzten Techniken unter ihnen sind Halo-basiert. Everts et al. [12] nutzen beispielsweise Halos um die Wahrnehmung der Tiefenstaffelung von Linien zu verstärken. 2015 folgte eine weitere Veröffentlichung von Everts et al. [7] in welcher Halos mit Ikonographischen Elementen kombiniert wurden (siehe Abbildung 1). Eine weitere illustrative Technik, welche besonders darauf abzielt unwesentliche Informationen des Datensatzes auszublenden und die wichtigen komprimiert darzustellen ist in Abbildung 2 zu sehen. Born et al. [11] entwickelten diese auf „Streamtapes" basierende Technik, um charakteristische Blutflussmuster einfach sichtbar zu machen.

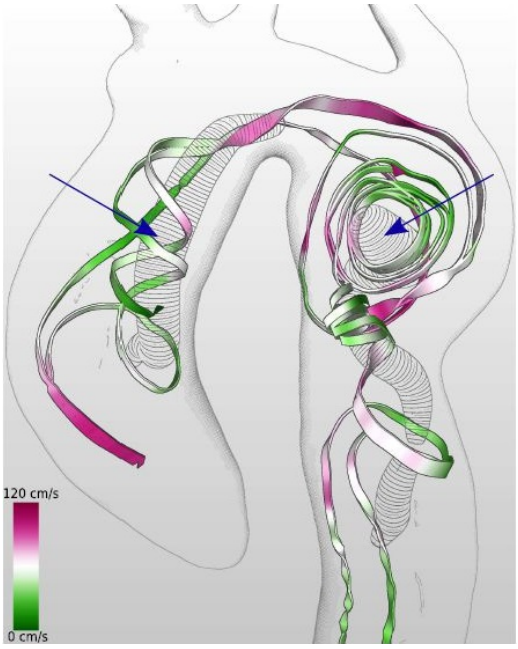


Abbildung 2: *Illustrative Visualisierung auf Basis von „Streamtapes" [11].*

I.d Animationstechniken

Beim Design interaktiver oder animierter Inhalte sind noch weitere Aspekte zu beachten. Zunächst ist zu entscheiden, welche visuelle Repräsentation zu wählen ist. Zu unterscheiden ist zwischen der Darstellung einzelner Partikel oder einer Repräsentation als Pathlet. Pathlets sind all jene Pfadlinien-Punkte, deren zeitlicher Abstand zur aktuellen Partikelposition geringer als ein definierter Schwellwert sind. Zusammen bilden sie ein Liniensegment. Somit wird der Bahnverlauf des Partikels stärker betont als bei der Darstellung einzelner Partikel. Zusätzlich besteht die Möglichkeit temporale Informationen auf Transparenz abzubilden, so dass die Opazität für alle Punkte, mit zunehmendem Abstand zur aktuellen Partikelposition abnimmt.

Das Darstellen einzelner Partikel hingegen ermöglicht es Glyphen, wie Pfeile, Kugeln oder Zapfen [5], in Abhängigkeit von verschiedenen Features darzustellen. Besonderes Augenmerk bei animiertem Blutfluss liegt auf interaktiven Methoden. In [13] können beispielsweise zum Explorieren des Blutflusses Partikel mittels Seedpoints in den Blutfluss eingefügt werden, um ihn qualitativ zu untersuchen. Andere Techniken ermöglichen eine quantitative Untersuchung des Blutflusses mithilfe achsensenkrechter Ebenen zur Anzeige relevanter Features [14].

Ein allgemeiner Überblick über existierende Interaktionstechniken zur Blutflussanalyse wird in [15] gegeben.

II Die LineAO-Methode

Im Folgenden soll in die LineAO-Methode [1] eingeführt werden. Da sie auf Ambient Occlusion beruht und im Speziellen eine Screen Space Methode ist, sollen zunächst diese beiden Verfahren vorgestellt werden.

II.a Ambient Occlusion

Ambiente Beleuchtung ist ein komplexes Problem globaler Beleuchtungsrechnung. Um realistische Ergebnisse zu erzielen, sind meist rechenintensive Verfahren nötig, da die gesamte Szene die Verteilung der Beleuchtung beeinflusst. Um kürzere Renderzeiten zu erreichen, ist es nötig auf phänomenologische Ansätze zurückzugreifen, welche jedoch keine physikalisch korrekten Ergebnisse liefern. Eine solche Methode ist Ambient Occlusion, welche als Grundlage des LineAO im Folgenden kurz vorgestellt werden soll.

Beim Ambient Occlusion wird für jeden Punkt einer Oberfläche ein AO-Wert berechnet, welcher angibt, wie viel ambientes Licht die Oberfläche nicht erreicht. Hierzu wird für den zu betrachtenden Punkt P einer Geometrie eine Oberfläche definiert, deren prozentuale Verdeckung bestimmt werden soll. Besagte Oberfläche ist die Fläche einer Einheitshämisphere, welche entlang der Normalen n am Punkt P ausgerichtet ist. In Abbildung 3 wird besagtes Schema dargestellt.

Dies führt zur Standard Ambient Occlusion Gleichung, welche von Kainz et al. [1] später für ihre LineAO-Methode adaptiert wird:

$$AO(P) = \frac{1}{\pi} \int_{\Omega} (1 - V(\omega, P)) \langle \omega n \rangle d\omega. \quad (1)$$

Das Integral über die Einheitshemisphäre Ω stellt die Oberfläche dar, der Sample-Vektor ω zeigt von Punkt P zur Oberfläche der Hemisphäre und die binäre Sichtbarkeitsfunktion V ermittelt, ob in Richtung des Sample-Vektors ω eine Verdeckung der Einheitshemisphäre vorliegt oder nicht. Das Skalarprodukt aus n und ω fungiert hierbei als eine Art Gewichtung und gibt an, wie viel Lichtenergie am Punkt P aus Richtung ω angelangt.

Da diese Gleichung schwer analytisch lösbar ist, wird in der Praxis eine approximierende Variante der Riemann-Summe genutzt, welche auf eine bestimmte Anzahl an Samples s reduziert wird:

$$\approx AO(P)_s = \frac{1}{s} \sum_{i=1}^{s} (1 - V(\omega_i, P)) \langle \omega_i n \rangle. \quad (2)$$

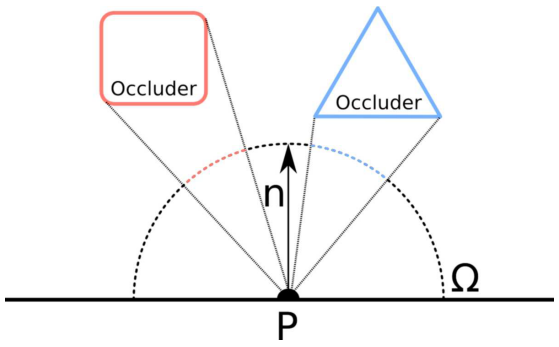


Abbildung 3: *Darstellung des durch Gleichung 1 beschriebenen Prinzips: Jede Geometrie der Szene (rot und blau) kann Teile, der einen Punkt P umschließenden Einheitshemisphäre, verdecken. Dieser Bruchteil wird mit der relativen Richtung zur Normalen n gewichtet und bildet somit den Occlusion Faktor für den Punkt P [1].*

Um artefaktarme Ergebnisse zu erzielen, ist auf eine angemessene Anzahl an Samples s sowie auf eine passende Wahl der Verteilung der Sample-Vektoren ω zu achten.

II.b LineAO

Wie bereits erwähnt, handelt es sich bei der LineAO-Methode um eine Screen Space Variante von Ambient Occlusion. Screen Space Lösungen dienen im Allgemeinen der Reduzierung von Komplexität, da es nicht möglich wäre für alle Oberflächenpunkte AO-Werte in Echtzeit zu berechnen. Daher wird das AO-Prinzip entsprechend adaptiert, so dass ein AO-Wert pro Pixel im Screen Space berechnet wird. Somit wird nicht mehr die Oberfläche einer Hemisphäre gesamplet, sondern die Fläche eines Kreises um den zu betrachtenden Pixel. Der Anschaulichkeit halber ist es nützlich, sich die von der Graphikkarte berechnete Tiefenkarte als Höhenfeld vorzustellen. Ein Punkt im Tal wird von den umliegenden Peaks „verdeckt", hingegen an Samplepunkten welche näher zur Kamera sind ist als der der zu berechnende Punkt, tritt keine Verdeckung auf, wodurch dieser Punkt heller wird.

Dies wird durch die Sichtbarkeitsfunktion V_l, wie sie bereits in ähnlicher Form als Teil des Ambient Occlusion vorgestellt wurde, berechnet:

$$V_l(r\omega, P) = \begin{cases} 0 & \text{if } d_l(P) - d_l(P+\omega) < 0 \\ 1 & \text{else.} \end{cases} \tag{3}$$

Die Funktion d_l berechnet hierbei den Tiefenwert des jeweiligen Punktes. Gemeinsam mit der Gewichtungsfunktion g_l funktioniert die Sichtbarkeitsfunktion als die Faktoren, welche bereits in Gleichung 1 vorgestellt wurden.

$$g_l(r\omega, P) = g_l^{depth}(\omega, P) g_l^{light}(\omega, P) \tag{4}$$

Ähnlich wie bei der AO-Variante wird für eine bestimmte Anzahl an Samples der Faktor aus Sichtbarkeitsfunktion und Gewichtungsfunktion bestimmt:

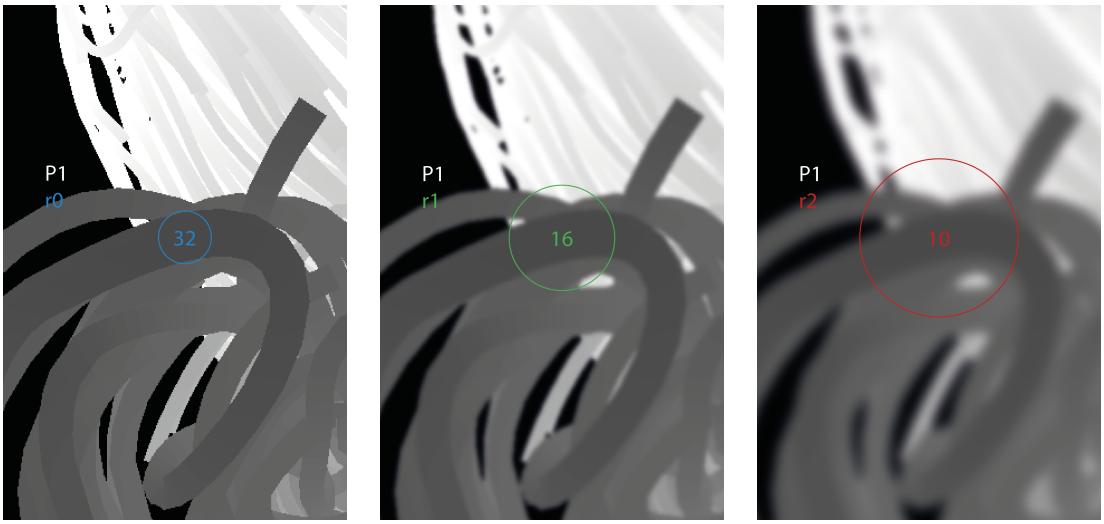
$$AO_{s,l}(P, r) = \frac{1}{s} \sum_{i=1}^{s} [\, (1 - V_l(r\omega_i, P))\, g_l(r\omega_i, P)\,] \tag{5}$$

Bevor auf die exakte Berechnung der Gewichtungsfunktion eingegangen wird, soll das Sampling erläutert werden, welches bei der LineAO-Methode angepasst wurde, um lokale und globale Features zu verstärken. Die bereits in Gleichung 5 zu sehenden Indizes s und l spielen für das Sampling eine maßgebliche Rolle. Das besondere am Sampling-Schema von LineAO ist, dass auf verschiedenen Radien l abgetastet wird. Der erste Radius ist so breit wie die Linienbreite, alle weiteren Radien sind Vielfache dieser. Weiterhin wird, um globalen Einfluss zu verstärken, für größer werdende Radien auf tiefpassgefilterten Texturen abgetastet. In Abbildung 4 ist besagter Sachverhalt exemplarisch für einen Punkt P für $r = 3$ dargestellt. In der Praxis bedeutet das, für $l = 0$ wird auf der untersten Ebene der Gaußpyramide gesamplet, für $l = 2$ beispielsweise entsprechend auf der zweiten. Weiterhin nimmt die Anzahl der Samples für größer werdende Radien ab.

Die mathematische Beschreibung des Samplings liefert die LineAO-Gleichung:

$$LineAO_{s_r, s_h, r_0}(P) = \sum_{j=0}^{s_r - 1} AO_{\frac{s_h}{j+1}, j}(P, r_0 + jz(P)) \tag{6}$$

Hier lässt sich sehen, wie die Anzahl der Samples durch den ersten Index $\frac{s_h}{j+1}$ variiert wird. Der Radius wird ebenso adaptiert, durch den zweiten Übergabewert der AO-Funktion $r_0 + jZ(P)$. Die sogenannte Zoom-Funktion $z(P)$ gibt an, wie groß der Durchmesser einer Einheitssphäre am Punkt P im Screenspace tatsächlich ist. Somit wird der Einfluss der Samplepunkte entsprechend der Lage in der Szene angepasst. Dies hat zur Folge, dass für das erste Level $l = 0$ der Sampleradius der Linienbreite entspricht.



(a) **(b)** **(c)**

Abbildung 4: *Darstellung der verschiedenen Sample-Radien und Einflussgrößen: Die in Summe 58 Samples verteilen sich auf drei verschiedene Sample-Radien. Mit zunehmendem Radius wird die Sampleanzahl geringer und das Gaußlevel steigt. Weiterhin ist es mittels der LineAO-Methode möglich, die Anzahl der Radien selbst zu wählen, um den globalen Einfluss der Szene auf die Beleuchtung zu regulieren.*

Somit wurde die allgemeine Struktur der LineAO-Funktion erläutert, welche, abgesehen vom Sampling-Schema, der allgemeinen AO-Funktion sehr ähnelt. Nun sollen noch die beiden essentiellen Funktionen g_l^{depth}[1] und g_l^{light} besprochen werden, welche miteinander multipliziert die Gewichtungsfunktion bilden. Die Funktion g_l^{depth} kann als eine Art Verfeinerung der Sichtbarkeitsfunktion verstanden werden. Es findet nicht mehr eine binäre Entscheidung statt, ob ein Sample in die Berechnung mit eingeht, sondern es werden kontinuierliche Werte berechnet.

$$g_l^{depth}(\omega, P) = \begin{cases} 0 & \text{if } \Delta d_l(\omega, P) > \delta(l) \\ 1 & \text{if } \Delta d_l(\omega, P) < \delta_0 \\ 1 - h\left(\frac{\Delta d_l(\omega,P) - \delta_0}{\delta(l) - \delta_0}\right) & \text{else} \end{cases} \tag{7}$$

In der Gleichung wird die Funktion Δd_l benutzt, welche die Tiefendifferenz zwischen Samplepunkt und dem Punkt P berechnet: $\Delta d_l(\omega, P) = d_l(P) - d_l(P + \omega) \in [0, 1]$. Eine weitere noch nicht eingeführte Funktion ist $\delta(l)$. Diese berechnet einen levelspezifischen Schwellwert, welcher bestimmt, welche Samples Einfluss auf das Resultat haben und welche nicht: $\delta(l) = \left(1 - \frac{l}{s_r}\right)^2 \in [0, 1]$. Das bedeutet, dass ein Samplepunkt auf dem ersten Level einen Tiefenunterschied von bis zu $\frac{4}{9}$ zum Punkt P haben kann und trotzdem Einfluss auf das Endergebnis hat. Auf dem zweiten

[1] Beim Nachimplementieren der LineAO-Methode ist darauf zu achten, dass in Gleichung 12 des Papers ein Tipp-Fehler enthalten ist. Die korrekte Form ist in diesem Projektbericht in Gleichung 7 zu sehen. Im Zähler steht $\Delta d_l(\omega, P)$ und nicht $d_l(P)$, womit es sich aufgrund des Tippfehlers leicht verwechseln lässt.

Level beträgt der Schwellwert dann nur noch $\frac{1}{9}$. Auf diese Weise wird verhindert, dass sich größere Strukturen zu sehr gegenseitig verdecken. Als letzte nicht eingeführte Funktion sei das Hermite Polynom genannt, welches genutzt wird um im Bereich zwischen den Schwellwerten zu interpolieren. Das heißt g_l^{depth} gibt 0 zurück, wenn der Tiefenunterschied zwischen Samplepunkt und dem Pixel P größer als der levelspezifische Schwellwert ist und somit kein Einfluss auf den Verdeckungs-Wert angenommen werden kann und 1 wenn der Tiefenunterschied weniger als $\delta_0 = 0.000001$ beträgt, was den größtmöglichen Einfluss darstellt. Dazwischen wird mittels des Hermite Polynoms interpoliert.

Abschließend soll nun auch die Beleuchtungsfunktion vorgestellt werden, welche zunächst den eigentlichen Beleuchtungswert auf einen Wertebereich auf $[0, 1]$ begrenzt: $g_l^{light}(\omega, P) = 1 - min(L_l(\omega, P), 1)$. Der hier somit eingegrenzte Wert stammt aus der folgenden Gleichung:

$$L_l(\omega, P) = \sum_{s \in Lights} BRDF(L_s, l_s, n_l(P), \omega). \tag{8}$$

Diese berechnet die Summe der Helligkeit über alle Lichter der Szene. Die Abkürzung BRDF (Bidirectional Reflectance Distribution Function) steht im Deutschen für „Bidirektionale Reflektanzverteilungsfunktion" und beschreibt das Reflexionsverhalten von Oberflächen. In diesem Fall wurde einfaches Blinn-Phong gewählt, aber auch andere Modelle, welche beispielsweise das Reflektanzverhalten matter Oberflächen darstellen, können hier gewählt werden.

III Methoden

Ausgehend von einer Beschreibung der Anforderungen wird im Folgenden vorgestellt, was auf welche Weise umgesetzt wurde, bevor es im nächsten Kapitel um die qualitative und quantitative Einschätzung dessen geht. Das grundlegende Ziel war es, einen 4D PC-MRI Blutfluss-Datensatz (siehe Kapitel I.b), welcher bereits vorverarbeitet mit sämtlichen Attributen, wie Zeitstempeln, Geschwindigkeiten und Tangenten, vorlag, mittels der von Eichelbaum et al. [1] vorgestellten LineAO-Methode zu animieren. Es wurden folgende Anforderungen an die zu entwickelnde Animation formuliert:

Performanz Ebenso, wie es die Anforderung an die LineAO-Methode war, auf der Hardware normaler Konsumenten lauffähig zu sein, soll dies auch für die zu entwickelnde, animierte Version gelten. Da es sich bei LineAO um eine Screen Space Ambient Occlusion Methode handelt, welche mittels Deferred Shading umgesetzt wird, spielt auch die Größe des Datensatzes keine Rolle, sondern ausschließlich die Anzahl der Pixel, welche von den Liniensegmenten bedeckt sind.

Ausfaden der Liniensegmente Um dem Erscheinungsbild einer Animation gerecht zu werden, ist es nötig, die harten Kanten zu Beginn und Ende eines Liniensegmentes kontinuierlich ein- und ausfaden zu lassen. Andernfalls würde der Betrachter die Bewegung als nicht flüssig, oder zumindest als unnatürlich wahrnehmen.

Order Independent Transparency Um ein Ausfaden der Liniensegmente zu realisieren, ist es nötig, OIT in die Renderpipeline zu integrieren, da LineAO ein Screen Space Ambient Occlusion Ansatz ist. Jegliche nicht sichtbaren Fragmente werden nicht gerendert und sind somit auch nicht für Alpha-Blending verfügbar. Hierbei wird zwar der AO-Effekt für verdeckte Fragmente vernachlässigt, jedoch kann so zu Beginn und Ende eines jeden Liniensegmentes ein kontinuierliches Ein- bzw. Ausfaden der Transparenz realisiert werden (mehr dazu in der Diskussion).

Hierzu wurde zunächst eine Methode zum Laden der Daten implementiert, welche die einzelnen Pathlines in einer entsprechenden Klasse speichert und für jedes Kantensegment die jeweilige Geschwindigkeit wie auch die für die zurückgelegte Strecke benötigte Zeit speichert. Die sich anschließende, tatsächliche Renderpipeline, welche bis auf wenige Ausnahmen vollständig auf der GPU stattfindet, wird im Folgenden besprochen (siehe Abbildung 6). Da verschiedene Implementierungen getestet und verglichen wurden, wird die umfassendste besprochen, in welcher alle anderen Implementierungen in verschiedenen Abwandlungen enthalten sind.

Zunächst sollen jedoch zwei Methoden erläutert werden, welche in der Pipeline genutzt wurden und deren Verständnis für das Verstehen der gesamten Pipeline nötig ist. Zum einen handelt es sich um *Deferred Shading*, zum anderen um *Order-independent Transparency* (OIT).

III.a Deferred Shading

Als Screen-Space Shading-Techniken werden Shading-Techniken bezeichnet, deren Berechnungen ausschließlich im Koordinatenraum des resultierenden 2D-Bildes während des Rendering einer 3D-Szene stattfindet. Deferred Shading (engl. verzögerte Schattierung) ist eine solche Technik und verdankt ihren Namen dem Fakt, dass im ersten Renderpass kein tatsächliches Shading stattfindet, sondern das Shading bis mindestens zum zweiten Renderpass „verzögert" wird. Auf diese Weise wird die Geometrieverarbeitung von der eigentlichen Lichtberechnung getrennt. Dadurch kann eine erhebliche Beschleunigung erreicht werden, da nicht mehr für jeden Vertex

der Szene, sondern nur noch für jeden Pixel im Screen-Space, die Beleuchtung berechnet werden muss. Eine ausführliche Beschreibung dieses Zusammenhangs am Beispiel des historischen Entwicklungsprozesses der CryEngine 2, bei welcher erstmalig Screen Space Ambient Occlusion realisiert wurde, kann hier gefunden werden [16].

Technisch wird Deferred Shading in zwei Renderpasses umgesetzt. Im ersten Renderpass werden alle für die Beleuchtungsrechnung benötigten geometrischen Informationen in 2D-Texturen gespeichert, deren Gesamtheit als G-Buffer (Geometric Buffer) bezeichnet wird. Diese Bezeichnung geht auf Saitoto et al. [17] zurück, welche den G-Buffer nutzten, um besondere Feature eines Datensatzes in ihm zu speichern und somit illustrative Rendertechniken zu realisieren. Im G-Buffer gespeicherte Informationen können Tiefenwerte, Positions-Vektoren, Normalen-Vektoren oder andere speziellere geometrie-spezifische Werte sein, wie es bei der LineAO-Methode der Fall ist. Im zweiten Renderpass findet nun das eigentliche Shading statt. Dies kann sowohl einfaches Phong-Shading sein, als auch komplexeres Ambient Occlusion-Shading, welches reale, ambiente Beleuchtungssituationen phänomenologisch approximiert, so wie es auch bei der LineAO-Methode der Fall ist.

Der große Vorteil dieser Technik besteht darin, dass nur noch pro Punkt im Screenspace die Beleuchtung berechnet werden muss und nicht mehr für jeden Punkt der Szene. Dies reduziert die Komplexität erheblich und ermöglicht es, rechenaufwändigeren Shading-Techniken echtzeitfähig zu werden.

Ein großer Nachteil, mit welchem sich auch in dieser Arbeit beschäftigt wurde, ist die Darstellung von Transparenzen. Da alle geometrischen Informationen ausschließlich als 2D-Texturen vorliegen, ist es nicht möglich verdeckte Objekte darzustellen. Bei einfachen Geometrien, wie beispielsweise semi-transparenten Fenstergläsern, ist es möglich dieses Problem durch einen weiteren Shader zu umgehen. Da die Flussdaten sich jedoch teilweise selbst verdecken ist dies im gegebenen Fall keine Option. Es wurde versucht mittels OIT Transparenzen darzustellen. Die Funktionsweise von OIT wird im nächsten Abschnitt erläutert.

III.b Order-independent Transparancy

Alpha-Blending ist eine klassiche Methode der Computergraphik. Sie wird genutzt, um semi-transparente Objekte darzustellen. Order-independent Transparency wird angewendet um eine korrekte Tiefenstaffelung aller Elemente zu realisieren, was für ein korrektes Alpha-Blending nötig ist. Dies kann jedoch problematisch sein, wenn es mehrere Objekte in der Szene gibt oder wenn sich Dreiecke gegenseitig durchkreuzen. Daher wird bei Order-independent Transparency pro Fragment eine Tiefenstaffelung durchgeführt. Die Idee hierzu stammt bereits aus dem Jahr 1984 von Carpenter et al. [18]. Diese benutzten pro Pixel je eine Linked List aller geschnittenen Fragmente um Aliasing Artefakte zu bekämpfen. Ihre Technik nannten sie A-Buffer. Diese Technik wurde von Yang et al. [19] aufgegriffen, welche eine Methode entwickelten, um 3D-Szenen mittels

Linked Lists auf der GPU repräsentieren zu können. Mittels dieser entwickelten Datenstruktur soll es möglich sein verschiedene komplexere Rendering-Techniken, wie beispielsweise OIT, echtzeitfähig realisieren zu können.

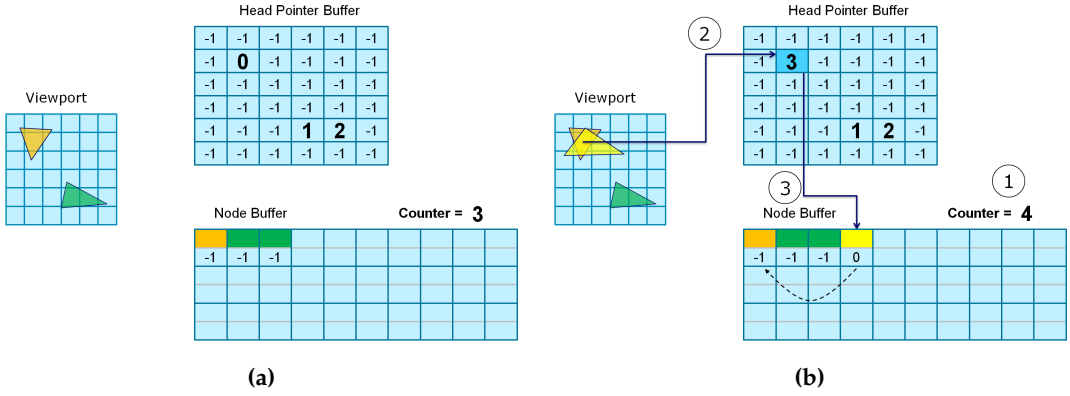


Abbildung 5: *Einfügen eines Knotens in die Linked List.* ***(a)*** *Initialer Zustand bevor ein neues Fragment hinzugefügt wurde.* ***(b)*** *Schritt 1: Finde den nächsten freien Platz im Node Buffer (Counter = 3) und erhöhe ihn. Schritt 2: Atomic Exchange zwischen Counter und Head Pointer Buffer (tausche 3 mit 0). Schritt 3: Füge den neuen Knoten in den Node Buffer und speichere den Pointer (0) an Position 3. [19]*

Abbildung 5 stellt das Einfügen eines Fragmentes in die Linked List Struktur dar. Hierbei werden die folgenden drei Schritte durchlaufen:

1. Finde die nächste freie Stelle und somit die neue Adresse (P') im Node Buffer. Diese entspricht dem Counter, welcher anschließend erhöht wird.

2. Tausche, ebenso in einer atomaren Operation, die neue Adresse (P') mit dem Pointer (P), welcher an der aktuell zu rendernden Pixelposition im Head Pointer Buffer steht. Auf diese Weise zeigt der neue Pointer im Head Pointer Buffer auf den Knoten im Node Buffer, welcher gerade eingefügt wird. Auf diese Weise wird die Linked List in umgekehrter Reihenfolge angelegt.

3. Nutze den alten Pointer P aus dem Head Pointer Buffer und speichere ihn als nächsten Pointer Wert im Node Buffer. In diesem Schritt werden auch alle weiteren Attribute des Knotens, welcher eingefügt wird, gespeichert. Dies sind im Falle von OIT Tiefenwert, Farbe als RGBA-Wert und besagter Pointer auf das letzte Element(P).

Im nächsten Renderpass wird pixelweise die Linked List ausgelesen. Hierzu wird die letzte gespeicherte Adresse (P) an der entsprechenden Stelle des Head Pointer Buffers abgerufen. Diese zeigt auf einen Knoten im Node Buffer. Ausgehend von diesem Knoten können mittels der pro Knoten gespeicherten Pointer alle Fragmente aus dem Node Buffer ausgelesen werden. Dies liefert eine Liste aller Fragmente, welche sich an eben dieser Pixelposition im resultierenden Bild befinden.

Anschließend muss diese Liste nach Tiefenwerten sortiert werden, um ein Alpha Blending auf dieser Liste zu ermöglichen. Spezifischere, die Implementierung betreffende Beschreibungen dieser Methode folgen im nächsten Absatz, der Beschreibung der Renderpipeline.

III.c Renderpipeline

Die Renderpipeline beginnt mit dem Leeren der Linked List, welche für OIT benutzt wurde. Hierfür werden alle Werte des Head Pointer Buffers auf einen nicht erreichbaren Wert gesetzt, welcher als quasi Null-Pointer dient. Im zweiten Renderpass wird die Datenstruktur, bestehend aus Counter, Head Pointer Buffer und Node Buffer, wieder wie im letzten Absatz beschrieben gefüllt. Um zu verhindern, dass in den Frame Buffer geschrieben wird, wird vor diesem Renderpass das Schreiben in alle Farbkomponenten mittels `glColorMask(disable)` gesperrt. In diesem Renderpass ist auch bereits der Geometry Shader integriert, welcher aus den einzelnen Liniensegmenten Vierecke generiert, welche sich immer orthogonal zum Viewing Ray ausrichten. Ebenso werden in diesem Schritt die Halos erzeugt, wenn dieser Parameter vom Nutzer ausgewählt wurde.

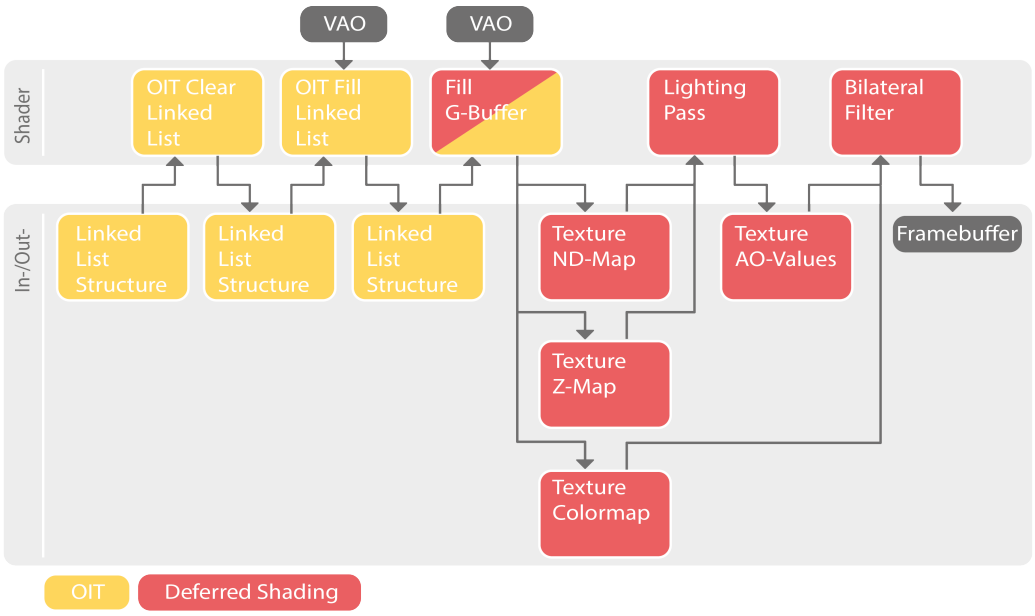


Abbildung 6: *Darstellung des Zusammenspiels von OIT und Deferred Shading in der Renderpipeline.*

Es folgt das Deferred Shading in der Pipeline. Im nächsten Renderpass werden Texturen aller, für die Berechnung des AO-Wertes benötigten Attribute erzeugt. Daher wird, um die Begriffe der LineAO-Methode zu benutzen, eine Zoommap-Texture, eine ND-Map-Texture sowie eine Color-Map-Textur (RGBA) erzeugt. Dies ist der aufwändigste Renderpass, da in ihm zusätzlich das Alpha-Blending stattfindet. Das heißt, die Linked List muss ausgelesen, alle enthaltenen Fragmente sortiert und anschließend zu einem Farbwert verrechnet werden. Weiterhin ist derselbe Geometry

Shader, welcher bereits beim Füllen der Linked List verwendet wurde, in diesem Renderpass enthalten.

Das Problem, welches hierbei beim Alpha-Blending auftritt, wird noch ausführlich im Teil IV besprochen. Soweit wurden alle nötigen Ressourcen vorbereitet, welche für das Berechnen des eigentlichen AO-Wertes nötig sind und der entsprechende Renderpass folgt.

In diesem werden die Werte aus den Texturen, entsprechend der in Abschnitt II erläuterten LineAO-Methode, genutzt, um pro Pixel einen AO-Wert zu berechnen. Das Endprodukt dieses Renderpasses ist dementsprechend wieder eine Textur der AO-Werte. Diese wird im nächsten und letzten Renderpass mittels eines Bilateralfilters geglättet und mit den Farbwerten der Colormap-Textur multipliziert. Alle weiteren Renderpasse, welche für das Rendern der Aorta-Rückseite und der Textelemente zuständig sind, werden nicht erläutert. Die gesamte Renderpipeline ist in Abbildung 6 zu sehen.

IV Ergebnisse und Diskussion

Den zu Beginn formulierten Anforderungen zufolge, wurden verschiedene Variationen getestet. Diese sollen im Folgenden sowohl qualitativ, als auch quantitativ analysiert werden. Eine qualitative Analyse wird für alle Visualisierungsparameter, wie Halo-Attribute, Liniendichte, Farbkodierungen etc. stattfinden.

Begonnen wird mit der simpelsten Implementierung, welche die Anforderung des Ausfadens der Liniensegmente nicht berücksichtigt, demzufolge eine direkte Anwendung der LineAO-Methode auf einen 4D PC-MRI Datensatz darstellt. Diese Variante ist somit besonders als Vergleichswert bezüglich der Performanz interessant.

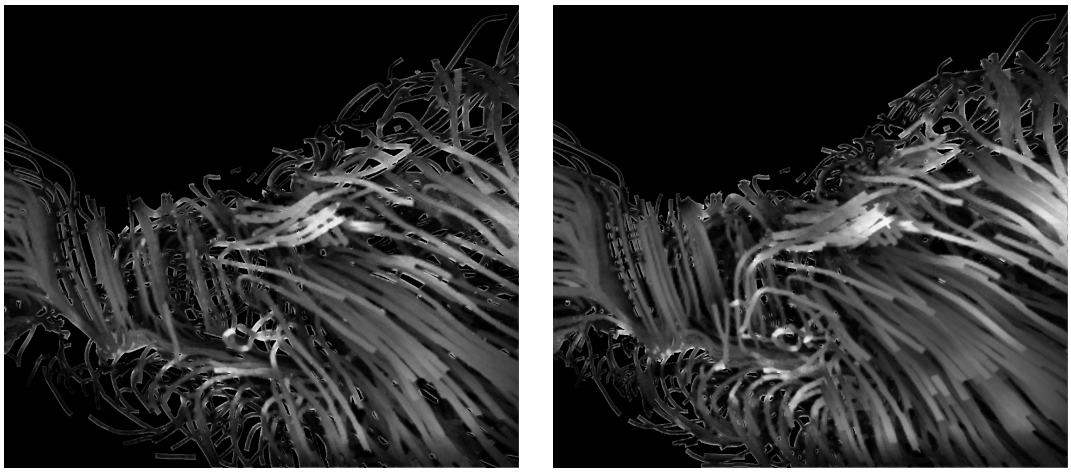
IV.a Variante 1 – Ausgangspunkt

Wie bereits errwähnt, wurde in dieser Variante kein Ausfaden der Linien realisiert und dient daher hauptsächlich dem Vergleich. Der rein optische Eindruck zeigt wenige Artefakte und auch die Performanz ist ausreichend für Interaktion und Explorieren des Datensatzes. Im Durchschnitt werden 10 fps erreicht, was vermuten lässt, dass ein besser ausgestatteter Desktoprechner diese Variante in Echtzeit rendern könnte (siehe hierzu Abschnitt Performanz).

Da zu diesem Zeitpunkt noch keine weiteren Abhängigkeiten von anderen Techniken und damit verbundenen Artefakten vorliegen, sollen nun allgemeine Visualisierungsparameter besprochen werden.

IV.b Visualisierungsparameter

Zunächst einmal wurde die **Liniendichte** variiert. Hierzu wurde derselbe Datensatz mit einer geringeren Anzahl an Linien gezeichnet. Um eine korrektere Untersuchung des Einflusses der Liniendichte durchzuführen, müssten neue Datensätze mit weniger Seeds und gröberem Raster integriert werden. Jedoch ist der Effekt, welchen weniger dichte Daten auf den AO-Effekt haben, deutlich sichtbar. Bei geringerer Liniendichte geht der Ambient Occlusion Effekt deutlich verloren, so wie es in Abbildung 7 zu sehen ist. Dies bringt uns zu allgemeinen Problemen, welche beim Anwenden von Ambient Occlusion Techniken auf animierte Blutflussdaten auftreten. Einerseits verändert sich die Liniendichte über die Zeit, was einen diskontinuierlichen Ambient Occlusion Effekt zur Folge hat, andererseits ist die Beleuchtung eines Pfadsegmentes während der Animation invariant und ist immer abhängig von seiner Umgebung. So kann es passieren, dass ein Segment sehr häufige Änderungen der Helligkeit erfährt, wenn sich die Zusammensetzung der Tiefenwerte in ihrer direkten Umgebung häufig ändert, was bei dichten Linienbündeln häufig vorkommt.



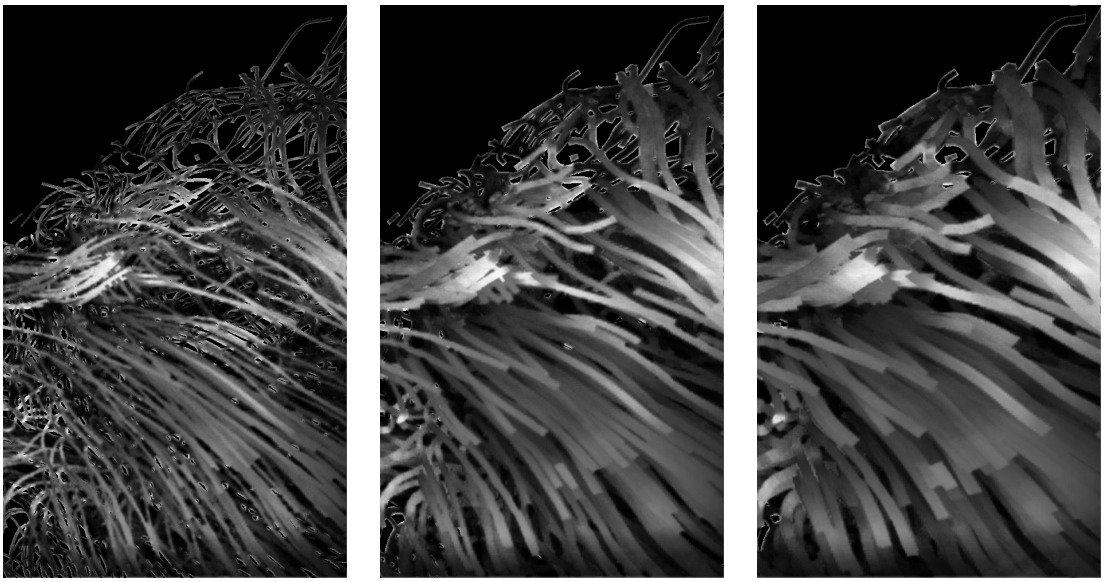
(a) **(b)**

Abbildung 7: *Vergleich verschiedener Liniendichten:* ***(a)*** *Niedrigere Liniendichte mit nur 1584 Linien, abgeschwächter AO-Effekt* ***(b)*** *Höhere Liniendichte mit 3169 Linien, deutlicherer AO-Effekt besonders in der Bildmitte [19].*

Entsprechend der Aussage des Autors „LineAO is not suited for coarse line data." wurde sich für den Datsansatz mit der höheren Liniendichte und 3169 Linien entschieden. In Abbildung 7 ist der Vergleich zu sehen.

Ein ähnlicher Kompromiss zwischen Verdeckung und Deutlichkeit des AO-Effektes ist bei der **Linienbreite** einzugehen. Ich habe mich, einem Kompromiss entsprechend für die mittlere Variante entschieden. Der Vergleich ist in Abbildung 8 zu sehen.

Ebenso ist in diesen Abbildungen ein Artefakt zu sehen, welches für Screen Space Ambient



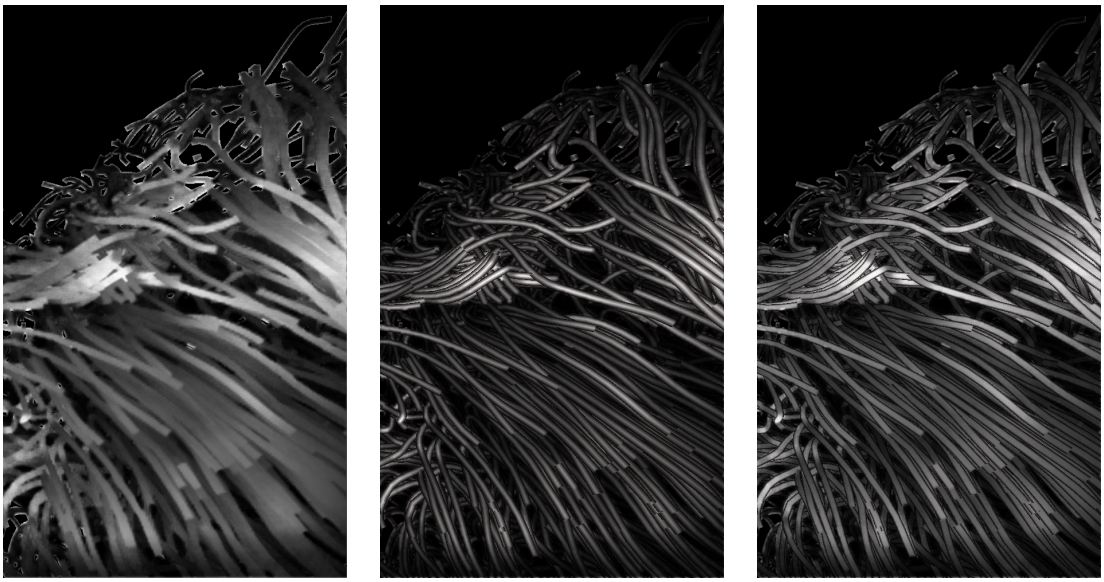
(a) **(b)** **(c)**

Abbildung 8: *Vergleich verschiedener Linienbreiten: Von **(a)** nach **(c)** zunehmende Linienbreite, sowohl als auch deutlicherer AO-Effekt.*

Occlusion typisch ist, welches Edge Bleeding genannt wird. Diese Artefakte können an den Rändern der Linien auftreten, wenn es in ihrer direkten Umgebung keine validen Samples gibt. Das heißt, wie im Abschnitt II erläutert, werden Samples, welche eine größere Tiefendifferenz zum momentan zu betrachtenden Pixel haben, als es der level-spezifische Schwellwert zulässt, nicht mit berücksichtigt. Somit kommt es zu einer niedrigen Anzahl an Samples, was das Edge Bleeding begünstigt.

Um diesem Problem auf einfache Weise Herr zu werden, wurde der Einfluss von **Halos** untersucht. Mittels der Halos lässt sich nicht nur das Edge Bleeding wirksam bekämpfen, sondern das gerenderte Bild erhält mehr Struktur und es entsteht ein Eindruck, die Integrallinien wären Röhren und nicht bloß bandartige Strukturen. Auch hier wurden verschiedene Breiten des Halos getestet, um diesen Effekt angemessen zu realisieren. Dabei ist aufgefallen, dass breitere Halos dazu neigen, farblich zu überfüllen, wodurch ihr Effekt etwas verloren geht. Dies lässt sich wie folgt erklären: Die Halos sind im Allgemeinen Farbgradienten, welche die Pathlines überlagern. Werden sie gestreckt, so ist der Gradient nicht mehr so deutlich durch sein Anfang und sein Ende zu erkennen, sondern es wirkt im Vergleich eher, als wäre die Pathline mit einem grauen Schein überlagert wurden. Die Breite der Halos wurde, wie in Abbildung 9 dargestellt, von 10% der Linienbreite bis zu 50% der Linienbreite variiert. Sowohl die Verdeckung des Edge Bleedings als auch der räumliche Effekt wurde bei 20% am passendsten empfunden (siehe Abbildung 9).

Als letzter Visualisierungsparameter wird die Farbkodierung diskutiert. Um verdeckungsfrei



(a) **(b)** **(c)**

Abbildung 9: *Vergleich verschiedener Breiten des Halos:* ***(a)*** *keine Halos* ***(b)*** *Halos nehmen 10% der Linienbreite ein,* ***(c)*** *Halos nehmen 50% der Linienbreite ein, was den Effekt abschwächt.*

zusätzliche Informationen darzustellen, eignet sich eine Farbkodierung in sehr vielen Fällen. Im gegebenen Fall stellen grün gefärbte Integrallinien relativ langsamen Blutfluss dar, rot gefärbte repräsentieren schnelleren Blutfluss, welcher sichtbar häufig an Engstellen auftritt. Da es keine Artefakte oder negativen Nebeneffekte gibt, ist die farbkodierte Version ohne Vergleich in Abbildung 10 dargestellt.

Nachdem nun alle relevanten Parameter an der ersten, direkten Umsetzung der LineAO-Methode getestet und evaluiert wurden, geht es im Folgenden um die Umsetzung der Anforderung eines Ausfadens der Liniensegmente. Dies soll zunächst mittels OIT realisiert werden.

IV.c Variante 2 – OIT

Da es sich bei der LineAO-Methode um einen Screen Space Ansatz handelt, liegen nach dem Füllen des G-Buffers ausschließlich 2D-Texturen aller Features, welche zur Berechnung der Farbe nötig sind, vor. Das hat zur Folge, dass bei einer naiven Implementierung des Ausfadens der Liniensegmente der Hintergrund durchscheinen würde. Um dem zu begegnen, wurde das in Abschnitt III besprochene OIT genutzt.

Mittels OIT wird eine 2D-Textur erstellt, welche hinter den ausfadenden Liniensegmenten die darunterliegenden Strukturen darstellt. Für diese, in der Renderpipeline als Colormap bezeichnete Textur, werden alle Punkte zunächst als gleich hell angenommen, was sich wie später zu sehen mit dem Prinzip des Ambient Occlusion nicht vereinbaren lässt. Jedoch existieren nicht für alle

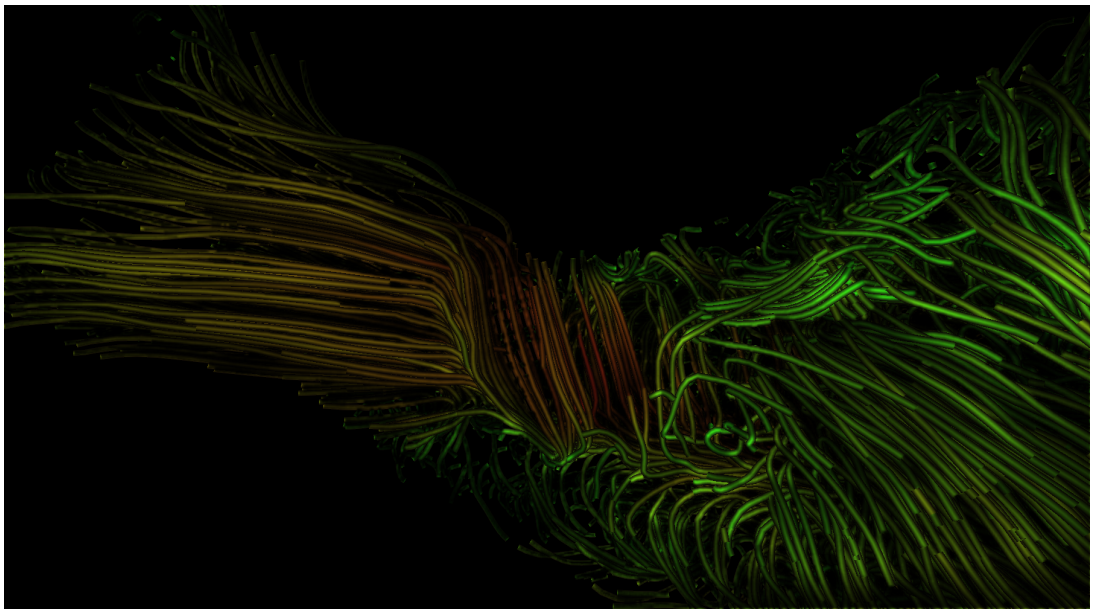


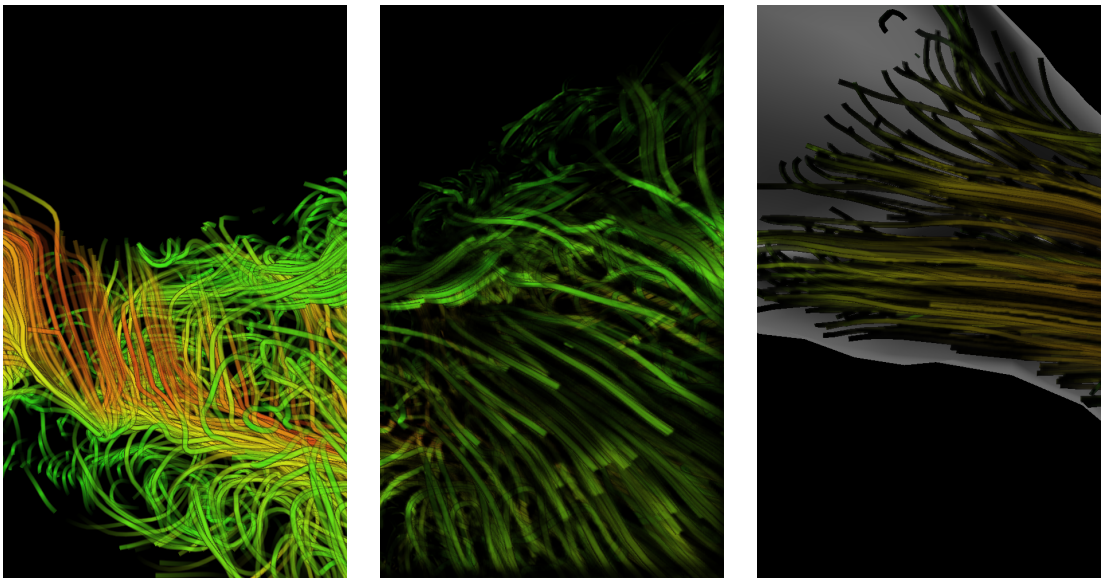
Abbildung 10: *Die Integrallinien werden entsprechend der Geschwindigkeit farbkodiert. Grün ist als langsam kodiert, rot als schnell.*

Punkte Ambient Occlusion Werte, sondern ausschließlich für die aus Kamerasicht nächsten, da es sich um eine Screen Space Methode handelt.

Im letzten Renderpass, dem Bilateralen Filter, werden die Werte der Colormap mit den Werten der geglätteten AO-Werte multipliziert. Hierbei tritt der eigentliche Fehler auf. Während noch beim Berechnen der AO-Werte von vollständigen Liniensegmenten ausgegangen wurde, kann es nun passieren, dass ausgefadete Liniensegmente trotzdem mit hohen AO-Werten multipliziert werden, was zu Artefakten, wie in Abbildung 11b führt. Weiterhin ist ein Anzeigen der Aorta-Wand nicht möglich, da auch hier die durchscheinenden Farbwerte der Aorta-Wand mit meist zu niedrigen AO-Werten multipliziert werden, was in Abbildung 11c zu sehen ist.

Die Farbwerte der Colormap, welche mittels OIT berechnet wurden, sind in Abbildung 11a zu sehen, wobei man hier gut sieht, dass die Farbwerte der ausfadenden Liniensegmente korrekt berechnet werden. All jene Segmente, welche nicht ausfaden, sind Anfang oder Ende einer gesamten Integrallinie. Das heißt, dass zum dargestellten Zeitpunkt entweder die Integration dieser Linie begonnen hat oder terminiert ist. Daher ist in diesen Fällen kein Ausfaden möglich.

Im Allgemeinen lässt sich feststellen, dass diese Variante sehr viele Artefakte mit sich bringt, welche zu einem starken Rauschen führen. Die Tiefenwahrnehmung, welche ursprünglich durch die LineAO-Methode verbessert werden sollte, leidet sehr unter diesem Ansatz. Weiterhin werden durch OIT sehr viele zusätzliche Rechenschritte und zusätzliche Renderpasse nötig, so dass die Echtzeitfähigkeit darunter deutlich leidet und im Mittel 6 fps erreicht werden.



(a) **(b)** **(c)**

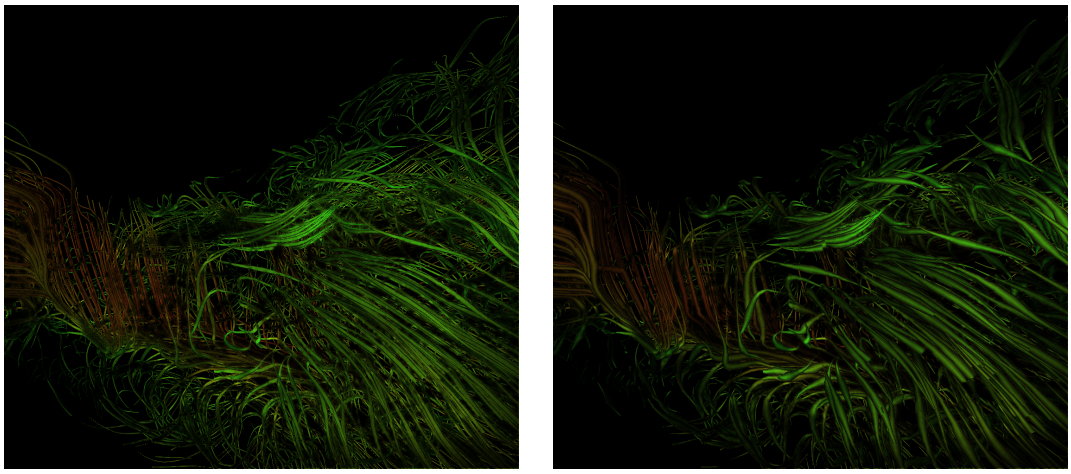
Abbildung 11: ***(a)*** *mittels OIT berechnete Colormap* ***(b)*** *durch OIT hervorgerufene Artefakte* ***(c)*** *unnatürliches Abdunkeln der Linienenden und der Aortawand.*

Da diese Variante den Anforderungen nicht gerecht werden konnte, wurde eine weitere Variante implementiert, welche im folgenden Abschnitt besprochen wird.

IV.d Variante 3 – Geometrisches Ausfaden

Nachdem die beiden bereits vorgestellten Varianten nicht allen Anforderungen gerecht werden konnten, wurde noch eine dritte Variante entwickelt, bei welcher ein alternatives Ausfaden der Linien realisiert wurde. Anstatt die Opazität zu Beginn und Ende der Linie zu verringern, wurde bei dieser Variante die Linienbreite verringert. Somit wird das Problem der fehlenden Tiefeninformationen umgangen. Jedoch entsteht dadurch ein weiteres Problem, welches sich bereits bei der OIT-Variante angedeutet hat, aufgrund der vielen Artefakte jedoch nicht als solches erkannt wurde. Ohne komplizierte Vorverarbeitungsschritte ist es nicht möglich alle Liniensegmente gleich lang zu zeichnen, sondern es werden alle Liniensegmente, deren Zeitstempel in einem zu definierenden Intervall liegen, gezeichnet. Somit ist die Länge des Ausfadens für jedes Liniensegment unterschiedlich. Besonders bei sehr langen Linienstücken führt das dazu, dass sich die Liniendicke verringert. In diesem Fall sind die zu Beginn evaluierten Visualisierungsparameter nicht anwendbar und wurden neu bestimmt. Der Vergleich der beiden Parametrisierungen ist in Abbildung 12b zu sehen.

In Abbildung 12 ist die Variante des geometrischen Ausfadens vor der Anpassung zu sehen. Es ist deutlich zu sehen, dass die sehr dünnen Strukturen den AO-Effekt mindern. Andererseits



(a) **(b)**

Abbildung 12: ***(a)*** *Variante 3, geometrisches Ausfaden der Linien mit zuvor ermittelten Parametern, schwacher AO-Effekt* ***(b)*** *Angepasste Liniendicke (*1.0 s*tatt* 0.5 *) und angepasstes Intervall (*0.6 s *statt* 1.0 s*), besserer AO-Effekt, dafür jedoch sehr unterschiedliche Liniendicken.*

ist im Vergleichsbild, nach Vergrößerung der Liniendicke und Verkleinerung des dargestellten Intervalls, zu sehen, dass die sehr verschiedenen Liniendicken sehr inhomogen wirken und direkt angenommen werden könnte, dass auch die Dicke der Segmente eine bestimmte Größe des Datensatzes kodieren soll. Abgesehen von dieser Mehrdeutigkeit wird diese Variante jedoch den Anforderungen noch am ehesten gerecht. Da bei dieser Variante auch kein weiterer Rechenaufwand entsteht, kann mit ca. 10 fps gerendert werden.

IV.e Performanz

Als letzter Punkt sollen die vorgestellten Varianten bezüglich ihrer Performanz verglichen werden. Hierbei ist einerseits die Framerate zur Einschätzung der Echtzeitfähigkeit relevant. Andererseits gibt die GPU time der einzelnen Shader Aufschluss darüber, wo besonders viel Zeit verloren geht. Da sich Variante 1 und 3 ausschließlich bezüglich der Parameter unterscheiden, werden sie in der Tabelle gemeinsam dargestellt. Bei der Variante 2 ist deutlich zu sehen, welchen Einfluss besonders das Auslesen der Linked List Strukturen hat. Im Renderpass, in welchem der G-Buffer gefüllt wird, findet auch gleichzeitig das Auslesen der Linked List Struktur statt. Im direkten Vergleich ist zu sehen, dass dafür in der OIT-Variante das ca. Fünffache an Zeit benötigt wird, da für eine korrekte Tiefenstaffelung pro Pixel alle Einträge der Linked List sortiert werden müssen. Im Allgemeinen lässt sich daher sagen, dass OIT für dieses Problem ungeeignet ist.

Variante	1 - Ausgangspunkt	3 - Geom. Ausfaden	2 - OIT
Framerate	≈ 10 fps	≈ 10 fps	≈ 6 fps
GPU time	1. Lighting Pass (74 559 µs) 2. Fill G-Buffer (46 049 µs) 3. Bilateral Filter (4738 µs)		1. Fill G-Buffer (205 448 µs) 2. Lighting Pass (25 545 µs) 3. Fill Linked List (23 591 µs) 4. Bilateral Filter (4670 µs) 5. Clear Linked List (1244 µs)

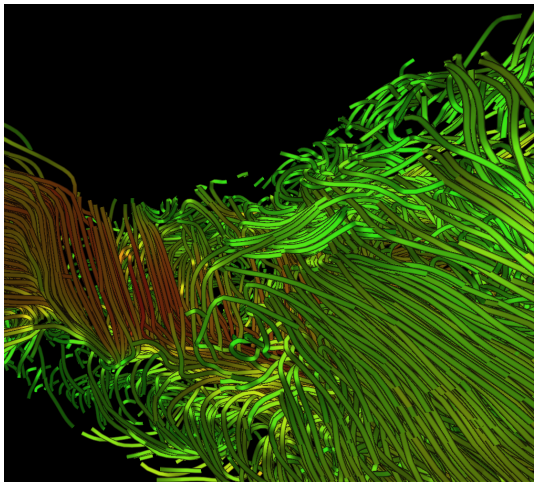
Tabelle 3: *Übersicht zu Framerates und GPU time der verschiedenen Varianten für einen Datensatz mit* 3169 *Linien. Alle aufgeführten Schritte werden pro Renderdurchgang durchlaufen.*

V Zusammenfassung und Ausblick

Es wurden verschiedene Möglichkeiten getestet, den Anforderungen gerecht zu werden, was jedoch bei allen Methoden auch Nachteile mit sich brachte. Es folgt eine kurze, auswertende Zusammenfassung der Vor- und Nachteile aller Methoden. Zusätzlich soll noch kurz die lokale Beleuchtungsmethode der Illuminated Stream Lines (ISL), welche von Zoeckler et al. [8] entwickelt wurde, als Vergleich gezeigt werden. In Abbildung 13 sind alle Varianten zusammen mit ihren Vor- und Nachteilen übersichtlich aufgeführt. Diese wurden bereits ausführlich diskutiert und es wurde deutlich, dass die zunächst fokussierte OIT-Variante zu wenig Mehrwert im Vergleich zu den verbundenen Einbußen mit sich bringt. Diesbezüglich könnte das geometrische Ausfaden einen Kompromiss darstellen.

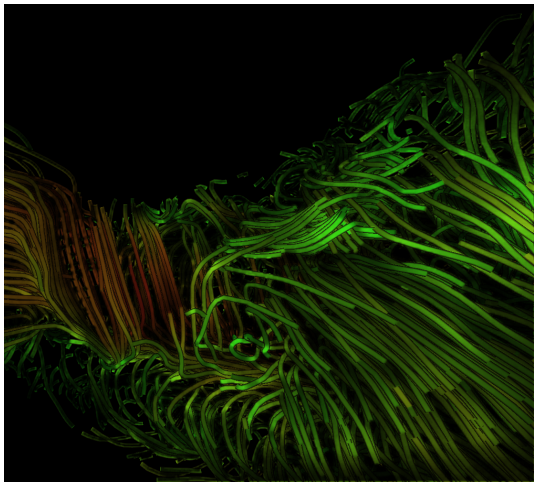
Im Allgemeinen konnte jedoch gezeigt werden, dass die LineAO-Methode auf Bluflussdaten angewandt werden kann und diese auch in einer Animation sowohl globale, als auch lokale Features angemessen beleuchten kann. Weiterhin konnte gezeigt werden, dass die Methode echtzeitfähig implementiert werden kann. Die aufgetretenen Probleme der einzelnen Varianten wurden in den Abschnitten ausreichend diskutiert, daher soll nun der Fokus auf den Ausblick gelegt werden.

Für Weiterentwicklungen der LineAO-Methode ist es in jedem Fall nötig, dass ein realistischer Kompromiss zwischen zusätzlichen Artefakten und Echtzeitfähigkeit gewählt wird. Hierfür wäre es unter anderem möglich Vorverarbeitungsschritte bereits auf der CPU zu realisieren. Ein denkbarer Anwendungsfall wäre es, die Liniendichte in der Umgebung eines jeden Punktes vorher zu berechnen, möglicherweise auch für eine diskrete Anzahl an Zeitpunkten während der



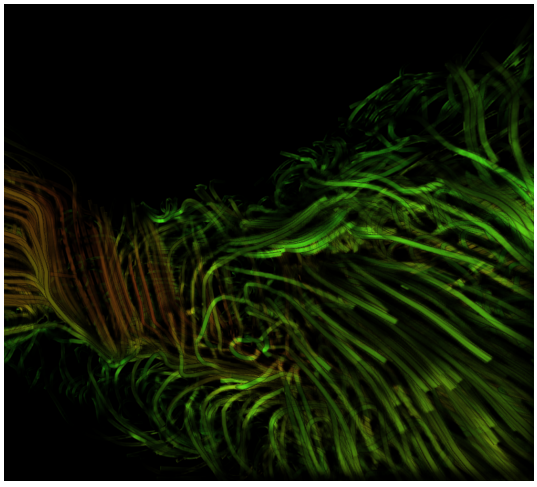
Illuminated Stream Lines

+ Einfach zu implementieren
+ Echtzeitfähig
− Keine globalen Features



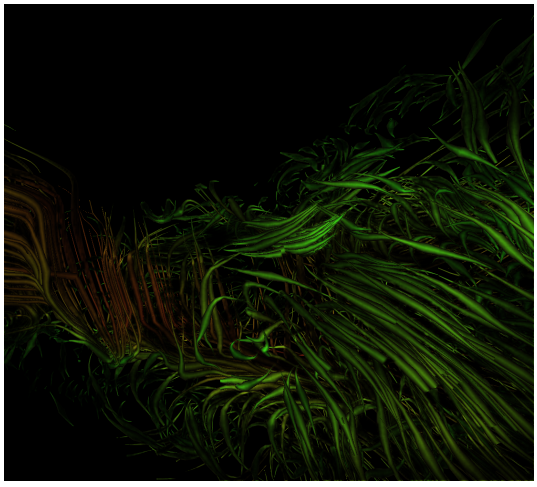
Variante 1 - Kein Ausfaden

+ Globale und lokale Features
+ Wenige Artefakte
− Eingeschränkt echtzeitfähig



Variante 2 - OIT

+ Ausfaden entsprechend Anforderungen
− Starke Artefakte
− Nicht echtzeifähig



Variante 3 - Geometrisches Ausfaden

+ Ausfaden ohne Artefakte
+ Keine Performanzeinbuße zu Variante 1
− Variierende Liniendicke

Abbildung 13: *Übersicht aller entwickelten Varianten und ihrer Vor- und Nachteile.*

gesamten Animation. Mit diesem Wissen zur Renderzeit könnte zum einen adaptiv gesamplet werden, um allgemeine Artefakte der LineAO-Methode, wie beispielsweise das Edge Bleeding, zu verhindern, andererseits ließe sich approximieren, wieviele Punkte tatsächlich für das Darstellen realistischer Transparenzen nötig sind. Somit ließe sich möglicherweise eine auf Line-Rendering zugeschnittene Methodenmischung aus echtem Ambient Occlusion und Screen Space Ambient Occlusion realisieren.

Weiterhin sind Kombinationen mit anderen Methoden denkbar, welche vielleicht nicht unbedingt das Darstellen von Transparenzen während der Animation verbessern, aber im Allgemeinen das Explorieren der Daten erleichtern. In dieser Hinsicht sind, da durch die LineAO-Methode im Allgemeinen sehr viel verdeckt und auch abgedunkelt wird, vor allem Verfahren wie Halo-basierte Techniken oder Feature-adaptives Anpassen von Transparenzen denkbar.

VI Literaturverzeichnis

[1] Sebastian Eichelbaum, Mario Hlawitschka und Gerik Scheuermann. „LineAO: Improved Three-Dimensional Line Rendering“. In: *IEEE Transactions on Visualization and Computer Graphics* 19.3 (2013), S. 433–445.

[2] Philip J Kilner u. a. „Helical and retrograde secondary flow patterns in the aortic arch studied by three-directional magnetic resonance velocity mapping.“ In: *Circulation* 88.5 (1993), S. 2235–2247.

[3] Xiao Liu u. a. „A numerical study on the flow of blood and the transport of LDL in the human aorta: the physiological significance of the helical flow in the aortic arch“. In: *American Journal of Physiology-Heart and Circulatory Physiology* 297.1 (2009), H163–H170.

[4] Umberto Morbiducci u. a. „Mechanistic insight into the physiological relevance of helical blood flow in the human aorta: an in vivo study“. In: *Biomechanics and modeling in mechanobiology* 10.3 (2011), S. 339–355.

[5] Benjamin Köhler u. a. „Semi-automatic vortex extraction in 4D PC-MRI cardiac blood flow data using line predicates“. In: *IEEE Transactions on Visualization and Computer Graphics* 19.12 (2013), S. 2773–2782.

[6] Mathias Neugebauer u. a. „Anatomy-Guided Multi-Level Exploration of Blood Flow in Cerebral Aneurysms“. In: *Computer Graphics Forum*. Bd. 30. 3. Wiley Online Library. 2011, S. 1041–1050.

[7] Maarten H Everts u. a. „Interactive illustrative line styles and line style transfer functions for flow visualization“. In: *arXiv preprint arXiv:1503.05787* (2015).

[8] Malte Zockler, Detlev Stalling und H-C Hege. „Interactive visualization of 3D-vector fields using illuminated stream lines“. In: *Visualization'96. Proceedings.* IEEE. 1996, S. 107–113.

[9] Leonard Wanger. „The effect of shadow quality on the perception of spatial relationships in computer generated imagery“. In: *Proceedings of the 1992 symposium on Interactive 3D graphics*. ACM. 1992, S. 39–42.

[10] Leonard R Wanger, James A Ferwerda und Donald P Greenberg. „Perceiving spatial relationships in computer-generated images“. In: *IEEE Computer Graphics and Applications* 12.3 (1992), S. 44–58.

[11] Silvia Born u. a. „Illustrative visualization of cardiac and aortic blood flow from 4D MRI data“. In: *Visualization Symposium (PacificVis), 2013 IEEE Pacific*. IEEE. 2013, S. 129–136.

[12] Maarten H Everts u. a. „Depth-dependent halos: Illustrative rendering of dense line data“. In: *IEEE Transactions on Visualization and Computer Graphics* 15.6 (2009).

[13] Roy Van Pelt u. a. „Interactive virtual probing of 4D MRI blood-flow“. In: *IEEE Transactions on Visualization and Computer Graphics* 17.12 (2011), S. 2153–2162.

[14] Benjamin Köhler u. a. „Motion-aware stroke volume quantification in 4D PC-MRI data of the human aorta“. In: *International journal of computer assisted radiology and surgery* 11.2 (2016), S. 169–179.

[15] Anna Vilanova u. a. „Visual exploration of simulated and measured blood flow“. In: *Scientific Visualization*. Springer, 2014, S. 305–324.

[16] Martin Mittring. „Finding next gen: Cryengine 2“. In: *ACM SIGGRAPH 2007 courses*. ACM. 2007, S. 97–121.

[17] Takafumi Saito und Tokiichiro Takahashi. „Comprehensible rendering of 3-D shapes“. In: *ACM SIGGRAPH Computer Graphics*. Bd. 24. 4. ACM. 1990, S. 197–206.

[18] Loren Carpenter. „The A-buffer, an antialiased hidden surface method“. In: *ACM Siggraph Computer Graphics* 18.3 (1984), S. 103–108.

[19] Jason C Yang u. a. „Real-Time Concurrent Linked List Construction on the GPU“. In: *Computer Graphics Forum*. Bd. 29. 4. Wiley Online Library. 2010, S. 1297–1304.

VII Abbildungsverzeichnis