Institut für Visualisierung und Interaktive Systeme

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

Beschleunigung des Renderings Paralleler Koordinaten mittels Auflösungsskalierung und Temporaler Rekonstruktion

Thomas Marmann

Studiengang:	Informatik
Prüfer/in:	Prof. Dr. Thomas Ertl
Betreuer/in:	Michael Becher, Dr. Guido Reina, Moritz Heinemann
Beginn am:	20. Juni 2020
Beendet am:	20. November 2020

Kurzfassung

Eine wichtige Visualisierung von hochdimensionalen Datensätzen sind Parallelen Koordinaten, bei denen die Daten als Linienzüge dargestellt werden, die parallele Achsen entsprechend ihrer Werte schneiden. Für große Datensätze kommt es hierbei jedoch zu sehr dichten Linienhaufen, welche durch Effekte wie Overdraw die Programmausführung verlangsamen können. Ein solches Problem tritt in der Visualisierungsumgebung MegaMol der Universität Stuttgart bei großen Datensätzen und Auflösungen auf. Zur Verbesserung dieser Echzeitinteraktion entsteht also Bedarf, den Rendervorgang, welcher auch im Wesentlichen die Reaktionszeit des Programms bestimmt, zu beschleunigen. In dieser Arbeit werden Verfahren vorgeschlagen, welche durch Nutzung von Auflösungsskalierung beim Rendern, Amortisieren der Renderkosten über mehrere Schritte und anschließender temporaler Rekonstruktion die Programmausführung beschleunigen. Es wird die Qualität der Anzeige unter Nutzung der Verfahren analysiert und die erzielte Beschleunigung in Hinblick auf Einflussfaktoren untersucht. Bildqualität wird anhand von Bildschirmschnappschüssen verglichen und bewertet. Ghosting-Artefakte, welche bei Bewegung der Szene bei amortisiertem Rendern entstehen können, werden durch Reprojektion der Abtastungen minimiert. Verbleibende Fehler werden aufgezeigt und deren Auswirkung auf den Bildeindruck untersucht. Zur Bewertung der Performanz der Verfahren wurden Messungen der Renderdauer unternommen und ausgewertet. Es wird gezeigt, dass die vorgeschlagenen Verfahren in vielen kritischen Situationen eine deutliche Beschleunigung des Rendevorgangs erreichen und ebenso die Grenzen der Verfahren aufgezeigt.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Algorithmen

1 Einleitung

Simulationen von komplexen Systemen spielen sowohl in der Wissenschaft als auch in Bereichen wie Wirtschaft oder dem Ingenieurwesen eine immer größere Rolle. Oftmals kommt es vor, dass so Datensätzen mit vielen Dimensionen und tausenden von Datenpunkten entstehen. Visualisierungen von solchen Datensätzen stellen besondere Herausforderungen dar, da die menschliche Wahrnehmung auf die drei räumlichen Dimensionen beschränkt ist. Dies erzeugt Bedarf für Werkzeuge, die die Interpretation und Analyse von solchen Datensätzen ermöglichen und vereinfachen. Parallele Koordinaten sind genau so ein Werkzeug. Hierbei werden die Datenpunkte als Linienzüge dargestellt, die parallele Achsen an bestimmten Werten schneiden. Es entstehen bei bestimmten Eigenschaften der Daten erkennbare Muster, welche leichter verstanden und interpretiert werden können. In der Praxis entstehen bei solchen Visualisierungen jedoch oft sehr dichte Haufen von Linien, die teilweise nur schwer voneinander unterschieden werden können. Solche Haufen verlangsamen zudem, aufgrund von Effekten wie Overdraw, den Ablauf von Visualisierungsprogrammen und erschweren oder verhinderen die Echtzeitinteraktion mit den Daten. Um diese Echtzeitinteraktion zu wahren, entsteht also Bedarf nach Verfahren, die den Rendervorgang beschleunigen, ohne die Details der Darstellung zu verlieren.
In dieser Arbeit werden Verfahren vorgeschlagen, die sich diesem Problem stellen und eine Beschleunigung der Rendervorgangs erwirken sollen. Zu diesem Zweck wird von Skalierung der Auflösung im Renderschritt, Amortisierung der Renderkosten über mehrere zeitlich versetzte Schritte und anschließender temporaler Rekonstruktion des Anzeigebildes gebrauch gemacht. Die so entstehende Bildqualität wird für jedes Verfahren untersucht mit der Ausgangssituation verglichen. Insbesondere wird auch bei Bewegung entstehende Ghosting-Artefakte eingegangen und ein Ansatz geschildert, diese zu minimieren. Die verbleibenden fehlerhaften Pixel und deren Auswirkungen auf die Bildqualität werden dannach untersucht und bewertet.
Zudem wird auf Einflussfaktoren eingegangen, die die Effektivität der Verfahren bestimmen. Diese Einflussfaktoren werden anhand von Zeitmessungen untersucht. In diesem Kontext werden die Grenzen der Verfahren abgeschätzt und anhand von Anwendungsszenarien erörtert. Die für diese Arbeit gewählte Visualisierungssoftware ist MegaMol, welche von der Universität Stuttgart entwickelt wird. Wie zuvor beschrieben, leidet auch MegaMol unter langsamen Reaktionszeiten bei großen Datensätzen und hohen Auflösungen. Ziel dieser Arbeit ist es diese Reaktionszeiten zu verbessern und eine Echtzeitinteraktion mit größeren Datensätzen zu erleichtern. Das Modul zur Visualisierung von parallelen Koordinaten wird erweitert und es wird gezeigt, dass für praktische Anwendungsfälle eine Beschleunigung des Renderns und eine Verbesserung der Nutzererfahrung führt.
Diese Arbeit behandelt zunächst in Kapitel 2 Grundlagen im Bereich vom Rendern von Linien und der Amortisierung des Rendervorgangs mit temporaler Rekonstruktion, die für das Verständis wichtig sind. Danach werden verwandete Arbeiten in Kapitel 3 beschrieben und diese Arbeit in den Forschungsstand eingeordnet. Zudem wird die verwendete Implementiertung in Kapitel 4 vorgestellt und Auswirkungen auf Bildqualität und Performanz erläutert. Anhand von Bildschirmschnappschüssen wird in Abschnitten 5.1 und 5.2 des Kapitels 5 die Darstellungsqualität bei Stillstand und

bei Bewegung der Szene analysiert. Um die Effektivität der Verfahren nachzuvollziehen, wird in Abschnitt 5.3 zunächst das Messverfahren erklärt, um dann in Abschnitten 5.4 bis 5.11 die Messdaten auszuwerten und Anwendungsfälle zu untersuchen. Zuletzt wird in 6 eine Schlussfolgerung gezogen und ein Ausblick auf weitere Untersuchung gegeben, die ausgehend von dieser Arbeit unternommen werden können.

2 Grundlagen

2.1 Parallele Koordinaten

Parallele Koodrinaten sind eine Möglichkeit zur Darstellung von hochdimensionalen Daten, welche bereits 1885 von d' Ocange vorgeschlagen wurden[Oca85]. Hierbei werden die Dimensionen der Daten als parallele Achsen dargestellt, die jeweils den Wertebereich der repräsentierten Dimension abbilden. Bei dieser Darstellung können Abstand und Ausrichtung der Achsen beliebig gewählt werden, jedoch werden Achsen typsicherweise abstandsgleich und vertikal ausgerichtet. Die Punkte des Datensatzes werden dann als Polylinien dargestellt, die die jeweilge Achse an dem Wert der entsprechenden Dimension schneiden.
Aus den entstehenden Mustern dieser Visualisierung können Schlüsse auf die Verteilung und Abhängigkeiten der verschiedenen Dimensionen gezogen werden. Beispiele für solche Muster und die zugrunde liegenden Zusammenhänge werden für den zweidimensionalen Fall in Abb. 2.1 abgebildet. Um solche Muster leichter zu erkennen, bieten die meisten Programme, die solche parallelen Koordinaten darstellen können, weitere Werkzeuge. Einfärbung von parallelen Koordinaten anhand von der Dichte der Linien oder anhand von Werten einer Dimension sind hier wichtige Beispiele. Hiermit können auch in dichten Linienhaufen oft noch wichtige Informationen ausgemacht werden.
Zudem ist auch Filtern und Selektieren von Linien wichtig. Oft liegen mehrere solcher Muster übereinander und können zunächst nicht voneinander unterschieden werden. Durch gewählte Auswahl von Linien können diese Muster dann oft erkennbar gemacht werden. Auch Verläufe von wenigen oder einzelnen Linien ist oft interessant, um Beispielsweise Haufenbildung zu erkennen.

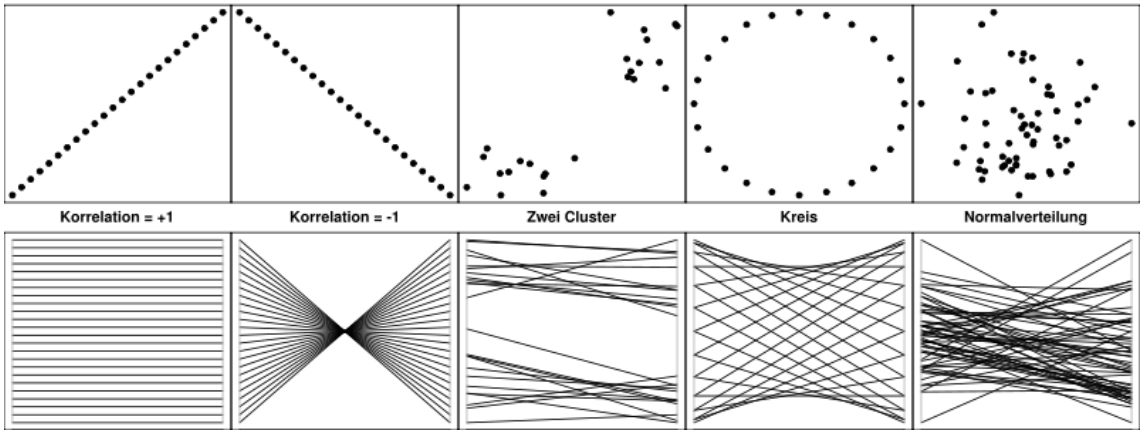


Abbildung 2.1: Beispiele für Muster, die bei parallelen Koordinaten auftreten können, und zugrundeliegende Zusammenhänge für den zweidimensionalen Raum [Sig].

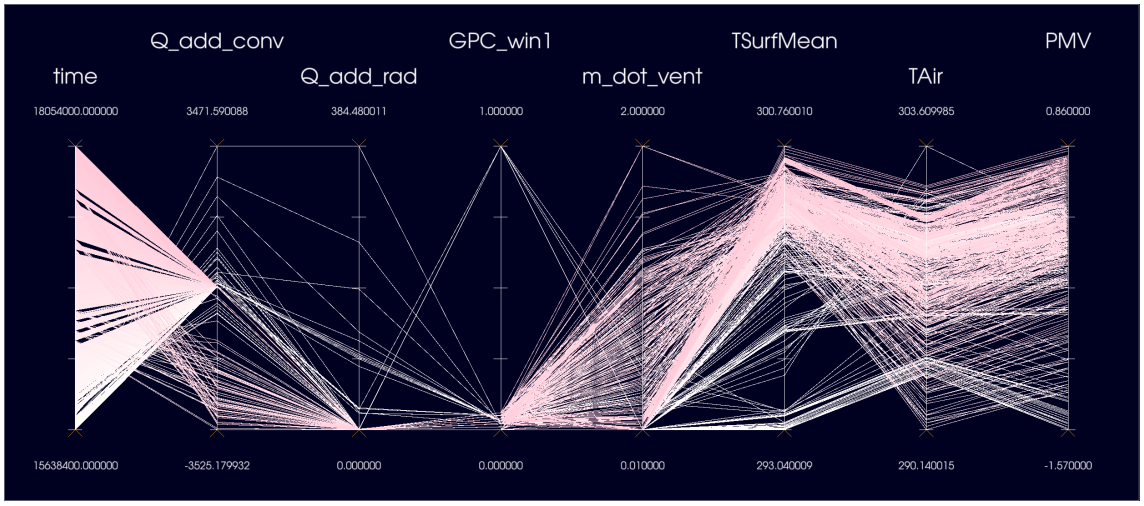


Abbildung 2.2: Beispiel-Screenshot von Parallelen Koordinaten in MegaMol

2.2 Megamol

Megamol ist ein von der Universität Stuttgart entwickeltes modulares Framework zur Prototypisierung von Visualisierungen von wissenschaftlichen Daten. Diese Datensätze können beispielsweise Partikeldaten aus physikalischen Simulationen oder Volumendaten sein. Durch den modularen Aufbau können dann verschiedene Visualisierungen auf diese Daten angewendet werden, um dem Nutzer eine Anaylse dieser Daten in Echtzeit zu erlauben. Eines der bereitgestellten Module dient der Darstellung von hochdimensionalen Daten in Form von parallelen Koordianten. Dieses Modul bietet zudem verschiedene Möglichkeiten mit diesen Daten zu interagieren, wie zum Beispiel Filtern und Hervorheben von Daten, zur Untersuchung auf Muster und verschiedene Möglichkeiten die Linien einzufärben, zum Beispiel anhand des Wertes einer der Dimensionen. Die Darstellung eines kleinen Datensatzes wird in Abb. 2.2 abgebildet.
Bei großen Datensätzen können jedoch Renderzeiten so lang werden, dass Echtzeitinteraktion mit den Daten schwierig oder sogar unmöglich wird. Im späteren Verlauf dieser Arbeit werden Ansätze vorgeschlagen, die den Renderprozess beschleunigen sollen und dabei die Qualität der Darstellung weitestgehend beibehalten.

2.3 Linien

Da die Darstellung von Parallelen Koordinaten grundlegend auf Linien beruht, spielen diese auch eine große Rolle für das untersuchte MegaMol-Modul. Zur Darstellung von Linien werden im folgenden zwei Verfahren näher beleuchtet.

2.3.1 Bresenham's Linien Algorithmus

Bresenham's Algorithmus wurde zuerst für einen digitalen Plotter vorgeschlagen, der sich nur in eine von 8 Richtungen auf einem festen Gitter, wie es auch Pixel auf einem Monitor sind, bewegen konnte [Bre65]. Auf einem solchen Gitter benötigt man zur Darstellung einer Linie genau zwei dieser acht Richtungen. Den Punkten in dem Gitter werden dann Ganzzahltupel zugewiesen und eine Linie mit Startpunkt $D_0 = (x_0, y_0)$ und Endpunkt $D_1 = (x_1, y_1)$ definiert. Durch eine Projektion von D_0 und D_1, sodass D_0 auf dem Ursprung liegt, kann allein durch den Oktanten, in dem sich D_1 befindet, die benötigten Richtungen bestimmt werden. Die Bewegungsrichtungen und die zu wählenden Richtungen werden in Abb. 2.3 dargestellt.
Mithilfe der Formel

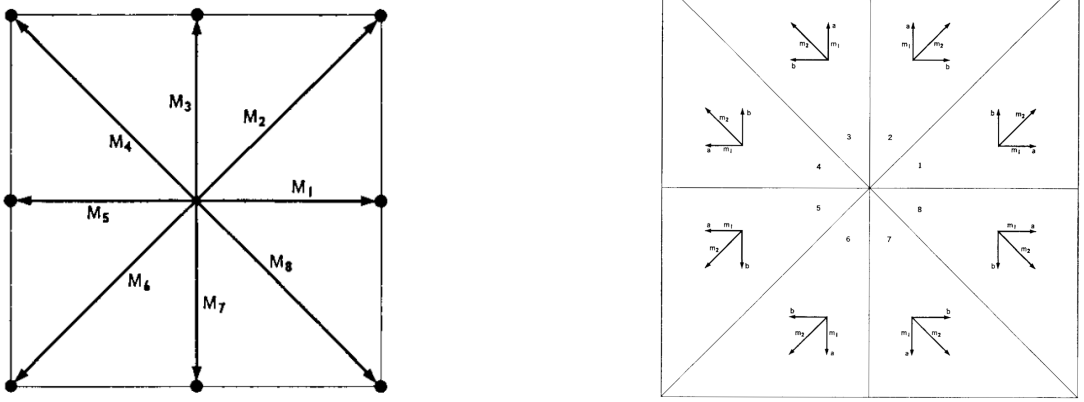
$$F(a,b) = (b - y_0) + \frac{(y_1 - y_0)}{(x_1 - x_0)}(a - x_0) = b(x_1 - x_0) + a(y_0 - y_1) + y_1x_0 - y_0x_1$$

kann für die Koordinaten (a, b) der Abstand zu der definierten Linie berechnet werden. Für Koordinaten die unterhalb der Linie liegen nimmt diese Funktion positive Werte an, für Koordinaten oberhalb der Linie nimmt $F(a, b)$ negative Werte an und ist genau 0, wenn sich (a, b) genau auf der Linie befindet. Um nun zu berechnen welche Pixel tatsächlich eingefärbt werden, ist es lediglich nötig die Funktion $F(a, b)$ für die Mitte der nächstgelegenen Pixel auszuwerten. Der Algorithmus wird in Listing 2.1 in Pseudocode aufgeführt.[1]
Ein Beispieldurchlauf dieses Algorithmus für den Fall D_1 liegt nach der Transformation in Oktant 1 und wird in Abb. 2.4 für die ersten drei Schritte dargestellt. Die benötigten Richtungen sind in diesem Fall Osten(M1) und Nordosten(M2). Der Algorithmus iteriert über $x = \{x_0, x_0 + 1, ..., x_1 - 1, x_1\}$ und berechnet in jedem Durchgang den Wert von $F(x, y)$, wobei (x, y) der Mittelpunkt der Pixel in östlicher und nordöstlicher Richtung, ausgehend von dem zuletzt gefärbten Pixel darstellt. Als Initialwert für y wählt man y_0. Die Auswertungen werden dann in der Variable d gespeichert und lautet zu Beginn der Rechnung $d = F(x_0+1, y_0+0.5)$, also dem Mittelpunkt zwischen dem östlichen und dem nordöstlichen Pixel, ausgehend von D_0. Der erste Durchlauf der Schleife, verdeutlicht in Schritt 1 in der Abb. 2.4, setzt Pixel (x_0, y_0) und stellt fest, dass $d > 0$ gilt, was einer Bewegung nach Osten(M1) entspricht und keine Veränderung der Variable y erfordert. Variable d wird nun mit dem Wert für $F(x+1, y+0.5) = F(x_0+2, y+0.5)$ aktualisiert. Da der Wert von $F(x_0+1, y_0+0.5)$ bereits bekannt ist und in d gespeichert wurde, reicht es die Differenz von $F(x_0+2, y_0+0.5)$ zu d zu addieren. Dadurch vereinfacht sich die Berechnung von $d_{neu} = d + F(x_0 + 2, y_0 + 0.5) - F(x_0 + 1, y_0 + 0.5)$ auf den Ausdruck $d_{neu} = d + (y_0 - y_1)$.
Im Folgenden Schritt 2 wird zunächst Pixel $(x_0 + 1, y_0)$ eingefärbt und festgestellt, dass $d < 0$ gilt und damit die nächste Auswertung $F(x_0 + 3, y_0 + 1.5)$ lautet. Auch hier kann durch $d_{neu} = d + F(x_0 + 3, y + 1.5) - F(x_0 + 2, y + 0.5)$ auf $d_{neu} = d + (y_0 - y_1)$ vereinfacht werden. Dies entspricht einer Bewegung nach Nordosten und erfordert daher auch eine Inkrementierung von y. Der nachfolgende Schritt 3 setzt dann Pixel $(x_0 + 2, y_0 + 1)$ und wertet erneut aus, was bis $x = x_1$ wiederholt wird.
Um den Algorithmus für andere Oktanten anwenden zu können genügt es die Eingabgedaten und die Funktion set_pixel() zu modifizieren. Für Oktant 2 wird der Algorithmus z.B. auf Punkten $D_n^* = (y_n, x_n)$ ausgeführt und die Funktion set_pixel() kehrt diese Änderung um, indem die beiden Koordinaten getauscht werden, bevor die Pixel eingefärbt werden.

[1] Wie gesehen in der Vorlesung Computergrafik der Universität Stuttgart.[Ert18]

Algorithmus 2.1 Bresenham's Linienalgorithmus

```
procedure DrawLine(D_0, D_1)
    y ← y_0
    d ← F(x_0, y_0 + 0.5)
    for x ← x_0, x_1 do
        set_pixel(x, y);
        if d < 0 then
            y ← y + 1
            d ← d + (x_1 − x_0) + (y_0 − y_1)
        else
            d ← d + (y_0 − y_1)
```



(a) Mögliche Bewegungsrichtungen eines digitalen Plotters

(b) Wahl der Bewegungsrichtungen, benannt m1 und m2

Abbildung 2.3: Funktionsweise eines digitalen Plotters

Der Algorithmus benötigt lediglich Addition und Subtraktion, um eine Linie zu zeichnen, welche im Allgemeinen schnell von aktueller Hardware ausgeführt werden kann. Zudem ist in aktuellen Verbraucher-Grafikkarten spezialisierte Hardware verbaut, um diesen Algorithmus weiter zu beschleunigen, wodurch sich der Algorithmus durch besondere Geschwindigkeit auszeichnet.
In Bresenham's Algorithmus ist sichergestellt, dass eine Linie ohne Unterbrechungen gezeichnet wird, da entlang einer Linie in jeder Pixelspalte genau ein Pixel gefärbt wird. Eine Bildung von sichtbaren Knoten, durch Färbung von mehreren übereinanderliegenden Pixeln ist bei dem gewählten Beispiel ebenso nicht möglich. Dieser Algorithmus färbt zudem ein Pixel entweder vollständig oder gar nicht ein, wodurch hier auch kein Anti-Aliasing, bei dem teilweise Einfärbungen benötigt werden, möglich ist und eine Linie im allgemeinen sichtbare Treppen bildet. Es wurden jedoch viele Erweiterungen vorgeschlagen, die den Algorithmus um diese Funktionalität erweitern [Wu91].

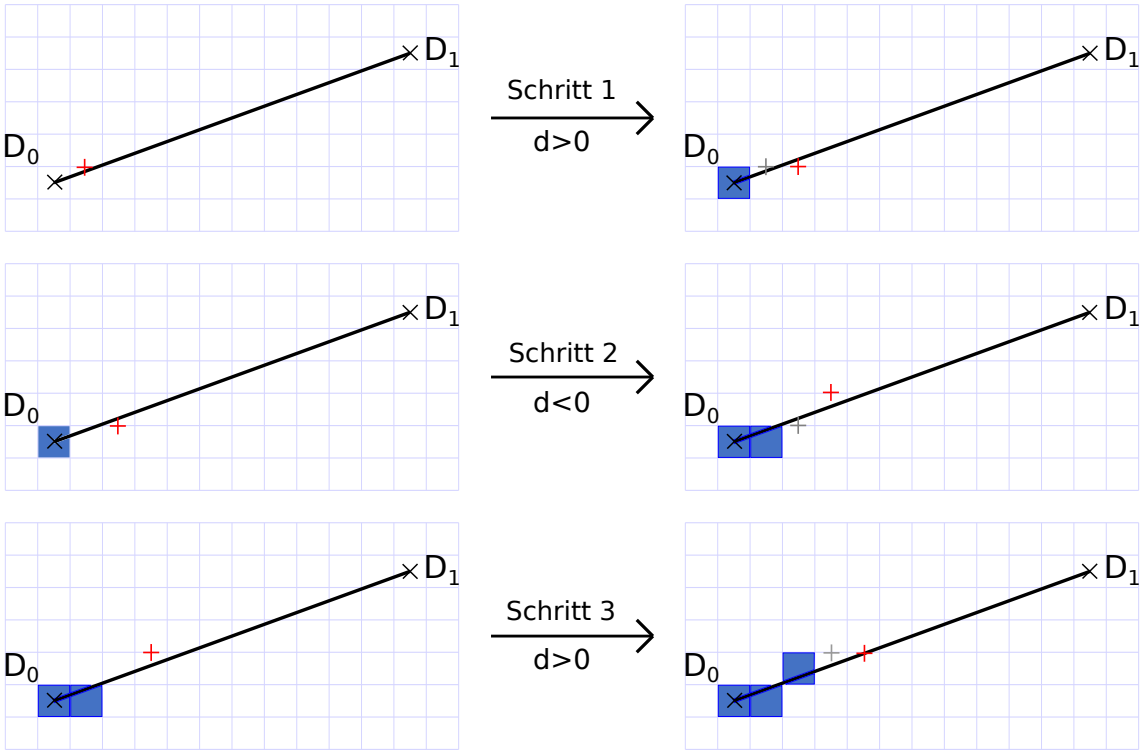


Abbildung 2.4: Beispielvisualisierung einer Linie (schwarz). Samplepositionen die in d zu dem jeweiligen Zeitpunkt gespeichert sind werden r dargestellt, mit letzen Positionen in grau.

2.3.2 Polygon-basierte Linien

Der zuvor beschriebene Algorithmus nach Bresenham ist für idealisierte Linien mit einer Dicke von einem Pixel vorgesehen. Auch wenn es möglich ist Linien mit einer Breite von mehr als einem Pixel zu erzeugen, werden bei diesem Algorithmus immer ganze Pixel eingefärbt, wordurch die Fläche die eine Linie im Screen-Space einnimmt abhängig von der Auflösung variieren kann. Zudem ist der Algorithmus lediglich für einfarbige, zweidimensionale Linien vorgesehen und stößt bei Visualisierung von Linien, bei denen Beleuchtung oder andere Tiefeneffekte gewünscht sind, an seine Grenzen. Für dichte dreidimensionale Abbildungen müssen häufig solche Effekte dargestellt werden, um die räumliche Interpretation zu gewährleisten. Die bisher beschriebenen Algorithmen sind für solche Anwendungen wenig geeignet, da Beleuchtung normalerweise in World-Space-Koordinaten berechnet wird und Bresenham's Algorithmus direkt Pixel im Screen-Space setzt. Um gängige Algorithmen für Beleuchtung und Tiefeneffekte leichter umsetzen zu können, sind Dreiecke zur Darstellung von Linien in solchen Fällen gut geeignet. Diese haben den weitere Vorteil, dass stilisierte Linien ebenfalls leichter umsetzbar sind, beispielsweise durch Überziehen der Dreiecke mit Texturen[MKG+97].
Die Dicke der Linie ist für polygonbasierte Linien damit allerdings auch im World-Space definiert und kann zu Lücken oder sogar unsichtbaren Linien führen, wenn die Dicke zu klein gewählt wird. Selbst wenn man eine Dicke von genau einem Pixelabstand wählt, kann es bei ungünstiger Lage, nämlich für den Fall, dass Linienränder genau durch Pixelmitten verlaufen, vorkommen, dass eine Linie eine Breite von zwei Pixeln im Screen-Space hat oder bei diagonalem Verlauf Lücken sichtbar

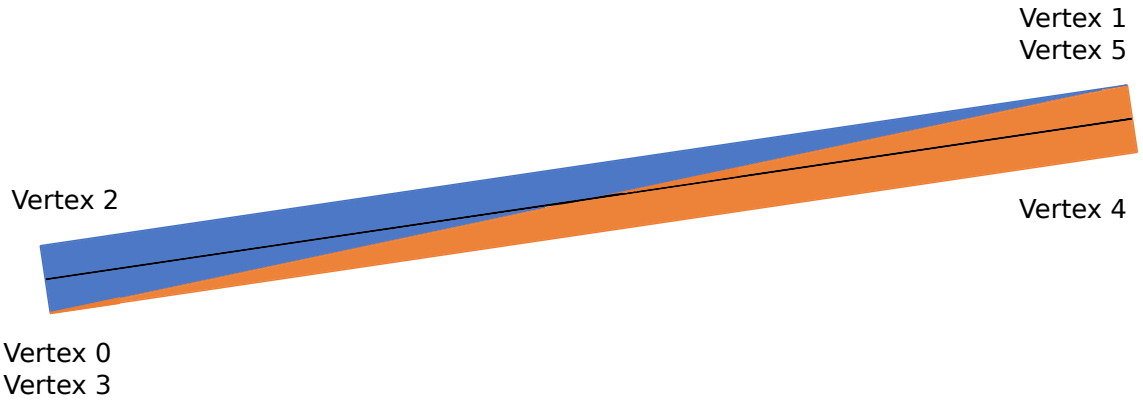


Abbildung 2.5: Darstellung von polygonbasierten Linien. Vertices bilden die 4 Eckpunkte der Linie. Auf dem unteren linken Eck und dem oberen Rechten Eck liegen 2 Vertices. Das obere, blaue Dreieck hat die Eckpunkte: Vertex 0, Vertex 1, Vertex 2. Das untere, orange Dreieck hat die Eckpunkte: Vertex 3, Vertex 4, Vertex 5

sind.
Für diese Arbeit spielen Polygon-basierte Linien eine große Rolle, da zur Beschleunigung des Renderings die Auflösung während des Rasterisierens nicht mit der Screen-Space-Auflösung übereinstimmt. Durch Definition der Linien durch Dreiecke bleibt die Fläche selbst nach Skalierung des Viewports im Screen-Space besser erhalten und konstante Sampling-Positionen liefern konstante Ergebnisse. Diese Eigenschaften sind notwendig zur originalgetreuen Rekonstruktion des Bildes aus den vorrangegangen Frames, da bei Verringerung der Auflösung, die Ergebnisse von Samplepositionen für beliebige Koordinaten im World-Space erhalten bleiben müssen.
Umgesetzt werden diese Linien durch zwei Dreiecke pro Liniensegment, deren Eckpunkte dann gemeinsam ein Rechteck bilden, wie dargestellt in Abb. 2.5. Ausgehend von Start- und Endpunkt der Liniensegmente werden dann die Vertices orthogonal zum Linienverlauf verschoben. Die Distanz diese Verschiebung kann beliebig gewählt werden und entspricht der Dicke der Linie. Wird dieser Wert als 0 gewählt, ist die Linie unsichtbar. Eine Dicke von exakten Pixelmengen ist jedoch schwierig, da Linien nun beliebig im Raum liegen können und daher auch bei einer Breite von genau einem Pixel Lücken entstehen können oder mehrere Samples die Linie treffen, wodurch sichtbare Knoten entstehen können.
Die letztendliche Anzeige wird dann durch die Rasterisierungspipeline der GPU erstellt. Dies führt damit auch zu Kompatiblität mit etablierten Anti-Aliasing-Methoden wie MSAA oder Supersampling, jedoch ist die Erzeugung auf diese Weise im allgemeinen teurer als Rendern durch Bresenhams Algorithmus.

2.4 Amortisiertes Rendern

In vielen grafischen Anwendungen kann die Menge benötigter Berechnungen im Fragment-Shader-Schritt den Ablauf der Programms verlangsamen und Antwortzeiten auf inakzeptable Werte erhöhen. Eine Möglichkeit diese Last zu verringern ist es die Renderauflösung zu verringern, was jedoch immer zu verringerter Anzeigequalität oder sogar zu Detailverlust führt. Durch amortisieren des Renderns über mehrere zeitlich aufeinander folgenden Frames, ist es möglich

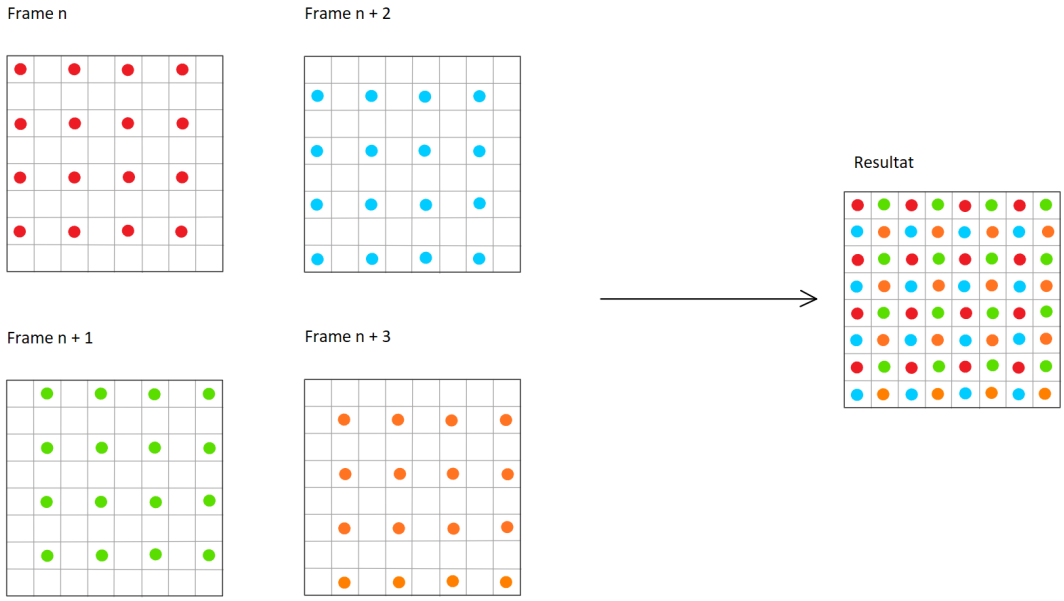


Abbildung 2.6: Amortisiertes Rendern mit Teilung in 4 Bilder mit jeweils einem Viertel der Auflösung. Abtastungspositionen markiert durch Kreise.

die Qualität des Bildes beizubehalten und gleichzeitig Antwortzeiten des Frames zu verbessern. Dabei werden also n aufeinanderfolgende Frames in verringerter Auflösung zu einem Bild in voller Auflösung zusammengesetzt. Die Samplepositionen der niedrigauflösenden Frames werden jeweils so verschoben, dass sie wieder mit den Samplepositionen der vollen Auflösung übereinstimmen. Wählt man zum Beispiel $n = 4$, wird jedes einzelne Frame in einem Viertel der Auflösung, also halber Breite und halber Höhe, gerendert und die Positionen um ein halbes Pixel der Anzeigeauflösung verschoben. Ein Beispiel dieses Vorgehens ist in Abbildung 2.6 dargestellt. Eine besonders weit verbreitete Variante dieses Vorgehens nennt sich "Checkerboard-Rendering", bei dem man das gewünschte Ergebnis in 2 Schachbrettmuster teilt. Diese beiden Muster sind so verschoben, dass weiterhin alle Sample der Zielauflösung getroffen werden. Diese werden dann abwechselnd gerendert und dann zu dem gewünschten Resultat zusammengesetzt. Dieses Vorgehen wurde in Abbildung 2.7 dargestellt.

Durch die verringerte Zeit die nun nötig ist um ein Frame zu erstellen verringert sich auch die Eingabeverzögerung und führt zu einer besseren Antwortzeit des Programms. Jedoch müssen für diese Art des Renderns alte Frames zum Zweck des Zusammensetztens gespeichert werden, was zusätzlichen Speicher beansprucht. Dies macht diese Vorgehen ungeeignet in Sitationen, die bereits durch Verfügbarkeit von Speicher oder Speicherbandbreite limitiert sind, wie es oft in mobilen Geräten, wie Tablets oder Laptops der Falls sein kann.

Ein weiteres Problem entsteht bei der Bewegung der Kamera oder Bewegung von Objekten in der Szene. Da nur ein Teil des Bildes in jedem Rendervorgang aktualisiert wird, zeigen Teile des Bildes alte Daten an und führen damit zu starken Ghosting-Artefakten. Dieses Problem kann durch Verschieben und Skalieren der alten Frames entsprechend ausgeglichen werden, was jedoch keine neuen Informationen erzeugt und damit zu Lücken führen kann. Es gibt viele Ansätze, wie solche Lücken möglichst gut gefüllt oder erraten werden können, da sich diese Arbeit jedoch auf

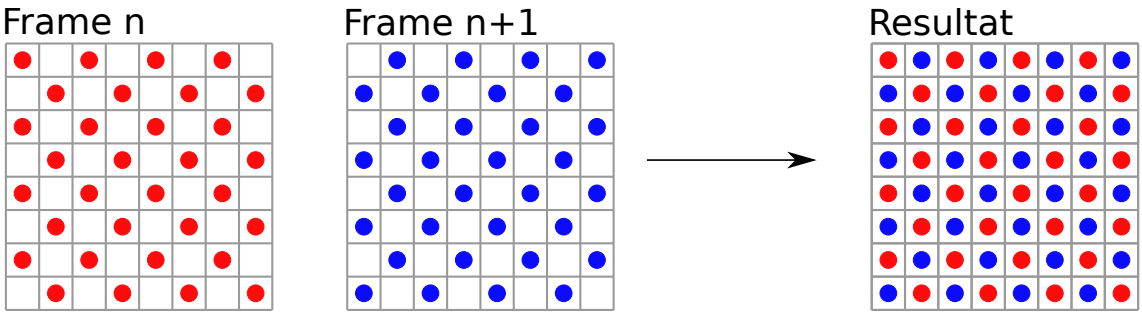


Abbildung 2.7: Beispieldarstellung von Checkerboard-Rendering. Samplepositionen markiert durch Kreise.

Darstellung von zweidimensionalen Grafiken beschränkt, wird nicht weiter auf diese eingegangen und die problemgerichtete Lösung für diesen Kontext wird im Kapitel Implementierung näher beschrieben.

2.4.1 Temporales Supersampling und Antialiasing

Ein mit dem zuvor beschriebenen Vorgehen verwandte Technik, namens Temporales Supersampling, ermöglicht Ermittlung von Daten, die im aktuellen Frame zwar fehlen, jedoch aus alten Frames erschlossen werden können. Bei unbeweglichen Bildern ist dies nicht möglich, da jedes Bild die selben Informationen liefert. Durch Bewegungen im Subpixelbereich können jedoch weitere Informationen über die Szene gewonnen werden. Dieser sogenannte Jitter der Kamera ändert sich dann für jedes Bild und ermöglicht Abtastung von leicht verschobenen Positionen und erschließt beim Rendervorgang weitere Daten. Bei Bewegung der dargestellten Szene erzeugt dies jedoch wie auch bei amortisiertem Rendering Ghosting-Artefakte durch Daten deren Position bei der Abtastung nun nicht mehr mit deren aktueller Position übereinstimmt. Ausgeglichen werden kann dies durch Korrektur der Position durch Reprojektion der alten Abtastungswerte auf die neue Position in der Szene.
Wird dieses Konzept angewandt, um Kantenglättung zu erreichen, spricht man von temporalem Anti-Aliasing (TAA). Es wird also versucht die Abtastungen durch Verschiebung der Kamera so zu platzieren, dass Farb- und Tiefendaten verfügbar werden. Solche Verschiebungen können zufällig oder entlang einer festen Sequenz von Positionen geschehen. Bei der Rekonstruktion wird dann zusammen mit diesen zusätzlichen Positionen aus alten Bildern ermittelt, wie ein Pixel eingefärbt wird. Diese Verfahren haben in Videospielen bereits andere weit verbreitete Techniken wie MSAA fast verdrängt und gelten als beliebteste Methode um Kantenglättung zu erreichen [YLS20].

3 Verwandte Arbeiten

Die Grundlage für alle Visualisierungen in dieser Arbeit bilden Parallele Koordinaten. Diese Methode zur Darstellung von multivarianten Daten wurde 1885 von Maurice d'Ocagne vorgeschlagen und hat sich vor allen in den letzten Jahrzehnten großer Beliebtheit erfreut[Oca85].

Bereits 1964 wurde von J. E. Bresenham der nach ihm benannte Algorithmus vorgeschlagen, um auf einem Digitalplotter Linien einzufärben. Dieser Algorithmus wurde in Kapitel 2.3.1 näher erleutert und spielt eine große Rolle, beim direkten Rendern von Linien. Der Algorithmus kommt mit Addition und Subtraktion aus und wird von Konsumentenhardware breitflächig unterstützt, wodurch sich der Algorithmus durch hohe Geschwindigkeit auszeichnet. Trotzdessen wurden zahlreiche Verbesserungen der Algorithmus vorgeschlagen. Eine dieser Verbesserungen, die ursprünglich von Reggiori vorgeschlagen [Reg72] und dann später von Bresenham weiter für den Spezialfall von Linien verbessert wurde, setzt ganze Segmente von Pixeln in einem Schritt und ist anwendbar auf beliebige Kurven. Hiermit kann also mehr als ein Pixel pro Schritt gesetzt werden, was besonders dann eine Verbesserung gegenüber des ursrpünglichen Algorithmus von Bresenham darstellt, wenn die Operationen, die zur Einfärbung eines Pixels benötigt werden teuer sind und man durch Setzten mehrere Pixel gleichzeitig eine Beschleunigung erreichen kann.
Wie bereits in Abschnitt 2.3.1 angesprochen, ist Bresenhams Algorithmus nicht nativ mit Anti-Aliasing vereinbar. Es wurden mehrere Möglichkeit vorgeschlagen, diese Hürde zu überwinden. 1981 wurde von Gupta und Sproull ein Algorithmus vorgeschlagen, der mithilfe eines Kegelfilters die Intensität der Pixel, die eine Linie von beliebiger Dicke darstellen, skaliert [S G81]. Anhand der Distanz der Pixel zum Mittelpunkt des Filters wird die Intensität der Pixel mit zunehmender Distanz linear verringert. Dadurch wird bei der Berechnung ein weiterer Schritt benötigt, der jedoch durch eine Look-up-Tabelle in eine Anzahl von Fällen unterteilt wird, um die Geschwindigkeit bestmöglich zu optimieren. Durch Setzen mehrerer Pixel pro Schritt, die unterschiedlich intensiv eingefärbt werden, entstehen dann letztendlich geglättete Kanten und Linien. Von Wu wurde 1991 ein Verfahren vorgeschlagen, das Bresenham's Algortihmus durch Einführung eines Fehlermaßes erweitert [Wu91]. Anhand dieses Fehlermaßes werden zunächst die Distanzen der Pixel zu der idealen Linie festgestellt und dann proportional dazu in verschiedenen Intensitäten eingefärbt. Somit entsteht wie auch bei Gupta und Sproull Glättung der Linien, jedoch kommt dieser Algorithmus mit Additionen und Subtraktionen aus, was die Geschwindigkeit im Vergleich verbessert.
Alternative Arten Linien zu rendern sind besonders im Bereich des "non-phtorealistic Rendering"verbreitet. 1997 wurden von Markosian et al. Verfahren vorgeschlagen aus 3D-Objekten vereinfachte, stilisierte Zeichnungen mit verschiedenen Linientypen zu erzeugen. Dies war einer der erste Vorschläge, die nonphotorealistisches Rendern in Echzeit umzusetzen und nutze dafür unter anderem Meshes, mit deren Hilfe Texturen entlang der Silhoutte des Objtektes aufgespannt wurden und so je nach Wahl der Textur einfaches Ändern des Linienstils ermöglichte [MKG+97]. Von Rössl und Kobbelt wurde ein Algorithmus vorgeschlagen, der aus 3D-Objekten Line-Art-Zeichnungen erstellt. Dabei wurde das Model auf eine Fishbone-Struktur reduziert, auf der dann wiederum die Linien aufgezogen wurden. Die Linien wurden durch Polygone mit einer konstanten Dicke umgesetzt, von

deren Fläche jedoch nur ein mittlerer Streifen schwarz und der verbleibende Rand weiß eingefärbt wird. Dadurch konnten sich Linien in dichten Bereichen überlappen und gegebenenfalls auslöschen, um einen Line-Art-Eindruck zu verleihen [RK00]. Von Everts et al. wurde der Tiefeneindruck von dichten Linienclustern durch verwendung von tiefenabhängigen "Halos"verbessert. Hierbei wurden Linien durch Triangle-Strips dargestellt. Diese Streifen beinhalten eine schwarz eingefärbte Linie und einen weißen Rand, ähnlich wie auch bei Rossl et al. Jedoch werden bei dem Shading der Fragmente die Tiefenwerte abhängig von der Distanz zu der Mittellinie des Streifens angepasst, indem Fragmente in dem weißen Bereichen weiter von der Kamera entfernt werden. Dadurch entsteht bei sich kreuzenden Linien eine Lücke in der räumlich weiter entfernten Linie und hebt die Linie im Vordergrund weiter hervor[EBRI09]. Eine effektive Methode, Sampling-Operationen zu reduzieren bei möglichst guter Bildqualität, ist das Entkoppeln von Sichtbarkeits- und Shading-Samples. Diese Funktionalität wurde bereits in der Reyes-Rendering-Arichtektur in 1987 von Pixar implementiert[CCC87]. Diese Architektur basiert auf Mikropolygonen, deren Farbwerte vor der Sichtbarkeit geshadet werden, um dann später, in einem separaten Schritt, mehrfach pro Pixel durch Sichtbarkeitssampling zu dem endgültigen Ergebnis für diesen Pixel zu kommen. Dies hat jedoch den Nachteil, dass beim Shading eine große Menge an Mikropolygonen berechnet werden, die nie angezeigt werden. Andererseits ermöglichte dieses Vorgehen auch Nutzung von Vorteilen wie Vektorisierung des Shadingprozesses und Vermeidung von Textur Thrashing, wobei nur sehr kleine Teile einer Textur abgerufen werden und oft zwischen verschiedenen Texturen gewechselt werden muss. Zudem implementierten Pixar bereits stochastische Samplepositionen um Aliasing-Artefakte zu reduzieren. Dieses Jittern von Samplepositionen wurde von Cook vorgeschlagen, da Point-Sampling-Methoden bei uniformen Samplingpositionen störende Aliasing-Effekte aufweisen, die oft als "Jaggies"bezeichnet werden . Mithilfe von stochastischem Jitter konnten diese Effekte jedoch durch noisebasierte Artefakte ausgetauscht werden, deren Auswirkung auf das Bild im allgemeinen weniger störend empfunden wird. Für viele der Schwächen, die bei der Reyes-Architektur auftraten, wurden später Verbesserungen vorgeschlagen. Ragan-Keylley et al. schlugen einen allgemeineren Ansatz vor, Sichtbarkeit und Shading in Graphic Pipelines zu entkoppeln[RLC+11]. Dieser Ansatz basiert nicht wie die Reyes-Architektur auf Micropolygonen die unabhängig von ihrer Sichtbakteit geshadet werden, sondern ermittelt zuerst die Sichtbarkeit von Fragmenten und shadet erst dann. Dies wurde durch eine on-demand Sampling Strategie umgesetzt, die nur dann ein Fragment shadet, wenn deren Wert nicht in dem bereitgestellten Z-Buffer gefunden wurde.
Da die genannten entkoppelten Renderverfahren Sichtbarkeit und Shading in verschiedenen Raten durchführen, ist Supersampling von Sichtbarkeit deutlich günstiger. Zur Verbesserung von Performance in Videospielen und andern interakitven Umgebungen ist der Ansatz des Checkerboard-Renderings besonders weit verbreitet. Diese Verfahren hat mit dem Erscheinen der Playstation 4 große Beliebtheit erlangt, da dieses Verfahren bereits in der Hardware implementiert wurde und daher ohne großen Aufwand für Entwickler zur Verfügung steht. Auch von Intel wurde in einem Paper über die Anwendung der Technik in Rainbow-Six Siege gesprochen, wobei hier die Samplepositonen mithilfe von MSAA bestimmt werden, um von weiterer Hardwareoptimierung zu profitieren [ML18]. Eine weitere Form des entkoppelten Renderens wurde von Xiao et al. vorgeschalgen, in dem weniger als ein Pixel pro Frame geshadet wird, weshalb dieses Verfahren auch als "coarse pixel shading"bezeichnet wird[XLV18]. Dannach wird durch verschiedene Filter, mithilfe der Sichtbarkeitssample ermittelt, welche Farbe ein Pixel tatsächlich haben soll. Dieser Ansatz beschleunigte Rendern noch weiter im Vergleich zu Checkerboard-Rendering, bei ähnlicher Bildqualität.

4 Implementierung

4.1 Linien

Zu Beginn dieser Arbeit verwendete Megamol zur Darstellung von Parallelen Koordinaten den von OpenGL bereitgestellten Algorithmus zum Zeichnen von Linien. Zur Rasterisierung von Linien verwendet OpenGL eine Variation des Bresenham-Algorithmus [SA], was für die visuelle Dicke der Linien eine Abhängigkeit von der Renderauflösung erzeugt. Da dieser Effekt negativen Einfluss auf die Qualität des Bildes bei amortisiertem Rendern haben kann, wurde zunächst die Definition der Linien durch Dreiecke implementiert. Um mehr Kontrolle über den visuellen Eindruck der Linien zu erhalten wurde das Rendern der Linien eine Serie von Dreiecken umgestellt. Die Konstruktion dieser Dreiecke geht zunächst von der bereits bekannten Folge von Vertices, die auf den jeweiligen parallelen Achsen liegen, aus und kopiert und verschiebt diese.

Die Berechnung dieser Positionen wird in Abb. 4.1 dargestellt. Eingabe für den folgenden Algorithmus sind also die n Punkte $D_0, D_1, ..., D_n$, die jeweils auf einer der $n + 1$ Achsen liegen. Der idealisierte Verlauf jedes Liniensegments kann durch die Vektoren $\overrightarrow{d_n} = \overrightarrow{D_n} - \overrightarrow{D_{n+1}}$ dargestellt werden. Anhand dieser Vektoren werden dann zunächst die Orthogonalen $\overrightarrow{o_n}$ durch Drehung der Verlaufsvektoren um 90° berechnet und normalisiert. Zur Ermittlung der Verschiebung für die Vertices der Dreiecke die später die Linie definieren, werden nun 2 Fälle unterschieden: (1.) Der behandelte Vertex befindet sich auf der ersten oder letzten Achse und (2.) der Vertex befindet sich einer der übrigen Achsen. Für Fall 1 ist es lediglich nötig die Orthognale o_0 bzw. o_{m+1} mit der

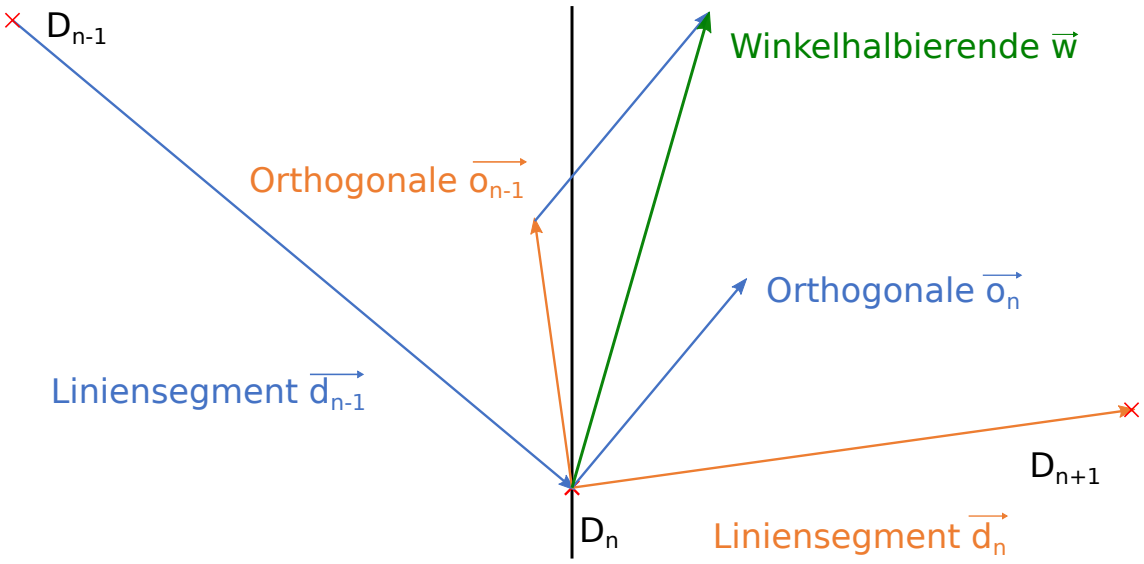


Abbildung 4.1: Berechnung der Winkelhalbierenden

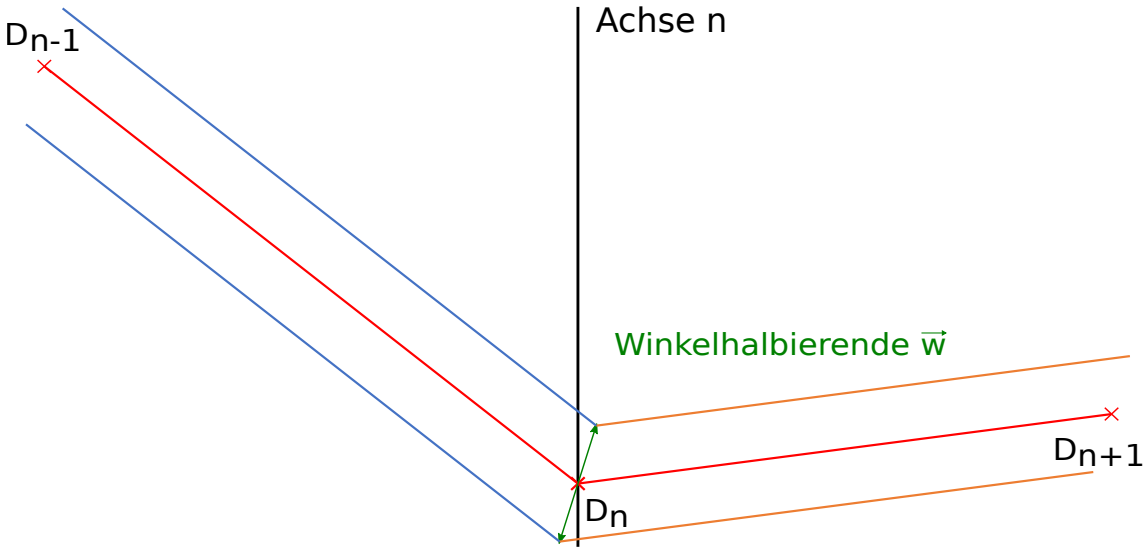


Abbildung 4.2: Ergebnis der Verschiebung der Vertices. Die idealisierte Linie nun in rot, die Grenzen der Polygone in orange und blau.

gewollten Dicke der Linie zu multiplizieren und auf den achsenparallelen Vektor $\vec{a} = (0, 1)$ zu projizieren. Fall 2 berechnet zunächst die Winkelhalbierende durch addieren der Orthogonalen $o_{n-1} + o_n$, für die der betrachtete Vertex entweder Start- oder Endpunkt bildet. Diese Summe wird nun Winkelhalbierende $\overrightarrow{w}$ benannt und normalisiert. Die letztendliche Verschiebung multipliziert diese Winkelhalbierende mit der gewünschten Dicke, falls der Vertex den oberen Rand abbildet, oder mit der negativen Dicke, falls es sich um den unteren Punkt handelt. Das Ergebnis dieser Berechnung wird in Abb. 4.2 dargestellt.

4.2 Amortisierung

Um den Renderaufwand pro Frame zu reduzieren wird die Renderauflösung der Frames pro Zeitschritt reduziert und später zusammen mit alten Frames zu einem vollen Anzeigebild zusammengesetzt. Zu diesem Zweck müssen potentiell viele alten Frames im Speicher erhalten bleiben, wodurch Rechenaufwand mit erhöhten Speicherbedarf getauscht wird. Es ist außerdem nötig die Kamera im Subpixelbereich zu verschieben, was durch eine Multiplikation einer Translationsmatrix auf die Projection-Matrix erreicht wird. Hierfür werden mehrere Ansätze implementiert, die sich in der Anzahl der Abtastungen pro Schritt und der Rekonstruktion unterscheiden, was Auswirkungen auf sowohl die Bildqualität als auch auf die Performanz hat.

4.2.1 Multisampling-Checkerboard Rendering (MS-CBR)

Dieses Verfahren macht Gebrauch von sogenanntem „Checkerboard Rendering“, was hier einer Verringerung der Abtastungen pro Frame auf die Hälfte entspricht. Die folgende Implementierung orientiert sich an einem von Intel vorgestelltem Whitepaper[ML18].
Die Renderauflösung wird hierfür zunächst auf ein Viertel der Anzeigeauflösung reduziert. Um

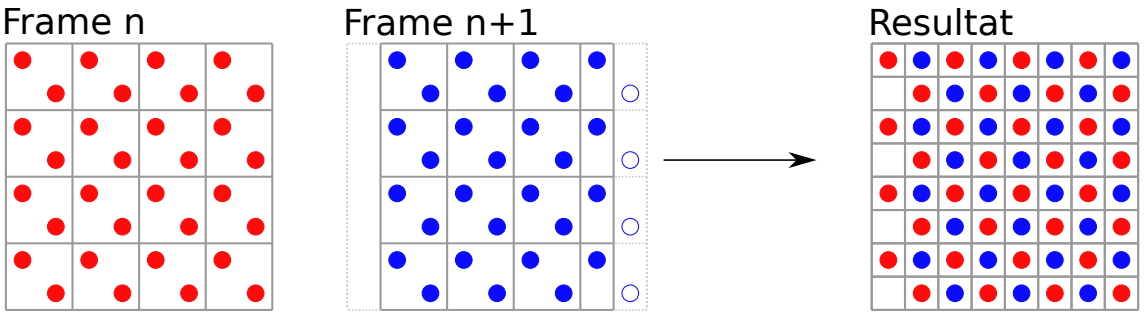


Abbildung 4.3: MS-CBR: Rote Abtastungen in Bild n, ohne Verschiebung erstellt. Blaue Abtastungen werden nach einer Verschiebung von einem Anzeigepixel nach rechts in Bild n + 1 ermittelt. Hohle Kreise markieren Abtastungen, die im Ergebnisbild nicht berücksichtigt werden, da sie außerhalb des Anzeigebereichs liegen.

nun weitere Abtastungen pro Frame zu erhalten wird mithilfe von 2x MSAA jedes Fragment zwei mal abgetastet. Die zusätzlichen Abtastungen können durch die gute Hardware- und Softwareoptimierung von MSAA hier sehr schnell ermittelt werden. Nach diesem Schritt sind also die Mitten der oberen rechten und unteren linken Quadranten jedes Pixels bekannt. Der folgende Frame wird nun mit einer Verschiebung von einem Pixel nach rechts gerendert um die fehlenden Positionen zu erhalten. Die Samplingpositionen stimmen letztendlich mit den Mitten der Anzeigepixel überein. Bei der Rekonstruktion werden dann die letzten zwei Frames entsprechend ihrer Abtastungspositionen zu einem Frame in Anzeigeauflösung zusammengesetzt. In diesem Schritt ist also für jeden Anzeigepixel eine Abtastung in der Pixelmitte bekannt. Zur Rekonstruktion ist es also lediglich nötig, die entsprechenden Abtastungen wieder auf die korrekten Pixel der Anzeige abzubilden. An dem linken Rand entsteht jedoch durch die Verschiebung der Abtastungsposition eine Lücke in jeder zweiten Zeile, die in dieser Arbeit akzeptiert und nicht gefüllt wird. Für dieses Verfahren müssen lediglich die letzten zwei Frames gespeichert werden, was in der Regel ein akzeptabler Speicherbedarf ist. Hierbei ist jedoch zu bedenken, dass nun pro Pixel beide ermittelten Abtastungspositionen gespeichert werden. Die Abtastungspositionen die in diesem Verfahren erhoben werden und die Rekonstruktion sind in Abb. 4.3 zu sehen.

4.2.2 Vierfach amortisiertes Rendern (4xAR)

Um die Anzahl der Abtastungen pro Frame noch weiter zu reduzieren wird nun auf MSAA verzichtet. Hierfür werden weiterhin Frames in einem Viertel der Anzeigeauflösung erzeugt, deren Abtastungspositionen je auf die Mitten der Pixel in Anzeigeauflösung verschoben werden, was in Abb. 4.4 abgebildet ist. Daraus ergibt sich ein Bedarf von 4 Frames um ein vollständiges Anzeigebild zu erzeugen, bei dem jedes Frame ein Viertel der benötigten Abtastungen enthält. Bei der Rekonstruktion entsprechen die Positionen der Abtastungen weiterhin den Pixelmitten der Anzeige, wodurch die Rekonstruktion wieder lediglich diese Zuteilung umsetzen muss. Der Speicherbedarf für diese Rekonstruktion steigt trotz der nun vier benötigten Frames nicht weiter im Vergleich zu MS-CBR an, da hier nur noch die Hälfte der Abtastungen pro Frame gespeichert werden muss.

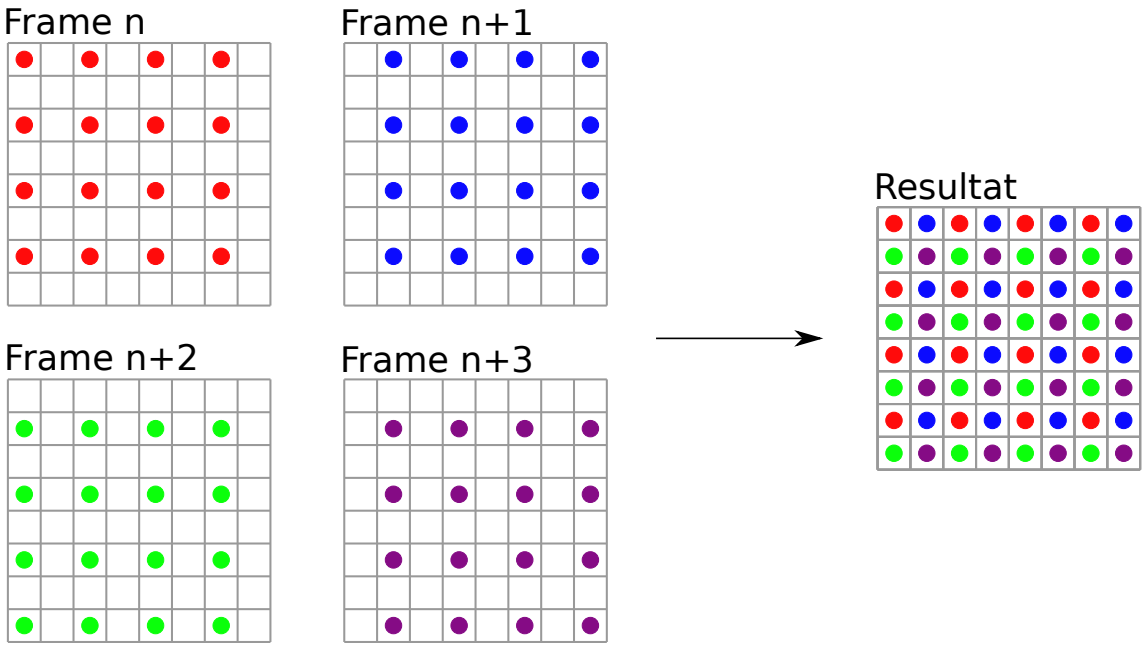


Abbildung 4.4: 4xAR: Je Frame wird ein Viertel der Pixelmitten abetastet. Rekonstruktion setzt dann die so ertasteten Werte für die entsprechenden Pixel der Anzeige.

4.2.3 Vierfach amortisiertes Rendern mit Eckenabtastung (4xAR-C)

Dieses Verfahren tastet nun nicht mehr die Pixelmitten der Anzeige ab, sondern verschiebt diese Abtastungspositionen auf die Ecken des Pixelgitters, wie in Abb. 4.5 zu sehen ist. Die Anzahl der Abtastungen pro Frame, sowie der Speicherbedarf dieses Verfahrens ist ansonsten identisch mit Verfahren 4xAR. Bei der Rekonstruktion des hochauflösenden Frames werden nun die vier nächsten Abtastungspositionen zu der Pixelmitte gemittelt, was jeweils den vier Ecken des Pixels entspricht der rekonstruiert wird. Zweck dieser Rekonstruktion ist es die Kanten die bei Linien entstehen durch geschickte Wahl der Samplingpositionen zu glätten, ohne aufwändige Verfahren wie Überabtastung anzuwenden.

4.2.4 Amortisiertes Rendern mit Supersampling (SS-AR)

Um das durch 4xAR-C eingeführte Anti-Aliasing weiter zu verbessern, kombiniert dieses Verfahren nun Supersampling mit den zuvor beschriebenen Verfahren 4xAR und 4xAR-C. Hierfür wird jedes Pixel 25 mal abgetastet, indem die Abtastungspositionen auf einem uniformen 5x5 Gitter in jedem Frame verschoben werden. Bei der Rekonstruktion kann man nun jedem Pixel der Anzeige 25 Abtastungen zuweisen. Um temporales Anti-Aliasing zu erreichen wird letztendlich der Wert der letzten 25 Abtastungen für einen Pixel gemittelt. Der Speicherbedarf für dieses Verfahren steigt jedoch enorm an, da für jeden Pixel der Anzeige nun 25 Abtastungen gespeichert werden müssen. Dies erzeugt aber nicht nur einen hohen statischen Speicherbedarf, sondern erfordert auch Auswerung dieser gespeicherten Werte, was hohe Speichertransferraten erfordert.

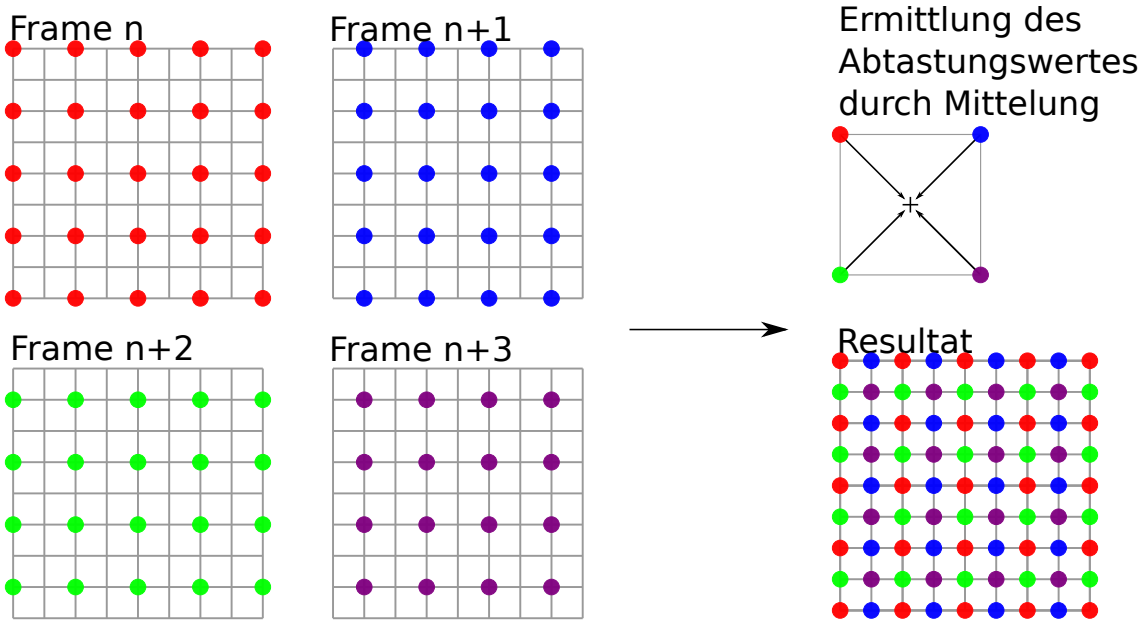


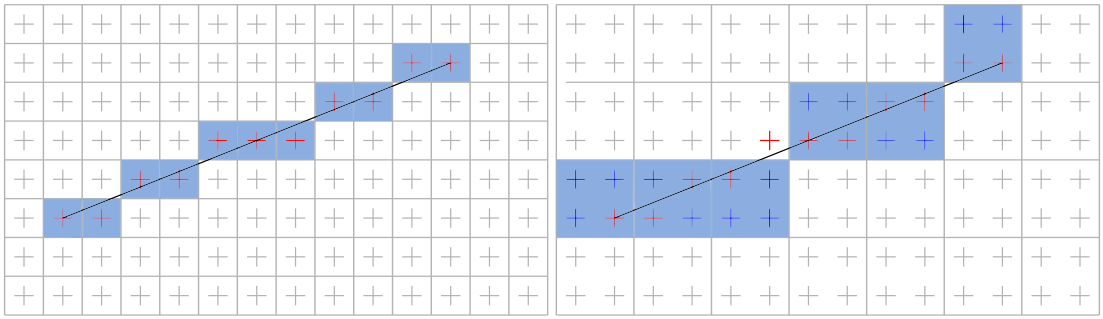
Abbildung 4.5: 4xAR-C: Abtastungspositionen sind nun auf den Pixelecken. Zur Rekonstruktion werden die Abtastungen an den Ecken zunächst gemittelt.

Algorithmus 4.1 Ermittlung der Reprojektionsmatrizen

$M_{pmv} \leftarrow M_p * M_{mv}$
$M_J \leftarrow$ jitter()
invMatrices[frameID % framesNeeded] $\leftarrow M_J * M_{pmv}$
for $i \leftarrow 0$, framesNeeded **do**
    moveMatrix[i] $\leftarrow$ invMatrices[i] $*$ inverse(M_{pmv})
end for=0

4.2.5 Bewegung

Alle bisher beschriebenen Rekonstruktionen wurden unter der Annahme beschrieben, dass während dem Rendern der benötigten Frames keine Bewegung der Szene stattfindet. Dies ist jedoch bei interaktiven Anwendungen wie Megamol nicht immer der Fall, weshalb ein Ausgleich für solche Bewegungen stattfinden muss. Da sich diese Arbeit auf die Darstellung von Parallelen Koordinaten beschränkt, welche eine zweidimensionale Anzeige mit orthographischer Kamera ist, können Probleme wie Überdeckung von Fragmenten in alten Frames nicht auftreten. Für 3D-Szenen müssen hierfür oft komplexe Lösungen angestrebt werden, um solche Artefakte zu korrigieren. In Megamol's Parallelen Koordinaten Modul ist die Position der Kamera durch die Model-View-Matrix und die Projection-Matrix definiert. Das bedeutet, dass Bewegung mithilfe von alten Matrizen nachvollzogen werden kann. Um Bewegung der Szene bei alten Bildern ausgleichen zu können, wird für jedes erstellte Bild das Produkt der aktuellen Model-View-Matrix M_{mv} mit der Projection-Matrix M_p und der Translationsmatrix die den Jitter umsetzt M_j gespeichert. Für jeden Frame wird dann auf alle gespeicherten Matrizen die Inverse der aktuellen Projection-Model-View-Matrix multipliziert und die resultierende Matrix zum Ausgleich der Bewegung verwendet. Diese Rechnung wird in 4.1 in Pseudocode aufgeführt. Für den aktuellen Frame löschen sich $(M_{pmv})^{-1}$ und M_{pmv} aus, für



(a) Einfärbung von Pixeln mit Bresenham's Algorithmus bei identischen Render- und Anzeigeauflösungen. Die Abtastungspositionen, die in diesem Schritt eingefärbt werden sind in rot markiert, die Pixel die in Renderauflösung gefärbt werden sind in blau eingefärbt. Hier ist kein Unterschied zwischen den beiden Auflösungen feststellbar.

(b) Einfärbungen bei unterschiedlicher Render- und Anzeigeauflösungen. Hier ist nun erkennbar, dass nicht nur die roten Abtastungspositionen von Bresenham's Linien eingefärbt werden, sondern auch die in blau markierten Abtastungen in den Linien liegen. Die Linie erscheint nun dicker, als eine idealisierte, ein Pixel dicke Linie (hier in rot markiert). Es ist auch möglich, dass Abtastungen, die zuvor in der Linie lagen, dies nun nicht mehr tun.

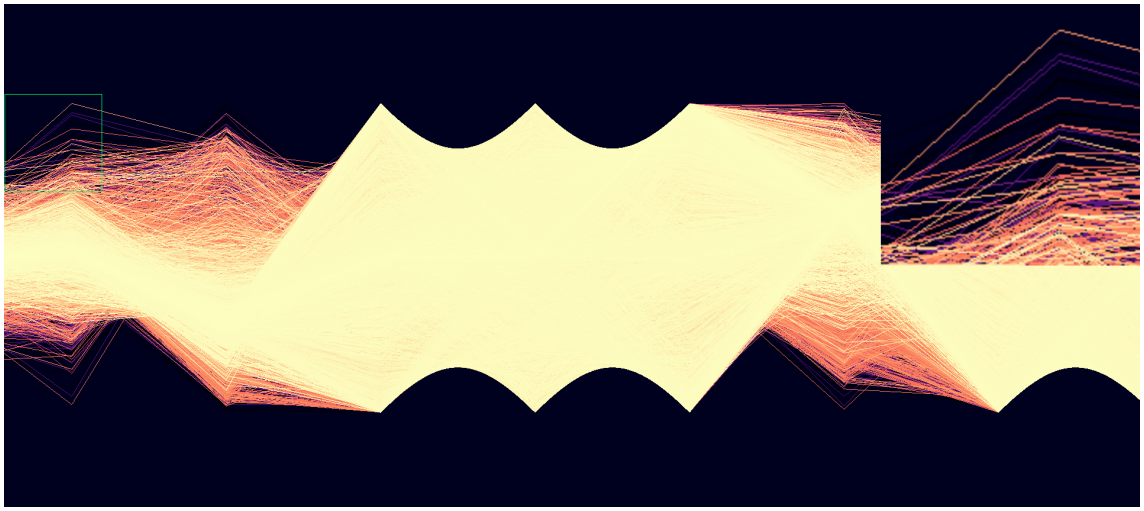
Abbildung 4.6: Vergleich von Liniendicken bei identischen und unterschiedlichen Render- und Anzeigeauflösungen.

ältere Frames ergibt sich für diese Rechnung jedoch eine Matrix, die die Skalierung und Translation der Szene widerspiegelt.

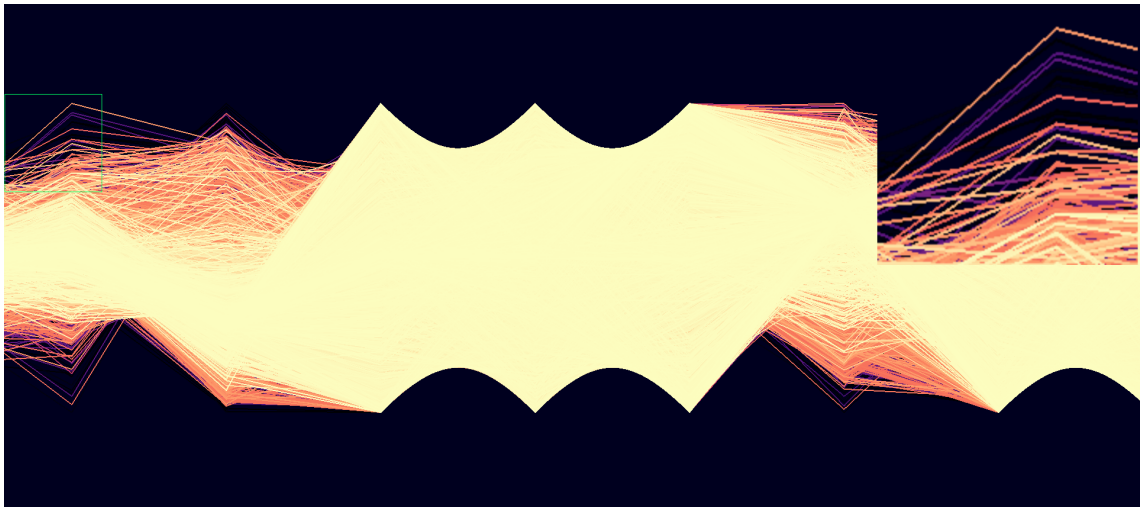
4.3 Amortisierung von Bresenham's Linien

Wie in vorrangegangenen Kapiteln beschrieben nimmt die Fläche, die eine Linie im Anzeigebild einnimmt, mit sinkender Renderauflösung zu. Grund dafür ist, dass Bresenham's Algorithmus immer vollständige Pixel in einem festen Farbton einfärbt. Reduziert man nun die Renderauflösung bei gleichbleibender Anzeigeauflösung, nehmen die Pixel der Linien nun mehr Fläche auf der Anzeige ein. Dieser Effekt ist in Abb. 4.6 verdeutlicht. In der Abbildung ist erkennbar, wie sichtbare Unterschiede bei Skalierung der Auflösung für Bresenham-Linien entstehen können. Wenn Linien jedoch als Polygone definiert sind, deren Positionen in Weltkoordinaten unveränderlich sind, werden bei jeder Wahl von Render und Anzeigeauflösung immer für gleiche Abtastungspositionen der gleiche Wert berechnet. Im Bezug auf die Abbildung liegen also bei Definition der Linien durch Polygone nur die roten Abtastungspositionen innerhalb der Linien.

In dem Fall von dünnen, stark gehäuften Strukturen, wie es bei Parallelen Koordinaten oft zu finden ist, gehen durch diesen Effekt viele Details verloren. Da in dieser Arbeit die ursprüngliche Bildqualität möglichst gut erhalten bleiben soll, sind veränderliche Liniendicken in dieser Form nicht akzeptabel. Die Unterschiede dieser Liniendicken sind in Abb. 4.7 für einen praktischen Fall erkennbar. Aufgrund von diesem Effekt wurden für alle weiteren Erwägungen Polygonlinien verwendet, solange nicht explizit von Bresenham-Linien gesprochen wird.



(a) Bild mit von Bresenham's Algorithmus gezeichneten Linien. Die Linien in diesem Bild sind genau ein Pixel breit, da Renderauflösung und Anzeigeauflösung identisch sind.



(b) Bild mit von Bresenham's Algorithmus gezeichneten Linien. Bild wurde mit 4xAR erstellt. Da dieses Bild mit einer verringerten Renderauflösung erstellt wurde, sind Linien visuell dicker.

Abbildung 4.7: Vergleich von Bresenham's Linien die durch 4xAR gezeichnet wurden mit einem Bild in voller Renderauflösung. Die Dicke der Linien ist erkennbar unterschiedlich.

5 Ergebnisse

5.1 Bildqualität ohne Bewegung

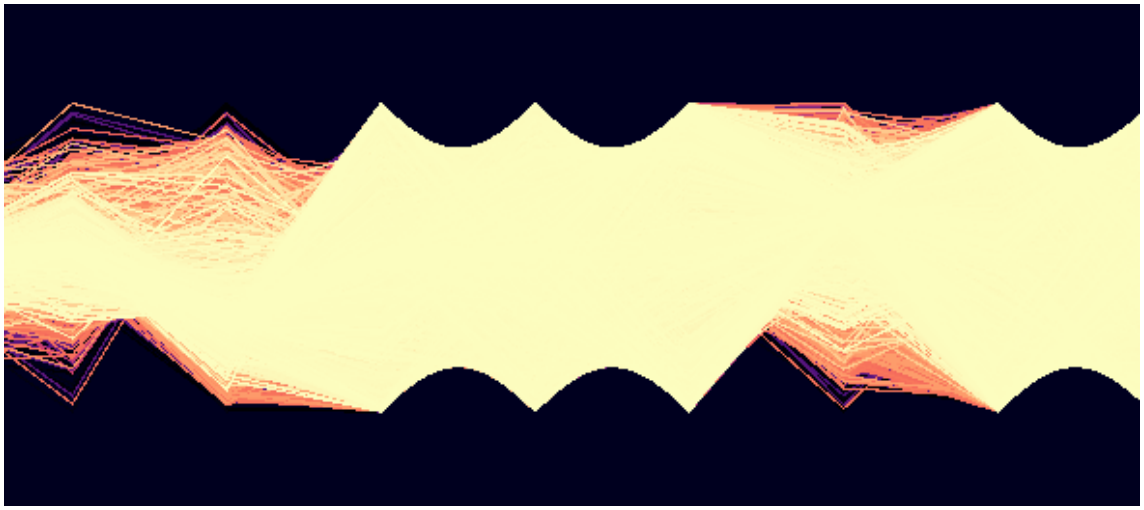
In diesem Abschnitt wird auf die Bildqualität bei Frames eingegangen, in denen sich die Szene nach einer Anzahl von Frames, die zur vollständigen Rekonstruktion benötigt werden, nicht bewegt hat. Hiermit werden keine Fehler durch Bewegung der Szene und fehlerhaften Ausgleich in das Bild eingeführt.

5.1.1 MS-CBR

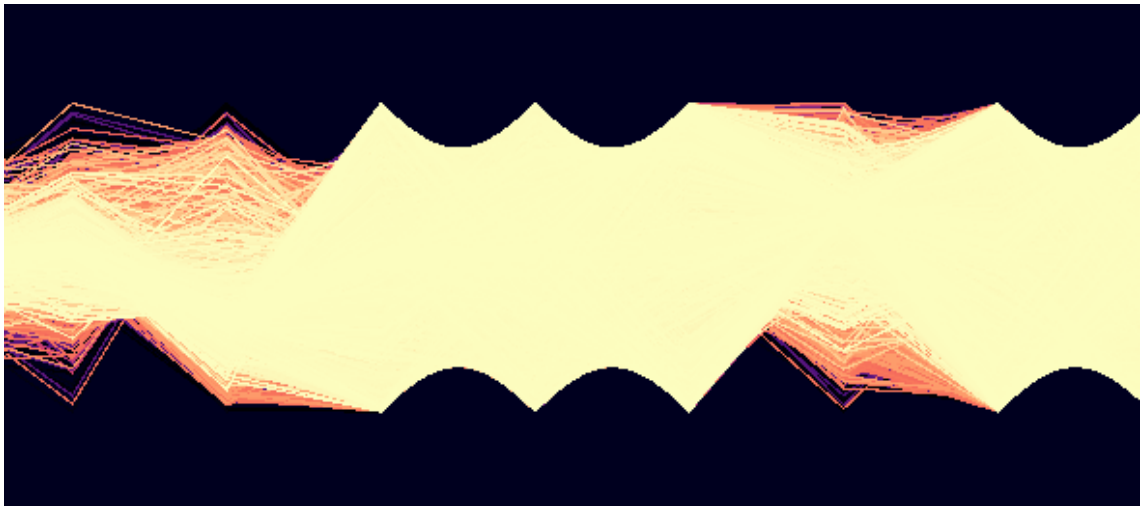
Dieses Verfahren tastet den Mittelpunkt jedes Anzeigepixels, aufgeteilt auf einen Zeitraum von zwei Frames ab. Die Abtastungspositionen sind bei idealisierter Berechnung der Kameraverschiebung hier exakt auf den Pixelmitten der gewünschten Anzeige. Im realen Fall wurde jedoch festgestellt, dass es vereinzelt zu Abweichungen kommen kann, was auf Ungenauigkeiten bei Fließkommaoperationen in den Matrixmultiplikationen, die die Kameraverschiebung umsetzen, zurückgeführt werden kann. Bei dichten Linienhaufen oder bei Abtastungspositionen sehr nah am Rand einer Linie führen diese Ungenauigkeiten dazu, dass andere Linien oder sogar der Hintergrund der Visualisierung, anstatt der gewünschten Linie, abgetastet wird, was zu einer Abweichung des Farbwertes des Anzeigepixels führt. Da dieses Verfahren keinen Gebrauch von Anti-Aliasing macht, sind weiterhin treppchenförmige Strukturen an den Linien klar erkennbar. In Abb. 5.1 werden ein nicht-amortisiertes Bild, ein mit MS-CBR erstelltes Bild und Fehler-Bild hervorgehoben. Dieses Fehlerbild wird durch pixelweisen Vergleich des nicht-amortisierten Bildes und des mit MS-CBR erstellten Bildes erzeugt. Pixel deren Werte voneinander Abweichen werden dann rot eingefärbt. Die Abbildung zeigt zwar vereinzelte Abweichungen, diese sind im subjektiven Bildeindruck nur sehr schwer zu identifizieren, wodurch man diese oft nicht ohne Hilfsmittel von nicht-amortisierten Bildern unterscheiden kann.

5.1.2 4xAR

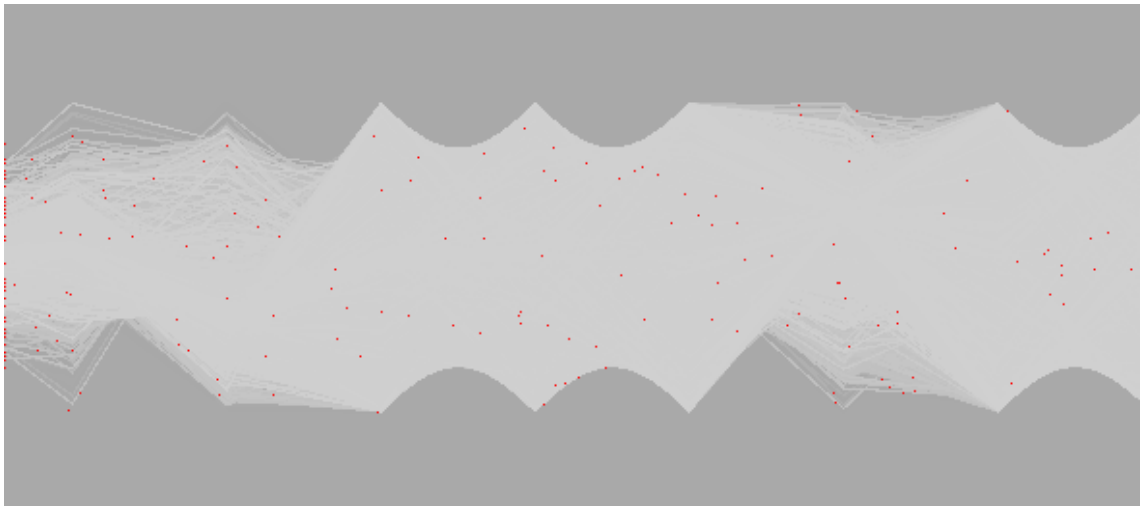
Dieses Verfahren tastet den Mittelpunkt jedes Anzeigepixels, aufgeteilt auf vier Bilder ab. Wie auch bei MS-CBR, sollte bei idealisierter Berechnung der Kameraverschiebung das rekonstruierte Bild und das nicht-amortisierte Bild identisch sein. Auch hier können bei der tatsächlichen Ausführung des Algorithmus vereinzelt Abweichungen festgestellt werden. Wie auch bei MS-CBR wird kein Gebrauch von Anti-Aliasing gemacht, was ebenfalls weiterhin treppchenförmige Linien abbildet. Ein Vergleich von nicht-amortisiertem Bild mit dem durch 4xAR erstellten Bildes sowie ein Fehlerbild sind in Abb. 5.2 abgebildet. Auch hier sind in dem Fehlerbild vereinzelte Abweichungen erkennbar, diese fallen allerdings im subjektiven Bildeindruck nicht mehr auf.



(a) Nicht-amortisiertes Bild

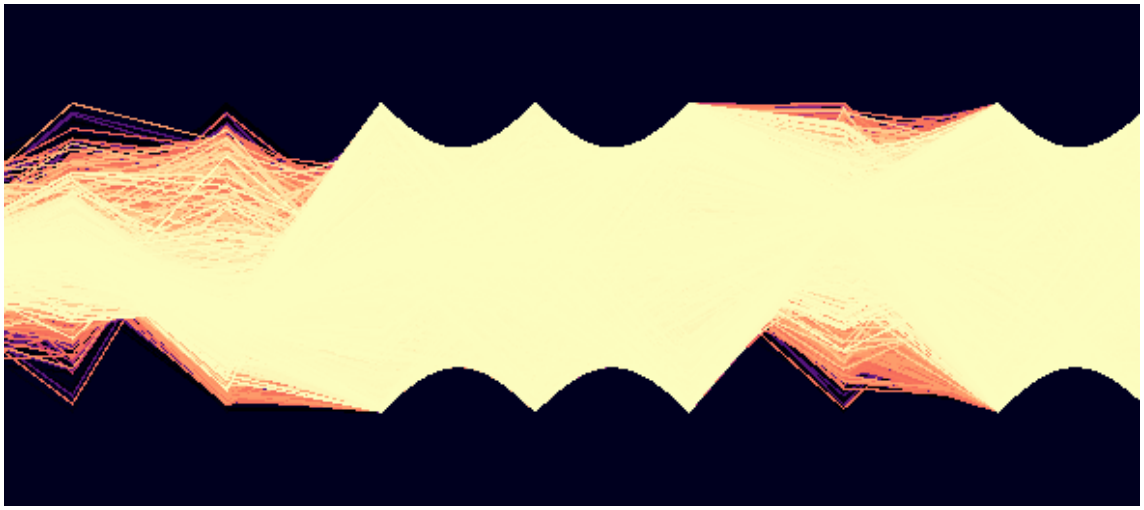


(b) Mit MS-CBR erstelltes Bild

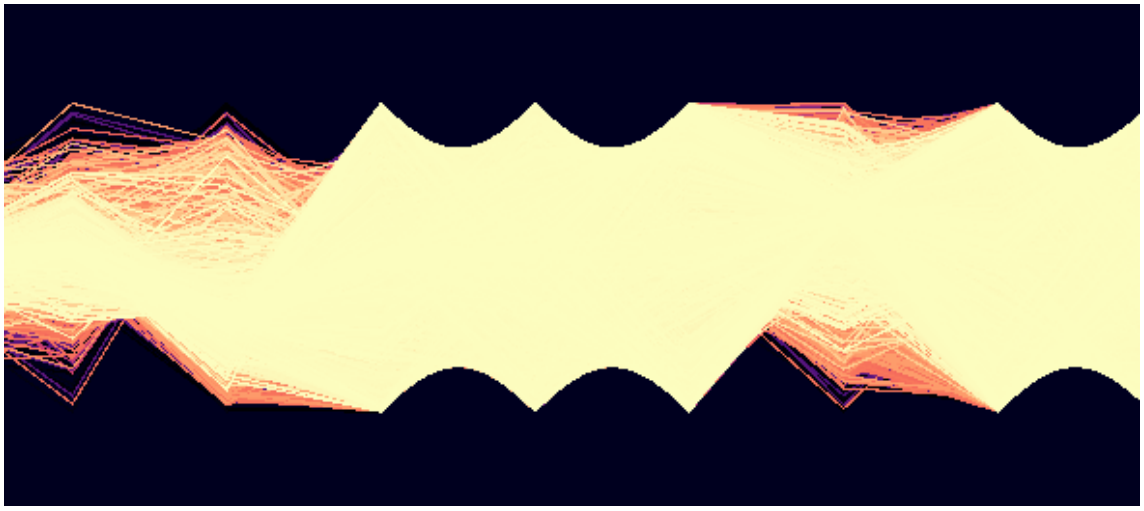


(c) Pixelabweichungen in rot hervorgehoben.

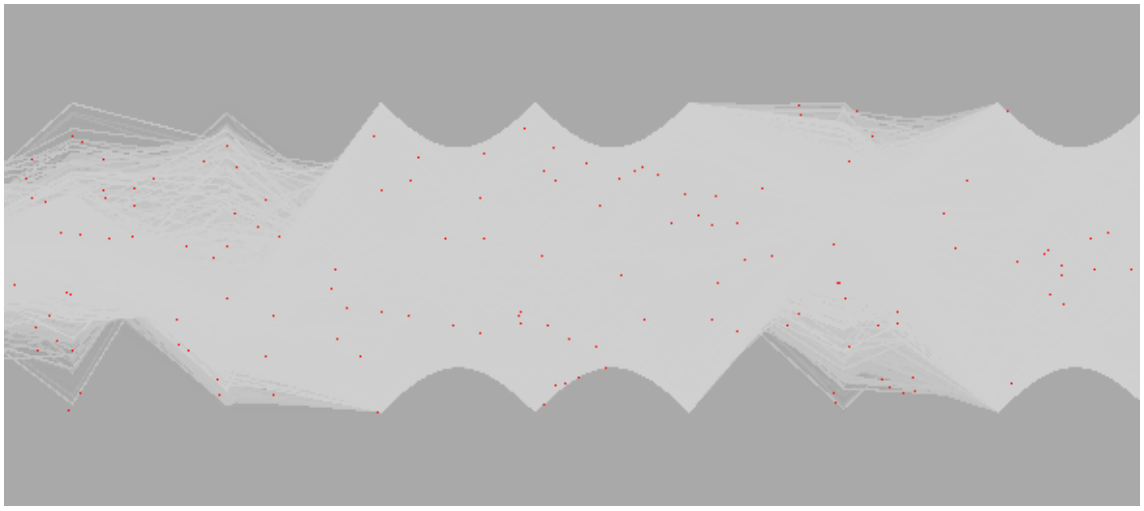
Abbildung 5.1: Vergleich von MS-CBR mit der nicht-amortisierten Basis für dieses Bild. In Unterabbildung 3 sind in rot vereinzelte Abweichungen erkennbar.



(a) Nicht-amortisiertes Bild



(b) Mit 4xAR erstelltes Bild



(c) Abweichende Pixel in rot hervorgehoben.

Abbildung 5.2: Vergleich von 4xAR mit der nicht-amortisierten Basis für dieses Bild. In Unterabbildung 3 sind in rot vereinzelte Abweichungen erkennbar.

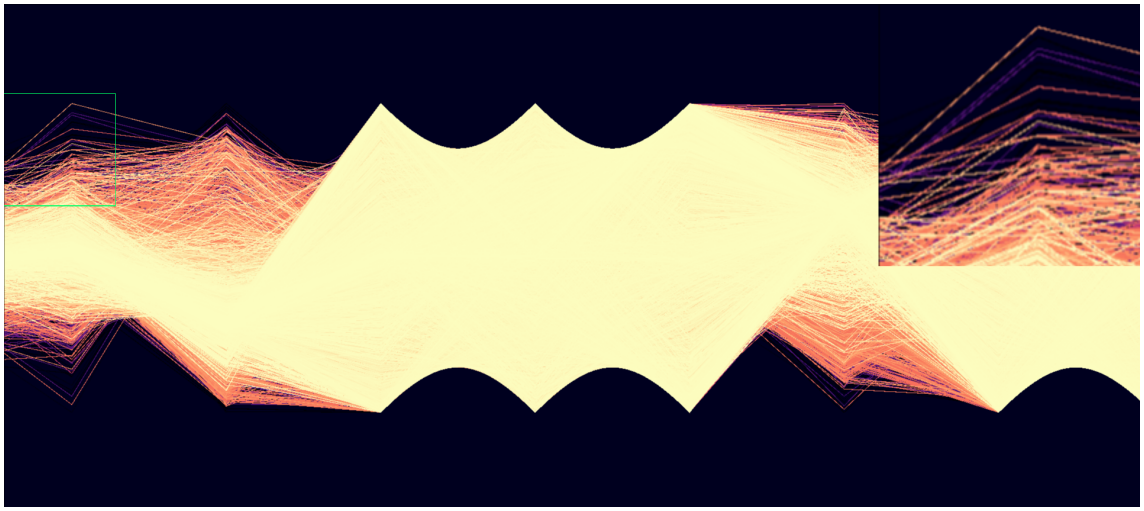


Abbildung 5.3: Mit 4xAR-C erstelltes Bild.

5.1.3 4xAR-C

Dieses Verfahren tastet die Ecken der Pixel ab und rekonstruiert der Wert eines Pixels durch Mittelung der Abtastungen auf den jeweiligen Ecken. Hier ist eine Abweichung von nicht-amortisierten Bildern gewünscht, um Kantenglättung zu erreichen. Ein so erstelltes Bild ist in Abb. 5.3 abgebildet. Hier ist erkennbar, dass die so versuchte Kantenglättung das Bild im allgemeinen verschwommen darstellt und viele Linien dunkler erscheinen, auch wenn Kanten etwas weniger klar erkennbar sind. Verglichen mit nicht-amortisierten Bildern, ist es schwieriger Details auszumachen. Zudem sind weiterhin Treppchen in den Linien erkennbar, was untermauert, dass Anti-Aliasing durch die hier Wahl dieser Samplingpositionen bei Strukturen wie Linien kein zufriedenstellendes Ergebnis liefert.

5.1.4 SS-AR

In diesem Ansatz wird durch mehrfaches Abtasten jedes Pixels Supersampling eingeführt. Auch hier weichen die Abtastungspositionen von den Pixelmitten des Anzeigebildes ab. Ein Ergebnisbild dieses Verfahrens ist in Abb. 5.4 abgebildet. Durch hohes Supersampling des Bildes entsteht, wie gewünscht, Anti-Aliasing. Kanten sind im Vergeleich zu MS-CBR und 4xAR deutlich weicher, ohne Details zu verlieren, wie es bei 4xAR-C der Fall war. Dies hat allerdings den Nebeneffekt, dass die vorher sichtbaren Treppchen nun eher als Knoten in Erscheinung treten, was durch steigende und fallende Helligkeit von Linienabschnitten entsteht.

5.2 Bildqualität bei Bewegung

Wie in den vorigen Kapiteln beschrieben, kann es bei Bewegung der Szene in Verbindung mit amortisiertem Rendern zu starken Ghosting-Artefakten kommen. Ein Beispiel für solche Ghosting-Artefakte, anhand von unausgeglichener Bewegung in einem Bild, welches mit MS-CBR erzeugt

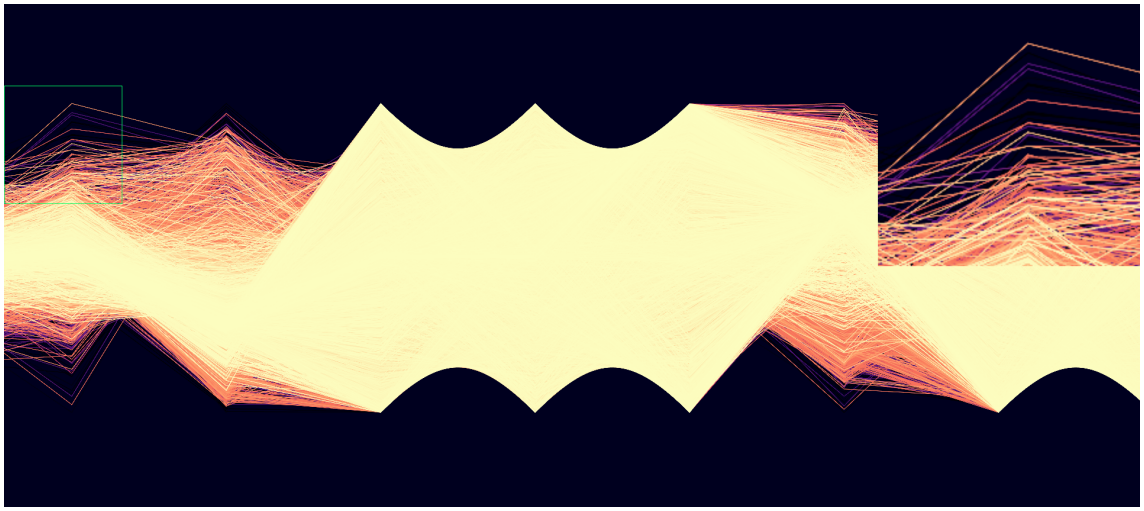


Abbildung 5.4: Ein mit SS-AR erstelltes Bild.

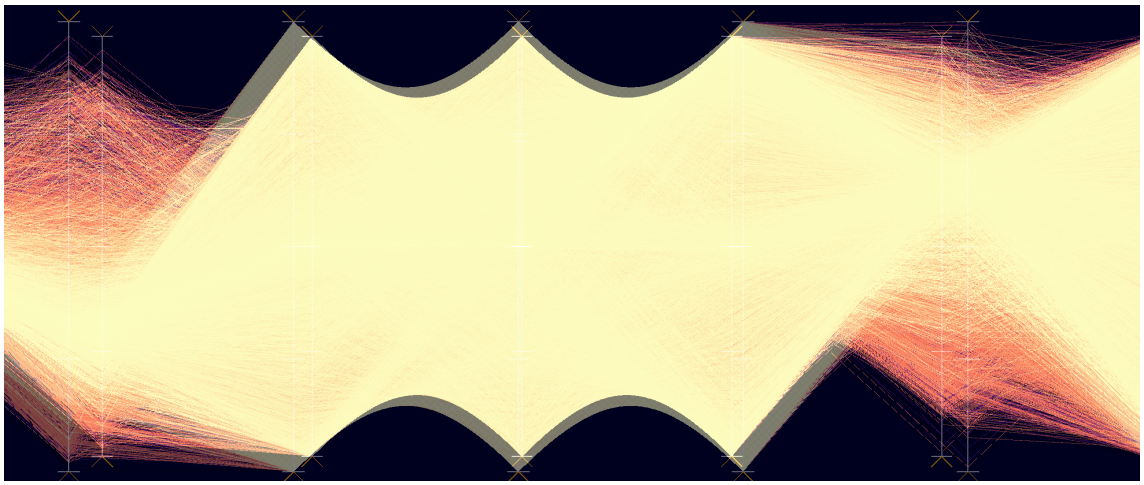


Abbildung 5.5: Ein mit MS-CBR erzeugtes Bild, in dem keine Reprojetion zum Ausgleich der Bewegung vorgenommen wurde. Ghosting-Artefakte in Form von zwei gleichzeitig erkennbaren und unterscheidbaren Zeitpunkten der Skalierung.

wurde, ist in Abb. 5.5 abgebildet. In diesem Bild sind zwei verschiedene Zeitschritte gleichzeitig klar erkennbar. Durch nachempfinden der Bewegung und Projizieren der Abtastungen aus alten Zeitschritten an die aktuelle Position wird versucht diese Fehler auszugleichen. Durch diese Reprojektion werden jedoch im allgemeinen Fehler in das Bild eingeführt, da oft eine perfekte Abbildung auf einen Anzeigepixel nach der Bewegung nicht mehr möglich ist. Dies ist insbesondere der Fall, wenn die Position einer Abtastung nach dieser Reprojektion genau auf der Position einer Abtastung eines anderen Zeitschrittes liegt. So verschobene Pixel können nun nicht mehr eindeutig den Anzeigepixeln zugewiesen werden und es ist nicht mehr klar, welchen Wert ein solcher Pixel annimmt. Hierbei können zwischen den Fehlern bei Translation und Skalierung der Szene unterschieden werden.
Bei Translation der Szene entstehen an dem Rand der Szene Lücken, welche in alten Bildern noch nicht sichtbar waren. Diese Lücken werden hier akzeptiert und nicht gefüllt. Hierdurch entsteht ein

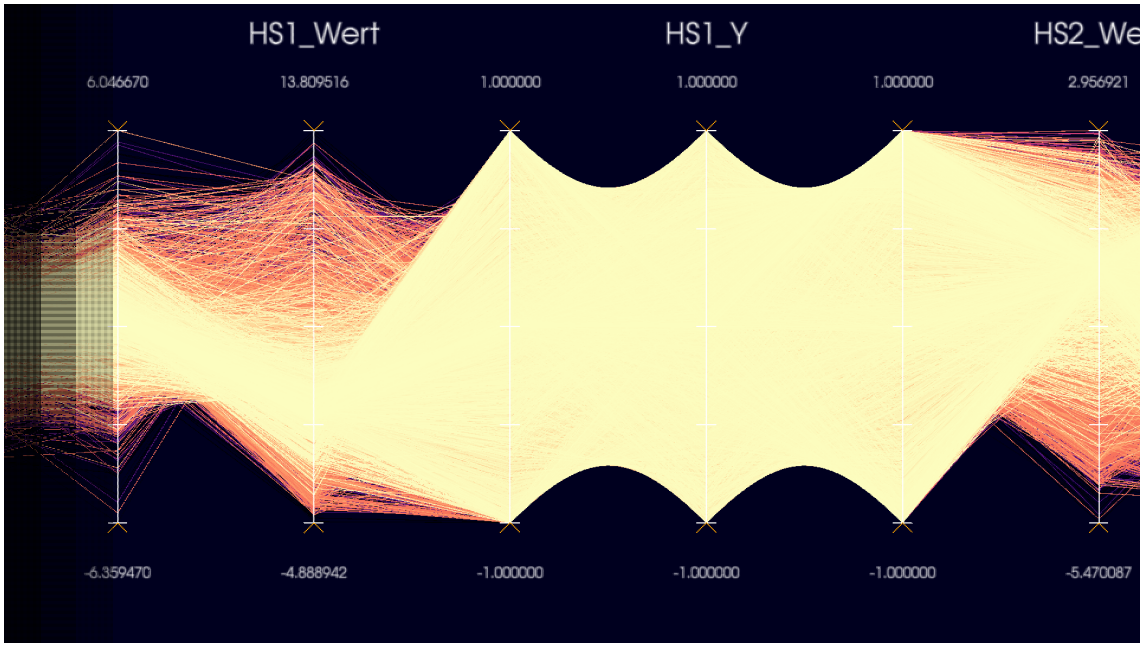
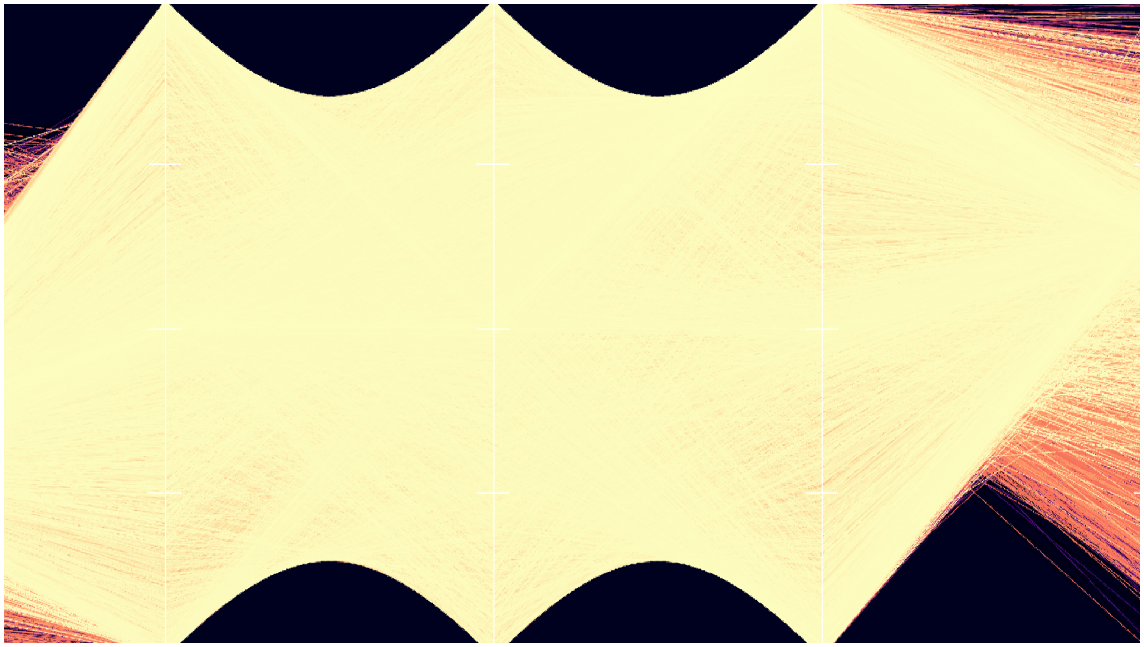
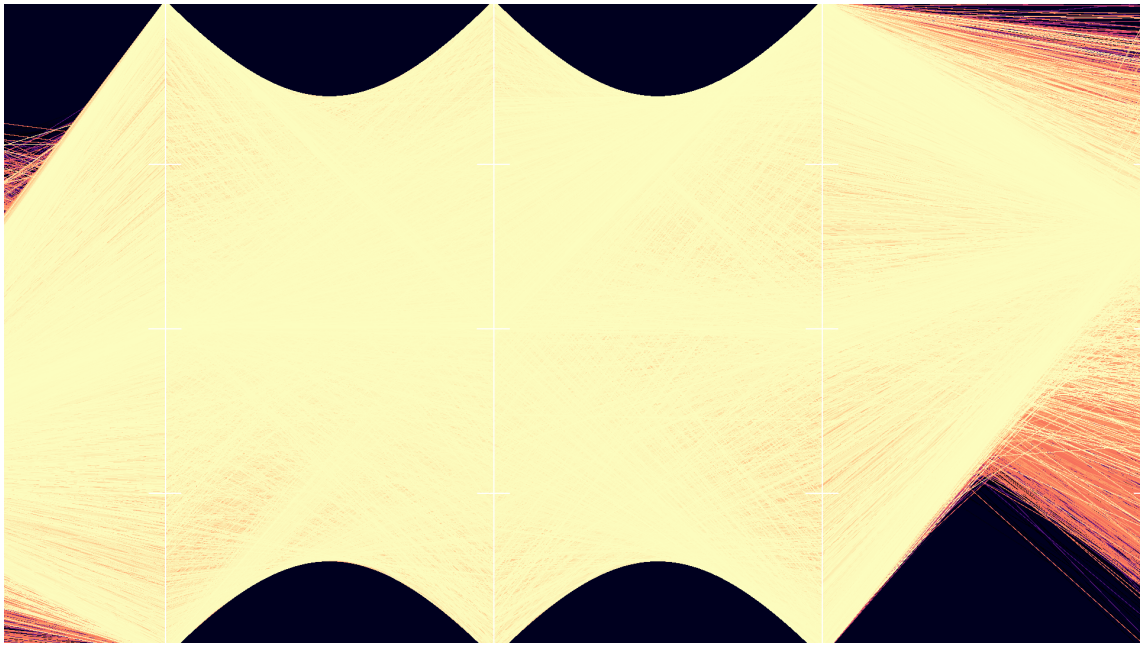


Abbildung 5.6: Ein mit MS-CBR erstelltes Bild, welches keinen Ausgleich für die Translation der Szene macht. Ghosting-Artefakte sind klar erkennbar. Am linken Rand sind Lücken an Stellen, die zuvor nicht sichtbar waren zu sehen.

Schatten am Rand des Bildes. Auch abgesehen von diesen Fehlern können bei der Reprojektion Abtastungen nun fehlerhaft Anzeigepixeln zugeordnet werden. Diese Fehler äußern sich in körnigen Linien. Wird die Szene skaliert, also die Zoomeinstellung der Kamera geändert, werden alte Abtastungspositionen entsprechend mitskaliert. Bei dieser Reprojektion nehmen die erhobenen Abtastungen plötzlich eine größere bzw. kleine Fläche ein, als dies zum Zeitpunkt der Abtastung der Fall war. Bei der Einfärbung der Anzeigepixel wird eine Abtastung nun möglicherweise mehreren oder keinen Anzeigepixeln zugewiesen. Somit entsteht ein körnigerer Bildeindruck, in dem zahlreiche fehlerhafte Pixel erkennbar sind. Ein solches Bild nach Reprojektion wird in Abb.5.7 mit einem Bild der gleichen Kameraposition verglichen, in dem das Bild jedoch lange genug nicht bewegt wurde und somit keine Reprojektionsfehler auftreten. Eine Gegenüberstellung von vergrößerten Bildausschnitten ist in Abb. 5.8 zu finden. Das für diese Bilder verwendete Beschleunigungsverfahren ist MS-CBR. In der Praxis fallen diese Fehler subjektiv jedoch kaum auf. Diese Fehler sind nur sichtbar, während die Szene bewegt wird und wenige Bilder dannach, bis dann diese reprojizierten Abtastungen neu ermittelt wurden. Selbst auf den zuvor beschriebenen Abbildungen ist das fehlerfreie Bild klar in den fehlerbehafteten Bildern wiederzuerkennen. Der so vorgenommene Ausgleich von Bewegung liefert also im Vergleich mit den sonst auftretenden Ghosting-Artefakten gute Ergebnisse und verbessert den Bildeindruck bei und nach Bewegungen der Szene signifikant.

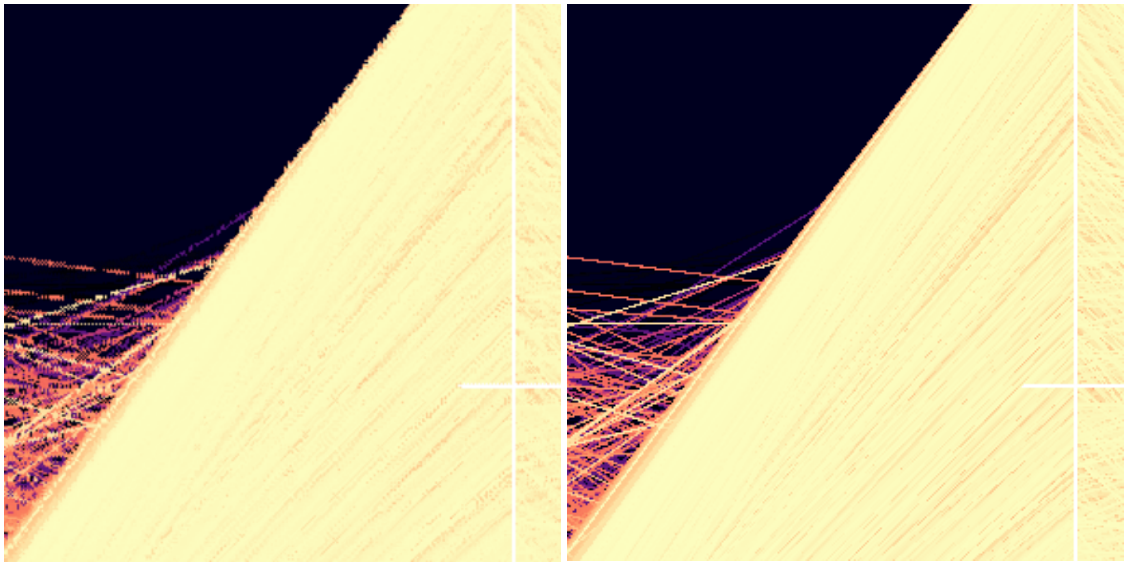


(a) Ein mit MS-CBR erstelltes Bild, nach Reprojektion. Abbildungsfehler sind in Form von königen Kanten klar erkennbar.



(b) Ein mit MS-CBR erstelltes Bild, welches keine reprojizierten Pixel hat. Hier sind keine Abbildungsfehler vorhanden.

Abbildung 5.7: Gegenüberstellung eines Bildes mit Reprojektionsfehlern bei Skalierung des Bildes mit einem Bild ohne Reprojektionsfehlern. Der verwendete Beschleunigungsansatz ist MS-CBR.



(a) Ein vergrößerter Bildausschnitt eines mit MS-CBR erstellten Bildes. Abbildungsfehler in Form von körnigen Kanten klar erkennbar.

(b) Ein vergrößerter Bildausschnitt eines mit MS-CBR erstellten Bildes. Abbildungsfehler in Form von körnigen Kanten klar erkennbar.

Abbildung 5.8: Gegenüberstellung von vergrößerten Bildausschnitten, welche mit MS-CBR erstellt wurden. Reprojektionsfehler in Unterabbildung a klar erkennbar.

Gerät	Datensatz	Auflösung	Kameraeinstellung	Beschleunigungsansatz
1060GTX	Tropfen	480p	Voll	Basis
2070S	Balken	720p	Nah	MS-CBR
Titan Xp	Klima	1080p		4xAR
Titan RTX		1440p		4xAR-C
Radeon VII		2140p		SS-AR
Vega56				

Tabelle 5.1: Mögliche Messvariablen

5.3 Methodik zur Performanzbestimmung

Um die Performanz und im besonderen die Beschleunigung des Rendervorgangs zu untersuchen wurden Messungen mit wählbaren Messvariablen durchgeführt. Ein Messdurchlauf ist also definiert durch Variablen: Gerät, Datensatz, Kameraeinstellung, Auflösung und Beschleunigungsverfahren. Eine Tabelle mit allen möglichen Messvariablen wird in Tabelle 5.1 aufgelistet. Bei jeder Messung wird für jede Spalte dieser Tabelle genau ein Wert gewählt. In der tatsächlichen Messung wird über einen Zeitraum von 1000 Bildern gerendert und für jedes erstellte Bild die benötigte Renderzeit aufgezeichnet. Aus diesen 1000 Zeitwerten wird dann zunächst ein Durchschnitt gebildet. Für die Performanz spielen dann zwei Größen eine besondere Rolle. Zum einen ist die durchschnittliche Renderzeit wichtig, da diese der Antwortzeit des Programms entspricht. Zum anderen ist die Beschleunigung die ein Verfahren gegenüber der Messbasis dieses Durchgangs erreicht wichtig,

Datensatz	Zeilen	Spalten
Tropfen	1010000	16
Balken	160105	22
Klima	672	8

Tabelle 5.2: Die Größe der Datensätze getrennt nach Anzahl der Zeilen und Spalten.

um Vergleichbarkeit der Messdurchgänge zu gewährleisten. Zudem können so Aussagen über Einflussfaktoren für die Effektivität der Verfahren, also der erreichten Beschleunigung, getätigt werden. In diesem Kontext ist die Messbasis der Durchgang, der in allen Variablen, außer dem verwendeten Beschleunigungsverfahren, übereinstimmt. Eine Messbasis bezeichnet immer einen Durchgang, der ohne Gebrauch von Beschleunigungsverfahren erstellt wurde. Die erreichte Beschleunigung ist damit der Faktor, um den die benötigte Renderzeit reduziert werden kann. Dies ist gleichbedeutend mit der Erhöhung an Bildern, die pro Sekunde erzeugt werden können.

5.3.1 Datensätze

Es wurden Messungen auf drei Datensätzen durchgeführt:

1. „Tropfen“: Simulation von Tropfen, die aus einer Düse ausgeworfen werden. Es wurden für 101 Zeitschritte für jeden der ca. 10000 entstehenden Tropfen verschiedene physikalische Eigenschaften berechnet.
2. „Balken“: Simulation durch Finit-Element-Methode von Spannungen, die in einem Betonbalken mit eingesetzten Fluidaktoren bei statischer Last entstehen. Jeder Datenpunkt umfasst einen Knoten der Finit-Element-Simulation und erfasst physikalische Eigenschaften, wie Spannungsvekotoren und Spannungstensoren. Dieser Datensatz beschreibt nur einen Zeitschritt.
3. „Klima“: Simulation eines modellprädiktivem Regelungssystems zur Klimatisierung eines Raumes. Jeder Datenpunkt enthält verschiedene Sensorgrößen und Regelungszustände.

Die Größe der Datensätze ist in Tabelle 5.2, getrennt nach Zeilen und Spalten, aufgeführt. Für diese Datensätze entsprechen die Zeilen den erhobenen Datenpunkten und die Spalten den Dimensionen je Datenpunkt.

5.3.2 Geräte

Die hier verwendeten Geräte zur Messung sind in Tabelle 5.3 aufgelistet. Im Folgenden werden Geräte nur durch ihre GPU identifiziert, da dies die wichtigste Eigenschaft der Geräte für die Performanz darstellt.

GPU	CPU	Arbeitsspeicher
Nvidia GeForce GTX 1060	Intel Core i7-4770	32 GB
Nvidia GeForce GTX 2070 Super	AMD Ryzen 9 3900X	32 GB
Nvidia Titan Xp	Intel Core i9-7900X	64 GB
Nvidia Titan RTX	Intel Core i7-9700K	64 GB
AMD Radeon VII	Intel Core i7-9700K	32 GB
AMD Vega 56	AMD Ryzen 7 3700X,	32 GB

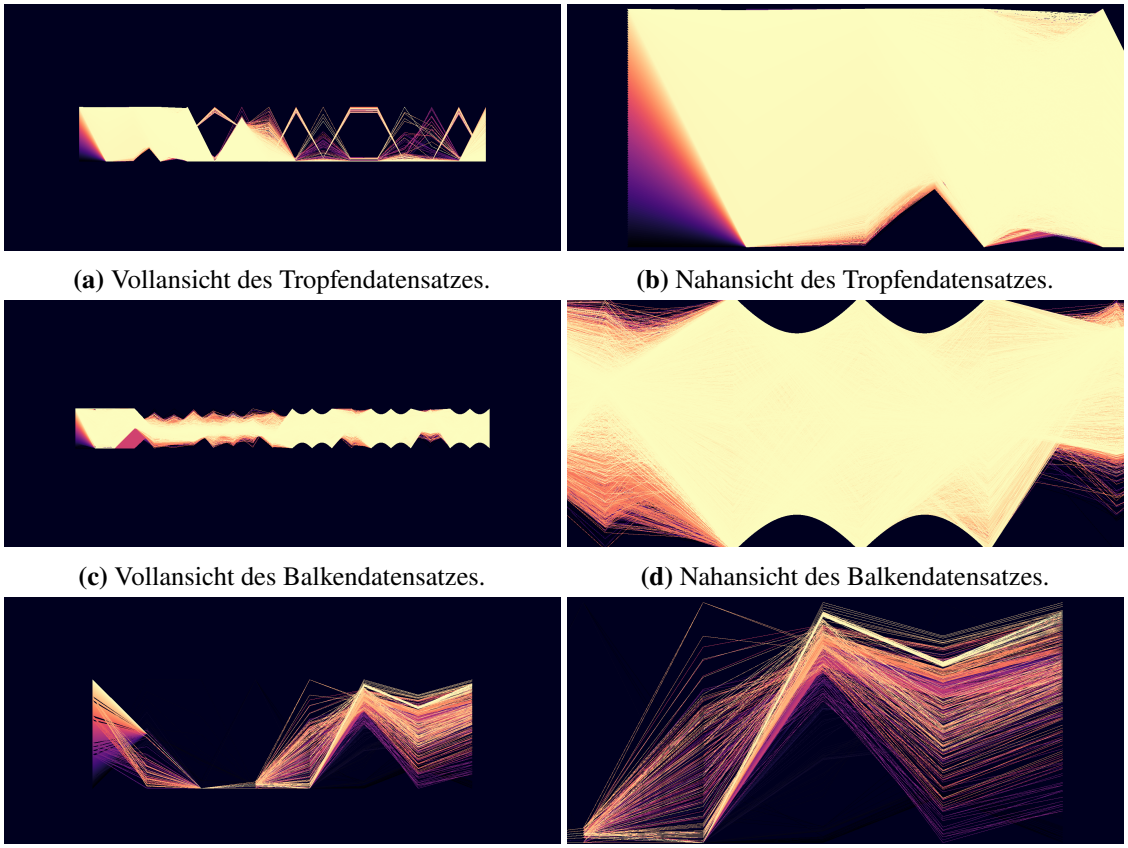
Tabelle 5.3: Die wichtigsten Eigenschaften der zur Messung verwendeten Geräte.

5.3.3 Kameraeinstellungen

Um den Einfluss der Kameraposition abzuschätzen wurden für jeden Datensatz zwei verschiedene Kamerapositionen festgelegt. Die Vollansicht wurde gewählt, da es oft gewünscht ist, den kompletten Datensatz zu sehen, um verlaufe von einzelnen Linien durch alle Dimensionen zu analysieren oder Muster in Linienhaufen zu untersuchen.
Die Nahansicht wurde zum einen gewählt, da hier die Fläche die von Linien auf der Anzeige eingenommen ist höher als bei der Vollansicht ist und zum anderen da bei Analyse eventuell nur wenige Achsen interessant sein könnten und diese dann detailierter betrachetet werden möchten. Bilder der jeweiligen Kameraeinstellungen sind in Abb. 5.9 abgebildet.

5.4 Effektivität der Ansätze

Im Folgenden wird die Beschleunigung des Rendervorgangs mithilfe der zuvor beschriebenen Verfahren untersucht. Zunächst wird die allgemeine Effektivität der Verfahren betrachtet. Zu diesem Zweck wird die Verteilung Messewerte in Abb. 5.10 in Boxplots abgebildet, bei denen auch die einzelnen Datenpunkt zufällig verteilt sind. Die Mediane der Beschleunigungsmessungen liegen zwar erkennbar über 1, jedoch sind bei jedem Ansatz auch Fälle erkennbar, deren Beschleunigung unter 1 liegt und damit die Programmausführung sogar verlangsamen kann. Zudem sind Unterschiede zwischen den Ansätzen erkennbar. Der Ansatz MS-CBR erzielt im Median die geringste Beschleunigung, während die Ansätze 4xAR und 4xAR-C sowohl im Median, als auch im Maximum höhere Beschleunigung erreichen. Der Ansatz SS-AR erzielt zwar weiterhin durchaus gute Beschleunigungen, die Minimalwerte die bei diesem Ansatz erreicht werden, sind jedoch nahe 0, was einem beinahe völligen Stopp der Programmausführung gleichkommt. Weiter ist erkennbar, dass Ansätze 4xAR, 4xAR-C und SS-AR besonders nach oben eine deutliche Streuung aufweisen, während Ansatz MS-CBR vergleichsweise eng verteilt ist. Im Allgemeinen ist ersichtlich, dass jeder der Vorgeschlagenen Ansätzen eine Beschleunigung der Programmausführung, gemessen an der Zeit zur Erstellung eines Bildes, erreichen kann. In den folgenden Abschnitten werden Einfluss- und Skalierungsfaktoren genauer beleuchtet. Sofern nicht explizit anders beschrieben, beziehen sich die folgenden Abschnitte auf Polygonlinien.



(a) Vollansicht des Tropfendatensatzes.

(b) Nahansicht des Tropfendatensatzes.

(c) Vollansicht des Balkendatensatzes.

(d) Nahansicht des Balkendatensatzes.

(e) Vollansicht des Klimadatensatzes.

(f) Nahansicht des Klimadatensatzes.

Abbildung 5.9: Gewählte Kamerapositionen für die Messungen, nach Datensatz

5.5 Skalierung mit Datensatzgröße

In Abb. 5.11 wird die durchschnittliche Beschleunigung, gruppiert nach Datensatz, für jedes Verfahren abgebildet. Für den kleinsten Datensatz „Klima“ist erkennbar, dass keins der vorgeschlagenen Verfahren im Durchschnitt eine tatsächliche Beschleunigung des Rendervorgangs erreichen kann. Die Renderzeiten sind für diesen Datensatz bereits so kurz, dass die Operationen, die für die Rekonstruktion nötig sind, die verringerten Renderkosten bei der Erstellung der Linien überwiegen. Besonders das SS-AR-Verfahren verlangsamt das Programm deutlich, da der Aufwand bei der Rekonstruktion hier am höchsten ist. Die Verfahren MS-CBR, 4xAR und 4xAR-C erzielen hier sehr ähnliche Ergebnisse, da sich deren Rekonstruktionen nur geringfügig im Aufwand unterscheiden.
Bereits bei dem nächstgrößeren Datensatz, genannt "Balken", erzielen alle vorgeschlagenen Verfahren im Durchschnitt eine deutliche Beschleunigung des Renderns. Da MS-CBR im Vergleich zu den anderen vorgeschlagenen Verfahren die meisten Abtastungen im Renderschritt erfordert, kann dieser Ansatz im Durchschnitt nur die geringste Beschleunigung erreichen. Auch diese Verfahren erreicht dennoch eine Beschleunigung von ca. 1.5 und Ansätze 4xAR und 4xAR-C erzielen auch hier sehr ähnliche Ergebnisse und beschleunigen das Rendern um einen Faktor vor ca. 2. Durch den erhöhten Rekonstruktionsaufwand kann auch hier SS-AR mit einem Wert von ca 1.6 nicht die Beschleunigung der Verfahren 4xAR oder 4xAR-C erreichen.
Bei dem größten Datensatz, dem sogenannten Tropfen-Datensatz, können erneut alle Verfahren

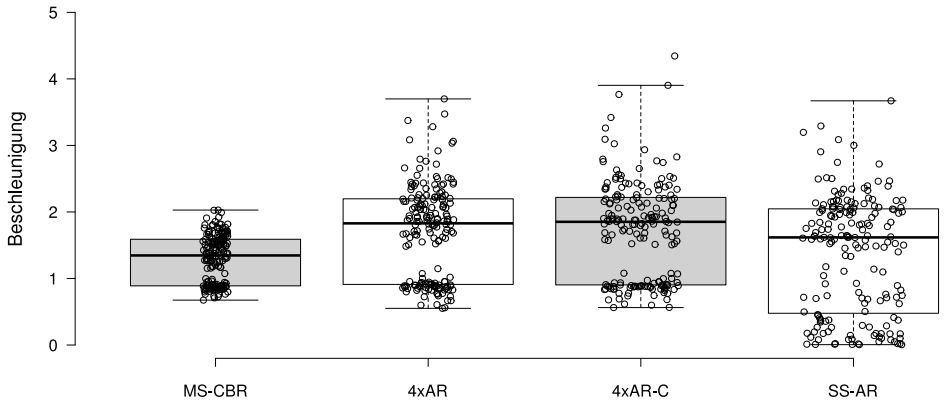


Abbildung 5.10: Verteilung der Messwerte je Beschleunigungsansatz dargestellt in Boxplots. Zusätzlich sind alle gemessenen Datenpunkte in Form von Kreisen in dem entsprechenden Boxplot sichtbar.

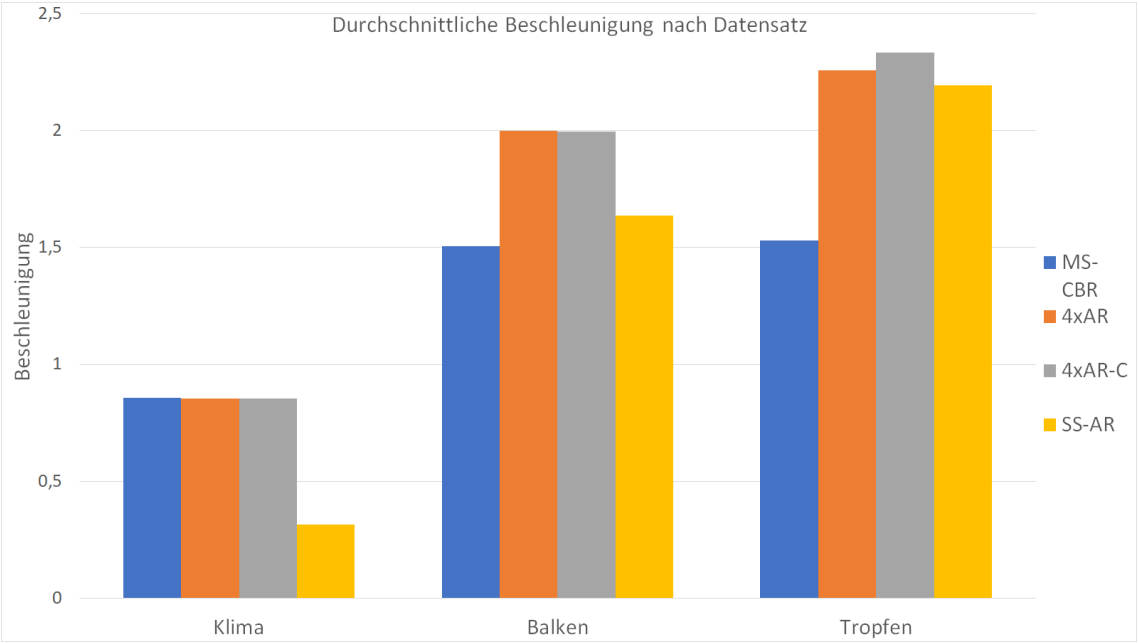


Abbildung 5.11: Durchschnittliche Beschleunigung gruppiert nach Datensatz. Jeder Balken repräsentiert einen Beschleunigungsansatz. Die Ausläufer zeigen den maximalen und minimalen Wert der mit diesem Ansatz ermittelt wurde.

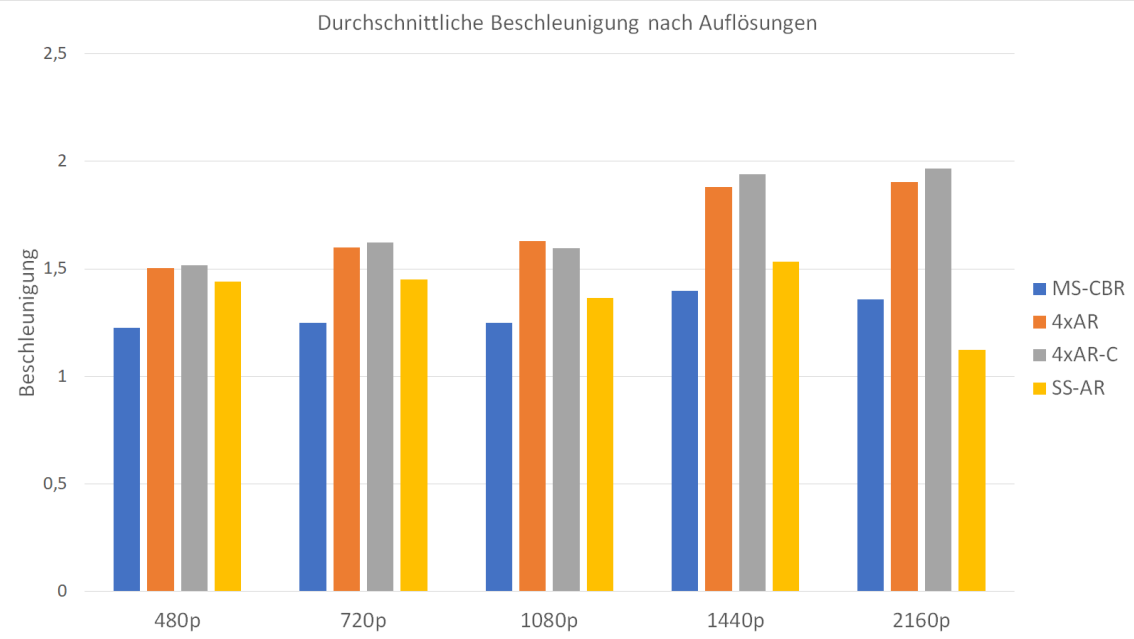


Abbildung 5.12: Im Durchschnitt erreichte Beschleunigung der vorgestellten Ansätze je getesteter Auflösung.

das Rendern beschleunigen. Die hier erreichten Beschleunigungen übertreffen für jedes Verfahren die für kleinere Datensätze beobachteten Werte. MS-CBR erreicht nur ein geringfügig besseres Ergebnis gegenüber den zuvor besprochenen Datensätzen, während die Anstäze 4xAR und 4xAR-C eine größere Verbesserung erreichen. Der Ansatz SS-AR erreicht die größte Steigerung im Vergleich zum Balken-Datensatz. Beim Rendern dieses Datensatzes rücken die hohen Rekonstruktionskosten von SS-AR im Vergleich zu der Verringerung der Renderkosten bei der Linienerstellung weiter in den Hintergrund. Durch diesen Effekt entwickelt sich die Beschleunigung durch SS-AR für große Datensätze besonders gut. Zudem ist festzustellen, dass 4xAR-C eine leichte Verbesserung gegenüber dem Ansatz 4xAR aufweist.
Im Allgemeinen ist zu beobachten, dass die erreichten Beschleunigungen mit Größe des Datensatzes steigt. Hieraus ist abzuleiten, dass die vorgeschlagenen Ansätze für besonders kleine Datensätze, deren Rendervorgang bereits ausreichend schnell abläuft, nicht empfohlen werden kann, während bei großen Datensätzen, deren Rendern vergleichsweise lange dauert, jedes der vorgeschlagenen Verfahren eine deutliche Beschleunigung erzielen kann. Für die kritischen Anwendungen, also große Datensätze mit hohen Antwortzeiten, erzielen diese Ansätze besonders gute Ergebnisse.

5.6 Skalierung mit Anzeigeauflösung

Im folgenden wird untersucht, wie die Beschleunigung der Ansätze mit der gewünschten Anzeigeauflösung zusammmenhängt. Zu diesem Zweck wurde in Abb. 5.12 die im Durchschnitt erreichte Beschleunigung, gruppiert nach Anzeigeauflösung, abgebildet. Zunächst ist feststellbar, dass im Durchschnitt jeder der vorgeschlagenen Ansätze eine Beschleunigung des Rendervorgangs erzielen kann. Die Skalierbarkeit der einzelnen Ansätze weist jedoch Unterschiede auf:

Für den Ansatz MS-CBR kann beobachtet werden, dass die Beschleunigung über die Auflösungen 480p, 720p und 1080p auf etwa 1.25 bleibt, für 1440p eine signifikante Verbesserung gegenüber den vorigen Auflösungen auf den Wert 1.40 aufweisen kann und dann für eine Auflösung von 2160p wieder einen Teil der Beschleunigung verliert und nur noch den Wert 1.36 erreicht. Verglichen mit den anderen vorgeschlagenen Verfahren benötigt MS-CBR die größte Anzahl von Abtastungen pro Bild, wodurch die niedrigste Beschleunigung zu erwarten ist. Dies äußert sich indem die mit diesem Verfahren erreichte Beschleunigung für alle Auflösungen von 4xAR und 4xAR-C übertroffen wird. Die Ansätze 4xAR und 4xAR-C erreichen bereits für die Auflösung 480p eine Beschleunigung in Höhe von etwa 1.5. Für die beiden Auflösungen 720p und 1080p kann dieser Wert noch weiter auf etwa 1.60 gesteigert werden. In den folgenden größeren Auflösungen 1440p und 2140p ist ein deutlicher Sprung auf eine Beschleunigung von ca. 1.9 zu beobachten. Damit ist also erkennbar, dass diese Verfahren mit steigenden Auflösungen ebenfalls bessere Beschleunigungswerte erreichen und für jede Auflösung durch diese Verfahren die optimale Beschleunigung erreicht wird.
Zuletzt ist festzustellen, dass der Ansatz SS-AR über die Auflösungen 480p, 720p, 1080p und 1440p zwar eine höhere Beschleunigung als MS-CBR erreichen kann, jedoch nie die Beschleunigung von 4xAR oder 4xAR-C übertreffen kann. Nur bei einer Auflösung von 2160p bricht die erreichte Beschleunigung ein und wird selbst von MS-CBR überschattet. Dies geht auf den besonders hohen Anspruch an das Speichersystem der Grafikkarten zurück, da in jedem Schritt eine sehr hohe Anzahl von Abtastungen aus vergangenen Bildern gelesen werden muss und dieser Bedarf mit der Anzeigeauflösung wächst. Es entsteht also ein Nadelöhr bei der Rekonstruktion in Speicherbedarf und -bandbreite, welches man vor Verwendung dieses Verfahrens bedenken sollte.
Anhand dieser Grafik ist also abzulesen, dass besonders Ansätze 4xAR und 4xAR-C die Antwortzeiten des Programms verbessern können.
Um das Verhalten von Ansatz SS-AR näher zu beleuchten wurden nun die getesteten Grafikkarten mit Hinblick auf die Speicherkapazitäten getrennt. Hierbei sind signifikante Unterschiede zwischen den beiden Grafikkartenherstellern Nvidia und AMD zu erkennen. In Abb. 5.13 wurde hierfür nur die Beschleunigung durch SS-AR-Ansatz im Bezug auf den Hersteller der Grafikkarte getrennt. Hierbei stellt man fest, dass die Grafikkarten von Nvidia zwar in niedrigeren Auflösungen eine etwas höhere Beschleunigung erreichen, diese jedoch bei einer Auflösung von 2160p um einen Wert von über 0.5 abfallen. Zwar ist bei Grafikkarten von AMD kein Einbruch bei einer Auflösung von 2160p zu beobachten, jedoch ist die Beschleunigung bei niedrigeren Auflösungen etwas geringer.
Der plötzliche Einbruch bei hohen Auflösungen ist durch den zusätzlichen Speicherbedarf für die Rekonstruktion zu begründen. Der Speicherbedarf von 25 Abtastungen pro Anzeigepixel erzeugt bei solchen Auflösungen eine so hohe Last, dass sich ein neues Nadelöhr für die Bilderstellung bildet. Dieses Nadelöhr ist für Nvidia-Grafikkarten aufgrund von deren vergleichsweise niedrigeren Speicherbandbreite schneller erreicht als es bei AMD-Grafikkarten der Fall ist. Im Allgemeinen ist hier festzustellen, dass der Ansatz SS-AR für speicherlimitierte Systeme keine geeignete Beschleunigung erreicht und besonders hohe Auflösungen selbst aktuelle Konsumenten-Hardware an ihre Grenzen bringt.

5.7 Einfluss der Kameraposition

Zur Untersuchung des Einflusses der Kameraposition auf die Beschleunigungen wurden zwei Einstellungen getestet. Die gewählten Positionen sind zum einen die volle Ansicht des Graphen, welche beim Zurücksetzen der Kamera durch Megamol eingestellt wird, und eine vergrößerte

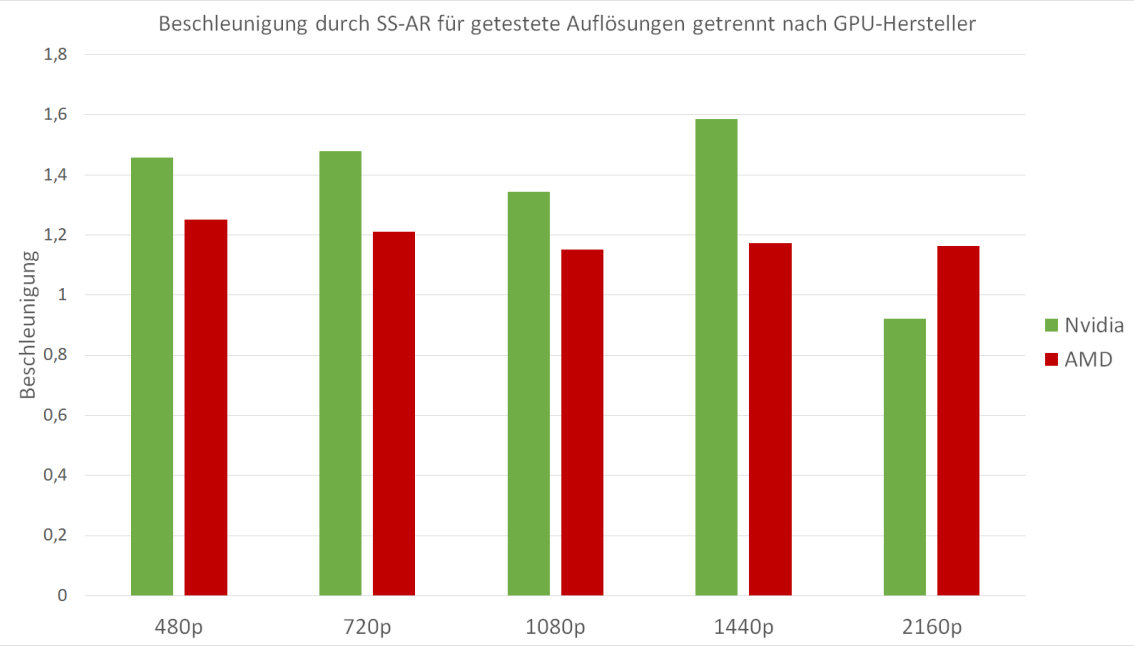


Abbildung 5.13: Gegenüberstellung der Beschleunigung durch SS-AR für Nvidia und AMD-Grafikkarten für jede Auflösung. Das entstehende Speichernadelöhr betrifft Nvidia-Grafikkarten mehr als AMD-Grafikkarten.

Ansicht, sodass der Graph eine größere Fläche der Anzeige einnimmt. In Abb. 5.14 werden die erreichten Beschleunigungen zusammen mit dem Minimum und Maximum der erreichten Beschleunigungen mit jedem Ansatz aufgeführt. Bei den gewählten Kamerapositionen unterscheiden sich die durchschnittlich erreichten Beschleunigungen nur geringfügig. Die Verteilung der Beschleunigungen ist bei der nahen Ansicht jedoch signifikant enger gestreut.

5.8 Abhängigkeit von der Grafikkarte

In den Messungen sind auch Unterschiede zwischen den getesteten Grafikkarten feststellbar. Wie in Abb. 5.15 erkennbar ist, erreichen zunächst alle getesteten Grafikkarten ähnliche Beschleunigungen für MS-CBR. Lediglich die Karten 1060GTX und 2070S fallen mit Beschleunigungswerten 1.21 und 1.24 leicht gegenüber den Karten RadeonVII und TitanXP, mit den Werten 1.37 und 1.33, ab. Hier spiegelt sich erneut ab, dass dieses Verfahren die niedrigste Streuung aufweist und damit zuverlässige Ergebnisse liefert.
Für die Verfahren 4xAR und 4xAR-C fallen die Grafikkarten RadeonVII und Vega56 deutlich gegenüber den Karten von Nvidia ab. Die beiden AMD-Grafikkarten erzielen bei diesen beiden Verfahren ähnliche Beschleunigungswerte, in Höhe von 1.50 bis 1.58, während die Nvidia-Grafikkarten 2070S, TitanXP und TitanRTX mit den Werten 1.79 bis 1.91 hier einen deutlichen Vorsprung aufbauen können. Lediglich die Nvidia-Grafikkarte 1060GTX kann solche Beschleunigungen im Durchschnitt nicht erreichen und erzielt nur Werte von 1.66 bis 1.68.
Bei Verfahren SS-AR schneidet die Grafikkarte 1060GTX im Vergleich nun auffallend schlecht ab. Hier kann diese Grafikkarte im Durchschnitt nur noch eine Beschleunigung von 1.19 erreichen, was

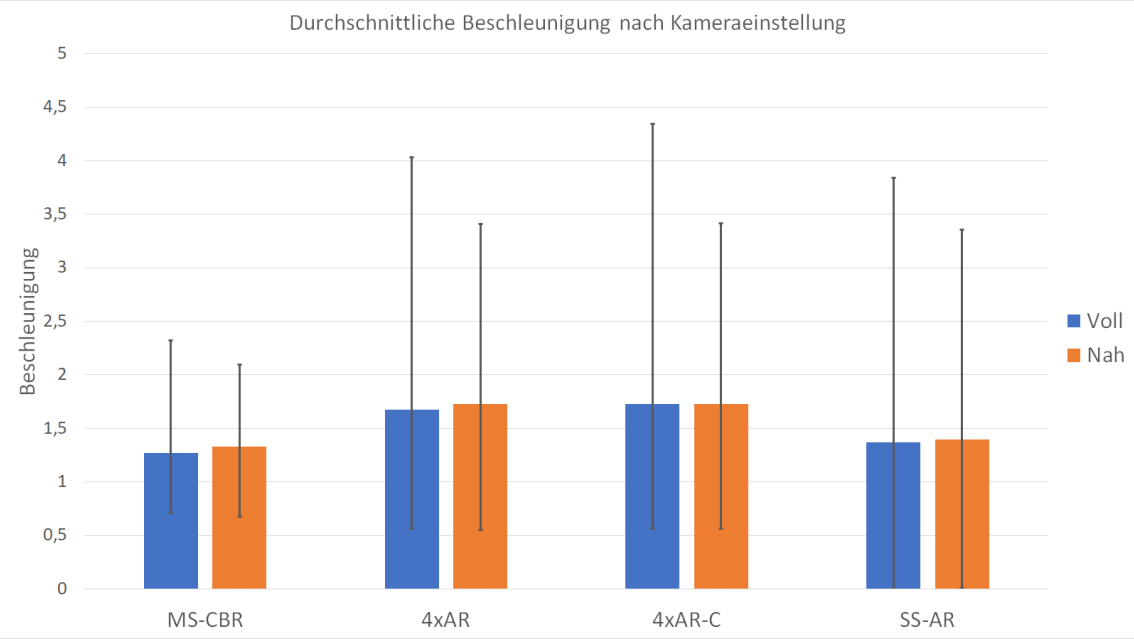


Abbildung 5.14: Beschleunigung der Ansätze in Kamerapositionen „Voll“und „Nah“. Die schwarzen Balken markieren jeweils die maximale und minimale erreichte Beschleunigung.

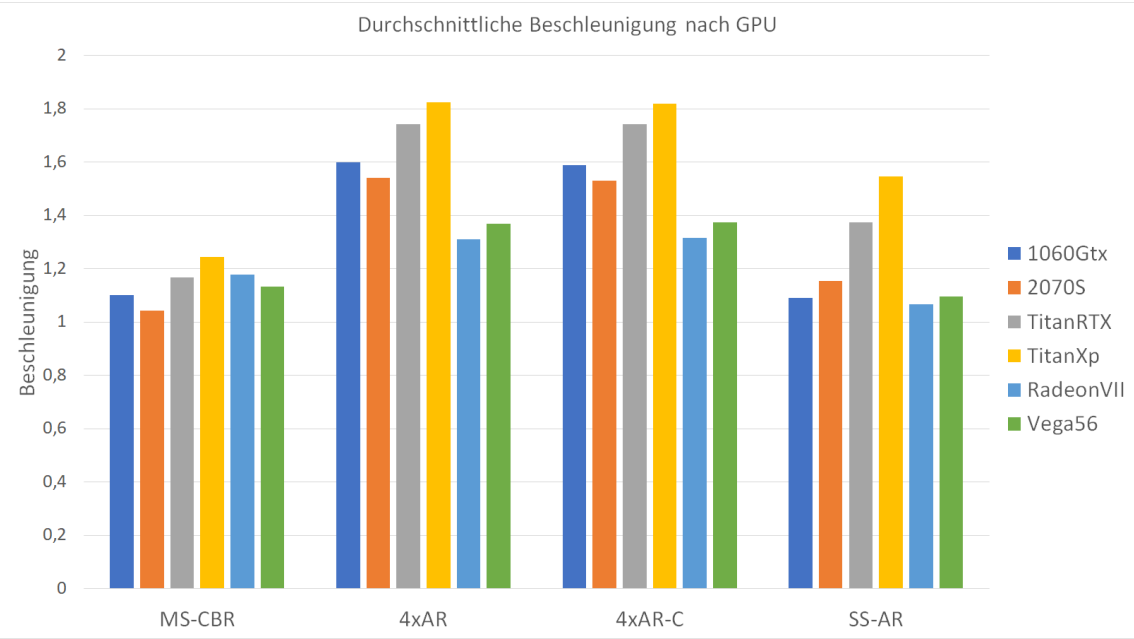


Abbildung 5.15: Durchschnittlich erreichte Beschleunigung gruppiert nach Beschleunigungsverfahren für jede getestete Grafikkarte.

auf die Limitierungen des Speichersystems der Grafikkarte zurückzuführen ist.
Ein solcher Einbruch im Vergleich zu den Verfahren 4xAR und 4xAR-C ist bei den anderen getesteten Nvidia-Grafikkarten nur in verringertem Ausmaß festzustellen. So erreichen die beiden Grafikkarten TitanXP und TitanRTX weiterhin die größten durchschnittlichen Beschleunigungswerte mit Werten von 1.52 und 1.58. Bei der Grafikkarte 2070S ist ein etwas größerer Verlust an Beschleuniugung mit einem Wert von 1.40 zu beobachten. Die beiden AMD-Grafikkarten RadeonVII und Vega56 erzielen auch hier sehr ähnliche Werte in Höhe von 1.28 und 1.31, welche jedoch niedriger sind als alle getesteten Grafikkarten von Nvidia mit der Ausnahme der 1060GTX.
Im Allgemeinen ist erkennbar, dass die AMD-Grafikkarten im Durchschnitt keine so hohen Beschleunigungen wie die getesteten Nvidia-Karten, mit Ausnahme der 1060GTX mit Verfahren SS-AR erreichen können. Dieser Vorsprung ist besonders bei den Ansätzen 4xAR und 4xAR-C signifikant.

5.9 Reaktionszeit und Aktualisierungszeit

Für die Performanz der verschiedenen Ansätze muss zusätzlich zu der Beschleunigung des Rendervorgangs auch bedacht werden, dass die Zeit ein vollständiges Anzeigebild zu erzeugen über mehrere Renderschritte verteilt wird. Dies führt zu einem Unterschied in der Reaktionszeit und der Bildaktualisierungszeit. Die Reaktionszeit des Programms hängt im Wesentlichen von der Zeit ab, die das Erstellen eines Bildes benötigt. Bei amortisiertem Rendern kann diese Zeit, durch Verteilung von Abtastungskosten auf mehrere Bilder, deutlich verringert werden. Da die Dauer ein vollständiges Anzeigebild zu aktualisieren wird jedoch durch genau diese Verteilung länger. Für 4xAR werden zum Beispiel vier Renderschritte benötigt, um die Anzeige vollständig zu aktualisieren. Die durchschnittlich benötigten Zeiten für beide Vergleichspunkte wurden in Abb. 5.16 für jedes Verfahren aufgeführt. Es ist erkennbar, dass auch wenn die Reaktionszeit von jedem Ansatz im Durchschnitt verringert werden kann, im Gegenzug jedoch die Bildaktualisierungszeit erhöht wird. Besonders Ansatz SS-AR benötigt große Mengen zusätzliche Zeit um ein Bild vollständig zu aktualisieren. Diese Zeit wurde in Abb. 5.16 abgeschnitten und beträgt tatsächlich 4870ms. Für viele praktische Anwendungen ist eine solche lange Wartezeit, um ein scharfes Bild zu erstellen, zu lang und behindert die Interaktion.
Ansatz MS-CBR erreicht zwar weniger starke Beschleunigung im Vergleich mit Ansätzen 4xAR und 4xAR-C, benötigt aber auch bedeutend weniger Zeit um ein vollständiges Bild zu rekonstruieren. Dieser Ansatz ist also geeignet für Situationen, in denen sowohl die Reaktionszeit als auch die Bildaktualisierungszeit eine Rolle spielen und ein guter Kompromiss zwischen beiden Zeiten gewünscht ist.
Die Ansätze 4xAR und 4xAR-C erzielen die größte Beschleunigung der Reaktionszeit, erhöhen aber auch die Bildaktualsierungszeit um einen signifikanten Faktor von 4. Diese Ansätze eignen sich daher besonders gut, wenn eine gute Reaktionszeit eine größere Rolle spielt als die Bildaktualisierungszeit.

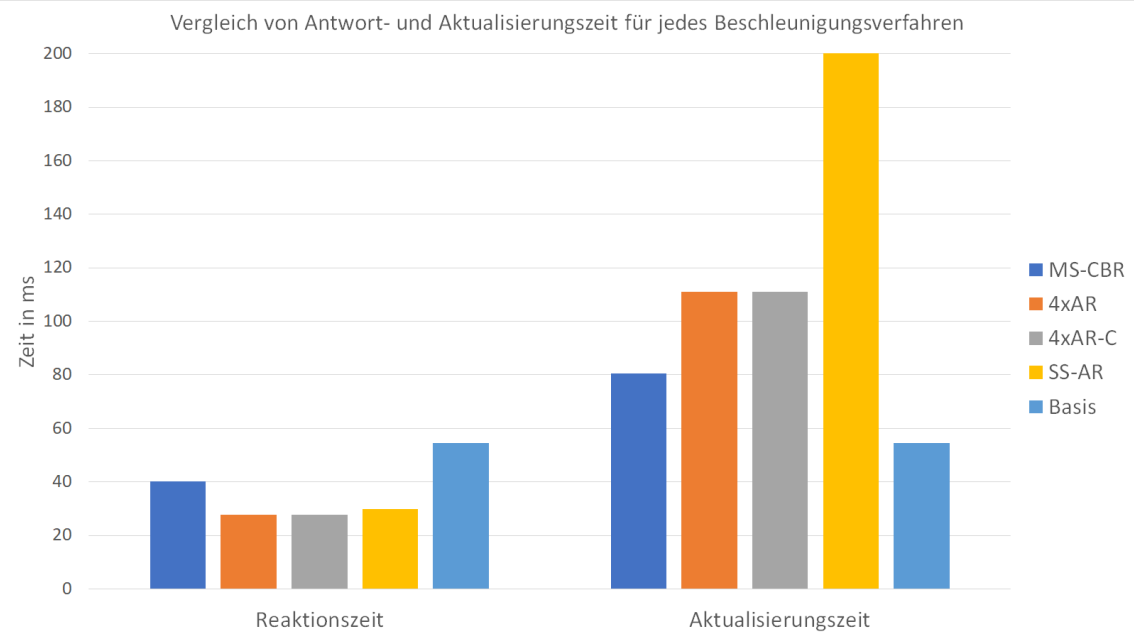


Abbildung 5.16: Visualisierung der durchschnittlichen Reaktionszeiten und Bildaktualisierungszeiten je Ansatz. Basis beschreibt die Dauer ohne Verwendung von Beschleunigungsansätzen. Die Aktualisierungsdauer für Ansatz SS-AR ist abgeschnitten und beträgt tatsächlich 4870ms.

5.10 Beschleunigung von Bresenham-Linien

Die vorgeschlagenen Beschleunigungsverfahren können auch auf Linien angewandt werden, die mit Bresenham's Algorithmus gezeichnet werden. Die dabei durchschnittlich erreichten Beschleunigungen werden für jeden Ansatz für die bisher beschriebenen Polygonlinien, sowie für die mit Bresenham's Algorithmus gezeichneten Linien in Abb. 5.17 gegenübergestellt. Für jeden Ansatz ist die erreichte Beschleunigung bei Bresenham-Linien etwas niedriger als es bei Polygonlinien der Fall ist. Bemerkenswert ist hierbei, dass der absolute Unterschied der erreichten Beschleunigung für jeden der Ansätze ca. 15% beträgt.
Da jedoch Zeichen von Linien mithilfe von Bresenham's Algorithmus sehr effizient geschieht, sind die benötigten Zeiten um ein Bild zu erstellen weiterhin schneller als es mit Polygonlinien der Fall ist. Eine Gegenüberstellung der Zeit um ein Frame zu erstellen ist in Abb. 5.18 abgebildet. Hier ist erkennbar, dass für jedes Beschleunigungsverfahren weiterhin bessere Zeiten erreicht werden können, wenn Linien mithilfe von Bresenham's Algorithmus gezeichnet werden. Zudem fällt auf, dass der relative Unterschied der benötigten Zeiten im Durchschnitt abnimmt. Ohne Nutzung von Beschleunigung beträgt der relative Unterschied zwischen den verglichenen Linienalgorithmen 53%, welcher dann mit Beschleunigung auf Werte zwischen 37% für 4xAR bis zu 24% für SS-AR sinkt. Dies ist durch die höhere Effektivität der Beschleunigungsverfahren für Polygonlinien zu erklären.
Letztendlich ist jedoch zu erkennen, dass die besten Antwortzeiten nur mithilfe von Bresenham's

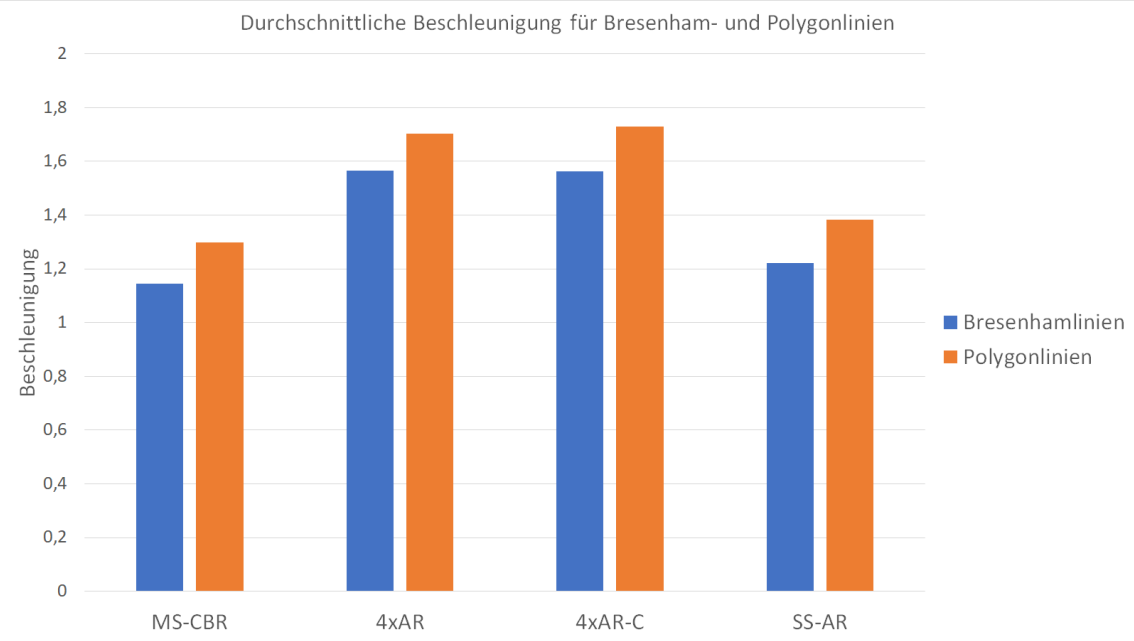


Abbildung 5.17: Gegenüberstellung von Beschleunigung von Bresenham-Linien und Polygonlinien in Form eines Balkendiagramms.

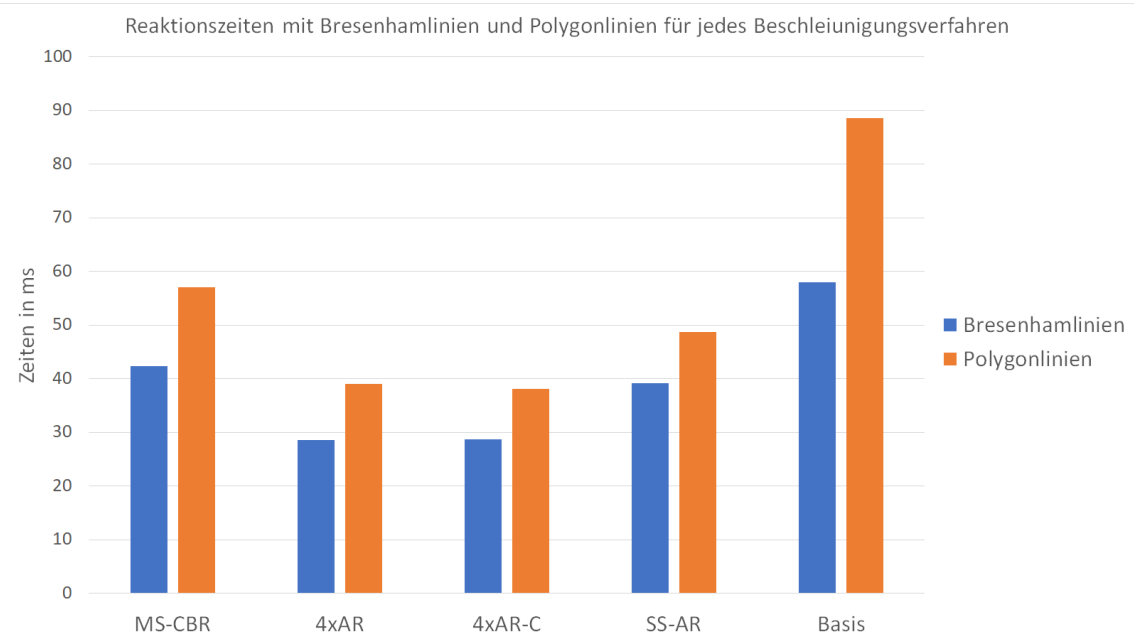


Abbildung 5.18: Benötigte Zeiten um ein Bild zu erstellen für Bresenham-Linien und Polygonlinien im Durchschnitt für jedes Beschleunigungsverfahren.

Linienalgorithmus erreicht werden können. Dies geschieht auf Kosten von Bildqualität wie in Kapitel 4.3 näher erläutert wurde und ist nur zu empfehlen, wenn der Detailverlust zugunsten von besseren Antwortzeiten in Kauf genommen werden kann.

5.11 Anwendungsfälle

Um nun den praktischen Nutzen der vorgeschlagenen Verfahren besser abschätzen zu können werden nun kleinere Mengen von Messdurchgängen beleuchtet. Hierbei werden die einzelnen Messungen als realistische Annäherung an praktische Anwendungen angesehen, da diese auf Daten vorgenommen wurden, welche man auch mit parallelen Koordinaten untersuchen wollen würde.

5.11.1 Tropfen-Datensatz, TitanXp, 1080p

Zunächst wird die Performanz der Visualisierung von parallelen Koordinaten für den Tropfen-Datensatz bei einer Auflösung von 1080p mit einer Nvidia TitanXP betrachtet. Die Mittelwerte der Reaktionszeit für die beiden getesteten Kamerapositionen wird in Abb. 5.19 abgebildet. Es ist erkennbar, dass Bresenham-Linien im Vergleich mit Polygonlinien schneller gezeichnet werden. Für beide Linientypen sind die Reaktionszeiten mit über 120ms jedoch lang und können bei der Interaktion mit der Visualisierung störend sein.
Bei dieser Visualisierung treten durchschnittliche Reaktionszeiten von 127ms im Fall von Bresenham-Linien und 167ms im Fall von Polygonlinien auf. Jeder der vorgeschlagenen Beschleunigungsverfahren erreicht hier eine Beschleunigung gegenüber der Basis, jedoch sind deutliche Unterschiede zwischen den Verfahren erkennbar. Das Verfahren MS-CBR kann die Reaktionszeit bei Nutzung von Polygonlinien lediglich auf 104.76ms senken, was zwar die Nutzungserfahrung verbessern kann, jedoch weiterhin eine hohe Reaktionszeit darstellt. Eine signifikante Verbesserung erzielen jedoch die Ansätze 4xAR, 4xAR-C und SS-AR, indem sie die Reaktionszeit für Polygonlinien auf ca 70ms senken. Hier ist eine deutliche Verbesserung der Nutungserfahrung zu erwarten. Eine weitere Verbesserung kann durch die Inkaufnahme von Detailverlust durch amoritisierte Bresenham-Linien erreicht werden. Bei Anwendung der Verfahren 4xAR, 4xAR-C oder SS-AR auf die Bresenham-Linien können Reaktionszeiten von etwa 48ms erreicht werden, was bereits einen relativ flüssigen Eindruck der Interaktion erzeugt.

5.11.2 Tropfen-Datensatz, Vega56, 720p

Im Folgenden wird die Visualisierung von parallelen Koordinaten für den Tropfen-Datensatz bei einer Auflösung von 720p mit einer AMD Vega 56 betrachtet. In Abb. 5.20 ist zunächst feststellbar, dass die Reaktionszeiten für Polygonlinien bereits deutlich länger sind, als es für Bresenham-Linien der Fall ist. Somit ist eine Echtzeitinteraktion mit den Daten in diesem Fall bei Wahl der Polygonlinien mit einer Reaktionszeit von ca. 165ms bereits schwierig, während Bresenham-Linien mit ca. 100ms Reaktionszeit bedeutend flüssigere Interaktion erlauben.
Der Beschleunigungsansatz MS-CBR kann hier im Fall von Polygonlinien keine Verbesserung gegenüber der Bresenham-Linien-Basis erzielen. Mit Reduktion der Reaktionszeit auf ca. 105ms für Polygonlinien und einer Reduktion auf ca. 77ms für Bresenham-Linien kann eine signifikante Reduktion der Reaktionszeit im Vergleich mit der Polygonlinienbasis erreicht werden. Eine Verbesserung

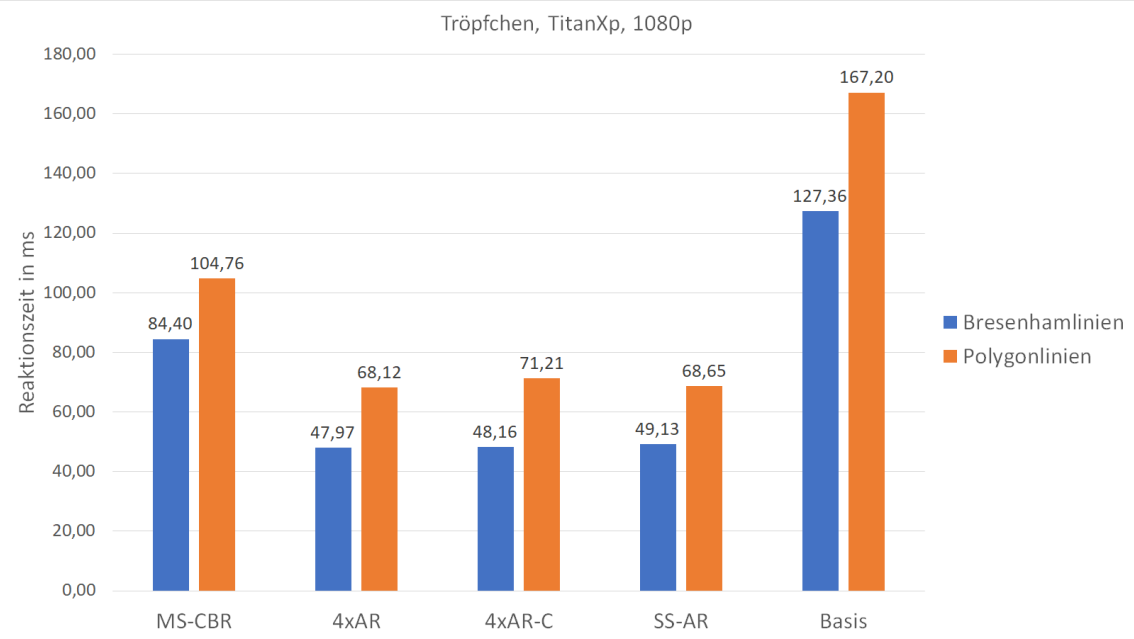


Abbildung 5.19: Durchschnittliche Reaktionszeiten der Visualisierung von parallelen Koordinaten für den Tropfen-Datensatz auf einer Nvidia TitanXp bei 1080p.

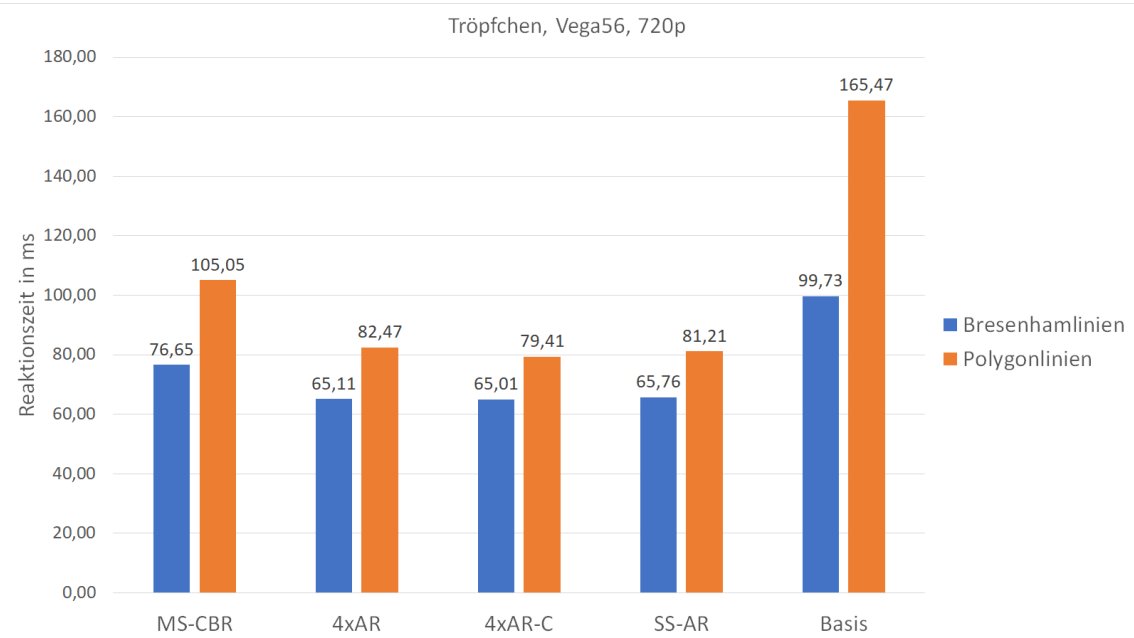


Abbildung 5.20: Durchschnittliche Reaktionszeiten der Visualisierung von parallelen Koordinaten für den Tropfen-Datensatz auf einer AMD Vega56 bei 720p.

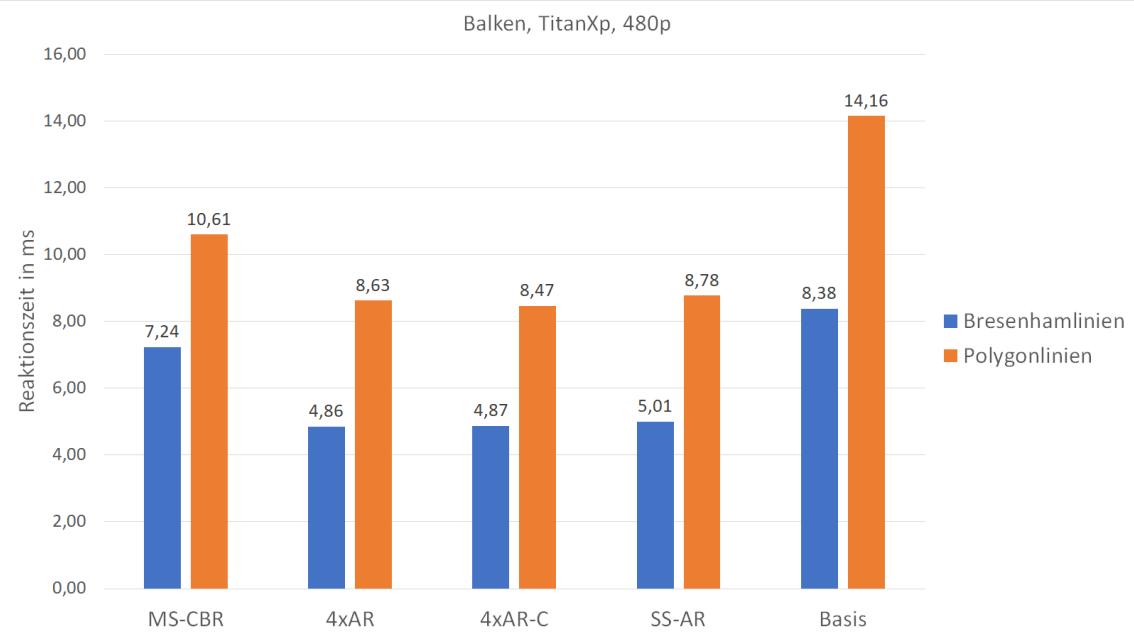


Abbildung 5.21: Durchschnittliche Reaktionszeiten bei der Visualisierung von parallelen Koordinaten für den Balken-Datensatz auf einer Nvidia TitanXp.

gegenüber der Bresenham-Linien-Basis ist also nur möglich, wenn hier die zugrundeliegenden Linien bei der Amortisierung auch durch Bresenham's Algorithmus gezeichnet werden. Dies führt jedoch zu einem Detailverlust und ist nicht zu empfehlen.
Auch hier liefern die Verfahren 4xAR, 4xAR-C und SS-AR ähnliche Ergebnisse. Für Polygonlinien erzielen diese Ansätze eine Reaktionszeit von ca. 81ms, während hier sogar Reaktionszeiten von ca. 65ms erreicht werden können, wenn Bresenham-Linien zugrunde liegen. Verglichen mit der Bresenham-Basis erzielen die Polygonlinien hier zwar nur relativ geringe absolute Beschleunigung, diese Verbesserung kann jedoch in Verbindung mit anderen, parallel laufenden Visualisierungen in MegaMol doch auch eine merkliche Verbesserung der Interaktion erzielen. Wie auch zuvor sind die optimalen Reaktionszeiten jedoch nur unter Inkaufnahme von Detailverlust, in Form von amortisierten Bresenham-Linien, erreichbar.

5.11.3 Balken-Datensatz, TitanXp, 480p

Der zuletzt betrachtete Fall bezieht sich auf die Visualisierung von parallelen Koordinaten für den Balken-Datensatz bei einer Auflösung von 480p mit einer Nvidia TitanXp. Die Reaktionszeiten die für diesen Fall im Durchschnitt für die beiden Kamerapositionen wurden für alle Beschleunigungsverfahren in Abb. 5.21 abgebildet. Wie auch in den anderen Fällen, lässt ohne Anwendung von Beschleunigungsverfahren die Wahl von Bresenham-Linien schnellere Reaktionszeiten zu. Für beide Linientypen erlaubt hier bereits die jeweilige Basis mit ca. 14ms für Poylgonlinien und ca. 8ms für Bresenham-Linien flüssige Interaktion mit der Visualisierung zu. Darüber hinaus ist erkennbar, dass unter Verwendung von Polygonlinien keines der vorgeschlagenen Verfahren eine Verbesserung der Reaktionszeit gegenüber der Bresenham-Basis erreichen kann. Zwar können mithilfe von Beschleunigungsverfahren auch unter Nutzung von Polygonlinien Reaktionszeiten gegenüber der

Polygonlinienbasis erreicht werden, jedoch stellt hier die Nutzung der Bresenham-Basis eine bessere Alternative dar. Die Bresenham-Basis erreicht eine äquivalente Bildqualität mit der der Polygon-Basis und sollte daher in diesem Fall jeder Art von Poylgonlinie, wenn möglich, vorgezogen werden.
Wenn weitere Beschleunigung in solch einem Fall gewünscht ist, ist dies nur noch mit Detailverlust möglich, indem man auch hier von amortisierten Bresenham-Linien gebrauch macht. In diesem Fall kann jedes der Beschleunigungsverfahren eine Verbesserung der Reaktionszeit erzielen, wobei die optimale Reaktionszeit von Verfahren 4xAR erreicht wird.
Dieser Fall zeigt, dass man dem Nutzer, wenn möglich, die Kontrolle über das verwendete Beschleunigungsverfahren gelassen werden sollte und zudem sogar, wenn möglich, eine Wahl zugestanden werden sollte, welcher Typ von Linien gezeichnet werden soll.

6 Schlussfolgerung

Ziel dieser Arbeit war die Beschleunigung des Renderns von Parallelen Koordinaten in MegaMol. Dieses Ziel wurde für eine große Anzahl von Anwendungsfällen durch Amortisierung des Renderns über mehrere Bilder mit anschließender temporaler Rekonstruktion erreicht. Große Datensätze und hohe Auflösungen, welche aufgrund der langen Renderzeiten unter hohen Reaktionszeiten leiden, können in manchem Fällen um einen Faktor von über 3.5 beschleunigt werden. Dies verbessert die Interaktion mit den Daten in Echtzeit und erlaubt Analyse von noch größeren Datensätzen. Hierbei wurde auch festgestellt, dass es Szenarien gibt, in denen die beschriebenen Beschleunigungsverfahren keine zufriedenstellenden Ergebnisse liefern können und sogar die Ausführung des Programms verlangsamen. Durch Verwenden von polygonbasierten Linien kann bei Amoritisierung Qualitätsverlust minimiert werden, jedoch sind optimale Reaktionszeiten des Programms nur unter inkaufnahme von Detailverlust durch Amortisierung von Bresenham-Linien möglich. Besonders die Ansätze 4xAR und 4xAR-C können für ausreichend große Datensätze häufig signifikante Beschleunigungen von Renderzeiten und somit auch von Reaktionszeiten erreichen. Hierbei ist jedoch zu bedenken, dass das Verfahren 4xAR-C das Ergebnisbild verschwommen erscheinen lässt, um hier mit von den anderen Verfahren abweichenden Abtastungspositionen Kantenglättung auf Kosten von Detailgrad einzuführen. Sollte keine Abweichung von dem nicht-amortisiertem Bild gewünscht sein, kann hier weiterhin mithilfe von MS-CBR und 4xAR eine Beschleunigung erreicht werden, wobei hier der ursprüngliche Bildeindruck erhalten bleibt. Mit dem Verfahren SS-AR kann die Bildqualität mit Kantenglättung durch Supersampling gegenüber dem nicht-amortisiertem Bild sogar verbessert werden. Dieser Ansatz erreicht jedoch leider oft keine oder nur geringe Beschleunigung für hohe Auflösungen, da zur Umsetzung von temporalem Supersampling hier hoher zusätzlicher Speicherbedarf erzeugt wird. Durch diese zusätzlichen Kosten, ist dieser Ansatz nur zu empfehlen, wenn entweder ausreichend Speicher und Speicherbandbreite zur Verfügung steht, oder die Bildqualität eine bedeutend größere Rolle spielt als Reaktionszeiten. Hier ist auch zu bedenken, dass alle Verfahren die Reaktionszeit zwar verringern, jedoch die Zeit um ein vollständig aktualisiertes Bild zu erhalten deutlich erhöht wird. Für Anwendungen, in denen auch die Bildaktualisierungszeit wichtig ist, kann MS-CBR, trotz verlgeichsweise schneller Aktualisierungszeit, oft eine Beschleunigung erreichen, während SS-AR mit Aktualisierungszeiten, die schnell im Bereich von mehreren Sekunden liegt, keine Anwendung findet.
Ghosting-Artefakte, die typischerweise bei amortisiertem Rendern auftreten, werden durch Reprojektion der alten Abtastungswerte vermieden. Trotz dieser Vermeidung entsteht bei Bewegung der Szene ein körniges Bild, welches jedoch eine gute Annäherung an ein fehlerfreies Bild ist und den Ghosting-Artefakten vorzuziehen ist.
Für alle Verfahren wurden die Effektivität mit Hinblick auf die Einflussfaktoren Auflösung, Datensatzgröße, Grafikkarte und Kameraposition untersucht. Dabei fällt auf, dass die beschriebenen Problemfälle, in Form von großen Datensätzen und hohen Auflösungen, im besonderen Maße von den Beschleunigungsverfahren profitieren.
Abschließend wird empfohlen dem Nutzer, wenn möglich, eine Wahl zu lassen, ob ein Beschleu-

nigungsverfahren zum Einsatz kommen soll und welches. Zudem ist auch eine Wahl über den verwendeten Algorithmus zur Zeichnung der Linien sinnvoll, besonders da die Amortisierung von Bresenham-Linien zu Detailverlust führen kann.

6.1 Ausblick

Diese Arbeit hat sich auf die Visualisierung von parallelen Koordinaten bezogen, jedoch sind die vorgeschlagenen Verfahren auch auf viele weitere Arten von Visualisierungen anwendbar. Einer Verallgemeinerung und Erweiterung für andere Fälle, besonders im zweidimensionalen Raum, steht hier wenig im Weg. Die durch die orthografische Kamera besonders einfache Reprojektion zum Ausgleich von Ghosting-Artefakten kann leicht für weitere Fälle verallgemeinert werden. Hier kann zudem untersucht werden, welche Effekte Amortisierung auf andere Arten von Visualisierung hat und ob auch in anderen Szenarien Effekte wie Qualitätsverlust bei Amortisierung von Bresenhamlinien auftreten.
Da selbst nach Reprojektion der Abtastungen in dieser Arbeit noch Artefakte zu erkennen waren, kann die Bildqualität bei Bewegung durch Verbesserung dieses Schrittes weiter verbessert werden. Für den dreidimensionalen Raum wurden bereits komplexere Methoden vorgeschlagen, solche Artefakte zu eliminieren und könnten zu diesem Zweck adaptiert werden.
Bei der Untersuchung der Effektivität wurden Unterschiede zwischen den verwendeten GPUs festgestellt. Weitere Untersuchungen können hier ein besseres Bild von Stärken und Schwächen der Geräte zeichnen und zur hardwarespezifischen Optimierung von Visualisierungen verwendet werden.
Dieser Arbeit ist es leider nur unter hohem Speicheraufwand gelungen zufriedenstellende Kantenglättung zu erreichen. Dies kann durch bessere Wahl der Abtastungspositionen und angemessener Gewichtung von vorangegangenen Werten optimiert werden und sowohl die erreichte Kantenglättung verbessern als auch die Speicherkosten senken. Insbesondere besteht die Möglichkeit aufeinander folgende Abtastungswerte direkt zu einem einzigen Wert zu aggregieren, was jedoch eine komplexe Gewichtungsfunktion erfordert, um keine weiteren Ghosting-Artefakte zu erzeugen.

Literaturverzeichnis

[Bre65] J. E. Bresenham. „Algorithm for computer control of a digital Plotter“. In: *IBM Systems Journal* 4.1 (1965), S. 25–30 (zitiert auf S. 17).

[CCC87] R. L. Cook, L. Carpenter, E. Catmull. „The Reyes image rendering architecture“. In: *ACM SIGGRAPH Computer Graphics* 21.4 (Aug. 1987), S. 95–102. DOI: 10.1145/37402.37414. URL: https://doi.org/10.1145/37402.37414 (zitiert auf S. 24).

[EBRI09] M. Everts, H. Bekker, J. Roerdink, T. Isenberg. „Depth-Dependent Halos: Illustrative Rendering of Dense Line Data“. In: *IEEE Transactions on Visualization and Computer Graphics* 15.6 (Nov. 2009), S. 1299–1306. DOI: 10.1109/tvcg.2009.138. URL: https://doi.org/10.1109/tvcg.2009.138 (zitiert auf S. 24).

[Ert18] P. D. T. Ertl. *Vorlesung Computergrafik*. Vorlesungsfolien. 2018 (zitiert auf S. 17).

[MKG+97] L. Markosian, M. A. Kowalski, D. Goldstein, S. J. Trychin, J. F. Hughes, L. D. Bourdev. „Real-time nonphotorealistic rendering“. In: *Proceedings of the 24th annual conference on Computer graphics and interactive techniques - SIGGRAPH '97*. ACM Press, 1997. DOI: 10.1145/258734.258894. URL: https://doi.org/10.1145/258734.258894 (zitiert auf S. 19, 23).

[ML18] T. Mcfarron, A. Lake. *Checkerboard Rendering for RealTime Upscaling on Intel Integrated Graphics*. https://software.intel.com/content/www/us/en/develop/articles/checkerboard-rendering-for-real-time-upscaling-on-intel-integrated-graphics.html. Sep. 2018 (zitiert auf S. 24, 26).

[Oca85] M. d'Ocagne. *Coordonnées Parallèles et Axiales: Méthode de transformation géométrique et procédé nouveau de calcul graphique déduits de la considération des coordonnées parallèlles*. Paris: Gauthier-Villars, 1885 (zitiert auf S. 15, 23).

[Reg72] G. B. Reggiori. „Digital computer transformations for irregular line drawings“. Diss. New York: New York University, Apr. 1972 (zitiert auf S. 23).

[RK00] C. Rossl, L. Kobbelt. „Line-art rendering of 3D-models“. In: *Proceedings the Eighth Pacific Conference on Computer Graphics and Applications*. IEEE Comput. Soc, 2000. DOI: 10.1109/pccga.2000.883890. URL: https://doi.org/10.1109/pccga.2000.883890 (zitiert auf S. 24).

[RLC+11] J. Ragan-Kelley, J. Lehtinen, J. Chen, M. Doggett, F. Durand. „Decoupled sampling for graphics pipelines“. In: *ACM Transactions on Graphics* 30.3 (Mai 2011), S. 1–17. DOI: 10.1145/1966394.1966396. URL: https://doi.org/10.1145/1966394.1966396 (zitiert auf S. 24).

[S G81] R. F. S. S. Gupta. „Filtering Edges for Gray-Scale Displays“. In: *Computer Graphics* 15.3 (1981), S. 1–5 (zitiert auf S. 23).

[SA] M. Segal, K. Akeley. *The OpenGL Graphics System: A Specification*. The Khronos Group Inc. Beaverton, OR, S. 471–477 (zitiert auf S. 25).

[Sig] Sigbert. *ParallelCoordinatePattern*. https://upload.wikimedia.org/wikipedia/commons/thumb/0/07/ParallelCoordinatePattern.svg/800px-ParallelCoordinatePattern.svg.png. zuletzt zugegriffen am 19.11.20, Lizenzvereinbarung: https://creativecommons.org/licenses/by-sa/3.0/legalcode (zitiert auf S. 15).

[Wu91] X. Wu. *An Efficient Antialiasing Technique*. 1991 (zitiert auf S. 18, 23).

[XLV18] K. Xiao, G. Liktor, K. Vaidyanathan. „Coarse pixel shading with temporal supersampling“. In: *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. ACM, Mai 2018. DOI: 10.1145/3190834.3190850. URL: https://doi.org/10.1145/3190834.3190850 (zitiert auf S. 24).

[YLS20] L. Yang, S. Liu, M. Salvi. „A Survey of Temporal Antialiasing Techniques“. In: *Computer Graphics Forum* 39.2 (Mai 2020), S. 607–621. DOI: 10.1111/cgf.14018. URL: https://doi.org/10.1111/cgf.14018 (zitiert auf S. 22).

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift